On Computing Deltas of RDF Knowledge Bases with Blank Nodes

Christina Lantzaki

Thesis submitted in partial fulfillment of the requirements for the
Masters’ of Science degree in Computer Science
University of Crete
School of Sciences and Engineering
Computer Science Department
Knossou Av., P.O. Box 2208, Heraklion, GR-71409, Greece

Thesis Advisor: Prof. Yannis Tzitzikas
UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

On Computing Deltas of RDF/S Knowledge Bases with Blank Nodes

Thesis submitted by
Christina Lantzaki
in partial fulfillment of the requirements for the
Masters’ of Science degree in Computer Science

THESIS APPROVAL

Author:  
Christina Lantzaki

Committee approvals:
Yannis Tzitzikas
Assistant Professor, Thesis Supervisor

Dimitris Pleksousakis
Professor, Committee Member

Georgios Georgakopoulos
Associate Professor, Committee Member

Departmental approval:
Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, February 2013
Abstract

The Semantic Web (SW) is an evolving extension of the World Wide Web in which the content can be expressed not only in natural language, but also in formal languages (e.g. RDF/S) that can be read and used by software agents, permitting them to find, share and integrate information more easily. The semantically structured content is expressed using RDF triples and a set of such triples constitute an RDF Knowledge Base (KB), or equivalently an RDF Graph.

The statement of Heraclitus “Everything flows, nothing stands still” holds also in the context of the SW since everything changes (the resources themselves, the ontologies, the resource descriptions, etc). Consequently, the ability to compute the differences, hereafter Delta, that exist between two RDF KBs is very important. In particular, Deltas can be employed to (a) aid humans understand the evolution of knowledge, and (b) reduce the amount of data that need to be exchanged and managed over the network in order to build SW synchronization, versioning and replication services.

The comparison problem becomes complicated, because RDF allows anonymous nodes. A anonymous node, else called blank node, is a node in an RDF graph which is not identified by a URI and is not a literal. Several RDF KBs rely heavily on blank nodes; e.g. 7.5% of Linked Data are estimated to be blank nodes, while in well known datasets (e.g. rdfabout.com) the percentage reaches 40%. From a functional perspective, blank nodes are convenient for representing complex attributes or resources whose identity is unknown but their properties are known. Considering blank nodes as "constants" unique to both graphs does not help either in detecting equivalence between graphs nor in reducing the Delta. On the contrary, matching the blank nodes of the two graphs can significantly reduce the produced Delta.

This work is the first study approaching the methods of matching blank nodes as an optimization problem. The optimization aims at finding the mapping that yields the minimum in size Delta (i.e. with the least number of triples to delete or add to make the graphs equivalent). We prove that in the general case finding the optimal blank node mapping is NP-Hard by reducing it to the sub-graph isomorphism problem. When graphs do not contain directly connected blank nodes (i.e. no triples with more than one blank nodes exists), we show that the polynomial Hungarian algorithm can be used to find the optimal blank node mapping. For the general case we present various polynomial algorithms returning approximate solutions. One of these algorithms is a variation of the optimal Hungarian. For making the application of our method feasible also to KBs with a heavy load of blank nodes we present a signature-based mapping algorithm with NlogN time complexity.

Finally, for the proposed algorithms we report extensive comparative experimental results, over real and synthetic KBs, regarding delta reduction (and its deviation from the optimal), equivalence detection, and computational requirements. The results are very interesting; indicatively the algorithms produce a
Delta of 12 to 7,000 times smaller than without blank node matching. The signature-based algorithm yields up to 0.34 times bigger Deltas than the Hungarian, but is two orders of magnitude faster. Note that it requires less than 11 seconds to match 150,000 pairs of blank nodes.
Περίληψη

Ο Σημασιολογικός Ιστός είναι μια εξελισσόμενη επέκταση του Παγκόσμιου Ιστού στην οποία το περιεχόμενο μπορεί να εκφραστεί όχι μόνο με φυσική γλώσσα, αλλά και με τυπικές γλώσσες (όπως η RDF/S), που χαρακτηρίζουν εφικτή την παροχή προηγμένων υπηρεσιών ανάζητησης, διαμοιρασμού και ολοκλήρωσης πληροφοριών. Το σημασιολογικά δομημένο περιεχόμενο, εκφράζεται με τριπλέτες RDF, και ένα σύνολο τέτοιων τριπλετών ορίζει μία Βάση Γνώσης RDF (ΒΓ), ή ισοδύναμα έναν RDF γράφο.

Η ρήση του Ηράκλειτου «τα πάντα ρει» ισχύει και στο Σημασιολογικό Ιστό, αφού οι πόροι (resources), οι οντολογίες, τα μεταδεδομένα διαρκώς αλλάζουν και εξελίσσονται. Εκ τούτου η ικανότητα υπολογισμού της διαφοράς μεταξύ δύο ΒΓ είναι πολύ σημαντική. Η διαφορά αυτή, στο εξής Δέλτα, μπορεί (α) να βοηθήσει τους χρήστες στην κατανόηση της εξέλιξης της γνώσης, και (β) να μειώσει τον όγκο των δεδομένων που χρειάζεται να ανταλλαχθούν και διαχειριστούν στο δίκτυο για την ανάπτυξη υπηρεσιών συγχρονισμού, διαχείρισης εκδοτών και αντιγράφων.

Το πρόβλημα της σύγκρισης περιπλέκεται, διότι η RDF υποστηρίζει ανώνυμους κόμβους. Ένας ανώνυμος κόμβος είναι ένας κόμβος σε ένα RDF γράφο, ο οποίος δεν είναι ούτε URI, ούτε literal και επομένως δεν έχει εξωτερική ταυτότητα. Αρκετές ΒΓ περιλαμβάνουν μεγάλο ποσοστό ανώνυμων κόμβων. Χαρακτηριστικά το 7.5% των Διασυνδεδεμένων Δεδομένων εκτιμάται ότι αντιστοιχεί σε ανώνυμους κόμβους, ενώ σε γνωστές ΒΓ (όπως η rdfabout.com) το ποσοστό φτάνει ως και το 40%. Από λειτουργική άποψη, οι ανώνυμοι κόμβοι είναι κατάλληλοι για την αναπαράσταση σύνθετων γνωρισμάτων ή πόρων των οποίων η ταυτότητα είναι άγνωστη, αλλά οι ιδιότητες τους είναι γνωστές. Η θεώρηση των ανώνυμων κόμβων σαν σταθερά, μοναδικές και στους δύο γράφους, δε βοηθάει τους χρήστες στην κατανόηση της εξέλιξης της γνώσης, και (β) να αντιμετωπίσουν τα διαφορές που χρειάζεται να αντιλαμβανθούν και διαχειριστούν στο δίκτυο για την ανάπτυξη υπηρεσιών συγχρονισμού, διαχείρισης εκδοτών και αντιγράφων.

Η παρούσα εργασία είναι η πρώτη μελέτη, που αντιμετωπίζει την αντιστοίχιση των ανώνυμων κόμβων ως πρόβλημα βελτιστοποίησης. Η βελτιστοποίηση αφορά στην εύρεση της αντίστοιχης που οδηγεί στο ελάχιστο σε μέγεθος Δέλτα. Επιπλέον αποδεικνύουμε ότι οι γράφοι δεν περιέχουν ανώνυμους κόμβους που συνδέονται άμεσα μεταξύ τους (ήτοι, καμία τριπλέτα δεν περιέχει περισσότερους από έναν ανώνυμο κόμβο), τότε ο πολυωνυμικός πολυπλοκότητας Ουγγρικός αλγόριθμος μπορεί να δώσει τη βέλτιστη λύση. Για τη γενική περίπτωση προτείνουμε διάφορους πολυωνυμικούς αλγορίθμους, που μπορούν να λύσουν το πρόβλημα προσεγγιστικά. Για να είναι εφαρμοστέος ή εφαρμοστέας την μειωμένη λύση, θα πρέπει να γίνεται τη βέλτιστη λύση. Για τη γενική περίπτωση προτείνουμε διάφορους πολυωνυμικούς αλγορίθμους, που μπορούν να λύσουν το πρόβλημα προσεγγιστικά.
Τέλος, για τους προτεινόμενους αλγόριθμους, αναφέρουμε εκτεταμένα συγκριτικά πειραματικά αποτελέσματα επί πραγματικών και συνθετικά παραγομένων ΒΓ, σχετικά με τις επιδόσεις τους στη μείωση του Δέλτα (και την απόκλιση τους από το βέλτιστο), στον εντοπισμό ισοδύναμων γράφων, και τις υπολογιστικές απαιτήσεις. Τα αποτελέσματα είναι πολύ ενδιαφέροντα: ενδεικτικά, οι αλγόριθμοι παράγουν Δέλτα από 12 μέχρι και 7,000 φορές μικρότερα απ’ ό,τι χωρίς αντιστοίχιση των ανώνυμων κόμβων. Ο αλγόριθμος που βασίζεται στις υπογραφές παράγει μέχρι και 0.34 φορές μεγαλύτερα Δέλτα από τον Ουγγρικό, αλλά είναι δύο τάξεων γρηγορότερος. Να σημειώσουμε ότι απαιτεί λιγότερο από 11 δευτερόλεπτα για να αντιστοιχίσει 150,000 ζευγάρια ανώνυμων κόμβων.
Acknowledgements

First of all, I would like to thank my supervisor, professor of the University of Crete, Yannis Tzitzikas, initially for trusting me and then for his continuous support and his valuable advice. His organizational capability was really helpful for me, too. I am also grateful to the professors Dimitris Pleksousakis and Georgios Georgakopoulos for participating in the supervisory committee.

I would also like to thank the Computer Science Department of Greece for offering a high level of academic education and the Information-Systems Laboratory of ICS-FORTH for its support in terms of graduate fellowship and for providing a high-levelled research environment.

Last but not least, I would like to thank my friends and my family for supporting me all over this period.

This work was partly supported by the NoE APARSEN (Alliance Permanent Access to the Records of Science in Europe, FP7, Proj. No 269977, 2011-2014), and the FP7 Research Infrastructures projects SCIDIP-ES (SCience Data Infrastructure for Preservation - Earth Science, 2011, 2014), and iMarine (FP7 Research Infrastructures, 2011-2014).
Στη συνειδητοποίηση της ανάγκης για απελευθέρωση
# Contents

1 Introduction .......................................................... 3  
  1.1 The Semantic Web: Concepts and Vision ......................... 3  
    1.1.1 The Semantic Web Stack: Components .................... 4  
  1.2 Linked Data .................................................. 6  
  1.3 Motivation for matching blank nodes when comparing KBs ...... 8  
    1.3.1 Motivating Scenarios ....................................... 9  
  1.4 Contributions ................................................ 11  
  1.5 Organization of the thesis .................................... 12  

2 Background .......................................................... 13  
  2.1 Managing the evolution of Knowledge Bases ..................... 13  
    2.1.1 Versioning Services ........................................ 13  
    2.1.2 Synchronization Services .................................... 15  
    2.1.3 Replication Services ....................................... 16  
  2.2 RDF and Blank Nodes .......................................... 17  
    2.2.1 Theoretical Perspective .................................... 18  
    2.2.2 Practical Perspective ...................................... 19  
    2.2.3 Blank Nodes in published data ............................ 23  
    2.2.4 The future of blank nodes .................................. 24  

3 Related Work ...................................................... 27  
  3.1 Introduction .................................................. 27  
  3.2 Ontoview ...................................................... 28  
    3.2.1 Rules for Changes .......................................... 30  
  3.3 Promptdiff .................................................... 31  
    3.3.1 Heuristic Matchers ......................................... 33  
  3.4 Semversion ..................................................... 34  
    3.4.1 Set-based Diff .............................................. 35
3.4.2 Structural Diff and Blank nodes .................................. 35
3.4.3 Semantic Diff ..................................................... 36
3.5 x-RDF-3X ............................................................ 36
3.6 RDF_utils ............................................................ 37
3.7 CWM of w3c .......................................................... 39
    3.7.1 Patch file format .............................................. 39
    3.7.2 Weak and Strong Deltas ..................................... 40
3.8 Jena ................................................................. 40
3.9 pOWL ................................................................. 41
3.10 RDF/S Diff .......................................................... 42
3.11 Synopsis ............................................................ 43

4 On reducing RDF Delta in KBs with blank nodes 45

4.1 Introduction ......................................................... 45
4.2 Preliminaries ......................................................... 46
    4.2.1 Basic Notation ................................................ 46
    4.2.2 RDF Knowledge Bases ..................................... 46
    4.2.3 Differential Function and Change Operations ........... 48
4.3 RDF KBs with Blank Nodes .................................... 50
    4.3.1 RDF graph equivalence ....................................... 50
    4.3.2 Bnode Name Tuning ......................................... 52
    4.3.3 Delta reduction size ......................................... 52
4.4 Bnode Matching as an Optimization Problem ............... 53
    4.4.1 Problem Formulation ......................................... 53
    4.4.2 Polynomially-solved (and Frequently Occurring) Cases . 55
4.5 Approximation Algorithms ..................................... 56
    4.5.1 Hungarian BNode Matching Algorithm .................. 56
    4.5.2 A Fast ($O(N \log N)$) Signature-based Algorithm .... 57
    4.5.3 More about the Signature Construction .................. 62
    4.5.4 A Faster ($O(N \log N)$) Signature-based Algorithm .... 63
    4.5.5 Comparing the approximation algorithms ................. 66
    4.5.6 On the serialization of the Knowledge Bases .......... 67
4.6 Discussing Semantics and Inference Rules .................. 69
4.7 Desired Delta Properties and BNode Matching ............. 70
5 Experimental Evaluation

5.1 TestBed

5.1.1 Real Datasets

5.1.2 Synthetic Datasets

5.2 Evaluation: not directly connected bnodes

5.3 Evaluation: directly connected bnodes

5.3.1 On Complex Bnode structures

5.3.2 Delta Reduction Potential

5.3.3 Time Efficiency

5.3.4 Equivalence Detection Potential

5.4 Scalability

5.5 Measuring the approximation

5.6 Synopsis of the results

6 System and Applications

6.1 Functionality

6.2 Architecture

6.3 Applications

7 Discussion on open issues

7.1 Optimally solved subcases

7.2 Alternative approximation algorithms

8 Conclusions and Future Work
List of Figures

1.1 The Semantic Web Stack ................................................. 4
1.2 The Semantic Web Stack in 3D presentation .......................... 6
1.3 The Linked Open Data Cloud .......................................... 7
1.4 Two Knowledge Bases with directly connected blank nodes ....... 10

2.1 Blank node with nested elements ...................................... 19
2.2 Giving blank node an identifier ....................................... 19
2.3 Examples of blank nodes with complex attributes ..................... 20
2.4 Example of blank nodes with a Bag container ......................... 21
2.5 A simple provenance trail with blank nodes .......................... 22

3.1 Comparing two ontologies in OntoView ................................ 29
3.2 The architecture of the PromptDiff .................................... 32
3.3 The structural diff showing the difference between two versions ... 32
3.4 The Layered Architecture of SemVersion .............................. 34
3.5 The granularity of the SW ranges from the universal graph to triple 37
3.6 Three RDF graphs showing personal information from three sources 38
3.7 pOWL Versioning ......................................................... 42

4.1 What set of change operations could transform K to K’? .............. 46
4.2 Distinctions of triples sets ............................................. 48
4.3 KB without unique reduction .......................................... 48
4.4 Added and deleted triples of the differential functions ............... 49
4.5 Two Knowledge Bases with directly connected blank nodes ....... 55
4.6 Alg. The Signature-based bnode matching algorithm .................. 58
4.7 Two Knowledge Bases of an address ontology ......................... 60
4.8 Signature Construction Algorithm .................................... 61
4.9 Lookup algorithm ....................................................... 62
4.10 The second Signature-based bnode matching algorithm .............. 64
4.11 GetPair algorithm ........................................... 64
4.12 A C-like version of SignatureMapping2 .................. 65
4.13 Scenario where the AlgSign gives a smaller delta than the AlgHung . 68
4.14 Comparing the explicit versus the complete graphs of two KBs . . 70

5.1 Univ Bench Schema ........................................ 75
5.2 Delta Reduction Potential with same serialization order .......... 77
5.3 Delta Reduction Potential with reverse serialization order ...... 78
5.4 Mapping times over the synthetic datasets .................... 78
5.5 Mapping times over large synthetic datasets .................. 78
5.6 $d_x$ over non equivalent KBs ............................... 80
5.7 $d_x$ over equivalent KBs ................................... 80

6.1 Importing the KBs on BNodeDelta .......................... 82
6.2 Selecting the blank node matching algorithm on BNodeDelta ... 82
6.3 Basic Statistics of the blank node matching .................. 83
6.4 The exported files of BNodeDelta ........................... 83
6.5 Command line version of BNodeDelta ....................... 84
6.6 The Sesame component Stack ............................... 85

7.1 Two Knowledge Bases with directly connected blank nodes .... 88
7.2 GDist vs SimpleDist-based mappings ....................... 90
7.3 GDist vs SimpleDist-based mappings over two equivalent KBs .. 90

8.1 Synopsis of BNodeDelta contribution ........................ 92
List of Tables

1.1 Delta on different matching of the blank nodes . . . . . . . . . . . . 11
2.1 Top publishers of blank nodes in the corpus . . . . . . . . . . . . . 24
2.2 Tree Width distribution . . . . . . . . . . . . . . . . . . . . . . . . . . 24
4.1 Signatures on bnodes of KBs of Fig. 4.7 . . . . . . . . . . . . . . . . 59
4.2 Bnode mappings and Delta for \textit{Alg$_{Hung}$} and \textit{Alg$_{Sign}$} algorithms . . . 68
5.1 Features of two real LOD datasets . . . . . . . . . . . . . . . . . . 74
5.2 Experimental results over real datasets . . . . . . . . . . . . . . . . 75
5.3 Blank node Features of the synthetic dataset . . . . . . . . . . . . . 76
Chapter 1

Introduction

“Utopia lies at the horizon. When I draw nearer by two steps, it retreats two steps. If I proceed ten steps forward, it swiftly slips ten steps ahead. No matter how far I go, I can never reach it. What, then, is the purpose of utopia? It is to cause us to advance.”

-Eduardo Hughes Galeano

1.1 The Semantic Web: Concepts and Vision

The content of the World Wide Web (WWW) is currently formatted in a natural language, mainly through HTML. Even though such a language is human-readable and human-understandable, the machines or else the software agents are only able to read this information. The machine-intelligibility cannot be achieved with the current technology.

This gap is called to be solved through the Semantic Web, an extension of the WWW. The term was coined by the Tim Berners-Lee, the inventor of the WWW and the director of the Word Wide Web Consortium (W3C), and aims at converting unstructured and semi-structured documents into semantically structured knowledge, that can be processed directly and indirectly by machines. The promoted formats give the ability to the machines to interpret the content of the web page and find, share and integrate information more easily.
CHAPTER 1. INTRODUCTION

The Semantic Web is standardized, in the context of Web 3.0, with a set of new languages organized in a layered architecture, which is often referred to as the Semantic Web stack. Current research proves that the promoted technology is already in use and starts to reach the market.

1.1.1 The Semantic Web Stack: Components

The Semantic Web stack (Figure 1.1) is a work in progress, where the layers are developed in a bottom-up manner. The so far well-established technologies are specified as W3C standards and include RDF, RDF Schema, SPARQL and OWL.

The basic components are given below:

- XML provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within. XML is not at present a necessary component of Semantic Web technologies in most cases, as alternative syntaxes exists (such as Turtle).

- RDF (Resource Description Framework) is a simple language for expressing data models, which refers to resources (web resources) and asserts binary relationships between them. Such assertions have the form of triples, called statements. The elements of a triple are called subject, predicate and object. A collection of RDF statements intrinsically represents a labeled, directed multi-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and
other ontological models. An RDF-based model can be represented in a variety of syntaxes, such as RDF/XML, Notation3, Turtle, NTriples and RDFa. As this thesis is focused on this layer, more details are given in the following sections.

- **RDFS (RDF Schema)** is a vocabulary that extends RDF for describing properties and classes of RDF-based resources, with semantics for generalized hierarchies. In particular, the statements of RDF Schema (RDFS) make it possible to define hierarchies of classes, hierarchies of properties and to describe domains and ranges of the properties.

- **OWL (Web Ontology Language)** builds-up on RDFS introducing more expressive description constructs. In particular, OWL adds relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL and OWL Full. OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. OWL DL supports users who want the maximum expressiveness while retaining computational completeness and decidability. OWL Full is meant for users who want very high expressiveness and the syntactic freedom of RDF with no computational guarantees. In OWL 2, there are three sublanguages. OWL 2 EL is a fragment that has polynomial time reasoning complexity; OWL 2 QL is designed to enable easier access and query to data stored in databases; OWL 2 RL is a rule subset of OWL 2.

- **SPARQL** is the most prevalent RDF query language for semantic web data resources, able to retrieve and manipulate data stored in Resource Description Framework format. In general, querying allows fine-grained data access.

The rest of the components are not yet fully standardized or realized, such as the Rule Interchange Format (RIF) in the Rule Layer or the Unifying Logic and Proof Layer.

In all cases the intention of the semantic web is to enhance the usability and usefulness of the Web and its interconnected resources through servers which expose existing data systems using the RDF and SPARQL standards, through converters to RDF, through documents "marked up" with semantic information, through common metadata vocabularies (ontologies) and maps between vocab-
ularies, through automated agents to perform tasks for users or finally through web-based services to supply information specifically to agents.

Alternatively, Figure 1.2 offers a 3D presentation of the Semantic Web stack, providing more information without sacrificing compactness and simplicity. The one side gives the concepts and the abstraction of the Semantic Web and the other side gives the specification and the solutions for each concept.

![The Semantic Web Technology Stack](image)

Figure 1.2: The Semantic Web Stack in 3D presentation

Most applications just use only a subset of the described stack. Indicatively, the Linked Data (see more in section 1.2) use a small selection of the technologies and in particular the RDF and OWL technology.

### 1.2 Linked Data

Linked Data is about using the Web to connect related data that wasn’t previously linked, or using the Web to lower the barriers to linking data currently linked using other methods. More specifically, Wikipedia defines Linked Data as "a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs".

and RDF.”

Tim Berners-Lee coined the term in a design note discussing issues around the Semantic Web project. However, the idea is very old and is closely related to concepts including database network models, citations between scholarly articles, and controlled headings in library catalogs.

The goal of the W3C Semantic Web community project is to extend the Web with a data commons by publishing various open datasets as RDF on the Web and by setting RDF links between data items from different data sources. In 2007, datasets consisted of over two billion RDF triples, which were interlinked by over two million RDF links. By 2011 this had grown to 31 billion RDF triples, interlinked by around 504 million RDF links. There is also an interactive visualization of the Linked datasets to browse through the cloud. Here we provide a static visualization of the cloud, seen in Figure 1.3.

![Figure 1.3: The Linked Open Data Cloud](image)

It follows a list with some main datasets inside or outside LOD cloud.

- **CKAN**: registry of open data and content packages provided by the Open Knowledge Foundation

- **DBpedia**: a dataset containing extracted data from Wikipedia. It contains about 3.4 million concepts described by 1 billion triples, including abstracts in 11 different languages
• GeoNames: provides RDF descriptions of more than 7,500,000 geographical features worldwide

• UMBEL: a lightweight reference structure of 20,000 subject concept classes and their relationships derived from OpenCyc, which can act as binding classes to external data. It also has links to 1.5 million named entities from DBpedia and YAGO

• FOAF: a dataset describing persons, their properties and relationships

1.3 Motivation for matching blank nodes when comparing KBs

The statement of Heraclitus "Everything flows, nothing stands still" holds also in the context of the Semantic Web (SW) since everything changes; the resources themselves, the ontologies, the resource descriptions, etc. Consequently, the ability to compute the differences that exist between two RDF Knowledge Bases, hereafter called Delta, is very important. In particular, Deltas can be employed to (a) aid humans understand the evolution of knowledge, and (b) reduce the amount of data that need to be exchanged and managed over the network in order to build SW synchronization [??], versioning [??] and replication [?] services.

Although RDF Knowledge Bases (KBs) can be serialized in various text formats, a straightforward application of existing version control systems for software code (such as RCS and CVS) or for XML data (such as [?], [?] and [?]) is not a viable solution for computing RDF Deltas. This is mainly due to the fact that RDF KBs essentially represent graphs which (a) may feature several possible serializations (since there is no notion of edge ordering) and (b) can be enriched with semantics of a particular specification (also including inferred triples). For these reasons several non text-based tools have been developed for comparing graphs produced autonomously on the SW. However, existing RDF differential tools have not yet focused enough on the size of the produced deltas, a very important aspect for building versioning services over SW repositories. There are works (e.g.[?]) proposing differential functions that yield reduced in size deltas (in certain cases) but treat anonymous nodes as named nodes.

The existence of anonymous nodes makes the comparison problem much more complex. A anonymous node, else called blank node (for short bnode),
1.3. MOTIVATION FOR MATCHING BLANK NODES WHEN COMPARING KBS

is a node in an RDF graph which is not identified by a URI and is not a literal. Several RDF KBs rely heavily on blank nodes. From a functional perspective, blank nodes are convenient (a) for representing complex attributes or resources whose identity is unknown but their attributes (either literals or associations with other resources) are known, and (b) for recording activities in the life cycle of digital objects (provenance trails). The prevalence of the blank nodes becomes clear through empirical study on the published data [7]. 7.5% of Linked Data are estimated to be blank nodes. Focusing only on files containing blank nodes and looking at terms in the data-level position, it was found that 57.8% of their unique terms were blank nodes. Indicatively, the “hi5.com foaf” domain consists of 87.5% of blank nodes, while the “opencalais.com” domain, which is part of the LOD cloud, contains 44.9% blank nodes.

Considering blank nodes as "constants" unique to both graphs does not help either in detecting equivalence between graphs nor in reducing the Delta. On the contrary, matching the blank nodes of the two graphs can significantly reduce the produced delta. Some works and systems (specifically Jena [8]) focus only on deciding whether two KBs that contain blank nodes are equivalent or not, and do not offer any delta size saving for the case where the involved KBs are not equivalent. There are other works that compare non equivalent KBs that neither aim directly at reducing the delta size nor treat the blank nodes for all the cases. [9] creates an identity for each blank node but is only restricted in finding the accurately same-structured blank nodes. [10] proposes a blank node matching that presupposes that blank nodes are part of uniquely identified triples.

In brief, and to the best of our knowledge, this work is the first study that focuses on methods of matching the blank nodes of the two graphs, by approaching it as an optimization problem. The optimization aims at finding the mapping that yields the minimum in size Delta (i.e. with the least number of triples to delete or add to make the graphs equivalent). Note that finding such a mapping can be considered as a preprocessing step, a task that is carried out before a differential function (like those described in [11]) is applied.

1.3.1 Motivating Scenarios

At this subsection we are going to give a motivating scenario that will be used in the rest of this work. Consider two simple KBs $K_1$ and $K_2$ of an Agenda Ontology (Figure 1.4). These two KBs can be considered as two subsequent versions.
CHAPTER 1. INTRODUCTION

As you can see all the triples of the two graphs contain at least one blank node. As a result, if we do not apply any blank node matching, then all the triples of the first graph are going to be considered different from the triples of the second graph. The Delta is going to contain 18 change operations (the 9 triples of $K_1$ are going to be deleted and the 9 triples of the $K_2$ are going to be added).

On the other hand, if we match the blank nodes of the first graph with the blank nodes of the second graph, the produced Delta can be reduced. Table 1.1 shows different blank node mappings and the corresponding produced Delta.

The first row presents the case where no blank node matching is applied, while the second row presents the worst case blank node mapping, which is going to give the same Delta size with that of no blank node matching. The third row is based on a blank node mapping, which is coherent with the serialization of the blank nodes in the two graphs. The produced Delta is significantly reduced and it actually gives the most intuitive change operations about the kind of change that became from $K_1$ to $K_2$. However, the blank node mapping of the fourth row gives the minimum Delta.

In the next chapters we are going to focus on how we should theoretically formulate the problem so as to always get the blank node mapping with the minimum in size Delta. Then we are going to comprehend the difficulties that arise in the practical application of this problem and focus on optimally solved sub-cases and approximation algorithms.

![Two Knowledge Bases with directly connected blank nodes](image)

Figure 1.4: Two Knowledge Bases with directly connected blank nodes
1.4. CONTRIBUTIONS

This work focuses on defining theoretically the blank node matching problem, as the problem of mapping the blank nodes from one graph to another in an optimal way. The optimality refers to the size of the produced delta (the number of triples to delete or add to make the graphs equivalent) considering the mapping of their blank nodes. We prove that finding the optimal blank node mapping is NP-Hard in the general case by reducing it in terms of the sub-graph isomorphism problem. When the graphs do not contain directly connected blank nodes (i.e. no triples contain more than one blank nodes), we show that the polynomial(cubic) Hungarian algorithm can be used directly to find the optimal blank node mapping.

When the blank nodes are connected (there are triples with more than one blank nodes), we propose to use polynomial heuristics algorithms returning approximate solutions. One algorithm uses the Hungarian algorithm but considers blank nodes as variables with no cost in matching. For making the application of our method feasible also to KBs with a great load of blank nodes we present a
CHAPTER 1. INTRODUCTION

signature-based algorithm with NlogN complexity. This algorithm computes and orders signatures (encoding the edges of the blank nodes) to find blank nodes with similar neighbourhoods.

The experimental results over real and synthetic datasets showed that the proposed algorithms significantly reduce the sizes of the produced deltas, while the time required is affordable (indicatively the NlogN algorithm requires a few seconds for KBs with up to 150,000 bnodes). We provide comparative results regarding their time efficiency, their potential for delta reduction (and deviation from the optimal Delta) and equivalence detection, as well as their scalability.

1.5 Organization of the thesis

The rest of this work is organized as follows. Chapter 2 gives a description of the evolution management services, as well as the role and characteristics of blank nodes in RDF. It follows Chapter 3 with a review of the related work. Chapter 4 is the main core of this thesis. It (a) introduces the RDF Deltas and the RDF graph equivalence, (b) elaborates on the problem of finding the optimal blank node mapping, and (c) proposes approximate blank node matching algorithms. It follows Chapter 5 reporting extensive experimental results and Chapter 6 analyzing the developed system and its architecture. Then, Chapter 7 discusses some methods that have not yet fully realized or are under consideration, yet. Finally, Chapter 8 concludes the work and identifies issues for further research.

This work has resulted in the publication of the following papers:

- Blank Node Matching and RDF/S Comparison Functions, Yannis Tzitzikas, Christina Lantzaki and Dimitris Zeginis, International Semantic Web Conference, November 2012, Boston MA

- Demonstrating BlankNode Matching and RDF/S Comparison Functions, Christina Lantzaki, Yannis Tzitzikas and Dimitris Zeginis, DemoPaper International Semantic Web Conference, November 2012, Boston MA

Chapter 2

Background

“Life is a series of natural and spontaneous changes. Don’t resist them, that only creates sorrow. Let reality be reality. Let things flow naturally forward in whatever way they like.”

-Lao Tzu

2.1 Managing the evolution of Knowledge Bases

As already mentioned, Semantic Web is an evolving extension of the World Wide Web. This means that a lot of data need to be stored or exchanged over the network. Three kinds of services are exploited to cope with those needs:

- Versioning services are used to control multiple versions of the same unit of information.

- Synchronization services are used to keep remote data consistent (i.e. both of them contain the same information although something changed at one side).

- Replication services are used to ensure consistency between redundant resources (stored in multiple storage devices)

2.1.1 Versioning Services

Version control is the management of multiple revisions of the same unit of information. It is commonly used in engineering and software development to manage ongoing development of digital documents like application source code and
other critical information that may be worked on by a team of people. Changes to these documents are identified by incrementing an associated number or letter code, termed the "revision number", "revision level", or simply "revision" and associated historically with the person making the change. A simple form of revision control, for example, has the initial issue of a drawing assigned the revision number "1". When the first change is made, the revision number is incremented to "2" and so on.

In computer software engineering, revision control is any practice that tracks and provides control over changes to source code. Software developers sometimes use revision control software to maintain documentation and configuration files as well as source code. Most revision control software can use delta encoding, which retains only the differences between successive versions of files. This allows more efficient storage of many different versions of files.

Delta encoding is a way of storing or transmitting data in the form of differences between sequential data rather than complete files. Delta encoding is sometimes called delta compression, particularly where archival histories of changes are required.

The differences are recorded in discrete files called "deltas" or "diffs". Because changes are often small, delta encoding greatly reduces data redundancy. Collections of unique deltas are substantially more space-efficient than their non-encoded equivalents. From a logical point of view the difference between two data values is the information required to obtain one value from the other. The difference between identical values (under some equivalence) is often called 0 or the neutral element. A good delta should be minimal, or ambiguous unless one element of a pair is present.

A delta can be defined in two ways, symmetric delta and directed delta. A symmetric delta can be expressed as: $\Delta(v_1, v_2) = (v_1 \setminus v_2) \cup (v_2 \setminus v_1)$, where $v_1$ and $v_2$ represent two successive versions.

A directed delta, also called a change, is a sequence of (elementary) change operations which, when applied to one version $v_1$, yield another version $v_2$ (note the correspondence to transaction logs in databases).

In delta encoded transmission over a network, where only a single copy of the file is available at each end of the communication channel, special error control codes are used to detect which parts of the file have changed since its previous version.

The nature of the data to be encoded influences the effectiveness of a particu-
lar compression algorithm. Delta encoding performs best when data has small or constant variation.

### 2.1.2 Synchronization Services

In computer science, synchronization refers to one of two distinct, but related concepts:

- **Process synchronization** refers to the idea that multiple processes are to join up or handshake at a certain point, so as to reach an agreement or commit to a certain sequence of action.

- **Data synchronization** refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain data integrity. Process synchronization primitives are commonly used to implement data synchronization.

At this thesis we study the problem of data synchronization. Data synchronization is the process of establishing consistency among data on remote sources and the continuous harmonization of the data over time. It is fundamental to a wide variety of applications, including file synchronization [?], Personal Digital Assistant synchronization [?], and Public Key Server synchronization.

Several theoretical models of data synchronization exist in the research literature. The models are classified based on how they consider the data to be synchronized. Some models consider the data to be unordered while others consider the data to be ordered.

The problem of synchronizing unordered data (also known as the set reconciliation problem) is modelled as an attempt to compute the symmetric difference $S_A \oplus S_B = (S_A \setminus S_B) \cup (S_B \setminus S_A)$ between two remote sets $S_A$ and $S_B$ [?]. Some solutions to this problem are typified by:

- **Wholesale transfer.** In this case all data is transferred to one host for a local comparison.

- **Timestamp synchronization.** In this case all changes to the data are marked with timestamps. Synchronization proceeds by transferring all data with a timestamp later than the previous synchronization.

- **Mathematical synchronization.** In this case data are treated as mathematical objects and synchronization corresponds to a mathematical process.
Considering ordered data two remote strings $\sigma_A$ and $\sigma_B$ need to be reconciled. Typically, it is assumed that these strings differ by up to a fixed number of edits (i.e. character insertions, deletions, or modifications). Some solution approaches to this problem include:

- shingling - splitting the strings into shingles in order to reduce this problem into an unordered synchronization problem.[8]

- synchronizing files and directories from one location to another while minimizing data transfer using delta encoding.

2.1.3 Replication Services

Replication is one of the most important topics in the area of distributed systems. It involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

There are two kinds of replication:

- data replication if the same data is stored on multiple storage devices

- computation replication if the same computing task is executed many times

In the context of this thesis we focus on data replication. These processes are passive and operate only to maintain the stored data, reply to read requests, and apply updates.

Knowledge Base replication can be seen as an evolution of database replication. Database replication can be used on many database management systems usually with a master/slave relationship between the original and the copies. The master logs the updates, which then ripple through to the slaves. The slave outputs a message stating that it has received the update successfully, thus allowing the sending (and potentially resending until successfully applied) of subsequent updates. Multi-master replication, where updates can be submitted to any database node, and then ripple through to other servers, is often desired, but introduces substantially increased costs and complexity which may make it impractical in many situations. The most common challenge that exists in multi-master replication is transactional conflict prevention or resolution. Most synchronous or eager replication do conflict prevention, while asynchronous solutions have to do conflict resolution. For instance, if a record is changed on two nodes simultaneously, an eager replication system would detect the conflict before confirming
the commit and abort one of the transactions. A lazy replication system would allow both transactions to commit and run a conflict resolution during resynchronization. The resolution of such a conflict may be based on a time-stamp of the transaction, on the hierarchy of the origin nodes or on much more complex logic, which decides consistently on all nodes.

In terms of the mobile domain it is very unlikely that an application will access all of them online, since mobile network connectivity is not always available for reasonable prices. To overcome this problem, and to decrease response times, data from remote sources can be replicated locally, and applications can operate on these local copies. However, it is not practical to duplicate several billions of RDF triples to a mobile device with limited computing power and memory capacity, and often this is not required at all for a specific application.

The replication of data from external sources, which can be selected based on the specific application context, is relatively straightforward for read-only data.

### 2.2 RDF and Blank Nodes

As we have already mentioned, RDF is a data model that represents knowledge in the form of simple statements, called RDF triples, which consist of a subject, a predicate and an object, like a simple sentence in a human language. The subject is a thing (resource) that a statement describes, the predicate of a statement identifies a property or a relation, while the object is a value of a property or a target of a relation.

In RDF, there are three types of nodes, URI references, blank nodes and literals. URI references identify resources, blank nodes represent anonymous resources that are not assigned a URI, and literals denote values such as numbers or dates. The subject of an RDF triple may be a URI reference or a blank node, the predicate must be a URI reference, and the object may be all three kinds of all three kinds (URI references, literals, blank nodes). When combined together, RDF triples form a direct labeled graph (RDF graph). Subjects and objects of RDF triples become nodes in an RDF graph and predicates become arcs connecting them.

Although RDF is based on a simple idea, there are some problems that make it complicated. One main problem is the existence of blank nodes.
2.2.1 Theoretical Perspective

Nodes without a name represent a special kind of nodes, called blank nodes (for short bnodes). These nodes simply indicate the existence of a thing, without using or saying anything about the name of that thing. Therefore, according to the standard, they are referred to as existential variables of an RDF graph.

Due to the absence of a name (URI), manipulating data containing blank nodes is much harder. They make otherwise trivial operations (like the comparison of two KBs or the simple entailment checking) far more complex or even intractable.

On the other hand, they enable a great flexibility in expressing.

In this flexibility a more profound reason is hidden, which perhaps can explain how blank nodes have survived as a part of the RDF model all these years despite all the headaches they have caused. The thing is, blank nodes reflect a human way of referencing things. Let us show an example:

If I want to talk about my left arm, it is quite unnatural to invent a new identifier for it. I will just say "my left arm", describing it relative to myself, and a listener will understand. This is possible due to human’s ability to understand the context. He or she knows that the pronoun "my" refers to me as something unique, the arm being part of me, and the "left" finally specifying the exact arm. So, my left arm, unique in the universe, is referenced quite simply and elegantly.

In RDF, it can be expressed with the two statements as: "I have an arm. It (has a property that) is left". Let us assume that we know that the blank node is of a type "ex:Arm" implicitly through the property:

\[
\text{ex} : \text{hasArm} \quad \text{rdfs} : \text{range} \quad \text{ex} : \text{Arm}
\]

Given that my URI is http://foaf/id/3243435, and assuming the relevant properties are defined in ex ontology, we can express it with the following triples:

\[
\text{http} : \text{/foaf/id/3243435} \quad \text{ex} : \text{hasArm[}
\]

\[
\text{ex} : \text{hasProperty} \quad \text{ex} : \text{Left}]
\]

The left arm is represented by the blank node, which is the object in the first triple and the subject in the second, thus chaining them and forming a rather readable code.
2.2. RDF AND BLANK NODES

2.2.2 Practical Perspective

Notation

While in theory blank nodes do not have a name, in practice, when publishing data, they can be assigned an ID in a local graph/document scope, in order to enable several RDF triples to reference the same unidentified resource. This local identifier is called a "blank node identifier" and it is different from URIs or literals, because it does not provide a unique name in the global context. This identifier is denoted by the characters ("_ : ") followed by a string. It follows an example for better understanding of the benefits of this assignment.

Figure 2.2.2 shows some triples in RDF-XML syntax, where the blank node is represented by nested elements.

```xml
<foaf:Person rdf:about="http://example.org/Person/John">
    <foaf:knows>
        <foaf:Person foaf:birthDate="04-21">
        </foaf:Person>
    </foaf:knows>
</foaf:Person>
```

Figure 2.1: Blank node with nested elements

If the same blank node is used more than once in the same RDF graph, it can be represented by giving an identifier, like in Figure 2.2.2. So, now we can express the fact that John and Mary have a common friend.

```xml
<foaf:Person rdf:about="http://example.org/Person/John">
    <foaf:knows>
        <foaf:Person rdf:nodeID=":_b1"/>
    </foaf:knows>
</foaf:Person>
<foaf:Person rdf:about="http://example.org/Person/Mary">
    <foaf:knows>
        <foaf:Person rdf:nodeID=":_b1"/>
    </foaf:knows>
</foaf:Person>
```

Figure 2.2: Giving blank node an identifier

Functionality

According to a current paper [? ], the usage of blank nodes is attributed to the following aspects:
Blank nodes have the capability to describe "Multi-component Structures" or else "Complex Attributes". Figure 2.3 is an example of an RDF graph with such functionality. Complex attributes (e.g. an attribute address of Figure 2.3) can be represented without having to name explicitly the auxiliary node that is used for connecting together the values that constitute the complex value (i.e. the particular street, number and postal code values).

Thus, the information is described in a multi-component structure and we use blank nodes to express the existence of the information. Also in RDF, we need to describe groups of things, for example several authors of a book or some students of a team. The container is a structural concept in RDF data model. There are three types of containers: the Bag (group of resources or literals), the Sequence (group of resources or literals, where their order is significant) and the Alternative (group of resources or literals that are alternatives). These structures can be described with blank nodes, like Figure 2.4, where the characters of a book are given through a blank node.

Blank nodes have the capability to describe the Reification. Reification or else provenance is used to describe other RDF statements using the RDF format, for instance to record information about when statements were made, who made them or other similar information. For example, we have an RDF triple as:
2.2. RDF AND BLANK NODES

<table>
<thead>
<tr>
<th><a href="http://www.lib.com/bookid/3243">http://www.lib.com/bookid/3243</a></th>
<th>hasTitle</th>
<th>“ForWhomTheBellTolls”</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.lib.com/bookid/3243">http://www.lib.com/bookid/3243</a></td>
<td>hasCharacters</td>
<td>_ : chars</td>
</tr>
<tr>
<td>_ : chars</td>
<td>rdf : type</td>
<td>rdf : Bag</td>
</tr>
<tr>
<td>_ : chars</td>
<td>hasCharacter</td>
<td>“Robert Jordan”</td>
</tr>
<tr>
<td>_ : chars</td>
<td>hasCharacter</td>
<td>“Pilar”</td>
</tr>
<tr>
<td>_ : chars</td>
<td>hasCharacter</td>
<td>“Pablo”</td>
</tr>
<tr>
<td>_ : chars</td>
<td>hasCharacter</td>
<td>“Maria”</td>
</tr>
</tbody>
</table>

Figure 2.4: Example of blank nodes with a Bag container

exproducts : No001  externs : weight “2.4”xsd : decimal

Then we get the reification of the triple in a graph, whose triples are shown below:

_ : b  rdf : type  rdf : Statement
_ : b  rdf : subject  exproducts : No001
_ : b  rdf : predicate  externs : weight
_ : b  rdf : object  “2.4”xsd : decimal

The blank node _ : b is intended to refer to the original (first) triple and it is called, rather confusingly, a reified triple. We can use another triple to describe the information about the original triple, like the following:

exstore : No010  externs : publish _ : b

In other words, the statement is itself an object that can be talked about in RDF, we can associate information to this very basic atom of data, such as who made this particular statement and when. The so-called provenance trails are used in many fields of science and business to support the needs of capturing, processing, presenting and preserving data in the digital object’s life cycle.

Figure 2.5 shows a small provenance trail with two subsequent events.

- Blank nodes can hide the Unexposable Information
  In many senses the publishers may not want to expose their data completely, so the blank nodes can help them to shield some sensitive information. For example, if a shopping center wants to publish some shopping information, we can replace the real customer’s identity with the blank node:
Figure 2.5: A simple provenance trail with blank nodes

Thus the browsers outside just only can get the information about the shopping of the shopping center, but can not know any other information about the identity of the costumer. The blank nodes actually protected the inner information in a good manner.

- Blank nodes can express the "Multi-relationship"

The idea behind this functionality is that RDF can only describe binary relationships directly like $p(s,o)$. As to multi-relationship $p(s_1, s_2, ..., s_n)$ the RDF must express this using an indirected form with the help of bnodes. We do it by choosing one participant ($s_1$) as the subject of the relationship $p$ and a bnode as the object, then we create a group of relationships $p_2, p_3, ..., p_n$ to express the relationships and the participants $s_2, s_3, ..., s_n$. Figure 2.3 also describes the "multi-relationship" between a person and his address. As depicted there, the blank node is used to connect all the participants like a bridge.

Parsing methods

As there have been developed various Semantic Web frameworks, there are different parsers that treat blank nodes differently. Some parsers use methods for automatically assigning URIs (skolemization), which complicates things further. In first-order logic, Skolemization is way of removing existential quantifiers from a formula in prenex normal form (a chain of quantifiers followed by a quantifier-free formula). The central idea of Skolemization is to replace existential quantified
variables for "fresh" constants that are not used in the original formula. When the original formula does not have universal quantifiers, only constants are needed in the Skolemization process. Since only existential quantifiers are found in simple RDF graphs, we need only to talk about Skolem constants. However, if Skolemization was used to study the satisfiability of logical formulae in more expressive languages (e.g. OWL), Skolem functions would be needed. Consequently, in terms of RDF, Skolemization refers to replacing existential variables with unique constants or simply a way of assigning URIs to blank nodes.

Other parsers just assign local identifiers/labels in a systematical way. This labelling is based on the explicit blank node labels or the order of appearance (serialization order) of the blank nodes inside the imported set of triples. A combination of both of them also occurs. All the above methods are preferable than a completely arbitrary labelling.

### 2.2.3 Blank Nodes in published data

In this section we give some information on the use of blank nodes in RDF data published on the Web, according to [?]. These information were collected over a domain-agnostic sample of RDF data containing 965 MB of unique triples.

Focusing on the files containing blank nodes and looking at their terms in the data-level position of triples, it was found that the 57.8% of the unique terms were blank nodes, while only 32.2% were URIs, and 10% were literals. Each blank node had on average 5.2 data-level occurrences. It occurred, on average, 0.99 times in the object position of a non-rdf:type triple, with 3.1 MB blank nodes (1.9% of all blank nodes) not occurring in the object position. Conversely, each blank node occurred on average 4.2 times in the subject position of a triple, with 0.04% not occurring in the subject position. Thus, we get that (i) blank nodes are prevalent on the Web; (ii) most blank nodes appear in both the subject and object position, but occur most prevalently in the former. Both their functionality and possibly the tree-based RDF/XML syntax can verdict in favour of these results.

Table 2.1 lists the top ten domains in terms of publishing unique blank nodes found in their corpus. "LOD?" indicates whether the domain features in the LOD cloud diagram. Of the 783 domains contributing to that corpus, 345 (44.1%) did not publish any blank nodes. The average percentage of unique terms which were blank nodes for each domain was 7.5%, indicating that although a small number of high-volume domains publish many blank nodes, many other domains publish blank nodes more infrequently. The analogous figure including only those
CHAPTER 2. BACKGROUND

Table 2.1: Top publishers of blank nodes in the corpus

<table>
<thead>
<tr>
<th>domain</th>
<th>bnodes</th>
<th>%bnodes</th>
<th>LOD?</th>
</tr>
</thead>
<tbody>
<tr>
<td>hi5.com</td>
<td>14,409,539</td>
<td>87.5%</td>
<td>no</td>
</tr>
<tr>
<td>livejournal.com</td>
<td>8,892,569</td>
<td>56.0%</td>
<td>no</td>
</tr>
<tr>
<td>ontologycentral.com</td>
<td>2,882,803</td>
<td>86.0%</td>
<td>no</td>
</tr>
<tr>
<td>opiumfield.com</td>
<td>1,979,915</td>
<td>17.4%</td>
<td>no</td>
</tr>
<tr>
<td>freebase.com</td>
<td>1,109,485</td>
<td>15.6%</td>
<td>yes</td>
</tr>
<tr>
<td>vox.com</td>
<td>843,503</td>
<td>58.0%</td>
<td>no</td>
</tr>
<tr>
<td>rdfabout.com</td>
<td>464,797</td>
<td>41.7%</td>
<td>yes</td>
</tr>
<tr>
<td>opencalais.com</td>
<td>160,441</td>
<td>44.9%</td>
<td>yes</td>
</tr>
<tr>
<td>soton.ac.uk</td>
<td>117,390</td>
<td>19.1%</td>
<td>yes</td>
</tr>
<tr>
<td>bbc.co.uk</td>
<td>101,899</td>
<td>7.4%</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 2.1: Top publishers of blank nodes in the corpus

Regarding the structure of the blank nodes inside the graphs of the published data, we get the following information. They found 23% of all documents containing blank nodes and a 9% of all documents contained “non-reflexive” blank triples (triples with only one blank node). As for the treewidth they give Table 2.2. Notably, 98.4% of the components are acyclical, but a significant number are cyclical (treewidth greater than 1).

<table>
<thead>
<tr>
<th>Treewidth</th>
<th>#components</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>518,831</td>
</tr>
<tr>
<td>2</td>
<td>8,134</td>
</tr>
<tr>
<td>3</td>
<td>208</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.2: Tree Width distribution

2.2.4 The future of blank nodes

Recent bibliography ([? ], [? ]) proposes the reduction of blank nodes in the RDF graphs by clearing all the blank nodes that represent redundant information and the blank nodes which can be mapped into some URI references. This would help to make leaner and cleaner RDF graphs.

In Linked Data publication, it is not encouraged to describe information using blank nodes, but the common mechanisms of publishing Linked Data cannot
avoid the usage of blank nodes, as it gives the developers the aforementioned conveniences.

Discussion on blank nodes is still open, but as the amount of published data grows rapidly, a consensus is very much needed. At the same time, because of the absence of an undisputed solution, the problems of blank nodes are an inevitable reality and research, like this thesis, is orientated on studying methods to handle them.
Chapter 3

Related Work

«Είναι αδύνατο να βελτιωθεί ο κόσμος εάν πρώτα δεν βελτιωθεί ο άνθρωπος.»
- Πλάτων

This section examines state of the art tools with the same or similar orientation and points out their differences in relation to our work.

3.1 Introduction

Several non text-based tools have been recently developed for comparing RDF graphs produced autonomously on the SW as for example:

- Ontoview [? ]. Is an ontology management system, able to compare two ontology versions and highlight their differences.

- PromptDiff [? ? ]. Is an ontology-versioning environment, that includes a version-comparison algorithm (based on heuristic matchers)

- SemVersion [? ]. Proposes two Diff algorithms one structure-based and one semantic-aware

- RDF_Utils [? ]. Introduce the notion of RDF molecules as the finest components to be used when comparing RDF graphs.

- CWM of w3c [? ]. Is a general-purpose semantic web data processing tool which can compare two RDF files. It uses a functional or inverse functional properties to identify a blank nodes.
• Jena \(^1\). Is a Java framework for building Semantic Web applications. It provides a tool for checking isomorphism between two RDF graphs.

• Powl \(^2\). Is a web based ontology management tool that tracks the editing actions that are made using the system.

• RDF/S Diff \(^3\). They introduce various RDF/S differential functions which take into account inferred knowledge from an RDF/S Knowledge Base.

Existing RDF comparison tools have neither focused on the size of the produced Deltas, nor in the blank nodes matching, two very important aspects for building versioning services over SW repositories. Furthermore, the output of these tools is exploited by humans, and thus an intuitive presentation of the comparison results (and other related issues) has received considerable attention.

Finally, tracking the evolution of ontologies when changes are preformed in more controlled environments (e.g. collaborative authoring tools) has been addressed in \(^4\).

It follows an analysis over the most important tools and some annotation about the way they treat blank nodes.

### 3.2 Ontoview

OntoView \(^5\) is a web-based system\(^6\) inspired by CVS \(^7\) that helps users to manage changes in ontologies. OntoView stores the contents of the versions, metadata, conceptual relations between constructs in the ontologies and the transformations between them. The internal version management is partly based on change specifications and the versions of ontologies themselves, but also uses additional human input about the meta-data and types of changes (as described below). It allows users to differentiate between ontologies at a conceptual level and to export the differences as adaptations or transformations.

Two types of change are distinguished. There can be changes in the logical definition of a concept which are not meant to change the concept, and, the other way around, a concept can change without a change in its logical definition. An example of the first case is attaching a slot “fuel − type” to a class “Car”. Both

\(^1\)http://jena.sourceforge.net/
\(^2\)currently there is no link working for this tool
class-definitions still refer to the same ontological concept, but in the second version it is described more extensively. On the other hand, a natural language definition of a concept might change, e.g. the new definition of “chair” might exclude “reclining – chairs” without a logical change of the concept. The former kind of change is referred in the literature as explication change, while the latter conceptual change. Since at the syntactic level, the same data can be the result of any of these types of change, more (human) input is needed to classify the change.

OntoView provides a web "diff" view for comparing two versions of an ontology (see Figure 3.2) at a structural level. The comparison function is inspired by UNIX diff, but the implementation is quite different. The UNIX diff compares file version at line-level, highlighting the lines that textually differ in two versions. OntoView, in contrast, compares versions of ontologies at a structural level, showing which definitions or properties are changed. So as to produce such meaningful difference for ontologies (where there is no inherent ordering), the ontology is canonicalized at the syntactic level before being given to the diff tool.

![Figure 3.1: Comparing two ontologies in OntoView](image-url)

The comparison function used by the OntoView distinguishes between the following types of change:

- Non-logical change (conceptual change). A change at the natural language
definition. e.g. changes in the rdfs:label of a concept or property, or in a comment inside a definition.

- Logical definition change (explication change). This is a change in the definition of a concept or property that affects its formal semantics. Examples of such changes are alterations of subClassOf, domain, or range statements. Additions or deletions of local property restrictions in a class are also logical changes.

- Identifier change. This is the case when a concept or property is given a new identifier, i.e. a renaming.

- Addition of definitions.

- Deletion of definitions.

Most of these changes can be detected completely automatically, except for the identifier change, because this change is not distinguishable from a subsequent deletion and addition of a simple definition. In this case, the system uses the location of the definition in the file as a heuristic to determine whether it is an identifier change or not.

3.2.1 Rules for Changes

The algorithm uses the fact that the RDF data model underlies a number of popular ontology languages, including RDF Schema and DAML+OIL. First, it splits the document at the first level of the XML document. This groups the statements by their intended “definition”. The definitions are then parsed into RDF triples, which results in a set of small graphs. Each of these graphs represent a specific definition of a concept or a property, and each graph can be identified with the identifier of the concept or the property that it represents.

Then, the algorithm locates for each graph in the new version the corresponding graph in the previous version of the ontology. Those sets of graphs are then checked according to a number of rules. Those rules specify the “required” changes in the triples set for a specific type of change.

The rules have the following format:

They specify a set of triples that should exist in one specific version, and a set that should not exist in another version (or the other way round) to signal a specific type of change. With this rule mechanism, the tool is able to specify almost all types of changes, apart from the “identifier change”.

The rules are specific for a particular RDF-based ontology language because they encode the interpretation of the semantics of the language for which they are intended. For another language other rules would have been necessary to specify other differences in interpretation. The semantics of the language are thus encoded in the rules. The mechanism relies on the “materialization” of all rdf:type statements that are encoded in the ontology. In other words, the closure of the RDF triples according to the used ontology language has to be computed.

Regarding the blank nodes, we realize that they can be determined as “identifier changes” through their blank node identifiers. As we have already mentioned the proposed rule mechanism is not applicable to them. The blank node matching is described and applied in an indirect way, by using their location in the file as a heuristic to determine their matching (or not) with a blank node in the second file. No more information are given on this issue.

### 3.3 PromptDiff

PromptDiff is an ontology-versioning tool that determines what has changed between two versions. It finds a structural diff between versions i.e. compares the structure of ontology versions and not their text serialization.

Figure 3.3 shows the overall architecture of the PromptDiff ontology-versioning system. Two versions of an ontology, $v_1$ and $v_2$, are inputs to the system. The heuristic-based algorithm for comparing ontology versions analyzes the two versions and automatically produces a diff between $v_1$ and $v_2$ called a structural
diff (Figure 3.3). The post-processing module uses the diff to identify complex changes. The results are presented to the user through the intuitive interface. The user then has the option of accepting or rejecting changes and these actions are reflected in the updated diff.

![Figure 3.2: The architecture of the PromptDiff](image)

![Figure 3.3: The structural diff showing the difference between two versions](image)

Given two versions of an ontology $O$, $v_1$ and $v_2$, a structural diff between $v_1$ and $v_2$, is a set of pairs $\langle r_1, r_2 \rangle$ where:

- $r_1 \in v_1$ or $r_1 = null$, $r_2 \in v_2$ or $r_2 = null$

- $r_2$ is an image of $r_1$ (matches $r_1$), that is, $r_1$ became $r_2$. If $r_1$ or $r_2$ is null, then we say that $r_2$ or $r_1$ respectively does not have a match.

- Each resource from $v_1$ and $v_2$ appears in at least one pair.

- For any resource $r_1$, if there is at least one pair containing $r_1$, where $r_2 \neq null$, then there is no pair containing $r_1$ where $r_2 = null$. The same is true for $r_2$.

The PromptDiff algorithm consists of two parts: (1) an extensible set of heuristic matchers and (2) a fixed-point algorithm to combine the results of the matchers to produce a structural diff between two versions. Each matcher employs a small
number of structural properties of the ontologies to produce matches. The fixed-point step invokes the matchers repeatedly, feeding the results of one matcher into the others, until they produce no more changes in the diff. Then the differences found by the algorithm are presented to the user who is responsible to accept or reject them.

### 3.3.1 Heuristic Matchers

The PromptDiff algorithm combines an arbitrary number of heuristic matchers, each of which looks for a particular property in the unmatched frames. The heuristic matchers compare two ontology versions looking for the following situations:

- **Resources of the same type with the same name.** In general, if \( r_1 \in K_1 \) and \( r_2 \in K_2 \) and \( r_1 \) and \( r_2 \) have the same name and type, then \( r_1 \) and \( r_2 \) match.

- **Single unmatched sibling.** In general, if \( c_1 \in K_1 \) and \( c_2 \in K_2 \), \( c_1 \) and \( c_2 \) match, and each of the classes has exactly one unmatched subclass, \( subC_1 \) and \( subC_2 \), respectively, then \( subC_1 \) and \( subC_2 \) match.

- **Siblings with the same suffixes or prefixes.** In general, if \( c_1 \in K_1 \) and \( c_2 \in K_2 \), \( c_1 \) and \( c_2 \) match, and the names of all subclasses of \( c_1 \) are the same as the names of all subclasses of \( c_2 \) except for a constant suffix or prefix, then the subclasses match.

- **Single unmatched Property.** In general, if \( c_1 \in K_1 \) and \( c_2 \in K_2 \), \( c_1 \) and \( c_2 \) match, and each of the classes has exactly one unmatched property, \( p_1 \) and \( p_2 \) respectively, and \( p_1 \) and \( p_2 \) have the same domain and range, then \( p_1 \) and \( p_2 \) match.

- **Unmatched inverse properties.** If a knowledge model allows definition of inverse relationships, then those relationships can be used to create matches as well. In general, if \( p_1 \in K_1 \) and \( p_2 \in K_2 \), \( p_1 \) and \( p_2 \) match, \( invP_1 \) and \( invP_2 \) are inverse properties for \( p_1 \) and \( p_2 \) respectively, and \( invP_1 \) and \( invP_2 \) are unmatched, then \( invP_1 \) and \( invP_2 \) match.

- **Split classes.** In general, if \( c_0 \in K_1 \) and \( c_1 \in K_2 \) and \( c_2 \in K_2 \), and for each instance of \( c_0 \), its image is an instance of either \( c_1 \) or \( c_2 \), then \( c_0 \) was split into \( c_1 \) and \( c_2 \). A similar matcher identifies classes that were merged.
The above heuristic matchers are mainly focusing on the matching of the nodes according to their structural difference or similarities. As a result, they can be applied for the matching of the blank nodes. However, no special attention is paid in their case.

3.4 Semversion

SemVersion [? ] is a Java library for providing versioning facilities to RDF data. It is based on RDF/RDFS, so it can be used for any ontology language built or adapted to this data model.

Semversion offers an easy to use (and thus, integrate with) API that closely follows the usual functions and concepts of CVS [? ]. To commit a new version, a user can either provide the complete contents of the version (which is an RDF model, i.e., simply a set of triples), or a diff, that is, the change that is to be applied on a preexisting version to create the new one.

At the implementation level (Figure 3.4), persistence is handled by RDF2Go 3, which provides common storage interfaces over triple- and quad-stores (SemVersion uses the abstraction of the latter), such as Jena 4, Sesame 5, YARS 6, NG4J 7, etc. SemVersion stores each version of an RDF model as a unique independent graph that contains the whole model.

![Diagram of SemVersion architecture](image)

Figure 3.4: The Layered Architecture of SemVersion

Diffs serve two purposes: First, SemVersion allows to compute (structural and

---

3http://ontoware.org/projects/rdf2go/
4http://jena.sourceforge.net/
5http://www.openrdf.org/
6http://sw.deri.org/2004/06/yars/
7http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/
3.4. **SEMVERSION**

Semantic diffs between two arbitrary chosen models, to inform the user about changes. This allows collaborative ontology engineering. Second, diffs can be used in an update command to apply changes to a remotely stored model. When dealing with very large models, it might not be feasible (nor efficient) to transfer the complete model, if only a small fraction has changed.

Semversion provides three type of diffs analyzed at the next sections.

### 3.4.1 Set-based Diff

For versioning, the set-based diff is simply the set-theoretic difference of two RDF triple sets. Such diffs can be computed by simple set arithmetics for triple sets that contain only URIs and literals.

### 3.4.2 Structural Diff and Blank nodes

Without the presence of blank nodes, the set-based diff is the same as the structural diff. With blank nodes, the set-based diff considers all blank nodes to be different and reports all statements involving blank nodes both as added and as removed.

SemVersion also handles the problem of uniquely identifying blank nodes. Blank nodes cannot be globally identified, as they lack a URI, and this poses a challenge at diff algorithms. This is overcome by adding functional properties to each blank node leading to a URI, effectively treating them, from that point on, as normal nodes. This procedure is called *blank node enrichment*. Other tools that process the RDF data are expected not to remove this property, so this will survive the roundtrip ”extract a version from the repository, manipulate it in some ontology editor, reinsert the changes at the repository to create a new version”, so that SemVersion can understand whether two blank nodes are the same. If this URI is missing, then SemVersion treats the node as new (since creating a new node from an external tool would be missing this, *of course*).

However, blank nodes are matched only in the case where they participate in exactly the same triples. In the context of a system where changes arise between subsequent versions, this blank node matching technique is a bit restrictive, as no closest matching is applied.
3.4.3 Semantic Diff

The semantic difference has to take the semantics of the used ontology language into account.

An intuitive way to understand the concept of a semantic diff goes like this: Let’s assume we use RDF Schema as our ontology language, and have two versions \((K\text{ and }K')\) of an RDFS ontology. Now, in order to compute the semantic RDFS diff, we use the closure of \(K\ (C(K))\). Then we do the same for \(K'\ (C(K'))\). Now we calculate a structural (syntactical, set-based) diff on \(C(K)\) and \(C(K')\). This is not the same as the structural diff between \(K\) and \(K'\). If the structural diff of two models is empty, then the semantic diff must also be empty. The inverse is not necessarily true: There might be two different RDF Knowledge Bases which encode the same semantic model, resulting in an empty semantic diff, but a nonempty structural diff.

3.5 x-RDF-3X

The work presented in presents an extension of the RDF-3x system that supports versioning and some other services as time-travel access and transactions on RDF databases. Versioning is achieved by maintaining versions of individual triples (updates are considered as pairs of insertions and deletions, so two timestamps fields are used, the created and deleted to denote the life of each triple version). The \([\text{created}, \text{deleted}]\) interval is the lifespan of the triple version, where deleted has a null value for versions that are presently alive. The database state of a given point in time \(t\) can be reconstructed by returning all triples for which \(t\) falls into the corresponding lifespan interval. Ideally, timestamps reflect the commit order of transactions, but unfortunately the commit order is not known when inserting new data. In order to cope with this problem each transaction is assigned a write timestamp once it starts updating the differential index, and this timestamp is then used for all subsequent operations. Ideally the migration is performed at transaction commit only, which means that the timestamps perfectly reflect the commit order and need no further updates. Moreover, a transaction inventory is used, that tracks transaction ids, their begin and commit times (BOT and EOT), the version number used for each transaction, and the largest version number of all committed transactions (highCV #) at the commit time of a transaction (Figure 2.14). This inventory serves to efficiently decide if a transaction committed before another one. Also, it relates the relative time of transaction ordering and version
numbering to wall clock times. This is needed for supporting time-travel queries and snapshot isolation. Regarding the experimental evaluation synthetic workloads were constructed based on real data (from the LibraryThing book-tagging web site) and the x-RDF-3x system was compared with two other systems: PostgreSQL and Jena. The results show that the first system was 3 times more space efficient than the others systems.

### 3.6 RDF_utils

RDF_utils is developed as part of the KnoBot project. KnoBot is an RDF-based content management system which stores all its data in a Jena Model. It is designed to allow decentralized exchange of information founded on trust relationships between individual persons/agents. While it is designed to maximally comply with standards and best practices it offers novel features (e.g. relevance based aggregation).

RDF_utils is a utility tool for dealing with RDF data, it provides the following features:

- **Leanify**: Remove redundant statements (and anonymous nodes) from rdf-graphs
- **Diff**: Show the difference between two rdf-graphs

![Figure 3.5: The granularity of the SW ranges from the universal graph to triple](image)

The main difference of this work to others is the choice of the level of granularity (Figure 3.6). The problem of granularity is well explained in [? ]. RDF documents and named graphs are too coarse for some particular application needs, such as in tracking provenance of an RDF graph. In this case, the overlap of
the graph at hand with other graphs is a key to identify its provenance. But a named graph can’t be used to express an overlap, as it will generally contain irrelevant triples too, unless explicitly calculating the intersection. On the other hand, triple-level is too fine-grained, due to the case of blank nodes. For example, see the RDF graphs of Figure 3.6. The first one shows an unnamed resource (blank node) with surname ’Ding’ and first name ’Li’. The second graph is identical, while the third described another ’Ding’ person, in particular ’Zhongli Ding’. If the triple-based overlap was meant to be used, the first and the third graph would appear that they share a common triple, while in fact the triples describe different people. This is due to the lack of universal identity of blank nodes; their identity is only derived by the named resources or literals connected to them. Clearly, when blank nodes are involved, equality of triples can’t reliably be used as identification of equal RDF content.

Figure 3.6: Three RDF graphs showing personal information from three sources

In [?], the decomposition is defined as follows. An RDF graph decomposition consists of three elements \((W, d, m)\): the background ontology \(W\), the decompose operation \(d(G, W)\) which breaks an RDF graph \(G\) into a set of sub-graphs \(G^*_1, G^*_2, ..., G^*_n\) using \(W\), and the merge operation \(m(G^*_1, W)\) which combines all elements in \(G^*_1\) into the a unified RDF graph \(G'\) using \(W\). In addition, a decomposition must be lossless such that
\[
\text{for any RDF graph } G, G = m(d(G, W), W).
\]

RDF molecules are defined as the finest and lossless subgraphs of a graph \(G\) according to a decomposition \((W, d, m)\). Worth of note is that this concept is very similar to the notion of Minimum Self-contained Graphs (MSG), described in [?], one of the differences being that molecules also consider an arbitrary reasoning -the "background ontology"- while MSG deals only with RDF).
3.7 CWM of w3c

CWM is part of SWAP, a Semantic Web Application Platform. SWAP consists of tools and applications to manipulate RDF graphs much like traditional tools manipulate text files. CWM is a command-line tool, written in python, for processing RDF in both the standard XML encoding and an experimental encoding, Notation3 [?].

CWM offers a utility that allows the user to compute the delta between two Knowledge Bases and then to apply the delta on the first Knowledge Base to obtain the second.

The authors state that in case which all the nodes are names, computing the difference between two graphs is simple and straightforward:

If $G_1$ and $G_2$ are ground RDF graphs, then the ground graph delta of $G_1$ and $G_2$ is a pair of (insertions, deletions) where insertions is the set difference $G_2 - G_1$ and deletions is $G_1 - G_2$. This form of delta is reasonably economical: the storage cost is linear in the size of the difference between the graphs.

3.7.1 Patch file format

By analogy to the text diff, there is a need not only for a difference-finding algorithm, but for a patch file format. Such a format needs:

- A way to uniquely identify what is changing.
- A way to distinguish between the pieces added and those subtracted.

It is straightforward to pinpoint the parts of the Knowledge Base that have changed when all nodes are named, but less so in the presence of anonymous nodes. To identify what is changing, Notation3 expressions are used and three new terms are introduced. For example:

```
prefix diff: < http://www.w3.org/2004/delta# >.
{ ?x bank:accountNo "1234578"; bank:balance 4000}

diff:replacement
{ ?x bank:accountNo "1234578"; bank:balance 3575}.
```

This one new property replacement can express any change. Deletions can be written `{...} diff:replacement {} and additions can be written {} diff:replacement {...}`.

The second alternative is very similar but involves two properties, one for inserting and one for deleting:
3.7.2 Weak and Strong Deltas

CWM distinguishes two types of RDF deltas:

- **Weak delta.** Gives enough information to apply it to exactly the Knowledge Base it was computed from.

- **Strong delta.** Specifies the changes in a context independent manner. The difference is not in the format of the output but in the information a particular delta gives.

For example, if bank account numbers are globally unique, then a blank node that represents a bank account can be identified by a particular bank account. In OWL terms, if `bank:accountNumber` is an `owl:InverseFunctionalProperty`, then the node must be the `owl:sameAs` any other node with the same account number. In that case, the delta will be **strong**. If however, many accounts can have the same number, applying that delta to another knowledge base may inadvertently alter the wrong account. The delta is **weak**.

In order to produce strong deltas CWM uses the `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` to assign labels to blank nodes in order to uniquely identify them. Strong deltas are provided if sufficient information can be found in the Web to fully label the input graphs.

It becomes clear that the blank nodes cannot be matched in the general case. The blank node matching is successfully built only in the case of ontologies with functional datatype properties, which are compatible in OWL/Full but not in OWL/DL, let alone in RDF.

3.8 Jena

Jena is a Semantic Web toolkit for Java programmers. The heart of the Semantic Web recommendations is the RDF Graph as a universal data structure. Jena similarly has the Graph as its core interface around which the other components are built.
3.9. **pOWL**

Powl, a web based ontology management tool. Its capabilities include parsing, storing, querying, manipulating, versioning, serving and serializing RDF and OWL knowledge bases for different target audiences. Powl is implemented in the web scripting language PHP.

Powl's architecture consists of 4 stacked tiers, while trying to minimize dependencies and supplying clean interfaces between tiers. It consists of the following tiers:

- Powl store: SQL compatible relational database.
- RDFAPI, RDFSAPI, OWLAPI: layered APIs for handling RDF, RDFS and OWL.
• Powl API: containing classes and functions to build web applications on top of those APIs.

• User interface: a set of PHP pages combining widgets provided by Powl API for accessing (browsing, viewing, editing) model data in a Powl store.

To enable domain experts to collaboratively develop shared conceptualizations based on the Ontology Web Language a key requirement is to support a versioning strategy. In order to support versioning, pOWL does provide a mechanism to compare Ontologies and find the differences between them, but supposes that all the changes were made through the pOWL platform and so they are tracked. One editing action by the user may be complex, but every editing action can be decomposed into smaller editing actions (Figure 3.9) and finally into adds and removes of RDF triples to or from the RDF model.

Powl enables rollback of every particular editing action by determining if the involved triples are still present (if added) or still missing (if removed). A parent action thus may only be rolled back if all sub-actions may be rolled back as well.

![Figure 3.7: pOWL Versioning](image)

### 3.10 RDF/S Diff

In this work the authors are interested in differential functions that produce RDF/S deltas exhibiting interesting formal properties (e.g. regarding their size and execution semantics) so they can be interpreted both by programs and humans. In particular, they interpret RDF/S deltas as sets of change operations (i.e. insertions and removals) that enable to successfully transform one RDF/S KB into another by taking into account or not inferred knowledge (i.e. with or without
side-effects).
More analysis of this work is going to be given in Section 4.2.

3.11 Synopses

To sum up, the existing related work indicates a gap on the treatment of blank nodes. On the one hand there are works (specifically [? ?]) that elaborate an integrated blank node matching, but focus only on the graph isomorphism detection. On the other hand, there are many works ([? ? ? ? ? ?]) that compare Knowledge Bases or ontologies in order to extract the diff. However, none of these works suggests a quite general approach of matching the blank nodes. All of these works can be applied successfully under specific conditions or domains. Other works from the field of matching named entities cannot be directly connected to the problem. These works are not only based on structural similarities, but they also focus on lexicographical similarities, which cannot be applied in the case of blank nodes.
Chapter 4

On reducing RDF Delta in KBs with blank nodes

“Man’s right to know, to learn, to inquire, to make bona fide errors, to investigate human emotions must, by all means, be safe, if the word "freedom" should ever be more than an empty political slogan.”

-Wilhelm Reich

4.1 Introduction

Versioning, Synchronization and Replication services need a method to compare two RDF Knowledge Bases with blank nodes and then transform the first Knowledge Base to the second (Figure 4.1). For this reason two modules are required:

- A differential function to report the differences between two RDF Knowledge Bases

- A change operation semantics that indicates the way the differences must be applied to the first RDF Knowledge Base to get the second one.

Obviously, a differential function that yields the smallest in size result is preferred. Furthermore, a pair of (differential function & change operation semantics) must at least satisfy correctness when synchronizing remote Knowledge Bases.
4.2 Preliminaries

We initially give some basic notation, that is followed in the rest of this work. Then we give the basic definition of RDF Knowledge Bases and determine the differential functions and change operations.

4.2.1 Basic Notation

Let $t$ be an RDF triple of the form $(s, p, o)$, where $s$ is called the subject, $p$ the predicate and $o$ the object of the triple. Let $T$ be the set of all possible RDF triples that can be constructed from an infinite set of URIs (for resources, classes and properties) as well as literals. In general, an RDF Knowledge Base can be seen as a finite subset $K_1$ of $T$, i.e. $K_1 \subseteq T$.

Another presentation of a Knowledge Base $K_1$ is as a directed labelled graph $G_1$. The nodes of the graph are the URIs, the literals and the blank nodes that appear as subjects or objects in the triples of $K$, while the edges of the graph are labelled (with URIs) arcs that connect the corresponding nodes. We can partition the nodes of $N_1$ into three sets $N_1 = U_1 \cup L_1 \cup B_1$, where $U_1$ is the set of all URI nodes, $L_1$ is the set of all literals nodes, and $B_1$ is the set of all blank nodes of G.

4.2.2 RDF Knowledge Bases

Simple entailment could be characterized as entailment which depends only on the basic triples syntax of RDF graphs, without making any further assumptions about the meaning of any URI references. Simple entailment is the vocabulary entailment of the empty vocabulary.

However, other semantics (like the RDF/S semantics \[?\]) or custom inference rules (like \[?\]) could be applied over a Knowledge Base $K$. In such cases, apart from the explicitly specified triples of $K$, other triples can be inferred based on the
the respective semantics. For this reason, we introduce the notion of closure and reduction of RDF Knowledge Bases focusing on the RDF/S Semantics.

The closure of $K$, denoted by $C(K)$, contains all the triples that either are explicitly asserted or can be inferred from $K$ by taking into account class or property assertions made by the associated RDFS schemas. Thus, we can consider that $C(K)$ is defined (and computed) by taking the reflexive and transitive closures of RDFS binary relations such as subClassof and type. It should be stressed that our work is orthogonal to the consequence operator of logic theories [?] actually employed to define the closure operator $C$. Specifically, if $P$ denotes the powerset of all possible sets of triples of $T$, then the closure operator can be defined as any function $C : P \rightarrow P$ that satisfies the following properties:

- $K \subseteq C(K)$ for all $K$, i.e. $C$ is extensive
- If $K \subseteq K'$ then $C(K) \subseteq C(K')$, i.e. $C$ is monotonically increasing
- $C(C(K)) = C(K)$ for all $K$, i.e. $C$ is an idempotent function

If it holds $C(K) = K$, then we will call $K$ completed. The elements of $K$ will be called explicit triples, while the elements of $C(K) - K$ will be called inferred. We can now define an equivalence relation between two Knowledge Bases.

**Def. 1** Two Knowledge Bases $K$ and $K'$ are equivalent, denoted by $K \sim K'$, iff $C(K) = C(K')$.

The reduction of a $K$, denoted by $R(K)$, is the smallest in size set of triples such that $C(R(K)) = C(K)$. Let $\Psi$ denote the set of all knowledge bases that have a unique reduction. Independently of whether the reduction of a $K$ is unique or not, we can characterize a $K$ as (semantically) redundancy free, and we can write $RF(K) = True$ (or just $RF(K)$), if it does not contain explicit triples which can be inferred from $K$. Formally, $K$ is redundancy free if there is not any proper subset $K'$ of $K$ (i.e. $K' \subsetneq K$) such that $K \sim K'$. Figure 4.2 illustrates the above sets of triples ($R(K)$ is enclosed in a dashed box because it is not always unique).

It is worth noticing that the reduction of a $K$ is not always unique. In general, uniqueness of the transitive reduction of a binary relation $R$ is guaranteed only when $R$ is antisymmetric and finite. Unfortunately, this is not the case of RDF/S Knowledge Bases allowing cycles in the subsumption relations. For example, in Figure 4.3 we have $K \sim K_1 \sim K_2$, moreover $RF(K_1), RF(K_2)$, but $K_1 \neq K_2$. 
4.2.3 Differential Function and Change Operations

In the context of this work, we focus on two basic change operations allowing to transform one Knowledge Base to another, namely triple addition \( \text{Add}(t) \) and deletion \( \text{Del}(t) \) where \( t \in T \). In this respect, a triple update is "split" into an addition and a deletion of triplets having the same subject and predicate (and thus keep both "old" and "new" values usually ignored by updates).

[? ] introduced five differential functions of RDF/S Knowledge Bases. They are given in Figure 4.4, namely, \( \Delta_e, \Delta_c, \Delta_d, \Delta_{dc} \) and \( \Delta_{ed} \).

\[
\begin{align*}
\Delta_e(K \rightarrow K') &= \{ \text{Add}(t) \mid t \in K' - K \} \cup \{ \text{Del}(t) \mid t \in K - K' \} \\
\Delta_c(K \rightarrow K') &= \{ \text{Add}(t) \mid t \in C(K') - C(K) \} \cup \{ \text{Del}(t) \mid t \in C(K) - C(K') \} \\
\Delta_d(K \rightarrow K') &= \{ \text{Add}(t) \mid t \in K' - C(K) \} \cup \{ \text{Del}(t) \mid t \in K - C(K') \} \\
\Delta_{dc}(K \rightarrow K') &= \{ \text{Add}(t) \mid t \in K' - C(K) \} \cup \{ \text{Del}(t) \mid t \in C(K) - C(K') \}
\end{align*}
\]

\( \Delta_e \) (where \( e \) stands for explicit) actually returns the set difference over the explicitly asserted triples, while \( \Delta_c \) (where \( c \) stands for closure) returns the set dif-
ference by also taking into account the inferred triples\(^1\). Three novel differential functions namely \(\Delta_d\) (where \(d\) comes from dense), \(\Delta_{dc}\) (\(dc\) comes from dense & closure) and \(\Delta_{ed}\) (\(ed\) comes from explicit & dense) are introduced in [?]. It results that \(\Delta_d\) produces the smallest in size set of change operations.

These five differential functions yield essentially sets of atomic change operations. More formally, for a differential function \(\Delta_x(K \rightarrow K')\) where \(x \in \{e,c,d,dc,ed\}\), \(\Delta^+_x\) is used to denote the corresponding set of triple additions (i.e. incremental changes) and \(\Delta^-_x\) the set of triple deletions (i.e. decremental changes).

Obviously, \(\Delta_x\) contains only sets of useful change operations reflecting the net effect of successive modifications over the same (explicit or inferred) triple of two Knowledge Base versions. In other terms, \(\Delta_x\) does not contain both \(Add(t)\) and \(Del(t)\) operations for a given \(t \in T\).

**Def. 2** A Delta \(\Delta(K \rightarrow K')\) is **useful** if it holds \(\Delta^+ \cap \Delta^- = \emptyset\).

By defining RDF Deltas as sets of atomic change operations, we avoid to specify an execution order as in an edit-script (i.e. a sequence of triple additions or

\(\text{deleted triples} \quad \text{added triples}\)

---

\(^1\) Mention that \(\Delta_e\) and \(\Delta_c\) define a symmetric set difference.
deletions). This design choice amends to simpler computation requirements for RDF/S Deltas while it provides the opportunity of applying alternative semantics of changes when transforming one Knowledge Base to another (i.e. with or without side-effects on the Knowledge Base closure).

We introduce two sets of change operations:

- **Atomic** change operations
  Here we have operations of the form $Add(t)$ and $Del(t)$ where $t \in T$.

- **Bulk** change operations
  Here we consider the operation $AddDel(A, D)$ where $A$ and $D$ are disjoint sets of triples (i.e. $A \subseteq T$, $D \subseteq T$, $A \cap D = \emptyset$).

It is not hard to see that the result of a differential function $\Delta(K \rightarrow K')$ could be applied on $K$ by issuing one single call to $AddDel(\cdot, \cdot)$ (i.e. $AddDel(\Delta^+, \Delta^-)$), or by issuing $|\Delta^+|$ calls to $Add(\cdot)$ and $|\Delta^-|$ calls to $Del(\cdot)$ in any possible order.

For the needs of this work we mostly use the differential function $\Delta_e$. For short, its output is called $\text{Delta}$.

### 4.3 RDF KBs with Blank Nodes

Let us now focus on the comparison of equivalent Knowledge Bases that contain blank nodes. In the context of this section we assume a simple entailment for the participating RDF graphs focusing only on their explicit triples. Discussion on how the proposed methods can be applied over various semantics, is given in Section 4.6. Section 4.3.2 describes an extra process that is applied over $\Delta$, while Section 4.3.3 provides an upper bound for the size of $\Delta$ after blank node matching is applied.

#### 4.3.1 RDF graph equivalence

The equivalence of $\text{RDF graphs}$ that contain $\text{blank nodes}$ is defined in [? ] as:

**Def. 3 (Equivalence of RDF Graphs that contain Blank nodes)**

Two RDF graphs $G_1$ and $G_2$ are equivalent if there is a bijection$^2$ $M$ between the sets of nodes of the two graphs ($N_1$ and $N_2$), such that:

- $M(\text{uri}) = \text{uri}$ for each $\text{uri} \in U_1 \cap N_1$

$^2$ A function that is both one-to-one (injective) and onto (surjective).
4.3. RDF KBS WITH BLANK NODES

- \( M(lit) = lit \) for each \( lit \in L_1 \)
- \( M \) maps blank nodes to blank nodes (i.e. for each \( b \in B_1 \) it holds \( M(b) \in B_2 \))
- The triple \((s, p, o)\) is in \( G_1 \) if and only if the triple \((M(s), p, M(o))\) is in \( G_2 \).

It follows that if two graphs are equivalent then it certainly holds \( L_1 = L_2 \) and \(|B_1| = |B_2|\).

Let us now relate the problem of equivalence with edit distances.

**Def. 4 (Edit Distance over Nodes given a Bijection)**

Let \( o_1 \) and \( o_2 \) be two nodes of \( G_1 \) and \( G_2 \), and suppose a bijection between the nodes of these graphs, i.e. a function \( h : N_1 \to N_2 \) (obviously \(|N_1| = |N_2|\)). We define the edit distance between \( o_1 \) and \( o_2 \) over \( h \), denoted by \( dist_h(o_1, o_2) \), as the number of additions or deletions of triples which are required for making the “direct neighborhoods” of \( o_1 \) and \( o_2 \) the same (considering \( h \)-mapped nodes the same). Formally, \( dist_h(o_1, o_2) = \)

\[
|\{(o_1, p, a) \in G_1 \mid (o_2, p, h(a)) \notin G_2\}| + |\{(a, p, o_1) \in G_1 \mid (h(a), p, o_2) \notin G_2\}| + \\
|\{(o_2, p, a) \in G_2 \mid (o_1, p, h^{-1}(a)) \notin G_1\}| + |\{(a, p, o_2) \in G_2 \mid (h^{-1}(a), p, o_1) \notin G_1\}|
\]

Now recall that if \( G_1 \) is equivalent to \( G_2 \) then there exists a bijection \( h \) such that \((a, p, b) \in G_1 \iff (h(a), p, h(b)) \in G_2 \). We will denote this by \( G_1 \equiv_h G_2 \). It follows that:

**Theorem 1 (RDF Graph Equivalence and Edit Distance)**

\( G_1 \equiv_h G_2 \iff dist_h(o, h(o)) = 0 \) for each \( o \in N_1 \).

**Proof:**

\((\Rightarrow)\)

*Trivial.* If \( G_1 \equiv_h G_2 \) then for all pairs of objects \( o, h(o) \) where \( o \in N_1 \), all summands of the equation of Def. 4 are zero.

\((\Leftarrow)\)

\( dist_h(o, h(o)) = 0 \) implies that:

- \( \forall (o, p, a) \in G_1 \Rightarrow (h(o), p, h(a)) \in G_2 \)
- \( \forall (a, p, o) \in G_1 \Rightarrow (h(a), p, h(o)) \in G_2 \)
- \( \forall (h(o), p, a) \in G_2 \Rightarrow (o, p, h^{-1}(a)) \in G_1 \)
- \( \forall (a, p, h(o)) \in G_2 \Rightarrow (h^{-1}(a), p, o) \in G_1 \).

If we take the implications of all \( dist_h(o, h(o)) \) where \( o \in N_1 \), we get that \((a, p, b) \in G_1 \iff (h(a), p, h(b)) \in G_2 \), and thus \( G_1 \equiv_h G_2 \). \( \diamond \)
Obviously the above theorem is useful for the case where the bijection $h$ respects the constraints of Def. 3 (i.e. maps named elements to named elements, and anonymous elements to anonymous).

### 4.3.2 Bnode Name Tuning

The basic idea for reducing the Delta is the following: if we match a blank node $b_1$ (of $B_1$) to a blank node $b_2$ (of $B_2$), through a bijection $M$, then these blank nodes can be considered as equal at the computation of Delta. For example, if $K_1$ contains a triple $(b_1, name, Joe)$ and $K_2$ contains a triple $(b_2, name, Joe)$ and we match $b_1$ to $b_2$, then these two triples will be considered equal and thus no difference will be reported. However we should note that in the context of versioning or synchronization services the change operations derived by a differential function should not be used as they are. For example, consider $K_1 = \{(b_1, name, Joe)\}$ and $K_2 = \{(b_2, name, Joe), (b_2, lives, UK)\}$ and suppose that we match again $b_1$ to $b_2$. In this case a mapping-aware comparison function will return the Delta $\{Add((b_2, lives, UK))\}$. If we want to apply it on $K_1$ then we have to replace $b_2$ by $b_1$, i.e we should apply on $K_1$ the operation $Add((b_1, lives, UK))$, and in this way, we will obtain $K_1' = \{(b_1, name, Joe), (b_1, lives, UK)\}$ which is equivalent to $K_2$. We call this step Bnode Name Tuning, and it actually replaces (renames) in the Delta the local names of the blank nodes of $B_2$ by the local names of the matched blank nodes in $B_1$. In this way the Delta does not need any rename operation (i.e. $rename(b_1, b_2)$) and hence not any particular execution order of its change operations.

### 4.3.3 Delta reduction size

Blank node matching cannot increase the Delta size. Without blank node matching any pair of blank nodes from different Knowledge Bases is considered different, and thus all triples to which they participate will be different and reported as change operations in the Delta. In particular, we denote $D_a$ as the average number of direct edges of the blank nodes (i.e. average number of triples to which a blank node participates) and $n_1 = |B_1|$, $n_2 = |B_2|$. Without blank node matching the produced Delta will contain at least $(n_1 + n_2) \ast D_a$ change operations.

On the other hand, if two blank nodes are matched then the Delta size is reduced if they participate to triples with the same predicate and the same other node (i.e. the same subject or object). In the worst case where all predicates/nodes of these
4.4. BNODE MATCHING AS AN OPTIMIZATION PROBLEM

4.4.1 Problem Formulation

Here we formulate the problem of finding a mapping between the blank nodes of two Knowledge Bases as an optimization problem. Let \( n_1 = |B_1|, n_2 = |B_2| \) and \( n = \min(n_1, n_2) \). We have to match \( n \) elements of \( B_1 \) with \( n \) elements of \( B_2 \), i.e. our objective is to find the unknown part of the bijection \( M \). To be more precise, \( M \) a priori contains the mappings of all the URIs and literals of the Knowledge Bases (URIs and literals are mapped as an identity function as in Def. 3), and its unknown part concerns \( B_1 \) and \( B_2 \). Suppose that \( n = n_1 < n_2 \). Let \( J \) denote the set of all possible bijections between \( B_1 \) and a subset of \( B_2 \) that comprises \( n \) elements. The number of all possible bijections (i.e. \(|J|\)) is \( n_2 \times (n_2 - 1) \times ... \times (n_2 - n_1 + 1) \), i.e. the first element of \( B_1 \) can be matched with \( n_2 \) elements of \( B_2 \), the second with \( n_2 - 1 \) elements, and so on. Consequently, the set of candidate solutions is exponential in size.

Since our objective is to find a bijection \( M \in J \) that reduces the Delta size (as regards the “unnamed” parts of the Knowledge Bases), we define the cost of a bijection \( M \) as follows:

\[
\text{Cost}(M) = \sum_{b_1 \in B_1} \text{dist}_M(b_1, M(b_1)) \quad (4.1)
\]

Def. 5 (The bijection yielding the less Delta size) The best solution (or solutions) is defined as the bijection with the minimum cost, i.e. we define:

\[
M_{\text{sol}} = \arg_M \min_{M \in J} \text{Cost}(M)
\]

The notation \( \arg_M \) returns the \( M \) in \( J \) that gives the minimum cost.
Theorem 2 (Equivalence and Mapping Cost) If \( G_1 \equiv_{M_{sol}} G_2 \) (according to Def. 3) then \( \text{Cost}(M_{sol}) = 0 \).

The proof follows easily from the definitions. It is also clear that the inverse of Th. 2 does not hold (i.e. \( \text{Cost}(M_{sol}) = 0 \not\Rightarrow G_1 \equiv_{M_{sol}} G_2 \)) because the cost is based on the distance between the direct neighborhoods of the blank nodes only, and not between the named parts of the graphs.

From the algorithmic perspective, one naive approach for finding the best solution (i.e. \( M_{sol} \)) would be to examine the set of all possible bijections. That would require at least \( n! \) examinations (true if \( n_1 = n_2 = n \), while if \( n_1 < n_2 \) then their number is higher than \( n! \)). However, the problem is intractable in general:

**Theorem 3** Finding the optimal bijection (according to Def. 5) is NP-Hard.

**Proof:**

We will show that subgraph-isomorphism (which is NP-complete problem) can be reduced to the problem of finding the optimal bijection (meaning that our problem is at least as hard as subgraph-isomorphism). Let us make the hypothesis that we can find the optimal bijection in polynomial time. We will prove that if that hypothesis were true, then we would be able to solve the subgraph isomorphism in polynomial time. The subgraph isomorphism decision problem is stated as: given two plain graphs \( G_1 \) and \( G_2 \) decide whether \( G_1 \) is isomorphic to a subgraph of \( G_2 \). Let \( G_1 = (N_1, R_1) \) and \( G_2 = (N_2, R_2) \). We can consider these graphs as two RDF graphs such that: all of their nodes are blank nodes and all property edges have the same label. Assume that \( |N_1| \leq |N_2| \) and let \( n = \min(|N_1|, |N_2|) \). If we can find in polynomial time whether there is a bijection between the \( n \) nodes of \( G_1 \) and \( n \) nodes of \( G_2 \) such that \( \text{Cost}(M_{sol}) = 0 \), then this means that we have found whether \( G_1 \) is isomorphic to a subgraph of \( G_2 \). Specifically, to decide whether there is a subgraph isomorphism, (a) we compute the optimal bijection, say \( M_{sol} \), and (b) we compute its cost. If the cost returned by step (b) is 0 then we return YES, i.e. that there is a subgraph isomorphism. Otherwise we return NO (i.e. there is no subgraph isomorphism). Note that step (a) is polynomial by hypothesis, while step (b) relies on Def. 4 and its cost is again polynomial. Regarding the latter, note that \( M_{sol} \) contains \( n \) pairs, and to compute \( \text{dist}_M(b_1, b_2) \) for each \((b_1, b_2)\) pair of \( M \), we consider only the direct neighborhoods of the two nodes in the two graphs (for \( G_2 \) we have to consider only those that connect nodes that participate in \( M_{sol} \)).

It follows that its computational cost is analogous to the number of edges of the graphs, and thus polynomial. Therefore given a bijection \( M_{sol} \), to compute \( \text{Cost}(M_{sol}) \) requires polynomial time.

Also note that Th. 1 holds also for plain graphs assuming a distance function over not labeled edges. We conclude that if our hypothesis were true, then we would be able to decide subgraph isomorphism in polynomial time.

We conclude that finding the optimal bijection is NP-Hard.

\(^3\) Alternatively, if \( \text{Cost}(M_{sol}) \neq 0 \) (using the distance as defined previously), we return YES only if \( \Delta_e(G_1 \rightarrow G_2) \) as defined in section 4.2.3, after blank node name tuning, contains only triples each containing one blank node in \( B_1 \) and one not in \( B_1 \).
Below we will show that there are algorithms of polynomial complexity for a frequently occurring case. For the general case, we will propose algorithms of polynomial complexity that return an approximate solution.

### 4.4.2 Polynomally-solved (and Frequently Occurring) Cases

#### No directly connected blank nodes

Consider the Knowledge Bases in Figure 4.5 and suppose that we want to compute $\text{dist}_h(\_ : 1, \_ : 6)$ (according to Def. 4). It is not hard to see that this distance depends on the mappings (known by $h$) of the blank nodes that are connected to $\_ : 1$ and $\_ : 6$, i.e. on the mappings of $\_ : 3, \_ : 4, \_ : 8$ and $\_ : 9$. However several datasets do not have directly connected blank nodes. For this reason, here we study a variation of the problem that is appropriate for this case. The key point is that the edit distance between two blank nodes does not depend on how the rest blank nodes are mapped (i.e. is independent of $h$).

![Figure 4.5: Two Knowledge Bases with directly connected blank nodes](image)

This is very important because in this case our problem is mapped with the assignment problem. In its most general form, the assignment problem is given as follows: There is a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task in such a way that the total cost of the assignment is minimized.
For the needs of our problem the blank nodes of the first graph, \( B_1 \), play the role of agents, the blank nodes of the second graph, \( B_2 \), play the role of tasks, and the edit distances of the pairs in \( B_1 \times B_2 \) play the role of the costs. The cost of the bijection is mapped to the total cost of the assignment.

The Hungarian algorithm \([?]\) is ideal for solving the assignment problem. We call this algorithm \( \text{Alg}_\text{Hung} \) for short. Consider for the moment that \(|B_1| = |B_2|\). If we compute the edit distances between all possible \( n^2 \) pairs, then \( \text{Alg}_\text{Hung} \) can find the optimal assignment at the cost of \( O(n^3) \) time. This means that finding the optimal solution costs polynomial time. An extension of \( \text{Alg}_\text{Hung} \) giving the ability to assign the problem in rectangular matrices (i.e. when \(|B_1| \neq |B_2|\)) is already provided in \([?]\). We conclude that if there are not directly connected blank nodes then the optimal mapping can be found in polynomial time.

**Theorem 4** Finding the optimal bijection (according to Def. 5) is a polynomial task if there are no directly connected blank nodes.

In the context of this work we managed to distinguish the aforementioned subcase, where the problem is solved optimally. However, between this blank node structure (no directly connected blank nodes) and the blank node structure described in the proof of intractability (all the blank nodes form a graph), there is space left for more subcases that may be open for a polynomial optimal solution. Indicatively, we have already started investigating an optimal solution for the case where the blank node structures form only rooted trees (like that in Figure 4.5). Actually, this is the first topic for our future research.

### 4.5 Approximation Algorithms

For the general case, where the blank node structures can even form a graph, we propose approximation algorithms. At section 4.5.1 we present a variation of \( \text{Alg}_\text{Hung} \) for getting an approximate solution for the general case, then at Section 4.5.2 we present a signature-based algorithm appropriate for larger datasets.

#### 4.5.1 Hungarian BNode Matching Algorithm

We have already stated that \( \text{Alg}_\text{Hung} \) can find the optimal mapping in polynomial time if no directly connected bnodes exist in the compared Knowledge Bases. For the cases where there are directly connected bnodes, \( \text{Alg}_\text{Hung} \) enriched with an assumption regarding how to treat the connected bnodes at the computation
of \( dist_h \), could be used for producing an approximate solution. Also in this case the algorithm will make \( n_1 \times n_2 \) distance computations (where \( n_1 = |B_1| \) and \( n_2 = |B_2| \)), and the complexity of the algorithm will be again \( O(n^3) \).

Regarding connected bnodes, at the computation of \( dist_h \), one could either assume that all of the connected bnodes are different, or all of them are the same. The first assumption does not require any bijection \( h \) contains only the identity functions of the URIs and literals). According to Definition 4, the fact that all the bnodes are different means by extension that the triples in the direct neighborhoods connecting blank nodes are different too, even in the case where these triples have the same properties. For instance, applying the Definition 4 between bnodes \( (_ : 1, _ : 6) \) and \( (_ : 1, _ : 7) \) of Figure 7.1, we get that \( dist_h(_ : 1, _ : 6) = 4 \) and \( dist_h(_ : 1, _ : 7) = 3 \) respectively. However, bnodes \( _ : 1, _ : 6 \) have two outgoing triples with exactly the same properties, while bnodes \( _ : 1, _ : 7 \) have only one. We observe that this assumption is not very good because we would prefer \( _ : 1 \) to be “closer” to \( _ : 6 \) than to \( _ : 7 \).

According to the alternative assumption, when comparing bnodes \( (_ : 1, _ : 6) \) in Figure 7.1, bnode \( _ : 3 \) can be matched either with bnode \( _ : 8 \) or with bnode \( _ : 9 \), depending on the existence of a common property between them. This yields \( dist_h(_ : 1, _ : 6) = 0 \) since both bnodes have two outgoing triples with common properties (i.e. \( (_ : 1, brother, _ : 3) \) is matched with \( (_ : 6, brother, _ : 8) \) and \( (_ : 1, friend, _ : 4) \) is matched with \( (_ : 6, friend, _ : 9) \)). Regarding \( _ : 1 \) and \( _ : 7 \), we get \( dist_h(_ : 1, _ : 7) = 1 \) because of the deleted triple \( (_ : 1, brother, _ : 3) \). It follows that the results of this assumption are better over this example, as \( _ : 1 \) is “closer” to \( _ : 6 \) than to \( _ : 7 \). In general it is better because it exploits common properties, and therefore we adopt this assumption in our experiments.

In other words, the above assumptions do not compute the "likelihood" of the blank nodes to be matched. They arbitrarily suppose that there is no likelihood (0) of matching (first assumption) by charging with 2 the edit distance (one deletion and one addition) or there is probability equal to 1 of the blank nodes to be matched, by not charging at all the edit distance. The computation of the "likelihood" would require an increase in the time and memory demands.

### 4.5.2 A Fast \( O(N \log N) \) Signature-based Algorithm

The objective here is to devise a faster mapping algorithm that could be applied to large Knowledge Bases (with a heavy load of bnodes), at the cost of probably
bigger Deltas and less chances to detect equivalences if they exist. We propose a signature-based mapping algorithm, for short AlgSign, which consists of two phases: the signature construction and the mapping construction phase. Regarding the signature construction phase, for each bnode b we produce a string based on the direct neighborhood of b. This string is called the signature of bnode b. At the end of this phase we get two lists of signatures, one for the bnodes of each Knowledge Base. These lists should be considered as bags rather than sets, as there is a probability that two or more bnodes get the same signature. This probability and by extension the deviation of this algorithm from the optimal solution depends on the way the signature is built (we discuss this later). In this context, we shall use the symbol $\oplus$ to denote the operation of union over bags, as the result can contain duplicates.

### Alg. SignatureMapping

**Input:** two sets of bnodes $B_1$ and $B_2$, where $|B_1| < |B_2|

**Output:** a bij. M between $B_1$ and $B_2$

1. $M = \emptyset$
2. $BS_1 = BS_2 = emptybag$
3. for each $b_1 \in B_1$
   4. $BS_1 = BS_1 \uplus \{Signature(b_1)\}$
5. for each $b_2 \in B_2$
   6. $BS_2 = BS_2 \uplus \{Signature(b_2)\}$
7. sort($BS_1$)
8. sort($BS_2$)
9. for each $bs_1 \in BS_1$
   10. $bs_2 = Lookup(BS_2, bs_1)$
   11. if ($bs_2 == bs_1$)
   12. $M = M \cup \{(bn_1[bs_1], bn_2[bs_2])\}$
   // $bn_1[stra]$ returns the $b \in B_1$ corresponding to str
13. $BS_2.remove(bs_2)$
14. $BS_1.remove(bs_1)$
15. for each $bs_1 \in BS_1$
   16. $bs_2 = Lookup(BS_2, bs_1)$
   17. $M = M \cup \{(bn_1[bs_1], bn_2[bs_2])\}$
18. $BS_2.remove(bs_2)$
19. return $M$

**Figure 4.6:** Alg. The Signature-based bnode matching algorithm

The mapping phase takes these two bags of strings and compares the elements of the first bag with those of the second. To make binary search possible, both bags (lists) are sorted lexicographically. Subsequently, we start from the smaller list, say $BS_1$, and for each string $bs_1$ in that list we perform a lookup in the second list $BS_2$ using binary search. If an exact match exists (i.e. we found the string $bs_1$ also in $BS_2$) we produce a bnode mapping, i.e. the pair $(bn_1[bs_1], bn_2[bs_1])$. Since more than one bnodes may have the same signature we select one. We prefer the order as provided by the managing software, which in many cases reflects the order by which bnodes appear in files. As there is a high probability for subsequent versions to keep the same serialization, using the original order increases the probability of matches in case of same signatures. More information about

---

4 However we theoretically and experimentally show that it gets the same chance to detect equivalence.
5 We do the same in AlgHung in case of ties in costs.
the role of serialization are given in Section 4.5.6. We continue in this way for all strings of $BS_1$. For each element $bs_1$ of $BS_1$ for which no exact match was found in $BS_2$ we perform a second lookup over the remainder part of $BS_2'$, say $BS_2''$, which will produce a mapping based on the closest element of $BS_2''$ to the $bs_1$ element. Specifically we will match $bs_1$ to the element of $BS_2''$ to which binary search stopped, i.e. to the lexicographically closer element. Note that we perform the closest matches after finishing with the exact matches in order to avoid the situation where an approximate (closest) match deterrs an exact match at a later step.

The algorithm is shown in Figure 4.6 and relies on an algorithm Signature for producing signatures, and on an algorithm Lookup, that are analyzed below. The complexity of this algorithm is $O(n \log N)$ where $N = \max(n_1, n_2)$ and $n = \min(n_1, n_2)$, assuming that the average graph degree of bnodes (and thus signature size) does not depend on $N$.

### Signature Construction

A crucial issue is how the signature of each bnode is derived. We would like to devise a signature construction method producing signatures with high discrimination power. Our objective is to derive a string that will allow good matches in the Lookup algorithm even if the bnodes do not match exactly. The more resources we capture inside the signature of a bnode, the more we increase its discrimination power. However, the need to keep the signature size independent of $N$ forces us to remain only in the direct neighborhood.

Figure 4.7 shows two subsequent Knowledge Bases of the Address Ontology. Table 4.5.2 gives the signatures of all the bnodes in Figure 4.7. The “ex:” namespaces of the resources are omitted here just for needs of space.

<table>
<thead>
<tr>
<th>Local Name</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ : 1</td>
<td>foaf:ChristinahasAddress\rdfs\tpertype\cityLondon\n\no14\n\streetOxfordStreet</td>
</tr>
<tr>
<td>_ : 3</td>
<td>foaf:ChristinahasAddress\rdfs\tpertype\cityLondon\n\no14\n\streetOxfordStreet</td>
</tr>
<tr>
<td>_ : 2</td>
<td>foaf:YanninishasAddress\rdfs\tpertype\cityNewYork\n\no445\n\streetBroadway</td>
</tr>
<tr>
<td>_ : 4</td>
<td>foaf:YanninishasAddress\rdfs\tpertype\cityChicago\n\no132\n\streetMichiganAvenue</td>
</tr>
</tbody>
</table>

Consider bnode _ : 1 of Figure 4.7 which is involved in the following triples:
CHAPTER 4. ON REDUCING RDF DELTA IN KBS WITH BLANK NODES

Figure 4.7: Two Knowledge Bases of an address ontology

Incoming triples: \{ (\text{foaf:Christina}, \text{hasAddress}, _ : 1) \},
Outgoing triples: \{ (_ : 1, \text{street}, \text{OxfordStreet}), (_ : 1, \text{No}, 14), (_ : 1, \text{city}, \text{London}) \},
ClassType triples: \{ (_ : 1, \text{rdf:typex} : \text{Address}) \}.

The set \text{ClassType} contains the triples with the \text{rdf:ty}pe property of the respective blank node. Each of these triples will be mapped to a substring (i.e. “\text{foaf:Christina} hasAddress” for the triple (\text{foaf:Christina}, \text{hasAddress}, _ : 1)).

For each one of the three different sets of triples (Incoming, Outgoing, Class Type) we are going to construct a concatenation of substrings (i.e. “\text{cityLondon} \ast \text{no14} \ast \text{streetOxfordStreet}” for the Outgoing set of triples). The substrings inside each set are sorted lexicographically and separated by a special character, here denoted by $\ast$.

These sets of substrings are then concatenated and separated by a special character, here denoted by $\diamondsuit$. This final concatenation yields the signature. In case there is a blank node as subject in the set of incoming triples or object in the set of outgoing triples, we replace it by a special character, here denoted by $\spadesuit$. In other words, we treat them as equal, as we did in the second assumption of approximation version of \text{AlgHung}. The reasoning of the structure of the signature is given later.

The exact steps of the signature construction algorithm are shown in algorithm \text{Signature1} at Figure 4.8. The method name(o) gives us the uri reference if o is a \textit{uri} and the label if o is a literal. The method isBNode(n) returns a boolean that is true if n is a bnode or false if it is not. The strings \text{inSet}, \text{outSet}, \text{classSet} are the three sets that are gradually built by concatenating the information from all the triples in the direct neighborhood.
4.5. APPROXIMATION ALGORITHMS

The Lookup algorithm

This algorithm is actually responsible for the high time efficiency of this approximation algorithm, as it is a slightly modified version of Binary Search. Instead of the classical BinarySearch, this algorithm Lookup always returns a string. If the Lookup succeeds, it returns the matching (i.e. the string that we search), otherwise it returns the closest (in lexicographical order) string. The closest string is defined as the string located either in the position the BinarySearch stopped or in the exact previous (if the last iteration decreased the high value) or in the next (if the last iteration increased the low value) position. In order to make a decision among the two strings (signatures), we define the elements of a signature, say $elems(bs)$, as the number of the building blocks (i.e. sub-strings split by the characters $\diamond$ or $\ast$) included in its string. The returned string among these two strings is the one that has as $elems$ a value closer to the $elems$ of the searching string. The exact steps of the algorithm LookUp are shown in Figure 4.9.

Applying the Signature Mapping algorithm in the simple example of Figure 4.7 we
Alg. Lookup

Input: a string $bs$ and a sorted bag of strings $BS$

Output: the best matching string $bs_2$

1. $a = NULL$
2. $low = 0$
3. $high = BS.length$
4. while ($low < high$)
   5. $mid = \lfloor (low + high) / 2 \rfloor$
   6. $bmid = BS[mid]$
   7. if ($bs < bmid$)
      8. $high = mid$
   9. else if ($bs > bmid$)
      10. $low = mid + 1$
11. else return $bmid$ // exact match found
12. end while
13. if ($bs < bmid$ ∧ ($|elems(bs) - elems(bmid)| > |elems(bs) - elems(BS(mid - 1))|$))
14. return $BS[mid - 1]$
15. else if ($bs > bmid$ ∧ ($|elems(bs) - elems(bmid)| > |elems(bs) - elems(BS(mid + 1))|$))
16. return $BS[mid + 1]$
17. else
18. return $bmid$

Figure 4.9: Lookup algorithm

get the final mapping: $:_1 \leftrightarrow _:3$ and $:_2 \leftrightarrow _:4$. Going to a more complex example (i.e. with connected bnodes), like the one of Figure 7.1 the final mapping would be $:_1 \leftrightarrow _:6$, $:_2 \leftrightarrow _:7$, $:_3 \leftrightarrow _:8$, $:_4 \leftrightarrow _:10$, $:_5 \leftrightarrow _:9$.

4.5.3 More about the Signature Construction

A key point in the efficiency of this algorithm is the order by which the sets of triples are concatenated. The prevailing option is to give a first priority to the set of the incoming triples, a second priority to the set with the rdf:type properties, and the last priority to the set of the outgoing triples. Let us remind that at this phase we are looking for a signature construction method that is completely general and domain-agnostic.

Some intuition for the proposed ordering stems from the evidence that the probability for the outgoing statements to change is higher than the ingoing (i.e. like in Figure 4.7 where updating the address of a person is more probable than changing his/her name).

A closer approach to the functionality of the blank nodes, could persuade the reader for the prevalence of their outgoing triples over their incoming triples. This fact entails that the probability for an outgoing triple to change is higher than an incoming triple. Based on the analysis of functionality of bnodes, described in Section 1, we get the following: (a) the multi-component structure yields that bnodes are arising more times as subjects than objects (see Figure 2.3), (b) the functionality of provenance imposes the usage of bnodes in the subject position more than in the object position (see Figure 2.5) and (c) in the "Multi-relationship expression" the bnode occurs once in the object position and
4.5. APPROXIMATION ALGORITHMS

more than once in the subject position.

Moreover, taking into account the empirical surveys of bnodes in Linked Data, recall that according to [?], surveying a corpus of 1G quadruples, they got that each bnode occurred 0.99 times in the object position of a non-rdf:type triple (with 1.9% of all bnodes not occurring at all in the object position), whereas each bnode occurred on average 4.2 times in the subject position of a triple (with 0.04% not occurring at all in the subject position).

A reason that urges the above results is the tree-based RDF/XML syntax.

We conclude that mainly the functionality purposes and secondly the prevailed RDF format (RDF/XML) indicate the dominance of bnodes in the subject position, justifying that the Incoming Set has more probability to stay stable from the one version to the other, than the Outgoing Set.

4.5.4 A Faster \( (O(N \log N)) \) Signature-based Algorithm

At this point we are going to present a variation of the Signature-based algorithm aiming at providing even faster mappings, without losing in accuracy.

The only difference of this variation is in the mapping phase. Instead of sorting the two bags of signatures and applying two passes (with binary search lookups), we now merge the signatures of both bags and then we sort the resulting bag. No search methods are needed. The key observation here is that thanks to the sorted merged bag each signature of \( BS_1 \) has its matching pair in the previous or next position inside this bag. So, by making two serial passes over this bag, one for the exact and one for the closest signatures, we get exactly the same signature matchings with those of the last algorithm.

We should note that the elements/signatures have markers indicating their provenance (\( BS_1 \) or \( BS_2 \)) inside the merged bag.

In Figure 4.10 we give the new mapping algorithm. Lines (7) - (26) correspond to the new mapping phase. We introduce the method get\(\text{Next}(BS, 1, \text{pointer}) \) that goes to the position \( \text{pointer} \) of the list \( BS \) and returns the next element (from that point), which belongs to the \( BS_1 \) list. Analogously, the get\(\text{Previous}(BS, 2, \text{pointer}) \) returns the previous element that belongs to the \( BS_2 \) list. In both the serial passes, each element of the list \( BS \) that belongs to \( BS_1 \) (actually the smaller in size list between \( BS_1 \) and \( BS_2 \)) is matched with an element of \( BS_2 \). The matching of each pair is performed with the algorithm get\(\text{Pair} \) seen in Figure 4.11. We output all the exact matches in the first pass and all the closer lexicographical matches in the second pass.

A closer view to the algorithm get\(\text{Pair} \) makes clear that the signature mappings of this method are exactly the same with those extracted with the Lookup algorithm.

Regarding the exact signature matching we should note that a possible same signature \( s_2 \in BS_2 \) of \( s_1 \in BS_1 \) is always positioned after \( s_1 \) inside the bag. This is set during the sorting of the two bags.

A C-like version of this algorithm is also given in Figure 4.12. The method \( s_1, prov() \)
returns the provenance of the signature $s_1$ ($B_1$ or $B_2$).

Regarding the complexity of the algorithm, we now make one sorting (line (7) in Figure 4.10) with complexity $O((n_1 + n_2)\log(n_1 + n_2))$, or simpler if we assume $n = n_1 = n_2$ we get $O(n\log n)$. The rest of the algorithm makes just two serial passes of the merged list with linear time; so it does not affect the total complexity. Even though this algorithm has the same complexity the needed time is expected to be better, as the binary searches are no more needed. The experimental results is the next step to come and argue in favour of this variation of the algorithm.
4.5. APPROXIMATION ALGORITHMS

Alg.C-like SignatureMapping2

Input: two sets of bnodes $B_1$ and $B_2$, where $|B_1| < |B_2|

Output: a bij. $M$ between $B_1$ and $B_2$

1. $M = \emptyset$
2. $BS_1 = BS_2 = emptybag$
3. for each $b_1 \in B_1$
4. \hspace{1em} $BS_1 = BS_1 \cup \{\text{Signature}(b_1)\}$
5. for each $b_2 \in B_2$
6. \hspace{1em} $BS_2 = BS_2 \cup \{\text{Signature}(b_2)\}$

//sorts the signatures of the two bags
(7) $BS = \text{sort}(BS_1 \cup BS_2)$

//exact signature matchings
(8) $n = |BS|
(9)$ for ($m = 1$ to $|BS_1|$)
(10) $i = 0$
(11) while ($i < n$)
(12) \hspace{1em} if ($BS[i].\text{prov}! = BS_1$) $i++$
(13) \hspace{1em} $s_1 = BS[i]$
(14) \hspace{1em} $j = i + 1$
(15) \hspace{1em} while ($j < n$)
(16) \hspace{2em} if ($s_1 == BS[j]$)
(17) \hspace{3em} $M = M \cup (s_1, BS[j])$
(18) \hspace{3em} $\text{BS.remove}(s_1)$
(19) \hspace{3em} $\text{BS.remove}(BS[j])$

//closest lexicographical matchings
(21) $n = |BS|
(22)$ for ($m = 1$ to $|BS_1| - |M|$)
(23) $i = 0$
(24) while ($i < n$)
(25) \hspace{1em} if ($BS[i].\text{prov}! = BS_1$) $i++$
(26) \hspace{1em} $s_1 = BS[i]$
(27) \hspace{1em} $j = i + 1$
(28) \hspace{1em} while ($j < n$)
(29) \hspace{2em} if ($BS[j].\text{prov}! = BS_2$) $j++$
(30) \hspace{2em} $s_{2next} = BS[j]$
(31) \hspace{2em} $j = i - 1$
(32) \hspace{2em} while ($j > 0$)
(33) \hspace{3em} if ($BS[j].\text{prov}! = BS_2$) $j--$
(34) \hspace{3em} $s_{2pre} = BS[j]$
(35) \hspace{3em} if ($s_{2next} == \text{null}$)
(36) \hspace{4em} $M = M \cup \{s_1, s_{2pre}\}$
(37) \hspace{4em} $\text{BS.remove}(s_{2pre})$
(38) \hspace{4em} $M = M \cup \{s_1, s_{2next}\}$
(39) \hspace{4em} $\text{BS.remove}(s_{2next})$
(40) \hspace{4em} else if ($|\text{elems}(s_1) - \text{elems}(s_{2next})| > |\text{elems}(s_1) - \text{elems}(s_{2pre})|$)
(41) \hspace{5em} $M = M \cup \{s_1, s_{2pre}\}$
(42) \hspace{5em} $\text{BS.remove}(s_{2pre})$
(43) \hspace{5em} else
(44) \hspace{6em} $M = M \cup \{s_1, s_{2next}\}$
(45) \hspace{6em} $\text{BS.remove}(s_{2next})$
(46) \hspace{5em} return $M$

Figure 4.12: A C-like version of SignatureMapping2
4.5.5 Comparing the approximation algorithms

In this section we are going to compare the algorithms \( \text{Alg}_{\text{Hung}} \) and \( \text{Alg}_{\text{Sign}} \) at a theoretical level. An analysis over their complexities has already been offered in the previous sections. It is clear that the \( \text{Alg}_{\text{Sign}} \) offers a lower time and main memory complexity. Now we are going to focus on the equivalence detection and delta reduction potential.

On equivalence detection potential

As regards the equivalence detection potential we get the following proposition:

**Theorem 5 (Comparing on Equivalence Detection Potential)** The \( \text{Alg}_{\text{Sign}} \) has the same equivalence detection potential with the \( \text{Alg}_{\text{Hung}} \).

**Proof:**

Let us suppose that we have two equivalent Knowledge Bases \((K_1 \text{ and } K_2)\). We have to investigate the following cases.

1st Case: The Knowledge Bases do not have any directly connected bnodes. In this case, the constructed signatures do not contain any blank node (as no bnode belongs to the direct neighborhood of a bnode). This means that during signature matching no assumptions have to be made about the treatment of the directly connected bnodes (in equivalence with the Optimal Hungarian). Let us focus on the signature matching phase and suppose that exact and closest signature mappings occur. Suppose a closest signature mapping occurs i.e. the signature \( s_1 \) of a bnode \( b_1 \) in \( K_1 \) is mapped to the signature \( s_2 \) of a bnode \( b_2 \) in \( K_2 \) and \( s_1 \neq s_2 \). The fact that \( s_1 \) was mapped to \( s_2 \) entails that there is no signature of \( K_2 \) that equals \( s_1 \), or that there are more signatures that are the same with \( s_1 \) in \( K_1 \) than there are in \( K_2 \) (so the rest of the same bnodes are already mapped and removed from the signature lists according to lines 9 - 14 of the Signature Mapping Algorithm). From the signature construction method we know that all triples of the direct neighborhood of \( b_1 \) are represented inside the \( s_1 \) in a deterministic manner, which is independent of the serialization order of these triples. In other words, two same signatures correspond to two bnodes with the same direct neighborhoods. As a result, we get that the closest signature matching entails that there is a bnode in \( K_1 \) with a direct neighborhood \( D_1 \) that does not exist in \( K_2 \) or that there are more bnodes in \( K_1 \) equivalent with \( D_1 \) than there are in \( K_2 \). Both entailments are wrong because \( K_1 \) and \( K_2 \) are equivalent. We conclude that the signature matching phase of two equivalent Knowledge Bases performs only exact signature mappings. For this subcase, this fact is equivalent to a total cost equal to zero, as an exact signature matching is equivalent with a zero edit distance and additionally is independent of the other blank node pairs. We conclude that for equivalent Knowledge Bases the \( \text{Alg}_{\text{Sign}} \) always finds the solution that has a zero total cost for the \( \text{Alg}_{\text{Hung}} \) and by extension it always finds the optimal bijection (according to Theorem 2 and Theorem 4).

2nd Case: The Knowledge Bases contain directly connected bnodes. Recall that signature construction phase represents each bnode in the direct neighborhood of a bnode with the same symbol. Consequently, all the connected bnodes are considered the same during the signature matching phase.
4.5. APPROXIMATION ALGORITHMS

phase, exactly like in the Alg\textsubscript{Hung}. This means that if we focus on the signature matching phase, it is clear that we only have exact signature mappings (similarly with the 1st Case). Each exact signature matching is equivalent to a zero edit distance between the bnodes of these signatures in the Alg\textsubscript{Hung}. As a result, the same bnode matchings are extracted for the Alg\textsubscript{Sign} and the Alg\textsubscript{Hung}. Remind that each Knowledge Base remains exactly the same (same serialization of bnodes) during the comparison with both the algorithms. ◦

On Delta Reduction Potential

The low time complexity of the Alg\textsubscript{Sign} makes its delta reduction potential more "vulnerable" to deviations from the optimal bijection.

Recall that binary search produces a mapping of a pair of signatures (i.e. blank nodes) giving priority to the sameness of their first part over their last. As a result, the order the triples are encoded inside the signature will impact on the mappings of the blank nodes. Consequently, its Delta reduction potential is built upon some assumptions based on predictions about the changes that we expect to be made on the direct neighborhoods of the blank nodes. This is the extra approximation factor in relation to the Alg\textsubscript{Hung} that justifies a probable bigger Delta.

Thus, in case of Knowledge Bases with no directly connected blank nodes the Alg\textsubscript{Sign} gives the same (optimal) or bigger Delta than the Alg\textsubscript{Hung}. However, in case of Knowledge Bases with directly connected blank nodes there are scenarios, where the Alg\textsubscript{Sign} exports bigger Deltas than the Alg\textsubscript{Hung}, but there are other scenarios where the Alg\textsubscript{Sign} exports smaller Deltas than the hung. The former scenarios are easily comprehensible from an intuitive perspective. In Figure 4.13 we offer one example for the latter scenario.

As you can see in Table 4.5.5 the exported Delta by Alg\textsubscript{Sign} is 50% smaller than that exported by Alg\textsubscript{Hung}.

So, we get that in case of directly connected blank nodes the extra approximation factor of the Alg\textsubscript{Sign} may impact even in a positive way to the exported Delta. The percentage of how smaller or bigger the Delta could be cannot be determined generally, only under specific domain statistics.

4.5.6 On the serialization of the Knowledge Bases

As we have already mentioned, the serialization of the two files may play an important role in the Delta reduction or the equivalence detection potential of both the algorithms. In particular, in case of equivalent KBs with directly connected blank nodes the only factor that restricts the Alg\textsubscript{Sign} and the Alg\textsubscript{Hung} algorithms from getting the optimal bijection is the serialization order of their blank nodes. In other words, the serialization order is the order of appearance of the blank nodes inside the file. Formally, we get:

**Theorem 6 (The serialization order as an approximation factor)** If two equivalent KBs have the same serialization order (for their blank nodes), both the Alg\textsubscript{Sign} and Alg\textsubscript{Hung}
Figure 4.13: Scenario where the Alg\textsubscript{Sign} gives a smaller delta than the Alg\textsubscript{Hung}

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Bnode Mappings</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg\textsubscript{Hung}</td>
<td>(_: 1, _ : 11)</td>
<td>Del((_: 6, \text{scOf, ex : ag} 1))</td>
</tr>
<tr>
<td></td>
<td>(_: 2, _ : 12)</td>
<td>Del((_: 6, \text{scOf, ex : ag} 2))</td>
</tr>
<tr>
<td></td>
<td>(_: 3, _ : 13)</td>
<td>Del((_: 6, \text{scOf, ex : ag} 3))</td>
</tr>
<tr>
<td></td>
<td>(_: 4, _ : 14)</td>
<td>Add((_: 1, \text{scOf, ex : ag} 1))</td>
</tr>
<tr>
<td></td>
<td>(_: 5, _ : 15)</td>
<td>Add((_: 1, \text{scOf, ex : ag} 2))</td>
</tr>
<tr>
<td></td>
<td>(_: 6, _ : 16)</td>
<td>Add((_: 1, \text{scOf, ex : ag} 3))</td>
</tr>
<tr>
<td></td>
<td>(_: 7, _ : 17)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(_: 8, _ : 18)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(_: 9, _ : 19)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(_: 10, _ : 20)</td>
<td></td>
</tr>
<tr>
<td>Alg\textsubscript{Sign}</td>
<td>(_: 1, _ : 16)</td>
<td>Del((_: \text{foaf : Maria, hasAgenda, } _ : 1))</td>
</tr>
<tr>
<td></td>
<td>(_: 2, _ : 12)</td>
<td>Del((_: 1, \text{friend, } _ : 5))</td>
</tr>
<tr>
<td></td>
<td>(_: 3, _ : 13)</td>
<td>Del((_: 1, \text{friend, } _ : 4))</td>
</tr>
<tr>
<td></td>
<td>(_: 4, _ : 14)</td>
<td>Del((_: \text{foaf : Nikos, hasAgenda, } _ : 6))</td>
</tr>
<tr>
<td></td>
<td>(_: 5, _ : 15)</td>
<td>Del((_: 6, \text{friend, } _ : 8))</td>
</tr>
<tr>
<td></td>
<td>(_: 6, _ : 11)</td>
<td>Del((_: 6, \text{friend, } _ : 9))</td>
</tr>
<tr>
<td></td>
<td>(_: 7, _ : 17)</td>
<td>Add((_: \text{foaf : Maria, hasAgenda, } _ : 6))</td>
</tr>
<tr>
<td></td>
<td>(_: 8, _ : 18)</td>
<td>Add((_: 6, \text{friend, } _ : 3))</td>
</tr>
<tr>
<td></td>
<td>(_: 9, _ : 19)</td>
<td>Add((_: 6, \text{friend, } _ : 4))</td>
</tr>
<tr>
<td></td>
<td>(_: 10, _ : 20)</td>
<td>Add((_: 1, \text{friend, } _ : 8))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add((_: 1, \text{friend, } _ : 9))</td>
</tr>
</tbody>
</table>

find the optimal bnode mapping.

Proof:

We will prove that if two equivalent KBs (\(K_1\) and \(K_2\)) have the same serialization order for their
4.6 Discussing Semantics and Inference Rules

Apart from the explicitly specified triples of a KB, other triples can be inferred based on the RDF/S semantics [?], or other custom inference rules. In some cases one may want to decide whether two KBs are equivalent or to compute their delta with respect to a
particular set of rules. In such scenarios, equivalence can be based again on the Def. 3 and the edit distance over nodes on the Def 4 with the only difference that the graphs should be completed according to the inferred triples. It follows that if the semantics is based on a set of inferences rules yielding a finite closure, then the graph is finite and thus our method can be applied. Some semantics offering finite closures are RDF/S semantics, Minimal RDFS semantics [? ], ter Horst’s pD* semantics and OWL 2 RL, or even applicationspecific like [? ].

It is worth mentioning, that the optimal bnode mapping over the complete graphs may be different from the optimal mapping when considering the explicit graphs. In the example of Figure 4.14, where fat arrows denote rdfs:subClassOf relationships and dotted arrows rdfs:typeof relationships, the bijection with the minimum cost over the explicit graphs (left) is \{(\_,1\_.4),(\_,2\_.3)\}, while at the completed graphs (right) the bijection with the minimum cost is \{(\_,1\_.3),(\_,2\_.4)\}

Furthermore, for checking equivalence (at the presence of bnodes) or computing deltas, one could use the reduced graphs in case they are unique (note that the reduction of a \(K_a\), is the smallest in size \(K_b\) that is equivalent to \(K_a\), i.e. \(K_a\) and \(K_b\) have the same closure).

![Figure 4.14: Comparing the explicit versus the complete graphs of two KBs](image)

### 4.7 Desired Delta Properties and BNode Matching

In this section we discuss bnode matching and bnode name tuning under the light of the delta properties described at [? , ? ]. The results are independent from the particular matching algorithm employed.

**Correctness.** This is the more important property for versioning and synchronization purposes. In general the correctness of a comparison function has to be studied in conjunction with the semantics of the change operations. According to [? ], a pair \((\Delta_x, U_y)\) where \(\Delta_x\) is a comparison function, and \(U_y\) is a change operation semantics, is correct if for any pair of knowledge bases \(K_1\) and \(K_2\), the application of \(\Delta_x(K_1 \rightarrow K_2)\) under \(U_y\) semantics on \(K\), denoted by \(\Delta_x^{U_y} (K_1 \rightarrow K_2)\), is equivalent to \(K_1\), i.e. \(\Delta_x^{U_y} (K_1 \rightarrow K_2) \sim K_2\). Finding a bnode mapping can be considered as a preprocessing step, a task that is carried out before a differential function is applied. It follows that all results of [? ], about
the correctness of deltas under different change operation semantics, hold without any change.

**Semantic Identity.** A differential function \( \Delta_x \) satisfies the property of semantic identity if \( K_1 \sim K_2 \) implies that \( \Delta_x(K_1 \rightarrow K_2) = \emptyset \). Those differential functions that satisfy the semantic identity property (i.e. \( \Delta_c, \Delta_d \) and \( \Delta_{dc} \)), preserve it under the application of bnode matching and bnode name tuning, if the bnode matching algorithm employed guarantees that isomorphism is detected whenever it exists. The adoption of \( Alg_{Sign} \) or \( Alg_{Hung} \) algorithms does not guarantee that isomorphism is always identified, as proposed by Theorem 6.

**Reversibility.** A differential function \( \Delta_x \) is reversible if \( Inv(\Delta_x(K_1 \rightarrow K_2)) = \Delta_x(K_2 \rightarrow K_1) \) where the inverse of a set of change operations \( U \) is the set of the inverse operations in \( U \), i.e. \( Inv(U) = \cup \{ Inv(u) \mid u \in U \} \), and \( Inv(Add(t)) = Del(t) \) and \( Inv(Del(t)) = Add(t) \). A differential function that is reversible (i.e. \( \Delta_c \) and \( \Delta_i \)) may not yield reversible deltas if applied on KBs with bnodes and bnode matching and bnode name tuning is applied. This is due to the different local names that a bnode can have in different KBs.

In our case we may have \( Inv(\Delta_x(K_1 \rightarrow K_2)) \neq \Delta_x(K_2 \rightarrow K_1) \) because the triples in \( \Delta_x(K_1 \rightarrow K_2) \) may refer to local bnode names of \( K_1 \), while the triples in \( \Delta_x(K_2 \rightarrow K_1) \) may refer to local bnode names of \( K_2 \). However the essential property (that reversibility aims at capturing), is the ability to have stored only one delta, i.e. \( \Delta_x(K_1 \rightarrow K_2) \), and from this delta to be able to move back and forth (i.e. from \( K_1 \) to \( K_2 \) and the vice versa).

Suppose the case where \( K_1 \) and \( \Delta_x(K_1 \rightarrow K_2) \) are stored. From these two we can compute \( K' \), specifically we can compute a KB, denoted \( K_{eq} \), that is equivalent to \( K_2 \) (assuming that \( \Delta_x \) is a differential function that ensures correctness). If we reverse \( \Delta_x(K \rightarrow K') \), and we apply \( Inv(\Delta_x(K \rightarrow K')) \) on \( K_{eq} \), then we will get a KB that is equivalent to \( K_1 \). This holds because the triples in the operations of \( \Delta_x(K_1 \rightarrow K_2) \) (and obviously in those of \( Inv(\Delta_x(K_1 \rightarrow K_2)) \)), refer to local bnode names of \( K_1 \). So those bnodes that participate in the bijection, will have the same local names in both \( K_{eq} \) and \( K_1 \). In conclusion, although the definition of reversibility (as defined at [?]) is not satisfied in case of bnode matching, we can enjoy the essential property of reversibility at the presence of bnodes. Below we provide a more general definition of reversibility for capturing our case.

**Def. 6 (Delta Reversibility (extended))**

A headed sequence of deltas \( (K_0, \Delta_{0 \rightarrow 1}, \ldots, \Delta_{m \rightarrow m+1}) \) is reversible (or bidirectional), if from any \( K_i \) we can reach a KB equivalent to \( K_{i+1} \) (by computing \( K_i \) and applying \( \Delta_{i \rightarrow i+1} \)) and from any \( K_{i+1} \) we can reach a KB equivalent to \( K_i \) (by computing \( K_{i+1} \) and applying \( Inv(\Delta_{i \rightarrow i+1}) \)) for all \( i = 0..m \).

**Composability.** Let us now discuss the property of composition. Consider two deltas \( \Delta_1 = \Delta^+_1 \cup \Delta^{-}_1 \) and \( \Delta_2 = \Delta^+_2 \cup \Delta^{-}_2 \), where the superscripts "+" denote additions and the superscripts "-" denote deletions. The composition of \( \Delta_1 \) and \( \Delta_2 \), denoted by \( \Delta_1 \circ \Delta_2 \), is a delta \( \Delta = \Delta^+ \cup \Delta^- \) defined as:

- \( \Delta^+ = (\Delta^+_1 \cup \Delta^+_2) - (\Delta^-_1 \cup \Delta^-_2) \)
• \( \Delta^- = (\Delta^-_1 \cup \Delta^-_2) - (\Delta^+_1 \cup \Delta^+_2) \)

Three or more deltas can be composed by first composing the first two, their result with the third, and so on. A differential function \( \Delta_x \) satisfies composability if: \( \Delta_x(K_1, K_2) \circ \ldots \circ \Delta_x(K_{n-1}, K_n) = \Delta_x(K_1, K_n) \).

The useful property that composability aims at capturing, is the ability to compute a sequence of successive deltas without having to materialize the involved intermediate RDF KB versions. The differential functions that satisfy composition are \( \Delta_e \) and \( \Delta_c \).

However if we apply them on KBs with bnodes, and adopt bnode matching, then can get non-composable deltas. For example assume that
\[
\Delta_e(K_1 \rightarrow K_2) = \{\text{Del}(_:1, \text{name}, \text{Tom})\},
\]
\[
\Delta_e(K_2 \rightarrow K_3) = \{\text{Add}(_:2, \text{name}, \text{Tom})\}.
\]
In this example we have
\[
\Delta_e(K_1 \rightarrow K_2) \circ \Delta_e(K_2 \rightarrow K_3) = \{\text{Del}(_:1, \text{name}, \text{Tom}), \text{Add}(_:2, \text{name}, \text{Tom})\}.
\]
Now suppose we compute \( \Delta_e(K_1 \rightarrow K_3) \) after first applying bnode matching. Bnode matching will return the bijection \( M=\{_:1 \leftrightarrow _:2\} \), and consequently we will get \( \Delta_e(K_1 \rightarrow K_3) = \emptyset \). We observe that \( \Delta_e(K_1 \rightarrow K_2) \circ \Delta_e(K_2 \rightarrow K_3) \neq \Delta_e(K_1 \rightarrow K_3) \). We conclude that with bnode matching composability is not satisfied. The reason is that given a sequence of KBs, the bnode matching between the first and the last KB can detect more matches than those which can be detected between successive KB.

We conclude that bnode matching does not affect the correctness and reversibility, properties of the differential functions. However a differential function that satisfies semantic identity continues to satisfy it after bnode matching only if the algorithms adopted for bnode matching guarantees that isomorphism is always detected. Composability is not satisfied.
Chapter 5

Experimental Evaluation

We performed experiments for evaluating the potential for delta reduction, equivalence detection and time efficiency, deviation from the optimal delta and scalability.

They were performed using Sesame RDF/S Repository (main memory), using a PC with Intel Core i3 at 2.2 Ghz, 3.8 GB Ram, running Ubuntu 11.10.

5.1 TestBed

In order to provide an integrated experimental evaluation experiments were performed over real and synthetic datasets.

5.1.1 Real Datasets

We used two real datasets available in the LOD cloud: the Swedish open cultural heritage dataset\(^1\), and the Italian Museums dataset\(^2\), published from LKDI\(^3\). From each one we downloaded two versions with a time difference of one week or month. A preprocessing was necessary for corrections (e.g. missing URIs for some classes) and for merging the files. The features of these two datasets are given in Table 5.1. In both datasets there are no directly connected blank nodes.

5.1.2 Synthetic Datasets

Although semantic data generators already exist in the bibliography, none of them deals with the blank node connectivity issues. Therefore we designed and developed a synthetic generator over the Lumb generator (Univ-Bench Artificial data generator) \(^?\).

\(^1\) http://thedatalib.org/dataset/swedish-open-cultural-heritage used from http://kringla.nu/kringla/ for providing information on cultural data of Sweden.

\(^2\) http://thedatalib.org/dataset/museums-in-italy

\(^3\) http://www.linkedopendata.it/
Generally, the generator should provide a quite random and repeatable data generation over a single ontology of moderate complexity and size. The ontology was thought to be part of a specific domain (in our case the university domain), so as to attach a piece of realism to our data.

Lumb was originally proposed for benchmarking the performances of OWL (or DAML) repositories which comprises:

- an OWL schema, called univ-bench (UB), for describing universities, departments and related activities,

- a synthetic generator of instances of the UB schema and

- a set of fourteen queries expressed in SPARQL3.

We rely on a version of the UB schema, which is based on RDF/S entailment rules, such as rdf:type, rdfs:property, rdfs:domain, rdfs:range, rdfs:Class and rdfs:Literal. The RDFS version of the UB schema we use is depicted in Figure 5.1. We also use the LUBM synthetic instance generator. By giving some parameters to the system, we control:

- the number of instances, which are blank nodes inside the dataset, as well as

- the density of the blank node structures, i.e. how the blank nodes are connected between them.

Each dataset corresponds to a department of the University and is stored in a different RDF/XML file.

We perform two rounds of generations. In the first round we generate a set of datasets such that all of them contain the same number of blank nodes but gradually their blank node structures become more complex (i.e. more dense in blank nodes). In the second round we generate a set of datasets with quite simple blank node structures but we gradually increase the number of blank nodes. More details about each round of generations are given in Section 5.3.

---

Table 5.1: Features of two real LOD datasets

<table>
<thead>
<tr>
<th></th>
<th>Swedish File 1</th>
<th>Swedish File 2</th>
<th>Italian File 1</th>
<th>Italian File 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>15/10/11</td>
<td>22/10/11</td>
<td>2/11/11</td>
<td>4/12/11</td>
</tr>
<tr>
<td>[Triples]</td>
<td>3,750</td>
<td>3,589</td>
<td>49,897</td>
<td>49,897</td>
</tr>
<tr>
<td>[BNodes]</td>
<td>535</td>
<td>509</td>
<td>6,390</td>
<td>6,390</td>
</tr>
<tr>
<td>[Tripleswithbnodes]</td>
<td>77.7%</td>
<td>77.2%</td>
<td>43.85%</td>
<td>43.85%</td>
</tr>
<tr>
<td>Total Size</td>
<td>378 KB</td>
<td>365 KB</td>
<td>5.49 MB</td>
<td>5.46 MB</td>
</tr>
</tbody>
</table>
5.2 Evaluation: not directly connected bnodes

Experiments were conducted with and without bnode matching. Regarding matching we tested: (a) the random, (b) the Hungarian, and (c) the Signature-based mapping methods. Regarding the proposed variation of the Signature-based algorithm, experiments were not conducted, because of time constraints. In future we plan to evaluate it, too. The results are shown in Table 5.2. The first rows show the size of the yielded deltas and the last rows the time required for loading the bnodes (BLoad), constructing signatures (SC), bnode mapping (BM), delta computation (Diff), bnode name tuning (Tuning Time), and the total time. We observe that the algorithms provide a delta of 12.7 to 7,294 times smaller than without bnode mapping. AlgHung yields an equal (for the Italian) or smaller (0.34 times smaller for the Swedish) delta than AlgSign, but it requires more time (from 15 to 624 times).

Table 5.2: Experimental results over real datasets

<table>
<thead>
<tr>
<th></th>
<th>Swedish without BM</th>
<th>Swedish with BM (bnode matching)</th>
<th>Italian without BM</th>
<th>Italian with BM (bnode matching)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random AlgHung AlgSign</td>
<td>Random AlgHung AlgSign</td>
<td>Random AlgHung AlgSign</td>
<td>Random AlgHung AlgSign</td>
</tr>
<tr>
<td>Added</td>
<td>2,805 2,726 75 127</td>
<td>21,885 19,762 3 3</td>
<td>2,805 2,726 75 127</td>
<td>21,885 19,762 3 3</td>
</tr>
<tr>
<td>Deleted</td>
<td>2,966 2,887 236 288</td>
<td>21,885 19,762 3 3</td>
<td>2,966 2,887 236 288</td>
<td>21,885 19,762 3 3</td>
</tr>
<tr>
<td>BLoad Time(msec)</td>
<td>631 630 634</td>
<td>428 423 421</td>
<td>631 630 634</td>
<td>428 423 421</td>
</tr>
<tr>
<td>SC Time(msec)</td>
<td>- 210</td>
<td>- 840</td>
<td>- 210</td>
<td>- 840</td>
</tr>
<tr>
<td>BM Time(msec)</td>
<td>1,3 1,3 130</td>
<td>4.9 576,592 82.5</td>
<td>1,3 1,3 130</td>
<td>4.9 576,592 82.5</td>
</tr>
<tr>
<td>Diff Time(msec)</td>
<td>64 30 47</td>
<td>166 169 163</td>
<td>64 30 47</td>
<td>166 169 163</td>
</tr>
<tr>
<td>Tuning Time(msec)</td>
<td>15 0.2 0.5</td>
<td>3,332 9.4 9.5</td>
<td>15 0.2 0.5</td>
<td>3,332 9.4 9.5</td>
</tr>
<tr>
<td>Total Time(msec)</td>
<td>57 593 5,024</td>
<td>147 3,935 1,521</td>
<td>57 593 5,024</td>
<td>147 3,935 1,521</td>
</tr>
</tbody>
</table>
5.3 Evaluation: directly connected bnodes

5.3.1 On Complex Bnode structures

Let \( \text{Nodes} \) be the set of all nodes in the graph, \( B \) be the set of bnodes (\( B \subseteq \text{Nodes} \)), and \( \text{conn}(o) \) be the nodes of \( G \) that are directly connected with a node \( o \in \text{Nodes} \). We define \( b_{\text{density}} \) as:

\[
b_{\text{density}} = \text{avg}_{b \in B} \frac{|\text{conn}(b) \cap B|}{|\text{conn}(b)|}
\]

Note that if there are no directly connected bnodes then \( b_{\text{density}} = 0 \). The extended generator can create datasets with the desired \( b_{\text{density}} \) and the desired maximum length of paths that consist of edges that connect bnodes (we denote by \( b_{\text{len}} \) their average). Using the synthetic generator, we created a sequence of 9 pairs of KBs (each pair has two subsequent versions of a KB). For instance, the first KB is \( K_0 \) and its pair is \( K_0' \). Each time we compare the subsequent versions of a pair with respect to mapping time and yielded delta size. From now on we express the delta size as a percentage of the number of triples of the KB, i.e. as

\[
\frac{|\Delta_e(K, K')|}{|K| + |K'|}
\]

Table 5.3 shows the blank node properties of each pair of KBs, its optimal delta size over its subsequent version (known by construction) and its variation over the next pair of KBs (we call \( b_{\text{Neighborhood}} \) every subgraph having as nodes only bnodes, and we call \( b_{\text{named}} \) triple every triple that contains one bnode). With \( D_a \) we denote the average number of direct edges of the bnodes (i.e. average number of triples to which a bnode participates).

|   | \( |\text{triples}| \) | \( |B| \) | \( D_a \) | \( b_{\text{density}} \) | \( b_{\text{len}} \) | Optimal delta size | Variation |
|---|---|---|---|---|---|---|---|
| \( K_{0e} \) | 5,846 | 240 | 13.4 | 0 | 0 | 1% | No connected blank nodes |
| \( K_{1e} \) | 5,025 | 240 | 10.5 | 0.1 | 1 | 0.5% | \( b_{\text{Neighborhoods}} \) of 2 bnodes, reduced \( b_{\text{named}} \) triples |
| \( K_{2e} \) | 2,381 | 240 | 7 | 0.15 | 1 | 1.5% | Reduced \( b_{\text{named}} \) triples |
| \( K_{3e} \) | 1,628 | 240 | 5 | 0.2 | 1 | 1.5% | Reduced \( b_{\text{named}} \) triples |
| \( K_{4e} \) | 1,399 | 240 | 4 | 0.25 | 1.15 | 1.7% | Reduced \( b_{\text{named}} \) triples |
| \( K_{5e} \) | 919 | 240 | 3 | 0.32 | 1.15 | 3.2% | \( b_{\text{Neighborhoods}} \) of up to 15 bnodes, reduced \( b_{\text{named}} \) triples |
| \( K_{6e} \) | 909 | 240 | 3.25 | 0.4 | 1.35 | 2.7% | Connect \( b_{\text{Neighborhoods}}, \) reduced \( b_{\text{named}} \) triples |
| \( K_{7e} \) | 1,001 | 240 | 3.94 | 0.5 | 21.5 | 2.5% | Connect \( b_{\text{Neighborhoods}} \) |

Table 5.3: Blank node Features of the synthetic dataset
5.3. EVALUATION: DIRECTLY CONNECTED BNODES

5.3.2 Delta Reduction Potential

Figure 5.2 gives the delta reduction potential of each algorithm in logarithmic scale. Without bnode mapping the delta size ranges from 95% (for the second pair of KBs) to 143% (for the ninth pair of KBs). Instead for \( \text{Alg}_\text{Hung} \) it ranges from 0.47% to 10.67% and for \( \text{Alg}_\text{Sign} \) it ranges from 1% to 11.5%. Notice that \( \text{Alg}_\text{Sign} \) does not reduce the delta to the optimal for any pair of datasets, while \( \text{Alg}_\text{Hung} \) achieves the optimal delta for most of the pairs.

Figure 5.3 shows the delta reduction potential for the same pairs with the difference that the two bnode lists are not scanned in the original order (as in the left figure), but the second list is reversed. We notice that as the areas of directly connected bnodes become bigger (after the sixth pair of datasets), we get different (here higher) deltas. In such areas the direct neighborhoods lose their discrimination ability and thus the delta reduction potential becomes more unstable, increasing the probability to get a bigger delta.

![Delta Reduction Potential graph](image)

Figure 5.2: Delta Reduction Potential with same serialization order

If we use the optimal delta as baseline, and compute the percentage \( \frac{|A| - |\Delta_{opt}|}{|\Delta_{opt}|} \), in the first diagram this percentage for \( \text{Alg}_\text{Hung} \) falls in \([0, 2.88]\), while the \( \text{Alg}_\text{Sign} \)'s percentage falls in \([0.4, 3.2]\) (in the second diagram they fall in \([0, 8]\) and \([0.4, 8]\) resp.).

5.3.3 Time Efficiency

Figure 5.4 shows the mapping times of each algorithm in logarithmic scale. \( \text{Alg}_\text{Sign} \) gives two orders of magnitude lower mapping times.

5.3.4 Equivalence Detection Potential

Regarding equivalent KBs, if there are no directly connected bnodes then \( \text{Alg}_\text{Hung} \) detects them at polynomial time (recall Th. 4). To investigate what happens if there are directly
CHAPTER 5. EXPERIMENTAL EVALUATION

Figure 5.3: Delta Reduction Potential with reverse serialization order

Figure 5.4: Mapping times over the synthetic datasets

Figure 5.5: Mapping times over large synthetic datasets
5.4 SCALABILITY

connected bnodes we compared the pairs \((Kia, Kia)\) for \(i=0\) to 8 of the synthetic KBs. In case of similarly ordered bnode lists both \(Alg_{Hung}\) and \(Alg_{Sign}\) detected equivalences for all the KBs, confirming Theorem 6, while for reverse scanned bnodes lists they detected 5 of the 9 equivalences. They did not detect equivalences for the KBs with \(b_{density} \geq 0.25\).

5.4 Scalability

To investigate the efficiency of \(Alg_{Sign}\) in bigger datasets, we created 7 pairs of KBs: the first pair contains 23,827 triples and 2,400 bnodes, the second pair has the double number of triples and bnodes, and so on, until reaching the last pair containing 153,600 bnodes. From Fig. 5.5 we can see that the mapping time for \(Alg_{Sign}\) was only 10.5 seconds for the seventh pair of KBs (153,600 bnodes). \(Alg_{Hung}\) could not be applied even to the third pair of KBs due to its high (quadratic) requirements in main memory space.

5.5 Measuring the approximation

The upper bound of the reduction of the delta size that can be achieved with bnode matching is the min number of bnodes of the two KBs multiplied by their average degree. Experimentally we have investigated whether \(b_{density}\) (which is zero if there are no directly connected bnodes, and equal to 1 if all nodes are bnodes as in the proof of Th. 3), is related with the deviation from the optimal delta

\[
d_x = \frac{\vert \Delta_x \vert - \vert \Delta_{opt} \vert}{\vert \Delta_{opt} \vert + 1}
\]

This deviation depicts how many times bigger is \(\vert \Delta_x \vert\) than \(\vert \Delta_{opt} \vert\). Results over equivalent and non-equivalent KBs are shown at Figure 5.7. Both algorithms give a much smaller deviation from optimal than without bnode matching (its \(d_x\) ranges [47,114]). We also observe that keeping the original order of the bnodes is beneficial for both algorithms. For the non equivalent KBs the \(Alg_{Hung}\) gives always equal or (mostly) smaller delta than the \(Alg_{Sign}\), while for the equivalent both algorithms give exactly the same deviation.

5.6 Synopsis of the results

In real datasets with no directly connected blank nodes \(Alg_{Sign}\) was two orders of magnitude faster than \(Alg_{Hung}\) (less than one second for KBs with 6,390 bnodes), but yielded up to 0.34 times (or 34%) bigger deltas than \(Alg_{Hung}\), i.e. than the optimal mapping. \(Alg_{Hung}\) also identified all equivalent KBs.

For checking the behavior of the algorithms in KBs with directly connected bnodes, we
created synthetic datasets, over which we compared \textit{Alg}_{Sign} and the \textit{Alg}_{Hung} approximation algorithm. \textit{Alg}_{Hung} yielded from 0 to 3 times smaller deltas than \textit{Alg}_{Sign}, but the latter was from 18 to 57 times faster. \textit{Alg}_{Sign} requires only 10.5 seconds to match 153,600 bnodes.
Chapter 6

System and Applications

“'The desire that guides me in all I do is the desire to harness the forces of nature to the service of mankind.”’
-Nicola Tesla

Here we discuss the system that we have implemented and its applicability in various applications.

6.1 Functionality

We have designed and implemented a tool, called BNodeDelta that supports all the blank node matching algorithms that have been described.

This tool contains two different versions. The first version is a command line version and the second one is a web based application. At this point we are going to give a brief presentation of these two versions.

**Web Based Application**

The web based version is available in [http://www.ics.forth.gr/is1/BNodeDelta](http://www.ics.forth.gr/is1/BNodeDelta). Initially the user/client gives the two Knowledge Bases to be compared (which can be given either as paths to local files of the client or through URLs). Then the server uploads the Knowledge Bases through FTP and imports them in the local main memory repository or fetches them from the network through HTTP respectively.

Figure 6.1 shows a screen shot, where the user has uploaded the two Knowledge Bases and selected their RDF format.

Afterwards, the user/client specifies the blank node matching algorithm to be used and selects some extra output options. Figure 6.2 shows such a screen shot.

After these two steps, the user/client submits the request and the BNodeDelta outputs:
Figure 6.1: Importing the KBs on BNodeDelta

- a file with the deleted and the added statements (or one file with the deleted statements and one file with the added statements) in the selected RDF format
- statistics regarding the delta
- a file with statistics regarding the KBs (optional)
- a static visualization of the delta as a graph, where the red parts are the deleted
6.1. FUNCTIONALITY

triples and the green parts are the added triples (only in case the delta size is less than sixty triples)

Figures 6.3 and 6.4 show the exported results from the blank node matching over the Address KBs using the AlgHung.

---

**Summary of results**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bnodes in Knowledge Base 1</td>
<td>&gt; 3</td>
</tr>
<tr>
<td>Bnodes in Knowledge Base 2</td>
<td>&gt; 3</td>
</tr>
<tr>
<td>Loading Bnodes time</td>
<td>201.326592 ms</td>
</tr>
<tr>
<td>Mapping time</td>
<td>3.912399 ms</td>
</tr>
<tr>
<td>Diff time</td>
<td>1.059485 ms</td>
</tr>
<tr>
<td>Tuning time</td>
<td>4.245882 ms</td>
</tr>
<tr>
<td>TOTAL TIME</td>
<td>519.945927 ms</td>
</tr>
</tbody>
</table>

**delta size**

**Deleted Statements:** 3

**Added Statements:** 3

---

Figure 6.3: Basic Statistics of the blank node matching

Figure 6.4: The exported files of BNodeDelta

An optional step can take as input a namespace mapping table (if a namespace nm1 is mapped to a nm2 then they are considered equal at the comparison phase). Such a phase
can be really useful, in case the namespace URI references has changed from the one version to the other, but we do not want this kind of change to impact the exported delta.

**Command line tool**

Regarding the command line version of BNodeDelta, the user (human or other program) can execute the jar file through the command line. The input is now given as parameters of the jar file. In Figure 6.5 you can see the same use case run over the command line.

![Figure 6.5: Command line version of BNodeDelta](image)

**6.2 Architecture**

BNodeDelta was developed over the Sesame Virtuoso provider, which is a fully operational Native Graph Model Storage Provider for the Sesame Framework, allowing users of Virtuoso to leverage the Sesame framework to modify, query, and reason with the Virtuoso quad store using the Java language. Figure 6.6 shows an overview of Sesame’s architecture.

Sesame’s purpose is to provide a Java-friendly access point to Virtuoso. For the needs of BNodeDelta, Sesame is used as a Java library, providing tools to parse, interpret, query and store this information, embedded in BNodeDelta, while abstracting the details of the underlying machinery.

Starting at the bottom, the Storage And Inference Layer, or SAIL API, is an internal Sesame API that abstracts from the storage format used (i.e., whether the data is stored in an RDBMS, in memory, or in files, for example), and provides reasoning support. Each Sesame repository has its own SAIL object to represent it. For the storage of BNodeDelta’s data, we used the Main Memory object with its two different implementations. The first implementation is the one that operates directly on top of a SAIL object and the second operates as a proxy to a Sesame repository available on a remote server, accessible
6.3 APPLICATIONS

Figure 6.6: The Sesame component Stack

through HTTP.

On top of the SAIL, we find Sesame’s functional modules, such as the SeRQL, RQL and RDQL query engines, the admin module, and RDF export. *BNodeDelta* makes use of the SeRQL language.

### 6.3 Applications

Here we sketch how our work could be integrated in other systems.

- In the middle-ware

Jena is an open source Semantic Web framework for Java, grown out of work with the HP Labs Semantic Web Programme. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract “model that can be sourced with data from files, databases, URLs or a combination of these. Jena is actually similar to Sesame; though, unlike Sesame, Jena provides support for OWL (Web Ontology Language).

The Jena’s *rdfcompare* is a command line tool written in Java which loads two RDF files into Jena RDF models and uses an API call to check if the models are isomorphic. Although it seems to correctly tell whether two graphs are isomorphic, it doesn’t give any analysis of the difference between the files, like someone would expect from UNIX diff.

The existence of blank nodes is the main reason that frameworks have not elaborated integrated methods to help the development of versioning systems. However, this tool could make use of the *AlgSign* algorithm in order to provide a quite
small Delta in tolerable time.

Apart from Jena, the Sesame Framework could also take advantage of the proposed algorithms. Its AliBaba\(^1\) auditing repository is a repository wrapper to track changes, but is completely focused on provenance data in the RDF store.

- Business Cases
  There are many Knowledge Bases in the business, such as the GO database that are updated on a daily or weekly basis. The tracking of the changes between the versions could be really informative for the case where the users or even the administrators want to have a quick idea of the kind of changes that took place. The visualization of Delta could be quite helpful. At the moment there is an online tool, called Resource:Onto-Compare. It allows researchers to screen tens of thousands of genes simultaneously. Typically, they have been used in exploratory research to help formulate hypotheses. It actually compares commercially available microarrays based on GO. However, no direct versioning is available.

\(^1\)http://www.openrdf.org/doc/alibaba/2.0-rc5/alibaba-repository-auditing/
Chapter 7

Discussion on open issues

«Όταν γνωρίζεις κάτι, να επιμένεις ότι το γνωρίζεις.
Όταν πάλι δεν το γνωρίζεις,
αμολόγησε ότι δεν το γνωρίζεις.
αυτό είναι γνώση.»
-Κομφούκιος

At this chapter we are going to discuss some methods that have not yet fully realized or are under consideration, yet. In particular, in Section 7.1 we investigate a subcase of the problem, where it might be solved optimally. In Section 7.2 we suggest a new edit distance metric, that goes in more depth inside the blank node structures.

7.1 Optimally solved subcases

BNode Neighborhoods with bounded tree width

As we have already mentioned, surveying the documents containing blank nodes inside the Linked Data, almost the 58% of them had directly connected blank nodes. However, the 98.4% of those blank node structures were acyclic [? ]. The acyclicity of a structure is equivalent with a tree width up to 1.

Taking the above statistics into account, it is worth investigating the existence of tractable solutions for the wider case of RDF graphs with acyclic node neighborhoods.

For the case of equivalence detection potential this problem is already solved. [? ] claims that their algorithm correctly solves the RDF entailment problem between two RDF graphs with bounded tree width. However, approaching it as an optimization problem in order to find the minimum delta, makes things even harder.
7.2 Alternative approximation algorithms

Another issue that is still under debate is whether it would be beneficial to devise a different edit distance metric that will not consider only the direct neighbourhood of the blank nodes. Towards this, below we provide the definition of the Neighborhood of a blank node.

From the direct neighborhood graph of $b$, i.e. $DNG(b) = (N, E)$, we can define the neighborhood graph of $b$, denoted by $NG(b)$ as follows. If $N$ contains no other blank node than $b$ (i.e. if $(N - \{b\}) \cap B = \emptyset$), we set $NG(b) = DNG(b)$. Otherwise we extend the graph by adding the $DNG(b')$ of each $b' \in (N - \{b\}) \cap B$. Specifically, the extension of the graph can be described in an incremental way as follows: $NG_0(b) = DNG(b)$ and $NG_{i+1}(b) = NG_i(b) \cup \{ DNG(b') | b' \in (N_i \cap B) - B_{used} \}$ where $B_{used}$ is the set of blank nodes for which their $DNG$ has already been used, and $N_i$ the set of nodes of $NG_i(b)$. The extension stops when we cannot extend the $NG$ any more. For example, for the $K$ shown at Figure 7.1 (left) the graph $DNG(_,2)$ is: $N = \{foaf: Orville, _, 2, _, 5\}$ and $E = \{(foaf: Orville, hasAgenda, _, 2), (_, 2, friend, _, 5)\}$, while the graph $NG(_,2)$ is: $N = \{foaf: Orville, _, 2, _, 5, John\}$ and $E = \{(foaf: Orville, hasAgenda, _, 2), (_, 2, friend, _, 5), (_, 5, name, John)\}$.

![Figure 7.1: Two Knowledge Bases with directly connected blank nodes](image)

**Greedy Bnode Distance (GDist)**

Regarding the new metric we have already defined $dist_h(b_1, b_2)$, however the problem is how to compute $h$. For this reason we introduce a distance function $GDist(b_1, b_2)$ as a means to eventually derive $h$ (note that $GDist$ does not take as input any bijection). During a call $GDist(b_1, b_2)$ the blank nodes $b_1$ and $b_2$ are assumed mapped, furthermore the blank nodes in their neighborhoods ($NG(b_1)$ and $NG(b_2)$ as in $dist_h$) are mapped in a greedy and recursive way. In particular, the implementation of $GDist$ is recursive, and during each recursive (i.e. nested) call, a new pair of blank nodes
(not already visited - in the context of the call $GDist(b_1, b_2)$) is added to a “local bijection”, say $L$. The criterion for selecting the pairs of blank nodes that will be used in a nested call, is connectivity through the same properties. For example, suppose that $K = \{(b_1, p_1, b_{1a}), (b_1, p_2, b_{1b})\}$ and $K' = \{(b_2, p_1, b_{2a}), (b_2, p_2, b_{2b})\}$, where all nodes referred by the property instances are blank nodes. In the context of the call $Gdist(b_1, b_2)$ we will have the following nested calls $Gdist(b_{1a}, b_{2a})$ and $Gdist(b_{1b}, b_{2b})$ since these pairs of blank nodes are connected through the same property with $b_1$ and $b_2$ respectively. If however, there are more than one choices, then all possible pairs will be called and the algorithm will select the pair that returned the smaller distance. For example, suppose that $K = \{(b_1, p_1, b_{1a}), (b_1, p_2, b_{1b})\}$ and $K' = \{(b_2, p_1, b_{2a}), (b_2, p_1, b_{2b})\}$. In this case we will have the nested call $Gdist(b_{1a}, b_{2a})$ and $Gdist(b_{1b}, b_{2b})$. The returned distance will be based on the choice that returned the minimum distance. In our case, both return the same distance.

Eventually $Gdist(b_1, b_2)$ is equal to $dist_h(b_1, b_2)$ where $h$ contains $(b_1, b_2)$ and the recursively formed mappings between the bnodes of their neighborhoods.

Below we will see how we exploit $GDist$ for forming the entire bijection $M$.

**Comparing $GDist$ to Simple Distances**

$GDist$ is more expensive than adopting a simple distance function that is based only on the direct neighborhoods of the compared bnodes (like the $dist_h$). This is the price to pay for having more chances to detect equivalence.

This can be made evident with a small example. Consider the distance function $dist_h$ such that $dist_h(b_1, b_2) = 0$ if $b_1$ and $b_2$ are connected through the same properties with exactly the same named nodes and they are (directly) connected through the same properties with the same number of blank nodes. Otherwise, $dist_h(b_1, b_2)$ equals to the edit distance between their direct and named neighborhoods plus the difference between the number of blank nodes with which they are connected through each occurring property. Obviously the computational cost of $dist_h$ is less than that of $GDist$. However the potential of $GDist$ in detecting equivalences is greater than that of $dist_h$.

For instance, Fig. 7.2 shows that $GDist$ and $dist_h$ yield different mappings for the KBs of Figure 7.1. However the delta size returned with both mappings is the same, and equals to 4.

Fig. 7.3 shows a variation of the KBs of Figure 7.1, such that $K_1$ and $K_2$ are equivalent. In this case $GDist$ will return the mapping that proves their equivalence, while $dist_h$ does not guarantee this. This means that $GDist$ will detect the equivalence (and thus yield an empty delta), while $dist_h$ with 50% probability will return the mapping that proves equivalence and with another 50% it will return a delta with size equal to 4. This is because all pairs between $\{\_ : 4, \_ : 5\}$ and $\{\_ : 9, \_ : 10\}$ have the same $dist_h$ value, and thus one randomly selected pair of mappings will be returned.

The above metric could be used when applying the algorithm $Alg_{Hung}$. More investigation on this issue would convince us for the efficiency of such an approach.
However, note that this approach does not guarantee optimality. Also note that the complexity can be exponential if the entire graph consists of blank nodes (as the scenario used in the proof of Theorem 3). Nevertheless, it is worth investigating this approach in the future.
Chapter 8

Conclusions and Future Work

“Our senses enable us to perceive only a minute portion of the outside world.”
-Nicola Tesla

In this thesis we showed how we can exploit blank node anonymity to reduce the delta size when comparing RDF Knowledge Bases. We proved that finding the optimal mapping between the blank nodes of two Knowledge Bases, i.e. the one that returns the smallest in size delta regarding the unnamed part of these Knowledge Bases, is NP-Hard in the general case, and polynomial in case there are not directly connected blank nodes. To cope with the general case we presented polynomial algorithms returning approximate solutions.

In real datasets with no directly connected blank nodes AlgSign was two orders of magnitude faster than AlgHung (less than one second for KBs with 6,390 bnodes), but yielded up to 0.34 times (or 34%) bigger deltas than AlgHung, i.e. than the optimal mapping. AlgHung also identified all equivalent KBs.

For checking the behavior of the algorithms in KBs with directly connected bnodes, we created synthetic datasets, over which we compared AlgSign and the AlgHung approximation algorithm. AlgHung yielded from 0 to 3 times smaller deltas than AlgSign, but the latter was from 18 to 57 times faster. AlgSign requires only 10.5 seconds to match 153,600 bnodes.

Figure 8.1 illustrates the main points of our work.

This is the first work on this topic. Several issues are interesting for further research. Most of them were already discussed in the previous chapter. For instance, it is worth investigating other special cases where the optimal mapping can be found polynomially (e.g. directly connected blank nodes that form graphs of bounded tree width). Another direction is to comparatively evaluate various (probabilistic) signature construction methods and greedy approximation algorithms.
CHAPTER 8. CONCLUSIONS AND FUTURE WORK

Figure 8.1: Synopsis of BNodeDelta contribution