Habilitation à diriger des recherches en informatique

présentée par

Fabian M. Suchanek

à l'Université Pierre et Marie Curie, Paris

Contributions à l'avancement des grandes bases de connaissances

Abstract
Une ontologie (ou *base de connaissances*) est une collection de connaissances qui peut être traitée par un ordinateur. Ma thèse de doctorat a décrit comment on peut construire de telles ontologies de manière automatisée. La présente thèse d'habilitation éclaire différents aspects de la gestion des ontologies. Un de ces aspects est la découverte de règles implicites entre entités, comme $conjoint(x, y) \wedge domicile(x, z) \Rightarrow domicile(y, z)$. Un autre aspect est l'exhaustivité des ontologies. Nous proposons d'intégrer les données de services Web dans l'ontologie pour l'étendre. Deux ontologies peuvent parler des mêmes concepts sous deux noms différents. Pour cela, nous traitons l'alignement des ontologies – au niveau des instances et au niveau du schéma. Nous expliquons aussi comment une ontologie peut être étendue par des annotations textuelles. Finalement, nous montrons comment une ontologie peut être protégée par le tatouage.

Remerciements
Je remercie les auteurs principaux des publications réunies dans ce manuscrit pour leur aimable autorisation.

Abstract
An ontology is a computer-processable collection of knowledge about the world. My PhD thesis described how an ontology can be constructed by automated means. This habilitation thesis elaborates on different aspects of the management of ontologies. One of these aspects is rule mining, i.e., the identification of patterns such as $spouse(x, y) \wedge livesIn(x, z) \Rightarrow livesIn(y, z)$. Another aspect is the incompleteness of the ontologies. We propose to integrate data from Web Services into an ontology to address this incompleteness. Sometimes existing ontologies overlap. This prompts us to look into the alignment of ontologies, both in terms of instances and in terms of the schema. We also discuss how an ontology can be extended by textual patterns. Finally, we show how ontologies can be protected by watermarking.

Acknowledgements
I would like to thank the main authors of the publications that are summarized here for their kind agreement to let these works contribute to this thesis.

Contents

Chapter 1

Introduction (en français)

1.1 Contexte

Ontologies. Une ontologie (ou *base de connaissances*) est une collection formelle de connaissances. Une ontologie peut savoir, par exemple, que Elvis Presley est un chanteur, qu'il joue à la guitarre, et que tous les chanteurs sont des personnes.[1] L'ontologie contient les connaissances sous une forme qui permet à l'ordinateur de répondre à des requêtes sur ces connaissances, de vérifier sa cohérence logique, et d'effectuer du raisonnement. Par conséquent, les ontologie sont utilisées dans de nombreux domaines, comme par exemple la traduction automatique des textes, la disambiguation, la classification de documents, la query expansion, et l'intégration d'information. Peut-être l'application la plus pertinante est que les ontologies peuvent aider l'ordinateur à répondre à des requêtes qui nécessitent beaucoup de connaissances – parfois même en langage naturel. Le système Watson d'IBM est un exemple. Ce système a affronté les champions humains dans le quiz télévisé *Jeopardy!*, et a gagné. Nous possédons maintenant de outils comme Siri de Apple, le knowledge graph de Google, ou Google Glasses, qui semblaient faire partie de la science fiction il y a quelques années. Ils utilisent, entre outre, des représentations ontologiques du monde réel. Leur succès montre les applications et l'impact des données ontologiques.

YAGO. Vu l'utilité des ontologies, il se pose la question comment on peut collecter les connaissances pour construire une ontologie. Ma thèse de doctorat [16] a montré comment une ontologie peut être construite de manière automatique à partir du Web. L'approche développée extrait des connaissances de Wikipédia[2], une encyclopédie accessible en ligne, et les combine avec WordNet [4], le dictionnaire de la langue anglaise. Le résultat est YAGO [20, 21], une ontologie qui rassemble des millions d'entités (des personnes, des endroits, des organisations, des pièces de musique, etc.), et des millions de faits sur les entités (comme par exemple, qui est né où, quelle ville se trouve dans quel pays, et quel chanteur a chanté quelle chanson). Une évaluation manuelle par échantillon a montré une

[1] Dans cette thèse, le terme *ontologie* est utilisé dans un sens très général, qui inclut les faits sur les instances.

[2] http://wikipedia.org

précision des données de 95%. Nous avons aussi montré comment l'ontologie peut être étendue par l'extraction de connaissances à partir d'autres sources du Web. Notre approche, SOFIE [22], utilise le raisonnement logique pour déduire le sens le plus plausible des phrases en langage naturel.

Depuis, YAGO a crû sans arrêt. Avec YAGO2 [8, 7], nous avons intégré plus de sources (Geonames[3] et Universal WordNet [9]). Nous avons aussi donné une dimension temporelle et spatiale à l'ontologie. Ainsi, beaucoup d'entités et faits sont maintenant ancrés dans le temps et dans l'espace. Avec YAGO2s [3], nous avons parallelisé et modularisé l'architecture de sorte que plusieurs personnes peuvent maintenant travailler sur le projet. En ce moment, YAGO contient pus de 10 millions d'entités et 120 millions de faits. L'ontologie est utilisée dans plusieurs projets. Un d'entre eux est le projet Watson mentionné ci-dessus.

Ontologies existantes. Grâce en partie au travail sur YAGO, mais grâce aussi à beaucoup d'autres projets [1, 10, 2], la construction automatique des ontologies est maintenant bien étudiée. Par conséquent, il existe plusieurs ontologies de grande taille. DBpedia [1], par exemple, est un projet qui a le même but de YAGO de construire une grande ontologie à partir de Wikipédia. Le projet résulte d'un effort collaboratif, et se focalise sur l'intégration des données avec la Linked Open Data cloud. NELL [10] ("never-ending language learner") est un système qui fonctionne perpétuellement pour extraire toujours plus d'information du Web. L'approche combine l'extraction de sources structurées et de sources non-structurées, et peut même interagir avec les utilisateurs pour améliorer la qualité d'extraction. TextRunner [2] est un système qui construit une ontologie en collectant un grand nombre de triplets de la forme sujet-verbe-objet du Web. Freebase[4] est une approche collaborative, qui se base sur Wikipédia et sur les contributions de communautés. On constate donc que le domaine de la construction des ontologies a avancé beaucoup ces dernières années.

1.2 Défis

Défis. Les avancements dans le domaine de l'extraction d'information ont amené à la construction de grandes ontologies. Ces ontologies posent des défis aussi après leur construction. Par exemple, il y a des ontologies qui contiennent la même entité, mais qui utilisent des identifieurs différents (par exemple *Elvis* et *ElvisPresley*). Cela implique que les données de ces ontologies ne peuvent pas être combinées facilement. Un autre défi est l'exhaustivité des ontologies. Les ontologies d'aujourd'hui contiennent les entités populaires, mais pas forcément les entités moins connues. Les ontologies peuvent aussi être réutilisées dans d'autres ontologies du Web – potentiellement sans l'accord de leurs créateurs. Dans ces cas, on doit prouver la provenance des données ontologiques. Finalement, les ontologies contiennent des informations implicites qui peuvent être de grande valeur, comme par exemple le fait qu'une personne vit d'habitude dans la même ville que son conjoint. La fouille de ces règles implicites est un défi en soi.

[3] http://geonames.org
[4] http://freebase.com

Ceux-ci sont seulement quelques unes des questions intéressantes qui se posent après la construction d'une ontologie. Après ma thèse de doctorat, j'ai travaillé sur ces aspects, et cette thèse d'habilitation est un résumé des résultats que j'ai obtenus avec mes collègues. Mon manuscrit de thèse commence par une introduction en anglais (Chapitre 2) qui reprend la présente introduction française. La suite du manuscrit est structurée en chapitres, dont chacun traite d'un défis particulier et d'une solution. Chaque chapitre se base sur une publication.[5]

- **Chapitre 3 : Fouille de règles avec AMIE**
  La fouille de règles est la tâche de trouver des correlations sémantiques dans les données. Par exemple, nous pouvons trouver qu'une personne vit d'habitude dans le même endroit que son conjoint : $conjoint(x, y) \wedge domicile(x, z) \Rightarrow domicile(y, z)$. Trouver ces règles implicites, en particulier sur des millions de faits, n'est pas facile. Avec AMIE [6, 5], nous avons développé une approche qui utilise l'élagage et une nouvelle mesure de qualité pour trouver des règles de Horn dans des grandes ontologies. Ce chapitre se base sur la publication suivante :

  > Luis Galárraga, Christina Teflioudi, Katja Hose, Fabian Suchanek
  > "AMIE: Association Rule Mining under Incomplete Evidence
  > in Ontological Knowledge Bases"
  > (International World Wide Web Conference (WWW) 2013)

  Ce papier a obtenu le best student paper award de la conférence.

- **Chapitre 4 : L'intégration des services Web avec ANGIE**
  Une ontologie ne peut jamais être complète. Pour cela, nous avons travaillé sur l'intégration dynamique des connaissances des services Web dans l'ontologie. Notre idée est de lancer des requêtes aux services Web en fonction de la requête de l'utilisateur, et d'étendre l'ontologie avec les informations reçues. Nous avons appelé cette approche ANGIE [13, 14], et ce travail a été fait dans :

  > Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek,
  > Thomas Neumann, Wenjun Yuan, Gerhard Weikum
  > "Active Knowledge: Dynamically Enriching RDF Knowledge Bases
  > by Web Services (ANGIE)"
  > (International Conference on Management of Data (SIGMOD) 2010)

- **Chapter 5 : L'utilisation des services Web avec SUSIE**
  Les services Web sont une ressource utile pour étendre les ontologies. Pour lancer une requête à un service Web, nous devons fournir des valeurs d'entrée. Par exemple, si nous appelons un service Web avec le nom d'un chanteur, le service peut nous donner les titres de ses chansons. Le problème est qu'il est impossible d'utiliser cette fonction pour obtenir le chanteur si on a la chanson. Même si le service Web contient ces information, il n'est pas possible de formuler une requête pour les obtenir. Nous avons développé une approche qui s'appelle SUSIE [15], et qui utilise le Web comme oracle. Le travail a été publié dans :

[5] Le premier auteur est toujours l'auteur principal. Pour AMIE [6], je suis l'encadrant du premier auteur.

Nicoleta Preda, Fabian M. Suchanek, Wenjun Yuan, Gerhard Weikum
"SUSIE: Search Using Services and Information Extraction"
(International Conference on Data Engineering (ICDE) 2013)

- **Chapitre 6 : L'alignement des ontologies avec PARIS**
  Le Web Sémantique est constitué de centaines d'ontologies. Cette richesse ne peut être utilisée que si les ontologies ont été alignées. Avec PARIS [17], nous avons développé une approche qui identifie des instances, des relations, et des classes équivalentes à travers deux ontologies. Ce chapitre se base sur la publication suivante :

  Fabian M. Suchanek, Serge Abiteboul, Pierre Senellart
  "PARIS: Probabilistic Alignment of Relations, Instances, and Schema"
  (International Conference on Very Large Databases (VLDB) 2012)

- **Chapter 7 : La fouille des textes avec PATTY**
  D'habitude, les ontologies ne contiennent qu'un seul nom pour chaque relation. Par exemple, YAGO connaît la relation *isMarriedTo*, qui tient entre une personne et son conjoint. Par contre, cette relation peut être exprimée sous différentes formes dans un texte en langage naturel. Par exemple, on peut dire "X is married to Y", "X's husband Y", ou bien "X and his wife Y". Avec PATTY [12, 11], nous avons développé une approche qui collecte ces motifs de manière systématique dans le Web. Ce travail a été publié dans :

  Ndapandula Nakashole, Gerhard Weikum, Fabian M. Suchanek
  "PATTY: A Taxonomy of Relational Patterns with Semantic Types"
  (International Conference on Empirical Methods
  in Natural Language Processing (EMNLP) 2012)

  Ce papier a été le second gagnant du best paper award de la conférence.

- **Chapitre 8: Le tatouage des ontologies**
  Beaucoup d'ontologies sont disponibles gratuitement sur le Web. La majorité d'entre elles sont disponibles sous une licence qui demande d'attribuer le mérite au créateur de l'ontologie, si les données sont utilisées dans d'autres bases de connaissances. Par contre, il n'est pas facile de prouver qu'une ontologie contient des données plagiées d'une autre ontologie, car les bases de connaissances contiennent des informations publiques que tout le monde peut collecter. Nous avons développé des approches de tatouage [19, 18] pour protéger les sources ontologiques. Ce chapitre se base sur la publication suivante :

  Fabian M. Suchanek, David Gross-Amblard, Serge Abiteboul
  "Watermarking for Ontologies"
  (International Semantic Web Conference (ISWC) 2011)

La thèse finit avec une conclusion dans le Chapitre 9.

References

[1] Sören Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *ISWC*. 2007.

[2] Michele Banko et al. "Open Information Extraction from the Web". In: *IJCAI*. 2007.

[3] Joanna Biega, Erdal Kuzey, and Fabian M. Suchanek. "Inside YAGO2s: a transparent information extraction architecture". In: *World Wide Web Conference (WWW, demo)*. 2013.

[4] C. Fellbaum, ed. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] Luis Antonio Galárraga, Nicoleta Preda, and Fabian M. Suchanek. "Mining Rules to Align Knowledge Bases". In: *Workshop on Automated Knowledge Base Construction (AKBC) at CIKM*. 2013.

[6] Luis Antonio Galárraga et al. "AMIE: association rule mining under incomplete evidence in ontological knowledge bases". In: *World Wide Web Conference (WWW)*. 2013.

[7] Johannes Hoffart et al. "YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia". In: *World Wide Web Conference (WWW, demo)*. 2010.

[8] Johannes Hoffart et al. "YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia". In: *Artificial Intelligence Journal, Special Issue on Artificial Intelligence, Wikipedia and Semi-Structured Resources* (2012).

[9] Gerard de Melo and Gerhard Weikum. "Towards a Universal Wordnet by Learning from Combined Evidence". In: *Conference on Information and Knowledge Management (CIKM)*. 2009.

[10] Tom M. Mitchell et al. "Populating the Semantic Web by Macro-Reading Internet Text". In: *International Semantic Web Conference*. 2009.

[11] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. "Discovering and Exploring Relations on the Web". In: *Proceedings of the Conference on Very Large Databases (VLDB, demo)* 5.12 (2012).

[12] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. "PATTY: A Taxonomy of Relational Patterns with Semantic Types". In: *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2012.

[13] Nicoleta Preda et al. "Active Knowledge: Dynamically Enriching RDF Knowledge Bases by Web Services (ANGIE)". In: *Conference on Management of Data (SIGMOD) 2010*. 2010.

[14] Nicoleta Preda et al. "ANGIE: Active Knowledge for Interactive Exploration". In: *Conference on Very Large Databases (VLDB, demo) 2009*. 2009.

[15] Nicoleta Preda et al. "SUSIE: Search using services and information extraction". In: *International Conference on Data Engineering*. 2013.

[16] Fabian M. Suchanek. “Automated Construction and Growth of a Large Ontology”. PhD thesis. Saarland University, 2009.

[17] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. “PARIS: Probabilistic Alignment of Relations, Instances, and Schema”. In: *Proceedings of the Conference on Very Large Databases (VLDB)* 5.3 (2011).

[18] Fabian M. Suchanek and David Gross-Amblard. “Adding fake facts to ontologies”. In: *World Wide Web Conference (WWW)*. 2012.

[19] Fabian M. Suchanek, David Gross-Amblard, and Serge Abiteboul. “Watermarking for Ontologies”. In: *International Semantic Web Conference*. 2011.

[20] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. “Yago: A Core of Semantic Knowledge”. In: *World Wide Web conference (WWW)*. ACM Press, 2007.

[21] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. “YAGO: A Large Ontology from Wikipedia and WordNet”. In: *Elsevier Journal of Web Semantics* (2008).

[22] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. “SOFIE: A Self-Organizing Framework for Information Extraction”. In: *World Wide Web conference (WWW)*. ACM Press, 2009.

Chapter 2

Introduction (in English)

2.1 Background

Ontologies. An ontology (or *knowledge base*) is a formal collection of world knowledge. An ontology can, e.g., know that Elvis Presley is a singer, that he played guitar, and that all singers are people[1]. An ontology contains knowledge in such a form that a computer can answer queries on it, check it for consistency, and reason on it. Consequently, ontologies find applications in numerous domains, such as machine translation, word sense disambiguation, document classification, query expansion, and information integration. Maybe most prominently, ontologies can help computers answer knowledge-intensive questions, sometimes even in natural language. IBM's Watson system is an example. It competed in the American TV show *Jeopardy!* against human champions and won. Other gadgets such as intelligent maps, Apple's Siri, the Google Glasses, or Google's knowledge graph, which sounded like fiction a few years ago, are establishing themselves – and are using, inter alia, ontological representations of the real world. These successes show the use and impact of ontological data.

YAGO. Given the use of ontologies, the question arises how we can gather enough knowledge to construct such an ontology. During my PhD thesis [16], we have shown how to construct an ontology automatically from Web sources. Our approach extracts knowledge from Wikipedia[2], the grand online encyclopedia, and combines it with WordNet [4], the lexicon of the English language. The result is YAGO [20, 21], a large ontology, which contains millions of entities (such as people, locations, organizations, and pieces of music), and millions of facts about them (such as who was born where, which city is located in which country, and which singer sang which song). A manual evaluation on a sample has shown an accuracy of 95%. We have also shown how to enrich the ontology by extracting knowledge from other Web sources. Our approach, SOFIE [22], uses logical reasoning to deduce the most plausible meaning of natural language

[1] In this thesis, we use the term *ontology* in a very general sense to include also facts about instances.

[2] http://wikipedia.org

input sentences.

Since then, YAGO has grown relentlessly. With YAGO2 [8, 7], more sources have been integrated (Geonames[3] and the Universal WordNet [9]). The ontology also received a temporal and geographical dimension, so that many entities and facts are now anchored in time and space. With YAGO2s [3], the architecture was parallelized and modularized, so that several people can now work on the project. At the time of this writing, YAGO contains 10 million entities and 120 million facts about them. It has found numerous applications, and one of them is the Watson project mentioned above.

Existing Ontologies. Thanks in part to the work on YAGO, but thanks also to many other projects (e.g., [1, 10, 2]), the construction of ontologies from unstructured and semi-structured sources is now reasonably well understood. As a consequence, there are several other large-scale ontologies. DBpedia [1], e.g., is a project that shares YAGO's goal of building a large-scale ontology from Wikipedia. The project is a collaborative effort by a community, and particular focus is put on the integration with the Linked Open Data cloud. NELL [10], the "never-ending language learner", is a system that runs perpetually to extract ever more and more information from Web sources. It combines extraction from structured and unstructured sources, and can even interact with users to improve the extraction quality. TextRunner [2] is a system that builds a knowledge base by harvesting a large number of subject-verb-object triples from Web crawls. Freebase[4] is a collaborative approach that builds on Wikipedia and contributions by a large user community. The entire area of research has made huge progress in these last years.

2.2 Challenges

Challenges. Recent advances in the area of information extraction have led to the creation of several large scale ontologies. These ontologies, however, come with their own challenges. For example, some ontologies talk about the same entities, but use different identifiers for them (e.g., *Elvis* and *ElvisPresley*). This means that information from these ontologies cannot be combined. Another challenge is the incompleteness of the ontologies. Today's large-scale ontologies contain mainly the popular information, but not necessarily the information on the long tail. Ontologies can also be re-used in other ontologies across the Web – possibly without consent of their creator. Then the issue of proof of provenance arises. Finally, ontologies may contain valuable implicit information, such as the fact that people usually live in the same place as their spouses. Mining such correlations is a challenge in itself.

These are just some of the interesting questions that arise after we have built an ontology. After my PhD thesis, I have worked on these dimensions, and the present habilitation thesis is a summary of the results that my colleagues and I have achieved. After the introduction in French and English, the thesis consists of the following chapters, which each illuminates one particular task

[3] http://geonames.org
[4] http://freebase.com

and a solution. Each chapter is based on a published paper[5].

- **Chapter 3: Rule Mining with AMIE**
  Rule mining is the task of finding semantic correlations in the data. For example, we can find that a person usually lives in the place where their spouse lives: $spouse(x, y) \wedge livesIn(x, z) \Rightarrow livesIn(y, z)$. Mining such rules, in particular on millions of facts, is no easy task. With AMIE [6, 5], we have developed an approach that uses clever pruning and a new quality metric to mine Horn rules on large ontologies. This chapter is based on

  > Luis Galárraga, Christina Teflioudi, Katja Hose, Fabian Suchanek
  > "AMIE: Association Rule Mining under Incomplete Evidence
  > in Ontological Knowledge Bases"
  > (International World Wide Web Conference (WWW) 2013)

  This paper has won the best student paper award at the WWW 2013 conference.

- **Chapter 4: Web Service Integration with ANGIE**
  No knowledge base can ever be complete. Therefore, we have worked on integrating knowledge from Web services dynamically into the ontology. The idea is to query the Web service in response to user queries, and extend the ontology on the fly with the necessary information. We have coined this approach ANGIE [13, 14], and the publication is

  > Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek,
  > Thomas Neumann, Wenjun Yuan, Gerhard Weikum
  > "Active Knowledge: Dynamically Enriching RDF Knowledge Bases
  > by Web Services (ANGIE)"
  > (International Conference on Management of Data (SIGMOD) 2010)

- **Chapter 5: Web Service Querying with SUSIE**
  Web services are a useful resource for extending the ontology. In order to query a Web service, we have to provide input values to obtain the output values. For example, if we call a Web service with the name of a singer, it can give us the titles of all his songs. The problem arises if we have the song and wish to know the singer. Although the Web service has this data, we cannot query it in this way. We have developed a solution to this problem, coined SUSIE [15], which uses the Web as an oracle. The publication is

  > Nicoleta Preda, Fabian M. Suchanek, Wenjun Yuan, Gerhard Weikum
  > "SUSIE: Search Using Services and Information Extraction"
  > (International Conference on Data Engineering (ICDE) 2013)

- **Chapter 6: Ontology Alignment with PARIS**
  There are literally hundreds on ontologies on the Semantic Web. Many of them contain overlapping and complementary information. This wealth

[5]The first author is always the main author of the publication. For AMIE [6], I am the PhD advisor of the first author.

of data can only be used if the terminologies of the ontologies have been aligned. With PARIS [17], we have developed an approach that links equivalent instances, relations, and classes across two ontologies. This chapter is based on

> Fabian M. Suchanek, Serge Abiteboul, Pierre Senellart
> "PARIS: Probabilistic Alignment of Relations, Instances, and Schema"
> (International Conference on Very Large Databases (VLDB) 2012)

- **Chapter 7: Mining Textual Patterns with PATTY**
  Ontologies usually contain only one single name for a relationship. For example, YAGO contains the relationship *isMarriedTo*, which holds between a person and their spouse. However, there are many different ways in which such a relationship can be expressed in natural language text. Examples include "X is married to Y", "X's husband Y", or "X and his wife Y". With PATTY [12, 11], we have developed an approach that harvests such patterns systematically from Web corpora. This led to the publication

  > Ndapandula Nakashole, Gerhard Weikum, Fabian M. Suchanek
  > "PATTY: A Taxonomy of Relational Patterns with Semantic Types"
  > (International Conference on Empirical Methods
  > in Natural Language Processing (EMNLP) 2012)

  This paper was a runner-up to the best paper award of the EMNLP 2012.

- **Chapter 8: Watermarking Ontologies**
  Many ontologies are freely available on the Internet. The majority of them comes with a license that requires giving credit to the creators of the ontology, if the data is used in other knowledge bases. However, it is very hard to prove that one source copied data from another source, because ontologies contain real world knowledge, which anyone can collect. We have developed watermarking techniques [19, 18] to protect ontological sources. The publication for this chapter is

  > Fabian M. Suchanek, David Gross-Amblard, Serge Abiteboul
  > "Watermarking for Ontologies"
  > (International Semantic Web Conference (ISWC) 2011)

This thesis finishes with a conclusion in Chapter 9.

References

[1] Sören Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *ISWC*. 2007.

[2] Michele Banko et al. "Open Information Extraction from the Web". In: *IJCAI*. 2007.

[3] Joanna Biega, Erdal Kuzey, and Fabian M. Suchanek. "Inside YAGO2s: a transparent information extraction architecture". In: *World Wide Web Conference (WWW, demo)*. 2013.

[4] C. Fellbaum, ed. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] Luis Antonio Galárraga, Nicoleta Preda, and Fabian M. Suchanek. "Mining Rules to Align Knowledge Bases". In: *Workshop on Automated Knowledge Base Construction (AKBC) at CIKM*. 2013.

[6] Luis Antonio Galárraga et al. "AMIE: association rule mining under incomplete evidence in ontological knowledge bases". In: *World Wide Web Conference (WWW)*. 2013.

[7] Johannes Hoffart et al. "YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia". In: *World Wide Web Conference (WWW, demo)*. 2010.

[8] Johannes Hoffart et al. "YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia". In: *Artificial Intelligence Journal, Special Issue on Artificial Intelligence, Wikipedia and Semi-Structured Resources* (2012).

[9] Gerard de Melo and Gerhard Weikum. "Towards a Universal Wordnet by Learning from Combined Evidence". In: *Conference on Information and Knowledge Management (CIKM)*. 2009.

[10] Tom M. Mitchell et al. "Populating the Semantic Web by Macro-Reading Internet Text". In: *International Semantic Web Conference*. 2009.

[11] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. "Discovering and Exploring Relations on the Web". In: *Proceedings of the Conference on Very Large Databases (VLDB, demo)* 5.12 (2012).

[12] Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. "PATTY: A Taxonomy of Relational Patterns with Semantic Types". In: *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2012.

[13] Nicoleta Preda et al. "Active Knowledge: Dynamically Enriching RDF Knowledge Bases by Web Services (ANGIE)". In: *Conference on Management of Data (SIGMOD) 2010*. 2010.

[14] Nicoleta Preda et al. "ANGIE: Active Knowledge for Interactive Exploration". In: *Conference on Very Large Databases (VLDB, demo) 2009*. 2009.

[15] Nicoleta Preda et al. "SUSIE: Search using services and information extraction". In: *International Conference on Data Engineering*. 2013.

[16] Fabian M. Suchanek. "Automated Construction and Growth of a Large Ontology". PhD thesis. Saarland University, 2009.

[17] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. "PARIS: Probabilistic Alignment of Relations, Instances, and Schema". In: *Proceedings of the Conference on Very Large Databases (VLDB)* 5.3 (2011).

[18] Fabian M. Suchanek and David Gross-Amblard. "Adding fake facts to ontologies". In: *World Wide Web Conference (WWW)*. 2012.

[19] Fabian M. Suchanek, David Gross-Amblard, and Serge Abiteboul. "Watermarking for Ontologies". In: *International Semantic Web Conference*. 2011.

[20] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "Yago: A Core of Semantic Knowledge". In: *World Wide Web conference (WWW)*. ACM Press, 2007.

[21] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Large Ontology from Wikipedia and WordNet". In: *Elsevier Journal of Web Semantics* (2008).

[22] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. "SOFIE: A Self-Organizing Framework for Information Extraction". In: *World Wide Web conference (WWW)*. ACM Press, 2009.

Chapter 3

Rule Mining with AMIE

3.1 Overview

Recent advances in information extraction have led to huge knowledge bases (KBs), which capture knowledge in a machine-readable format. Inductive Logic Programming (ILP) can be used to mine logical rules from the KB. These rules can help deduce and add missing knowledge to the KB. While ILP is a mature field, mining logical rules from KBs is different in two aspects: First, current rule mining systems are easily overwhelmed by the amount of data (state-of-the art systems cannot even run on today's KBs). Second, ILP usually requires counterexamples. KBs, however, implement the open world assumption (OWA), meaning that absent data cannot be used as counterexamples. In this chapter, we develop a rule mining model that is explicitly tailored to support the OWA scenario. It is inspired by association rule mining and introduces a novel measure for confidence. Our extensive experiments show that our approach outperforms state-of-the-art approaches in terms of precision and coverage. Furthermore, our system, AMIE, mines rules orders of magnitude faster than state-of-the-art approaches. This chapter is based on

> Luis Galárraga, Christina Teflioudi, Katja Hose, Fabian Suchanek
> "AMIE: Association Rule Mining under Incomplete Evidence
> in Ontological Knowledge Bases"
> (International World Wide Web Conference (WWW) 2013)

3.2 Introduction

In recent years, we have experienced the rise of large knowledge bases (KBs), such as Cyc [23], YAGO [34], DBpedia [5], and Freebase[1]. These KBs provide information about a great variety of entities, such as people, countries, rivers, cities, universities, movies, animals, etc. Moreover, KBs also contain facts relating these entities, e.g., who was born where, which actor acted in which movie, or which city is located in which country. Today's KBs contain millions of entities and hundreds of millions of facts.

[1] http://freebase.com

Yet, even these large KBs are not complete. Some of them are extracted from natural language resources that inevitably exhibit gaps. Others are created and extended manually. Making these KBs complete requires great effort to extract facts, check them for correctness, and add them to the KB. However, KBs themselves often already contain enough information to derive and add new facts. If, for instance, a KB contains the fact that a child has a mother, then the mother's husband is most likely the father:

$$motherOf(m, c) \wedge marriedTo(m, f) \Rightarrow fatherOf(f, c)$$

As for any rule, there can be exceptions, but in the vast majority of cases, the rule will hold. Finding such rules can serve four purposes: First, by applying such rules on the data, new facts can be derived that make the KB more complete. Second, such rules can identify potential errors in the knowledge base. If, for instance, the KB contains the statement that a totally unrelated person is the father of a child, then maybe this statement is wrong. Third, the rules can be used for reasoning. Many reasoning approaches rely on other parties to provide rules (e.g., [31, 27]). Last, rules describing general regularities can help us understand the data better. We can, e.g., find out that countries often trade with countries speaking the same language, that marriage is a symmetric relationship, that musicians who influence each other often play the same instrument, and so on.

The goal of this paper is to mine such rules from KBs. We focus on RDF-style KBs in the spirit of the Semantic Web, such as YAGO [34], Freebase[1], and DBpedia [5]. These KBs provide binary relationships in the form of RDF triples[2]. Since RDF has only positive inference rules, these KBs contain only positive statements and no negations. Furthermore, they operate under the *Open World Assumption* (OWA). Under the OWA, a statement that is not contained in the KB is not necessarily false; it is just *unknown*. This is a crucial difference to many standard database settings that operate under the *Closed World Assumption* (CWA). Consider an example KB that does not contain the information that a particular person is married. Under CWA we can conclude that the person is not married. Under OWA, however, the person could be either married or single.

Mining rules from a given dataset is a problem that has a long history. It has been studied in the context of association rule mining and inductive logic programming (ILP). Association rule mining [3] is well-known in the context of sales databases. It can find rules such as "If a client bought beer and wine, then he also bought aspirin". The confidence of such a rule is the ratio of cases where beer and wine was actually bought together with aspirin. Association rule mining inherently implements a closed world assumption: A rule that predicts new items that are not in the database has a low confidence. It cannot be used to (and is not intended to be used to) add new items to the database.

ILP approaches deduce logical rules from ground facts. Yet, current ILP systems cannot be applied to semantic KBs for two reasons: First, they usually require negative statements as counter-examples. Semantic KBs, however, usually do not contain negative statements. The semantics of RDF are too weak to deduce negative evidence from the facts in a KB.[3] Because of the OWA,

[2] http://www.w3.org/TR/rdf-primer/

[3] RDF has only positive rules and no disjointness constraints or similar concepts.

absent statements cannot serve as counter-evidence either. Second, today's ILP systems are slow and cannot handle the huge amount of data that KBs provide. In our experiments, we ran state-of-the-art approaches on YAGO2 for a couple of days without obtaining any results.

In this paper, we propose a rule mining system that is inherently designed to work under the OWA, and efficient enough to handle the size of today's KBs. More precisely, our contributions are as follows:
(1) A method to simulate negative examples for positive KBs (the Partial Completeness Assumption)
(2) An algorithm for the efficient mining of rules.
(3) A system, AMIE, that mines rules on millions of facts in a few minutes without the need for parameter tuning or expert input.
The rest of this paper is structured as follows. Section 3.3 discusses related work and Section 3.4 introduces preliminaries. Sections 3.5 and 3.6 are the main part of the paper, presenting our mining model and its implementation. Section 3.7 presents our experiments before Section 3.8 concludes.

3.3 Related Work

We aim to mine rules of the form

$$motherOf(m, c) \wedge marriedTo(m, f) \Rightarrow fatherOf(f, c)$$

Technically, these are Horn rules on binary predicates. Rule mining has been an area of active research for the past couple of years. Some approaches mine association rules, some mine logical rules, others mine a schema for the KB, and again others use rule mining for application purposes.

Association Rule Mining. Association rules [3] are mined on a list of *transactions*. A transaction is a set of items. For example, in the context of sales analysis, a transaction is the set of products bought together by a customer in a specific event. The mined rules are of the form $\{ElvisCD,\ ElvisBook\} \Rightarrow ElvisCostume$, meaning that people who bought an Elvis CD and an Elvis book usually also bought an Elvis costume. However, these are not the kind of rules that we aim to mine in this paper. We aim to mine Horn rules.

One problem for association rule mining is that for some applications the standard measurements for support and confidence do not produce good results. [36] discusses a number of alternatives to measure the interestingness of a rule in general. Our approach is inspired by this work and we also make use of a language bias [2] to reduce the search space.

Logical Rule Mining. Sherlock [32] is an unsupervised ILP method to learn first-order Horn clauses from a set of extracted facts for a given target relation. It uses probabilistic graphical models (PGMs) to infer new facts. It tackles the noise of the extracted facts by extensive filtering in a preprocessing step and by penalizing longer rules in the inference part. For mining the rules, Sherlock uses 2 heuristics: statistical significance and statistical relevance.

The WARMR system [12, 11] mines patterns in databases that correspond to conjunctive queries. It uses a declarative language bias to reduce the search space. An extension of the system, WARMER [13], modified the approach to

support a broader range of conjunctive queries and increase efficiency of search space exploration.

ALEPH[4] is a general purpose ILP system, which implements Muggleton's Inverse Entailment algorithm [25] in Prolog. It employs a variety of evaluation functions for the rules, and a variety of search strategies.

These approaches are not tailored to deal with large KBs under the Open World Assumption. We compare our system, AMIE, to WARMR and ALEPH, which are the only ones available for download. Our experiments do not only show that these systems mine less sensible rules than AMIE, but also that it takes them much longer to do so.

Expert Rule Mining. Another rule mining approach over RDF data [28] was proposed to discover causal relations in RDF-based medical data. It requires a domain expert who defines targets and contexts of the mining process, so that the correct transactions are generated. Our approach, in contrast, does not rely on the user to define any context or target. It works out-of-the-box.

Generating Schemas. In this paper, we aim to generate Horn rules on a KB. Other approaches use rule mining to generate the schema or taxonomy of a KB. [7] applies clustering techniques based on context vectors and formal concept analysis to construct taxonomies. Other approaches use clustering [21] and ILP-based approaches [9]. For the friend-of-a-friend network on the Semantic Web, [14] applies clustering to identify classes of people and ILP to learn descriptions of these groups. Another example of an ILP-based approach is the DL-Learner [19], which has successfully been applied [15] to generate OWL class expressions from YAGO [34]. As an alternative to ILP techniques, [37] propose a statistical method that does not require negative examples. In contrast to our approach, these techniques aim at generating a schema for a given RDF repository, not logical rules in general.

Learning Rules From Hybrid Sources. [8] proposes to learn association rules from hybrid sources (RDBMS and Ontologies) under the OWA. For this purpose, the definition of frequency (and thus of support and confidence) is changed so that unknown statements contribute with half of the weight of the true statements. Another approach [20] makes use of an ontology and a constraint Datalog program. The goal is to learn association rules at different levels of granularity w.r.t. the type hierarchy of the ontology. While these approaches focus more on the benefits of combining hybrid sources, our approach focuses on pure RDFS KBs.

Further Applications of Rule Mining. [17] proposes an algorithm for frequent pattern mining in KBs that use DL-safe rules. Such KBs can be transformed into a disjunctive Datalog program, which allows seeing patterns as queries. This approach does not mine the Horn rules that we aim at.

Some approaches use rule mining for ontology merging and alignment [24, 10, 30]. The AROMA system [10], e.g., uses association rules on extracted terms to find subsumption relations between classes and properties of different ontologies. Again, these systems do not mine the kind of rules we are interested

[4] http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph_toc.html

in.

In [1] association rules and frequency analysis are used to identify and classify common misusage patterns for relations in DBpedia. In contrast to our work, this approach does not mine logical rules, but association rules on the co-occurrence of values. Since RDF data can be seen as a graph, mining frequent subtrees [6, 18] is another related field of research. However, as the URIs of resources in knowledge bases are unique, these techniques are limited to mining frequent combinations of classes.

Several approaches, such as Markov Logic [31] or URDF [27] use Horn rules to perform reasoning. These approaches can be consumers of the rules we mine with AMIE.

3.4 Preliminaries

RDF KBs. In this paper, we focus on RDF knowledge bases[5]. An RDF KB can be considered a set of facts, where each fact is a triple of the form $\langle x, r, y\rangle$ with x denoting the subject, r the relation (or predicate), and y the object of the fact. There are several equivalent alternative representations of facts; in this paper we use a logical notation and represent a fact as $r(x, y)$. For example, we write *father(Elvis,Lisa)*.

The facts of an RDF KB can usually be divided into an *A-Box* and a *T-Box*. While the A-Box contains instance data, the T-Box is the subset of facts that define classes, domains, ranges for predicates, and the class hierarchy. Although T-Box information can also be used by our mining approach, we are mainly concerned with the A-Box, i.e., the set of facts relating one particular entity to another.

In the following, we assume a given KB $\mathcal{K}$ as input. Let $\mathcal{R} = \pi_{relation}(\mathcal{K})$ denote the set of relations contained in $\mathcal{K}$ and $\mathcal{E} = \pi_{subject}(\mathcal{K}) \cup \pi_{object}(\mathcal{K})$ the set of entities.

Functions. A *function* is a relation r that has at most one object for every subject, i.e., $\forall x : |\{y : r(x, y)\}| \leqslant 1$. A relation is an *inverse function* if each of its objects has at most one subject. Since RDF KBs are usually noisy, even relations that should be functions (such as *hasBirthdate*) may exhibit two objects for the same subject. Therefore, we use the notion of *functionality* [33]. The functionality of a relation r is a value between 0 and 1, that is 1 if r is a function:

$$fun(r) := \frac{\#x : \exists y : r(x, y)}{\#(x, y) : r(x, y)}$$

with $\#x : X$ as an abbreviation for $|\{x : X \in \mathcal{K}\}|$. The inverse functionality is defined accordingly as $ifun(r) := fun(r^{-1})$. Without loss of generality, we assume that $\forall r \in \mathcal{R} : fun(r) \geqslant ifun(r)$ (*FUN-Property*). If that is not the case for a relation r, we can replace all facts $r(x, y)$ with the inverse relation, $r^{-}(y, x)$, which entails $fun(r^{-}) \geqslant ifun(r^{-})$. For example, if the KB contains the inverse functional relation *directed(person,movie)*, we can create the functional relation *isDirectedBy(movie,person)* and use only that one in the rule mining process.

[5] http://www.w3.org/TR/rdf-primer/

Manual inspection shows, however, that relations in semantic KBs tend to be more functional than inverse functional. Intuitively, this allows us to consider a fact $r(x, y)$ as a fact about x.

Rules. An *atom* is a fact that can have variables at the subject and/or object position. A *(Horn) rule* consists of a head and a body, where the head is a single atom and the body is a set of atoms. We denote a rule with head $r(x, y)$ and body $\{B_1, ..., B_n\}$ by an implication

$$B_1 \wedge B_2 \wedge ... \wedge B_n \Rightarrow r(x, y)$$

which we abbreviate as $\vec{B} \Rightarrow r(x, y)$. One example of such a rule is

$$hasChild(p, c) \wedge isCitizenOf(p, s) \Rightarrow isCitizenOf(c, s)$$

An *instantiation* of a rule is a copy of the rule, where all variables have been substituted by entities. A *prediction* of a rule is the head atom of an instantiated rule if all body atoms of the instantiated rule appear in the KB. For example, the above rule can predict *isCitizenOf(Lisa,USA)* if the KB knows a parent of Lisa (*hasChild(Elvis,Lisa)*) who is American (*isCitizenOf(Elvis,USA)*).

Language Bias. As most ILP systems, AMIE uses a language bias to restrict the search space. We say that two atoms in a rule are *connected* if they share a variable or an entity. A rule is *connected* if every atom is connected transitively to every other atom of the rule. AMIE mines only connected rules, i.e., it avoids constructing rules that contain unrelated atoms. We say that a rule is *closed* if every variable in the rule appears at least twice. Such rules do not predict merely the existence of a fact (e.g. $diedIn(x, y) \Rightarrow \exists z : wasBornIn(x, z)$), but also concrete arguments for it (e.g. $diedIn(x, y) \Rightarrow wasBornIn(x, y)$). AMIE mines only closed rules. We allow *recursive rules* that contain the head relation in the body.

Parallels to Association Rule Mining. Association Rule Mining discovers correlations in shopping transactions. Thus, association rules are different in nature from the Horn rules we aim at. Still, we can show some similarities between the two approaches. Let us define one transaction for every set of n entities that are connected in the KB. For example, in Figure 1, we will define a transaction for the entities *Elvis*, *Lisa* and *Priscilla*, because they are connected through the facts *mother(Priscilla,Lisa)*, *father(Elvis,Lisa)*, *marr(Elvis, Priscilla)*. We label the transaction with the set of these entities. Each atom $r(x_i, x_j)$ on variables indexed by $1 \leqslant i, j \leqslant n$ corresponds to an item. A transaction with label $\langle C_1, \ldots, C_n \rangle$ contains an item $r(x_i, x_j)$ if $r(C_i, C_j)$ is in the KB. For example, the transaction $\langle$*Elvis, Lisa, Priscilla*$\rangle$ contains the items {*mother*(x_3,x_2), *father*(x_1,x_2), *marr*(x_1,x_3)}, since the ground atoms *mother(Priscilla,Lisa)*, *father(Elvis,Lisa)* and *marr(Elvis, Priscilla)* are in the KB. In this representation, association rules are Horn rules. In the example, we can mine the association rule

$$\{mother(x_3, x_2), marr(x_1, x_3)\} \Rightarrow \{father(x_1, x_2)\}$$

which corresponds to the Horn rule

$$mother(x_3, x_2) \wedge marr(x_1, x_3) \Rightarrow father(x_1, x_2)$$

Transaction Label	Transaction Items
⟨Elvis,Lisa,Priscilla⟩	{mother(x_3,x_2),father(x_1,x_2),marr(x_1,x_3)}
⟨Barack,Mali,Mich.⟩	{mother(x_3,x_2),father(x_1,x_2),marr(x_1,x_3)}
⟨François,Flora,Ségo⟩	{mother(x_3,x_2),father(x_1,x_2)}

Figure 1: Mining Rules with 3 Variables

Constructing such a table with all possible combinations of entities is practically not very viable. Apart from that, it faces a number of design issues (e.g., how to deal with transactions that contain the same entities in different orderings). Therefore, association rule mining cannot be used directly to mine Horn rules. However, we take inspiration from the parallels between the two types of mining for our system, AMIE.

3.5 Mining Model

Model. Let us consider a given Horn rule $\vec{B} \Rightarrow r(x, y)$. Let us look at all facts with relation r (Figure 2). We distinguish 4 types of facts: True facts that are known to the KB (KBtrue), true facts that are unknown to the KB (NEWtrue), facts that are known to be false in the KB (KBfalse), and facts that are false but unknown to the KB (NEWfalse). The rule will make certain predictions (blue circle). These predictions can be known to be true (A), known to be false (C), or unknown (B and D). When they are unknown to the KB, they can still be true (B) or false (D) with respect to the real world.

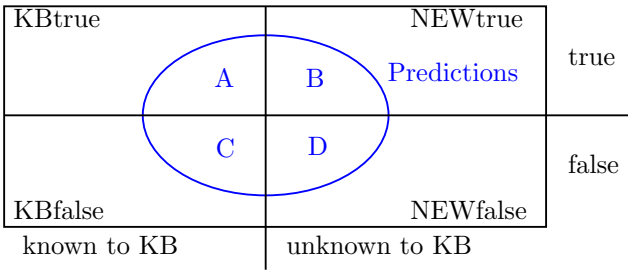

Figure 2: Prediction under Incompleteness

Goal. Our goal is to find rules that make true predictions that go beyond the current KB. In the figure, we wish maximize the area B, and to minimize the area D. There are two obvious challenges in our context: First, the areas NEWtrue and NEWfalse are unknown. So if we wish to maximize B at the expense of D, we are operating in an area outside our KB. We would want to use the areas KBtrue and KBfalse to estimate the unknown area. This, however, leads to the second challenge: Semantic KBs do not contain negative evidence. Thus, the area KBfalse is empty. We will now present different measures that address these challenges.

Support. The *support* of a rule quantifies the number of correct predictions, i.e., the size of A. There are several ways to define the support: It can be the

number of instantiations of the rule that appear in the KB. This is what our analogy to association rule mining [3] suggests (Section 3.4). This measure, however, is not monotonic if we add atoms to the body. Consider, for example, the rule

$$marriedTo(x, y) \Rightarrow marriedTo(y, x)$$

If we add *hasGender(x,male)* to the body, the number of instantiations that are in the KB decreases. If we add an atom with a fresh variable, e.g., *hasFriend(x,z)*, to the body, the number of instantiations increases for every friend of x. This is true even if we add another atom with z to make the rule closed. Alternatively, we can count the number of facts in one particular body atom. This definition, however, depends on the choice of the body atom, so that the same rule can have different supports. We can also count the number of facts of the head atom. This measure decreases monotonically if more body atoms are added and avoids equivalent rules with different support values. With this in mind, we define the support of a rule as the number of distinct pairs of subjects and objects in the head of all instantiations that appear in the KB:

$$supp(\vec{B} \Rightarrow r(x, y)) := \#(x, y) : \exists z_1, ..., z_m : \vec{B} \wedge r(x, y)$$

where $z_1, ..., z_m$ are the variables of the rule apart from x and y.

Head Coverage. Support is an absolute number. This means that a user who thresholds on support has to know the absolute size of the KB to give meaningful values. To avoid this, we also define a proportional version of support. A naive way would be to use the absolute number of support, as defined in the previous paragraph, over the size of the KB. In this case, however, relations that do not have many facts (either because of the incompleteness of the KB or because of their nature), will not be considered in the head of rules, i.e. we will not learn rules predicting such relations. Therefore, we propose to use the notion of *head coverage*. This is the proportion of pairs from the head relation that are covered by the predictions of the rule

$$hc(\vec{B} \Rightarrow r(x, y)) := \frac{supp(\vec{B} \Rightarrow r(x, y))}{\#(x', y') : r(x', y')}$$

Negative Examples. The central challenge of our setting is to provide counter-examples for the rule mining. These can take the role of KBfalse, so that we can estimate the areas NEWtrue and NEWfalse. There are several approaches to this problem: The standard confidence, the standard positive-only learning evaluation score of ILP, and our new partial completeness assumption.

Standard Confidence. The standard confidence measure takes all facts that are not in the KB (i.e., NEWtrue and NEWfalse) as negative evidence. Thus, the standard confidence of a rule is the ratio of its predictions that are in the KB, i.e., the share of A in the set of predictions:

$$conf(\vec{B} \Rightarrow r(x, y)) := \frac{supp(\vec{B} \Rightarrow r(x, y))}{\#(x, y) : \exists z_1, ..., z_m : \vec{B}}$$

The standard confidence is blind to the distinction between "false" and "unknown". Thus, it implements a closed world setting. It mainly describes the known data and penalizes rules that make a large number of predictions in the unknown region. We, in contrast, aim to maximize the number of true predictions that go beyond the current knowledge. We do not want to describe data, but to predict data.

Positive-Only Learning. For cases where the KB does not contain negative examples, Muggleton has developed a *positive-only learning evaluation score* for ILP [26],[22]. It takes random facts as negative evidence:

$$Score = log(P) - log\frac{R+1}{Rsize+2} - \frac{L}{P}$$

Here, P is the number of known true facts covered (A in the figure), R is the number of randoms covered, $Rsize$ is the total number of randoms and L is the number of atoms in the hypothesis. The intuition is that a good rule should cover many positive examples, and few or no randomly generated examples. This ensures that the rule is not overly general. Furthermore, the rule should use as few atoms as possible, and thus achieve a high compression. This measure is implemented (among others) in the ALEPH system.

Partial Completeness. We propose to generate negative evidence by the *partial completeness assumption* (PCA). This is the assumption that if $r(x, y) \in KBtrue$ for some x, y, then

$$\forall y' : r(x, y') \in KBtrue \cup NEWtrue \Rightarrow r(x, y') \in KBtrue$$

In other words, we assume that if the database knows some r-attribute of x, then it knows all r-attributes of x. This assumption is certainly true for functional relations r, such as birth dates, capitals, etc. Thanks to the FUN-Property (see Section 3.5), it is also true for inverse-functional relations, such as *owns*, *created*, etc. The assumption is also true in the vast majority of cases for relations that are not functional, but that have a high functionality. Even for other relations, the PCA is still reasonable for knowledge bases that have been extracted from a single source (such as DBpedia and YAGO). These usually contain either all r-values or none for a given entity.

PCA Confidence. Under the PCA, we normalize the confidence not by the entire set of facts, but by the set of facts of which we know that they are true, together with the facts of which we assume that they are false. If the head atom of the rule is $r(x, y)$, then this set is just the set of facts $\{r(x, y') : r(x, y') \in \mathcal{K}\}$. Thanks to the FUN-Property, the PCA is always applied to the first argument of the head atom:

$$pcaconf(\vec{B} \Rightarrow r(x, y)) := \frac{supp(\vec{B} \Rightarrow r(x, y))}{\#(x, y) : \exists z_1, ..., z_m, y' : \vec{B} \wedge r(x, y')}$$

We show in our experiments that the PCA confidence identifies much more productive rules than the other measures.

3.6 AMIE

After having outlined the basic definitions and the mining model in Sections 3.4 and 3.5, we now outline the core algorithm of our framework and its implementation.

3.6.1 Algorithm

Goal. Our goal is to mine rules of the form defined in Section 3.4. One of the main problems of any mining approach is to find an efficient way to explore the search space. The naive algorithm of enumerating all possible rules is infeasible for large KBs. Hence, we explore the search space by iteratively extending rules by *mining operators*.

Mining Operators. We see a rule as a sequence of atoms. The first atom is the head atom and the others are the body atoms. In the process of traversing the search space, we can extend a rule by using one of the following operators:

1. **Add Dangling Atom ($\mathcal{O}_D$)**
   This operator adds a new atom to a rule. The new atom uses a fresh variable for one of its two arguments. The other argument (variable or entity) is shared with the rule, i.e., it occurs in some other atom of the rule.

2. **Add Instantiated Atom ($\mathcal{O}_I$)**
   This operator adds a new atom to a rule that uses an entity for one argument and shares the other argument (variable or entity) with the rule.

3. **Add Closing Atom ($\mathcal{O}_C$)**
   This operator adds a new atom to a rule so that both of its arguments are shared with the rule.

By repeated application of these operators, we can generate the entire space of rules as defined in Section 3.4. The operators generate even more rules than those we are interested in, because they also produce rules that are not closed. An alternative set of operators could consist of $\mathcal{O}_D$ and an operator for instantiation. But these operators would not be monotonic, in the sense that an atom generated by one operator can be modified in the next step by the other operator. Therefore, we chose the above 3 operators as a canonic set.

Algorithm. We mine rules with Algorithm 1. The algorithm maintains a queue of rules, which initially just contains the empty rule. The algorithm iteratively dequeues a rule from the queue. If the rule is closed (see Section 3.4), the rule is output, otherwise, it is not. Then, the algorithm applies all operators to the rule and adds the resulting rules to the queue (unless they are pruned out, s.b.). This process is repeated until the queue is empty. We parallelize this process by maintaining a centralized queue, from which the threads dequeue and enqueue. We do not feed predictions of the rules back into the KB. All measures (such as confidence and support) are always computed on the original KB.

Algorithm 1 Rule Mining

```
1: function AMIE(KB K)
2:     q = ⟨[]⟩
3:     Execute in parallel:
4:     while ¬q.isEmpty() do
5:         r = q.dequeue()
6:         if r is closed ∧ r is not pruned for output then
7:             Output r
8:         end if
9:         for all operators o do
10:            for all rules r′ ∈ o(r) do
11:                if r′ is not pruned then
12:                    q.enqueue(r′)
13:                end if
14:            end for
15:        end for
16:    end while
17: end function
```

Pruning. If executed naively, our algorithm will have prohibitively high runtimes. The instantiation operator $\mathcal{O}_I$, in particular, generates atoms in the order of $|\mathcal{R}| \times |\mathcal{E}|$. We first observe that we are usually not interested in rules that cover only very few facts of the head relation. Rules that cover, for example, less than 1% of the facts of the head relation can safely assumed to be marginal. Therefore, we set $\theta = 0.01$ as a lower bound for the head coverage. We observe that head coverage decreases monotonically as we add more atoms. This allows us to safely discard any rule that trespasses the threshold (Lines 11 and 12).

The monotonicity of head coverage gives us another opportunity to prune: If a rule $B_1 \wedge ... \wedge B_n \wedge B_{n+1} \Rightarrow H$ does not have larger confidence than the rule $B_1 \wedge ... \wedge B_n \Rightarrow H$, then we do not output the longer rule. This is because both the confidence and the head coverage of the longer rule are necessarily dominated by the shorter rule. This way, we can reduce the number of produced rules (Lines 6 and 7).

Last, we never enqueue a rule that is already in the queue. It is expensive to check two rules for equality. However, it is easy to compute measures such as head coverage, confidence, and PCA confidence for each rule. Two rules can only be equal if they have the same values for these measures. This restricts the rules that have to be checked. If a rule is duplicate, we do not enqueue it (Lines 11 and 12). We can be sure that any potential duplicates will still be in the queue. This is because the length of the rules increases monotonically: When we dequeue a rule with n atoms, no rule with $n+1$ atoms has ever been dequeued. Thus, when we apply the operators to the rule with n atoms, and generate a rule with $n+1$ atoms, any potential duplicate of that new rule must be in the queue.

Projection Queries. No matter what operator is applied in particular, the algorithm needs to choose a relation for the new atom that is added to the rule. In addition, the instantiation operator $\mathcal{O}_\mathcal{I}$ also allows the choice of an entity.

In order to select only relations and entities that will fulfill the head coverage constraint, we rely on the KB to answer *projection queries*. These are queries of the form

SELECT $?x$ WHERE $H \wedge B_1 \wedge ... \wedge B_n$
HAVING COUNT$(H) \geqslant k$

where $B_1, ..., B_n$ are atoms and k is a natural number. H is the *projection atom* on which we project. $?x$ is the *selection variable*. It is a variable that appears in one or more atoms at the position of one of the arguments or at the position of the relation (as it is common in SPARQL[6]). Such queries select an entity or relation x such that the result of the query $H \wedge B_1 \wedge ... \wedge B_n$ on the KB contains more than k distinct query answers for H.

Using Projection Queries. Projection queries allow us to select the relationship for the operators $\mathcal{O}_\mathcal{D}$, $\mathcal{O}_\mathcal{I}$, and $\mathcal{O}_\mathcal{C}$ in such a way that the head coverage of the resulting rule is above θ. This works by firing a projection query of the form

SELECT $?r$ WHERE $H \wedge B_1 \wedge ... \wedge B_n \wedge ?r(X, Y)$
HAVING COUNT$(H) \geqslant k$

where X and Y are variables or constants, depending on the type of atoms that the operator generates. The results for $?r$ will be the relations that, once bound in the query, ensure that the support of the rule $B_1 \wedge ... \wedge B_n \wedge ?r(X, Y) \Rightarrow H$ is greater than k. If we choose k equal to θ times the number of facts of the relation of H, then the head coverage of the resulting rules will be greater than θ – which is what we want. For the instantiation operator $\mathcal{O}_\mathcal{I}$, we first fire a projection query to select relations, and then fire projection queries to retrieve entities. This way, projection queries allow us to choose the relationships and entities for the operators in such a way that the head coverage for the new rules is guaranteed to be above θ. Next, we discuss how to implement projection queries efficiently.

3.6.2 Implementation

SQL and SPARQL. Projection queries are essential for the efficiency of our system. Yet, standard database implementations do not provide special support for these types of queries. Assuming that the KB $\mathcal{K}$ is stored as a three-column table (i.e., each fact is a row with three elements), the projection query template in SQL would be:

SELECT $?x$
FROM $\mathcal{K}$ AS H, $\mathcal{K}$ AS $B_1, \ldots B_n$
WHERE $H.x_i = B_j.x_m, \ldots$
GROUP BY$(H.x_1, H.x_r, H.x_2)$
HAVING COUNT$(*) \geqslant k$

where $?x$ is replaced with a reference to any of the introduced columns. The WHERE clause lists all variables that are shared between any two atoms in the

[6] http://www.w3.org/TR/rdf-sparql-query/

rule, i.e., all join columns and conditions between atom tables. Since SELECT can only select variables that appear in the GROUP BY statement, the above template is for the case where $?x$ appears in H. The case where $?x$ does not appear in H will require a nested query. Our experience shows that already running the non-nested query on a database of a few million facts can easily take several minutes on an off-the-shelf RDBMS. Hence, efficient SPARQL engines such as RDF-3X [29] are an alternative option. In SPARQL 1.1, the projection query template is:

SELECT $?x$
WHERE {
  $H.x_1$, $H.x_r$, $H.x_2$.
  $B_1.x_1$, $B_1.x_r$, $B_1.x_2$.
  ...
  $B_n.x_1$, $B_n.x_r$, $B_n.x_2$.
}
GROUP BY $H.x_1$ $H.x_r$ $H.x_2$
HAVING COUNT(*) $\geqslant k$

Again, this is only for the case where $?x$ appears in H. RDF-3X does not support aggregate functions in this way. Thus, we would need extensive postprocessing of query results to compute a projection query – already in the case where $?x$ is in H.

In-Memory Database. We have implemented a vanilla in-memory database for semantic KBs. Our implementation indexes the facts aggressively with one index for each permutation of subject, relation, and object. Each index is a map from the first item to a map from the second item to a set of third items (e.g., a map from relations to a map from subjects to a set of objects). This allows retrieving the instantiations of a single atom in constant time. The existence of a query answer can be checked naively by selecting the atom with fewest instantiations, running through all of its instantiations, instantiating the remaining atoms accordingly, and repeating this process recursively until we find an instantiation of the query that appears in the KB. Select queries are similar.

Projection Queries. Algorithm 2 shows how we answer projection queries. The algorithm takes as input a selection variable $?x$, a projection atom $H = R(X, Y)$, remaining atoms $B_1, ...B_n$, a constant k, and a KB $\mathcal{K}$. We first check whether $?x$ appears in the projection atom. If that is the case, we run through all instantiations of the projection atom, instantiate the query accordingly, and check for existence. Each existing instantiation increases the counter for the respective value of $?x$. We return all values whose counter exceeds k. If the selection variable does not appear in the projection atom, we iterate through all instantiations of the projection atom. We instantiate the query accordingly, and fire a SELECT query for $?x$. We increase the counter for each value of $?x$. We report all values whose counter exceeds k.

Summary. We have identified projection queries as the crucial type of queries for rule mining. Since standard database systems and standard SPARQL systems provide no specifically tuned support for these queries, we have imple-

mented a vanilla in-memory database, which has specific support for projection queries. Our entire implementation is in Java.

Algorithm 2 Answering Projection Queries

```
function SELECT(?x, R(X,Y) ∧ B1 ∧ ... ∧ Bn, k, K)
    map = ∅
    if R ≡ ?x ∨ X ≡ ?x ∨ Y ≡ ?x then
        for all instantiations r(x,y) of R(X,Y) ∈ K do
            q = B1 ∧ ... ∧ Bn
            In q, replace R by r, X by x, Y by y
            if exists instantiation q ∈ K then
                map(value of ?x) + +
            end if
        end for
    else
        for all instantiations r(x,y) of R(X,Y) ∈ K do
            q = B1 ∧ ... ∧ Bn
            In q, replace R by r, X by x, Y by y
            for all x ∈ SELECT ?x FROM K WHERE q do
                map(x) + +
            end for
        end for
    end if
    return {x : map(x) ⩾ k}
end function
```

3.7 Experiments

3.7.1 Overview

Experiments. We conducted 3 groups of experiments: In the first group, we compare AMIE to two popular, state-of-the-art systems that are publicly available, WARMR [12, 11] and ALEPH[4]. In the second group of experiments, we compare the standard confidence to the novel PCA confidence that we have introduced in this paper (Section 3.5). In the third group of experiments, we run AMIE on different datasets to show the applicability of the system.

Settings. By default, AMIE finds all rules whose head coverage exceeds the default threshold of $\theta = 1\%$. AMIE ranks the resulting rules by decreasing PCA confidence. There is no need to deviate from this default configuration when a user runs AMIE. There are no parameters to tune. All experiments with AMIE on all datasets are run in this setting, unless otherwise mentioned.

For some experiments, we want to compare AMIE's runtime with other systems. To have an equal basis, we make AMIE simulate the metrics of the competitor systems. AMIE can threshold on support, head coverage, confidence, and PCA confidence, and she can rank by any of these. AMIE can also count the support not on two variables, but on a single variable. AMIE can also

output non-closed rules. Since this is just a choice of what to output, it does not influence runtime. All experiments with all systems are run on a server with 48GB RAM and 8 CPUs. We always mine rules without constants (i.e., without the instantiation operator), unless otherwise mentioned.

Knowledge Bases. We run our experiments on different KBs. In all cases, we removed the *rdf:type* relationship, because it inflates the size of the KBs. We are aware that the *rdf:type* relationship can be very helpful for rule mining. However, currently no approach (including ours) makes specific use of it. We plan to make use of it in future work. Furthermore, we removed all facts with literals (numbers and strings) from the KBs. Literal values (such as geographical coordinates) are shared by only very few entities, which makes them less interesting for rule mining.

Evaluations. In all experiments, our goal is twofold: First, we want to produce as many predictions as possible beyond the current KB. Second, the percentage of correct predictions shall be as large as possible. The particular challenge is that we want to evaluate *predictions that go beyond the current KB*. We are not interested in describing the existing data, but in generating new data. Therefore, we proceed as follows: We run the systems on an older dataset (YAGO2 [16]). We generate all predictions, i.e., the head atoms of the instantiated rules (see Section 3.4). We remove all predictions that are in the old KB. Then we compare the remaining predicted facts to the successor of that dataset (YAGO2s [35]). A prediction is "correct" if it appears in the newer KB. A prediction is "incorrect" if it has a highly functional or highly inverse functional relation and contradicts an existing fact in the newer KB, e.g., a different birth place. For all other predictions, we manually validated the facts by checking a sample of 30 of them against Wikipedia pages. This classifies the remaining predictions as "correct" or "incorrect" – except for a few cases where the fact is "unknown", such as the death place of a person that is still alive. The ratio of correct predictions out of the correct and incorrect predictions yields the *precision* of the rule.

Outlook. We note that with the project of predicting beyond current knowledge, we are entering a new, and very risky area of research. We do not expect Horn rules to have extraordinary precisions in the unknown region. Rules can only yield hypotheses about possible facts.

3.7.2 AMIE vs. WARMR and ALEPH

In this section, we compare AMIE to WARMR and ALEPH. For each system, we conduct 3 experiments: We first compare the usability of the competitor system to AMIE. Then, we compare their runtimes. Last, we compare their outputs.

3.7.2.1 AMIE vs. WARMR

Usability. WARMR is a system that unifies ILP and association rule mining. Similar to APRIORI algorithms [4], it performs a breadth-first search in order

to find frequent patterns. WARMR generates Datalog queries of the form "? – $A_1, A_2, ..., A_n$", where A_i are logical atoms.

To discover frequent patterns (as in association rule mining), we need to have a notion of frequency. Given that WARMR considers queries as patterns and that queries can have variables, it is not immediately obvious what the frequency of a given query is. Therefore, the user needs to specify the predicate that is being counted by the system (the *key predicate*). In the usual scenario of market basket analysis, e.g., the system counts customer transactions. In a scenario in which the database is a KB, one solution is to count entities. Since the key predicate determines what is counted, it is necessary that it is contained in all queries. Therefore, we add a predicate $entity(x)$, which we fill with all entities of the KB. AMIE does not require such a choice.

For WARMR, the user needs to provide specific information about which predicates can be added to a query, which of their variables can be fresh, and which arguments of predicates are allowed to be unified (type declarations). In contrast, AMIE requires none of these. AMIE simply takes as input the KB in triple format.

WARMR is also able to mine rules with constants. The user can define which predicates and arguments should be instantiated with constants (we call this mode MODE1). WARMR then checks all the constants appearing in the facts of that specific predicate and argument and afterwards uses them in the queries. MODE1 naturally entails an increase of the branching factor in the search space and an explosion in the number of candidates that need to be evaluated. Alternatively, WARMR allows the user to set a maximum number of constants to be used for each predicate and argument (MODE2). Unfortunately, though, it does not provide a way for the user to influence the selection of these constants. In other words, there is no guarantee that the constants that WARMR will use are the most promising ones.

WARMR produces rules as output. These rules are not necessarily connected. For example, WARMR mines

$$isMarriedTo(\text{B,C}), \wedge\ isLeaderOf(\text{A,D}) \\ \Rightarrow hasAcademicAdvisor(\text{C,E})$$

This rule is not only nonsensical from a semantic perspective, but also redundant, because the second atom does not influence the implication. Therefore, the user has to filter out these rules from the output.

Thus, we conclude that the broader mission and the broader applicability of WARMR entails that much more configuration, acquaintance, and expert knowledge is needed to make it mine Horn rules on semantic KBs.

Runtime. YAGO2[16] contains around 940K facts about 470K entities. WARMR was not able to terminate on this data in a time period of 1 day. Therefore, we created a sample of YAGO2. Randomly selecting a number of facts from the initial dataset could break the interesting links between the entities. Therefore, we randomly selected 10,000 seed entities and included their 3-hop neighborhood. This yielded 14K entities and 47K facts. This sample contains all available information in a radius of 3 hops around the seed entities, but much less information about the entities at the periphery of the subgraph. Therefore, we restricted the values for the key predicate to the seed entities only.

Since the sample is much smaller than the original KB, we lowered the support threshold to 5 entities. We ran AMIE with these parameters on the sample. AMIE mined her rules in 3.90 seconds. WARMR, in contrast, took 18 hours. We also ran both systems allowing them to mine rules with constants. AMIE completed the task in 1.53 minutes. WARMR in MODE1 for all relations did not terminate in 3 days. Therefore, we ran it also only for the relations *diedIn*, *livesIn*, *wasBornIn*, for which it took 48h. We also ran WARMR in MODE2. To have reasonable runtimes, we allowed WARMR to find constants only for one predicate (*diedIn*). We also restricted it to find only 20 constants. WARMR ran 19 hours. Table 3 summarizes the runtime results. We conclude that AMIE is better suited for large KBs than WARMR. This is because WARMR is an ILP algorithm written in a logic programming environment, which makes the evaluation of all candidate queries inefficient.

Constants	WARMR	AMIE
no	18h	3.90s
yes	(48h) / (19.3h)	1.53min

Table 3: Runtimes on YAGO2 Sample

Results. After filtering out non-connected rules, WARMR mined 41 closed rules. AMIE, in contrast, mined 207 closed rules, which included the ones mined by WARMR. We checked back with the WARMR team and learned that for a given set of atoms $B_1, ...B_n$, WARMR will mine only one rule, picking one of the atoms as head atom (e.g., $B_1 \wedge ... \wedge B_{n-1} \Rightarrow B_n$). AMIE, in contrast, will mine one rule for each possible choice of head atom (as long as the thresholds are met). In other words, AMIE with the standard support and confidence measures simulates WARMR, but mines more rules. Furthermore, it runs orders of magnitude faster. Especially for large datasets for which the user would have needed to use complicated sampling schemes in order to use WARMR, AMIE can be a very attractive alternative. Even for smaller datasets with rules with constants, AMIE can provide results while WARMR cannot. Moreover, AMIE comes with metrics that go beyond the standard confidence and the standard support. We will show later that these improve the quality of the results.

3.7.2.2 AMIE vs. ALEPH

Usability. ALEPH can be run with different commands that influence the search strategy. We chose the *induce* command, which runs fastest. For running ALEPH, the user has to specify the target predicate for learning (the head predicate of the rules). In the following, we ran ALEPH successively with all predicates of the KB as targets. In addition, the user has to specify a series of type and mode declarations (similar to WARMR), which will be used as a language bias in order to restrict the search space. In addition, the user needs to provide ALEPH with files containing the background knowledge and positive examples for the target predicate. In contrast, AMIE requires no such input. It will run on a KB without any prespecified choices of predicates.

KB	Facts	ALEPH	AMIE
YAGO2 full	948k	4.96s to > 1 day	3.62min
YAGO2 Sample	47k	0.05s to > 1 day	5.41s

Table 4: Runtimes ALEPH vs. AMIE

Relations	Runtime
isPoliticianOf, hasCapital, hasCurrency	< 5min
dealsWith, hasOfficialLanguage, imports	< 5min
isInterested, hasMusicalRole	<19min
hasAcademicAdvisor, hasChild	> 1 day
isMarriedTo, livesIn, worksAt, isLocatedIn	> 1 day

Table 5: Runtimes of ALEPH on YAGO2

Relations	Runtime
diedIn, directed, hasAcademicAdvisor	< 2min
graduatedFrom, isPoliticianOf, playsFor	< 2min
wasBornIn, worksAt, isLeaderOf	< 2min
exports, livesIn, isCitizenOf	< 1.4h
actedIn, produced, hasChild, isMarriedTo	> 1 day

Table 6: Runtimes of ALEPH on YAGO2 Sample

Runtime. We ran AMIE and ALEPH on YAGO2 [16]. For ALEPH, we used the positive-only evaluation function with $Rsize = 50$ and we considered only clauses that were able to explain at least 2 positive examples, so that we will not get grounded facts as rules in the output. For a fair comparison, we also instructed AMIE to run with a support threshold of 2 facts. AMIE terminated in 3.62 minutes, and found rules for all relations. ALEPH ran for one head relation at a time. For some relations (e.g.*isPoliticianOf*), it terminated in a few seconds. For others, however, we had to abort the system after 1 day without results (Tables 4 and 5). For each relation, ALEPH treats one positive example at a time. Some examples need little processing time, others block the system for hours. We could not figure out a way to choose examples in such a way that ALEPH runs faster. Hence, we used the sample of YAGO2 that we created for WARMR. Again, runtimes varied widely between relations (Table 6). Some relations ran in a few seconds, others did not terminate in a day. The runtimes with constants are similarly heterogenous, with at least 7 relations not terminating in 1 day.

Results. We compared the output of ALEPH on the head relations for which it terminated to the output of AMIE on these head relations, on the sample dataset. ALEPH mined 56 rules, while AMIE mined 335 rules. We order the rules by decreasing score (ALEPH) and decreasing PCA confidence (AMIE). Table 7 shows the number of predictions, and their total precision as described in Section 3.7.1. We show the aggregated values at the points where both approaches have produced around 3K, 5K and 8K predictions. AMIE's PCA confidence succeeds in sorting the rules roughly by descending precision, so that the initial rules have an extraordinary precision compared to ALEPH's.

AMIE needs more rules to produce the same number of predictions as ALEPH (but she also mines more). We suspect that ALEPH's positives-only evaluation function manages to filter out overly general rules only to some extent. ALEPH will mine, e.g, $livesIn(A, C)$,$isLocatedIn(C, B) \Rightarrow isPoliticianOf(A, B)$. The problem is that ALEPH generates counterexamples by randomly using valid constants for variables A and B. This means that the probability of creating a random example in which B is the place of residence of the specific person A is very low.

System	Top n	Predictions	Precision
ALEPH	7	2997	27%
AMIE	13	3180	66%
ALEPH	9	5031	26%
AMIE	29	5003	47%
ALEPH	17	8457	30%
AMIE	52	8686	45%

Table 7: Top Rules of ALEPH vs. AMIE

3.7.3 AMIE with Different Metrics

In this section, we compare the standard confidence measure to the PCA confidence measure. We ran AMIE with the default head coverage threshold on the YAGO2 dataset. It contains nearly 500K entities and 948K facts. We sort the rules first by descending PCA confidence, and then by descending standard confidence, and look at the top rules. For each rule, we evaluated the predictions beyond YAGO2 as described in Section 3.7.1. Figure 8 uses aggregated predictions and aggregated precision to illustrate the results. The n-th dot from the left tells us the total number of predictions and the total precision of these predictions, aggregated over the first n rules. As we see, ranking the rules by standard confidence is a very conservative approach: It identifies rules with reasonable precision, but these do not produce many predictions. Going down in the list of ranked rules, the rules produce more predictions – but at lower precision. The top 30 rules produce 113K predictions at an aggregated precision of 32%. If we rank the rules by PCA confidence, in contrast, we quickly get large numbers of predictions. The top 10 rules already produce 135K predictions – at a precision of 39%. The top 30 rules produce 3 times more predictions than the top 30 rules by standard confidence – at comparable precision. This is because the PCA confidence is less conservative than the standard confidence.

Discussion. The precision of the rules is in the range of 30%-40%. Thus, only a third of the predictions in the unknown region will be correct. Still, even imperfect rules can be useful: If, e.g., a human checks the facts before they are added, then reducing the number of false predictions is a great advantage. If games with a purpose are employed, then the rules can help pre-select candidate facts. If multiple sources are combined, then rules can contribute evidence. If reasoning approaches are used, then rules can be taken into consideration according to their estimated performance. Finally, the precision is better if standard confidence is used.

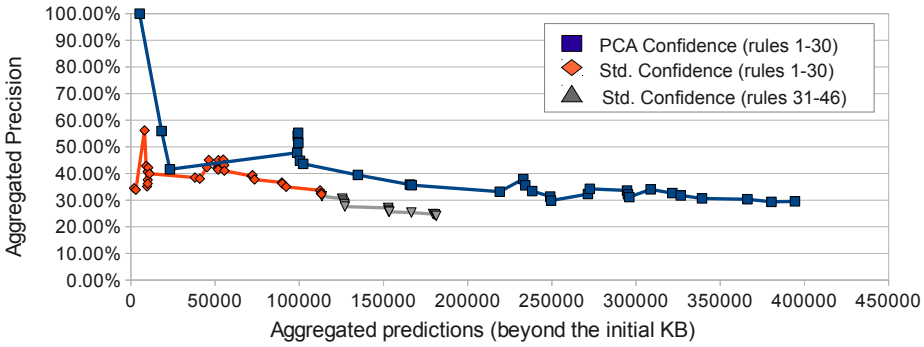

Figure 8: Std. Confidence vs. PCA Confidence

Predicting Precision. The confidence measures can serve to estimate the actual precision of a rule. In Table 9, we rank the mined rules by their precision and report the average absolute error of the standard and PCA confidence weighted by the number of predictions produced by the rules. We can observe that, on average, the PCA confidence estimates the precision of the rules better than the normal confidence. Thus, reasoning approaches can use the PCA confidence as a weight for the rule.

	Top 20 rules	Top 30 rules	All rules
Confidence	0.76	0.63	0.33
PCA Confidence	0.32	0.29	0.29

Table 9: Average Absolute Error to Precision

We also note that our rules are insightful. Table 10 shows some of the rules we mined. Being able to mine reasonable rules on semantic KBs of this size is an achievement beyond the current state of the art.

$isMarriedTo(x, y) \wedge livesIn(x, z) \Rightarrow livesIn(y, z)$
$isCitizenOf(x, y) \Rightarrow livesIn(x, y)$
$hasAdvisor(x, y) \wedge graduatedFrom(x, z) \Rightarrow worksAt(y, z)$
$wasBornIn(x, y) \wedge isLocatedIn(y, z) \Rightarrow isCitizenOf(x, z)$
$hasWonPrize(x, G.\ W.\ Leibniz) \Rightarrow livesIn(x, Germany)$
$hasWonPrize(x, Grammy) \Rightarrow hasMusicalRole(x, Guitar)$

Table 10: Some Rules by AMIE

3.7.4 AMIE on Different Datasets

As a proof of concept, we ran AMIE on YAGO2 [16], YAGO2 with constants, and DBpedia [5]. We chose an older version of DBpedia (2.0), so that we can evaluate the output to a newer version of DBpedia (3.8). Due to the large number of relations in DBpedia 2.0, there is an enormous number of rules to be found. We show the time taken to mine rules with 2 atoms. We provide also

the number of predicted facts that are in the newer version of the KB but not in the old one (hits). As Table 11 shows, AMIE can produce rules with or without constants in a reasonable time.

Dataset	Entities	Facts	Runtime	Rules	Hits
YAGO2	470475	948044	3.62min	138	74K
YAGO2 const	470475	948044	17.76min	18886	159K
DBpedia	1376877	6704524	2.89min	6963	122K

Table 11: AMIE on Different Datasets

All rules and results are available at http://www.mpi-inf.mpg.de/departments/ontologies/projects/amie/.

3.8 Conclusion

In this paper, we have presented an approach for mining Horn rules on large RDF knowledge bases. We have introduced a formal model for rule mining under the Open World Assumption, a novel measure to simulate counter-examples, and a scalable algorithm for the mining. In contrast to state-of-the-art approaches, our system (AMIE) requires no input other than the KB, and does not need configurations or parameter tuning. As our extensive experiments have shown, AMIE runs on millions of facts in only a few minutes and outperforms state-of-the-art approaches not only in terms of runtime, but also in terms of the number and quality of the output rules. Our confidence measure can reasonably predict the precision of the rules. In our future work, we plan to consider also the T-Box of the KB in order to produce more precise rules. We also aim to explore the synergies when several rules predict the same fact, and extend the set of rules beyond Horn rules, so that even more complex facts and hidden knowledge can be predicted.

References

[1] Z. Abedjan, J. Lorey, and F. Naumann. “Reconciling ontologies and the web of data”. In: *CIKM*. 2012.

[2] Hilde Adé, Luc Raedt, and Maurice Bruynooghe. “Declarative bias for specific-to-general ILP systems”. In: *Machine Learning* 20 (1 1995).

[3] R. Agrawal, T. Imieliński, and A. Swami. “Mining association rules between sets of items in large databases”. In: *SIGMOD*. 1993.

[4] Rakesh Agrawal et al. “Fast discovery of association rules”. In: *Advances in knowledge discovery and data mining*. 1996.

[5] Sören Auer et al. “DBpedia: A Nucleus for a Web of Open Data”. In: *ISWC*. 2007.

[6] Yun Chi et al. “Frequent Subtree Mining - An Overview”. In: *Fundam. Inf.* 66.1-2 (2004).

[7] Philipp Cimiano, Andreas Hotho, and Steffen Staab. “Comparing Conceptual, Divisive and Agglomerative Clustering for Learning Taxonomies from Text”. In: *ECAI*. 2004.

[8] Claudia d'Amato, Volha Bryl, and Luciano Serafini. "Data-Driven Logical Reasoning". In: *URSW*. 2012.

[9] Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito. "Inductive learning for the Semantic Web: What does it buy?" In: *Semant. web* 1.1,2 (Apr. 2010).

[10] Jérôme David, Fabrice Guillet, and Henri Briand. "Association Rule Ontology Matching Approach". In: *Int. J. Semantic Web Inf. Syst.* 3.2 (2007).

[11] Luc Dehaspe and Hannu Toironen. "Discovery of relational association rules". In: *Relational Data Mining*. Springer-Verlag New York, Inc., 2000.

[12] Luc Dehaspe and Hannu Toivonen. "Discovery of frequent DATALOG patterns". In: *Data Min. Knowl. Discov.* 3.1 (Mar. 1999).

[13] Bart Goethals and Jan Van den Bussche. "Relational Association Rules: Getting WARMER". In: *Pattern Detection and Discovery*. Vol. 2447. Springer Berlin / Heidelberg, 2002.

[14] Gunnar Aastrand Grimnes, Peter Edwards, and Alun D. Preece. "Learning Meta-descriptions of the FOAF Network". In: *ISWC*. 2004.

[15] Sebastian Hellmann, Jens Lehmann, and Sören Auer. "Learning of OWL Class Descriptions on Very Large Knowledge Bases". In: *Int. J. Semantic Web Inf. Syst.* 5.2 (2009).

[16] Johannes Hoffart et al. "YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia". In: *Artificial Intelligence Journal* (2013).

[17] Joanna Jozefowska, Agnieszka Lawrynowicz, and Tomasz Lukaszewski. "The role of semantics in mining frequent patterns from knowledge bases in description logics with rules". In: *Theory Pract. Log. Program.* 10.3 (2010).

[18] Michihiro Kuramochi and George Karypis. "Frequent Subgraph Discovery". In: *ICDM*. IEEE Computer Society, 2001.

[19] Jens Lehmann. "DL-Learner: Learning Concepts in Description Logics". In: *Journal of Machine Learning Research (JMLR)* 10 (2009).

[20] Francesca A. Lisi. "Building Rules on Top of Ontologies for the Semantic Web with Inductive Logic Programming". In: *TPLP* 8.3 (2008), pp. 271–300.

[21] Alexander Maedche and Valentin Zacharias. "Clustering Ontology-Based Metadata in the Semantic Web". In: *PKDD*. 2002.

[22] T Mamer, CH Bryant, and JM McCall. "L-modified ILP evaluation functions for positive-only biological grammar learning". In: *Inductive logic programming*. LNAI 5194. Springer-Verlag, 2008.

[23] Cynthia Matuszek et al. "An introduction to the syntax and content of Cyc". In: *AAAI Spring Symposium*. 2006.

[24] Deborah L. McGuinness et al. "An Environment for Merging and Testing Large Ontologies". In: *KR*. 2000.

[25] Stephen Muggleton. "Inverse Entailment and Progol". In: *New Generation Comput.* 13.3&4 (1995).

[26] Stephen Muggleton. "Learning from Positive Data". In: *ILP*. Springer-Verlag, 1997.

[27] Ndapandula Nakashole et al. "Query-Time Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules". In: *Workshop on Very Large Data Search (VLDS) at VLDB*. 2012.

[28] Victoria Nebot and Rafael Berlanga. "Finding association rules in semantic web data". In: *Knowl.-Based Syst.* 25.1 (2012).

[29] Thomas Neumann and Gerhard Weikum. "RDF-3X: a RISC-style engine for RDF". In: *Proc. VLDB Endow.* 1.1 (Aug. 2008).

[30] Natalya Fridman Noy and Mark A. Musen. "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment". In: *AAAI/IAAI*. AAAI Press, 2000.

[31] Matthew Richardson and Pedro Domingos. "Markov logic networks". In: *Machine Learning* 62.1-2 (2006).

[32] Stefan Schoenmackers et al. "Learning first-order Horn clauses from web text". In: *EMNLP*. 2010.

[33] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. "PARIS: Probabilistic Alignment of Relations, Instances, and Schema". In: *PVLDB* 5.3 (2011).

[34] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "Yago: a core of semantic knowledge". In: *WWW*. 2007.

[35] Fabian M. Suchanek et al. "YAGO2s: Modular High-Quality Information Extraction". In: *German Database Symposium (BTW)*. 2013.

[36] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. "Selecting the right interestingness measure for association patterns". In: *KDD*. 2002.

[37] Johanna Völker and Mathias Niepert. "Statistical schema induction". In: *ESWC*. 2011.

Chapter 4

Integrating Web Services with ANGIE

4.1 Overview

The proliferation of knowledge-sharing communities and the advances in information extraction have enabled the construction of large knowledge bases using the RDF data model to represent entities and relationships. However, as the Web and its latently embedded facts evolve, a knowledge base can never be complete and up-to-date. On the other hand, a rapidly increasing suite of Web services provide access to timely and high-quality information, but this is encapsulated by the service interface. We propose to leverage the information that could be dynamically obtained from Web services in order to enrich RDF knowledge bases on the fly whenever the knowledge base does not suffice to answer a user query.

To this end, we develop a sound framework for appropriately generating queries to encapsulated Web services and efficient algorithms for query execution and result integration. The query generator composes sequences of function calls based on the available service interfaces. As Web service calls are expensive, our method aims to reduce the number of calls in order to retrieve results with sufficient recall. Our approach is fully implemented in a complete prototype system named ANGIE[1]. The user can query and browse the RDF knowledge base as if it already contained all the facts from the Web services. This data, however, is gathered and integrated on the fly, transparently to the user. We demonstrate the viability and efficiency of our approach in experiments based on real-life data provided by popular Web services. This chapter is based on

Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek,
Thomas Neumann, Wenjun 'Clement' Yuan, Gerhard Weikum
"Active Knowledge: Dynamically Enriching RDF Knowledge Bases
by Web Services (ANGIE)"
(International Conference on Management of Data (SIGMOD) 2010)

[1] ANGIE: *A*ctive K*n*owled*g*e for *I*nteractive *E*xploration

4.2 Introduction

4.2.1 Motivation

Recent projects like DBpedia [5], YAGO-NAGA [35, 23], Freebase [36], KnowItAll [6], or Intelligence-in-Wikipedia [42] have successfully created very large semantic databases with many millions of facts. The knowledge is typically represented in RDF, the W3C standard for Semantic-Web data. An RDF knowledge base can be seen as a graph, whose nodes are entities (e.g., persons, companies, movies, locations) and whose edges are relationships (e.g., *bornOnDate*, *isCEOof*, *actedIn*). Often, this graph can be visualized in a browser, so that users can explore the graph interactively. To query the knowledge base, the user (or a program on behalf of the user) can pose queries in the W3C-endorsed SPARQL[41] language which supports filters and joins in a schema-free manner.

The knowledge stored in these knowledge bases may be huge, but it can never be complete. It inevitably exhibits gaps and these may irritate the user during exploration and knowledge discovery. Consider, for example, a user who is interested in finding more information about Herta Müller, who just received the Nobel prize in literature. Knowledge bases will likely contain incomplete information. For instance, on the day she received the prize, Wikipedia contained her birthdate, birthplace, citizenship, and only a few books and awards, although she is the author of many more novels, stories, essays, and poems. Another query where the knowledge base is bound to be incomplete would be: "Which other 21st-century writers have won prestigious prizes and could be considered for the Nobel prize?".

On the other hand, there is a growing number of Web services that provide a wealth of high quality information. If these Web services could be tapped for the knowledge graph, many more user queries could be answered. For example, there are several Web services about books and authors (*ISBNdb*, *Amazon*, *AbeBooks*). Other Web services provide data about songs and music albums, movies and videos, etc. The eConsultant[2], for example, lists hundreds of public Web services, and their number is constantly growing. But these services can be accessed only through an encapsulated API; answers to queries are returned in a semi-structured format (XML) but we cannot directly access the data and we cannot observe a database schema that the service may use internally. There are tools for mapping the XML structure of service-call results into the RDF representation that we need for the knowledge graph, but there is no viable solution for automatically generating the service calls that are needed in order to answer a user's knowledge query.

A naive solution would exhaustively generate all possible service calls beforehand, and materialize all the data returned by these services and integrate them into the knowledge base. However, this is practically infeasible, due to both query-load constraints and legal reasons, and it could not guarantee the freshness of the knowledge base either. Web sites bound the number of calls coming from the same IP address, or charge each call with a fee. However, if *only* the data that is relevant for a given query could be dynamically retrieved in the current user context, a much larger number of user queries could be answered with satisfactory recall.

[2] http://webdeveloper.econsultant.com/web-services-api-services/

This is our vision of *Active Knowledge*. An active knowledge base is a dynamic federation of knowledge sources where some knowledge is maintained locally and other knowledge is dynamically retrieved from Web services and mapped into the local knowledge base on the fly. This process should be transparent to the user or application program that runs on the knowledge base, so that the user sees the data from both the local knowledge base and the external Web services as a single comprehensive RDF knowledge base. This simplifies the querying and application development against the knowledge base, and would enable a knowledge-as-a-service paradigm as recently pursued by companies like *cyc.com*, *freebase.com*, or *trueknowledge.com* (but none of these is able to tap on third-party Web services). More precisely, the task that we tackle in this chapter is, given a SPARQL query, to retrieve the necessary – and only the necessary – data from Web service via automatically generated function calls and dynamically mapping it into the knowledge base. This poses several technical challenges:

(1) Web service calls typically require input parameters of certain types. For example, certain Web services require as input the name of an author, others require the name of a book or an ISBN. The system has to call the appropriate Web services with appropriate input parameters. Even worse, the answer to the query may require the composition of multiple service calls. For example, the Web service of *MusicBrainz* requires as input the *id* of a singer (as defined by the *MusicBrainz* Web site) and returns the titles of the songs of that singer. Hence, if the user starts with the singer's name, the *id* of the singer must be obtained by a prior service call before invoking the request for the song titles.

(2) The data offered by different Web services overlaps, and there are multiple methods from the same Web service API that could fulfill the same purpose. For instance, information about a book can be obtained by a search by author name, by ISBN, by title, or by year. Therefore, different sequences of service calls can be used to retrieve the same data. Given the cost and latency of a Web service call, unnecessary calls have to be avoided by all means. This requires the calls to be planned in a way that redundancy of answers is minimized. Since a Web service API encapsulates the data and does not expose a data source schema, schema properties cannot be exploited for this purpose.

(3) Different Web services come with different quality-of-service properties like latency and coverage. We would like to satisfy the user's information need with a minimal number of calls. Hence, Web service qualities have to be taken into account, too, when computing the best sequence of Web service calls in order to satisfy a user query.

These considerations lead us to the following problem of coupling an RDF knowledge base with external Web services: *Given a SPARQL query against the knowledge base, and given a maximal number of Web service calls that can be executed, compute the largest number of answers to the query.* If infinitely many calls are allowed, compute the maximal number of answers that can be obtained by sequences of Web service calls.

4.2.2 Contributions and Outline

We have developed the system ANGIE that carries out our paradigm of active knowledge for interactive querying and exploration. Web services act as dynamically and transparently incorporated components of the knowledge base, with

seamless on-the-fly integration of service results into query answers.

We propose a Semantic-Web-oriented model for the *declarative definition of service functions*, to register and naturally embed the Web services in the local knowledge base. When a Web service is called, its results are transformed into RDF, and are dynamically added to the local knowledge base.

The salient property of our system is its reconciliation of two paradigms: data warehousing and query mediation. We see the warehousing of Web service results as an elegant solution to address the incompleteness of both knowledge bases and Web services. At every moment of the query evaluation, the local knowledge base maintains a measure of the incompleteness of the current answer. Subsequent calls can use the data returned by the previous calls to fill the gaps. No special mechanism is needed to handle the transfer of values from the output of a service call to the input of another call in the compositions of function calls. Despite these innovations, our approach does not require a new query processing engine. Rather, it naturally extends existing work on local RDF query processing, and we actually employ one of the fastest RDF/SPARQL open-source engines for our prototype [27].

This paper's key contributions are: (1) A language to represent Web service interfaces in our framework. The language extends Datalog with limited access patterns [20], by distinguishing between relationships among input parameters that must be verified as a condition to execute the call, and the relationships returned by the call. (2) An algorithm for automatically generating appropriate service calls, with awareness of call execution costs and quality-of-service properties of the Web services. The algorithm orders the service calls according to a principled cost model. (3) A prototype system and extensive experiments, using the publicly available YAGO collection [35] as a local knowledge base and additional data dynamically obtained from comprehensive, timely, and popular Web services about books and music.

We discuss related work in Section 4.3. Sections 4.4 and 4.5 describe our Semantic-Web-oriented framework for the dynamic coupling of Web services and the local knowledge base. Section 4.7 and 4.8 are dedicated to the query-evaluation algorithms of our framework. Section 4.9 describes the system architecture of our prototype, and finally, we evaluate our system in Section 4.10.

4.3 Related Work

A method of a Web Service API is a function that takes as input some parameters and returns as value a semi-structured document, usually XML. Using existent tools [16], mappings can be predefined in the system so that the XML fragments in the results of calls to the function are translated to RDF-style graphs, according to the schema of the knowledge base. Hence, we can simply see a function as a parameterized SPARQL [41] query, whose result is an RDF fragment.

4.3.1 Answering queries using views

As shown in [28], SPARQL queries can be translated to Datalog queries, hence we are essentially dealing with conjunctive queries on views. Syntactically, the definition of a function is similar to the views with limited access patterns [14,

15]. The functions have limited access patterns because in order to execute a Web service call, one must provide binding values for the input parameters. However, there is a semantic difference between the Web service APIs and the views in data integration systems: The schema of the methods in the Web service API is typically not known. Hence, one cannot apply optimization techniques e.g., [22], to eliminate redundant views. A site can define two distinct functions with the same signature. For instance, the Web site *last.fm* exports two functions that take as input an artist name and return songs sang by that artist. The first function returns the top ten songs, while the second function returns the last ten songs that were listened recently by the users of *last.fm*.

Our problem is similar to that of answering queries using views in data integration system in that we answer user queries by rewriting the initial query into a set of queries (rewritings) that are executed on the remote data sources. However, in our setting the query evaluation is bounded by the maximal number of calls. This changes the approach to solve the problem, as we show next. Depending on the setting (query optimization or data integration) there are two distinct query rewriting problems. In the first case, the views are complete, the number of the views is limited, and the result is an equivalent rewriting. In the second case, the views are incomplete, the algorithm must scale up to a large number of views, and the result is a maximal contained query. Because Web services are incomplete we shall compare only to the second problem. We use as basis of comparison the language of the rewritings.

Conjunctive queries. There exist a number of standard algorithms, most notably BUCKET [26] and MINICON [29]. These algorithms, however, do not consider services whose only role is to provide values for the parameters of a subsequent service, even though this might be the only way to use this second service. Furthermore, they do not support recursive query plans. But, as shown in [25], even non-recursive queries require recursive query plans if the views have limited access patterns.

Conjunctive queries with binding patterns. In [14, 15] (with improvements in [22, 18]), the authors show that for every query there is a finite rewriting using the views, albeit a recursive one. This algorithm proceeds by rewriting a so-called Local As View system into a so-called Global As View system, by inverting each Datalog rule and introducing Skolem terms. The algorithm first computes the consequences of the rules (i.e. it instantiates all possible function calls); then it uses a bottom-up evaluation to compute the answers. The central weak point of this approach for our scenario is the bottom-up computation. It is simply impossible to enumerate all Web service results, because most Web services restrict the number of calls coming from an IP address, so that the algorithm would be stopped before the first step is completed. Our approach, in contrast, searches to minimize the number of calls after which execution answers are output.

The authors of [32, 43] provide a different approach to query rewriting, targeting equivalence of rewritings. This approach, however, assumes that the views are complete.

4.3.2 Knowledge Representation Formalisms

XML languages. Since XML is the de facto standard for Web services, one could think of choosing XML as the data model for the global schema. Still, this does

not eliminate the necessity of mapping the schema of the Web service to the schema of the knowledge base. With XML as with RDF, the result of a Web service call must be re-structured according to the schema of the knowledge base. Furthermore, it is infeasible to store the semantic graph in XML documents, and to query it using XPath. The query engine would spend most of the time chasing XLink links. Our approach, in contrast, uses the RDF-3X [27] engine, which is a native RDF query engine.

Object oriented. Early works in data integration [19] used an object-oriented language for the rewritings. For the same reasons as above, these approaches are less adequate for the Semantic-Web-oriented (RDF-style) knowledge bases we consider.

Description Logics. The standard reasoning formalism for RDF data is OWL, with its flavors OWL Full, OWL DL and OWL lite. In [9], it is shown that the Description Logic language *DL-Lite* can be embedded in an extension of the Datalog language. The major extension consists in allowing existentially qualified variables in the heads of the rules. The problem of reasoning on the knowledge base, however, is orthogonal to our problem.

Sources with querying capabilities. Recent works have investigated sources that have querying capabilities by themselves [38, 10]. Our work, in contrast, focuses on Web services, which have no such capabilities.

4.3.3 Other styles of data integration

ActiveXML. The concept of embedded Web services was introduced by the ActiveXML paradigm [1]. But the ActiveXML framework has no capabilities to compute all the Web call compositions that answer the query. It does not address the problem of answering querying using views.

Mashup Systems. In contrast to the mash-up approaches [34, 21], our system acts like a mediator system, where the query dynamically combines data from local and external sources, on demand.

Linking Open Data Project. The Linking Open Data Project aims to link Semantic Web resources into a global and distributed graph that spans several Web sites [8]. The resources are linked using URIs and RDF specifications. However, the integration of dynamic components such as Web services has not been considered yet.

4.3.4 Complementary problems

The Deep Web. The Deep Web is the part of the Web that is accessible only by Web forms and Web service APIs. There is much work on the automatic construction of wrappers for Web forms (e.g., [11, 17]). This work is complementary to ours, because it is not concerned with the exploitation of Web services. On the contrary, the work can be used to construct Web services from Web forms [33]. These Web services can then provide the input to our system.

Schema mappings and Data Fusion on the Fly. In the present work, we assume that the mapping between the schemas of the Web Services and the schema of the knowledge base is given. The (semi) automatic creation of schema mappings has been addressed in a large corpus of works, e.g., [16, 3, 24]. Furthermore, we are not concerned with entity disambiguation and data fusion in this paper. Data fusion is an important component of our system, but it has

been vividly addressed in previous work (e.g., [4]). In this work, we develop at a clean model for query rewriting, and we see the disambiguation and data fusion algorithms as an orthogonal problem.

4.3.5 Other Web Services applications

Web service composition. A number of works address the problem of automatic composition (or orchestration) of the Web services carrying out complex interactions between Web applications [7, 12]. Our work, in contrast, is concerned with answering queries using Web services wrapping parameterized queries.

Another problem is to determine the composition of Web services that can answer a parameterized user query [37], or return objects of a given type [31]. In our model, the user queries are not parameterized. Furthermore, in Section 4.8, we show how to use the constants in the query, as well as the relationships from the knowledge base where the constants appear to reduce the number of Web calls.

4.4 Data model

4.4.1 RDFS and Semantic Graphs

In tune with recent work [35, 5, 23], we represent our knowledge base in the RDFS standard [39]. In RDFS, knowledge is modeled as a *semantic graph*. A semantic graph is a directed labeled graph, in which the nodes are entities (such as individuals, classes, and literals) and the labeled edges represent relationships between the entities. A fragment of a sample semantic graph is shown in Figure 1. Formally, a semantic graph can be defined as follows.

Definition 1 (Semantic Graph) *Let Rel be a set of relation names and let Ent* $\supseteq$ *Rel be a set of entities. A semantic graph over Rel and Ent is a set of edges* $G \subset Ent \times Rel \times Ent$.

Thus, a semantic graph is seen as a set of triples. This allows two entities to be connected by two different relationships (e.g., two people can be colleagues and friends at the same time). A triple of a semantic graph is called a *statement*.

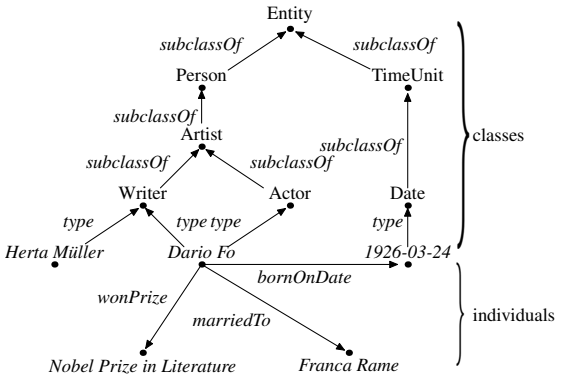

Figure 1: A semantic graph

In RDFS, there is a distinction between individual entities (such as Dario Fo) and class entities (such as the class *Actor*). Individuals are linked by the *type* relationship to their class. For example, Dario Fo is linked to the class *Actor* by an edge *(Dario Fo, type, Actor)*. The classes themselves form a hierarchy. More general classes (such as *Artist*) include more specific classes (such as *Actor*). This hierarchy is expressed in the semantic graph by edges with the *subclassOf* relationship, e.g. *(Actor, subclassOf, Artist)*.

4.4.2 Query Language

As query language, we consider a subset of the standard RDFS query language SPARQL [41].

Definition 2 (Query) *A query over a set of variables Var for a semantic graph $G \subset Ent \times Rel \times Ent$ is a semantic connected graph $Q \subset (Ent \cup Var) \times (Rel \cup Var) \times (Ent \cup Var)$.*

Figure 2 shows two example queries. The first one asks for the prizes won by Dario Fo. The name of the prize is represented by a variable. The second query asks for Dario Fo's relationship to Franca Rame. In this case, the relationship itself is a variable.

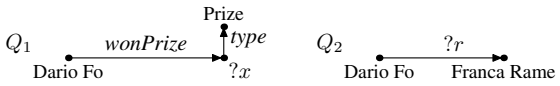

Figure 2: Two sample queries

Definition 3 (Query Answer) *An answer to a query Q on a semantic graph G is a graph homomorphism $\sigma : Q \to \sigma_Q \subseteq G$ that preserves the entity and relation names, and substitutes the variables in Q with entities and relationships names from G.*

For instance, consider again the semantic graph shown in Figure 1. The query Q_2 has an answer in the semantic graph because there is the substitution $\sigma(?r)$=*marriedTo*. $\sigma(Q_2)$=*(Dario Fo, marriedTo, Franca Rame)* is a sub-graph of the semantic graph.

4.4.3 Functions

In our model, the user can query for data that is not yet in the knowledge base. This knowledge is retrieved on the fly by calling Web services. We consider a Web service method as being a function. We see a function as a parameterized query.

Definition 4 (Parameterized Query) *A parameterized query is a query $Q \subset (Ent \cup Var) \times (Rel \cup Var) \times (Ent \cup Var)$, where the set of variables is partitioned in two sets: input variables, denoted with I, and output variables, denoted with O.*

The variables in I must be bound to their actual values before the query is executed. Thus, at execution time, the parameterized query becomes a query

as defined above. The answer of the query associates binding values for the variables in O.

While the local knowledge base is given extensionally, the knowledge provided by the functions is given intensionally. That is, the function definitions do not provide the data itself, but they define a way to obtain it. Conceptually, the extensional data and the intensional data form one large knowledge base. The user can browse this knowledge base transparently, without noticing the difference between intensional and extensional knowledge.

Definition 5 (Function Definition) *A function definition is a parameterized query* $f \subset (Ent \cup I \cup O) \times (Rel \cup I \cup O) \times (Ent \cup I \cup O)$ *where the set of edges is partitioned in input edges (input conditions) and output edges so that each variable occurring in an input edge is an input variable.*

As a condition to execute the function call, the inputs edge conditions must be verified in the local knowledge. The output edges are instantiated once the result of the function is retrieved.

Consider a function that returns for a given writer the books that he/she wrote. The valid inputs are person names. Consequently, a input edge would be the edge *(?x, type, Person)*, and as an output conditions the edges *(?x, type, Writer)*, *(?x, wrote, ?y)*, where $?y$ is instantiated by the function call.

In practice, Web services may define also optional input parameters. The optional parameters can be seen as variables having a dual state. They can be either input or output variables. Because a call must provide bindings for at least one input variable, they do not change the problem. With some technical details, the algorithms can handle them. For clarity, we ignore optional parameters here.

Graph representation. In our model, the function definitions themselves are part of the local knowledge base. Every function definition is identified by an entity representing the function. The edges of the function definition are reified. In this process, variable names become nodes in the semantic graph. Each reified edge is connected to the function identifier by an edge representing the *inputEdgeOf* or *outputEdgeOf* relationship. Figure 3 shows an example. The function *getBooks* has the input edge ($?x$, *type*, *Writer*). It has as output edge ($?x$, *wrote*, $?y$).

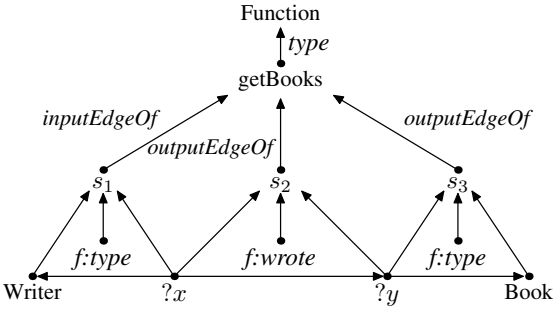

Figure 3: A function definition

This way, the function definitions are integrated completely into the semantic graph. Thereby, function definitions become first class citizens of the knowledge

base. Thus, it is possible to query the knowledge base for functions that have certain properties.

Comparison with Datalog. Datalog queries [2] are conjunctive queries of the form:

$$q(\bar{X}) \leftarrow r_1(\bar{X_1}), r_2(\bar{X_2}), \ldots r_n(\bar{X_n})$$

where q and $r_1, r_2, \ldots r_n$ are predicate names. The predicate names refer to database relations. The tuples $\bar{X_1}, \bar{X_2}, \ldots \bar{X_n}$ contain either variables or constants. The query must be *safe*, i.e., $\bar{X} \subseteq \bar{X_1}, \bar{X_2}, \ldots \bar{X_n}$ (every variable in the head must also appear in the body).

Furthermore, in order to model the input and the output parameters, adornments attached to queries have been introduced in [32]. If the head of the query has n attributes, then an adornment consists of a string of length n composed of the letters b and f. The meaning of b is that a binding value *must* be provided for the variable in that position. For example, the function in Figure 3, can be written in Datalog syntax as follows:

$$getBooks(?x,?y)^{bf} \leftarrow wrote(?x,?y),\ Writer(?x),\ Book(?y)$$

where the adornment bf says that $?x$ must be bound (input variable) and $?y$ is free (output variable).

There are two semantic differences between our function definition and the views with limited access pattern [14]. First, the function definition is not a source description, but a description of the result returned by a call to the function. The methods in Web service APIs published by some Web site are not defined with respect to a schema of the source. Hence, one cannot apply optimization techniques e.g., [22], to eliminate redundant views. Second, we make the distinction between input and output edges. The role of input edges is to restrict the set of values that are valid inputs for function calls. Previous works ignored that Web sites protect the access to their Web services (or forms) against too many requests coming from the same IP.

Intensional Rules. In Datalog, predicates can be defined extensionally (by declaring instantiated atoms) or intensionally (by declaring a rule). This principle carries over naturally to our data model. The role of extensionally defined predicates in Datalog is taken over by concrete edges in the semantic graph. The role of intensionally defined predicates is taken over by *intensional rules*. An intensional rule is given by an ordinary function definition that is not connected to any Web service. Furthermore, all variables that occur in output edges must occur in input edges. If such an intensional rule is called, the output edges are added to the knowledge graph without reference to any Web service. Since the input edges act as conjunctive conditions, intensional rules have the same expressive power as domain-restricted Horn clauses with binary predicates in Datalog.

4.5 Query Evaluation

Given a user query, the problem is to evaluate it using the local knowledge base and the set of functions defined in the system. As described in Section 4.4.3,

we treat the functions as intensional (i.e. active) components of the knowledge base and represent them within the knowledge base. The vision is to unfold (i.e. materialize) intelligently the intensional parts of the knowledge base so that the local knowledge base be able to answer to the user query.

Given a SPARQL query (see Section 4.4.2), one should compute the sequence of calls whose results, when added to the knowledge base, can answer the query. Note that the functions have input parameters. Hence, for a function, the number of corresponding calls is equal to the cardinality of the function domain. A brute force solution that enumerates and executes all possible function calls is not feasible due to the large volumes of data generated by all the calls.

The problem is challenging because, in order to obtain the desired values for the output of a function, one should not only provide valid input values, but those judicious input values that return the desired result. Furthermore, some input values may only be obtained as the results of other calls. For instance, in Figure 5, the function f_1 can be called to retrieve the books written by *Herta Müller* only if her *id* is provided as input. Input values for the f_1 can be directly obtained from the database, or can be obtained using the function call f_2*(Herta Müller)*.

4.5.1 Existing techniques

Algorithms such as BUCKET [26] or MINICON [29] that were developed for data integration systems cannot be directly applied to our framework due to the presence of limited access patterns. These algorithms do not consider functions whose only role is to provide values for a subsequent Web call. The algorithm for views with limited access patterns presented in [14] is closest to our framework. It is guaranteed to produce the maximal contained rewritings. The focus is on computing the maximal number of answers and not how to compute the first answers fast, with a limited number of calls. The key idea in [14] is to construct a set of inverse rules for each view. An inverse rule shows how to construct tuples for the database relations from the result of a view. For instance, consider a function that given an author name and a conference, returns the papers published by the author in that conference:

$$getPapers^{bfb}(?a,?p,?c) \leftarrow authorOf(?a,?p),\ publishedIn(?p,\ ?c)$$

For the relations in the body of the rule above, the inverse rules are:

$$\begin{aligned} authorOf(?a,?p) \quad &\leftarrow getPapers^{bfb}(?a,?p,?c) \\ publishedIn(?p,\ ?c) &\leftarrow getPapers^{bfb}(?a,?p,?c) \end{aligned}$$

A special rule *dom* (domain) is also added. Its role is to generate the domain from which the output variables of functions may take values. This set is extended with the constants in the query and in the database base. A Datalog query is evaluated on the newly obtained Datalog program using the bottom-up technique [2]. We cannot test such technique due to the limitations on the number of Web calls that one IP address can invoke. One can think that the top-down technique [2] can be used instead, because of its properties of pushing the selections. For example, let us consider the query:

$$q(?p) \leftarrow authorOf('Alice',?p),\ publishedIn(?p,\ 'SIGMOD')$$

Then, the term *authorOf('Alice',?p)* can be expanded in an SLD (Selective Linear Definite) derivation tree that uses the above defined inverse rules:

$$\begin{array}{c} \mathit{authorOf('Alice',?p)}, \quad \mathit{publishedIn(?p,\ 'SIGMOD')} \\ | \\ \mathit{dom(?c)},\ \mathit{getPapers}^{bfb}\mathit{('Alice',\ ?p,\ ?c)} \end{array}$$

Note that $?c$ must be bound in order to execute the call. As suggested in [14], the literal dom($?c$) is introduced in order to be used to bind values for $?c$, in a left to right evaluation. Hence, this technique is also limited in its ability to push selections to views with limited access patterns.

Our next algorithms can match at once a subset of edges in a query with a subset of edges in a function. For instance, in our example we can detect that the two edges of the query can be obtained in the answer of the function *getPapers*, and we can directly bind $?a$ to *Alice* and $?c$ to *SIGMOD*. We can see this as an extension of the inverse rules. The extension consists in allowing for multiples literals in the heads of the inverse rules.

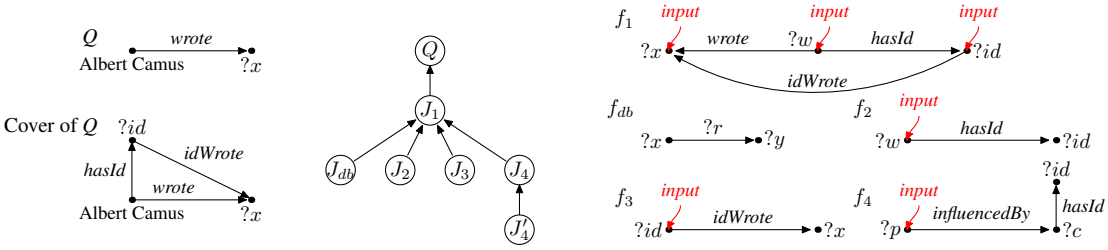

Figure 4: Query and cover (left), derivation tree (center) and function definitions (right)

4.6 Preliminaries

We model function calls by help of *partial instantiations*.

Definition 6 (Partial Instantiation) *A* partial instantiation *for a function* $f \subset (Ent \cup I \cup O) \times (Rel \cup I \cup O) \times (Ent \cup I \cup O)$ *is a graph homomorphism*

$$\sigma : f \to J \subset (Ent \cup Var) \times (Rel \cup Var) \times (Ent \cup Var)$$

that preserves the names of entities and relationships, and maps variables either to entity and relationship names or to new variables ($\sigma : I \to Ent \cup Var$ *and* $\sigma : O \to Ent \cup Var$).

The partial instantiation will instantiate some variables of the function with entity or relation names. Other variables of the function definition are simply given new variables names.

Now, a *function call* is simply a partial instantiation that binds all input variables:

Definition 7 (Function Call) *A* function call *for a function definition* $f \subset (Ent \cup I \cup O) \times (Rel \cup I \cup O) \times (Ent \cup I \cup O)$ *is a partial instantiation that maps variables in* I *to entity and relationship names so that the input edges form a sub-graph of the local semantic graph, and maps the variables in* O *to new variables in a new variable set* Var.

$$\sigma : f \rightarrow W \subset (Ent \cup Var) \times (Rel \cup Var) \times (Ent \cup Var)$$

The execution of the function call will instantiate variables in *Var*. The result of a function call W is a new graph $R \subseteq Ent \times Rel \times Ent$ that is homomorphic with W. Now we are ready to define the *evaluation of a query*:

Definition 8 (Evaluation) *An* evaluation *for a query* Q*, with a set* F *of function definitions for Web services, and a semantic knowledge graph* $G \supseteq F$*, is a list of function calls* $W_1, W_2, \ldots, W_n$*, with the corresponding results* $R_1, R_2, \ldots R_n$*, so that*

$$Q((G \setminus F) \cup R_1 \cup R_2 \ldots \cup R_n) \subseteq Q(G)$$

where $Q(G)$ *is the answer of* Q *for the knowledge base* G.

If the input value in W_j comes from the result of some call W_i , then we write $W_i \prec W_j$, and we say that W_i and W_j are executed in pipeline. The construction of evaluation expressions relies on an intermediary structure that we define in the following. We first note that, for every edge of the knowledge graph, we can construct a trivial function that has this edge as output edge. Let F_{db} denote this set of functions.

Definition 9 (Query cover) *A* cover *for a query* Q *on an instance of our data model, is a set of function instantiations* $J_1, J_2, \ldots, J_n$ *for the functions* $f_1, f_2, \ldots, f_n \in F \cup F_{db}$*, (on a common graph* $Q \cup J_1 \cup J_2 \cup \ldots J_n$*), so that:*

(*i*) *For each input attribute of some* J_i*, there is an instantiation* J_j *where the attribute is output.*

(*ii*) *For each edge (triple) of* Q*, there is an instantiation* J_j *where the edge is output edge.*

For two partial instantiations J_i and J_j that have the property (i) above, we denote $J_j \prec J_i$. Note that two Web calls $W_j \prec W_i$ correspond to two instantiation J_i respective J_j so that $J_j \prec J_i$. One can see the partial instantiation as nodes in a directed graph structure where the edges are $\prec$ relationships.

Example 1 *Consider the Figure 5. More RDF triples matching the edge (?w, wrote,?x) can be obtained as the result of calls to the function* f_1*. A partial instantiation* J_{f_1} *for* f_1 *is then defined so that* $\sigma_1(?id_1) = ?id$ *and* $\sigma_1(?w_1) = ?w$*. The graph in the right side represents the query cover. Now,* f_2 *can provide triples matching the edge (?w, hasId, ?id). We add to the query cover* J_{f_2}*, a partial instantiation for* f_2*, so that* $\sigma_2(?id_2) = ?id$ *and* $\sigma_2(?w_2) = ?w$*. Note that ?id is output attribute in* J_{f_2} *and input attribute in* J_{f_1}*. Hence* $J_{f_2} \prec J_{f_1} \prec Q$*. Furthermore, let* f_3 *be a function that can provide relationships that match the edge (?w wonPrize ?p). In summary, we have three partial instantiations in the query cover.*

J_{f_1} = {(?w, wrote, ?x), (?w, hasId, ?id)}
J_{f_2} = {(?w, hasId, ?id)}
J_{f_3} = {(?w, wonPrize, "Nobel prize in literature")}

Evaluation for infinite number of Web calls. In [25], the authors show that for views with limited access patterns there might be *no* bound on the size of the rewriting. As an example, consider the following function:

$$f^{bf}(?x,?y) \leftarrow Artist(?x),\ Artist(?y),\ Collaborated(?x,?y)$$

and a query requesting all the entities of type *Artist*. For each artist in the knowledge base and, for each retrieved artist, a lot more artists could be discovered by calling the function f. Note that the chain of recursive calls of f is not bound by the size of the query. In general, one can envision Web services which implement functions that generate infinite domains. For instance, consider a function that given a year, returns the next year in which a total sun eclipse will occur.

4.7 Depth-first algorithm

This algorithm relies on a depth-first-search strategy to expand the query cover and on a left to right evaluation. Algorithm 3 computes and outputs the rewritings for a query in the spirit of the chronological backtracking strategy used in Prolog.

Algorithm 3 DF(Funct F, Cover C, Stack St, Query Q)

```
1: J ← St.pop();
2: if (J.depth = MAX || St.empty) then return C;
3: end if
4: for all (f ∈ {f_db} ∪ F) do
5:     for all (J_i so that (J_i = σ(f)) ∧ (J_i ≺ J) ∧ (J_i ∉ C)) do
6:         J.addChild(J_i);
7:         DF(C ∪ J_i, St.push(J_i));
8:         executeWebCallsFor(J_i)
9:     end for
10: end for
```

The algorithm implements a recursive depth-first search for constructing the derivation tree. The query cover is constructed by recursive calls to the function DF. The function takes as input the set of functions F, the current configuration of the query cover C, a stack St with the instantiations J for which the derivation tree should be computed. The function f_{db} is a local function that is added for uniformity, so that both the extensional predicates and the intensional predicates are accessed using functions. In the beginning, both the stack St and the query cover consist of the query Q. At each recursion step, one partial instantiation J is removed from St and *new* partial instantiations J_i so that $J_i \prec J$ are added to C and to the stack St.

Consider for instance the query Q in Figure 4, and the function definitions defined on the right-hand side of the figure. Q asks for the books written by Albert Camus, f_1 is a function-composition rule equivalent to the Horn clause:

wrote(?w,?x) ← *hasId(?w,?id), idWrote(?id,?x)*

function f_2 maps an artist to its *id* in the *LibraryThing* database, f_3 maps the *id* to the books written by the author and f_4 returns the writers influenced by a writer (in this example Albert Camus) and the *id*s of the influenced writers.

The algorithm searches a substitution that satisfies at least an edge of Q. In our case, we can have a substitution σ for f_1 i.e. $\sigma(?x)$=*Albert Camus* and $\sigma(?w)$=*?x*. Hence, the partial instantiation $J_1 = \sigma(f_1)$ is appended to the query cover, and J_1 is pushed into the stack. In the derivation tree we have $J_1 \prec Q$. In the next steps, the algorithm removes J_1 from the stack and unfolds the derivation tree for it. It tries to find the functions that satisfy the edges marked with *hasId* and *idWrote*. The algorithm always tries to use the local database function to early bind input parameters. In our example, the algorithm chooses f_{db} to satisfy the edge marked with *hasId*. The *id* of *Albert Camus* is extracted from the database. Note that another possible choice is f_2. In the next step (if at least one of the function returned an *id* of *Albert Camus*), the algorithm chooses f_3 to satisfy the edge marked with *idWrote*. The result is a rewriting that "covers" the initial query completely. The tree in Figure 4 denotes the derivation tree that is constructed by the *DF* algorithm. J_1, J_2, and J_3 denote the partial instantiations for f_1, f_2, and f_3, respectively.

Note that the presented algorithm relies excessively on a depth-first search strategy. In case there is an infinite rewriting of the input query, the algorithm will descend into a non-terminating recursion. For instance, note that the function f_4 returns also relationships for the relation *hasId*. Furthermore, the function can be called recursively, so that more *hasId* are returned. For the completeness of the solution, one should take into account these relationships. In order to prevent infinite loops, we bound the depth of the derivation tree by *MAX*. Our next algorithm exploits an intuitive cost model for the early binding input parameters in function instantiations, to prioritize the evaluation of partial instantiations.

4.8 F-RDF algorithm

The Depth-First algorithm may descend into one branch of the derivation tree and it may use the total number of allowed function calls without even producing a single answer. Our next algorithm improves on the depth first algorithm in that it first unfolds lazily the query cover without executing the Web calls, then, it partially orders the values used as inputs for calls. It prioritizes those values from the local knowledge base that are in the results of local queries consisting of some of the predicates and nodes in the query. Hence, instead of pushing the constants in a recursive chain of nested calls, without any target, the new approach tries to use as bindings the values that are in relationship with possible query answers. Furthermore, the algorithm estimates the cost of executing a pipeline of Web calls that might contribute with relationships to the query answer.

For a function f, one can extract from the database the values for its input bindings, and execute the set of function calls. Note that the sequence of calls might be unbound, because the response of a call can contain new binding values for the inputs for f. We call the list of all function calls that can be defined for f,

the exhaustive list of calls for f. This algorithm avoids producing the exhaustive list of calls from the beginning, and instead, searches first for a better strategy. The key is to choose first those bindings that verify some of the conditions in the query.

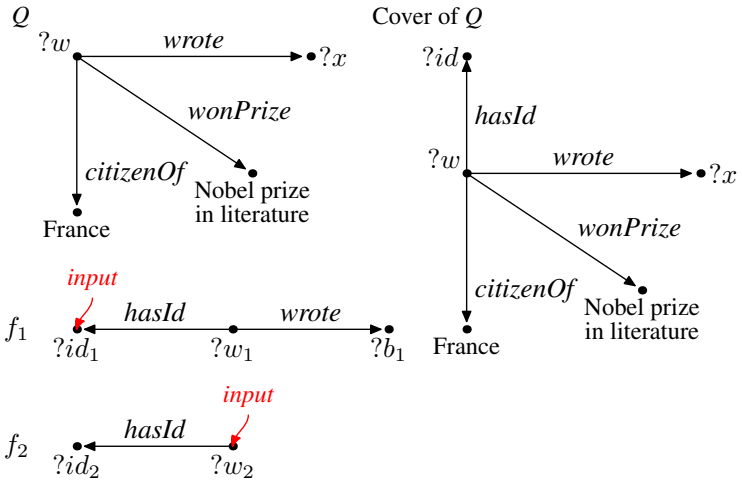

Figure 5: Query Q and function definitions f_1 and f_2 (left) and query cover $Q \cup J_{f_1} \cup J_{f_2}$ (right)

4.8.1 Lazy construction of the query cover

To simplify, but without losing the generality of the solution, we ignore in the following the edges labeled with *type* in the query cover $Q \cup J_1 \ldots \cup J_n$. As a pre-filtering step, we check for every node of the function instantiation if its type is compatible with the type of the matched node in the cover. For each partial instantiation J in the cover, we construct local queries that extract bindings for the input attributes. The queries are constructed as connected sub-graphs of the query cover. The local queries contain at least a node that is labeled with a constant in the initial query. We denote the local queries using the term *binding queries*.

As example, consider the query in Figure 5. Note that unlike the example in Figure 4, the name of the writer $?w$ must to be computed. Those values that satisfy already some constraints in Q are better candidates to be part of the final solution. Several sub-queries of Q can be considered. For instance, one can select values for $?w$ so that:

$$\{\ (?w,\ citizenOf,\ 'France'),\ (?w,\ wonPrize,\ 'Nobel\ prize\ in\ literature')\}$$

Let B be a binding query for a partial instantiation J. For each binding query B, we can write it as the conjunction of two queries $B = Q_B \cup P_B$, where $Q_B \subseteq Q$. P_B is a path (or union of paths) from an input attribute in J to a node in Q. We construct the binding queries, incrementally, as follows.

Case (1) If $J \cap Q \neq \emptyset$. Since the query cover is connected (union of connected graphs), then for each input attribute in J, there is a path to nodes in Q.

Case (2) If $J \cap Q = \emptyset$. In this case, there is a sequence

$$J = J_i \prec J_{i-1} \ldots \prec J_1 \prec Q$$

in the derivation graph. Note that $J_k \cap J_{k-1} \neq \emptyset$ for all $k \in \{1, \ldots n\}$, and $J_1 \cap Q \neq \emptyset$. Let B_i be a binding query for J_i. Then, one can compute a binding pattern B_i for J_i as the conjunction between B_{i-1} (of J_{i-1}) and a path (or union of paths) in $J_i \cup J_{i-1}$. A path has the origin in an input attribute of J_i and reaches an input attribute in J_{i-1} that occurs also in B_{i-1}.

Example 2 *Consider the query in Figure 6. The query asks for the titles that the* Queen Victoria *has had. The function h_1 can be used to obtain the titles. Note that h_1 takes as input the house (dynasty) and the name of the person. Let J_1 be the partial instantiation for h_1 so that $\sigma_1(?p)$ = "Queen Victoria". Assume that the house information is missing from the local database. However, the function h_2 can provide it. Let J_2 be the partial instantiation for h_2 so that $\sigma_2(?p)$ = "Queen Victoria". Assume that the local base contains the names of some of her children, but not their house attribute. Hence, the binding query B_2 =($?c_1$, hasParent, "Queen Victoria") has answers. Therefore, the lazy construction continues to append another instantiation for h_2, as showed in Figure 6. Let J_3 where $\sigma_3(?p)$ =$?c_1$, be the new instantiation. The recursive construction can continue until a descendant for whom the house attribute is stored in the local base is found (e.g. "Elisabeth II" of house Windsor), or no new generations are reached. Once a binding for all inputs is found, the evaluation proceeds bottom-up.*

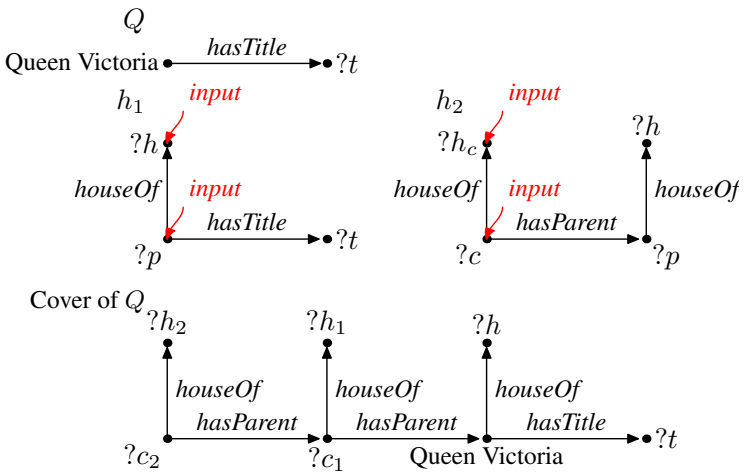

Figure 6: Lazy query construction

4.8.2 Quality of Web service calls

The operation that has the largest impact on the total execution time is the execution of Web calls. The algorithm should prioritize the execution of the calls in Web call composition that lead to a solution with a smaller cost. Consider an instantiation J_i so that $J_i \prec \ldots \prec J_1 \prec Q$. Based on the *quality* of the functions $f_i, \ldots, f_1$ that correspond to the instantiations, we can estimate the quality of the instantiation J_i.

Definition 10 (Quality of Service) *The* quality of service *for a function f is a triple $(p, t, a\vec{v}g)$, where p is the probability for which the function f returns a*

non-empty answer, t is the average response time, $\vec{avg}$ is a vector that contains for each edge, the average number of corresponding triples in the results of the calls.

We define the *quality of a partial instantiation* J_i as a hierarchy of parameters:

$$QoJ(J_i) = (\sum_{k\in\{i,\ldots 1\}} t_k, \prod_{k\in\{i,\ldots 1\}} p_k, \prod_{k\in\{i,\ldots 1\}} p_k \cdot avg_k)$$

Consider a Web call W_i of J_i. The first component of the *QoJ(J_i)* denotes the probability for which a complete list of calls $W_i \prec \ldots \prec W_1$ is computed if the bindings for the Web call W_{k-1} are chosen arbitrarily from the result of W_k. The second component is the time necessary to execute the Web calls $W_i \prec \ldots \prec W_1$. The third component is the estimation for the total number of Web calls corresponding to $J_{i-1} \prec \ldots \prec J_1$, where avg_k is a scalar that denotes the average number of Web calls W_{k-1} that use as inputs, values that are output by W_k.

4.8.3 Algorithm

The *F-RDF* algorithm uses the lazy construction strategy described in Section 4.8.1 to construct the query cover. The algorithm checks for early bindings for the input attributes of the instantiations. For each partial instantiation, we keep the list of query bindings, and of their results. More precisely, for a partial instantiation J we keep the list of tuples (B, Q_B, R_B, $\mathcal{W}_B$) where R_B denotes the list of answer of B, and $\mathcal{W}_B$ represents the list of Web calls whose inputs are obtained in the result of B. The list of tuples is sorted according to the subquery $Q_B \subseteq B$, from the most selective Q_B (with the largest number of edges) to the least selective one. The Algorithm 4 stores the partial instantiations in a priority queue denoted with L. The instantiations are partially sorted according to the next rule, and secondly, according to the quality of the instantiation *QoJ*.

Rule 1 (Precedence of Web calls) *If $J_j \prec J_i$ and B_j and B_i are two binding queries corresponding to J_j and J_i, respectively so that $Q_{B_j} \supseteq Q_{B_i}$, then all the calls in $\mathcal{W}_{B_i}$ are executed before the calls in $\mathcal{W}_{B_j}$*

The rule makes sure that the information present in the local database is used to obtain fast the first results. Note that a Web call of J_j is pipelined with a Web call of J_i in order to be used in an answer of the query. As a consequence of the rule, we have the following property:

Property 1 *For a pipeline of calls*

$$W_1 \prec \ldots W_{i-1} \prec W_i \ldots \prec W_n$$

so that their input values are the results of $B_1, \ldots, B_n$, where $Q_{B_1} \supseteq \ldots \supseteq Q_{B_n}$, if the input values of W_i are already in the knowledge base, then the calls W_k for $k \geq i$ are executed first.

The binding input of W_i can be present in the knowledge base by other means than the result of W_{i-1}. Usually, the same metadata is replicated on multiple Web sites. As observed in [13], many of the Web databases copy from one another. This property is important because a new result of the query is produced with only calling at most $(n - i)$ Web calls in the pipeline.

Algorithm 4 *F-RDF*(Funct F, Query Q, int MAX_CALLS)

```
1: while L.notEmpty() && calls ≤MAX_CALLS do
2:     J(B, Q_B, R_B, W_B) = L.getFirst();
3:     for all (W ∈ W_B) do
4:         W.execute();
5:     end for
6:     for all (J_i(B_i, _, _, _) so that B_i ⊆ B ∪ J) do
7:         update R_{B_i};
8:     end for
9:     LazyCoverExtension();
10: end while
```

4.8.4 Properties

Our algorithm guarantees that it produces first the composition of function calls that take as input the constants in the query, and whose answers added to the knowledge base produce new answers for the query. Only after this sequence is produced, it considers the exhaustive lists of calls.

Property 2 *If there is a sequence of Web calls*

$$W_1 \prec W_2 \prec \ldots \prec W_n$$

so that the inputs of each W_i are in the result of a binding query B_i (possibly as the result of previous calls), then the sequence of calls is output by the algorithm before the exhaustive list of calls.

Proof Sketch: For simplicity sake, assume that all functions have exact one input. We prove that a list of partial instantiations $J_1 \prec J_2 \prec \ldots \prec J_n$, corresponding to the Web calls is added to the cover. J_1 is added because B_1 has no empty results. When the results of W_1 is added to the knowledge base, then the query $B_1 \cup J_1$ has answers. Then, there is a binding query $B_2 = B_1 \cup J_2$ whose answers contain the input value for W_2. According to the principle used to construct the cover, J_2 is added to the cover, and W_2 will be then executed. The proof is then by induction.

In practice, such sequence of calls exists. Web sources define functions for both direct and inverse relationships between input and output. Hence, Web sites allow for navigation in the remote graph of resources following both senses of a relationship (edge).

In the following, we show that determining finite rewritings to a query is PSPACE-complete.

Theorem 1 *Let Q be a query expressed by our query language (as defined in Section 4.4.2), and let F be the set of function definitions. Determining a finite query cover of Q with functions from F is PSPACE-complete in the size of G.*

Proof Sketch: For the lower bound, our proof is by reduction from the General Geography (GG) problem, which is known to be PSPACE-hard. As for the upper bound, we give an algorithm that computes a finite rewriting to the query in PSPACE. For a given GG graph with a designated node Q, we run a breadth-first search from Q and construct for each edge (u, v) that we encounter

at an odd level (assuming that the outgoing edges of Q are at level 1), a new function with output variable v and input variable u. Analogously, for each edge (v, z) that we encounter at an even level, we introduce a new function with input variable z and output variable v. The node Q represents our query. The intuition behind this reduction is that Player II can choose any possible output, and Player I has to respond with the appropriate input. It can be shown that there is a winning strategy for Player I if and only if there exists a cover for the query.

Theorem 2 *The* F-RDF *produces the complete list of Web calls for an evaluation with an unbounded number of calls.*

Proof Sketch: The proof is based on the following observation: for any partial instantiation J_j, *F-RDF* will expand all instantiations J_i for which $J_i \prec J_j$. We call this property the exhaustive neighborhood search (ENS) property. The proof is by contradiction. Assume that there is a query cover $J_1, ..., J_k$ that cannot be found by *F-RDF*. Then there must be some partial instantiation $J_i, 1 \leqslant i \leqslant k$ with $J_i \prec J_j$ for some $j, 1 \leqslant j \leqslant k$ that could not be discovered by the *F-RDF*. By the ENS property follows that J_j could not have been discovered by the *F-RDF* either. We can continue this argumentation recursively until we reach Q, which could not have been discovered either by the *F-RDF*. This is by the construction of *F-RDF* a contradiction.

4.9 System architecture

The overall architecture of our system is illustrated in Figure 7. The system uses the existing YAGO ontology [35], which consists of 2 million entities and 20 million facts extracted from various encyclopedic Web sources. In addition, we extended the knowledge with a built-in collection of function definitions for the following Web services: *MusicBrainz*, *LastFM*, *LibraryThing*, *ISBNdb*, *AbeBooks*, and *IVA* (Internet Video Archive). In our envisioned long-term usage, the function definitions would either be automatically acquired from a Web-service broker/repository or they could be semi-automatically generated by a tool, e.g., [3].

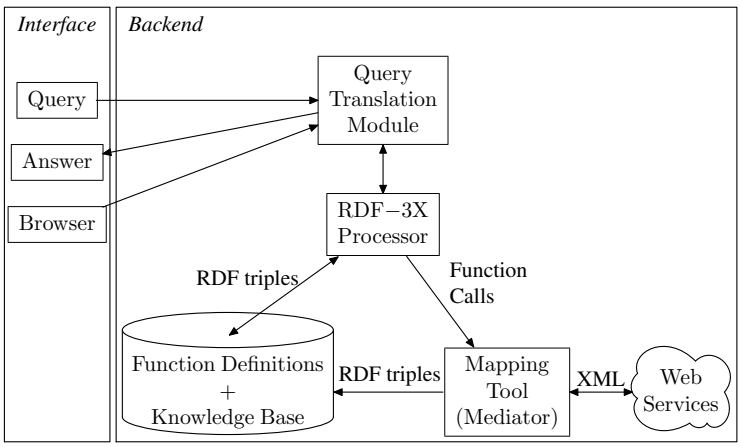

Figure 7: System architecture of ANGIE

Query Translation Module This is the core component of the project. The module takes as input a user query, and translates it into a sequence of function compositions. It implements the algorithms *DF* and *F-RDF*. The translation module continuously sends SPARQL queries and Web calls to the RDF-3X processor, which responds with new results.

SPARQL processor The SPARQL queries are executed using the RDF-3X processor [27]. The processor has been modified to accommodate the management of Web service calls. It is responsible for scheduling the execution of the function calls, and integrating the results in the processing of the input query. The calls are executed via the *Mapping Tool* (discussed below), which is in charge of remote invocation of the Web services. The Mapping Tool responds to the processor with the list of triples representing the answers of the calls. The RDF-3X processor combines the triples from the local knowledge base and the triples received from the mapping tool to produce a uniform output. The query translation and the query execution are interleaved.

The Mapping Tool This component executes the Web service calls. It mediates between the function declarations in the knowledge base and the schema of the XML documents that the function call returns. For this purpose, every function has two mappings associated with it: The *lowering mapping* defines how the input values of a function call are translated to the parameters of a REST (or SOAP) call. The call is sent to the remote site that provides the Web service. A call will yield values for the output variables in an XML fragment. The *lifting mapping* defines how the XML nodes in the answer are mapped to entities in the knowledge base. We use the XSLT standard [40] for this purpose. The entities can then be handled by the RDF-3X processor.

User Interface The user interface allows the user to query the knowledge base in the language described in Section 4.4.2. Queries are sent to the query translator module and answers are retrieved from there. Furthermore, the user can also display the knowledge base as a hyperbolic graph. One exploration step in this GUI retrieves and visualizes the neighborhood around a given entity. Such a browsing step translates into a simple query that retrieves the neighbors of that entity.

A detailed description of the system architecture can be found in the demo paper [30].

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
DF	1 / 6	32 / 490	3 / 78	1 / 13	2 / 11	11 / 71	4 / 7
F-RDF (R)	1 / 0	28 / 409	5 / 268	1 / 11	26 / 107	23 / 154	4 / 3
F-RDF	1 /0	42 / 80	5 / 238	1 / 5	26 / 34	23 / 48	4 / 2

Table 8: Evaluation results (answers/calls)

4.10 Performance evaluation

In this section, we present the experimental evaluation of our system on a set of popular Web services. Our results demonstrate the effectiveness and the efficiency of the *F-RDF* algorithm proposed in Section 4.8.2. We compare the *F-RDF* algorithm with the Prolog-style backtracking strategy of the *DF* algorithm, presented in Section 4.7. As a second competitor, we choose a modi-

fication of the *F-RDF* algorithm with a randomized strategy for choosing the next function calls. *F-RDF* and *F-RDF Random* both use the lazy strategy for constructing the query cover; and both algorithms generate Web calls using, as bindings, values extracted from the local knowledge base via binding queries. In contrast to *F-RDF*, *F-RDF Random* does not prioritize the list of partial instantiations according to the rules given in Section 4.8. Note that for a sequence of interrelated Web calls, the order cannot be changed by any of the strategies. Finally, the *DF* algorithm relies on depth-first search with backtracking.

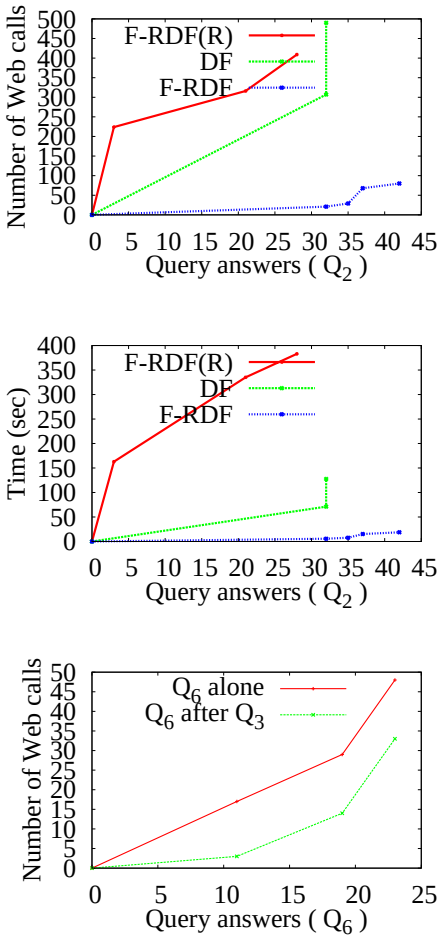

Figure 9: Distribution of answers during the evaluation

4.10.1 Testbed and Methodology

Evaluated Methods. We have implemented all algorithms, i.e. *DF*, *F-RDF* and *F-RDF Random*, as part of the query answering component of our prototype system. The fully functional system is implemented in Java. For all the algorithms, we set the bound for the total number of Web service calls to 1500. As performance metrics, we measured the total number of answers output by

each algorithm, the number of Web service calls, and the time necessary to output the answers.

Platform. The configuration of the machine that we used for the experiments is as follows: Pentium(R) Dual-Core CPU 2.50GHz with 2 MB cache, 2 GB memory, Debian 4.3.2, Kernel version 2.6.30, ServerX, JVM 1.6, gcc 4.3.2 .

Data sources. For our experiments, we consider the following Web sites, which export rich information from different domains via Web services: *isbndb.org*, *librarything.com*, and *abebooks.com* for books, *internetvideoarchive.com* for movies, *musicbrainz.org*, *last.fm*, *discogs.com*, and *lyricWiki.org* for music. All these Web sites allow users to query the underlying data based on various search criteria supported by corresponding service calls. For each Web service API, we defined mapping functions from the XML output into the RDF knowledge base.

Queries. For our experiments, we consider only queries for which the answers cannot be found in the local knowledge base (i.e., they can only be retrieved through Web services). We have tested our system for various queries. In Table 10, we show seven representative queries for which we report results. For each of the seven query templates, we evaluate a set of similar queries by varying the constants. A total of 70 queries have been used in the measurements. Most of the queries have different alternative ways of composing function instantiations. Usually, this leads to a high number of Web service calls.

Q_1	"Frank Sinatra" bornOnDate ?birthday
Q_2	?author wrote ?book ?author hasWon "Nobel Prize in Literature" ?author isCitizenOf "Greece"
Q_3	"Frank Sinatra" hasChild ?child "Frank Sinatra" isMemberOf ?collaboration ?child isMemberOf ?collaboration
Q_4	"Reese Witherspoon" marriedTo ?spouse "Reese Witherspoon" actedIn ?movie ?spouse actedIn ?movie
Q_5	"Jane Austen" wrote ?book ?title titleOf ?book ?title titleOf ?movie ?actor actedIn ?movie
Q_6	"Frank Sinatra" hasChild ?child ?child sang ?song
Q_7	"Kristin Scott Thomas" actedIn ?movie ?title titleOf ?movie ?title titleOf ?book ?author wrote ?book

Table 10: Queries

Profiling Web Services. In order to calibrate the cost parameters of different

Web services, we ran series of service calls. For each function definition, we executed 200 corresponding service calls, using as input entries selected from the YAGO knowledge based [35], which in turn was extracted from Wikipedia. We computed the average response time for each type of functions. For each edge in the function definition, we compute the average number of RDF triples matching the function edge in the call results. Furthermore, for each edge in the function definition, we measure the incompleteness of the external Web service, as the fraction of call results returning non-matching (i.e., irrelevant) triples. Figures 11 and 12 show the results for the edges *(?x, sang, ?y)* and *(?x, wrote, ?y)*, respectively. Note that the singer name cannot be used alone as input to any of the functions about the *sang* relationship. Instead, the function can accept only the *id* of a person, as defined by the *MusicBrainz* Web site. One can obtain the *id* using a prior call to another function; the function *getArtistId* provides the *id* of a singer when the singer name is given as input.

Web Site	Function Name	*(?x,?y)*	avg.	p	t (sec)
Music-Brainz	songsByArtistId	(O,O)	15.22	0.99	0.93
	songBySongId	(O,O)	0.89	0.99	0.57
	songByTitle	(O,I)	5.06	0.99	0.92
last.fm	songBySongId	(O,O)	1.00	0.08	0.46
	songsByTitle	(O,I)	5.81	1.00	0.89

Table 11: For the function edge *(?x, sang, ?y)*, the average number of matching triples in Web service call results

Web Site	Function Name	$(?x,?y)$	avg.	p	t (sec)
isbndb	booksByAuthorId	(O,O)	0.56	1.00	0.69
	booksByTitle	(I,O)	1.46	0.99	1.41
AbeBooks	bookByTitle	(O,I)	2.19	0.99	1.92
	booksByAuthor	(I,O)	7.28	0.99	1.93
	booksByISBN	(I,O)	6.10	1.00	1.07
Library-Thing	bookByTitle	(O,I)	0.71	0.85	1.85
	bookByISBN	(O,O)	0.31	0.96	0.86
	bookById	(O,O)	1.85	0.74	0.80

Table 12: For the function edge *(?x, wrote, ?y)*, the average number of matching triples in Web service call results

4.10.2 Results

Experiment 1: Answers & Web calls. In Figure 8, we present the results of the seven queries showed in Figure 10. The evaluation in each case was bound to 1500 Web calls. In this setting, the number of answers returned by each algorithm serves as comparison metric. The *F-RDF* algorithm produces the largest number of answers in each case. The figure also shows the number of calls after which all output answers are computed. We note that for the cases where the constants can be pushed as input parameters in Web calls e.g. Q_1, the number of Web calls leading to answers is small for all the algorithms. For queries where compositions of Web calls are necessary, the number of Web

calls increases considerably, and the difference between *F-RDF* and *DF* becomes obvious. In Figures 9, the left most graph shows the relation between the number of answers and the number of Web calls that are executed in order to obtain the answers. The second graph shows the relationship between the number of output answers and the evaluation time. The *F-RDF* algorithm converges fast to its total number of answers, and outputs the largest number of answers with respect to its competitors.

Experiment 2. The second experiment illustrates the effect of warehousing the data used in the evaluations of similar queries that preceded the current query. We show that the algorithm *F-RDF* makes uses of the local data in order to reduce the number of Web calls. Consider the following scenario. A user is exploring information about *Frank Sinatra*. Assume that he asks Q_3 and then Q_6. We measure the number of calls necessary to execute Q_6 after Q_3 was executed, and we compare it with the case where only Q_6 is executed. In Figures 9, the right most graph shows the results.

Experiment 3. This experiment measures the precision and the recall of the query results. We consider 100 queries that ask for the books published by an author whose name is given. More than 98% of the output books were correct answers.

4.11 Conclusion

This paper has introduced a system for dynamically incorporating data from Web services into an RDF knowledge base, on demand for given user queries. We call this paradigm "Active Knowledge", as it allows the knowledge base to actively and automatically complete or update its facts on entities or topics that the user is currently exploring. This happens transparently to the user, so that all browsing and querying of facts remains to be via RDF and SPARQL.

We emphasize again that our setting is different from a source-schema integration problem, as the Web services only provide encapsulated functions and do not expose any data schemas. The technical focus of this paper has been on efficiently generating sequences of function calls, with appropriately set parameters, to be executed by Web services based on judicious cost estimates. We could demonstrate, by experiments with a large knowledge base and prominent real-life Web services, that our algorithms obtain high recall with good answers delivered within the allowed cost budget for external service calls.

References

[1] Serge Abiteboul, Omar Benjelloun, and Tova Milo. "The Active XML project: an overview". In: *VLDB J.* (2007).

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Bernd Amann et al. "Mapping XML Fragments to Community Web Ontologies". In: *WebDB*. 2001.

[4] Arvind Arasu and Raghav Kaushik. "A grammar-based entity representation framework for data cleaning". In: *SIGMOD Conference*. 2009.

[5] Sören Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *The Semantic Web* (2008).

[6] Michele Banko et al. "Open Information Extraction from the Web". In: *IJCAI*. 2007.

[7] Daniela Berardi et al. "Automatic Composition of Transition-based Semantic Web Services with Messaging". In: *VLDB*. 2005.

[8] Christian Bizer et al. "Linked data on the web (LDOW2008)". In: *WWW*. 2008.

[9] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. "A general datalog-based framework for tractable query answering over ontologies". In: *PODS*. 2009.

[10] Bogdan Cautis, Alin Deutsch, and Nicola Onose. "Querying data sources that export infinite sets of views". In: *ICDT*. 2009, pp. 84–97.

[11] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. "Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web". In: *CIDR*. 2005.

[12] Alin Deutsch, Liying Sui, and Victor Vianu. "Specification and Verification of Data-driven Web Services". In: *PODS*. 2004.

[13] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. "Truth Discovery and Copying Detection in a Dynamic World". In: *PVLDB* 2.1 (2009).

[14] Oliver M. Duschka and Michael R. Genesereth. "Answering Recursive Queries Using Views". In: *PODS*. 1997.

[15] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. "Recursive Query Plans for Data Integration". In: *J. Log. Program.* 43.1 (2000).

[16] Ronald Fagin et al. "Clio: Schema Mapping Creation and Data Exchange". In: *Conceptual Modeling: Foundations and Applications*. 2009.

[17] Dayne Freitag and Nicholas Kushmerick. "Boosted Wrapper Induction". In: *AAAI/IAAI*. 2000.

[18] Marc Friedman and Daniel S. Weld. "Efficiently Executing Information-Gathering Plans". In: *IJCAI (1)*. 1997.

[19] Hector Garcia-Molina et al. "The TSIMMIS Approach to Mediation: Data Models and Languages". In: *J. Intell. Inf. Syst.* 8.2 (1997).

[20] Alon Y. Halevy. "Answering queries using views: A survey". In: *VLDB J.* 10.4 (2001).

[21] Mustafa Jarrar and Marios D. Dikaiakos. "MashQL: a query-by-diagram topping SPARQL". In: *ONISW*. 2008.

[22] Subbarao Kambhampati et al. "Optimizing Recursive Information Gathering Plans in EMERAC". In: *J. Intell. Inf. Syst.* 22.2 (2004).

[23] Gjergji Kasneci et al. "NAGA: Searching and Ranking Knowledge". In: *ICDE*. 2008.

[24] Ioanna Koffina et al. "Mediating RDF/S Queries to Relational and XML Sources". In: *Int. J. Semantic Web Inf. Syst.* 2.4 (2006).

[25] Chung T. Kwok and Daniel S. Weld. "Planning to Gather Information". In: *AAAI/IAAI, Vol. 1*. 1996.

[26] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. "Querying Heterogeneous Information Sources Using Source Descriptions". In: *VLDB*. 1996.

[27] Thomas Neumann and Gerhard Weikum. "RDF-3X: a RISC-style engine for RDF". In: *PVLDB* 1.1 (2008).

[28] Axel Polleres. "From SPARQL to rules (and back)". In: *WWW*. 2007.

[29] Rachel Pottinger and Alon Y. Levy. "A Scalable Algorithm for Answering Queries Using Views". In: *VLDB*. 2000.

[30] Nicoleta Preda et al. "ANGIE: Active Knowledge for Interactive Exploration". In: *PVLDB* 2.2 (2009).

[31] Ken Q. Pu, Vagelis Hristidis, and Nick Koudas. "Syntactic Rule Based Approach to Web Service Composition". In: *ICDE*. 2006.

[32] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. "Answering Queries Using Templates with Binding Patterns". In: *PODS*. 1995.

[33] Pierre Senellart et al. "Automatic Wrapper Induction from Hidden-Web Sources with Domain Knowledge". In: *WIDM*. 2008.

[34] David E. Simmen et al. "Damia: data mashups for intranet applications". In: *SIGMOD*. 2008.

[35] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Core of Semantic Knowledge". In: *16th international World Wide Web conference (WWW 2007)*. New York, NY, USA: ACM Press, 2007.

[36] Metaweb Technologies. *The Freebase project*. http://freebase.com.

[37] Snehal Thakkar, José Luis Ambite, and Craig A. Knoblock. "Composing, optimizing, and executing plans for bioinformatics web services". In: *VLDB J.* 14.3 (2005).

[38] Vasilis Vassalos and Yannis Papakonstantinou. "Describing and Using Query Capabilities of Heterogeneous Sources". In: *VLDB*. 1997.

[39] Word Wide Web Consortium. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation 2004-02-10.

[40] Word Wide Web Consortium. *XSL Transformations (XSLT)*. W3C Recommendation 1999-11-16.

[41] World Wide Web Consortium. *SPARQL Query Language for RDF (W3C Recommendation 2008-01-15)*. http://www.w3.org/TR/rdf-sparql-query/. 2008.

[42] Fei Wu and Daniel S. Weld. "Automatically refining the Wikipedia infobox ontology". In: *Proc. of the Int. WWW Conf.* 2008.

[43] Vladimir Zadorozhny et al. "Efficient evaluation of queries in a mediator for WebSources". In: *SIGMOD Conference*. 2002.

Chapter 5

Web Service Querying with SUSIE

5.1 Overview

The API of a Web service restricts the types of queries that the service can answer. For example, a Web service might provide a method that returns the songs of a given singer, but it might not provide a method that returns the singers of a given song. If the user asks for the singer of some specific song, then the Web service cannot be called – even though the underlying database might have the desired piece of information. This asymmetry is particularly problematic if the service is used in a Web service orchestration system.

In this chapter, we propose to use on-the-fly information extraction to collect values that can be used as parameter bindings for the Web service. We show how this idea can be integrated into a Web service orchestration system. Our approach is fully implemented in a prototype called SUSIE. We present experiments with real-life data and services to demonstrate the practical viability and good performance of our approach. This chapter is based on

> Nicoleta Preda, Fabian M. Suchanek, Wenjun 'Clement' Yuan, Gerhard Weikum
> "SUSIE: Search Using Services and Information Extraction"
> (International Conference on Data Engineering (ICDE) 2013)

5.2 Introduction

5.2.1 Motivation

There is a growing number of Web services that provide a wealth of information. There are Web services about books (*isbndb.org*, *librarything.com*, *Amazon*, *AbeBooks*), about movies (*api.internetvideoarchive.com*), about music (*musicbrainz.org*, *lastfm.com*), and about a large variety of other topics. Usually, a Web service is an interface that provides access to an encapsulated back-end database. For example, the site *musicbrainz.org* offers a Web service for accessing its database about music albums. The Web service defines functions that can be called remotely. *musicbrainz.org* offers the function *getSongs*, which takes a

singer as input parameter and delivers the songs by that singer as output. If the user wants to know all songs by Leonard Cohen, she can call *getSongs* with Leonard Cohen as input. The output will contain the songs *Suzanne*, *Hallelujah*, etc.

Web services play a crucial part in the trend towards data-centric applications on the Web. Unlike Web search engines, Web services deliver crisp answers to queries. This allows the user to retrieve answers to a query without having to read through several result pages. Web services can also be used to answer precise conjunctive queries, which would require several searches on the Web and joins across them, if done manually with a search engine. The results of Web services are machine-readable, which allows query answering systems to cater to complex user demands by orchestrating the services. These are advantages that Web services offer over keyword-based Web search.

Web services allow querying remote databases. However, the queries have to follow the *binding patterns* of the Web service functions, by providing values for mandatory input parameters before the function can be called. In our example of musicbrainz, the function *getSongs* can only be called if a singer is provided. Thus, it is possible to ask for the songs of a given singer, but it is not possible to ask for the singers of a given song. If the user wants to know, e.g., who sang *Hallelujah*, then the Web service cannot be used to answer this question – even though the database contains the desired information. This restriction is not due to missing data, but a design choice of the Web service owner, who wants to prevent external users from extracting and downloading large fractions of its back-end database. On the client side, this is a highly inconvenient limitation, in which the data may be available, but cannot be queried in the desired way. We call this the problem of *Web service asymmetry*.

Intuitively, a binary relation R is asymmetric with respect to a set of functions, if the functions allow querying for one argument of R but not for the other one. Even among the most prominent data-service providers, many relations are asymmetric. We have examined *isbndb.org*, *librarything.com*, and *abebooks.com* for books, *internetvideoarchive.com* for movies, *musicbrainz.org*, *last.fm*, *discogs.com*, and *lyricWiki.org* for music. Table 1 lists relations that can be queried for the second argument, but not for the first.

citizenOf(pers,country)	*rating(movie,x)*
bornIn(pers,year)	*graduatedFrom(pers,univ)*
livesIn(pers,place)	*published(book,year)*
hasWon(pers,award)	*publishedBy(book,editor)*

Table 1: Asymmetric relations in Web services

The asymmetric relations are by no means outlandish: It is legitimate, e.g., to ask for singers who won a certain prize. The only way to deal with such asymmetric Web services is to try out all possible input values until the Web service delivers the desired output value. For example, if the user asks for the singer of the song *Hallelujah*, then we can use a semantic knowledge base such as YAGO[35], Freebase[1] or DBpedia[4] to get a list of singers. Then we call *getSongs* with every singer, and remember those for which the Web service returns *Hallelujah*. Obviously, this approach quickly becomes infeasible. The

[1] *http://freebase.com*

first limitation is runtime, with Web service calls taking up to 1 second to complete. Trying out thousands of singers that a knowledge base such as YAGO contains could easily take hours. The second limitation is the data provider itself, which most likely restricts aggressive querying from the same IP address. If the asymmetric relations could be queried on a case-by-case basis, without materializing the entire function, then both the user and the provider would be helped: The user, because she can answer her queries; and the provider, because aggressive querying is avoided.

The functions exported by Web service APIs can be seen as views with binding patterns [33]. There are several methods to evaluate queries on such views efficiently[28, 23, 12, 33, 15, 31, 11]. Yet, these approaches do not address the asymmetry issue. If faced with an asymmetric relation, these approaches have to enumerate an entire domain until they stumble upon the correct value. Since the approaches assume an infinite budget of function calls, they can afford to enumerate the domain. In the context of Web services, however, this approach is out of question.

The problem is even more challenging, because asymmetric relations may appear in query plans that orchestrate several Web service functions. Assume, e.g., that the user wishes to find the albums that feature the song *Hallelujah*. Assume that we have a service that delivers the album of a song, if we provide the id of the song. To retrieve the id of the song, we have to call another function that requires the singer of the song. This gives rise to a query composition tree where the asymmetric relation is a leaf node. If an alternative service composition also promises to deliver the album, then both compositions have to be weighted against each other. This is not trivial. Assume, e.g., a service that, given a song, delivers songs that it inspired, together with their album. A service orchestration system might call this service with *Hallelujah*, retrieve inspired songs, and then call the service again with these results, and again and again, in the vain hope to find the album of *Hallelujah*. This way, it descends into a chaining of derived songs. We show in this paper that, even if the inverse functions are available, standard approaches to query answering cannot prioritize them over the chains.

5.2.2 Contribution

In this paper, we develop a solution to the problem of Web service asymmetry. We propose to use Web-based information extraction (IE) on the fly to determine the right input values for the asymmetric Web services. For example, to find all singers of *Hallelujah*, we issue a keyword query "singers Hallelujah" to a search engine. We extract promising candidates from the result pages, say, *Leonard Cohen*, *Lady Gaga*, and *Elvis*. Next, we use the existing Web service to validate these candidates. In the example, we would call *getSongs* for every candidate, and see whether the result contains *Hallelujah*. This confirms the first singer and discards the others. This way, we can use an asymmetric Web service as if it allowed querying for an argument that its API does not support. We show how such functions can be integrated into a Web orchestration system, and how they can be prioritized over infinite chains of calls. Our technique works only if the Web provides adequate candidate entities. We show in our experiments that this is the case in an interesting spectrum of applications. Our paper makes the following contributions:

1. A fully automated solution to the problem of Web service asymmetry, where input values for the Web services are extracted on-the-fly from Web pages found by keyword queries.

2. A modification to the standard datalog evaluation procedure that prioritizes inverse functions over infinite call chains.

3. An experimental evaluation of our approach with the APIs of real Web services, showing how we can improve the performance of a real-world query answering system.

Our methods are fully implemented in the SUSIE system (Search Using Services and Information Extraction). The rest of this paper is structured as follows. Section 2 contains preliminaries. Section 3 discusses execution plans. Section 4 introduces the key concept of inverse functions and their scheduling. Section 5 discusses the generation of inverse functions and their implementation. Section 6 presents comprehensive experiments with real-world Web services. Section 7 discusses related work, and we conclude in Section 8.

5.3 Preliminaries

Global Schema. A Web service defines an API of functions that can be called over the Internet. Given some input values, a function returns as output a semi-structured document, usually in XML. In all of the following, we assume that the set of Web service functions is known. We also assume that all functions operate on the same, conceptually global database with a unified schema. This is an important assumption in our context, which we make in line with other works in the area [16, 33, 12, 32, 15, 28, 23]. The mappings from the actual schemas of the functions to the global schema can be defined manually, or they can be automatically generated using tools [18]. We see this as a challenge that is orthogonal to the present work. The fact that the functions operate on different data sources can be modeled by source designators, as we shall see later. As a running example, consider the global schema and the database shown in Figure 2.

sang

singer	*songId*
Cohen	1
Elvis	2
Elvis	3

hasTitle

songId	*title*
1	Hallelujah
2	All Shook Up
3	Teddy Bear

onAlbum

songId	*album*
1	Various Positions
2	Memphis To Vegas
3	Loving You

Table 2: A sample global database

Function Definitions. Users and application programs do not have access to

the global database. Rather, access has to go through the functions provided by Web services operating on the global database. For example, a function can be *getSongs(in:singer, out:songId, out:title)*. This function expects singer as input and returns the song id and title as output. In all of the following, *i, t, a,* and *s* will be variables for song ids, song titles, albums, and singers, respectively. If we call *getSongs(Cohen, i, t)*, we receive as output *i=1*, *t=Hallelujah*. Seen this way, the function provides a view on the global database. More formally, and again in line with [16, 33, 12, 32, 15, 28, 27, 23], we see the Web service functions as views with binding patterns over the global database.

Definition 11 (Function Definition) *A function definition over a global database schema is a rule of the form*

$$f^{\bar{A}}(\bar{X}_0) \leftarrow r_1(\bar{X}_1), r_2(\bar{X}_2), \ldots r_n(\bar{X}_n)$$

where $r_1, r_2, \ldots r_n$ *are database relations and the* $\bar{X}_i$ *are tuples of variables and/or constants. All variables of the head must occur in the body of the rule.* $\bar{A}$ *is the adornment of the function. It is a string of length* $|\bar{X}_0|$ *composed of the letters b and f. The meaning of the adornment b is that a binding value must be provided for the variable in that position, whereas the adornment f does not impose such a restriction.*

The function *getSongs*, e.g., can be written as follows:

$$getSongs^{bff}(s,i,t) \leftarrow hasTitle(i,t),\ sang(s,i)$$

This rule defines how the Web service provider computes the function *getSongs*. From the caller's point of view, the adornment *bff* states that *getSongs* can be called only if a value for *s* is provided. The evaluation of the function call binds the variables *i* and *t* to the values that satisfy the conditions *hasTitle(i,t)* and *sang(s,i)*. Variables that appear in the body of the rule, but not in its head are called *existential variables*. For example, a function may return all albums by a singer:

$$getAlbums^{bf}(s,a) \leftarrow sang(s,i),\ onAlbum(i,a)$$

Again, this rule specifies how the Web service provider computes the function. These functions are the only way for a user or application to access the global database.

Incomplete Functions. Web services are often incomplete. Suppose, e.g., that *musicbrainz* contains only Elvis. We model this incompleteness by source designators. A source designator is an atom that can be appended to the body of a function to indicate that the underlying data source is incomplete. In the example, we define

$$getAlbums^{bf}_{mb}(s,a) \leftarrow sang(s,i),\ onAlbum(i,a),\ mb(s)$$

where *mb(s)* is a global relation that holds for those singers that *musicbrainz* knows. Appropriate source designators can model the fact that some functions of the same Web service talk about the same entities, or conversely, that functions of different services operate on different data sources.

Database Functions. In some cases, the user has access to a database or knowledge base such as YAGO [35] or DBpedia [4]. These databases can also be seen as (incomplete) functions on the global database. They have only *f*-adornments and no *b*-adornments, and whenever they are called, they are instantiated with tuples from the database. The following function, e.g., is a database function that delivers all singers from YAGO:

$$getSingers^{f}_{yago}(s) \leftarrow sang(s,i),\ yago(s)$$

Queries. We consider the evaluation of *conjunctive queries* defined over the global schema. For uniformity and in line with [33, 16] we adopt the datalog notation for queries.

Definition 12 (Query) *A query is a datalog rule of the form*

$$q(\bar{X}_0) \leftarrow r_1(\bar{X}_1), r_2(\bar{X}_2), \ldots r_n(\bar{X}_n)$$

where $r_1, r_2, \ldots r_n$ *are database relations and the* $\bar{X}_i$ *are tuples of variables and/or constants. All variables of the head must occur in the body of the rule.*

The following query, e.g., asks for the singer of *Hallelujah*:

$$q(s) \leftarrow hasTitle(i,\ Hallelujah),\ sang(s,i)$$

Variables that appear in the body of the rule, but not in its head are called *existential variables*. An *answer to a query* is an answer to the query on the global database. In the example, an answer to the query is *s=Cohen*.

5.4 Execution Plans

Goal. Our goal is to answer queries by using only function calls. This goal is in line with the works [33, 12, 15]. Since the user does not have access to the global database, we cannot execute the query directly on the tables of the global database. Rather, the algorithms automatically translate the query into query plans expressed in terms of the available Web service functions, respecting their binding pattern restrictions. This is a non-trivial endeavor that we discuss next.

Different from previous work, we consider a given budget of calls. This changes the goal of the evaluation. Previous work aimed to compute the maximal number of query answers. The goal was to compute the *maximal contained rewriting*. Unfortunately, when the views have binding pattens, even the evaluation of conjunctive queries requires rewritings of unbound length [33]. Thus, these works will produce pipelines of calls of an unbound length in order to obtain the maximal number of answers. This is infeasible in the context of Web services. Our goal, in contrast, is to compute the largest number of answers using the given budget of calls. Hence, we have to prioritize calls that are likely to lead to an answer. This is different from the problem of join ordering in previous work. Therefore, we have to use a different model for the query answering process: our execution plans are ordered sequences of calls rather than ordered join plans.

Execution Plans. Consider the following query, which asks for albums by Cohen:

$q_1(a) \leftarrow sang(Cohen, i), onAlbum(i,a)$

We cannot access the tables *sang* and *onAlbum* directly. Rather, we have to access the tables through functions. Suppose, e.g., that we have the following functions:

$getAlbum^{bf}(i,a) \leftarrow onAlbum(i,a)$
$getSongs^{bff}(s,i,t) \leftarrow sang(s,i), hasTitle(i,t)$

We can first call $getSongs^{bff}(Cohen, i,t)$, which will return the ids and titles of all songs by Cohen (i.e., i=1, t=*Hallelujah*). Then, we call $getAlbum^{bff}(i,a)$ with every binding of i, which will give us values for a. In the example, there is only one value, a=*VariousPositions*. The sequence of calls is:

$getSongs^{bff}(Cohen,i,t), getAlbum^{bff}(i,a)$

This is a valid execution plan, because every b-argument of a function will have a value at execution time. We make this notion more formal now.

Definition 13 (Function Call) *A function call of a function f is a literal of the form $f(\bar{X})$, where $\bar{X}$ is a sequence of variables and constants of the same arity as f.*

In the example, $getSongs^{bff}(Cohen,i,t)$ and $getAlbum^{bff}(i,t,a)$ are function calls.

Definition 14 (Consequences) *The consequences of a function call $f(\bar{X})$ are the body atoms of the function definition of f, with variables substituted by the values from $\bar{X}$. Existential variables of f are substituted by fresh variables in the consequences.*

In the example, the consequences of $getSongs^{bff}(Cohen,i,t)$ are $\{hasTitle(i,t), sang(Cohen,i)\}$.

Definition 15 (Execution Plan) *An execution plan for a query Q is a sequence of function calls. Each argument in the execution plan has to be either (1) a constant symbol that appears in Q or (2) a variable.*

The plan is *admissible* if, for every variable, the first occurrence of the variable in the call sequence is in an f-position of a function call. The plan is *effective* if all body atoms of Q appear in the conjunction of the function call consequences (where existential variables of Q can be matched with any variables or constants in the consequences). Let us look again at our sample plan above. The plan is admissible because every b-argument of a function call is either a constant or has been bound by a previous function call. The plan is effective because the query atoms $sang(Cohen,i)$, and $onAlbum(i,a)$ appear in the conjunction of the consequences of the calls:

$sang(Cohen,i), hasTitle(i,t), onAlbum(i,a)$

Guessing Plans. Not all plans are promising. Assume, e.g., that the user asks for the album and singer of *Hallelujah*:

$$q_2(a,s) \leftarrow hasTitle(i, Hallelujah),\ onAlbum(i,a),\ sang(s,i)$$

Assume that we have the following functions:

$$getSongInfo^{bfff}(i,t,s,a) \leftarrow hasTitle(i,t), sang(s,i), onAlbum(i,a)$$
$$getRelSongs^{bff}(t_1,i_2,t_2) \leftarrow influenced(i_1,i_2),\ hasTitle(i_1,t_1),\ hasTitle(i_2,t_2)$$

The first function returns all information about a song id. The second function returns the id and title of a song that was influenced by the input song title. A similar function is published by *musicbrainz*. This allows for the following plan:

$$getRelSongs^{bff}(Hallelujah,\ i_2,t_2),\ getSongInfo^{bfff}(i_2,\ Hallelujah,\ s,a)$$

This plan starts with the song *Hallelujah*, and finds which songs it influenced. Each of these songs comes with an id i_2 and a title t_2. Then the plan calls $getSongInfo^{bfff}$ to check whether i_2 was by any chance the id of *Hallelujah*, in which case we get the singer s and the album a. Obviously, this is unlikely to succeed in reality. We call such a plan a *guessing plan* (a notion that we will make more precise later).

Unbound Guessing Plans. In some cases, there are infinitely many guessing plans. Our query q_2, e.g., gives rise to the following unbound number of plans:

$$getRelSongs^{bff}(Hallelujah,i_2,t_2),\ getRelSongs^{bff}(i_2,i_3,t_3),\ \ldots\ getRelSongs^{bff}(i_{n-1},i_n,t_n),\ getSongInfo^{bfff}(i_n,\ Hallelujah,\ s,a)$$

These plans will enumerate the entire domain of songs, in order to "guess" the input value of $getSongInfo^{bfff}$. In general, there can be infinitely many pipelines for a given query under a given set of functions [25]. We call such plans *unbound plans*. The goal of SUSIE is to avoid guessing plans and unbound guessing plans by adding *inverse functions*.

5.5 Execution Plans with Inverse Functions

This section will introduce the core contribution of SUSIE, inverse functions. We will show how to prioritize them over guessing plans.

5.5.1 Adding Inverse Functions

Let us consider again the query for the singer and album of *Hallelujah*:

$$q_2(a,s) \leftarrow hasTitle(i, Hallelujah),\ onAlbum(i,a),\ sang(s,i)$$

As before, we consider the following functions defined above: $getSongInfo^{bfff}(i, t,s,a)$, $getSongs^{bff}(s,i,t)$, $getAlbum^{bf}(i,a)$, and $getRelSongs^{bff}(t_1,i_2, t_2)$. We have already seen that these functions lead to the enumeration of the domain of songs by unbound guessing plans. Now let us add the following function:

$$getSinger^{bf}(t,s) \leftarrow sang(s,i),\ hasTitle(i,t)$$

This function retrieves the singer of a song. It has the same body as $getSongs^{bff}$, but a different binding pattern. We call it an *inverse function* of $getSongs^{bff}$, because it allows asking for the input parameter of $getSongs^{bff}$. If this function is provided, then the query that asks for the singer and album of *Hallelujah* can be answered by the following plan:

$$getSinger^{bf}(Hallelujah,s),\ getSongs^{bff}(s,\ Hallelujah,\ i),\ getAlbum^{bf}(i,a)$$

This plan first retrieves the singer of *Hallelujah*, s. Then, it retrieves all songs by s. If s was computed correctly, this call yields the identifier i of *Hallelujah*, which can then be used to call $getAlbum^{bf}$ and retrieve the album. Thus, the addition of the inverse function allows us to produce answers to the query without guessing. We will now discuss how to prioritize such functions in execution plans.

5.5.2 Standard Approaches to Query Answering

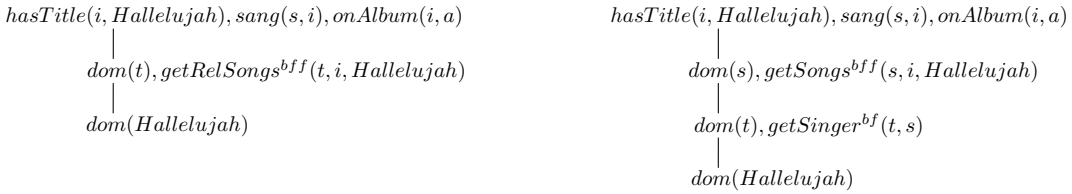

Figure 3: Sample SLD derivations for q: without the inverse function (left), and with the inverse function (right)

Transformation to datalog. The standard way of answering queries with binding patterns is to transform the function definitions into inverse rules.[2] This yields a datalog program, on which the query can be evaluated. An algorithm that is guaranteed to produce the maximal contained rewritings was introduced in [16]. Techniques for reducing the number of calls have been developed in [12, 28]. Their goal remains the computation of the maximum number of answers. As we shall show next, these methods cannot prioritize plans with inverse functions over unbound plans.

Inverse Rules. We illustrate the construction of the datalog program proposed in [16] for our function.

[2]Inverse rules have nothing to do with the inverse *functions* of SUSIE.

$$getRelSongs^{bff}(t_1,i_2,t_2) \leftarrow influenced(i_1,i_2),\ hasTitle(i_1,t_1),\ hasTitle(i_2,t_2)$$

For each body atom, we construct an *inverse rule*. The atom forms the head of the inverse rule, while the body consists of a *dom* atom for every bound variable of the function, followed by the function call atom. In the example, this yields

$$\begin{aligned} influenced(f_1,i_2) &\leftarrow dom(t_1),\ getRelSongs^{bff}(t_1,i_2,t_2)\\ hasTitle(i_2,t_2) &\leftarrow dom(t_1),\ getRelSongs^{bff}(t_1,i_2,t_2)\\ hasTitle(f_1,\ t_1) &\leftarrow dom(t_1),\ getRelSongs^{bff}(t_1,i_2,t_2) \end{aligned}$$

where $f_1 = f(t_1, getRelSongs)$ is a Skolem term [2] that replaces the existential variable i_1. Furthermore, we add one domain rule for every f-variable of the function. These rules look similar to the inverse rules, but have *dom* in their head:

$$\begin{aligned} dom(i_2) &\leftarrow dom(t_1),\ getRelSongs^{bff}(t_1,i_2,t_2)\\ dom(t_2) &\leftarrow dom(t_1),\ getRelSongs^{bff}(t_1,i_2,t_2) \end{aligned}$$

In addition, we add a domain rule with an empty body for each constant of the query. For the query q_2, we get:

$$dom(Hallelujah) \leftarrow$$

This way, we can produce a datalog program for a given query and a given set of function definitions. If the body predicates of a rule are evaluated left to right, then this program ensures that all input parameters of a function call are bound before the function is called.

Evaluation Strategies. Even if inverse functions are present, a datalog evaluation strategy has to enumerate all unbound plans in the worst case. This is because these plans could be the only plans that yield results. Thus, unbound plans cannot be excluded upfront. In [16], the authors suggest to use a bottom-up approach for evaluating the new datalog program. This is guaranteed to compute the maximum number of answers. However, it will also enumerate entire domains (e.g., all singers), and call all possible functions on all possible constants – no matter whether these calls contribute to the answer of the query or not. Obviously, this approach is infeasible in the context of Web services. Magic-set techniques [5] have been developed for optimizing bottom-up evaluations to ressemble top-down evaluations. This technique simulates the QSQ technique [2] for top-down evaluations. We consider next the top-down evaluation of the new datalog program and we show that even if an inverse function can avoid a guessing plan, standard top-down evaluation techniques cannot prioritize such plans.

Blindness to Inverse Functions. Let us now consider the top-down evaluation of

$$q_2(a,s) \leftarrow hasTitle(i,\ Hallelujah),\ onAlbum(i,a),\ sang(s,i)$$

Figure 3 shows how the atom *hasTitle(i, Hallelujah)* is expanded in two possible SLD (Selective Linear Definite) derivation trees using the rules of the new datalog program. Although the SLD trees are constructed top-down, the evaluation of the calls is bottom-up.

The left derivation tree is the guessing strategy: It starts with *dom(Hallelujah)*, and then calls $getRelSongs^{bff}$ *(t,i, Hallelujah)* with *t=Hallelujah*. It "hopes" that Hallelujah is a related song for itself. Then it calls $getSongInfo^{bfff}$ in a new branch. The right derivation tree uses the inverse function and leads to a solution in the case the functions contain the necessary data.

However, nothing allows the datalog processor to distinguish which of the two plans at Figure 3 is better. One may even think that the left plan is better since it uses less calls than the right one. This is because the crucial link between $getSongs^{bff}$ *(s,i,t)* and $getSinger^{bf}$ *(s,t)*, namely the fact that both functions share the same body, gets lost in the transformation to datalog. Worse, the left plan can be extended to an unbound guessing plan, in which the bottom *dom(Hallelujah)* is replaced by a branch where $getRelSongs^{bff}$ is repeatedly applied to its own outputs. Standard datalog evaluation cannot prioritize inverse functions over guessing plans, because the link between $getSongs^{bff}$ *(s,i,t)* and its inverse is lost.

5.5.3 Our Approach to Query Answering

Promising Function Calls. We aim to distinguish the guessing plan

$getRelSongs^{bff}$ *(Hallelujah,* i_2*,*t_2*),*
  $getSongInfo^{bfff}$ *(*i_2*, Hallelujah, s,a)*

from the "promising plan"

$getSinger^{bf}$ *(Hallelujah,s),*
  $getSongs^{bff}$ *(s, Hallelujah, i),* $getAlbum^{bf}$ *(i,a)*

Definition 16 (Promising Function Call) *A promising function call in an execution plan is a function call whose inputs come from the query or from previous promising function calls, and whose consequences are a subset of the consequences of the following function calls.*

In the example, the call to $getSinger^{bf}$ *(Hallelujah,s)* is a promising function call, because its consequences ({*sang(s,Hallelujah)*}) are a subset of the consequences of the following calls, {*sang(s, Hallelujah), hasTitle(i, Hallelujah), onAlbum(i,a)*}. The call $getRelSongs^{bff}$ of the left plan, in contrast, is not a promising function call. It guesses that its output will be the desired input of the following call to $getSongInfo^{bfff}$. Obviously, promising function calls should be preferred wherever possible. We can give the following theorem to support this intuition.

Theorem 3 (Promising Function Calls) *Given a query Q, and given an effective execution plan E for Q, such that there is some choice of input variables such that E returns answers to Q, the following holds: If we can add promising function calls to E, so that E becomes admissible, then E will deliver an answer for Q.*

Proof 1 *Assume that there is an effective execution plan E for a query Q, such that there is some choice of input variables such that E returns an answer A*

for Q. Consider a function call $f(\bar{X})$, so that the consequences of $f(\bar{X})$ are a subset of the consequences of E. Be $f \wedge E$ the conjunction of the consequences of $f(\bar{X})$ and the consequences of E. Then the evaluation of $f \wedge E$ on the global schema has the same results as the evaluation of the consequences of E on the global schema. Hence, the evaluation of $f \wedge E$ will contain A. If $f(\bar{X})$ makes E admissible, then $E \wedge f(\bar{X})$ will deliver A.

Intuitively, Theorem 3 tells us that promising function calls will not deteriorate the chance of obtaining an answer from a function composition. That is, if we have a function composition that can deliver answers to the query, but where we need to bind the inputs, then adding the promising function calls to bind the inputs is a safe step.

Algorithm. We will now discuss how to prioritize promising function calls in SLD derivations. We first generate the plans without calling any functions. Our plans are bounded by the call budget. Then, we check for every plan whether every function call either answers an unanswered query atom, or is a promising function call.

In our promising example plan, the function calls $getSongs^{bff}$ and $getAlbum^{bf}$ serve to answer the query. The function call $getSinger^{bf}$ serves to bind the inputs of $getSongs^{bff}$. Its consequences are contained in the consequences of the following call $getSongs^{bff}$*(s,i,Hallelujah)*:

$$\{sang(s,i),\ hasTitle(i,\ Hallelujah)\} \subseteq \{sang(s,i),\ hasTitle(i,Hallelujah)\}$$

Let us now look at the guessing plan. The call to $getRelSongs^{bff}$*(Hallelujah, i_2,t_2)* is not promising, because its consequences are not included in the consequences of the call $getSongInfo^{bfff}$*(i_2,Hallelujah,s,a)* that follows:

$$\{hasTitle(i,\ Hallelujah),\ influenced(i",i),\ hasTitle(i",t)\} \not\subseteq \{hasTitle(i,\ Hallelujah),\ sang(s,i),\ onAlbum(i,a)\}$$

Therefore, this plan should be postponed. Non-promising function calls cannot always be avoided. But plans with less non-promising calls are more likely to succeed. This allows the query evaluation to prioritize promising function calls over guessing function calls.

5.6 Inverse Functions

The previous section has shown how inverse functions can be prioritized in execution plans. The present section will show how inverse functions can be generated and implemented.

5.6.1 Definition

Asymmetric Relations. Consider a function definition that contains a relation r:

$$f^{b\ldots bf\ldots f}(x_1,\ldots,x_i,x_{i+1},\ldots,x_n) \leftarrow \ldots r(x_k,\ldots,x_l)\ldots$$

Let us consider a query atom $r(y_k, \ldots, y_l)$, in which some arguments are bound and some arguments are unbound. Thus, the query atom defines a binding pattern of bound and free variables on r, $\bar{A}_r$. If $\bar{A}_r$ does not provide bound values for all input arguments of f, then f cannot be called. We say that f *does not support* the binding pattern $\bar{A}_r$. If the set of functions F contains no function that supports $\bar{A}_r$, then we say that $\bar{A}_r$ is *unsupported* in F. If there exists a binding pattern of r that is supported in F, and if there exists another binding pattern that binds at least one variable and that is unsupported in F, then we say that r is *asymmetric* with respect to F. Intuitively speaking, this means that there are arguments of r for which we can query and there are other arguments for which we cannot query.

Inverse Functions. If a function f does not support a binding pattern of a relationship, then the query evaluation might have to enumerate an entire domain to "guess" input values. Therefore, we introduce the *inverse functions* of f.

Definition 17 (Inverse Function) *Given a function $f^{\bar{A}}(\bar{X}_0)$, an inverse function of f is a function $f^{\bar{A}'}(\bar{X}_0)$ with the same rule body. $\bar{A}'$ is an adornment that has at least one bound argument, and that leaves at least one input argument of f free. As an example, consider again the following function:*

$getSongs^{bff}(s,i,t) \leftarrow sang(s,i), hasTitle(i,t)$

This function requires the singer as input, and delivers the song id and title as output. It has 3 inverse functions:

$getSongs^{ffb}(s,i,t)$, $getSongs^{fbf}(s,i,t)$, $getSongs^{fbb}(s,i,t)$

Adding Inverse Functions. Inverse functions can be used to answer query atoms that have an otherwise unsupported binding pattern. Yet, inverse functions are not always necessary. If, say, there is a function f that supports the desired binding pattern for a relation, then it is not necessary to add an inverse function for some other function g that does not support the desired binding pattern. However, in the context of Web services, the inverse of g would still be necessary in order to obtain as many results as possible. This is because Web services are typically incomplete. The function f may not yield all the instances that g stores. Thus, one potentially loses results if one queries only f.

Even if f is complete, it can still be beneficial to have the inverse of g. This is because Web services typically come at a cost. This can be a cost in bandwidth, in time, or also in financial terms (for commercial Web services). It can be that the inverse of g is cheaper than f. In this case, the inverse of g can still be preferable to f.

Now assume that there is an orchestration of functions that allows finding the input values for g, so that g delivers instances even for an unsupported binding pattern. Since Web services tend to be incomplete, this orchestration might still not find *all* input values of g. Thus, g might store more instances than can be obtained this way. Thus, the inverse of g is still necessary to maximize the number of results.

In all of these cases, the inverse of g proves useful to have – be it to maximize the number of results or to minimize the cost of calls. Therefore, we aim to add as many inverse functions to our program as possible.

5.6.2 Implementation

Providing Inverse Functions. We will now explain how certain inverse functions can be provided and implemented. Our method is intended for cases where at most one input argument of the original function is unbound, and where the necessary information appears on the surface Web. We discuss these limitations in Section 5.6.3. As an example, consider again the function *getSongs*:

$$getSongs^{bff}(s,i,t) \leftarrow sang(s,i),\ hasTitle(i,t)$$

Information about singers and songs are likely to appear on the Web. Therefore, we can provide the inverse functions where one of these values is bound. As an example, consider

$$getSongs^{ffb}(s,i,t) \leftarrow sang(s,i),\ hasTitle(i,t)$$

Our system, SUSIE, implements this inverse function as follows. Whenever $getSongs^{ffb}(s,i,t)$ is called, SUSIE issues a keyword query of the form "Singer t" to a Web search engine (e.g., "Singer Hallelujah"). Then, SUSIE will extract possible candidates for s from the result Web pages. Last, SUSIE will call the real Web service $getSongs^{bff}(s,i,t)$ with all candidates for s. If any of them succeeds and delivers the given t as output, then SUSIE returns the values of s and i as an output of the inverse function $getSongs^{ffb}$. Thereby, the inverse function acts just like a real Web service function. We will now discuss three subtasks of this endeavor: (1) the task of finding suitable Web pages, (2) the task of extracting candidates from the Web pages, and (3) the task of verifying the results. In all of the following, we treat the generic case of implementing the inverse functions for a function $f^{b\ldots bf\ldots f}(x_1,\ldots,x_n)$, where the first argument of the inverse function is unbound.

Finding Web Pages. We assume that the function f comes with a domain for its free argument x_1, the *target type*. In the example of *getSongs*, we know that the target type is *Singer*. We map each inverse function $f^{\bar{A}}$ of f to a *keyword query*. This keyword query is a string of the form $tt\ \bar{X}$, where tt is the target type and $\bar{X}$ are the bound variables of $\bar{A}$. In the example, the keyword queries are

$getSongs^{ffb}(s,i,t)$: "singer t"
$getSongs^{fbb}(s,i,t)$: "singer t i"

In actual implementations, the keyword queries can be more sophisticated. For example, it may turn out to be beneficial to map $getSongs^{fbb}(s,i,t)$ to the keyword query "List of singers who sang t", ignoring the song id i. Whenever the query evaluation calls the function $f^{\bar{A}}(x_1,\ldots,x_n)$, SUSIE issues the keyword query of $f^{\bar{A}}$ to an Internet search engine (we used Google). SUSIE collects the top ten result Web pages.

Extracting Candidates. Once the Web pages have been retrieved, it remains to extract the candidate entities. Information extraction is a challenging endeavor, because it often requires near-human understanding of the input documents. Our scenario is somewhat simpler, because we are only interested in

extracting the entities of a certain type from a set of Web pages. We have implemented two simple yet effective IE algorithms as a proof of concept.

String Matching Algorithm. This algorithm extracts only entities that are already known to a knowledge base. In our experiments, we use the YAGO knowledge base [35]. YAGO feeds from Wikipedia and thus covers a large number of entities of common interest. The algorithm first loads all entities of the target type from the knowledge base into a trie [19]. Then the algorithm runs through the Web pages and extracts all entities from the documents that appear in the trie. This processing can be done in time $O(n)$ in the best case and in time $O(m \cdot n)$ in the worst case, where n is the total number of characters in the Web pages and m is the number of characters in the longest entity name.

Structured Extraction Algorithm. The String Matching Algorithm has the disadvantage that it can only find entities that appear in the knowledge base. If we wish to venture beyond this limitation and find new entities, we may exploit that many result Web pages will have a structured form. Typically, tables represent a natural way to organize sets of relationships in Web pages. However, as shown in [20], only a small fraction of the Web tables are encoded using the `<table>` markup in HTML. In many cases, they are encoded using lists or loosely repetitive structures. Our algorithm identifies structures of repetitive rows, where each row contains items that are separated by special strings or tags that re-appear in each row. Furthermore, the items in one column have to be of the same syntactic type (numbers, strings or dates). This procedure finds standard tables and standard lists as well as other types of repetitive structures. By comparing the elements of each column with the instances of the target type in YAGO, our algorithm finds the column that constitutes most likely the answers to the query.

We note that these are just two possible implementations. They can be replaced by more sophisticated ones [17, 10].

Verifying Candidates. The IE algorithms have delivered a set of candidate entities for the free argument x_1 of f. In case of the String Matching Algorithm, these candidates have been filtered by a knowledge base. Still, this does not mean that the candidates would be correct: Web pages can contain many more entities of the target type than the desired ones. Therefore, all candidates have to be checked by the Web service to see whether they fulfill the conditions of the function definition. To do so, SUSIE will call the original Web service function f with all of these candidate entities, one after the other. This yields one, multiple, or no result tuples of the form $\langle x_1, \ldots, x_n \rangle$ for each candidate entity x_1. If the values in the bound positions of $\bar{A}$ correspond to the input values of the inverse function call $f^{\bar{A}}(x_1, \ldots, x_n)$, then SUSIE delivers the tuple $\langle x_1, \ldots, x_n \rangle$ as an output of the inverse function. Thereby, each candidate entity that has been extracted from the Web pages is verified by the Web service.

5.6.3 Properties of SUSIE

IE and Web Services. The candidate entities often appear multiple times in the Web pages. This increases the chances of an IE algorithm to find them. A high precision and a high recall are desirable for the IE algorithms, but they are not strictly necessary. If the precision of the extraction is low, this will

result in more Web service calls, but not in diminished precision of the final query answers. This is because all answers are checked by the Web service. If the recall this low, this will result in fewer answers. Fewer answers, however, are better than trying out all possible input values for the function, which may result in no answer at all due to the limited call budget. By verifying each result with the original Web service, our approach mitigates the central weakness of classical information extraction, its imperfect precision. By using information extraction to extract candidates, our approach mitigates the central limitation of Web services, the restrictive access patterns.

Limitations of SUSIE. The current implementation of SUSIE creates inverse functions only if at most one input argument of the original function is unbound. The case of functions with multiple unbound inputs is significantly harder: It could require fact extraction of n-ary facts. This is, by itself, a hard problem. If one uses a naive entity recognition algorithm (such as the string matching algorithm), then one can generate a number of candidate tuples that is exponential in the number of inputs. Therefore, we leave the case of multiple inputs for future work. As we show in the experiments, even the case with one input delivers significant mileage in practice.

We can implement inverse functions only if good candidate entities for a query can be found on the Web. This is certainly not true for a large number of functions. In these cases, we do not provide the inverse functions. However, experience from our experiments indicates that information of common interest is publicly available on the Web in a large spectrum of cases. In these cases, we can provide the inverse functions. Our claim is not that SUSIE could make all queries answerable. Rather, our claim is that SUSIE can make queries answerable in an interesting spectrum of cases. Our experiments show that in these cases, SUSIE improves the query results drastically.

5.7 Performance evaluation

We conducted two types of experiments. We first evaluate the performance of the information extraction algorithms. Then, we evaluate the performance of SUSIE on real-world queries.

5.7.1 Information Extraction

Test Set. To evaluate the IE algorithms, we targeted three query types: Queries that ask for actors with a certain birth year, for actors with a certain nationality and for authors who received a certain prize. For each query type, we chose 10 arbitrary property values (10 birth years, 10 nationalities and 10 literature prizes). For each property value, we generated the keyword query that SUSIE would generate, sent it to Google and retrieved the top 10 pages. This gave us 100 pages for each test set. The pages are quite heterogeneous, containing lists, tables, repetitive structures and full-text listings of entities. We manually extracted the target entities from these pages to create a gold standard. Then, we ran the IE algorithms and measured their performance with respect to the gold standard.

Database Competitor. In SUSIE, the extraction algorithms are used to generate candidates for the input values for Web services. Query evaluation algorithms without this capability need to generate the input values for Web services by enumerating a domain. For example, while SUSIE will guess actors born in 1970 by searching for "actors 1970" on the Web, a non-IE-based query evaluation will guess actors born in 1970 by enumerating all available actors. To judge how many of the candidate actors would actually be actors born in 1970, we report the number of entities of the target type in the YAGO database that have the desired property. For example, for actors born in 1970, we report the proportion of actors in YAGO that are born in 1970 (among those actors that have a birth date). This is an estimator for the chance that a candidate generated by enumerating a domain will be a valid input value for the Web service.

Results. Figures 4, 5, and 6 and show the results. Every row contains the values averaged for 10 Web pages. #E is the average number of entities per page. SMA is the String Matching Algorithm, and SEA is the Structured Extraction algorithm. The column "DB" is the naive algorithm of regarding all instances of the target type in the YAGO database as candidates. The precision and recall of the IE algorithms are nearly always in the range between 30% and 75%. Only the precision on the birth year queries is disappointing, with values below 10% (Figure 5). This is because the Google queries returned lists of all actors, not just of the ones born in a certain year. Thus, the algorithms find far too many irrelevant entities in the pages. The SMA, with its slightly higher recall, suffers particularly for the precision. We record this as a case where the information extraction approach is less practical, because the Internet does not provide the lists of entities that the approach needs.

Award	#E	SMA		SEA		DB
		Prec	Rec	Prec	Rec	Prec
Franz Kafka	2	25 %	73 %	13 %	34 %	N/A
Golden Pen	9	36 %	33 %	29 %	56 %	N/A
Jerusalem	6	23 %	52 %	69 %	24 %	N/A
National Book	69	38 %	59 %	45 %	76 %	0.9 %
Nobel Prize	44	41 %	29 %	46 %	40 %	2.9 %
Phoenix	4	47 %	71 %	18 %	76 %	N/A
Prix Decembre	4	29 %	6 %	18 %	25 %	N/A
Prix Femina	21	31 %	13 %	32 %	32 %	0.6 %
Prix Goncourt	73	63 %	46 %	7 %	1 %	1.12%
Pulitzer	42	78 %	79 %	60 %	46 %	2.0 %
	27	43 %	44 %	34 %	35 %	1.5%

Table 4: IE Results for "Authors who won prize X"

Year	#E	SMA		SEA		DB
		Prec	Rec	Prec	Rec	Prec
1940	2	2 %	73 %	1 %	80 %	0.8 %
1945	1	2 %	96 %	1 %	100 %	1.0 %
1950	2	2 %	81 %	1 %	83 %	1.2 %
1955	2	6 %	39 %	3 %	56 %	1.2 %
1960	18	12 %	60 %	6 %	72 %	1.3 %
1965	8	17 %	72 %	14 %	71 %	1.5 %
1970	4	20 %	96 %	1 %	66 %	1.7 %
1975	2	8 %	91 %	1 %	67 %	1.6 %
1980	6	8 %	52 %	4 %	90 %	1.6 %
1985	2	8 %	56 %	0 %	43 %	1 %
	5	9 %	71 %	3 %	74 %	1.3 %

Table 5: IE results for "Actors born in year X"

Country	#E	SMA		SEA		DB
		Prec	Rec	Prec	Rec	Prec
Australia	15	37 %	82 %	51 %	66 %	11%
Canada	5	28 %	92 %	40 %	50 %	20%
England	46	46 %	85 %	71 %	74 %	0 %
France	153	48 %	42 %	50 %	64 %	2 %
Germany	45	50 %	57 %	51 %	99 %	2 %
Greece	26	38 %	58 %	2 %	14 %	0 %
Italy	138	29 %	54 %	42 %	59 %	0 %
Mexico	25	44 %	52 %	51 %	78 %	0 %
South Africa	12	29 %	76 %	29 %	63 %	0%
Spain	24	54 %	63 %	67 %	94 %	0 %
	47	38 %	65 %	46 %	63 %	3.5 %

Table 6: IE Results for "Actors of nationality X"

Discussion. A precision of 30% may not sound extraordinary. Yet, it has to be seen in comparison to the naive approach of sending all entities of the target type to the Web service. In general, the proportion of entities in the database that have the desired property is very low. The percentages for writer awards are already an overestimation, because they consider only those writers that did win an award, while many writers do not win any award at all in their life. So let us e.g. assume that 1% of the entities have the desired property. This means that an expected 100 calls would have to be sent to the Web service before finding one of them. This number of calls is already above the budget we are considering, meaning that the user would likely not get any response at all. An IE precision of 30%, in contrast, means that for every 3 queries that are sent to the Web service, only 2 are sent in vain. Likewise, a recall of 30% means that we can find one third of the entities that the user is potentially interested in – as opposed to none if the call budget is exhausted by enumerating a domain. Thus, even in the cases with lower precision, our approach allows answering queries that would be impossible to answer otherwise.

5.7.2 Real-world Queries

In this section, we evaluate our approach on real queries with real Web Services.

Service	Function
Music-Brainz	$getArtists_{mb}^{bfff}$ *(artist, id, born, died)* $getAlbums_{mb}^{bfff}$ *(album, id, artist, date)*
Abe-Books	$booksByTitle_{abe}^{bfffff}$ *(title, id, isbn, author, publisher)* $booksByIsbn_{abe}^{bfffff}$ *(isbn, title, id, author, publisher)* $booksByAuthor_{abe}^{bfffff}$ *(author, title, id, isbn, publisher)*
Library-Thing	$getAuthors_{lt}^{bfffff}$ *(x, born, place, country, prize)* $getBooks_{lt}^{bfffffff}$ *(title, author, prize, date)*

Table 7: Some of the functions integrated in SUSIE

Web Services and Queries. We integrated 40 functions exported by 7 Web service providers: *isbndb.org*, *librarything.com*, *Amazon*, *AbeBooks*, *api.internetvideoarchive.com*, *musicbrainz.org*, *lastfm.com*. Figure 7 shows the signatures of some of these functions. We selected a variety of query templates, which can be organized in the following classes (Figure 8): star queries with constants at the endpoints (Q_1-Q_2, Q_7), star queries with variables and constants at the endpoints (Q_3-Q_4, Q_8-Q_{10}), and chain queries with constants at the endpoints (Q_5-Q_6, Q_{11}). For every query template, we evaluate a set of similar queries by varying the constants. The queries were chosen such that they have different alternative ways of composing function instantiations. Usually, this leads to a high number of Web service calls.

No.	Query	
Q1	wonAward	(?person, p)
Q2	wonAward	(?person, p)
	isCitizenOf	(?person, c)
Q3	wonAward	(?person, p)
	wrote	(?person, ?book)
Q4	wonAward	(?person, p)
	isCitizenOf	(?person, c)
	wrote	(?person, ?book)
Q5	wonAward	(?person, ?prize)
	isTitled	(?prize, p)
	awardedInYear	(?prize, y)
Q6	wrote	(?person, ?book)
	wonAward	(?person, ?prize)
	isTitled	(?prize, p)
	awardedInYear	(?prize, y)
Q7	isFamousActor	(?person, True)
	isCitizenOf	(?person, c)

No.	Query	
Q8	isFamousActor	(?person, True)
	isCitizenOf	(?person, c)
	actedIn	(?person, ?movie)
Q9	wonAward	(?movie, p)
	actedIn	(?person, ?movie)
Q10	wonAward	(?movie, p)
	producedIn	(?movie, ?country)
	actedIn	(?person, ?movie)
Q11	sang	(?person, ?song)
	wonAward	(?person, ?prize)
	isTitled	(?prize, p)
	awardedInYear	(?prize, y)

Table 8: Query templates. p, c, y are given values.

Settings and Algorithms. We distinguish two settings. In the first setting we try to answer the query using only Web services. We compare 3 different approaches. The first approach ("TD") uses a naive top-down evaluation of the queries without inverse functions. This approach implements a Prolog-style backtracking strategy. The second approach uses ANGIE [32] for the query evaluation. ANGIE is a state-of-the-art system for top-down query evaluation with views with binding patterns. The third approach uses SUSIE, i.e., the approach uses both Web services and inverse functions. We used the SEA algorithm for IE.

In the second setting, we allow the approaches to make use of the YAGO knowledge base [35]. The first approach in this setting is a baseline solution, which simply issues the query to YAGO, and does not call any functions. The second approach is YAGO+TD, which is allowed to use both functions and the YAGO knowledge base. This means that it can retrieve answers from YAGO, it can retrieve answers from functions, and it can also use data from YAGO to call the functions. The third approach is YAGO+ANGIE, and the fourth is YAGO+SUSIE. Only SUSIE uses inverse functions and we use again the SEA algorithm for IE. For all the algorithms, we set the budget to 15 for the number of calls to one service and to 100 for the total number of calls. These constrains appear naturally in practice due to the response time overhead of the Web calls. Furthermore, Web service providers limit the number of calls per user. As performance metrics, we measured the total number of answers output by each algorithm.

Results. Figure 9 shows the results for the queries corresponding to the templates in Figure 8. We report the number of answers for all algorithms in both settings: With only functions and with both functions and YAGO. In both settings, all algorithms consume their entire budget of calls. Since all algorithms use the same number of calls, the total number of answers returned by each serves as comparison metric. For SUSIE we show the number of calls that were consumed by the algorithm for two moments in time: when the first answer was

output (#c1a) and when the final answer was output (#cFa). Subsequent calls used up the budget, but did not return answers.

In the first setting, where only functions can be used, only SUSIE delivers any answers at all. This is because algorithms without inverse functions have to compose existing functions to compute answers, which often consumes the entire budget before any answer is returned. It may also be just impossible to find such a composition. In the second setting, the algorithms can also use YAGO. YAGO already contains some answers to the queries (reported in the column "YAGO"). All algorithms first return these answers from YAGO, and then embark to call functions. We observe that, for all queries, SUSIE returns more answers than the two other algorithms, or at least an equal number. For instance, for the first query of the template Q_4, SUSIE outputs almost twice as many answers as ANGIE.

Q	Constants	Just Functions					YAGO	YAGO	YAGO	**YAGO**		
	YAGO	TD	ANGIE	**SUSIE**	#c1a	#cFa		+TD	+ANGIE	**+SUSIE**	#c1a	#cFa
Q1	Nobel Prize in Literature	0	0	**14**	3	55	103	103	103	**103**	0	0
	Golden Pen Award	0	0	**11**	4	16	0	0	0	**11**	4	16
	Franz Kafka Prize	0	0	**5**	4	8	0	0	0	**5**	4	8
	American Book Medal	0	0	**16**	3	18	0	0	0	**16**	3	18
	Jerusalem Prize	0	0	**11**	3	21	0	0	0	**11**	3	21
Q2	France, Nobel Prize Literature	0	0	**5**	2	9	6	6	6	**9**	0	8
	UK, Franz Kafka Prize	0	0	**1**	2	2	0	0	0	**1**	2	2
Q3	Nobel Prize Literature	0	0	**198**	43	100	234	235	453	**457**	0	94
	Golden Pen Award	0	0	**228**	18	87	0	0	0	**226**	6	99
	Franz Kafka Prize	0	0	**132**	19	97	0	0	0	**181**	5	92
	American Book Medal	0	0	**296**	19	97	0	0	0	**522**	3	111
	Jerusalem Prize	0	0	**220**	22	90	0	0	0	**233**	4	91
Q4	France, Nobel Prize Literature	0	0	**144**	11	89	2	61	74	**133**	0	107
	UK, Franz Kafka Prize	0	0	**79**	3	63	0	0	0	**70**	3	61
Q5	Nobel Prize Literature, 2004	0	0	**1**	2	2	0	0	0	**1**	2	2
	Golden Pen Award, 2006	0	0	**1**	2	2	0	0	0	**1**	2	2
	Franz Kafka Prize, 2006	0	0	**1**	2	2	0	0	0	**1**	2	2
	American Book Medal, 2004	0	0	**1**	2	2	0	0	0	**1**	2	2
	Jerusalem Prize, 1981	0	0	**1**	2	2	0	0	0	**1**	2	2
Q6	Nobel Prize in Literature 2004	0	0	**31**	3	51	0	0	0	**31**	3	51
	Golden Pen Award, 2006	0	0	**57**	3	52	0	0	0	**64**	3	51
	Franz Kafka Prize, 2006	0	0	**61**	3	59	0	0	0	**89**	2	59
	American Book Medal, 2004	0	0	**77**	3	75	0	0	0	**243**	2	85
	Jerusalem Prize 1981	0	0	**60**	3	71	0	0	0	**90**	2	69
Q7	United States Of America	0	0	**7**	5	20	0	0	0	**7**	5	20
	United Kingdom	0	0	**7**	3	26	0	0	0	**7**	3	26
Q8	United States Of America	0	0	**309**	5	40	0	0	0	**330**	5	40
	United Kingdom	0	0	**213**	3	52	0	0	0	**234**	3	66
Q9	Academy Award Best Picture	0	0	**85**	2	56	0	0	0	**187**	2	88
Q10	Academy Award Best Picture	0	0	**79**	2	16	0	0	0	**212**	2	94
Q11	Grammy Awards 2009	0	0	**110**	69	75	0	0	0	**377**	69	75

Table 9: Number of answers for the queries of the templates in Figure 8

If we compare SUSIE with YAGO to SUSIE using just functions, we remark that using YAGO improves the number of answers in most cases. In some cases SUSIE performs better without YAGO. This is because the query planner sometimes chooses to enumerate a domain from YAGO, even if the inverse function is present. In almost all cases, SUSIE improves over the answers that YAGO alone provides. Even for queries that have no answers in YAGO ($Q3$ - four, $Q6$ - four, $Q9$, $Q10$, $Q11$), SUSIE makes smart use of the database to double the number of answers with respect to the case where just functions are used.

5.8 Related Work

Query Answering. Most related to our setting is the problem of answering

queries using views with limited access patterns [33]. The approach of [33] rewrites the initial query into a set of queries to be executed over the given views. The authors show that for a conjunctive query over a global schema and a set of views over the same schema, determining whether there exists a conjunctive query plan over the views that is equivalent to the original query is NP-hard in the size of the query. This rewriting strategy assumes that the views are complete (i.e., contain all the tuples in their definition). This assumption is unrealistic in our setting with Web services, where sources may overlap or complement each other but are usually incomplete.

When sources are incomplete, one aims to find maximal contained rewritings of the initial query, in order to provide the maximal number of answers. [16] present algorithms for rewriting a query into a datalog program, requiring recursive datalog even if the initial query is non-recursive. Subsequent studies [28, 23, 12, 9, 8] proposed solutions for reducing the number of accesses. Notions of minimal rewritings have been proposed in [12, 15]. However, the goal remains the computation of maximal results. The guessing accesses are not eliminated nor do they have a special treatment since they relevant for this goal. The same problem was studied for different query languages: unions of conjunctive queries with negation [31], with additional function dependencies [11], or with integrity constraints [15]. The Magic Set algorithm [5] reduces the number of sub-queries in a bottom-up evaluation.

Our own ANGIE system [32] also answers queries using views as surrogates for Web services, and it imposes an upper bound on the number of function calls. The strategy is use the local data when possible to prioritize function calls that are more likely to deliver answers.

None of these approaches can cope with situations where the available functions have binding patterns that do not allow answering the query. For example, if there is only one function, $getSongs^{bf}(singer, song)$, and the user asks who sang "Hallelujah", then no query evaluation procedure, no optimization strategy, and no smart orchestration mechanism can deliver an answer to the query. This is because the function cannot be called without a singer. This limitation applies to all approaches listed above. What is needed in such cases are the inverse functions that SUSIE provides.

Deep Web Querying. A Deep Web page is a form, which requires certain values to be filled in, and which delivers results for these values. Thus, guessing the right values for the forms has similarities to guessing the right input values for Web services. Much work has addressed the probing, semantic categorization, or materialization of Deep Web forms. Google's "surfacing" technique [30] aims to materialize the Deep Web by determining the most promising input values for the forms. [37] aims to match the schema of two Deep Web forms. This approach finds instances for Deep Web attributes by querying the surface Web, and then validating the instances through the original Deep Web form. [22] estimates the domain and the size of the domain of Deep Web attributes by systematic probing. Other work [7, 29] has focused on probing form fields and reconstructing the "schema" of a single form, so that queries can place properly typed values into specific fields, avoiding unnecessary requests.

These works differ in three aspects from our setting. First, the approaches typically analyze the Deep Web form in an off-line fashion, where all necessary information is available before query time. This allows for approaches that

use training and learning. In our setting, in contrast, information has to be extracted and integrated on the fly, because it depends dynamically on the constants of the query. This restricts the set of applicable methods to techniques that work at query time. Second, work on the Deep Web typically aims to fill unary relations (e.g., finding all reasonable values for the field "author"). The population of unary relations can draw upon a large pool of techniques (such as Hearst patterns [21]). In our setting, in contrast, we have to find arguments for binary relations (e.g., the "author of the book 'Don Quixote"'). Thus, the techniques from the Deep Web do not directly transfer to our setting. Third, our setting requires the dynamic composition of Web service functions in order to answer user queries. The IE-based functions have to be integrated into this orchestration and called on the fly. The above work on the Deep Web does not address the prioritization of inverse functions in execution plans.

Information Extraction. Information extraction (IE) is concerned with extracting structured data from documents. IE methods suffer from the inherent imprecision of the extraction process. Usually, the extracted data is way too noisy to allow direct querying. SUSIE overcomes this limitation, by using IE solely for finding candidate entities of interest and feeding these as inputs into Web service calls. Named Entity Recognition (NER) approaches [38, 26, 14] aim to detect interesting entities in text documents. They can be used to generate candidates for SUSIE. The first approach discussed in this paper matches noun phrases against the names of entities that are registered in a knowledge base – a simple but effective technique that circumvents the noise in learning-based NER techniques. The second IE approach used in this paper is to extract structured data from Web tables [10, 17, 20]. For SUSIE, we have developed judiciously customized methods along these lines. These are not limited to lists and tables, but detect arbitrary repetitive structures that could contain candidates. Alternative IE methods such as Wrapper Induction [24], fact extraction [3, 6, 36], or entity extraction [14, 39] could be also considered, but they are not practical in our scenario as they require training data and, thus, human supervision.

Web service orchestration. There is plenty of prior work on Web service orchestration. The goal is to describe complex business processes that are carried out using Web service compositions. The standard for specifying such workflows is BPEL [1]. A suite of papers addressed problems ranging from system architecture [13] to optimization issues [34]. However, none of these works addresses the issue of service asymmetry.

5.9 Conclusion

This paper has introduced the problem of asymmetric Web services. We have shown that a considerable number of real-world Web services allow asking for only one argument of a relationship, but not for the other. We have proposed to use information extraction to guess bindings for the input variables and then validate these bindings by the Web service. Through this approach, a whole new class of queries has become tractable. We have shown that providing inverse functions alone is not enough. They also have to be prioritized accordingly. We have implemented our system, SUSIE, and showed the validity of our approach

on real data sets. We believe that the beauty of our approach lies in the fruitful symbiosis of information extraction and Web services, which each mitigate the weaknesses of the other.

Our current implementation uses naive information extraction algorithms that serve mainly as a proof of concept. Future work will explore new algorithms that could step in. We also aim to automize the discovery of new Web services and their integration into the system.

References

[1] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Eugene Agichtein et al. "Snowball: a prototype system for extracting relations from large text collections". In: *SIGMOD Records* (2001).

[4] Sören Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *Semantic Web* (2008).

[5] François Bancilhon et al. "Magic Sets and Other Strange Ways to Implement Logic Programs". In: *PODS*. 1986.

[6] Michele Banko et al. "Open Information Extraction from the Web". In: *IJCAI*. 2007.

[7] L. Barbosa et al. "Creating and exploring web form repositories". In: *SIGMOD*. 2010.

[8] M. Benedikt, P. Bourhis, and C. Ley. "Querying Schemas With Access Restrictions". In: *PVLDB* (2012).

[9] M. Benedikt, G. Gottlob, and P. Senellart. "Determining relevance of accesses at runtime". In: *PODS*. 2011.

[10] Michael J. Cafarella et al. "Uncovering the Relational Web". In: *WebDB*. 2008.

[11] Andrea Calì, Diego Calvanese, and Davide Martinenghi. "Dynamic Query Optimization under Access Limitations and Dependencies". In: *J. UCS* (2009).

[12] Andrea Calì and Davide Martinenghi. "Querying Data under Access Limitations". In: *ICDE*. 2008.

[13] S. Ceri, A. Bozzon, and M. Brambilla. "The Anatomy of a Multi-domain Search Infrastructure". In: *ICWE*. 2011.

[14] Hamish Cunningham and Donia Scott. "Software Architecture for Language Engineering". In: *Nat. Lang. Eng.* (2004).

[15] Alin Deutsch, Bertram Ludäscher, and Alan Nash. "Rewriting queries using views with access patterns under integrity constraints". In: *Theor. Comput. Sci.* (2007).

[16] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. "Recursive Query Plans for Data Integration". In: *J. Log. Program.* (2000).

[17] Hazem Elmeleegy, Jayant Madhavan, and Alon Y. Halevy. "Harvesting Relational Tables from Lists on the Web". In: *PVLDB* (2009).

[18] Ronald Fagin et al. "Clio: Schema Mapping Creation and Data Exchange". In: *Conceptual Modeling: Foundations and Applications*. 2009.

[19] Edward Fredkin. "Trie memory". In: *Commun. ACM* 9 (Sept. 1960).

[20] Wolfgang Gatterbauer et al. "Towards domain-independent IE from web tables". In: *WWW*. 2007.

[21] Marti A. Hearst. "Automatic acquisition of hyponyms from large text corpora". In: *ICCL*. Association for Computational Linguistics, 1992.

[22] Xin Jin, Nan Zhang, and Gautam Das. "Attribute domain discovery for hidden web databases". In: *SIGMOD*. 2011.

[23] Subbarao Kambhampati et al. "Optimizing Recursive Information Gathering Plans in EMERAC". In: *J. Intell. Inf. Syst.* (2004).

[24] Nicholas Kushmerick. "Wrapper induction for information extraction". PhD thesis. U. Washington, 1997.

[25] Chung T. Kwok and Daniel S. Weld. "Planning to Gather Information". In: *AAAI/IAAI, Vol. 1*. 1996.

[26] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *ICML*. Morgan Kaufmann Publishers Inc., 2001.

[27] Chen Li. "Computing complete answers to queries in the presence of limited access patterns". In: *VLDB J.* (2003).

[28] Chen Li and Edward Y. Chang. "Query Planning with Limited Source Capabilities". In: *ICDE*. 2000.

[29] W. Liu, X. Meng, and W. Meng. "ViDE: A Vision-Based Approach for Deep Web Data Extraction". In: *IEEE Trans. Knowl. Data Eng.* (2010).

[30] J. Madhavan et al. "Google's Deep Web crawl". In: *PVLDB* (2008).

[31] Alan Nash and Bertram Ludäscher. "Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns". In: *EDBT*. 2004.

[32] N. Preda et al. "Active Knowledge : Dynamically Enriching RDF Knowledge Bases by Web Services. (ANGIE)". In: *SIGMOD*. 2010.

[33] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. "Answering Queries Using Templates with Binding Patterns". In: *PODS*. 1995.

[34] Utkarsh Srivastava et al. "Query Optimization over Web Services". In: *VLDB*. 2006.

[35] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Core of Semantic Knowledge". In: *WWW*. 2007.

[36] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. "SOFIE: A Self-Organizing Framework for Information Extraction". In: *WWW*. 2009.

[37] Wensheng Wu, AnHai Doan, and Clement T. Yu. "WebIQ: Learning from the Web to Match Deep-Web Query Interfaces". In: *ICDE*. 2006.

[38] Jun Zhu et al. "2D Conditional Random Fields for Web information extraction". In: *ICML*. Bonn, Germany: ACM, 2005.

[39] Jun Zhu et al. “Simultaneous record detection and attribute labeling in web data extraction”. In: *KDD*. 2006.

Chapter 6

Ontology Alignment with PARIS

6.1 Overview

One of the main challenges that the Semantic Web faces is the integration of a growing number of independently designed ontologies. In this work, we present PARIS, an approach for the automatic alignment of ontologies. PARIS aligns not only instances, but also relations and classes. Alignments at the instance level cross-fertilize with alignments at the schema level. Thereby, our system provides a truly holistic solution to the problem of ontology alignment. The heart of the approach is probabilistic, i.e., we measure degrees of matchings based on probability estimates. This allows PARIS to run without any parameter tuning. We demonstrate the efficiency of the algorithm and its precision through extensive experiments. In particular, we obtain a precision of around 90 % in experiments with some of the world's largest ontologies. This chapter is based on

6.2 Introduction

Motivation. An ontology is a formal collection of world knowledge. In this paper, we use the word *ontology* in a very general sense, to mean both the schema (classes and relations), and the instances with their assertions. In recent years, the success of Wikipedia and algorithmic advances in information extraction have facilitated the automated construction of large general-purpose ontologies. Notable endeavors of this kind include DBpedia [2], KnowItAll [10], WikiTaxonomy [27], and YAGO [31], as well as commercial services such as freebase.com, trueknowledge.com, and wolframalpha.com. These ontologies are accompanied by a growing number of knowledge bases[1] in a wide variety of

[1] http://www.w3.org/wiki/DataSetRDFDumps

domains including: music[2], movies[3], geographical data[4], publications[5], medical and biological data[6], or government data[7].

Many of these ontologies contain complementing data. For instance, a general ontology may know who discovered a certain enzyme, whereas a biological database may know its function and properties. However, since the ontologies generally use different terms (identifiers) for an entity, their information cannot be easily brought together. In this respect, the ontologies by themselves can be seen as isolated islands of knowledge. The goal of the Semantic Web vision is to interlink them, thereby creating one large body of universal ontological knowledge [5, 6]. This goal may be seen as a much scaled-up version of record linking, with challenges coming from different dimensions:

- unlike in record linkage, both instances and schemas should be reconciled;
- the semantics of the ontologies have to be respected;
- the ontologies are typically quite large and complex. Moreover, we are interested in performing the alignment in a fully automatic manner, and avoid tedious tuning or parameter settings.

A number of recent research have investigated this problem. There have been many works on entity resolution, i.e., on what is traditionally known as the "A-Box" [4, 1, 12, 26, 29, 28, 22, 17, 18]. In another direction, much research has focused on schema alignment, i.e., on the so-called "T-Box" [14, 21, 3, 20, 34]. However, in recent years, the landscape of ontologies has changed dramatically. Today's ontologies often contain both a rich schema and, at the same time, a huge number of instances, with dozens of millions of assertions about them. To fully harvest the mine of knowledge they provide, their alignment has to be built on cross-fertilizing the alignments of both instances and schemas.

In this paper, we propose a new, holistic algorithm for aligning ontologies. Our approach links not just related entity or relationship instances, but also related classes and relations, thereby capturing the fruitful interplay between schema and instance matching. Our final aim is to discover and link identical entities automatically across ontologies on a large scale, thus allowing ontologies to truly complement each other.

Contribution. The contribution of the present paper is three-fold:

1. We present PARIS[8], a probabilistic algorithm for aligning instances, classes, and relations simultaneously across ontologies.
2. We show how this algorithm can be implemented efficiently and that it does not require any tuning
3. We prove the validity of our approach through experiments on real-world ontologies.

[2] http://musicbrainz.org/
[3] http://www.imdb.com/
[4] http://www.geonames.org/
[5] http://www.informatik.uni-trier.de/~ley/db
[6] http://www.uniprot.org/
[7] http://www.govtrack.us/, http://source.data.gov.uk/data/
[8] Probabilistic Alignment of Relations, Instances, and Schema

The paper is organized as follows. Section 6.3 provides an overview of related work. We then introduce some preliminaries in Section 6.4. Section 6.5 describes our probabilistic algorithm and Section 6.6 its implementation. Section 6.7 discusses experiments. To ease the reading, some technical discussions are postponed to the final section.

6.3 Related Work

Overview. The problem of ontology matching has its roots in the problem of identifying duplicate entities, which is also known as record linkage, duplicate detection, or co-reference resolution. This problem has been extensively studied in both database and natural language processing areas [7, 9]. These approaches are less applicable in the context of ontologies for two reasons. First, they do not consider the formal semantics that ontologies have (such as the *subclassOf* taxonomy). Second, they focus on the alignment of instances and do not deal with the alignment of relations and classes.

There are a number of surveys and analyses that shed light on the problem of record linking in ontologies. Halpin et al. [15] provide a good overview of the problem in general. They also study difficulties of existing *sameAs*-links. These links are further analyzed by Ding et al. [8]. Glaser, Jaffri, and Millard [13] propose a framework for the management of co-reference in the Semantic Web. Hu et al. [19] provide a study on how matches look in general.

Schema Alignment. Traditional approaches to ontology matching have focused mostly either on aligning the classes (the "T-Box") or on matching instances (the "A-Box"). The approaches that align the classes are manifold, using techniques such as sense clustering [14], lexical and structural characteristics [21], or composite approaches [3]. Unlike PARIS, these approaches can only align classes and do not consider the alignment of relations and instances.

Most similar to our approach in this field are [20] and [34], which derive class similarity from the similarities of the instances. Both approaches consider only the equivalence of classes and do not compute subclasses, as does PARIS. Furthermore, neither can align relations or instances.

Instance Matching. There are numerous approaches to match instances of one ontology to instances of another ontology. Ferrara, Lorusso, and Montanelli [12] introduce this problem from a philosophical point of view. Different techniques are being used, such as exploiting the terminological structure [26], logical deduction [29], declarative languages [1], relational clustering [4], or a combination of logical and numerical methods [28]. The Sig.ma engine [22] uses heuristics to match instances. Perhaps closest to our approach is [17], which introduces the concept of functionality. Different from their approach, PARIS does not require an additional smoothening factor.

The silk framework [33] allows specifying manual mapping rules. The ObjectCoref approach by Hu, Chen, and Qu [18] allows learning a mapping between the instances from training data. With PARIS, we aim at an approach that uses neither manual input nor training data. We compare some of the results of ObjectCoref to that of PARIS on the datasets of the ontology alignment evaluation

initiative [11] in Section 6.7. Hogan [16] matches instances and proposes to use these instances to compute the similarity between classes, but provides no experiments. Thus, none of these approaches can align classes and relations like PARIS.

Holistic Approaches. Only very few approaches address the cause of aligning both schema and instances: the RiMOM [23] and iliads [32] systems. Both of these have only been tested on small ontologies. The RiMOM system can align classes, but it cannot find *subclassOf* relationships. Furthermore, the approach provides a bundle of heuristics and strategies to choose from, while PARIS is monolithic. None of the ontologies the iliads system has been tested on contained full-fledged instances with properties. In contrast, PARIS is shown to perform well even on large-scale real-world ontologies with millions of instances.

6.4 Preliminaries

In this section, we recall the notions of ontology and of equivalence. Finally, we introduce the notion of functionality as one of the key concepts for ontology alignment.

Ontologies. We are concerned with ontologies available in the Resource Description Framework Schema (RDFS [35]), the W3C standard for knowledge representation. An RDFS ontology builds on *resources*. A resource is an identifier for a real-world object, such as a city, a person, or a university, but also the concept of mathematics. For example, *London* is a resource that represents the city of London. A *literal* is a string, date or number. A *property* (or *relation*) is a binary predicate that holds between two resources or between a resource and a literal. For example, the property *isLocatedIn* holds between the resources *London* and *UK*. In the RDFS model, it is assumed that there exists a fixed global set $\mathcal{R}$ of resources, a fixed global set $\mathcal{L}$ of literals, and a fixed global set $\mathcal{P}$ of properties. Each resource is described by a URI. An RDFS *ontology* can be seen as a set of triples $O \subset \mathcal{R} \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{L})$, called *statements*. In the following, we assume given an ontology O. To say that $\langle x, r, y\rangle \in O$, we will write $r(x, y)$ and we call x and y the *arguments* of r. Intuitively, such a statement means that the relation r holds between the entities x and y. We say that x, y is a *pair* of r. A relation r^{-1} is called the *inverse* of a relation r if $\forall x, y : r(x, y) \Leftrightarrow r^{-1}(y, x)$. We assume that the ontology contains all inverse relations and their corresponding statements. Note that this results in allowing the first argument of a statement to be a literal, a minor digression from the standard.

An RDFS ontology distinguishes between classes and instances. A class is a resource that represents a set of objects, such as, e.g., the class of all singers, the class of all cities or the class of all books. A resource that is a member of a class is called an *instance* of that class. We assume that the ontology partitions the resources into classes and instances.[9] The *rdf:type* relation connects an instance to a class. For example, we can say that the resource *Elvis* is a member of the class of singers: *rdf:type(Elvis, singer)*.

[9] RDFS allows classes to be instances of other classes, but in practice, this case is rare.

A more specific class c can be specified as a *subclass* of a more general class d using the statement *rdfs:subclassOf(c,d)*. This means that, by inference, all instances of c are also instances of d. Likewise, a relation r can be made a sub-relation of a relation s by the statement *rdfs:subpropertyOf(r,s)*. This means that, by inference again, $\forall x, y : r(x, y) \Rightarrow s(x, y)$. We assume that all such inferences have been established and that the ontologies are available in their *deductive closure*, i.e., all statements implied by the subclass and sub-property statements have been added to the ontology.

Equivalence. In RDFS, the sets $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{L}$ are global. That means that some resources, literals, and relations may be *identical* across different ontologies. For example, two ontologies may contain the resource *London*, therefore share that resource. (In practice, *London* is a URI, which makes it easy for two ontologies to use exactly the same identifier.) The semantics of RDFS enforces that these two occurrences of the identifier refer to the same real-world object (the city of London). The same applies to relations or literals that are shared across ontologies. Conversely, two different resources can refer to the same real-world object. For example, *London* and *Londres* can both refer to the city of London. Such resources are called *equivalent*. We write $Londres \equiv London$.

The same observation applies not just to instances, but also to classes and relations. Two ontologies can talk about an identical class or relation. They can also use different resources, but refer to the very same real-world concepts. For example, one ontology can use the relation *wasBornIn* whereas another ontology can use the relation *birthPlace*. An important goal of our approach is to find out that $wasBornIn \equiv birthPlace$.

In this paper, we make the following assumption: *a given ontology does not contain equivalent resources.* That is, if an ontology contains two instances x and x', then we assume $x \not\equiv x'$. We assume the same for relations and classes. This is a reasonable assumption, because most ontologies are either manually designed [24, 25], or generated from a database (such as the datasets mentioned in the introduction), or designed with avoiding equivalent resources in mind [31]. If the ontology does contain equivalent resources, then our approach will still work. It will just not discover the equivalent resources within one ontology. Note that, under this assumption, there can never be a chain of equivalent entities. Therefore, we do not have to take into account transitivity of equivalence.

Functions. A relation r is a *function* if, for a given first argument, there is only one second argument. For example, the relation *wasBornIn* is a function, because one person is born in exactly one place. A relation is an *inverse function* if its inverse is a function. If r is a function and if $r(x, y)$ in one ontology and $r(x, y')$ in another ontology, then y and y' must be equivalent. In the example: If a person is born in both *Londres* and *London*, then $Londres \equiv London$. The same observation holds for two first arguments of inverse functions.

As we shall see, functions play an essential role in deriving alignments between ontologies. Nevertheless, it turns out that the precise notion of function is too strict for our setting. This is due to two reasons:

- First, a relation r ceases to be a function as soon as there is one x with y and y' such that $r(x, y)$ and $r(x, y')$. This means that just one erroneous fact can make a relation r a non-function. Since real-world ontologies

usually contain erroneous facts, the strict notion of function is not well-suited.

- Second, even if a relation is not a function, it may contribute evidence that two entities are the same. For example, the relation *livesIn* is not a function, because some people may live in several places. However, a wide majority of people live in one place, or in very few places. So, if most people who live in *London* also live in *Londres*, this provides a strong evidence for the unification of *London* and *Londres*.

Thus, to derive alignments, we want to deal with "quasi-functions". This motivates introducing the concept of *functionality*, as in [17]. The *local functionality* of a relation r for a first argument x is defined as:

$$fun(r,x) = \frac{1}{\#y : r(x,y)} \tag{6.1}$$

where we write "$\#y : \varphi(y)$" to mean "$|\{y \mid \varphi(y)\}|$". Consider for example the relationship $isCitizenOf$. For most first arguments, the functionality will be 1, because most people are citizens of exactly one country. However, for people who have multiple nationalities, the functionality may be $\frac{1}{2}$ or even smaller. The *local inverse functionality* is defined analogously as $fun^{-1}(r,x) = fun(r^{-1},x)$. Deviating from [17], we define the *global functionality* of a relation r as the harmonic mean of the local functionalities, which boils down to

$$fun(r) = \frac{\#x : \exists y : r(x,y)}{\#x,y : r(x,y)} \tag{6.2}$$

We discuss design alternatives for this definition and the rationale of our choice in Section 6.10.1. The *global inverse functionality* is defined analogously as $fun^{-1}(r) = fun(r^{-1})$.

6.5 Probabilistic Model

6.5.1 Equivalence of Instances

We want to model the probability $Pr(x \equiv x')$ that one instance x in one ontology is equivalent to another instance x' in another ontology. Let us assume that both ontologies share a relation r. Following our argument in Section 6.4, we want the probability $Pr(x \equiv x')$ to be large if r is highly inverse functional, and if there are $y \equiv y'$ with $r(x,y)$, $r(x',y')$ (if, say, x and x' share an e-mail address). This can be written pseudo-formally as:

$$\exists r,y,y' : r(x,y) \wedge r(x',y') \wedge y{\equiv}y' \wedge fun^{-1}(r) \text{ is high } \implies x{\equiv}x' \tag{6.3}$$

Using the formalization described in Section 6.10.2, we transform this logical rule into a probability estimation for $x \equiv x'$ as follows:

$$Pr_1(x \equiv x') := 1 - \prod_{\substack{r(x,y)\\ r(x',y')}} \left(1 - fun^{-1}(r) \times Pr(y \equiv y')\right) \tag{6.4}$$

In other words, as soon as there is one relation r with $fun^{-1}(r) = 1$ and with $r(x, y)$, $r(x', y')$, and $Pr(y \equiv y') = 1$, it follows that $Pr_1(x \equiv x') = 1$. We discuss a design alternative in Section 6.10.3.

Note that the probability of $x \equiv x'$ depends recursively on the probabilities of other equivalences. These other equivalences may hold either between instances or between literals. We discuss the probability of equivalence between two literals in Section 6.6. Obviously, we set $Pr(x \equiv x) := 1$ for all literals and instances x.

Equation (6.4) considers only positive evidence for an equality. To consider also evidence against an equality, we can use the following modification. We want the probability $Pr(x \equiv x')$ to be small, if there is a highly functional relation r with $r(x, y)$ and if $y \not\equiv y'$ for all y' with $r(x', y')$. Pseudo-formally, this can be written as

$$\exists r, y : r(x, y) \wedge (\forall y' : r(x', y') \Rightarrow y \not\equiv y') \wedge fun(r) \text{ is high } \implies x \not\equiv x. \tag{6.5}$$

This can be modeled as

$$Pr_2(x \equiv x') := \prod_{r(x,y)} \left(1 - fun(r) \prod_{r(x',y')} (1 - Pr(y \equiv y')) \right) \tag{6.6}$$

As soon as there is one relation r with $fun(r) = 1$ and with $r(x, y)$, $r(x', y')$, and $Pr(y \equiv y') = 0$, it follows that $Pr_2(x \equiv x') = 0$. We combine these two desiderata by multiplying the two probability estimates:

$$Pr_3(x \equiv x') := Pr_1(x \equiv x') \times Pr_2(x \equiv x') \tag{6.7}$$

In the experiments, we found that Equation (6.4) suffices in practice. However, we discuss scenarios where Equation (6.7) can be useful in Section 6.7.

6.5.2 Subrelations

The formulas we have just established estimate the equivalence between two entities that reside in two different ontologies, if there is a relation r that is common to the ontologies. It is also a goal to discover whether a relation r of one ontology is equivalent to a relation r' of another ontology. More generally, we would like to find out whether r is a sub-relation of r', written $r \subseteq r'$.

Intuitively, the probability $Pr(r \subseteq r')$ is proportional to the number of pairs in r that are also pairs in r':

$$Pr(r \subseteq r') := \frac{\#x, y : r(x, y) \wedge r'(x, y)}{\#x, y : r(x, y)} \tag{6.8}$$

The numerator should take into account the resources that have already been matched across the ontologies. Therefore, the numerator is more appropriately phrased as:

$$\#x, y : r(x, y) \wedge (\exists x', y' : x \equiv x' \wedge y \equiv y' \wedge r'(x', y')) \tag{6.9}$$

Using again our formalization from Section 6.10.2, this can be modeled as:

$$\sum_{r(x,y)} \left(1 - \prod_{r'(x',y')} (1 - (Pr(x \equiv x') \times Pr(y \equiv y'))) \right) \tag{6.10}$$

In the denominator, we want to normalize by the number of pairs in r that have a counterpart in the other ontology. This is

$$\sum_{r(x,y)} \left(1 - \prod_{x',y'} (1 - (Pr(x \equiv x') \times Pr(y \equiv y'))) \right) \tag{6.11}$$

Thus, we estimate the final probability $Pr(r \subseteq r')$ as:

$$\frac{\sum_{r(x,y)} \left(1 - \prod_{r'(x',y')} (1 - (Pr(x \equiv x') \times Pr(y \equiv y'))) \right)}{\sum_{r(x,y)} \left(1 - \prod_{x',y'} (1 - P(x \equiv x') \times Pr(y \equiv y')) \right)} \tag{6.12}$$

This probability depends on the probability that two instances (or literals) are equivalent.

One might be tempted to set $Pr(r \subseteq r) := 1$ for all relations r. However, in practice, we observe cases where the first ontology uses r where the second ontology omits it. Therefore, we compute $Pr(r \subseteq r)$ as a contingent quantity.

We are now in a position to generalize Equation (6.4) to the case where the two ontologies do not share a common relation. For this, we need to replace every occurrence of $r(x', y')$ by $r'(x', y')$ and factor in the probabilities that $r' \subseteq r$ or $r \subseteq r'$. This gives the following value to be assigned to $\Pr(x \equiv x')$:

$$\begin{aligned} 1 - \prod_{\substack{r(x,y) \\ r'(x',y')}} & \left(1 - Pr(r' \subseteq r) \times fun^{-1}(r) \times Pr(y \equiv y')\right) \\ & \times \left(1 - Pr(r \subseteq r') \times fun^{-1}(r') \times Pr(y \equiv y')\right) \end{aligned} \tag{6.13}$$

If we want to consider also negative evidence as in Equation (6.7), we get for $Pr(x \equiv x')$:

$$\begin{aligned} \Big(1 - \prod_{\substack{r(x,y) \\ r'(x',y')}} & \left(1 - P(r' \subseteq r) \times fun^{-1}(r) \times Pr(y \equiv y')\right) \\ & \times \left(1 - Pr(r \subseteq r') \times fun^{-1}(r') \times Pr(y \equiv y')\right) \Big) \\ & \times \prod_{\substack{r(x,y) \\ r'}} \Big(1 - fun(r) \times Pr(r' \subseteq r) \times \prod_{r'(x',y')} (1 - Pr(x \equiv x'))\Big) \\ & \times \Big(1 - fun(r') \times Pr(r \subseteq r') \times \prod_{r'(x',y')} (1 - Pr(x \equiv x'))\Big) \end{aligned} \tag{6.14}$$

This formula looks asymmetric, because it considers only $Pr(r' \subseteq r)$ and $fun(r)$ one the one hand, and $Pr(r \subseteq r')$ and $fun(r')$ on the other hand (and not, for instance, $Pr(r' \subseteq r)$ together with $fun(r')$). Yet, it is not asymmetric, because each instantiation of r' will at some time also appear as an instantiation of r. It is justified to consider $Pr(r' \subseteq r)$, because a large $Pr(r' \subseteq r)$ implies that $r'(x, y) \Rightarrow r(x, y)$. This means that a large $Pr(r' \subseteq r)$ implies that $fun(r) < fun(r')$ and $fun^{-1}(r) < fun^{-1}(r')$.

If there is no x', y' with $r'(x', y')$, we set as usual the last factor of the formula to one, $\prod_{r'(x',y')}(1 - Pr(x \equiv x'))) := 1$. This decreases $Pr(x \equiv x')$ in case one instance has relations that the other one does not have.

To each instance from the first ontology, our algorithm assigns multiple equivalent instances from the second ontology, each with a probability score. For each instance from the first ontology, we call the instance from the second ontology with the maximum score the *maximal assignment*. If there are multiple instances with the maximum score, we break ties arbitrarily, so that every instance has at most one maximal assignment.

6.5.3 Subclasses

A class corresponds to a set of entities. One could be tempted to treat classes just like instances and compute their equivalence. However, the class structure of one ontology may be more fine-grained than the class structure of the other ontology. Therefore, we aim to find out not whether one class c of one ontology is equivalent to another class c' of another ontology, but whether c is a subclass of c', $c \subseteq c'$. Intuitively, the probability $Pr(c \subseteq c')$ shall be proportional to the number of instances of c that are also instances of c':

$$Pr(c \subseteq c') = \frac{\#\, c \cap c'}{\#c} \tag{6.15}$$

Again, we estimate the expected number of instances that are in both classes as

$$\mathbb{E}(\#\, c \cap c') = \sum_{x:type(x,c)} \left(1 - \prod_{y:type(y,d)} (1 - P(x \equiv y))\right) \tag{6.16}$$

We divide this expected number by the total number of instances of c:

$$Pr(c \subseteq c') = \frac{\sum_{x:type(x,c)} \left(1 - \prod_{y:type(y,d)} (1 - P(x \equiv y))\right)}{\#x : type(x, c)} \tag{6.17}$$

The fact that two resources are instances of the same class can reinforce our belief that the two resources are equivalent. Hence, it seems tempting to feed the subclass-relationship back into Equation (6.13). However, in practice, we found that the class information is of less use for the equivalence of instances. This may be because of different granularities in the class hierarchies. It might also be because some ontologies use classes to express certain properties (*MaleSingers*), whereas others use relations for the same purpose ($gender = male$). Therefore, we compute the class equivalences only *after* the instance equivalences have been computed.

6.6 Implementation

6.6.1 Iteration

Our algorithm takes as input two ontologies. As already mentioned, we assume that a single ontology does not contain duplicate (equivalent) entities. This corresponds to some form of a domain-restricted unique name assumption. Therefore, our algorithm considers only equivalence between entities from different ontologies.

Strictly speaking, the functionality of a relation (Equation (6.2)) depends recursively on the equivalence of instances. If, e.g., every citizen lives in two countries, then the functionality of *livesIn* is $\frac{1}{2}$. If our algorithm unifies the two countries, then the functionality of *livesIn* jumps to 1. However, since we assume that there are no equivalent entities within one ontology, we compute the functionalities of the relations within each ontology upfront.

We implemented a fixpoint computation for Equations (6.12) and (6.13). First, we compute the probabilities of equivalences of instances. Then, we compute the probabilities for sub-relationships. These two steps are iterated until convergence. In a last step, the equivalences between classes are computed by Equation (6.17) from the final assignment. To bootstrap the algorithm in the very first step, we set $Pr(r \subseteq r') = \theta$ for all pairs of relations r, r' of different ontologies. We chose $\theta = 0.10$. The second round uses the computed values for $Pr(r \subseteq r')$ and no longer θ.

We have not yet succeeded in proving a theoretical condition under which the iteration of Equations (6.12) and (6.13) reaches a fixpoint. In practice, we iterate until the entity pairs under the maximal assignments change no more (which is what we call convergence). In our experiments, this state was always reached after a few iterations. We note that one could always enforce convergence of such iterations by introducing a progressively increasing dampening factor.

Our model changes the probabilities of two resources being equal – but never the probability that a certain statement holds. All statements in both ontologies remain valid. This is possible because an RDFS ontology cannot be made inconsistent by equating resources, but this would not be the case any more for richer ontology languages.

6.6.2 Optimization

The equivalence of instances (Equation (6.13)) can be computed in different ways. In the most naïve setting, the equivalence is computed for each pair of instances. This would result in a runtime of $O(n^2m)$, where n is the number of instances and m is the average number of statements in which an instance occurs (a typical value for m is 20). This implementation took weeks to run one iteration. We overcame this difficulty as follows.

First, we optimize the computation of Equation (6.13). For each instance x in the first ontology, we traverse all statements $r(x, y)$ in which this instance appears as first argument. (Remember that we assume that the ontology contains all inverse statements as well.) For each statement $r(x, y)$, we consider the second argument y, and all instances y' that the second argument is known to be equal to ($\{y' : Pr(y \equiv y') > 0\}$). For each of these equivalent instances y', we consider again all statements $r(x', y')$ and update the equality of x and x'. This results in a runtime of $O(nm^2e)$, where e is the average number of equivalent instances per instance (typically around 10). Equations (6.12) and (6.17) are optimized in a similar fashion.

Generally speaking, our model distinguishes *true* equivalences ($Pr(x \equiv x') > 0$) from *false* equivalences ($Pr(x \equiv x') = 0$) and *unknown* equivalences ($Pr(x \equiv x')$ not yet computed). Unknown quantities are simply omitted in the sums and products of the equations. Interestingly, most equations contain a probability $Pr(x \equiv x')$ only in the form $\prod(1 - P(x \equiv x'))$. This means that the formula

will evaluate to the same value if $Pr(x \equiv x')$ is unknown or if $Pr(x \equiv x') = 0$. Therefore, our algorithm does not need to store equivalences of value 0 at all.

Our implementation thresholds the probabilities and assumes every value below θ to be zero. This greatly reduces the number of equivalences that the algorithm needs to store. Furthermore, we limit the number of pairs that are evaluated in Equations (6.12) and (6.17) to $10,000$. For each computation, our algorithm considers only the equalities of the previous maximal assignment and ignores all other equalities. This reduces the runtime by an order of magnitude without affecting much the relation inclusion assessment.

We stress that all these optimizations have for purpose to decrease the running time of the algorithm *without significantly affecting the outcome of the computation*. We have validated in our experiments that it is indeed the case.

Our implementation is in Java, using the Java Tools developed for [30] and Berkeley DB. We used the Jena framework to load and convert the ontologies. The algorithm turns out to be heavily IO-bound. Therefore, we used a solid-state drive (SSD) with high read bandwidth to store the ontologies. This brought the computation time down from the order of days to the order of hours on very large ontologies. We considered parallelizing the algorithm and running it on a cluster, but it turned out to be unnecessary.

6.6.3 Literal Equivalence

The probability that two literals are equal is known a priori and will not change. Therefore, such probabilities can be set upfront (*clamped*), for example as follows:

- The probability that two numeric values of the same dimension are equal can be a function of their proportional difference.
- The probability that two strings are equal can be inverse proportional to their edit distance.
- For other identifiers (social security numbers, etc.), the probability of equivalence can be a function that is robust to common misspellings. The checksum computations that are often defined for such identifiers can give a hint as to which misspellings are common.
- By default, the probability of two different literals being equal should be 0.

These functions can be designed depending on the application or on the specific ontologies. They can, e.g., take into account unit conversions (e.g., between Kelvin and Celcius). They could also perform datatype conversions (e.g., between *xsd:string* and *xsd:anyURI*) if necessary. The probabilities can then be plugged into Equation (6.13).

For our implementation, we chose a particularly simple equality function. We normalize numeric values by removing all data type or dimension information. Then we set the probability $Pr(x \equiv y)$ to 1 if x and y are identical literals, to 0 otherwise. The goal of this work is to show that even with such a simple, domain-agnostic, similarity comparison between literals, our probabilistic model is able to align ontologies with high precision; obviously, precision could be raised even higher by implementing more elaborate literal similarity functions.

6.6.4 Parameters

Our implementation uses the following parameters:

1. The initial value θ for the equivalence of relations in the very first step of the algorithm. We show in the experiments that the choice of θ does not affect the results.
2. Similarity functions for literals. These are application-dependent. However, we show that even with the simple identity function, the algorithm performs well.

Therefore, we believe we can claim that our model has no dataset-dependent tuning parameters. Our algorithm can be (and in fact, was) run on all datasets without any dataset specific settings. This contrasts PARIS with other algorithms, which are often heavily dependent on parameters that have to be tuned for each particular application or dataset. Traditional schema alignment algorithms, for example, usually use heuristics on the names of classes and relations, whose tuning requires expertise (e.g., [23]). A major goal of the present work was to base the algorithm on probabilities and make it as independent as possible from the tuning of parameters. We are happy to report that this works beautifully.

In order to improve results further, one can use smarter similarity functions, as discussed in Section 6.6.3.

6.7 Experiments

6.7.1 Setup

All experiments were run on a quad-core PC with 12 GB of RAM, running a 64bit version of Linux; all data was stored on a fast solid-state drive (SSD), with a peak random access bandwidth of approximately 50 MB/s (to be compared with a typical random access bandwidth of 1 MB/s for a magnetic hard drive).

Our experiments always compute relation, class, and instance equivalences between two given ontologies. Our algorithm was run until convergence (i.e., until less than 1 % of the entities changed their maximal assignment). We evaluate the instance equalities by comparing the computed final maximal assignment to a gold standard, using the standard metrics of precision, recall, and F-measure. For instances, we considered only the assignment with the maximal score. For relation assignments, we performed a manual evaluation. Since PARIS computes sub-relations, we evaluated the assignments in each direction. Class alignments were also evaluated manually. For all evaluations, we ignored the probability score that PARIS assigned, except when noted.

6.7.2 Benchmark Test

To be comparable to [18, 23, 26, 29], we report results on the benchmark provided by the 2010 edition of the ontology alignment evaluation initiative (OAEI) [11]. We ran experiments on two datasets, each of which consists of two ontologies.[10] For each dataset, the OAEI provides a gold standard list of

[10] We could not run on the third dataset, because it violates our assumption of non-equivalence within one ontology.

instances of the first ontology that are equivalent to instances of the second ontology. The relations and classes are identical in the first and second ontology. To make the task more challenging for PARIS, we artificially renamed the relations and classes in the first ontology, so that the sets of instances, classes, and relations used in the first ontology are disjoint from the ones used in the second ontology.

		Instances				Classes				Relations			
Dataset	**System**	Gold	Prec	Rec	F	Gold	Prec	Rec	F	Gold	Prec	Rec	F
Person	PARIS	500	100%	100%	100%	4	100%	100%	100%	20	100%	100%	100%
	ObjCoref		100%	100%	100%		-	-	-		-	-	-
Rest.	PARIS	112	95%	88%	91%	4	100%	100%	100%	12	100%	66%	88%
	ObjCoref		N/A	N/A	90%		-	-	-		-	-	-

Table 1: Results (precision, recall, F-measure) of instance, class, and relation alignment on OAEI datasets, compared with ObjectCoref [18]. The "Gold" columns indicate the number of equivalences in the gold standard.

For the person dataset, PARIS converged after just 2 iterations and 2 minutes. For the restaurants, PARIS took 3 iterations and 6 seconds. Table 1 shows our results.[11] We achieve near-perfect precision and recall, with the exception of recall in the second dataset. As reported in [18], all other approaches [23, 26, 29] remain below 80 % of F-measure for the second dataset, while only ObjectCoref [18] achieves an F-measure of 90 %. We achieve an F-measure of 91 %. We are very satisfied with this result, because unlike ObjectCoref, PARIS does not require any training data. It should be further noted that, unlike all other approaches, PARIS did not even know that the relations and classes were identical, but discovered the class and relation equivalences by herself in addition to the instance equivalences.

6.7.3 Design Alternatives

To measure the influence of θ on our algorithm, we ran PARIS with $\theta = 0.001, 0.01, 0.05, 0.1, 0.2$ on the restaurant dataset. A larger θ causes larger probability scores in the first iteration. However, the sub-relationship scores turn out to be the same, no matter what value θ had. Therefore, the final probability scores are the same, independently of θ. In a second experiment, we allowed the algorithm to take into account all probabilities from the previous iteration (and not just those of the maximal assignment). This changed the results only marginally (by one correctly matched entity), because the first iteration already has a very good precision. In a third experiment, we allowed the algorithm to take into account negative evidence (i.e., we used Equation (6.14) instead of Equation (6.13)). This made PARIS give up all matches between restaurants. The reason for this behavior turned out to be that most entities have slightly different attribute values (e.g., a phone number "213/467-1108" instead of "213-467-1108"). Therefore, we plugged in a different string equality measure. Our new measure normalizes two strings by removing all non-alphanumeric characters and lowercasing them. Then, the measure returns 1 if the strings are equal

[11] Classes and relations accumulated for both directions. Values for ObjCoref as reported in [18]. Precision and recall are not reported in [18]. ObjCoref cannot match classes or relations.

and 0 otherwise. This increased precision to 100 %, but decreased recall to 70 %. Our experience with YAGO and DBpedia (see next experiment) indicates that negative evidence can be helpful to distinguish entities of different types (movies and songs) that share one value (the title). However, in our settings, positive evidence proved sufficient.

6.7.4 Real-world Ontologies

Ontology	#Instances	#Classes	#Relations
YAGO	2,795,289	292,206	67
DBpedia	2,365,777	318	1,109
IMDb	4,842,323	15	24

Table 2: YAGO [31], DBpedia [2] and IMDb.

We wanted to test PARIS on real-world ontologies of a large scale, with a rich class and relation structure. At the same time, we wanted to restrict ourselves to cases where an error-free ground truth is available. Therefore, we first chose to align the YAGO [31] and DBpedia [2] ontologies, and then to align YAGO with an ontology built out of the IMDb[12].

YAGO vs. DBpedia. With several million instances, these are some of the largest ontologies available. Each of them has thousands of classes and at least dozens of relations. We took only the non-meta facts from YAGO, and only the manually established ontology from DBpedia, which yields the datasets described in Table 2. Both ontologies use Wikipedia identifiers for their instances, so that the ground truth for the instance matching can be computed trivially.[13] However, the statements about the instances differ in both ontologies, so that the matching is not trivial. The class structure and the relationships of YAGO and DBpedia were designed completely independently, making their alignment a challenging endeavor.

Instances					Classes					Relations				
Change to prev.	Time	Prec	Rec	F	Time	YAGO ⊆ DBp Num	YAGO ⊆ DBp Prec	DBp ⊆ YAGO Num	DBp ⊆ YAGO Prec	Time	YAGO ⊆ DBp Num	YAGO ⊆ DBp Prec	DBp ⊆ YAGO Num	DBp ⊆ YAGO Prec
-	4h04	86%	69%	77%	-	-	-	-	-	19in	30	93%	134	90%
12.4%	5h06	89%	73%	80%	-	-	-	-	-	21in	32	100%	144	92%
1.1%	5h00	90%	73%	81%	-	-	-	-	-	21in	33	100%	149	92%
0.3%	5h26	90%	73%	81%	2h14	137k	94%	149	84%	24in	33	100%	151	92%

Table 3: Results on matching YAGO and DBpedia over iterations 1–4

We ran PARIS for 4 iterations, until convergence. Table 3 shows the results per iteration. To compute recall, we counted the number of shared instances in DBpedia and YAGO. Since YAGO selects Wikipedia pages with many categories, and DBpedia selects pages with frequent infoboxes, the two resources share only 1.4 million entities. PARIS can map them with a precision of 90 % and a recall of 73 %. If only entities with more than 10 facts in DBpedia are considered, precision and recall jump to 97 % and 85 %, respectively.

[12] The Internet Movie Database, http://www.imdb.com

[13] We hid this knowledge from PARIS.

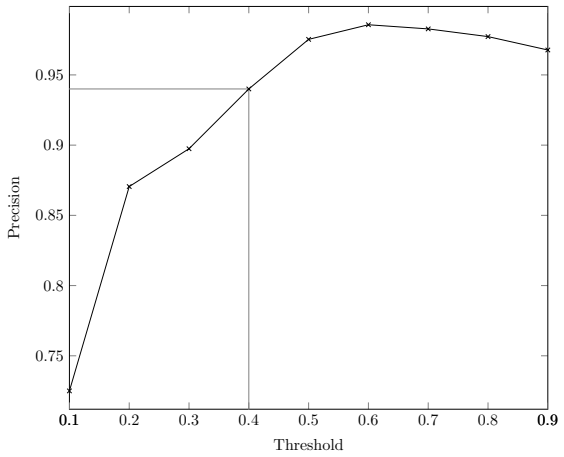

Figure 4: Precision of class alignment $YAGO \subseteq DBpedia$ as a function of the probability threshold.

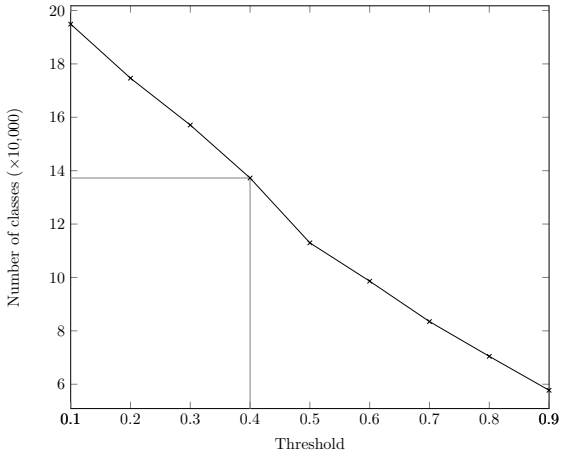

Figure 5: Number of YAGO classes that have at least one assignment in DBpedia with a score greater than the threshold.

PARIS assigns one class of one ontology to multiple classes in the taxonomy of the other ontology, taking into account the class inclusions. Some classes are assigned to multiple leaf-classes as well. For our evaluation, we excluded 19 high-level classes (such as *yagoGeoEntity*, *physicalThing*, etc.). Then, we randomly sampled from the remaining assignments and evaluated the precision manually. It turns out that the precision increases substantially with the probability score (see Figure 4). We report the numbers for a threshold of 0.4 in Table 3 (the number of evaluated sample assignments is 200 in both cases). The errors come from 3 sources: First, PARIS misclassifies a number of the instances, which worsens the precision of the class assignment. Second, there are small inconsistencies in the ontologies themselves (YAGO, e.g., has several people classified as *lumber*, because they work in the wood industry). Last, there may be biases in the instances that the ontologies talk about. For example, PARIS estimates that 12 % of the people convicted of murder in Utah were soccer

players. As the score increases, these assignments get sorted out. Evaluating whether a class is always assigned to its most specific counterpart would require exhaustive annotation of candidate inclusions. Therefore we only report the number of aligned classes and observe that even with high probability scores (see Figure 5 and Table 3) we find matches for a significant proportion of the classes of each ontology into the other.

YAGO $\subseteq$ DBpedia			
y:actedIn	$\subseteq$	dbp:starring^{-1}	0.95
y:graduatedFrom	$\subseteq$	dbp:almaMater	0.93
y:hasChild	$\subseteq$	dbp:parent^{-1}	0.53
y:hasChild	$\subseteq$	dbp:child	0.30
y:isMarriedTo	$\subseteq$	dbp:spouse^{-1}	0.56
y:isMarriedTo	$\subseteq$	dbp:spouse	0.89
y:isCitizenOf	$\subseteq$	dbp:birthPlace	0.25
y:isCitizenOf	$\subseteq$	dbp:nationality	0.88
y:created	$\subseteq$	dbp:artist^{-1}	0.13
y:created	$\subseteq$	dbp:author^{-1}	0.17
y:created	$\subseteq$	dbp:writer^{-1}	0.30

Table 6: Some relation alignments YAGO $\subseteq$ DBpedia with their scores

DBpedia $\subseteq$ YAGO			
dbp:birthName	$\subseteq$	rdfs:label	0.96
dbp:placeOfBurial	$\subseteq$	y:diedIn	0.18
dbp:headquarter	$\subseteq$	y:isLocatedIn	0.34
dbp:largestSettlement	$\subseteq$	y:isLocatedIn^{-1}	0.52
dbp:notableStudent	$\subseteq$	y:hasAdvisor^{-1}	0.10
dbp:formerName	$\subseteq$	rdfs:label	0.73
dbp:award	$\subseteq$	y:hasWonPrize	0.14
dbp:majorShrine	$\subseteq$	y:diedIn	0.11
dbp:slogan	$\subseteq$	y:hasMotto	0.49
dbp:author	$\subseteq$	y:created^{-1}	0.70
dbp:composer	$\subseteq$	y:created^{-1}	0.61

Table 7: Some relation alignments DBpedia $\subseteq$ YAGO with their scores

The relations are also evaluated manually in both directions. We consider only the maximally assigned relation, because the relations do not form a hierarchy in YAGO and DBpedia. In most cases one assignment dominates clearly. Tables 6 and 7 show some of the alignments. PARIS finds non-trivial alignments of more fine-grained relations to more coarse-grained ones, of inverses, of symmetric relations, and of relations with completely different names. There are a few plainly wrong alignments, but most errors come from semantic differences that do not show in practice (e.g., *burialPlace* is semantically different from *deathPlace*, so we count it as an error, even though in most cases the two will

coincide). Recall is hard to estimate, because not all relations have a counterpart in the other ontology and some relations are poorly populated. We only note that we find alignments for half of YAGO's relations in DBpedia.

YAGO vs. IMDb. Next, we were interested in the performance of PARIS on ontologies that do not derive from the same source. For this purpose, we constructed an RDF ontology from the IMDb. IMDb is predestined for the matching, because it is huge and there is an existing gold standard: YAGO contains some mappings to IMDb movie identifiers, and we could construct such a mapping for many persons from Wikipedia infoboxes.

Instances					Classes					Relations				
Change to prev.	Time	Prec	Rec	F	Time	YAGO ⊆ IMDb Num	YAGO ⊆ IMDb Prec	IMDb ⊆ YAGO Num	IMDb ⊆ YAGO Prec	Time	YAGO ⊆ IMDb Prec	YAGO ⊆ IMDb Rec	IMDb ⊆ YAGO Prec	IMDb ⊆ YAGO Rec
-	16h47	84%	75%	79%	-	-	-	-	-	4min	91%	73%	100%	60%
40.2%	11h44	94%	89%	91%	-	-	-	-	-	5min	91%	73%	100%	80%
6.6%	11h48	94%	90%	92%	-	-	-	-	-	5min	100%	80%	100%	80%
0.2%	11h44	94%	90%	92%	2h17	8	100%	135k	28%	6min	100%	80%	100%	80%

Table 8: Results on matching YAGO and IMDb over iterations 1–4

The content of the IMDb database is available for download as plain-text files.[14] The format of each file is *ad hoc* but we transformed the content of the database in a fairly straightforward manner into a collection of triples. For instance, the file `actors.list` lists for each actor x the list of all movies y that x was cast in, which we transformed into facts $actedIn(x, y)$. Unfortunately, the plain-text database does not contain IMDb movie and person identifiers (those that we use for comparing to the gold standard). Consequently, we had to obtain these identifiers separately. For this purpose, and to avoid having to access each Web page of the IMDb Web site, which would require much too many Web server requests, we used the advanced search feature of IMDb[15] to obtain the list of all movies from a given year, or of all persons born in a certain year, together with their identifiers and everything needed to connect to the plain-text databases.

Since our IMDb ontology has only 24 relations, we manually created a gold standard for relations, aligning 15 of them to YAGO relations.

As Table 8 shows, PARIS took much longer for each iteration than in the previous experiment. The results are convincing, with an F-score of 92 % for the instances. This is a considerable improvement over a baseline approach that aligns entities by matching their *rdfs:label* properties (achieving 97 % precision and only 70 % recall, with an F-score of 82 %). Examining by hand the few remaining alignment errors revealed the following patterns:

- Some errors were caused by errors in YAGO, usually caused by incorrect references from Wikipedia pages to IMDb movies.
- PARIS sometimes aligned instances in YAGO with instances in IMDb that were not equivalent, but very closely related: for example, *King of the Royal Mounted* was aligned with *The Yukon Patrol*, a feature version of this TV series with the same cast and crew; *Out 1*, a 13-hour movie, was aligned with *Out 1: Spectre*, its shortened 4-hour variation.

[14] http://www.imdb.com/interfaces#plain
[15] http://akas.imdb.com/search/

- Some errors were caused by the very naïve string comparison approach, that fails to discover, e.g., that *Sugata Sanshirô* and *Sanshiro Sugata* refer to the same movie. It is very likely that using an improved string comparison technique would further increase precision and recall of PARIS.

PARIS could align 80 % of the relations of IMDb and YAGO, with a precision of 100 %. PARIS mapped half of the IMDb classes correctly to more general or equal YAGO classes (at threshold 0). It performs less well in the other direction. This is because YAGO contains mostly famous people, many of whom appeared in some movie or documentary on IMDb. Thus, PARIS believes that a class such as *People from Central Java* is a subclass of *actor*.

As illustrated here, alignment of instances and relations work very well in PARIS, whereas class alignment leaves still some room for improvement. Overall, the results are very satisfactory, as this constitutes, to the best of our knowledge, the first holistic alignment of instances, relations, and classes on some of the world's largest ontologies, without any prior knowledge, tuning, or training.

6.8 Conclusion

We have presented PARIS, an algorithm for the automated alignment of RDFS ontologies. Unlike most other approaches, PARIS computes alignments not only for instances, but also for classes and relations. It does not need training data and it does not require any parameter tuning. PARIS is based on a probabilistic framework that captures the interplay between schema alignment and instance matching in a natural way, thus providing a holistic solution to the ontology alignment problem. Experiments show that our approach works extremely well in practice.

PARIS does not use any kind of heuristics on relation names, which allows aligning relations with completely different names. We conjecture that the name heuristics of more traditional schema-alignment techniques could be factored into the model.

Currently, PARIS cannot deal with structural heterogeneity. If one ontology models an event by a relation (such as *wonAward*), while the other one models it by an event entity (such as *winningEvent*, with relations *winner*, *award*, *year*), then PARIS will not be able to find matches. The same applies if one ontology is more fine-grained than the other one (specifying, e.g., cities as birth places instead of countries), or if one ontology treats cities as entities, while the other one refers to them by strings. For future work, we plan to address these types of challenges. We also plan to analyze under which conditions our equations are guaranteed to converge. It would also be interesting to apply PARIS to more than two ontologies. This would further increase the usefulness of PARIS for the dream of the Semantic Web.

6.9 Acknowledgments

This work has been supported in part by the Advanced European Research Council grant Webdam on Foundations of Web Data Management, grant agreement 226513 (http://webdam.inria.fr/).

6.10 Supplementary Considerations

6.10.1 Global Functionality

There are several design alternatives to define the *global functionality*:

1. We can count the number of statements and divide it by the number of pairs of statements with the same source:

$$fun(r) = \frac{\#x, y : r(x, y)}{\#x, y, y' : r(x, y) \wedge r(x, y')}$$

   This measure is very volatile to single sources that have a large number of targets.

2. We can define functionality as the ratio of the number of first arguments to the number of second arguments:

$$fun(r) = \frac{\#x \quad \exists y : r(x, y)}{\#y \quad \exists x : r(x, y)}$$

   This definition is treacherous: Assume that we have n people and n dishes, and the relationship $likesDish(x, y)$. Now, assume that all people like all dishes. Then *likesDish* should have a low functionality, because everybody likes n dishes. But the above definition assigns a functionality of $fun(likesDish) = \frac{n}{n} = 1$.

3. We can average the local functionalities, as proposed in [17]:

$$\begin{aligned} fun(r) &= \operatorname*{avg}_x fun(r, x) = \operatorname*{avg}_x \left(\frac{1}{\#y : r(x, y)} \right) \\ &= \frac{1}{\#x \quad \exists y : r(x, y)} \sum_x \frac{1}{\#y : r(x, y)}. \end{aligned}$$

   However, the local functionalities are ratios, so that the arithmetic mean is less appropriate.

4. We can average the local functionalities not by the arithmetic mean, but by the harmonic mean instead

$$\begin{aligned} fun(r) &= \operatorname*{HM}_x fun(r, x) = \operatorname*{HM}_x \left(\frac{1}{\#y : r(x, y)} \right) \\ &= \frac{\#x \quad \exists y : r(x, y)}{\sum_x \#y : r(x, y)} = \frac{\#x \quad \exists y : r(x, y)}{\#x, y : r(x, y)}. \end{aligned}$$

5. We may say that the global functionality is the number of first arguments per relationship instance:

$$fun(r) = \frac{\#x \quad \exists y : r(x, y)}{\#x, y : r(x, y)}$$

   This notion is equivalent to the harmonic mean.

With these considerations in mind, we chose the harmonic mean for the definition of the global functionality.

6.10.2 Probabilistic Modeling

In Section 6.5, we presented a model of equality based on logical rules such as Equation (6.5), reproduced here:

$$\exists r, y : r(x, y) \wedge (\forall y' : r(x', y') \Rightarrow y \not\equiv y') \wedge \mathit{fun}(r) \text{ is high} \implies x \not\equiv x \tag{6.18}$$

To transform these equations into probability assessments (Equation (6.6)), we assume mutual independence of all distinct elements of our models (instance equivalence, functionality, relationship inclusion, etc.). This assumption is of course not true in practice but it allows us to approximate efficiently the probability of the consequence of our alignment rules in a canonical manner. Independence allows us to use the following standard identities:

$$\begin{aligned}
Pr(A \wedge B) &= Pr(A) \times Pr(B) \\
Pr(A \vee B) &= 1 - (1 - Pr(A))(1 - Pr(B)) \\
Pr(\forall x : \varphi(x)) &= \prod_x Pr(\varphi(x)) \\
Pr(\exists x : \varphi(x)) &= 1 - \prod_x (1 - Pr(\varphi(x))) \\
\mathbb{E}(\#x : \varphi(x)) &= \sum_x Pr(\varphi(x))
\end{aligned}$$

Then, a rule $\varphi \implies \psi$ can be translated into a probability assignment $Pr(\psi) := Pr(\varphi)$. φ is recursively decomposed using the above identities. Following the example of Equation (6.5), we derive the value of $Pr_2(x \equiv x')$ in Equation (6.6) as follows:

$$\begin{aligned}
&1 - Pr\big(\exists r, y\, r(x, y) \wedge (\forall y'\, r(x', y') \Rightarrow y \not\equiv y') \wedge \\
&\qquad \mathit{fun}(r) \text{ is high}\big) \\
&= \prod_{r,y} \Bigg(1 - Pr(r(x, y)) \times \\
&\qquad \prod_{y'} \Big(1 - Pr(r(x', y') \wedge y \equiv y') \Big) \times \mathit{fun}(r) \Bigg) \\
&= \prod_{r(x,y)} \Bigg(1 - \mathit{fun}(r) \prod_{r(x',y')} (1 - Pr(y \equiv y')) \Bigg)
\end{aligned}$$

since $r(x, y)$ and $r(x', y')$ are crisp, non-probabilistic facts.

Similarly, when we need to estimate a number such as "$\#x : \varphi(x)$", we compute $\mathbb{E}(\#x : \varphi(x))$ using the aforementioned identities.

The computed values reflect the probability that two entities are equal in the model, if the following conditions hold: (1) the literal equality functions of Section 6.6.3 return the probability of two values being equal, (2) a relation r is either a function or not, and $fun(r)$ reflects the probability of this event, and (3) all probabilities are independent. Although these conditions are certainly not true to their full extent in practice, the equations still deliver useful approximations.

6.10.3 Equivalence of Sets

We compare two instances for equivalence by comparing every statement about the first instance with every statement about the second instance (if they have the same relation). This entails a quadratic number of comparisons. For example, if an actor x acted in the movies y_1, y_2, y_3, and an actor x' acted in the movies y'_1, y'_2, y'_3, then we will compare every statement $actedIn(x, y_i)$ with every statement $actedIn(x', y'_j)$. Alternatively, one could think of the target values as a set and of the relation as a function, as in $actedIn(x, \{y_1, y_2, y_3\})$ and $actedIn(x', \{y'_1, y'_2, y'_3\})$. Then, one would have to compare only two sets instead of a quadratic number of statements. However, all elements of one set are potentially equivalent to all elements of the other set. Thus, one would still need a quadratic number of comparisons.

One could generalize a set equivalence measure (such as the Jaccard index) to sets with probabilistic equivalences. However, one would still need to take into account the functionality of the relations: If two people share an e-mail address (high inverse functionality), they are almost certainly equivalent. By contrast, if two people share the city they live in, they are not necessarily equivalent. To unify two instances, it is sufficient that they share the value of one highly inverse functional relation. Conversely, if two people have a different birth date, they are certainly different. By contrast, if they like two different books, they could still be equivalent (and like both books). Our model takes this into account. Thus, our formulas can be seen as a comparison measure for sets with probabilistic equivalences, which takes into account the functionalities.

References

[1] A. Arasu, C. Re, and D. Suciu. “Large-Scale Deduplication with Constraints Using Dedupalog”. In: *Proc. ICDE*. 2009, pp. 952–963.

[2] Sören Auer et al. “DBpedia: A Nucleus for a Web of Open Data”. In: *Proc. ISWC*. 2007, pp. 722–735.

[3] David Aumueller et al. “Schema and ontology matching with COMA++”. In: *Proc. SIGMOD*. Baltimore, Maryland, 2005, pp. 906–908.

[4] Indrajit Bhattacharya and Lise Getoor. “Collective entity resolution in relational data”. In: *ACM TKDD* 1 (1 Mar. 2007).

[5] Chris Bizer. “Web of Linked Data. A global public data space on the Web”. In: *Proc. WebDB*. 2010.

[6] Christian Bizer et al. “Linked data on the Web”. In: *Proc. WWW*. Beijing, China, 2008, pp. 1265–1266.

[7] Jens Bleiholder and Felix Naumann. “Data fusion”. In: *ACM Comput. Surv.* 41.1 (2008).

[8] Li Ding et al. “SameAs networks and beyond: Analyzing deployment status and implications of owl:sameAs in linked data”. In: *Proc. ISWC*. Shanghai, China, 2010, pp. 142–147.

[9] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. “Duplicate Record Detection: A Survey”. In: *IEEE TKDE* 19.1 (2007), pp. 1–16.

[10] Oren Etzioni et al. “Web-scale information extraction in KnowItAll (preliminary results)”. In: *Proc. WWW*. New York, NY, USA, 2004, pp. 100–110.

[11] J. Euzénat et al. “Results of the ontology alignment evaluation initiative 2010”. In: *Proc. OM*. 2010.

[12] Alfio Ferrara, Davide Lorusso, and Stefano Montanelli. “Automatic Identity Recognition in The Semantic Web”. In: *Proc. IRSW*. 2008.

[13] Hugh Glaser, Afraz Jaffri, and Ian Millard. “Managing Co-reference on the Semantic Web”. In: *Proc. LDOW*. 2009.

[14] Jorge Gracia, Mathieu d’Aquin, and Eduardo Mena. “Large scale integration of senses for the semantic Web”. In: *Proc. WWW*. Madrid, Spain, 2009, pp. 611–620.

[15] Harry Halpin et al. “When owl:sameAs isn’t the Same: An Analysis of Identity in Linked Data”. In: *Proc. ISWC*. 2010, pp. 305–320.

[16] Aidan Hogan. “Performing object consolidation on the semantic Web data graph”. In: *Proc. I3*. 2007.

[17] Aidan Hogan et al. “Some entities are more equal than others: statistical methods to consolidate Linked Data”. In: *Proc. NeFoRS*. 2010.

[18] Wei Hu, Jianfeng Chen, and Yuzhong Qu. “A self-training approach for resolving object coreference on the semantic Web”. In: *Proc. WWW*. 2011, pp. 87–96.

[19] Wei Hu et al. “How Matchable Are Four Thousand Ontologies on the Semantic Web”. In: *Proc. ESWC*. 2011, pp. 290–304.

[20] Antoine Isaac et al. “An empirical study of instance-based ontology matching”. In: *Proc. ISWC*. Busan, Korea, 2007, pp. 253–266.

[21] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. “Ontology matching with semantic verification”. In: *J. Web Semantics* 7.3 (2009), pp. 235–251.

[22] Simon Lacoste-Julien et al. “SIGMa: simple greedy matching for aligning large knowledge bases”. In: *KDD*. 2013.

[23] Juanzi Li et al. “RiMOM: A Dynamic Multistrategy Ontology Alignment Framework”. In: *IEEE TKDE* 21.8 (2009), pp. 1218–1232.

[24] Cynthia Matuszek et al. “An introduction to the syntax and content of Cyc”. In: *Proc. AAAI Spring Symposium*. 2006.

[25] Ian Niles and Adam Pease. “Towards a standard upper ontology”. In: *Proc. FOIS*. Ogunquit, Maine, USA, 2001, pp. 2–9.

[26] Jan Noessner et al. “Leveraging Terminological Structure for Object Reconciliation”. In: *Proc. ESWC*. 2010, pp. 334–348.

[27] Simone Paolo Ponzetto and Michael Strube. “Deriving a Large-Scale Taxonomy from Wikipedia”. In: *Proc. AAAI*. 2007, pp. 1440–1445.

[28] Fatiha Saïs, Nathalie Pernelle, and Marie-Christine Rousset. “Combining a Logical and a Numerical Method for Data Reconciliation”. In: *J. Data Semantics* 12 (2009), pp. 66–94.

[29] Fatiha Saïs, Nathalie Pernelle, and Marie-Christine Rousset. "L2R: A Logical Method for Reference Reconciliation". In: *Proc. AAAI*. 2007, pp. 329–334.

[30] Fabian M. Suchanek, Georgiana Ifrim, and Gerhard Weikum. "Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents". In: *KDD*. 2006, pp. 412–417.

[31] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Core of Semantic Knowledge. Unifying WordNet and Wikipedia". In: *Proc. WWW*. 2007, pp. 697–706.

[32] Octavian Udrea, Lise Getoor, and Renée J. Miller. "Leveraging data and structure in ontology integration". In: *Proc. SIGMOD*. Beijing, China, 2007, pp. 449–460.

[33] Julius Volz et al. "Discovering and Maintaining Links on the Web of Data". In: *Proc. ISWC*. 2009, pp. 650–665.

[34] Shenghui Wang, Gwenn Englebienne, and Stefan Schlobach. "Learning Concept Mappings from Instance Similarity". In: *Proc. ISWC*. 2008, pp. 339–355.

[35] Word Wide Web Consortium. *RDF Primer (W3C Recommendation 2004-02-10)*. http://www.w3.org/TR/rdf-primer/. 2004.

Chapter 7

Pattern Mining with PATTY

7.1 Overview

This chapter presents PATTY: a large resource for textual patterns that denote binary relations between entities. The patterns are semantically typed and organized into a subsumption taxonomy. The PATTY system is based on efficient algorithms for frequent itemset mining and can process Web-scale corpora. It harnesses the rich type system and entity population of large knowledge bases. The PATTY taxonomy comprises 350,569 pattern synsets. Random-sampling-based evaluation shows a pattern accuracy of 84.7%. PATTY has 8,162 subsumptions, with a random-sampling-based precision of 75%. The PATTY resource is freely available for interactive access and download. This chapter is based on

> Ndapandula Nakashole, Gerhard Weikum, Fabian M. Suchanek
> "PATTY: A Taxonomy of Relational Patterns with Semantic Types"
> (International Conference on Empirical Methods
> in Natural Language Processing (EMNLP) 2012)

7.2 Introduction

Motivation. WordNet [11] is one of the most widely used lexical resources in computer science. It groups nouns, verbs, and adjectives into sets of synonyms, and arranges these synonyms in a taxonomy of hypernyms. WordNet is limited to single words. It does not contain entire *phrases* or *patterns* between entities. For example, WordNet does not contain the pattern *X is romantically involved with Y*. Just like words, patterns can be synonymous, and they can subsume each other. The pattern *X is romantically involved with Y* is synonymous with the pattern *X is dating Y*. Both are subsumed by *X knows Y*. Patterns for relations are a vital ingredient for many applications, including information extraction and question answering. If a large-scale resource of relational patterns were available, this could boost progress in NLP and AI tasks.

Yet, existing large-scale knowledge bases are mostly limited to abstract binary relationships between entities, such as "bornIn" [4, 7, 27, 35]. These do not correspond to real text phrases. Only the ReVerb system [12] yields a larger number of relational textual patterns. However, no attempt is made to organize these patterns into synonymous patterns, let alone into a taxonomy. Thus, the patterns themselves do not exhibit semantics.

Goal. Our goal in this paper is to systematically compile relational patterns from a corpus, and to impose a semantically typed structure on them. The result we aim at is a WordNet-style taxonomy of binary relations. In particular, we aim at patterns that contain *semantic types*, such as ⟨*singer*⟩ *sings* ⟨*song*⟩. We also want to automatically generalize syntactic variations such as *sings her* ⟨*song*⟩ and *sings his* ⟨*song*⟩, into a more general pattern *sings [prp]* ⟨*song*⟩ with POS tag [prp]. Analogously but more demandingly, we want to automatically infer that the above patterns are semantically subsumed by the pattern ⟨*musician*⟩ *performs on* ⟨*musical composition*⟩ with more general types for the entity arguments in the pattern.

Compiling and organizing such patterns is challenging for the following reasons. 1) The number of possible patterns increases exponentially with the length of the patterns. For example, the string "Amy sings 'Rehab"' can give rise to the patterns ⟨*singer*⟩ *sings* ⟨*song*⟩, ⟨*person*⟩ *sings* ⟨*artifact*⟩, ⟨*person*⟩ *[vbz]* ⟨*entity*⟩, etc. If wildcards for multiple words are allowed (such as in ⟨*person*⟩ *sings* * ⟨*song*⟩), the number of possible patterns explodes. 2) A pattern can be semantically more general than another pattern (when one relation is implied by the other relation), and it can also be syntactically more general than another pattern (by the use of placeholders such as [vbz]). These two subsumption orders have a non-obvious interplay, and none can be analyzed without the other. 3) We have to handle pattern sparseness and coincidental matches. If the corpus is small, e.g., the patterns ⟨*singer*⟩ *later disliked her song* ⟨*song*⟩ and ⟨*singer*⟩ *sang* ⟨*song*⟩, may apply to the same set of entity pairs in the corpus. Still, the patterns are not synonymous. 4) Computing mutual subsumptions on a large set of patterns may be prohibitively slow. Moreover, due to noise and vague semantics, patterns may even not form a crisp taxonomy, but require a hierarchy in which subsumption relations have to be weighted by statistical confidence measures.

Contributions. In this paper, we present PATTY, a large resource of relational patterns that are arranged in a semantically meaningful taxonomy, along with entity-pair instances. More precisely, our contributions are as follows:

1) *SOL patterns:* We define an expressive family of relational patterns, which combines syntactic features (S), ontological type signatures (O), and lexical features (L). The crucial novelty is the addition of the ontological, semantic dimension to patterns. When compared to a state-of-the-art pattern language, we found that SOL patterns yield higher recall while achieving similar precision.

2) *Mining algorithms:* We present efficient and scalable algorithms that can infer SOL patterns and subsumptions at scale, based on instance-level overlaps and an ontological type hierarchy.

3) A large *Lexical resource:*. On the Wikipedia corpus, we obtained 350,569 pattern synsets with 84.7We make our pattern taxonomy available for further research at http://www.mpi-inf.mpg.de/yago-naga/patty/ .

The paper is structured as follows. Section 2 discusses related work. Section 3 outlines the basic machinery for pattern extraction. Section 4 introduces our SOL pattern model. Sections 5 and 6 present the syntactic and semantic generalization of patterns. Section 7 explains how to arrange the patterns into a taxonomy. Section 8 reports our experimental findings.

7.3 Related Work

A wealth of taxonomic knowledge bases (KBs) about entities and their semantic classes have become available. These are very rich in terms of unary predicates (semantic classes) and their entity instances. However, the number of *binary relations* (i.e., relation types, not instances) in these KBs is usually small: Freebase [7] has a few thousand hand-crafted relations. WikiNet [27] has automatically extracted ca. 500 relations from Wikipedia category names. DBpedia [4] has automatically compiled ca. 8000 names of properties from Wikipedia infoboxes, but these include many involuntary semantic duplicates such as *surname* and *lastname*. In all of these projects, the resource contains the *relation names*, but not the *natural language patterns* for them. The same is true for other projects along these lines [28, 31, 32, 35].

In contrast, knowledge base projects that automatically populate relations from Web pages also learn *surface patterns* for the relations: examples are TextRunner/ReVerb [5, 12], NELL [9, 25], Probase [39], the dynamic lexicon approach by [18, 38], the LDA-style clustering approach by [40], and projects on Web tables [22, 37]. Of these, only TextRunner/ReVerb and NELL have made large pattern collections publicly available.

ReVerb [12] constrains patterns to verbs or verb phrases that end with prepositions, while PATTY can learn arbitrary patterns. More importantly, all methods in the TextRunner/ReVerb family are blind to the ontological dimension of the entities in the patterns. Therefore, there is no notion of semantic typing for relation phrases as in PATTY.

NELL [9] is based on a fixed set of prespecified relations with type signatures, (e.g., *personHasCitizenship:* $\langle person \rangle \times \langle country \rangle$), and learns to extract suitable noun-phrase pairs from a large Web corpus. In contrast, PATTY discovers patterns for relations that are a priori unknown.

In *OntExt* [25], the NELL architecture was extended to automatically compute new relation types (beyond the prespecified ones) for a given type signature of arguments, based on a clustering technique. For example, the relation *musicianPlaysInstrument* is found by clustering pattern co-occurrences for the noun-phrase pairs that fall into the specific type signature $\langle musician \rangle \times \langle musicinstrument \rangle$. This technique works for one type signature at a time, and does not scale up to mining a large corpus. Also, the technique is not suitable for inferring semantic subsumptions. In contrast, PATTY efficiently acquires patterns from large-scale corpora and organizes them into a subsumption hierarchy.

Class-based *attribute discovery* is a special case of mining relational patterns (e.g., [2, 30, 29, 33]). Given a semantic class, such as *movies* or *musicians*, the task is to determine relevant attributes, such as *cast* and *budget* for movies, or *albums* and *biography* for musicians, along with their instances. Unlike PATTY's patterns, the attributes are not typed. They come with a prespecified type for

the domain, but without any type for the range of the underlying relation.

There are further *relation-centric tasks in NLP and text mining* that have commonalities with our endeavor, but differ in fundamental ways. The SemEval-2010 task on classification of semantic relations between noun-phrase pairs [16] aimed at predicting the relation for a given sentence and pair of nominals, but used a fixed set of prespecified relations. Another task in this research avenue is to characterize and predict the argument types for a given relation or pattern [21, 26]. This is closer to KB population and less related to our task of discovering relational patterns and systematically organizing them.

From a *linguistic perspective*, there is ample work on patterns for unary predicates of the form *class(entity)*. This includes work on entailment of classes, i.e., on *is-a* and *subclassOf* relationships. Entailment among binary predicates of the form *relation(entity1, entity2)* has received less attention [23, 10, 14, 6]. These works focus solely on verbs, while PATTY learns arbitrary phrases for patterns.

Several lexical resources capture verb categories and entailment: WordNet 3.0 [11] contains about 13,000 verb senses, with troponymy and entailment relations; VerbNet [20] is a hierarchical lexicon with more than 5,000 verb senses in ca. 300 classes, including selectional preferences. Again, all of these resources focus solely on verbs.

ConceptNet 5.0 [15] is a thesaurus of commonsense knowledge built as a crowdsourcing endeavor. PATTY, in contrast, is constructed fully automatically from large corpora. Automatic learning of paraphrases and textual entailment has received much attention (see the survey of [3]), but does not consider fine-grained typing for binary relations, as PATTY does.

7.4 Pattern Extraction

This section explains how we obtain basic textual patterns from the input corpus. We first apply the Stanford Parser [24] to the individual sentences of the corpus to obtain dependency paths. The dependency paths form a directed graph, with words being nodes and dependencies being edges. For example, the sentence *"Winehouse effortlessly performed her song Rehab."* yields the following dependency paths:

nsubj(performed-3, Winehouse-1)
advmod(performed-3, effortlessly-2)
poss(Rehab-6, her-4)
nn(Rehab-6, song-5)
dobj(performed-3, Rehab-6)

While our method also works with patterns obtained from shallow features such as POS tags, we found that dependency paths improve pattern extraction precision especially on long sentences.

We then detect mentions of named entities in the parsed corpus. For this purpose, we use a dictionary of entities. This can be any resource that contains named entities with their surface names and semantic types [4, 35, 17, 7]. In our experiments, we used the YAGO2 knowledge base [17]. We match noun phrases that contain at least one proper noun against the dictionary. For disamiguation,

we use a simple context-similarity prior, as described in [36]. We empirically found that this technique has accuracy well above 80% (and higher for prominent and thus frequently occurring entities). In our example, the entity detection yields the entities *Amy Winehouse* and *Rehab (song)*.

Whenever two named entities appear in the same sentence, we extract a textual pattern. For this purpose, we traverse the dependency graph to get the shortest path that connects the two entities. In the example, the shortest path between "Winehouse" and "Rehab" is: Winehouse *nsubj* performed *dobj* Rehab. In order to capture only relations that refer to subject-relation-object triples, we only consider shortest paths that start with subject-like dependencies, such as *nsubj, rcmod and partmod*. To reflect the full meaning of the patterns, we expand the shortest path with adverbial and adjectival modifiers, for example the *advmod* dependency. The sequence of words on the expanded shortest path becomes our final textual pattern. In the example, the textual pattern is *Amy Winehouse effortlessly performed Rehab (song)*.

7.5 SOL Pattern Model

Textual patterns are tied to the particular surface form of the text. Therefore, we transform the textual patterns into a new type of patterns, called syntactic-ontologic-lexical patterns (SOL patterns). SOL patterns extend lexico-syntactic patterns by ontological type signatures for entities. The SOL pattern language is expressive enough to capture fine-grained relational patterns, yet simple enough to be dealt with by efficient mining algorithms at Web scale.

A SOL pattern is an abstraction of a textual pattern that connects two entities of interest. It is a sequence of words, POS-tags, wildcards, and ontological types. A POS-tag stands for a word of the part-of-speech class. We introduce the special POS-tag [word], which stands for any word of any POS class. A wildcard, denoted $*$, stands for any (possibly empty) sequence of words. Wildcards are essential to avoid overfitting of patterns to the corpus. An ontological type is a semantic class name (such as $\langle singer \rangle$) that stands for an instance of that class. Every pattern contains at least two types, and these are designated as *entity placeholders*.

A string and a pattern *match*, if there is an order-preserving bijection from sequences of words in the string to items in the pattern, so that each item can stand for the respective sequence of words. For example, the pattern $\langle person \rangle$ *'s [adj] voice* * $\langle song \rangle$ matches the strings "Amy Winehouse's soft voice in 'Rehab"' and "Elvis Presley's solid voice in his song 'All shook up"'. The *type signature* of a pattern is the pair of the entity placeholders. In the example, the type signature is $person \times song$. The *support set* of a pattern is the set of pairs of entities that appear in the place of the entity placeholders in all strings in the corpus that match the pattern. In the example, the support set of the pattern could be $\{(Amy, Rehab), (Elvis, AllShookUp)\}$. Each pair is called a *support pair* of the pattern.

Pattern B is *syntactically more general* than pattern A if every string that matches A also matches B. Pattern B is *semantically more general* than A if the support set of B is a superset of the support set of A. If A is semantically more general than B and B is semantically more general than A, the patterns are called *synonymous*. A set of synonymous patterns is called a *pattern synset*.

Two patterns, of which neither is semantically more general than the other, are called *semantically different*.

To generate SOL patterns from the textual patterns, we decompose the textual patterns into n-grams (n consecutive words). A SOL pattern contains only the n-grams that appear frequently in the corpus and the remaining word sequences are replaced by wildcards. For example, in the sentence "was the first female to run for the governor of" might give rise to the pattern ** the first female * governor of*, if "the first female" and "governor of" are frequent in the corpus.

To find the frequent n-grams efficiently, we apply the technique of frequent itemset mining [1, 34]: each sentence is viewed as a "shopping transaction" with a "purchase" of several n-grams, and the mining algorithm computes the n-gram combinations with large co-occurrence support[1]. These n-grams allow us to break down a sentence into wildcard-separated subsequences, which yields an SOL pattern. We generate multiple patterns with different types, one for each combination of types that the detected entities have in the underlying ontology.

We quantify the *statistical strength* of a pattern by means of its support set. For a given pattern p with type signature *t1* $\times$ *t2*, the *support* of p is the size of its support set. For confidence, we compare the support-set sizes of p and an untyped variant p^u of p, in which the types $\langle t1 \rangle$ and $\langle t2 \rangle$ are replaced by the generic type $\langle entity \rangle$. We define the *confidence* of p as the ratio of the support-set sizes of p and p^u.

7.6 Syntactic Pattern Generalization

Almost every pattern can be generalized into a syntactically more general pattern in several ways: by replacing words by POS-tags, by introducing wildcards (combining more n-grams), or by generalizing the types in the pattern. It is not obvious which generalizations will be reasonable and useful. We observe, however, that generalizing a pattern may create a pattern that subsumes two semantically different patterns. For example, the generalization $\langle person \rangle$ *[vb]* $\langle person \rangle$ subsumes the two semantically different patterns $\langle person \rangle$ *loves* $\langle person \rangle$ and $\langle person \rangle$ *hates* $\langle person \rangle$. This means that the pattern is semantically meaningless.

Therefore, we proceed as follows. For every pattern, we generate all possible generalizations. If a generalization subsumes multiple patterns with disjoint support sets, we abandon the generalized pattern. Otherwise, we add it to our set of patterns.

7.7 Semantic Pattern Generalization

The main difficulty in generating semantic subsumptions is that the support sets may contain spurious pairs or be incomplete, thus destroying crisp set inclusions. To overcome this problem, we designed a notion of a *soft set inclusion*, in which one set S can be a subset of another set B to a certain degree. One possible measure for this degree is the confidence, i.e., the ratio of elements in S that

[1] Our implementation restricts n-grams to length 3 and uses up to 4 n-grams per sentence

are in B, $deg(S \subseteq B) = |S \cap B|/|S|$. However, if a support set S has only few elements due to sparsity, it may become a subset of another support set B, even if the two patterns are semantically different. Therefore, one has to take into account also the *support*, i.e., the size of the set S. Traditionally, this is done through a weighted trade-off between confidence and support.

To avoid the weight tuning, we instead devised a probabilistic model. We interpret S as a *random sample* from the "true" support set S' that the pattern would have on an infinitely large corpus. We want to estimate the ratio of elements of S' that are in B. This ratio is a Bernoulli parameter that can be estimated from the ratio of elements of the sample S that are in B. We compute the Wilson score interval $[c-d, c+d]$ [8] for the sample. This interval guarantees that with a given probability (set a priori, usually to $\alpha = 95\%$), the true ratio falls into the interval $[c-d, c+d]$. If the sample is small, d is large and c is close to 0.5. If the sample is large, d decreases and c approaches the naive estimation $|S \cap B|/|S|$. Thereby, the Wilson interval center naturally balances the trade-off between confidence and the support. Hence we define $deg(S \subset B) = c$. This estimator may degrade when the sample size is too small We can alternatively use a conservative estimator $deg(S \subset B) = c - d$, i.e., the lower bound of the Wilson score interval. This gives a low score to the case where $S \subset B$ if we have few samples (S is small).

7.8 Taxonomy Construction

We now have to arrange the patterns in a semantic taxonomy. A baseline solution would compare every pattern support set to every other pattern support set in order to determine inclusion, mutual inclusion, or independence. This would be prohibitively slow. For this reason, we make use of a prefix-tree for frequent patterns [13]. The prefix-tree stores support sets of patterns. We then developed an algorithm for obtaining set intersections from the prefix-tree.

7.8.1 Prefix-Tree Construction

Suppose we have pattern synsets and their support sets as shown in Table 1. An entity pair in a support set is denoted by a letter. For example, in the support set for the pattern $\langle Politican \rangle$ *was governor of* $\langle State \rangle$, the entry $\langle A,80 \rangle$ may denote the entity pair *Arnold Schwarzenegger, California*, with an occurrence frequency *80*. The contents of the support sets are used to construct a prefix-tree, where nodes are entity pairs. If synsets have entity pairs in common, they share a common prefix; thus the shared parts can be represented by one prefix-path in the tree. This enables subsumptions to be directly "read off" from the tree, while representing the tree in a compact manner. To increase the chance of shared prefixes, entity pairs are inserted into the tree in decreasing order of occurrence frequency.

The prefix-tree of support sets is a prefix-tree augmented with synset information stored at the nodes. Each node (entity pair) stores the identifiers of the pattern sysnets whose support sets contain that entity pair. In addition, each node stores a link to the next node with the same entity pair.

ID	Pattern Synset & Support Sets
P_1	$\langle Politician \rangle$ was governor of $\langle State \rangle$ A,80 B,75 C,70
P_2	$\langle Politician \rangle$ *politician from* $\langle State \rangle$ A,80 B,75 C,70 D,66 E,64
P_3	$\langle Person \rangle$ *daughter of* $\langle Person \rangle$ F,78 G,75 H,66
P_4	$\langle Person \rangle$ *child of* $\langle Person \rangle$ I,88 J,87 F,78 G,75 K,64

Table 1: Pattern Synsets and their Support Sets

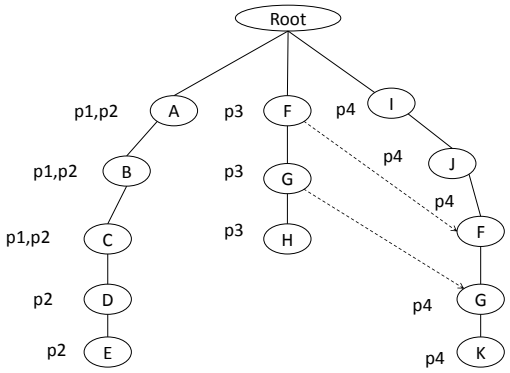

Figure 2: Prefix-Tree for the Synsets in Table 1

Figure 2 shows the tree for the pattern synsets in Table 1. The left-most path contains synsets P_1 and P_2. The two patterns have a prefix in common, thus they share the same path. This is reflected by the synsets stored in the nodes in the path. Synsets P_2 and P_3 belong to two different paths due to dissimilar prefixes although they have common nodes. Instead, their common nodes are connected by the same-entity-pair links shown as dotted lines in Figure 2. These links are created whenever the entity pair already exists in the tree but with a prefix different from the prefix of the synset being added to the tree. The size of the tree is at most the total number of entity pairs making up the supports sets of the synsets. The height of the tree is at most the size of the the largest support set.

7.8.2 Mining Subsumptions from the Prefix-Tree

To efficiently mine subsumptions from the prefix-tree, we have to avoid comparing every path to every other path as this introduces the same inefficiencies that the baseline approach suffers from.

From the construction of the tree it follows that for any node N_i in the tree, all paths containing N_i can be found by following node N_i*'s* links including the same-entity-pair links. By traversing the entire path of a synset P_i, we can reach all the pattern synsets sharing common nodes with P_i. This leads to our

main insight: if we start traversing the tree bottom up, starting at the last node in $P_i's$ support set, we can determine exactly which paths are subsumed by P_i. Traversing the tree this way for all patterns gives us the sizes of the support set intersection. The determined intersection sizes can then be used in the Wilson estimator to determine the degree of semantic subsumption and semantic equivalence of patterns.

7.8.3 DAG Construction

Once we have generated subsumptions between relational patterns, there might be cycles in the graph we generate. We ideally want to remove the minimal total number of subsumptions whose removal results in an a directed acyclic graph (DAG). This task is related to the minimum feedback-arc-set problem: given a directed graph, we want to remove the smallest set of edges whose removal makes the remaining graph acyclic. This is a well known NP-hard problem [19]. We use a greedy algorithm for removing cycles and eliminating redundancy in the subsumptions, thus effectively constructing a DAG. Starting with a list of subsumption edges ordered by decreasing weights, we construct the DAG bottom-up by adding the highest-weight subsumption edge. This step is repeated for all subsumptions, where we add a subsumption to the DAG only if it does not introduce cycles or redundancy. Redundancy occurs when there already exists a path, by transitivity of subsumptions, between pattern synsets linked by the subsumption. This process finally yields a DAG of pattern synsets – the PATTY taxonony.

7.9 Experimental Evaluation

7.9.1 Setup

The PATTY extraction and mining algorithms were run on two different input corpora: the New York Times archive (*NYT*) which includes about 1.8 Million newspaper articles from the years 1987 to 2007, and the English edition of Wikipedia (*WKP*), which contains about 3.8 Million articles (as of June 21, 2011). Experiments were carried out, for each corpus, with two different type systems: a) the type system of YAGO2, which consists of about 350,000 semantic classes from WordNet and the Wikipedia category system, and b) the two-level domain/type hierarchy of Freebase which consists of 85 domains and a total of about 2000 types within these domains.

All relational patterns and their respective entity pairs are stored in a MongoDB database. We evaluated PATTY along four dimensions: quality of patterns, quality of subsumptions, coverage, and design alternatives. These dimensions are discussed in the following four subsections. We also performed an extrinsic study to demonstrate the usefulness of PATTY for paraphrasing the relations of DBpedia and YAGO2. In terms of runtimes, he most expensive part is the pattern extraction, where we identify pattern candidates through dependency parsing and perform entity recognition on the entire corpus. This phase runs about a day for Wikipedia a cluster. All other phases of the PATTY system take less than an hour. All experimental data is available on our Web site at http://www.mpi-inf.mpg.de/yago-naga/patty/ .

7.9.2 Precision of Relational Patterns

To assess the precision of the automatically mined patterns (patterns in this section always mean pattern synsets), we sampled the PATTY taxonomy for each combination of input corpus and type system. We ranked the patterns by their statistical strength (Section 4), and evaluated the precision of the *top 100* pattern synsets. Several human judges were shown a sampled pattern synset, its type signature, and a few example instances, and then stated whether the pattern synset indicates a valid relation or not. Evaluators checked the correctness of the type signature, whether the majority of patterns in the synset is reasonable, and whether the instances seem plausible. If so, the synset was flagged as meaningful. The results of this evaluation are shown in column four of Table 3, with a 0.9-confidence Wilson score interval [8]. In addition, the same assessment procedure was applied to *randomly sampled* synsets, to evaluate the quality in the long tail of patterns. The results are shown in column five of Table 3. For the top 100 patterns, we achieve above 90% precision for Wikipedia, and above 80% for 100 random samples.

Corpus	Types	Patterns	Top 100	Random
NYT	YAGO2	86,982	0.89±0.06	0.72±0.09
	Freebase	809,091	0.87 ±0.06	0.71±0.09
WKP	YAGO2	350,569	0.95±0.04	**0.85**±0.07
	Freebase	1,631,531	0.93±0.05	0.80±0.08

Table 3: Precision of Relational Patterns

From the results we make two observations. First, Wikipedia patterns have higher precision than those from the New York Times corpus. This is because some the language in the news corpus does not express relational information; especially the news on stock markets produced noisy patterns picked up by PATTY. However, we still manage to have a precision of close to 90% for the top 100 patterns and around 72% for random sample on the NYT corpus. The second observation is that the YAGO2 type system generally led to higher precision than the Freebase type system. This is because YAGO2 has finer grained, ontologically clean types, whereas Freebase has broader categories with a more liberal assignment of entities to categories.

7.9.3 Precision of Subsumptions

We evaluated the quality of the subsumptions by assessing 100 top-ranked as well as 100 randomly selected subsumptions. As shown in Table 4, a large number of the subsumptions are correct. The Wikipedia-based PATTY taxonomy has a random-sampling-based precision of 75%.

Corpus	Types	# Edges	Top 100	Random
NYT	YAGO2	12,601	0.86±0.07	0.68±0.09
	Freebase	80,296	0.89±0.06	0.41±0.09
WKP	YAGO2	**8,162**	0.83±0.07	**0.75**±0.07
	Freebase	20,339	0.85±0.07	0.62±0.09

Table 4: Quality of Subsumptions

Example subsumptions from Wikipedia are:

- ⟨*person*⟩ *nominated for* ⟨*award*⟩ ⊐ ⟨*person*⟩ *winner of* ⟨*award*⟩
- ⟨*person*⟩ *' s wife* ⟨*person*⟩ ⊐ ⟨*person*⟩ *'s widow* ⟨*person*⟩

7.9.4 Coverage

To evaluate the coverage of PATTY, we would need a complete ground-truth resource that contains all possible binary relations between entities. Unfortunately, there is no such resource[2]. We tried to approximate such a resource by manually compiling all binary relations between entities that appear in Wikipedia articles of a certain domain. We chose the domain of popular music, because it offers a plethora of non-trivial relations (such as *addictedTo(person,drug)*, *coveredBy(musician,musician)*, *dedicatedSongTo(musician,entity)*). We considered the Wikipedia articles of five musicians (Amy Winehouse, Bob Dylan, Neil Young, John Coltrane, Nina Simone). For each page, two annotators hand-extracted all relationship types that they would spot in the respective articles. The annotators limited themselves to relations where at least one argument type is ⟨*musician*⟩. Then we formed the intersection of the two annotators' outputs (i.e., their agreement) as a reasonable gold standard for relations identifiable by skilled humans. In total, the gold-standard set contains 163 relations.

We then compared our relational patterns to the relations included in four major knowledge bases, namely, YAGO2, DBpedia (DBP), Freebase (FB), and NELL, limited to the specific domain of music. Table 5 shows the absolute number of relations covered by each resource. For PATTY, the patterns were derived from the Wikipedia corpus with the YAGO2 type system.

gold standard	PATTY	YAGO2	DBP	FB	NELL
163	**126**	31	39	69	13

Table 5: Coverage of Music Relations

PATTY covered 126 of the 163 gold-standard relations. This is more than what can be found in large semi-curated knowledge bases such as Freebase, and twice as much as Wikipedia-infobox-based resources such as DBpedia or YAGO offer. Some PATTY examples that do not appear in the other resources at all are:

- ⟨*musician*⟩ *PRP idol* ⟨*musician*⟩ for the relation *hasMusicalIdol*
- ⟨*person*⟩ *criticized by* ⟨*organization*⟩ for *critizedByMedia*
- ⟨*person*⟩ *headliner* ⟨*artifact*⟩ for *headlinerAt*
- ⟨*person*⟩ *successfully sued* ⟨*person*⟩ for *suedBy*
- ⟨*musician*⟩ *wrote hits for* ⟨*musician*⟩ for *wroteHitsFor*,

This shows (albeit anecdotically) that PATTY's patterns contribute added value beyond today's knowledge bases.

[2]Lexical resources such as WordNet contain only verbs, but not binary relations such as *is the president of*. Other resources are likely incomplete.

	Reverb-style patterns	PATTY without types	PATTY full
# Patterns	5,996	184,629	**350,569**
Patterns Precision	0.96±0.03	0.74±0.08	**0.95**±0.04
# Subsumptions	74	15,347	**8,162**
Subsumptions Precision	0.79 ±0.09	0.58±0.09	**0.83**±0.07
# Facts	192,144	6,384,684	**3,890,075**
Facts Precision.	0.86 ±0.07	0.64±0.09	**0.88** ±0.06

Table 6: Results for Different Pattern Language Alternatives

Relation	Paraphrases	Precision	Sample Paraphrases
DBPedia/artist	83	0.96±0.03	[adj] studio album of, [det] song by ...
DBPedia/associatedBand	386	0.74±0.11	joined band along, plays in ...
DBPedia/doctoralAdvisor	36	0.558±0.15	[det] student of, under * supervision ...
DBPedia/recordLabel	113	0.86±0.09	[adj] artist signed to, [adj] record label ...
DBPedia/riverMouth	31	0.83±0.12	drains into, [adj] tributary of ...
DBPedia/team	1,108	0.91±0.07	be * traded to, [prp] debut for ...
YAGO/actedIn	330	0.88±0.08	starred in * film, [adj] role for ...
YAGO/created	466	0.79±0.10	founded, 's book ...
YAGO/isLeaderOf	40	0.53±0.14	elected by, governor of ...
YAGO/holdsPoliticalPosition	72	0.73±0.10	[prp] tenure as, oath as ...

Table 7: Sample Results for Relation Paraphrasing

7.9.5 Pattern Language Alternatives

We also investigated various design alternatives to the PATTY pattern language. We looked at three main alternatives: the first is verb-phrase-centric patterns advocated by ReVerb [12], the second is the PATTY language without type signatures (just using sets of n-grams with syntactic generalizations), and the third one is the full PATTY language. The results for the Wikipedia corpus and the YAGO2 type system are shown in Table 6; precision figures are based on the respective top 100 patterns or subsumption edges. We observe from these results that the type signatures are crucial for precision. Moreover, the number of patterns, subsumptions and facts found by verb-phrase-centric patterns (ReVerb [12]), are limited in recall. General pattern synsets with type signatures, as newly pursued in this paper, substantially outperform the verb-phrase-centric alternative in terms of pattern and subsumption recall while yielding high precision.

7.9.6 Extrinsic Study: Relation Paraphrasing

To further evaluate the usefulness of PATTY, we performed a study on relation paraphrasing: given a relation from a knowledge base, identify patterns that can be used to express that relation. Paraphrasing relations with high-quality patterns is important for populating knowledge bases and counters the problem of semantic drifting caused by ambiguous and noisy patterns.

We considered relations from two knowledge bases, DBpedia and YAGO2, focusing on relations that hold between entities and do not include literals. PATTY paraphrased 225 DBpedia relations with a total of 127,811 patterns, and 25 YAGO2 relations with a total of 43,124 patterns. Among these we evaluated

a random sample of 1,000 relation paraphrases. Table 7 shows precision figures for some selected relations, along anecdotic example patterns.

Some relations are hard to capture precisely. For *DBPedia/doctoralAdvisor*, e.g., PATTY picked up patterns like "worked with" as paraphrases. These are not entirely wrong, but we evaluated them as false because they are too general to indicate the more specific doctoral advisor relation.

Overall, however, the paraphrasing precision is high. Our evaluation showed an average precision of **0.76**±0.03 across all relations.

7.10 Conclusion and Future Directions

This paper presented PATTY, a large resource of text patterns. Different from existing resources, PATTY organizes patterns into synsets and a taxonomy, similar in spirit to WordNet. Our evaluation shows that PATTY's patterns are semantically meaningful, and that they cover large parts of the relations of other knowledge bases. The Wikipedia-based version of PATTY contains 350,569 pattern synsets at a precision of 84.7%, with 8,162 subsumptions, at a precision of 75%. The PATTY resource is freely available for interactive access and download at http://www.mpi-inf.mpg.de/yago-naga/patty/.

Our approach harnesses existing knowledge bases for entity-type information. However, PATTY is not tied to a particular choice for this purpose. In fact, it would be straightforward to adjust PATTY to using surface-form noun phrases rather than disambiguated entities, as long as we have means to infer at least coarse-grained types (e.g., person, organization, location). An interesting future direction is to study this generalized setting. We would also like to investigate the enhanced interplay of information extraction and pattern extraction, and possible applications for question answering.

References

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. "Mining Association Rules between Sets of Items in Large Databases". In: *SIGMOD Conference*. 1993.

[2] Enrique Alfonseca, Marius Pasca, and Enrique Robledo-Arnuncio. "Acquisition of instance attributes via labeled and related instances". In: *SIGIR*. 2010.

[3] Ion Androutsopoulos and Prodromos Malakasiotis. "A Survey of Paraphrasing and Textual Entailment Methods". In: *Journal of Artificial Intelligence Research 38, 135–187*. 2010.

[4] Sören Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *ISWC*. 2007.

[5] Michele Banko et al. "Open Information Extraction from the Web". In: *IJCAI*. 2007.

[6] Jonathan Berant, Ido Dagan, and Jacob Goldberger. "Global Learning of Typed Entailment Rules". In: *ACL*. 2011.

[7] Kurt D. Bollacker et al. “Freebase: a collaboratively created graph database for structuring human knowledge”. In: *SIGMOD Conference*. 2008.

[8] Lawrence D. Brown, T.Tony Cai, and Anirban Dasgupta. “Interval Estimation for a Binomial Proportion”. In: *Statistical Science 16, 101–133*. 2001.

[9] Andrew Carlson et al. “Toward an Architecture for Never-Ending Language Learning”. In: *AAAI*. 2010.

[10] Timothy Chklovski and Patrick Pantel. “VerbOcean: Mining the Web for Fine-Grained Semantic Verb Relations”. In: *EMNLP*. 2004.

[11] Christiane Fellbaum (Editor). “WordNet: An Electronic Lexical Database”. In: *MIT Press*. 1998.

[12] Anthony Fader, Stephen Soderland, and Oren Etzioni. “Identifying Relations for Open Information Extraction”. In: *EMNLP*. 2011.

[13] Jiawei Han and Yiwen Yin Jian Pei. “Mining frequent patterns without candidate generation”. In: *SIGMOD*. 2000.

[14] Chikara Hashimoto et al. “Large-Scale Verb Entailment Acquisition from the Web”. In: *EMNLP*. 2009.

[15] Catherine Havasi, Robert Speer, and Jason Alonso. “ConceptNet 3: a Flexible and Multilingual Semantic Network for Common Sense Knowledge”. In: *RANLP*. 2007.

[16] Iris Hendrickx et al. “SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations Between Pairs of Nominals”. In: *ACL International Workshop on Semantic Evaluation*. 2010.

[17] Johannes Hoffart et al. “YAGO2: Exploring and Querying World Knowledge in Time and Space and Context and and Many Languages”. In: *WWW*. 2011.

[18] Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. “Learning 5000 Relational Extractors”. In: *ACL*. 2010.

[19] Vigo Kann. “On the approximability of NP-complete optimization problems”. In: *PhD thesis and Department of Numerical Analysis and Computing Science and Royal Institute of Technology and Stockholm*. 1992.

[20] Karin Kipper et al. “A Large-scale Classification of English Verbs”. In: *Language Resources and Evaluation Journal, 42(1), 21-40*. 2008.

[21] Zornitsa Kozareva and Eduard H. Hovy. “Learning Arguments and Supertypes of Semantic Relations Using Recursive Patterns”. In: *ACL*. 2010.

[22] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. “Annotating and Searching Web Tables Using Entities and Types and Relationships”. In: *PVLDB 3(1), 1338-1347*. 2010.

[23] Dekang Lin and Patrick Pantel. “DIRT: discovery of inference rules from text”. In: *KDD*. 2001.

[24] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. “Generating Typed Dependency Parses from Phrase Structure Parses”. In: *LREC*. 2006.

[25] Thahir Mohamed and Tom M. Mitchell Estevam R. Hruschka Jr. "Discovering Relations between Noun Categories". In: *EMNLP*. 2011.

[26] Preslav Nakov and Marti A. Hearst. "Solving Relational Similarity Problems Using the Web as a Corpus". In: *ACL*. 2008.

[27] Vivi Nastase et al. "WikiNet: A Very Large Scale Multi-Lingual Concept Network". In: *LREC*. 2010.

[28] Roberto Navigli and Simone Paolo Ponzetto. "BabelNet: Building a Very Large Multilingual Semantic Network". In: *ACL*. 2010.

[29] Marius Pasca and Benjamin Van Durme. "Weakly-Supervised Acquisition of Open-Domain Classes and Class Attributes from Web Documents and Query Logs". In: *ACL*. 2008.

[30] Marius Pasca and Benjamin Van Durme. "What You Seek Is What You Get: Extraction of Class Attributes from Query Logs". In: *IJCAI*. 2007.

[31] Andrew Philpot, Eduard Hovy, and Patrick Pantel. "The Omega Ontology". In: *Ontology and the Lexicon and Cambridge University Press*. 2008.

[32] Simone Paolo Ponzetto and Michael Strube. "Deriving a Large-Scale Taxonomy from Wikipedia". In: *AAAI*. 2007.

[33] Joseph Reisinger and Marius Pasca. "Latent Variable Models of Concept-Attribute Attachment". In: *ACL/AFNLP*. 2009.

[34] Ramakrishnan Srikant and Rakesh Agrawal. "Generalizations and Performance Improvements". In: *EDBT*. 1996.

[35] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: a Core of Semantic Knowledge". In: *WWW*. 2007.

[36] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. "SOFIE: a self-organizing framework for information extraction". In: *WWW*. 2009.

[37] Petros Venetis et al. "Recovering Semantics of Tables on the Web". In: *PVLDB 4(9), 528-538*. 2011.

[38] Fei Wu and Daniel S. Weld. "Automatically refining the wikipedia infobox ontology". In: *WWW*. 2008.

[39] Wentao Wu et al. "Towards a Probabilistic Taxonomy of Many Concepts". In: *Technical Report MSR-TR-2011-25, Microsoft Research*. 2011.

[40] Limin Yao et al. "Structured Relation Discovery using Generative Models". In: *EMNLP*. 2011.

Chapter 8

Watermarking for Ontologies

8.1 Overview

In this chapter, we study watermarking methods to prove the ownership of an ontology. Different from existing approaches, we propose to watermark not by altering existing statements, but by removing them. Thereby, our approach does not introduce false statements into the ontology. We show how ownership of ontologies can be established with provably tight probability bounds, even if only parts of the ontology are being re-used. We finally demonstrate the viability of our approach on real-world ontologies. This chapter is based on

> Fabian M. Suchanek, David Gross-Amblard, Serge Abiteboul
> "Watermarking for Ontologies"
> (International Semantic Web Conference (ISWC) 2011)

8.2 Introduction

An ontology is a formal collection of world knowledge. Creating an ontology usually involves a major human effort. In the case of manually constructed ontologies, human effort is needed to collect the knowledge, to formalize it and to maintain it. The same applies to ontologies constructed by a community, such as Freebase or DBpedia. In the case of automatically constructed ontologies, human effort comes in the form of scientific investigation and the development of algorithms. Consequently, the creators of an ontology usually do not give away the ontology for free for arbitrary use. Rather, they request their users to pay for the content, to follow the terms of a specific license, or to give credit to the creators of the ontology. In most cases, it is prohibited to re-publish the data, or allowed only with proper acknowledgment.

This restriction is most obvious in the case of commercially sold ontologies such as [9]: The use of the data is restricted by the sale contract. The contract usually prohibits the re-publication of the data. Any dissemination of the data into other data sets constitutes a breach of contract.

One might think that the picture would be different for the public ontologies of the Semantic Web. The general spirit of the Semantic Web wants data to be shared across application and community boundaries[1]. However, even the ontologies of the Semantic Web are not available for arbitrary re-publication. Table 1 shows some popular ontologies mentioned together with their licenses. None of the ontologies is available in the public domain. All of them require at least an acknowledgment when their data is re-published. It is considered dishonest to sell or re-publish the data from an ontology elsewhere without giving due credit to the original.

Some data sets are not freely available at all (see Table 1). The Wolfram Alpha data set[2], for example, can be queried through an API, but cannot be downloaded. Its terms of use prohibit the systematic harvesting of the API to re-create the data set. Any re-publication of a substantial portion of such data constitutes a breach of the terms of use. Similar observations apply to trueknowledge[3] or the commercial version of Cyc [9]. In all of these cases, the extraction and systematic dissemination of the data is prohibited.

License	Conditions	Ontologies
GFDL	attribution, copyleft	DBpedia [2]
GPL	attribution, copyleft	SUMO [10]
CC-BY	attribution	YAGO [15], Freebase, Geonames, OpenCyc [9]
CC-BY-ND	attribution, no derivatives	UniProt
–	access under restrictions	TrueKnowledge, KnowItAll, WolframAlpha, full Cyc [9]

Table 1: Common licenses for ontologies

This raises the issue of how we can detect whether an ontology has been illegally re-published. We call a person who re-publishes an ontology (or part of it) in a way that is inconsistent with its license an *attacker*. The attacker could, e.g., re-publish the ontology under his own name or use parts of the ontology in his own ontology without giving due credit. We call the source ontology the *original ontology* and the re-published ontology the *suspect ontology*. We want to solve the problem of ownership proof: How can we prove that the suspect ontology contains part of the original ontology? Obviously, it is not sufficient to state that the suspect ontology contains data from the original ontology. This is because ontologies contain world knowledge, that anybody can collect. Take the example of an ontology about scientists: the attacker could claim that he also collected biographies of scientists and that he happened to produce the same data set as the original ontology. A similar argument applies to ontologies that have been derived from public sources. YAGO [15] and DBpedia [2], e.g., have both been extracted from Wikipedia. An attacker on DBpedia could claim that he also extracted data from Wikipedia and happened to produce a similar output.

One might be tempted to assume that we could simply publish the original ontology with a time stamp (in the style of proof of software ownership). For example, we could upload the ontology to a trusted external server. If someone

[1] http://www.w3.org/2001/sw/
[2] http://www.wolframalpha.com/
[3] http://www.trueknowledge.com/

else publishes the same data later, then we could point to the original copy. However, this does not prove our ownership. The other publisher could have had the data before we published ours. The fact that he did not publish his data cannot be used against him.

Therefore, a more sophisticated approach is needed to enable ownership proofs. This is the goal of the present paper. We present an approach that uses digital watermarking to detect whether a suspect ontology contains data derived from an original ontology. Watermarking techniques aim at hiding some relevant information in a data set, in an invisible or robust way. Finding such information in a suspect data set acts as the proof of ownership. Several works consider watermarking for relational databases by performing voluntarily alteration of data. These approaches could be extended to ontologies. However, data alteration invariably decreases the precision of the ontology.

Therefore, we develop an alternative method that is specifically adapted to the Semantic Web: We propose to watermark an ontology by removing carefully selected statements before publishing. The suspect absence of these statements in an ontology with a significant overlap will act as the proof of theft. We argue that this does less harm than altering statements, because the Semantic Web operates under the Open World Assumption: Most ontologies are incomplete.

More specifically, our contributions are as follows:

1. A formalization of the problem of ontological data re-publication,

2. An algorithm for watermarking ontologies, which allows detecting malicious re-publication without harming the precision,

3. Extensive experiments that show the validity of our approach.

The rest of this paper is structured as follows: Section 2 summarizes related work. Section 3 formalizes our scenario and lists different attack models. Section 4 presents our watermarking algorithm with a formal analysis. Section 5 details our experiments before Section 6 concludes.

8.3 Related Work

In [4], the authors introduce named graphs as a way to manage trust and provenance on the Semantic Web. Named graphs, however, cannot guard against the misuse of ontologies that are publicly available.

One of the oldest attempts to prove ownership of factual data is the use of fictitious entries in dictionaries. Since the 19th century, dictionaries, maps, encyclopedias and directories have had occasional fake entries. The New Columbia Encyclopedia, for example, contained an entry about a fictitious photographer called Lillian Virginia Mountweazel. If such an entry ever appeared in another encyclopedia, it was clear that the data was copied. To mark an ontology, however, it is not sufficient to add a single isolated entity, because an attacker can simply remove unconnected entities.

A classical way to achieve ownership proofs is to apply watermarking techniques. Some recent proposals have targeted ontologies [6]. This previous effort uses a purely syntactical rewriting of the RDF XML source layout into an equivalent layout to hide information. This approach can be circumvented by

normalizing the XML file. This can be done, e.g., by loading the file into a tool such as Protégé [11] and then saving it again as an XML document.

Quite a number of approaches have targeted semi-structured data [14] and relational databases [1, 13, 7, 8, 12]. These works provide one way to prove ownership of ontologies. Most of them are blind, i.e., they do not require the original data set for detection. However, all of these approaches presume that the schema of the data is known. This is not necessarily the case on the Semantic Web. Some ontologies (such as DBpedia or YAGO) have hundreds of relationships. An attacker just has to map some of them manually to other relationships to obscure the theft. We will develop approaches that can still identify the suspect data, but in a non-blind way. Furthermore, the classical methods work by voluntarily altering data. Therefore, we call these methods *modification approaches* in the sequel. Such approaches could, e.g., change the birth year of Elvis Presley from the year 1935 to the year 1936. While the introduced errors are only gradual for numerical data, they are substantial for categorical data. Such an approach could, e.g., change Elvis' nationality from American to Russian. Apart from the fact that an ontology owner will find it controversial to voluntarily alter clean data, such an approach will also decrease the precision of the ontology with respect to the real world. Furthermore, the altered facts can lead to contradictions with other data sets. This is not only annoying to the user, but can also allow the attacker to detect and remove the altered facts. Therefore, we present an alternative approach in this paper, which works by deleting carefully selected facts. Since the Semantic Web operates under the open world assumption, the absence of true information is less harmful than the presence of false information.

8.4 Model

8.4.1 Watermarking

A watermarking protocol is a pair of algorithms $(\mathcal{M}, \mathcal{D})$, where $\mathcal{M}$ stands for the marker and $\mathcal{D}$ the detector (Figure 2). Given an original ontology O and a secret key $\mathcal{K}$, the marker algorithm outputs a watermarked ontology $O^* = \mathcal{M}(O, \mathcal{K})$. Given a suspect ontology O', the original ontology and the secret key, the detector decides if O' contains a mark, that is if $\mathcal{D}(O', O, \mathcal{K})$ is `true`.

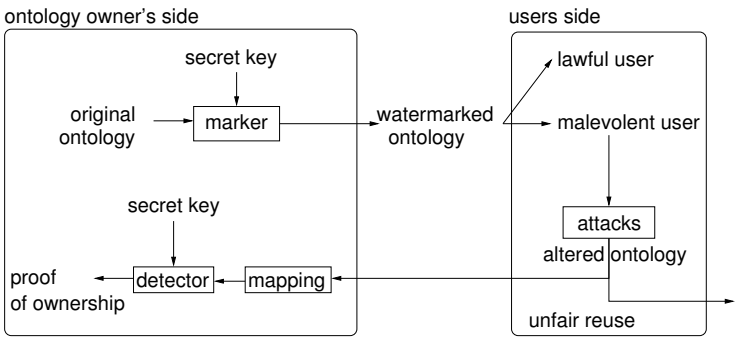

Figure 2: Watermarking protocol for ontologies

If O' contains the mark, then it is assumed that O' has indeed been derived from O^*. However, the watermarking protocol may erroneously say that O' has been derived from O^*, even though O' just contains the mark by chance. For example, O' may be a totally unrelated ontology. In this case, O' is called a *false positive*. Watermarking protocols are designed so that the probability of a false positive is provably below a confidence threshold ξ. The parameter ξ is called the *security parameter* of the protocol and is typically in the order of $\xi = 10^{-6}$.

In an adversarial setting, the attacker can try to evade the detection of the mark in the ontology by various means, including:

- **Subset attack:** The attacker uses only a subset of the stolen ontology. For example, the attacker could choose a certain thematic domain from the original ontology (such as, say, sports), and re-use only this portion. While publishing just a few statements should be allowed, larger stolen parts should be detected by the protocol.

- **Union attack:** The attacker merges the ontology with another one. For example, the attacker could combine two ontologies about gene expressions, thereby hiding the stolen ontology in a larger data set. A naive union attack would keep facts from both ontologies, even if they are inconsistent with each other.

- **Intersection attack:** The attacker keeps only ontology elements that are found in another ontology. For example, an attacker could remove entities from the stolen ontology that do not appear in a reference ontology.

- **Comparison attack:** The attacker compares the ontology to another data source and eliminates inconsistent information. For example, the attacker could cross-check birth dates of one ontology with the birth dates in a reference ontology and remove inconsistent dates.

- **Random alteration attack:** The attacker alters values, relations or entities. This action is limited as the attacker still desires valuable data.

The ability of the watermarking protocol to withstand these attacks is called its *robustness*. Watermarking protocols are typically designed so that the probability of a successful attack is provably below their security parameter ξ.

An attacker may also publish someone else's ontology as part of an offensive or illegal data set. An attacker may, e.g., take an ontology about chemistry and publish it as part of a data set about weapons of mass destruction. If the ontology was marked, it will appear as if the creator of the original ontology authored the offensive data set. Our method can fend off this so-called *copy attack*.

8.4.2 Ontologies

An RDFS ontology over a set of entities (resources and literals) E and a set of relation names R can be seen as a set of triples $O \subset E \times R \times E$. Each triple is called a *statement*, with its components being called *subject*, *predicate* and *object*. We will write *e.name* to refer to the identifier of the entity in the ontology. This can be the URI (in case of a resource) or the string value (in case

of a literal). RDFS specifies certain *semantic rules*. These rules state that if the ontology contains certain statements, certain new statements should be added to the ontology. We call these added statements *redundant* and assume that redundant statements have been removed from the ontology [5]. RDFS rules are strictly positive. Therefore, an RDFS ontology cannot become inconsistent if a statement is removed or altered. This predestines RDFS ontologies for watermarking and we leave more sophisticated ontology models for future work.

To prove that a suspect ontology O' copied data from an original ontology O, we will first have to determine whether O' and O contain similar data. This is a challenging task in itself, because it amounts to finding a mapping from O' to O. This problem has attracted a lot of research in the context of the Web of Linked Data [3]. Finding such a mapping is outside the scope of the present paper. Here, we limit ourselves to 2 assumptions: (1) We know a partial mapping function σ, which maps (some of) the entities of O' to entities of O; (2) there exists a partial mapping function σ_R, which maps the relations of O' to relations of O. Note that we do not need to know σ_R. Our method has been designed so that it is transparent to the "schema" of the ontologies. We need the existence of σ_R just to assure that O and O' have some structural similarity. An investigator needs to find only σ. If the ontology was truly stolen, then the entities will probably be recognizable in some way, because otherwise the stolen ontology would be less useful. In the best case, σ will reflect mainly syntactic variations of the identifiers and values. We note that the modification approaches like [1] also require σ, because they determine the tuples to mark based on the value of the primary key, which has to be available at detection. In addition, the modification approaches also require the mapping of the schema, σ_R, which our approach does not require. In the sequel, we assume that σ has been found and has been applied to O'. We are now concerned with the question that follows: We want to prove that O' and O are similar not just by chance, but because O' copied data from O.

8.4.3 Ethical considerations

Ontologies represent world knowledge. Therefore, it is questionable whether one can "own" such data. Can the creator have a copyright on the ontology, given that it is nothing more than a collection of facts about the world? In this paper, we do not deal with the legal implications of owning or copying ontologies. We only provide a method to prove that one data source copied from another source – independently of whether such behavior is considered legal or not.

Our approach will remove facts from the ontology before publishing. Then the question arises whether it is honest to withhold information that one could publish. However, an ontology always represents only part of reality. In most cases, the creator of an ontology is not obliged to make it exhaustive. The Open World Assumption of the Semantic Web makes the absence of information a normal and tolerable circumstance.

In general, the watermarking of an ontology always remains a trade-off between the ability to prove ownership and the truthfulness of the data. One should not willfully alter ontologies that are highly sensitive to even small misrepresentations of reality, such as ontologies in the domain of medicine, security, or aeronautics. However, unlike the modification approaches [1], our approach of deleting facts can be applied even if truthfulness of the data has to be preserved,

as long as some incompleteness can be tolerated.

8.5 Watermarking Ontologies

8.5.1 Watermarking Basics

Our starting point is a watermarking protocol from Agrawal et al. [1] for relational databases. Their statistical watermarking uses cryptographically secure pseudo-random number generators (CSPRNGs). A CSPRNG is a function which, given an integer seed value, produces a sequence of pseudo-random bits. The bits are pseudo-random in the sense that, without the seed, and given the first k bits of a random sequence, there is no polynomial-time algorithm that can predict the $(k+1)$th bit with probability of success better than 50% [16]. A CSPRNG is a *one-way-function*. This means that, given a CSPRNG and a sequence of random bits it produced, it is close to impossible to determine the seed value that generated it. More formally, given a CSPRNG f, for every randomized polynomial time algorithm A, $Pr[f(A(f(x))) = f(x)] < \frac{1}{p(n)}$ for every positive polynomial $p(n)$ and sufficiently large n, assuming that x is chosen from a uniform distribution [17]. In the following, we use a given CSPRNG $\mathcal{G}$.

After $\mathcal{G}$ has been seeded with a value by calling $\mathcal{G}.seed(value)$, one can repeatedly call the function $\mathcal{G}.nextBit()$ to obtain the next bit in the random sequence. By combining multiple calls to this function, we can construct a pseudo-random integer value. The function that delivers a pseudo-random integer number greater or equal to 0 and below a given upper bound k is denoted by $\mathcal{G}.nextInt(k)$.

Our watermarking algorithm makes use of a *secret key*. A secret key is a integer number that is only known to the owner of the original ontology. We will use the key as a seed value for $\mathcal{G}$. We also make use of a *cryptographically secure hash function*. A hash function is a function which, given an object, returns an integer number in a certain range. A cryptographically secure hash function is such that it is infeasible to find two inputs with the same output, or the inverse of the output. We assume a given hash function, such as SHA, denoted *hash*. In order to resist the aforementioned copy attack, we will first compute the secure hash of the original ontology. All our subsequent computations will depend on this hash, so that no attacker can pretend to own the watermarked version by finding another ontology with the same watermarking.

8.5.1.1 Algorithm

Our watermarking method shall be transparent to relation names, because we do not want to require an investigator to find a mapping of the schema. Therefore, we define the notion of a *fact pair*. A fact pair of an ontology O is a pair of entities $\langle e_1, e_2 \rangle$, such that there exists r such that $\langle e_1, r, e_2 \rangle \in O$. Algorithm 5 marks an ontology by removing fact pairs. It takes as input the original ontology O, a secret key $\mathcal{K}$, and the number of facts $delTotal$ to remove. The secret key is an arbitrary chosen number. Section 8.5.2.1 will discuss how to find a suitable value for $delTotal$. For each fact pair of the ontology, the algorithm seeds $\mathcal{G}$ with the names of the entities, the hash of O, and $\mathcal{K}$. If the next random integer of $\mathcal{G}$ between 0 and the total number of fact pairs in O divided by $delTotal$

happens to be 0, the fact pair is removed. This removes $delTotal$ facts from the ontology[4]. Note that our algorithm does not consider the relation names at all. After running Algorithm 5, the marked ontology O^* is published. The original ontology O is kept secret.

Algorithm 5 subtractiveMark(orig. ontology O, secret key $\mathcal{K}$, integer $delTotal$)

$O^* \leftarrow O$
for all $\langle e_1, e_2 \rangle \in \pi_{subject,object}(O)$ **do**
  $\mathcal{G}.seed(e_1.name \oplus e_2.name \oplus hash(O) \oplus \mathcal{K})$
  if $\mathcal{G}.nextInt(|\pi_{subject,object}(O)|/delTotal) = 0$ **then**
    Remove $\langle e_1, *, e_2 \rangle$ from O^*
  end if
end for
return O^*

To detect whether a suspect ontology stole data from an original ontology, we run Algorithm 6 on the original ontology O (without the mark) and the suspect ontology O' (after having applied the mapping from Section 8.4.2). The algorithm runs through all fact pairs of O and computes the proportion of published fact pairs that appear in the suspect ontology. It also computes the proportion of removed fact pairs that appear in the suspect ontology. Since we only consider fact pairs and not facts, a mapping of the relation names (the schema) is not necessary. It is possible that the suspect ontology contains some of the fact pairs that we removed from the original before publishing. This can be for two reasons: Either the suspect ontology is innocent and just happens to have a thematic overlap with our original, or the suspect ontology imported data from other sources, thus complementing the facts we removed. The algorithm then compares the ratio of removed fact pairs that appear in O' to the ratio of published fact pairs that appear in O. If O' is innocent, these ratios should be the same. If the ratio of deleted facts is lower than the ratio of published facts, and significantly so, this indicates a theft and the algorithm will return *true*. It seems highly counter-intuitive that the absence of a fact should prove theft of data. Yet, the proof comes from the fact that the removed statements form a pattern of present and absent facts in O. The probability that this pattern appears by chance in another ontology is extremely low.

Let us detail the check of significance. The suspect ontology will cover a certain portion of facts of the original ontology. The central observation is that, if this overlap is by chance, then the suspect ontology should cover the same portion of published facts as it covers of the deleted facts, because the watermarking is randomized. Thus, we have to determine whether any difference between these two ratios is statistically significant. This is the last step in Algorithm 6. This significance is determined by a χ^2 test. Be $pubTotal$ the total number of published fact pairs. Be $pubFound$ the number of published fact pairs that appear in the suspect ontology. Be $delTotal$ the total number of deleted fact pairs. Be $delFound$ the number of deleted fact pairs that appear in the suspect ontology and be $N = pubTotal + delTotal$ the total number of

[4]If less than $delTotal$ facts got removed, we rerun the algorithm with a different key.

Algorithm 6 subtractiveDetect(original O, suspect O', key $\mathcal{K}$, integer $delTotal$)

$O^* \leftarrow subtractiveMark(O, \mathcal{K}, delTotal)$
$pubFound \leftarrow 0; delFound \leftarrow 0;$
for all $\langle e_1, e_2 \rangle \in \pi_{subject,object}(O)$ **do**
  if $\langle e_1, e_2 \rangle \in \pi_{subject,object}(O')$ **then**
    if $\langle e_1, e_2 \rangle \in \pi_{subject,object}(O^*)$ **then** $pubFound++$
    else $delFound++$
  end if
end for
$pubTotal \leftarrow |\pi_{subject,object}(O^*)|$
if $delFound/delTotal \geqslant pubFound/pubTotal$ **then return false return** $delFound/delTotal$ significantly different from $pubFound/pubTotal$

fact pairs. We get

$$\chi^2 = \frac{N(delFound \times (pubTotal - pubFound) - pubFound \times (delTotal - delFound))^2}{(pubFound + delFound) \times (N - pubFound - delFound) \times delTotal \times pubTotal}.$$

If $\chi^2 > \chi^2(1, \xi)$, where ξ is the security parameter, then the two ratios are not independent. Since the removal of the fact pairs was purely random, any significant difference between the ratios indicates a dependence on the original ontology. In the extreme case, the suspect ontology reproduces the published facts and omits all (or nearly all) removed facts.

In order to be applicable, the standard χ^2 test requires the total number of samples to be greater than 100 and the expected number of samples for each case to be greater than 5. Therefore, our algorithm returns *true* iff $N > 100$ and $(pubFound + delFound) \times delTotal/N > 5$ and $\chi^2 > \chi^2(1, \xi)$.

8.5.2 Analysis

8.5.2.1 Impact

We are interested in how many fact pairs we have to remove in order to achieve significance in the χ^2 test. This number depends on the total number of fact pairs N. It also depends on the types of attacks against which we want to protect. The first property of an attack is the overlap ratio of found facts, $\omega = (pubFound + delFound)/N$. We choose $\omega = 1$ if we want to protect only against a theft of the complete ontology. We choose a smaller value if we want to protect also against a theft of a sub-portion of the ontology. The second property of an attack is the ratio of removed facts $\delta = delFound/delTotal$ that appear in the stolen ontology. If the attacker just republishes our published ontology, $\delta = 0$. If he adds data from other sources, this can complement some of the facts we removed. Ratio δ should be the proportion of removed facts that we expect in the stolen ontology. If δ is larger, and ω is smaller, the protection is safer, but the marking will remove more facts. Abbreviating $delTotal = d$, this yields $pubFound = \omega N - \delta d, delFound = \delta d, pubTotal = N - d$ and thus

$$\chi^2 = \frac{N((\omega N - \delta d)(1 - \delta)d - \delta d((1 - \omega)N - (1 - \delta)d))^2}{\omega N(1 - \omega)Nd(N - d)}.$$

For $\chi^2 > \chi^2(1,\xi)$, this yields

$$d > \frac{N\omega(1-\omega)\chi^2(1,\xi)}{N(\delta(1-\omega)-\omega(1-\delta))^2+\omega(1-\omega)\chi^2(1,\xi)}.$$

We have to impose $N > 100$ as a precondition for the χ^2 test. We also have to impose $d > 5/(\omega(1-\omega))$, i.e., $d > 20$ in the worst case. Finally, $\delta < \omega$, because we cannot prove theft if the ratio of appearing deleted facts is greater than the ratio of appearing published facts. As an example, take a choice of $\xi = 10^{-6}$, which leads to $\chi^2(1,\xi) = 23.9284$. Assuming an overlap ratio of $\omega = \frac{1}{2}$, a fault tolerance of $\delta = 0.2$, and $N = 30 \times 10^6$ fact pairs, we get $d = 67$, i.e., 67 fact pairs have to be deleted from the ontology.

8.5.2.2 Robustness

In general, the χ^2 test tends to err on the safe side, concluding independence only in the presence of overwhelming evidence. Thus, our algorithm will signal theft only in very clear cases ("in dubio pro reo"). However, our algorithm is also well-protected against attacks. First, a marked ontology is protected against intersection attacks. Intersection attacks can happen, e.g., when an attacker wants to misuse someone else's ontology to clean up his own noisy data set. Since an intersection does not add facts, our marks survive such an attack. The marked ontology is also protected against comparison attacks, because the ontologies we target generally suffer from incompleteness. Thus, a fact that is absent in the original ontology but present in a reference ontology will not raise suspicions. A marked ontology is also safe against subset attacks, if ω is chosen smaller than the proportion of stolen facts. A union attack, in contrast, could add information that fills up some of the removed facts. In this case, the marks will be reduced to those portions of the ontology that do not appear in the other ontology. By choosing $1-\delta$ equal to the portion that we estimate to be proper to the ontology, and adjusting d accordingly, we can still guard against the union attack. Random alteration attacks fall into the scope of the classical analysis of robustness [1]: an attacker being ignorant on the positions of the deleted facts can only try at random to delete more facts or fill missing ones. For this attack to be successful, a large number of facts have to be altered, a much larger number than the watermark algorithm used. Finally, an attacker can try to modify the fact pairs. Deleted facts are of course not sensitive to this attack, but the number *pubFound* can be altered. However, as the subset of published facts we are looking for at detection are chosen pseudo-randomly, the attacker has no way to locate them efficiently. The only valid strategy for the attacker is again to alter a huge amount of fact pairs, which reduces drastically the quality of the stolen ontology.

8.5.2.3 Comparison to Modification Watermarking

The modification approach [1] changes a fact from the ontology. In the majority of cases, it will change a correct fact to an incorrect fact. Thereby, precision invariably suffers. In contrast, our approach does not decrease the precision of the ontology at all. To see this, assume that O contains n statements, c of which are correct. If a correct fact is deleted, which will happen in $\frac{c}{n}$ of the cases, the precision drops to $\frac{c-1}{n-1}$. If an incorrect fact is deleted, the precision increases

to $\frac{c}{n-1}$. Thus, on average, the precision is $\frac{c}{n} \times \frac{c-1}{n-1} + \frac{n-c}{n} \times \frac{c}{n-1}$, which is $\frac{c}{n}$, just as before. As a comparison, the modification approaches have an average impact of $\frac{c}{n} \times \frac{c-1}{\mathbf{n}} + \frac{n-c}{n} \times \frac{c}{\mathbf{n}}$ (a modified correct fact turns incorrect, and a modified incorrect fact will still be incorrect, while the number of total facts is the same).

Now let us consider the number of facts that have to be modified in the classical approach. Assuming that $\delta = 0, \xi = 10^{-6}$ and $N = 30 \times 10^6$, we used the estimates in [1] to compute the number of tuples (fact pairs, in our setting) that are required to be modified in order to resist a subset attack of parameter ω. This leads to the numbers in Table 3.

ω	50%	10%	5%	2.5%	0.5%	0.05%
Removing	24	215	456	935	4775	47900
Modifying	96	480	950	1900	9500	95000

Table 3: Number of facts that have to be marked.
$\alpha = 0, \xi = 10^{-6}$ **and** $N = 30 \times 10^6$

The modification method hides a list of 0 and 1 on secretly chosen positions. If such a position is not selected by the subset attack, the marked bit is lost, whatever its value. But for our method, the subset attack has no impact on already deleted facts. Therefore, the modification approach has to modify more fact pairs than we have to delete. Overall, the number of facts that have to be modified is comparable to the number of facts that have to be deleted. Given that removal maintains precision, while modification does not, and that modification yields false facts, the ontology owner might decide to remove instead of to modify.

8.6 Experiments

8.6.1 Applicability

8.6.1.1 Impact

We were interested in how applicable our method is to real world ontologies. For this purpose, we collected 5 ontologies from the Semantic Web that cover a wide range of complexity, size, and topics (Table 4): The core part of YAGO [15], the manually supervised part of DBpedia [2], the Universal Protein Resource[5], an ontology about city finances (provided by the UK government[6]), and a subset of the IMDb[7]. For each ontology, we report the number of facts, fact pairs, relations and instances. We also report the number of entities that are connected to an object by more than one relation. This number is usually small, except for YAGO, where every entity has a label and a (mostly equivalent) preferred name. We compute the number of facts that have to be removed to protect against various subset attacks ($\xi = 10^{-6}$). As a comparison, the last column gives the number of alterations needed for the modification approach [1]. It is roughly independent of the size of the data set. Values for the modification method are

[5] http://www.uniprot.org/
[6] http://data.gov.uk/
[7] http://imdb.com/

not given for $\delta = 10\%$, because the scenario where the attacker irons out the marks has not been considered in [1].

	YAGO	DBpedia	UniProt	Finance	IMDb	*Modification*
# relations	83	1,107	4	11	12	
# instances	2,637,878	1,675,741	1,327	1,948	4,657,880	
# facts	18,206,276	19,788,726	6,096	14,926	34,699,697	
dup. objects	3,529,697	450,171	0	0	14,907	
# fact pairs	14,676,579	19,338,555	6,096	14,926	34,685,090	
Facts to remove						
$\delta = 0, \omega = 50\%$	24	24	24	24	24	[97]
$\delta = 0, \omega = 5\%$	456	456	424	442	456	[975]
$\delta = 0, \omega = 0.5\%$	4,775	4,774	2,677	3,618	4,775	[9700]
$\delta = 0, \omega = 0.05\%$	47,820	47,825	-	-	47,909	[97500]
$\delta = 10\%, \omega = 50\%$	37	37	37	37	37	N/A

Table 4: Marking different ontologies

8.6.1.2 Removing the Marks

We were interested in how an attacker could try to identify the missing facts in order to reinstate them. The attacker could, e.g., compare all instances of a class and see whether some of them lack a relation that all the other instances have. This entity is suspect from an attackers point of view, because he has to assume that we deleted that relation. More precisely, we call an entity e *suspect* in an ontology O, if there exists a class c, an entity e', and a relation r, such that

$$e \in c, e' \in c, |\{e'' : \langle e', r, e'' \rangle\}| > |\{e'' : \langle e, r, e'' \rangle\}|.$$

We call a fact *discreet* if we can remove it without creating a suspect entity. We computed the proportion of discreet facts, their relations and the proportion of instances with at least one discreet fact (Table 5). Roughly half of the facts are discreet, meaning that we can delete them without raising suspicions. Even if the attacker correctly identifies the fact we removed, he cannot simply discard the entity, because this would still keep the mark. Instead, he has to find the correct value for the missing link to plug in the hole. This may be hard or even close to impossible, because the attacker has no access to the original ontology and cannot run the detection algorithm. Also, he does not know the ratio of discreet facts. Furthermore, from the attacker's point of view, nearly all instances are suspect on the original data set already. Thus, nearly every instance could potentially have been marked. Filling up what could be a hole would amount to reconstructing the ontology. The only exception to this rule is the finance data set, which contains rather complete information (most entities have the maximal number of links in their class). We note, however, that removing facts might still be preferable to modifying facts in this ontology.

	YAGO	DBpedia	UniProt	Finance	IMDb
discreet instances	99%	92%	74%	69%	97%
discreet facts	74%	86%	48%	39%	75%
discreet relations	96%	99%	50%	45%	92%
suspect instances	99%	99%	100%	75%	100%

Table 5: The ontologies from an attackers' point of view

8.6.1.3 False Positive Detection

A risk with watermarking approaches is the occurrence of false positives. The probability of considering a non-marked ontology suspect has to be bounded. While this poses a problem for blind watermarking methods that do not rely on the original for detection, it is very unlikely that a random ontology is signaled as stolen by our approach, because our approach first checks that the overlap of the suspect ontology with the original ontology is significant. Even in the unlikely situation of a large overlap, the innocent suspect ontology will match with published facts and deleted facts uniformly. Thus the ratio of deleted and found facts will be similar, leading to a correct non-detection. This risk is taken into account and formalized in the significance level of the χ^2 test. We provide next some subset attacks whose overlap is so small that our method does not signal theft.

8.6.2 Robustness

8.6.2.1 Attack Simulations

To demonstrate the robustness of our watermarking, we simulated suspect ontologies that overlap only partially with the original ontology. This partial overlap could be due to an incomplete mapping σ or due to the fact that the attacker chose to steal only a subset of the original. We watermarked the Finance ontology with 30 removed facts. This should make the mark resistant to a partial overlap of $\omega = 50\%$ or more. We varied ω and created 10 random overlaps for each value of ω. Figure 6 shows the average rate of successful detection ($\xi = 10^{-6}, \delta = 0$). As predicted, any suspect ontology that overlaps more than half is identified as stolen.

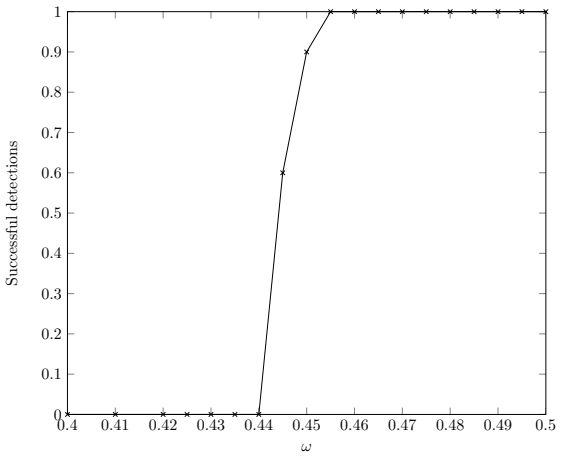

Figure 6: Rate of successful detection with varying ω, ($\delta = 0$)

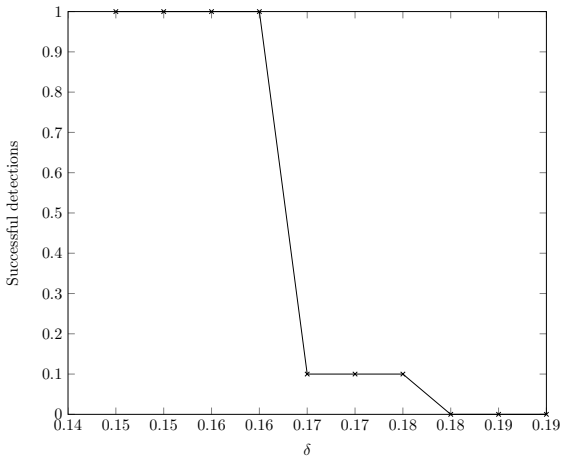

Figure 7: Rate of successful detection with varying δ ($\omega = 0.5$)

We also simulate suspect ontologies that do contain portions of the deleted facts. This can be the case if the attacker merged the stolen ontology with another data set. We removed 50 facts from the Finance ontology. At an overlap of $\omega = 50\%$, this should protect the ontology up to $\delta = 15\%$. We varied δ and simulated 10 random suspect ontologies for each value of δ. Figure 7 shows that the rate of successful detection. As expected, the rate is 1 for $\delta \leqslant 15\%$.

8.6.2.2 Thematic Subset Attacks

Next, we analyze *thematic subset attacks*, i.e., attacks that steal a certain class with all its instances and all their facts. We call such a set facts a *theme*. Table 8 shows populous classes in YAGO together with their number of fact pairs. We computed the ratio ω and the number of fact pairs that would have to be deleted in total (at $\xi = 10^{-6}$). The numbers in brackets show the number of fact pairs that the modification method would consume. Table 9 shows the same characteristics for populous classes in DBpedia. The number of facts to delete fades in comparison to YAGO's 15m fact pairs in total and DBpedia's 19m fact pairs in total. We experimentally verified the subset attacks on DBpedia with 1000 removed facts, achieving significance levels of $\chi^2 = 68, 18, 38, 51, 38$, respectively. This means that, with 1000 removed facts, we could detect all thematic subset attacks except for the one on Television Episodes, because the subset is too small for the chosen number of marks. This experiment confirms our predictions.

Theme	# fact pairs	ω	Fact pairs to remove	
			$\delta = 0$	$\delta = 1\%$
Person	10,594,790	72.19%	20 [67]	20
Village	582,984	3.97%	580 [1,225]	1,037
Album	588,338	4.01 %	574 [1,215]	1,020
Football player	1,200,459	8.18 %	269 [596]	350
Company	400,769	2.73%	854 [1,785]	2,129

Table 8: Protecting different themes in YAGO

Theme	# fact pairs	ω	Fact pairs to remove	
			$\delta = 0$	$\delta = 1\%$
Album	120,7561	6.24%	360 [782]	511
TelevisionEpisode	342,277	1.77%	1,331 [2750]	7,035
Village	701,084	3.63%	637 [1345]	1,213
SoccerPlayer	924,299	4.78%	478 [1020]	764
Film	694,731	3.59%	644 [1360]	1,238

Table 9: Protecting different themes in DBpedia

8.6.2.3 Union Attacks

We wanted to evaluate how our method works if an attacker merges the stolen ontology with another, possibly similar ontology. This might fill up some of the removed facts and thus destroy our marks. We simulated attacks that steal a certain theme from DBpedia and merge it into YAGO. We merged by matching resources (DBpedia and YAGO use the same local identifiers with different prefixes), matching identical strings and identical numbers, and matching numbers that share the value (ignoring the unit). This yields an overlap of 1.6×10^6 fact pairs between the two ontologies.[8] This overlap is 8% of DBpedia. This means that 8% of the marks that we add to DBpedia can be filled up by merging with YAGO. Hence, we have to choose $\delta > 8\%$ in order to protect against a theft.

Table 10 shows the ontologies obtained from merging YAGO with a certain theme from DBpedia. The table shows the total number of fact pairs of these ontologies as well as the absolute and relative overlap with the original DBpedia. The relative overlap corresponds to ω. The last column shows the number of fact pairs that have to be removed from DBpedia to protect the theme, calculated from ω and $\delta = 8\%$.

YAGO + DBpedia Theme	# fact pairs	overlap with DBpedia	overlap as % of DBpedia (=ω)	Fact pairs to remove
Album	15,920,534	2,831,716	15%	624
TelevisionEpisode	15,108,014	2,019,289	10%	5398
Village	15,399,557	2,310,784	12%	1583
SoccerPlayer	15,585,500	2,496,744	13%	1085
Films	15,369,042	2,280,321	12%	1583

Table 10: Merging different themes of DBpedia into YAGO

We experimentally verified these theoretical results by marking DBpedia with the removal of 2000 facts. Then, we stole different themes from the marked DBpedia and merged them into YAGO. We ran our detection algorithm and obtained significance levels of $\chi^2 = 28, 15, 34, 47$, and 31, respectively. This means that we could successfully detect all thefts, except for the theft of the

[8] Part of the reason for the small overlap is the rather crude mapping (YAGO normalizes numbers to SI units, while DBpedia does not). However, manual inspection also shows that YAGO knows many types for the entities, many labels, and some facts that DBpedia does not know. DBpedia, in turn, captures many infobox attributes that YAGO does not capture. The ontologies share just 1.4 million instances.

Television Episode theme. This set is smaller, so that it requires the removal of more facts, as predicted. This shows that our method works even if part of the mark is destroyed.

8.7 Conclusion

We have presented an alternative approach for the watermarking of ontologies. Instead of altering facts, we remove facts. Thereby, we do not lower the precision of the ontology. We have shown that even on large ontologies, only a few hundred facts have to be removed to guarantee protection from theft. Through experiments, we have shown that our approach is well applicable to real world ontologies. In the future, we intend to explore whether ontologies can also be watermarked by adding artificial facts.

8.8 Acknowledgements

This work was supported by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant Webdam, agreement 226513 (http://webdam.inria.fr/).

References

[1] Rakesh Agrawal, Peter J. Haas, and Jerry Kiernan. "Watermarking Relational Data: Framework, Algorithms and Analysis". In: *VLDB J.* 12.2 (2003), pp. 157–169.

[2] Sören Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *Proceedings of the International Semantic Web Conference (ISWC)*. Vol. 4825. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2007, pp. 722–735.

[3] Christian Bizer et al. "Linked data on the Web". In: *Proceedings of the International Conference on World Wide Web (WWW)*. New York, NY, USA: ACM, 2008, pp. 1265–1266.

[4] Jeremy J. Carroll et al. "Named graphs, provenance and trust". In: *WWW '05: Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA: ACM, 2005, pp. 613–622.

[5] Stephan Grimm and Jens Wissmann. "Elimination of Redundancy in Ontologies". In: *ESWC*. 2011, pp. 260–274.

[6] Hongbin Kong et al. "Techniques for OWL-based Ontology Watermarking". In: *WRI Global Congress on Intelligent Systems (GCIS)*. Xiamen, 2009, pp. 582–586.

[7] Julien Lafaye et al. "Watermill: An Optimized Fingerprinting System for Databases under Constraints". In: *IEEE Trans. Knowl. Data Eng. (TKDE)* 20.4 (2008), pp. 532–546.

[8] Yingjiu Li, Vipin Swarup, and Sushil Jajodia. "Fingerprinting Relational Databases: Schemes and Specialties". In: *IEEE Trans. Dependable Sec. Comput. (TDSC)* 2.1 (2005), pp. 34–45.

[9] Cynthia Matuszek et al. "An introduction to the syntax and content of Cyc". In: *Proceedings of the AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*. Menlo Park, CA, USA: AAAI Press, 2006, pp. 44–49.

[10] Ian Niles and Adam Pease. "Towards a standard upper ontology". In: *Proceedings of the international conference on Formal Ontology in Information Systems*. New York, NY, USA: ACM, 2001, pp. 2–9.

[11] N.F. Noy, R.W. Fergerson, and M.A. Musen. "The knowledge model of Protégé-2000: Combining interoperability and flexibility". In: *Lecture Notes in Computer Science* 1937 (2000), pp. 69–82.

[12] Mohamed Shehab, Elisa Bertino, and Arif Ghafoor. "Watermarking Relational Databases Using Optimization-Based Techniques". In: *IEEE Trans. Knowl. Data Eng. (TKDE)* 20.1 (2008), pp. 116–129.

[13] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. "Protecting Rights over Relational Data using Watermarking". In: *IEEE Trans. Knowl. Data Eng. (TKDE)* 16.12 (Dec. 2004), pp. 1509–1525.

[14] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. "Resilient Information Hiding for Abstract Semi-Structures". In: *Proceedings of the 4th Workshop on Digital Watermarking, IWDW'03*. Ed. by Springer Verlag. Vol. 2939. 2003, pp. 141–153.

[15] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Core of Semantic Knowledge - Unifying WordNet and Wikipedia". In: *Proceedings of the International Conference on World Wide Web (WWW)*. Ed. by Carey L. Williamson, Mary Ellen Zurko, and Prashant J. Patel-Schneider Peter F. Shenoy. Banff, Canada: ACM, 2007, pp. 697–706.

[16] Wikipedia. *CSPRNG*. 2011.

[17] Wikipedia. *One-way function*. 2011.

Chapter 9

Conclusion

Summary. This thesis has summarized 6 publications that address challenges in the mining, linking, and extension of ontologies. Our work on AMIE allowed mining the ontologies for semantic rules. The work on ANGIE and SUSIE allowed for the integration of Web services into the ontologies. The PARIS project developed an approach to align two ontologies. PATTY extends an ontology by natural language phrases. Finally, we have seen an approach to watermark ontologies. These projects have addressed some of the research challenges that we encounter with large ontologies.

Outlook. Research on information extraction and the management of ontologies has made huge progress during the last decade. However, many challenges remain. One issue is scaling the methods to the size of the Web. Computers become ever more powerful, but we also produce ever more data. Currently, the growth rate of data outpaces the advancement rate of computers. Therefore, new methods will have to be developed to extract information at scale, to integrate it with the existing information, and also to make use of large-scale semantic data. It is not just the size of the data that poses a challenge, but also the different types of information that we encounter. Social media, such as Twitter, Blogs, or Facebook, have seen a rise in recent years. The public parts of these sources could be harvested for ontologies. Finally, new applications for semantic data will be explored. This includes its use in search, translation, decision making, or education. Ultimately, our goal is to make computers ever more useful for mankind.