Data Integration over Distributed and Heterogeneous Data Endpoints

Data Integration over Distributed and Heterogeneous Data Endpoints

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 4 februari 2014 om 15:00 uur

door **Samur Felipe CARDOSO DE ARAUJO**

Master of Science in Computer Science, Pontifical Catholic University of Rio de Janeiro,
geboren te Belo Horizonte, Minas Gerais, Brazil.

Dit proefschrift is goedgekeurd door de promotoren:
Prof.dr.ir. M.J.T. Reinders
Prof.dr.ir. A. P. de Vries

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. M.J.T. Reinders	Technische Universiteit Delft, promotor
Prof.dr.ir. A. P. de Vries	Technische Universiteit Delft, supervisor
Prof.dr. D. Schwabe	Pontifical Catholic University of Rio de Janeiro
Prof. dr. ir. P.M.G. Apers	University of Twente
Prof. dr. ir. A.van Deursen	Technische Universiteit Delft
Prof. dr. ir. F. van Harmelen	Vrije Universiteit Amsterdam
Assist.Prof. dr. ir A.J.H. Hidders	Technische Universiteit Delft
Prof.dr. A. Hanjalic	Technische Universiteit Delft (reservelid)

SIKS Dissertation Series No. 2014-08

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Published and distributed by: Samur Araujo
E-mail: samuraraujo@gmail.com

ISBN: 978-90-6464-752-9
Keywords: data integration, semantic web, rdf, structured data, distribute querying, string transformation.

"To attain knowledge, add things everyday. To attain wisdom, remove things every day."

Lao Tzu

Summary

Data integration is a broad area encompassing techniques to merge data between data sources. Although there are plenty of efficient and effective methods focusing on data integration over homogeneous data, where instances share the same schema and range of values, their applications over heterogeneous data are less clear. This thesis considers data integration within the environment of the Semantic Web. More particularly, we propose a novel architecture for instance matching that takes into account the particularities of this heterogeneous and distributed setting. Instead of assuming that instances share the same schema, the proposed method operates even when there is no overlap between schemas, apart from a key label that matching instances must share. Moreover, we have considered the distributed nature of the Semantic Web to propose a new architecture for general data integration, which operates on-the-fly and in a pay-as-you-go fashion. We show that our view and the view of the traditional data integration school each only partially address the problem, but together complement each other. We have observed that this unified view gives a better insight into their relative importance and how data integration methods can benefit from their combination. The results achieved in this work are particularly interesting for the Semantic Web and Data Integration communities.

Samenvatting

Data-integratie is een breed gebied dat technieken omvat voor het samenvoegen van data uit verschillende gegevensbronnen. Alhoewel er genoeg efficinte en effectieve methodes zijn die zich richten op data-integratie voor homogene data, waar instanties hetzelfde schema en bereik van waardes delen, is hun toepassing op heterogene data minder voor de hand liggend. Deze thesis beschouwt data-integratie binnen de context van het Semantic-Web. In het bijzonder introduceren wij een nieuwe architectuur voor instantie-matching die rekening houdt met de bijzonderheden van deze heterogene en gedistribueerde setting. In plaats van aan te nemen dat instanties hetzelfde schema delen werkt de voorgestelde methode zelfs als er geen overlap is tussen de schemas met uitzondering van een identificerend label dat matchende instanties delen. Bovendien hebben we de gedistribueerde aard van het Semantic-Web in beschouwing genomen om een architectuur voor te stellen voor algemene data-integratie dat on-the-fly werkt volgens het pay-as-you-go principe. We laten zien dat onze visie en die van de traditionele data-integratie school beide slechts een deel van het probleem afdekken, maar gezamenlijk elkaar complementeren. We hebben waargenomen dat deze genificeerde visie een beter inzicht geeft in hun relatieve belang en hoe data-integratie kan profiteren van hun combinatie. De resultaten die in dit werk zijn bereikt zijn bijzonder interessant voor de Semantic-Web en Data-Integratie gemeenschappen.

Acknowledgements

There are no proper words to convey my deep gratitude and respect for my research promoters and supervisors, Professor Marcel Reinders and Professor Arjen de Vries. Thank you for your support and help to make my thesis possible.

A very special thanks goes out to Dr. Jan Hidders for his support and collaboration during all my Ph.D, our great discussions always helped me to improve my work. I would like to express my gratitude to my co-author Dr. Duc-Thanh Tran for sharing his broad knowledge in the field and for his extremely valuable coaching in writing research articles. I am deeply indebted to Professor Daniel Schwabe who has supported me since the beginning of my research carrier. His co-supervision was fundamental on this thesis.

Appreciation also goes out to other members of EWI for the fellowship and collaboration: Bebei Hu, Erwin Leonard, Qi Gao, and Fabian Abel. Thanks also to all members of DMIR lab for the excellent working environment. Thanks Robbert for his technical assistance; Saskia and Esther for their secretarial support. Also I am thankful for the CWI group for their fellowship. I would like to express my appreciation for the essential support of Ilse Oonk and Sophie Ronde during my Ph.D.

I deeply thank my parents and family for their unconditional support. Finally, I want to acknowledge the support of my beloved Ekaterina Churakova, without her love and patience, this thesis could not have been finished.

Contents

List of Figures

List of Tables

To my family.

Chapter 1

Introduction

Data integration is the problem of combining data from different sources. It has been extensively studied by the database community for the last 30 years [1–3]. Currently, data integration becomes particularly important in the world of the Semantic Web [4], which aims at converting the dominating unstructured web of documents into a more structured web of data. The benefits of constructing a web of data could be immense, as it would allow applications to share and reuse data in an intricate decentralized network. Fig. 1.1 shows an example of such network in the Linked Open Data Project (LOD) [5], the most concrete realization of the Semantic Web.

Particularly, datasets in the web of data are connected by interlinking their individual instances and schemas, in a process usually referred to as *instance matching* [1, 2, 6, 7] and *schema matching* [8–12]; respectively. Differently from the database context, where datasets are homogeneous, in the web of data context datasets are heterogeneous [13], meaning that their instances and schemas largely vary. This intrinsic characteristic of heterogeneous data poses new problems, causing existing instance matching approaches to perform less well than expected. In addition, the decentralized nature of the web of data brings new challenges. For instance, in most of the cases data are only available via querying a remote data endpoint. Consequently, previous assumptions that data can be downloaded and processed locally no longer apply. Summing up, the nature of the heterogeneous data and the distributed architecture of the web of data are crucial aspects to be understood and considered in any instance matching approach focusing on operating over this setting.

Mostly, this thesis presents the results of our research, in which we propose a set of novel methods to improve state-of-art of instance matching over the web of data.

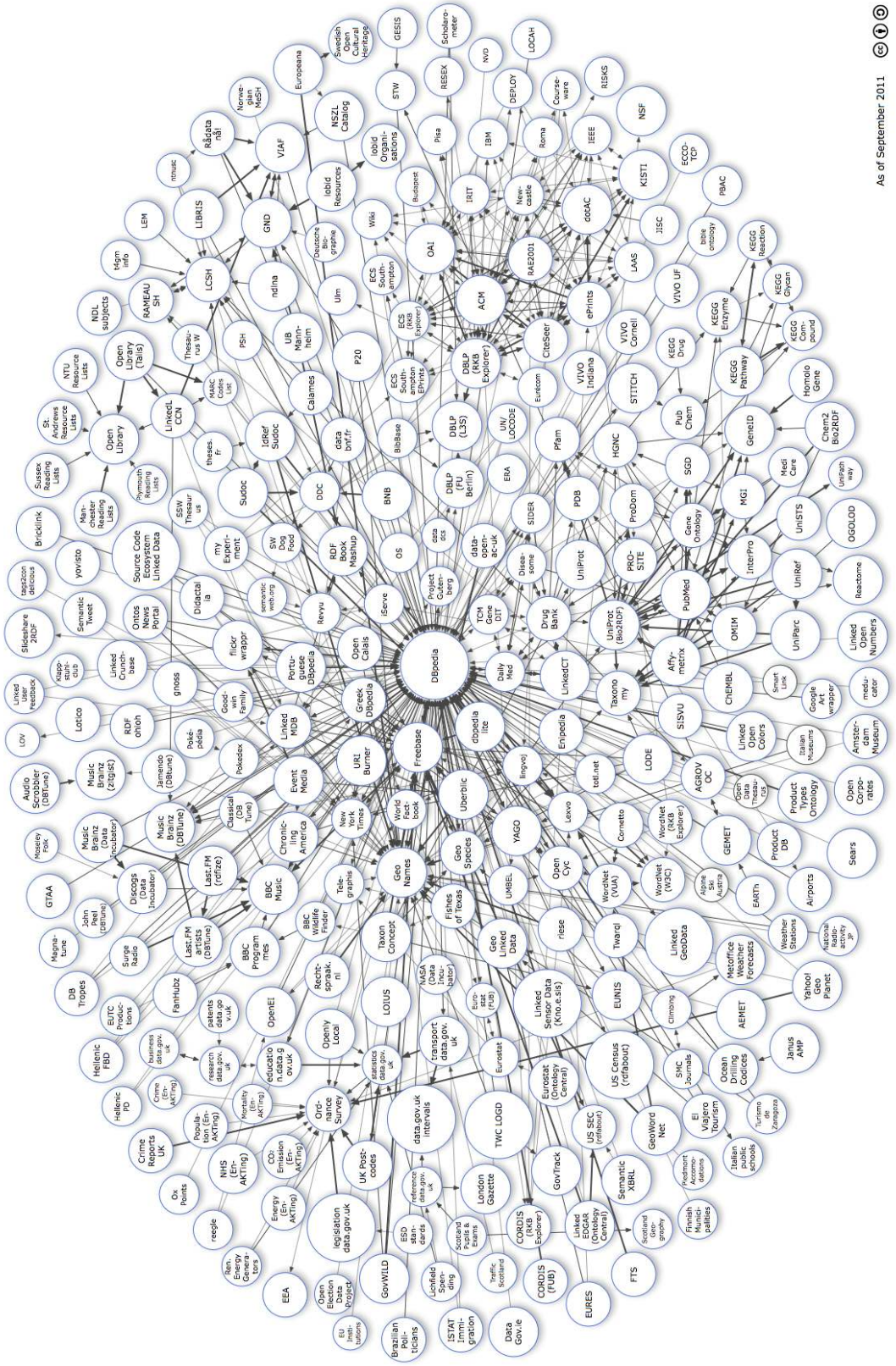

FIGURE 1.1: Snapshot of Linked Open Data Project in 2011.

1.1 A Brief Overview of Instance Matching

In this section, we informally introduce the instance matching problem and its challenges.

1.1.1 What is instance matching?

Instance matching is the problem of finding two or more distinct instance representations that refer to the same real world entity. For example, consider two representations of the city of Paris, one extracted from DBPedia.org[1] and the other one from Geonames.org[2].

Paris Dbpedia.org:

```
dbpedia:Paris rdfs:label          'Paris'
dbpedia:Paris dbpedia:populationTotal    2211297
dbpedia:Paris dbpedia:area        1.054e+08
```

Paris Geonames.org:

```
geonames:2988507 geo:name         'Paris'
geonames:2988507 geo:lat          '48.853'
geonames:2988507 geo:long         '2.349'
```

They both represent the city of Paris, but they use complete different schemas to describe different attributes of Paris. Instance matching allows a computer system to recognized that these two representations, in fact, refer to the same thing in the real world.

1.1.2 How does it work?

Generally, instance matching is applied over two distinct datasets containing thousands to millions of instances. The process of instance matching is usually divided in two steps: a *candidate selection* step and a *matching refinement* step. The candidate selection step aims to select from the entire target collection a few possible *candidate matches* for a source instance. The matching refinement step aims to find among the candidates those instances that are the same as a source instance. Basically, instances are considered the same if they share the same values on their attributes. This process requires their schemas to be previously aligned via schema matching.

[1] http://www.dbpedia.org
[2] http://www.geonames.org

1.1.3 Why is it challenging to do instance matching on Linked Data?

Data on Linked Data are heterogeneous, meaning that their schemas largely vary. In some cases, two instances only have an attribute that label an instance in common (e.g. `rdfs:label`). Usually, the value of this attribute is not enough to determine whether they match or not; consequently, more information in the data have to be considered in the process. An immediate issue due to the data heterogeneity is to determine the similarity between instances when their schemas do not align or overlap. This problem has received much less attention in the heterogeneous settings than in the homogeneous settings so far.

Another issue is that Linked Data cloud is a distributed environment, meaning that data are scattered in different datasets via internet. In some cases, these datasets are available only via querying a data endpoint. As these endpoints pose query limits and timeouts, an additional challenge is to do instance matching considering these constraints. To the best of our knowledge, we are the first to propose a fully automated instance matching solution in this context.

In this thesis, we elaborate on these issues, and we propose our solution.

1.2 Thesis Scope and Outline

1.2.1 Towards Self-Linking Linked Data

In **Chapter 2**, we describe an architecture for data integration over a distributed network of heterogeneous data, the Linked Data. The aim of this position chapter is to motivate the subsequent chapters. Mainly, the work done in this thesis are components of a visionary data integration architecture proposed in **Chapter 2**.

1.2.2 SERIMI: Class-based Matching for Instance Matching Across Heterogeneous Datasets

In **Chapter 3**, we introduce a novel method of instance matching, namely *class-based matching*. It is used to refined candidate matches obtained from a previous candidate selection step (discussed in the Chapter 4). Class-based matching is designed to work when there is no data or schema overlap between the source instance and target candidate matches. Basically, considering that source instances belong to a class of interest, class-based matching exploits the assumption that correct matches should also belong to a class (i.e. share some attribute/values in common). However, it does not assume that the

class semantics is explicitly given so that a direct matching at the class level is possible between the source (e.g. Drugs) and target (e.g. Medication). Then, by comparing the candidates, class-based matching can leverage a class of target candidates that are more likely to be the positive matches to the source instances. During this process, there is no comparison between source and target but only data from the target is used for matching. The main research question in Chapter 3 can be defined as follows:

How can we obtain correct matches for a set of source instances when there is little overlap between the source and target schemas?

1.2.3 Efficient and Effective On-the-fly Candidate Selection over Sparql Endpoints

In **Chapter 4**, we tackle the problem of candidate selection over the web of data. To do so, we cast the problem of candidate selection as querying over remote SPARQL endpoints for possible candidate matches for a given instance. The biggest obstacle is to create queries that can be effective in retrieving all and only the correct matches for a source instance, and that can also be time efficient. Consequently, the main research question in Chapter 4 can be defined as follows:

How can we obtain candidate matches for a set of source instance in an effective and time efficiency way, by querying a target remote endpoint?

In addition, as the source and target instances have a heterogeneous schema, *instance-based queries* are considered in this task, which are queries that use a local attribute of an instance of data, as opposed to an attribute that occurs in a global schema. To build these queries, we not only consider instances attributes but also class-related information learned on-the-fly from candidates obtained at query time, which are treated as training examples. Targeting both precision and recall requires dealing with a large number of candidate queries. To improve efficiency, we propose a *heuristic-based search optimization framework* that aims to select and execute a small number of queries considering quality of the results and run-time of the queries.

1.2.4 Learning Edit-Distance Based String Transformation Rules From Examples

In **Chapter 5**, we present a string transformation algorithm. The task of transforming a string from a source form into a target form is relevant for many information processing tasks. Particularly, we designed a string transformation algorithm to be applied in

the candidate selection process. It can be used to transform the attribute values in the source format to the target format (e.g. $Michael\ Jackson \rightarrow Jackson\ Michael$). In this way, *exact queries* can be constructed, which only retrieve candidates with an attribute value equal to a query attribute value. Exact queries are more precise and consequently more time efficient than queries that compare string values based on their similarity. Consequently, the integration of this algorithm in the candidate selection step could potentially improve the efficiency of the process.

We observed that our string transformation algorithm is quite generic and can be applied to a broader range of string transformation tasks than the one that initially motivated its development. Consequently, in Chapter 5, the string transformation algorithm is presented as a generic and task independent algorithm.

In Chapter 5, we are particularly interested in learning string transformation rules from a limited set of example transformations. Then, these learned rules can be used to transform a large amount of unseen strings that are similar to the original strings used as examples. Therefore, the main research question in Chapter 5 can be defined as follows:

How can we learn string transformation rules from a limited set of examples that can correctly transform a large amount of unseen strings similar to the examples?

1.2.5 Exercises on Knowledge Based Acceleration

Chapter 6 is not directly connected to the previous chapters. In that chapter, we describe the retrieval models used and the results obtained in the Knowledge Base Acceleration track in TREC 2012 (TREC-KBA). The TREC-KBA focuses on a single task: to filter a time-ordered corpus for documents that are highly relevant to a predefined list of Wikipedia entities. A successful KBA system must do more than matching an entity to the correct documents in the corpus: it must also distinguish centrally relevant documents that are worth citing in the entity's Wikipedia article.

Basically, we focus our attention on establishing matches between an entity and its related news documents, which is a challenging task. To do so, we exploited the web of data, enriching the original given entity descriptions with additional information that could help to identify the correct matches in this large stream corpus. In addition, we tried to model document centrality using an annotated set of examples. Overall, we obtained good results in this challenge, ranking among the top three best score systems. The results indicate that using the web of data to enrich the entity descriptions is a good strategy. Also, this helps to demonstrate the benefits of using structured information

sources in this task, such as Linked Data. The main research question investigated in Chapter 6 can be defined as follows:

How to use structured resources related to the entity to estimate centrality and relevance of news documents?

1.2.6 Conclusions

In the final **Chapter 7**, we describe the contributions of this thesis by addressing the main research questions of each chapter, and summarizing the findings. Also we discuss future research directions.

Chapter 2

Towards a self-linking Linked Data

2.1 Introduction

The vision of the Semantic Web [4], undoubtedly powerful, promises a structured web of data that would greatly improve the access to data by humans and machines. Currently, initiatives such as the Linked Open Data project [5] have published and interlinked hundreds of structured datasets following Semantic Web standards (e.g. RDF[1], OWL[2]). The set of these interlinked datasets forms a web of data called Linked Data. Two datasets are interlinked by connecting their objects through semantic links called *RDF links*. Theoretically, data users (humans and machines) can easily navigate from one dataset to another through these links, potentially exploring the entire Linked Data Cloud [14][15][16][17]. Unfortunately, in practice, the establishment of these RDF links has shown to be a non-trivial task [18][19][20][21]. So far, this issue is one of the factors [22][23] that has considerably limited the development of a global-scale interlinked dataspace, the Semantic Web.

Tummarello et al. [24] have discussed the interlinking issue, proposing a centralized data integration architecture to solve the poor interlinking in the Linked Data. However, this issue has not been resolved, even though we are already three years later. As a resolution, this thesis argues in favor of a decentralized data integration architecture for the Linked Data that can coexist with their centralized architecture. Additionally, we propose concrete components to be added to the Linked Data to make this vision a welcome reality.

[1] http://www.w3.org/RDF/
[2] http://webont.org/owl/1.1/

Mainly, RDF links are established by connecting two data objects that refer to the same world entity using the semantic predicate `owl:sameas`[3]. This *data integration process* is known as *instance matching* or *schema matching*, depending whether the process is applied at instance or schema level, respectively. Instance and schema matching have been studied extensively by the database community for the last 30 years. However, the heterogeneous and decentralized nature of the Linked Data pose additional challenges for data integration in this setting, where assumptions embodied in the existing methods no longer apply.

We argue that adopting existing data integration paradigms to the scenario of Linked Data considerably limits the interlinking of its datasets, especially regarding new datasets to be added to the cloud. Current methods require the data to be available locally, typical of a centralized and off-line dataspace; while, in the Linked Data, data are decentralized and scattered among many servers, in some cases only accessible via Semantic Web protocols, such as the SPARQL protocol [4]. This decentralized architecture requires that we develop ways of thinking about integration that are as rigorous as the existing paradigm, but different. They should incorporate characteristics that exist only in the Linked Data and are relevant for building the interlinks. To a large extent, it requires that we change our philosophy about data integration, in the full sense of the term.

We envision a more organic interpretation of the Linked Data architecture where each dataset in the cloud behaves as an independent organism having as one of its functions the ability of self-linking to other datasets in the cloud. The analogy of a dataset in the Linked Data would be a cell in a living organism. Analogous to *cell signaling* in a living cell (a communication mechanism that governs basic cellular activities and coordinates cell actions), a communication mechanism could orchestrate dataset interlinking, which would happen independent from human intervention. Notice that although a living cell behaves independently, intra-cellular structures guide cell behavior. Analogously, we propose here structures that should be part of a dataset in the Linked Data to guide its self-interlink behavior. Ideally, we would propose that a dataset should be automatically interlinked as soon as it is published (becomes "alive" in cloud), without human intervention. The ideas proposed here will play a crucial role to make this vision a reality. Fig. 2.1 shows the structures that we will introduce.

The envisioned architecture can boost interlinking in the Linked Data, greatly contributing to speed-up the vision of a Semantic Web. In this thesis, we describe the components of this architecture, and present a prototype tool as a proof of concept. We have evaluated this new paradigm on reference benchmarks in the field [25], and the results show

[3] We ignore that other RDF predicates can also act as RDF links (e.g. db:livesin).
[4] http://www.w3.org/TR/rdf-sparql-query/

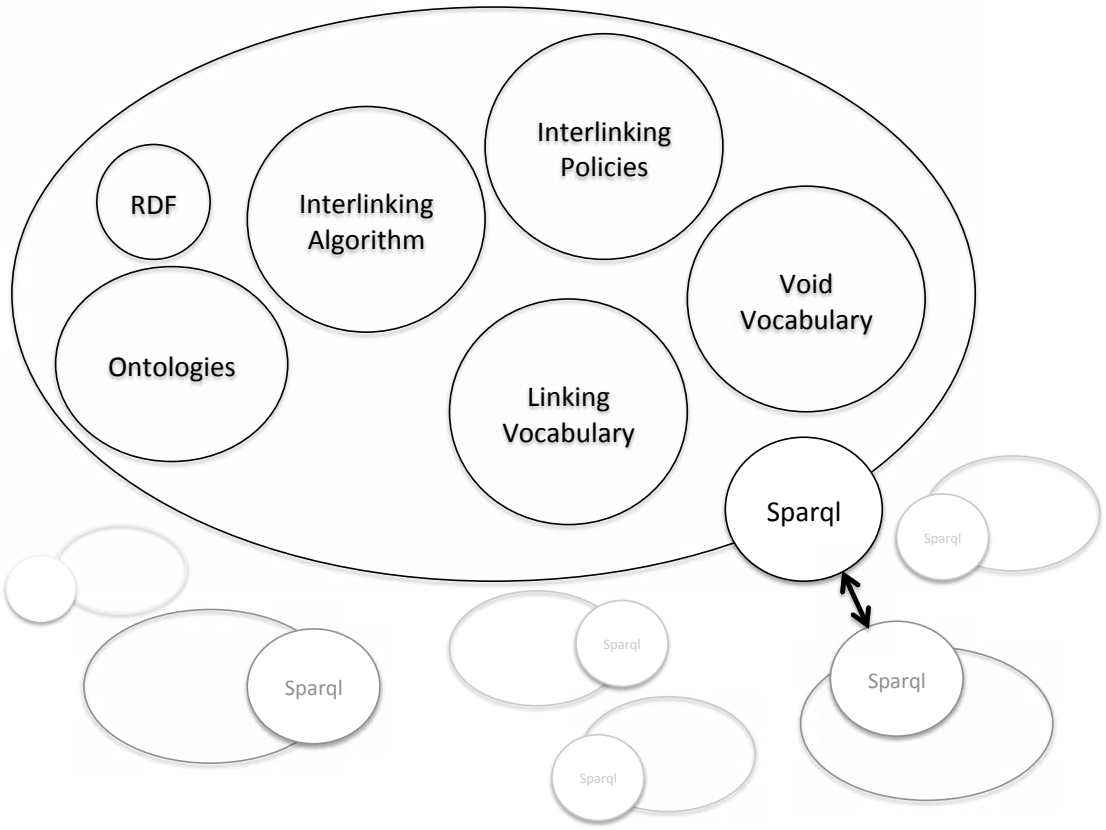

FIGURE 2.1: Overview of the structures of in self-linking Linked Data.

that this architecture is feasible and more time efficient than the traditional data integration methods in certain conditions. Concluding, it indicates that a self-linking behavior can be part of the Linked Data environment.

2.2 A General Architecture

Here we address our research questions: How can we boost the interlinking between datasets in the Linked Data?

2.2.1 Building a Self-Linking Linked Data

We argue that to create a truly linked data, datasets should be capable to self-link as soon as they are published. For that, the Linked Data architecture must include components to allow this to happen. In the foundation of these components lies a different data integration paradigm that we discuss next.

As we mentioned in the introduction, there is a predominant school of data integration that proposes an off-line approach for the problem. As a principle, the source and target

data to be integrated must be available locally so the data can be processed and links established. To use this method, datasets in the Linked Data have to be downloaded to a central server (or cluster) so that the interlinking can computed. Many authors [26] have successfully applied this paradigm on the Linked Data; however, they paid the price of having to allocate a large amount of computer resources and human labor to accomplish this task. More importantly, this strategy cannot be used for all datasets in the cloud, given that many are not available for download. Also, adding new data and modifying existing data will become increasingly more expensive.

Part of the problem with existing data integration paradigms lies in the fact that the Linked Data was designed to be accessed via querying a remote SPARQL endpoint or via *dereferencing* URIs; while, traditional data integration methods require all data to be fully available locally, to be indexed and processed [27][28][29][28]. Although data can be obtained by querying a remote endpoint, to download large datasets through this method is inefficient and quite often reaches timeouts imposed by the remote endpoints. Apart from that, dataset sizes largely vary on the Linked Data, consequently to interlink a small source dataset to large target dataset requires only part of the target data. In these cases, an efficient selection of the necessary target data was not considered as an important issue so far because it is assumed that the data is available for local processing, which is not always true in the Linked Data scenario.

For these reasons, we argue that the Linked Data requires a different paradigm of data integration that must operate directly over the SPARQL endpoint in an on-the-fly and pay-as-you-go fashion. We propose a set of principles to support this vision.

2.2.2 Interlinking as a Query Problem

We argue that interlinking can be done directly by querying the source and target endpoints, instead of fully downloading it and processing it locally. This has been underestimated or neglected so far.

From now on, we will mostly focus on the interlinking at instance level; however, the architecture that we propose here can be used for schema integration as well. We pose the problem of interlinking by querying Linked Data. Finding a target instance that matches a source instance should translate into issuing a SPARQL query in the target endpoint, selecting the target instance with attributes similar to the source instance [30, 31]. These *matching queries* contain a query pattern that identifies the source instances and can potentially retrieve candidate matches to these source instances.

Approaching the problem in this fashion, we benefit that the data have been already processed in each endpoint, avoiding the pre-processing (e.g., indexing, data cleaning) necessary in the traditional scenario. Consequently, accessing the data via query endpoints saves human labor and computer resources; more importantly, it removes the pre-processing step from the process, facilitating the implementation of a self-linking mechanism in the Linked Data.

Basically, at instance level, two data objects are considered the same if they share common attributes of data. Mainly, instance matching requires two data objects to be directly compared, in a process called *direct matching*. Instances are interlinked when their similarities are above a threshold. By analyzing the data, studies [1] showed that the attributes to be compared, the similarity functions and threshold can be determined, automatically, in an unsupervised fashion.

As the number of data objects may be large, to speed up the process of comparisons, most of the matching approaches split the problem of instance matching in two steps: *candidate selection* and *match refinement*. The candidate selection step uses a low cost method for fast retrieval of possible candidate matches for the source instances; the subsequent match refinement step uses more elaborate methods for detecting among the candidates the correct target matches for the source instances. The two step process reduces the number of comparisons necessary to find the matches, which initially would require $S \times T$ comparisons, where S and T are the number of instances in the source and target datasets, respectively.

In the architecture that we propose, candidate selection is done by querying the SPARQL endpoint. Once the candidates are selected, they are treated as the target dataset and the query refinement can be done using any of the data integration methods available in the literature. However, as this method evolves, the candidates obtained during the candidate selection step can be so precise that the refinement step will be unnecessary. The challenge is to build SPARQL queries to obtain the candidate matches, or the correct matches in the optimistic scenario. As a requirement, these queries must be effective in retrieving all the correct candidates but also must be executed efficiently.

In Chapter 4, we propose a SPARQL based candidate selection method, and we demonstrated that it produces good candidate matches, with high recall and precision. Apart from that, we have shown that this mechanism is more efficient that downloading the entire data, when a certain condition holds, i.e., when the datasets sizes largely varies.

Fig. 2.2 illustrates the configuration of the candidate selection and match refinement components in the Linked Data architecture.

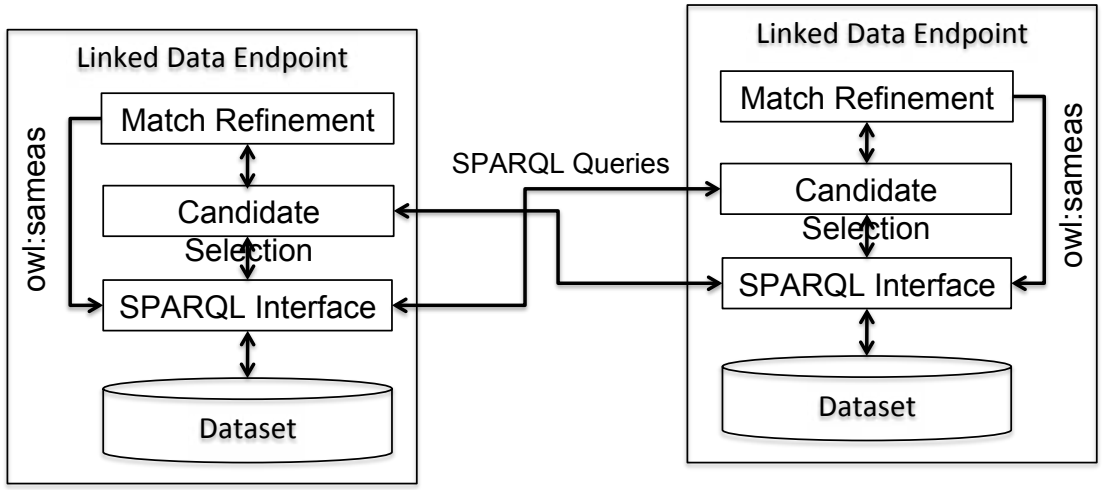

FIGURE 2.2: Interlinking Apparatus inside Linked Data Endpoints.

2.2.3 Self-linking Policies

On the top of the proposed components, data integration policies are applied to enable the endpoint to self-maintain its RDF links. The interlinking could be triggered at four distinct moments. It could be triggered by an explicit user command, automatically as soon as the data is published, every time that the data is updated in the endpoint, and when a new dataset is published in the cloud. These are basic policies that would allow the endpoints to self-maintain its RDF links, responding to any internal or external data update in the Linked Data. Additional policies could be defined to optimize the computation of the interlinks, for example, controlling the query load during the generation of the candidate matches. Practically, these policies should be described in a vocabulary to be specified and integrated in the design of the Linked Data. A standard policy, to be defined by the community, must be used to guarantee the self-linking behavior in the cloud.

2.2.4 SPARQL Extensions to Support Self-linking

To implement our vision transparently, the SPARQL language would have to be extended with a few primitives to support *approximate string matching*. Approximate String matching primitives are fundamental during the process of candidate selection because exact queries do not account for syntactical differences that exist between instances in datasets (e.g. Michael Jackson vs. Jackson Michael). Although the current SPARQL specification supports filters and regular expressions that can simulate approximate matching queries, many of the available implementations do not support efficient query processing techniques for queries using these operators. For this reason, we argue that it would help to add to the current specification new language operators (e.g., *like*,

ilike - ignore case version of *like*) that require an efficient implementation by future RDF store implementations. Currently, some RDF stores, such as Open Link Virtuoso[5], support an efficient computation of such approximate matching queries, using non-standard notations (e.g. bif:contains[6]). Nevertheless, this would have to exist in the entire Linked Data in a standard way so the proposed method can be truly functional.

We acknowledge that strategies for executing the matching queries in the federated fashion should be consider as well. This subject is well studied in the literature [32][33][34], and we assume here that the query engines are in charge to delegate the matching queries to relevant endpoints. However, how to optimize these strategies to this problem is an interesting research question.

2.2.5 Standardization of Interlinking Algorithms

We argue that a candidate selection and a matching refinement algorithm must be designed as a standard, so all RDF stores would have the expected behavior implemented. Most of the existing instance matching algorithms that target the Linked Data are converging. In their foundations, they all exploit the same meta-properties of the data (e.g. discriminative power and coverage of predicates) to determine RDF links [35][36][37][38][39]. Currently, the state-of-art interlinking methods can be formalized into a unique and default way of doing interlinking. This is a fundamental step to embed the self-linking behavior in the datasets in the Linked Data. Of course, particular interlinking methods focusing on more specific data (e.g. bio data, statistical data and stream data.) would continue coexisting with the standard method.

2.3 Proof of Concept

We illustrate a real case scenario to show the benefit of a self-linking Linked Data for an ordinary data owner.

Scenario. John has a collection of 5000 band names that he would like to know their member's names. He decides to make use of the Linked Data because he heard that it contains other datasets that could be used to enrich his own data. Then, he generates a single RDF triple for every band names. For example, `example:band1 rdfs:label "Metallica"`. He publishes these data using a RDF store that supports the self-linking behavior standard. As soon as the data is published, the RDF engine starts looking for possible target interlinks in the cloud. It finds the MusicBrainz dataset as a good

[5] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/
[6] http://docs.openlinksw.com/virtuoso/rdfsparql.html

candidate for interlinking because all of the band names (strings) in John's data also occur in this dataset. Immediately, it starts to interlink John Band's data to the Music Brainz's data, without John even being aware of it. A few minutes later, John queries his RDF store to check his data, and then he notices that his data have been already interlinked to MusicBrainz. Navigating through the just created RDF links, he goes from his data to MusicBrainz and from MusicBrainz to DBPedia, where unexpectedly he encounters not only information about the band member's names but their origin, discography, etc. Thanks to the Self-Linking Linked Data, now John can build a richer application over his collection than the one that he had initially conceived.

Let us start with a use a subset of Linked Data datasets loaded into the Open Linking Virtuoso server. As the band collection, we selected 5000 band names from the internet archive, which are available for download at GitHub[7]. To emulate the self-linking behavior, we use SONDA[8] as the candidate selection module and SERIMI[9] as the match refinement module, the open source implementations of Chapter 4 and Chapter 3, respectively. A single command obtains the RDF links, namely,

```
sondaserimi -source localhost:8890/sparql -target localhost:8891/sparql
```

where the two URIs represent John's data and Music Brainz data, respectively. Notice that only two URIs where given to this method. In a full implementation of the self-linking Linked Data, even the target URI would be discovery automatically.

To measure the quality of the interlinks produced, we sampled the data and check it manually. We obtained an accuracy of 93%. Although this exercise does not operate over a complete implementation of the presented concept, it shows that the vision of a self-linking Linked Data is feasible, and it brings immediate benefits.

2.4 Future Work

Chapter 4 discuss how interlinking can be done via SPARQL endpoints. However, at large scale, considering all the Linked Data, a few problems remain to be the addressed. Consider for example, the problem of selecting the dataset in the cloud to be interlinked to. LDIF is a framework that addressed some of these problems [40]. Although quite elaborated, the framework is still designed to be operated manually as it requires matching rules to be supplied by the data designer. Although elements existing in these

[7] https://github.com/samuraraujo/internetarchive
[8] https://github.com/samuraraujo/Sonda
[9] https://github.com/samuraraujo/SERIMI-RDF-Interlinking

approaches are necessary to build our vision, much more have to be considered. We propose that the community to develop a research agenda to identify and tackle all these issues.

Automatic and unsupervised techniques and algorithms to produce matching rules are available in the literature [41][38][35]. To accomplish our vision, as they share the same principles, a research agenda should be defined aiming to converge these techniques to a basic acceptable standard approach. Vocabularies to describe endpoint have been already proposed [42], they could be extended to support self-linking policies.

2.5 Conclusion

The success of Linked Data depends on pragmatic designing decisions putting the self-linking behavior at the foundation of the Linked Data ideology.

We have motivated our work with general considerations about the use of traditional data integration techniques on building Linked Data. We have focused on a family of interrelated problems that are centered on the notion of datasets self-linking in an on-the-fly fashion. As a result, we have obtained a data integration architecture to boost interlinking in the Linked Data, contributing to speed-up the realization of the vision of the Semantic Web.

Chapter 3

SERIMI: Class-based Matching for Instance Matching Across Heterogeneous Datasets

State-of-the-art instance matching approaches do not perform well when used for matching instances *across heterogeneous datasets*. This shortcoming derives from their core operation depending on *direct matching*, which involves a direct comparison of instances in the source with instances in the target dataset. Direct matching is not suitable when the overlap between the datasets is small. We propose a new paradigm called *class-based matching* to solve this problem. Given a class of instances from the source dataset, called the *class of interest*, and a set of candidate matches retrieved from the target, class-based matching refines the candidates by filtering out those that do not belong to the class of interest. For this refinement, only data in the target is used, i.e., no direct comparison between source and target is involved. Based on extensive experiments using public benchmarks, we show our approach greatly improves the results of state-of-the-art systems, especially on difficult matching tasks.

TABLE 3.1: Instances represented as RDF triples.

Source Dataset		
Subject	Predicate/Attribute	Object/Value
nyt:2223	rdfs:label	'San Francisco'
nyt:5962	rdfs:label	'Belmont'
nyt:5962	geo:lat	'37.52'
nyt:5555	rdfs:label	'San Jose'
nyt:4232	nyt:prefLabel	'Paris'
geo:525233	rdfs:label	'Belmont'
geo:525233	in:country	geo:887884
geo:525233	geo:lat	'37.52'
Target Dataset		
Subject	Predicate/Attribute	Object/Value
db:Usa	owl:sameas	geo:887884
db:Paris	rdfs:label	'Paris'
db:Paris	db:country	db:France
db:Belmont_France	rdfs:label	'Belmont'
db:Belmont_France	db:country	db:France
db:Belmont_California	rdfs:label	'Belmont'
db:Belmont_California	db:country	db:Usa
db:San_Francisco	rdfs:label	'San Francisco'
db:San_Francisco	db:country	db:Usa
db:San_Francisco	db:locatedIn	db:California
db:San_Jose_California	rdfs:label	'San Jose'
db:San_Jose_California	db:locatedIn	db:California
db:San_Jose_Costa_Rica	rdfs:label	'San Jose'
db:San_Jose_Costa_Rica	db:country	db:Costa_Rica

3.1 Introduction

A large number of datasets has been made available on the Web as a result of initiatives such as Linking Open Data. As a general graph-structured data model, RDF[1] is widely used especially for publishing Web datasets. In RDF, an entity, also called an instance, is represented via $\langle subject, predicate, object \rangle$ statements (called *triples*). Predicates and objects capture *attributes* and *values* of an instance, respectively (terms that are used interchangeably here). Table 3.1 shows examples of RDF triples.

Besides RDF, OWL[2] is another standard language for knowledge representation, widely used for capturing the "same-as" semantics of instances. Using `owl: sameas`, data providers can make explicit that two distinct URIs actually refer to the same real world entity. The task of establishing these same-as links is known under various names such as entity resolution and *instance matching*.

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/owl-features/

Semantic-driven approaches [43–45] use specific OWL semantics, such as explicit `owl:sameas` statements, to allow the same-as relations to be inferred via logical reasoning. Complementary to this, *data-driven approaches* derive same-as relations mainly based on attribute values of instances [1]. While they vary with respect to the selection and weighting of features, existing data-driven approaches are built upon the same paradigm of *direct matching*, namely, two instances are considered the same when they have many attribute values in common [30]. Hence, they produce only high quality results when there is sufficient overlap between instance representations. Overlap may, however, be small in heterogeneous datasets; especially, because the same instance represented in two distinct datasets may not use the same schema.

For example, in Table 3.1, the source instance `nyt:5962` and the target instances `db:Belmont_France` and `db:Belmont_California` share the same `rdfs:label` value, i.e., the string 'Belmont' (see Fig. 3.1). However, `rdfs:label` is the only attribute whose values overlap across both datasets, as the source and target graphs use rather distinct schemas. This overlap alone is not sufficient to determine whether `nyt:5962` is the same as `db:Belmont_France` (or `db:Belmont_California`). In this scenario of *instance matching across heterogeneous datasets*, direct matching alone cannot be expected to deliver high quality results.

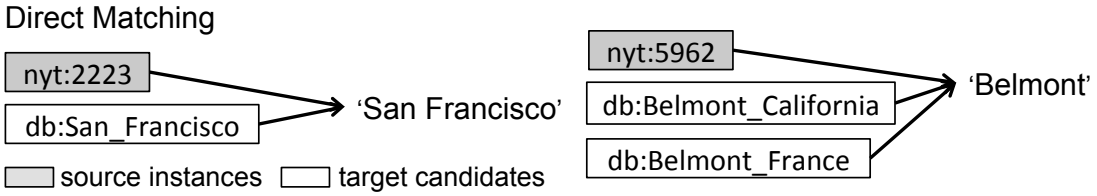

FIGURE 3.1: Examples of instances that share a common attribute value.

Contributions. We provide a (1) *detailed analysis* of many datasets and matching tasks investigated in the OAEI 2010 and 2011 [26, 46] instance matching benchmarks. We show that tasks greatly vary in their complexity. There are difficult tasks with a small overlap between datasets that cannot be effectively solved using state-of-the-art direct matching approaches. Aiming at these tasks, we propose to use direct matching in combination with (2) *class-based matching (CBM)*.

In this chapter, we employ the following class notion: a class is set of instances where each instance in this set must share at least one feature (vide Definition 3.3) in common with any other instance in this set.

Based on this notion, CBM works as follows: given a class of instances from the source dataset (e.g., `nyt:2223` and `nyt:5962`), called the *class of interest*, and a set of candidate matches retrieved from the target via direct matching (e.g., `db:San_Francisco`,

`db:Belmont_France` and `db:Belmont_California`), CBM aims to refine the set of candidates by filtering out those that do not match the class of interest. This matching however does not assume that the class semantics are explicitly given so that a direct matching at the class level is possible between the source (e.g. Nations) and target (e.g. Countries). Instead, CBM is based on this idea: given the instances are known to form a class (they have some features in common), their matches should also form a class in the target dataset (matches should also have some features in common). Thus, correct matches can be found by computing the subset of candidates in which members have the most features in common. Because these candidates correspond to source instances (as computed by the direct matching method), the class they form correspond to the source instance, i.e. the instances found by CBM belong to a class, which matches the class of interest. Note that in this process, the source and target instances are compared only during the candidate selection step. During class-based matching, *only data from the target dataset* is needed. This is the main difference to direct matching, which compares the source and the target data.

In the example depicted in Fig. 3.1, class-based matching would select `db:Belmont_California` and `db:San_Francisco` as correct matches, because this subset of instances are the most similar among the candidates: they have the predicate `db:country` and value `db:Usa` in common, as depicted in Fig. 3.2.

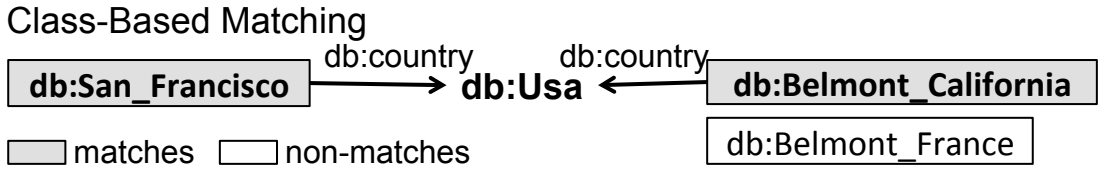

FIGURE 3.2: Class-based matching.

We (3) evaluated this approach, called SERIMI, using data from OAEI 2010 and 2011, two reference benchmarks in the field. These *extensive experiments* show that SERIMI yields superior results. Class-based matching achieved competitive results when compared to direct matching; most importantly, the improvements are complementary, achieving good performance when direct matching's performance was bad. Thus, using only a simple combination of the two, our approach greatly improves the results of existing systems. Considering all tasks in OAEI 2010, it increases average F1 result of the second best by 0.21 (from 0.76 to 0.97). For 2011 data, SERIMI also greatly improves the results of recently proposed approaches (*PARIS* [37] and *SIFI-Hill* [47]). Compared to the best system participating in OAEI 2011, SERIMI achieved the same performance. However, while that system leverages domain knowledge and assumes manually engineered mappings, our approach is generic, completely automatic and does not use training data.

Outline. This chapter is organized as follows: In Section 3.2, we introduce some definitions. In Section 3.3, we provide an overview of SERIMI. In Section 3.4, we discuss class-based matching and in Section 3.5 we propose a solution. In Section 3.6, we present a detailed analysis of matching tasks. Also, we discuss experiments and results. In Section 3.7, we discuss related works. Finally, we conclude in Section 3.8.

3.2 Preliminary Definitions

In this section, we present some important definitions.

Data. We use an RDF-based graph-structured model to accommodate different kinds of structured data.

Definition 3.1 (Data Graph)**.** The data is conceived as a set of graphs $\mathbf{G}$. Let U denote the set of Uniform Resource Identifiers (URIs) and L the set of literals, every $G \in \mathbf{G}$ is a set of triples of the form $\langle s, p, o \rangle$, where $s \in U$ (called subject), $p \in U$ (predicate) and $o \in U \cup L$ (object).

Every (set of) instance is represented as a set of triples.

Definition 3.2 (Instance Representation)**.** It is defined as: $IR(G, S) = \{\langle s, p, o \rangle \,|\langle s, p, o \rangle \in G, s \in S\}$, where G is a graph and S a set of instances in G. It yields a set of triples in which $s \in S$ appears as the subject. We denote the set of objects associated with an instance s over the predicate p in G as $O(s, p, G)$, with $O(s, p, G) = \{o|\langle s, p, o \rangle \in G\}$.

The representation of a single instance s is $IR(G, \{s\})$.

Features. Now, we define the features of a set of instances X.

Definition 3.3 (Features)**.** Let G be a dataset and X be a set of instances in G. The features of X are:

- $A(X) = \{p|(s, p, o) \in IR(G, X) \wedge s \in X\}$,
- $D(X) = \{o|(s, p, o) \in IR(G, X) \wedge s \in X \wedge o \in L\}$,
- $O(X) = \{o|(s, p, o) \in IR(G, X) \wedge s \in X \wedge o \in U\}$,
- $T(X) = \{(p, o)|(s, p, o) \in IR(G, X) \wedge s \in X\}$,
- $F(X) = A(X) \cup D(X) \cup O(X) \cup T(X)$.

Note $A(X)$ is the set of predicates, $D(X)$ the set of literals, $O(X)$ the set of URIs, and $T(X)$ is the set of predicate-object pairs in the representation of X.

Considering X ={`db:Belmont_California`}, its features are: $A(x)$ ={`rdfs:label`, `db:country`}, $D(x)$ = { 'Belmont'}, $O(x)$ ={`db:Usa`}, and $T(x)$ ={(`rdfs: label`, 'Belmont'), (`db:country`, `db:Usa`)}. Hence, $F(X)$ ={ `rdfs:label`, `db:country`, 'Belmont', `db:Usa` (`rdfs: label`, 'Belmont'), (`db:country`, `db:Usa`)}.

Note that $A(x)$ captures the predicates, which are schema-level features instances of a class typically have in common. However, we do not only use $A(x)$ but the whole union set $F(X)$, which comprises both schema- and data-level features. This is due to our special notion of class and the way we compute it: instances belong to a class when they share some features - no matter schema or data-level features. In this way, both types of features are leveraged for inferring the class instances belong to.

Class. We define a class as follows:

Definition 3.4 (Class)**.** Let G be a dataset and X a set of instances in G, X is a class if $\forall x \in X : F(\{x\}) \cap F(X - \{x\}) \neq \emptyset$.

Intuitively, a class is set of instances, where an instance in this set has at least one feature in common with at least one other instance in this set.

3.3 Overview of the Approach

In this section, we present an overview of SERIMI, our solution for instance matching.

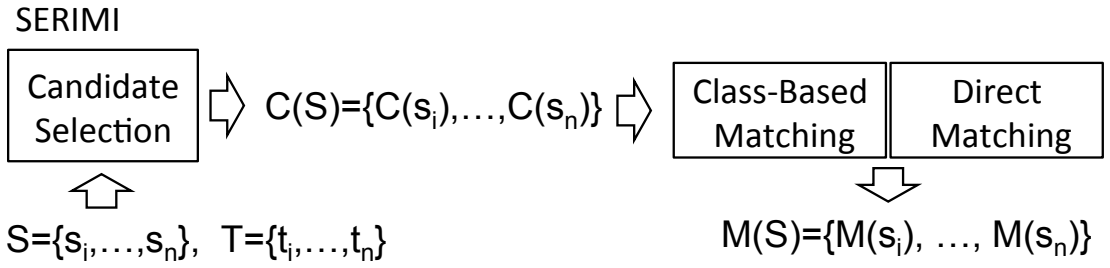

FIGURE 3.3: The instance matching in SERIMI.

The process of instance matching performed by SERIMI is illustrated in Fig. 3.3. SERIMI focuses on the problem of *instance matching across heterogeneous datasets*. In particular, the inputs are conceived to be partitioned into two datasets, the source S and target T. For every instance in $s \in S$, the goal is to find matching instances $t \in T$,

i.e. s and t refer to the same real-world object. This matching is performed in two main steps, candidate selection and match refinement.

Candidate Selection. For each $s \in S$, we firstly perform a low cost candidate selection step to obtain a candidate set $C(s) \subset T$. The set of all candidate sets is denoted as $C(S) = \{C(s)|s \in S\}$, and the union of all candidate instances is denoted as $C = \{t|s \in S : t \in C(s)\}$. This step reduces the number of comparisons needed to find matches between the source and target, i.e., from a maximum of $|S| \times |T|$ comparisons to $|S| \times |C|$.

Existing, so called, blocking techniques [48–50] can be used to quickly select candidates. Typically, a predicate (a combination of predicates) that is useful in distinguishing instances is chosen, and its values are used as blocking keys. In this setting of cross-dataset matching, a predicate in the source is chosen (e.g. `rdfs:label`) and its values (e.g. 'San Francisco') are used to find target candidate instances that have similar values in their predicates. Using the current example, the candidates matches for S ={`nyt:2223`, `nyt:5962`, `nyt:5555`} would be C(`nyt:2223`) = {`db:San_Francisco`} , C(`nyt:5962`) = {`db:Belmont_California`, `db:Belmont_ France`} and C(`nyt:5555`) = {`db:San_Jose_California`, `db:San_Jose_Costa_Rica`}, these candidates were selected based on high (lexical) similarity with the value of the `rdfs:label` predicate of the source instances.

To generate candidates in this work, we use simple boolean matching: we construct boolean queries using tokens extracted from candidate labels. Standard pre-processing is applied to lowercase the tokens and to remove stop words. These queries retrieve candidates, which have values that share at least one token with the values of the corresponding source instance. This method is primarily geared towards quickly finding all matches, i.e. high recall, but may produce many incorrect candidates. Higher precision can be achieved using other techniques known in literature [51].

Direct Matching. After the candidates have been determined, a more refined matching step is performed to find correct matches, $M(s) \subseteq C(s)$. For this, it is applied state-of-the-art approaches that perform more complex *direct matching*. Usually, instead of a simple blocking key, they use a combination of weighted similarity functions defined over several predicate values [37, 47]. Precisely, in direct matching, two given instances s and t are considered as a match when their similarity, $sim(s,t)$, exceeds a threshold δ. Typically, $sim(s,t)$ is captured by an *instance matching scheme*, which is a weighted combination of similarity functions (Edit Distance, Jaccard, ect.) defined over the predicate values of s and t [37, 47]:

$$sim(s,t) = \sum_{p \in P} w_p \cdot sim(O(s,p,S), O(t,p,T)) > \delta \tag{3.1}$$

Limitations. The above scheme assumes that s and t share predicates p based on which they can be directly compared (e.g. `rdfs:label`, `db:incountry`). In the heterogeneous setting, S and T may exhibit differences in their schemas. Instead of assuming p, more generally, we can define the instance matching problem in this setting based on the notion of comparable predicates $\langle p_s, p_t \rangle$. The predicate p_s is a predicate in S, whose values can be compared with those of p_t, a predicate in T.

For example, the instance `nyt:4232` does not share any predicate with the target instances but we can assume that the predicate `nyt:prefLabel` (p_s) is comparable to the predicate `rdfs:label` (p_t) because they have a similar range of values. Solutions, which specifically target this setting of cross-datasets matching, employ automatic schema matching or manually find the pairs of comparable predicates [35, 37, 52]. Let P_{st} be the set of all comparable predicates. We define the instance matching scheme for this setting as follows:

$$sim(s,t) = \sum_{\langle p_s, p_t \rangle \in P_{st}} w_{\langle p_s, p_t \rangle} sim(O(s, p_s, S), O(t, p_t, T)) > \delta \tag{3.2}$$

Since the direct overlap at the level of predicates (or values) between instances may be too small to perform matching in the heterogeneous setting, we propose class-based matching.

SERIMI. Class-based matching can be applied in combination with direct-matching, on top of the candidate selection step; as illustrated in Fig. 3.3. Candidate selection yields a set of candidates $C(S)$, which is refined by a module that combines class-based and direct matching to obtain $M(S) = \{M(s) | s \in S : M(s) \subseteq C(s) \in C(S)\}$.

While this work focuses on class-based matching, we are also proposing a complete instance matching pipeline called SERIMI. Existing state-of-the-art solutions are adopted for the candidate selection and direct matching components of SERIMI. Candidate sets $C(s) \in C(S)$ are determined for each instance $s \in S$ using a predicate value of s as key. The predicate is selected automatically based on the notion of coverage and discriminative power of predicates, also employed by [52]. Then, for direct matching, we use simple schema matching to compute comparable predicates P_{st}. The matching between a source instance s and a target instance t is then performed using values of predicates in P_{st}. As $sim(s,t)$, we use Jaccard similarity. The main difference to existing works lies in the selection of the threshold: for this, we use the same method that we propose for class-based matching.

We observe in the experiments that this simple combination of direct- and class-based matching produces good results. In SERIMI, direct- and class-based matching components are treated as black boxes that yield two scores considered independent. SERIMI multiplies and normalizes these scores to obtain a value in [0,1].

3.4 Class-Based Matching

Let S be the instances from the source dataset and M^* be the ground truth, containing all and only correct matches in the target dataset. The candidate instances C computed via direct matching might be not sound and not complete, i.e. there is a candidate in C that is not in M^* and there is a an element in M^* that is not in C, when some $s \in S$ and corresponding elements $t \in C$ only have few features that directly match. Class-based matching aims to find those non-sound matches in C (to improve soundness / precision), using only features of the candidate instances $t \in C$.

Particularly, CBM is built upon the observation that matching is usually performed for a class of source instances. That is, all $s \in S$ belong to a specific class[3]. Our idea is that if S is a class, i.e., its instances share some features, then correct matches for $s \in S$ should also belong to a class, i.e., instances in M^* should also share some common features. Then, we aim to compute M^* by finding a subset $M \subseteq C$, whose instances are most similar to each other (compared to other candidate subsets). These instances are considered *class-based matches* because they form a class that matches the class of interest.

3.4.1 Formal Definition

For the sake of presentation, we formalize the basic version of our problem first: let assume that individual datasets do not contain duplicates such that for each source instance, the goal is to find exactly one match in the target dataset, i.e. $|M| = |S|$ with $|M(s)| = 1$, for all $s \in S$. Then, the CBM problem can be formulated as follows:

Definition 3.5 (Class-based Matching (CBM))**.** The solution for the class-based matching problem can be computed as

$$\begin{aligned} M^* \approx & \underset{M \in \mathbf{M}}{\operatorname{argmax}} \frac{\sum_{t \in M} Sim(t, M)}{|M|} \\ & \text{Subject to:} \\ & \forall s \in S : |C(s) \cap M| = |M(s)| = 1 \end{aligned} \tag{3.3}$$

[3]Notice that when the input S captures different classes, it can be partitioned into sets of instances representing specific classes [53].

where $\mathbf{M}$ is the set containing all possible candidate subsets M as elements, $Sim(t, M)$ is a function that returns the similarity between an instance t and the subset of candidates M.

As an approximation for $M^* \in \mathbf{M}$, we compute a subset of candidate M containing instances that are similar to itself, i.e. the goal is to maximize $Sim(t, M)$ for all $t \in M$. Compared to all other possible candidate subsets, the solution is the one that is most similar to its instances. Further, in this basic setting, it contains exactly one candidate for every source instance.

As an example, we have as candidate subsets $M_1 = \{$`db:Belmont_California`, `db:San_Francisco` and `db:San_Jose_California`$\}$ and $M_2 = \{$`db:Belmont_France`, `db:San_Francisco` and `db:San_Jose_California`$\}$ for the data in our scenario. Instances in M_1 are more similar to M_1 than instances in M_2 are similar to M_2. In other words, the similarity among instances in M_1 is higher than the similarity among instances in M_2: the candidate `db:Belmont_California` shares the predicate `db:country` and value `db:Usa` with the instance `db:San_Francisco`, which in turn, shares the predicate `db:locatedIn` and value `db:California` with `db:San_Jose_California`. Thus, CBM considers M_1 as a better approximation of M^* than M_2.

We note that typically, instance matching approaches do not provide a theoretically sound and complete solution. As captured above, CBM is also only an approximate solution in that sense. The quality of this approximation taken by our approach is studied in experiments using real-world matching tasks and datasets.

Computational Complexity. The following theorem captures the complexity of this problem:

Theorem 3.6. *CBM is an instance of the maximum edge-weighted clique problem (MEWCP) [54], therefore CBM is NP-hard.*

Proof. Each candidate $t \in C$ can be mapped to a vertex in an undirected graph G. Two vertices $x, y \in C$ are connected if and only if $x \in C(s_i)$ and $y \in C(s_j)$, where $s_i \neq s_j$. The weight of an edge $\{x, y\}$ is given by $sim(x, y)$. Any clique in G contains exactly one candidate for each $C(s) \in C(S)$. Then, a solution to the CBM problem is a clique in G with maximum weight. □

CBM Variations. Apart from the introduced basic setting, two other variants exist: *1-to-many class-based matching (1-to-many CBM)* and *unrestricted class-based matching (UCBM)*. The former assumes $\forall s \in S : |M(s)| > 0$, while the latter, assumes $\forall s \in S : |M(s)| \geq 0$. 1-to-many CBM considers the cases where there is at least one match for

each source instance, while UCBM considers the cases where some candidate set $C(s)$ may not contain a match to $s \in S$. To capture the UCBM problem, the constrain should be removed and the term

$$Z = \frac{\frac{\sum_{s \in S} |C(s) \cap M|}{|C(s)|}}{|S|} \tag{3.4}$$

should be added to Eq. 3.3. Z is simply an auxiliary term introduced to deal with the general case where $|M(s)| = |C(s) \cap M|$ might be zero. It helps to assign a solution set $M \in \mathbf{M}$ a higher score, when the majority of its matches $M(s)$ has cardinality higher than zero; hence, it avoids solution sets with many empty matches.

In the next section, we propose an approach to solve CBM and its variants, 1-to-many CBM and UCBM.

3.5 Class-based Matching: A Solution

We will first present the main idea and then discuss extensions to this basic solution.

3.5.1 Basic Solution

Here we present our implementation of the presented CBM approach.

Class-based Matching. Given a set of instances S and the candidate sets $C(S) = \{C(s_1), \ldots, C(s_n)\}$, we implement class-based matching by finding the instances t from each candidate set (i.e. $t \in C(s) \in C(S)$) that are similar to the candidate sets $C(S)$.

Our method starts computing a score of similarity between $t \in C(s)$ and $C(S)$ itself, i.e., $Sim(\{t\}, C(S))$. In this process $C(S)$ is considered the class of interest but not the solution set M; differently from the formal problem definition where M is both the class of interest and a solution set. In this approach, we depart from $C(S)$ to obtain the solution set M and $M(S)$.

This solution exploits the intuition that given t and any candidate set $C(s) \in C(S)$, if $F(\{t\})$ does not share any feature with $F(C(s))$, then t is not similar to any instance in this candidate set. If t is not similar to any candidate set $C(s) \in C(S)$, it cannot form a class with any candidate instance; therefore, based on the class-based matching assumption, it cannot be a correct match for s. Contrarily, a candidate t that is more similar to other candidate sets are more likely to form a class with other candidates, and therefore, can be a correct match. This heuristic is implemented as follows.

The computation of $Sim(\{t\}, C(S))$ obtains a score for each individual instance $t \in C$. Then, the solution set M is composed of $t \in M(s) \subseteq C(s)$, where for all $t \in M(s)$, $Sim(\{t\}, C(S)) > \delta$. Below, we define Sim and further we describe how we compute the threshold δ.

$$Sim(t, C(S)) = \sum_{C(s') \in C(S)^-} \frac{SetSim(\{t\}, C(s'))}{|C(s')|} \tag{3.5}$$

where $t \in C(s)$ and $C(S)^- = C(S) \setminus C(s)$.

First, note in Eq. 3.5, $t \in C(s)$ is not compared with $C(s)$ but the other candidate sets $C(s')$. $C(s)$ in our implementation is computed via direct matching and thus contains candidates very similar to t. Just like the other candidate sets $C(s')$, these candidates also help to capture the class of interest. However, due to their relative high similarity to t, they have a too strong impact, compared to $C(s')$. Excluding it from the class similarity computation helps to avoid this strong bias towards $C(s)$. Secondly, note the individual score $SetSim(\{t\}, C(s'))$ is weighted by the cardinality of $C(s')$ such that a $C(s')$ with high cardinality has a smaller impact on the aggregated similarity measure. We do this to leverage the observation that small sets contain few but more representative instances. They are better representations of the class of interest.

We further normalize the result of Eq. 3.5 by the maximum score among all instances in $C(s)$ as

$$Sim(t, C(s), C(S)) = \frac{Sim(t, C(S))}{MaxScore(C(s), C(S))} \tag{3.6}$$

where

$$MaxScore(C(s), C(S)) = MAX\{Sim(t', C(S)) | t' \in C\} \tag{3.7}$$

This yields a class-based similarity score that is in $[0, 1]$. This algorithm takes $O(|C(S)| \times |C|)$ (note $|S| = |C(S)|$), in the worse case. Using this function, an instance t is considered as a correct match for s, if $Sim(t, C(s), C(S))$ is higher than a threshold δ or when it is the top-1 result. We will refer to these two variants as the Threshold (for 1-to-many CBM and UCBM) and the Top-1 approach (for CBM), respectively.

The Top-1 approach makes sense for those cases where datasets are duplicate-free or one-to-one mapping between a source and a target instance can be guaranteed. In this case, as every instance in every dataset stands for a distinct real-world entity, there exist at most only one correct match in the target for every instance in the source (i.e. likely

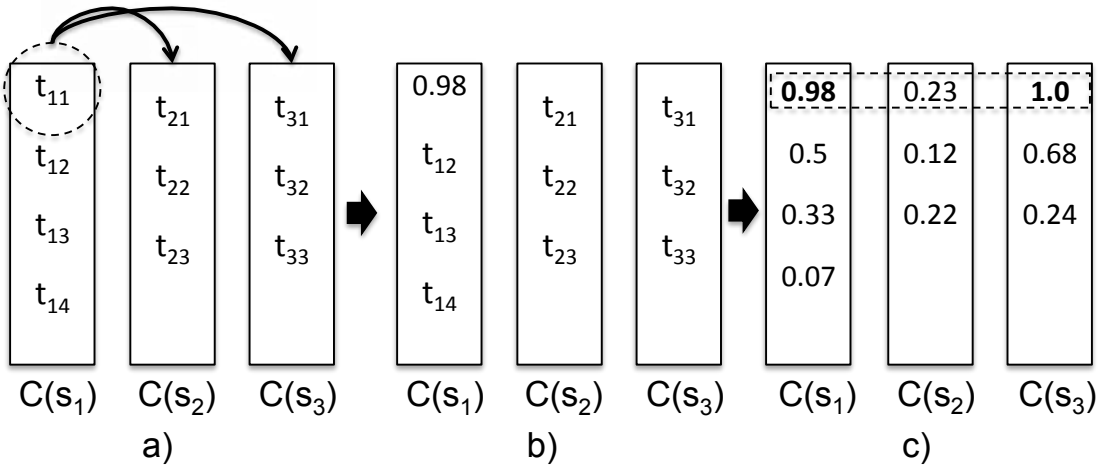

FIGURE 3.4: (a) Class-based similarity score for the candidate t_{11} is obtained by comparing it with $C(s_2)$ and $C(s_3)$, (b) the score for t_{11} and (c) the scores for all other candidates.

the top-1). In the other cases where there are one-to-many matches, the Threshold approach is used. Notice that the Threshold is a general approach. As we will show empirically, it yields competitive accuracy to the Top-1 approach.

Class-based matching is illustrated in Fig. 3.4 for the instance t_{11}, where it is compared to the candidate sets $C(s_2)$ and $C(s_3)$, where $C(S)^- = \{C(s_2), C(s_3)\}$. Notice that, in the end, $Sim(t_{11}, C(S))$ compares the features of $F(\{t_{11}\})$ to $F(C(s_2))$ and to $F(C(s_3))$. This is done for all instances in $C(s_1)$ and the one with the highest score Sim is assumed to be the correct match for s_1. Notice that for $C(s_2)$, $C(S)^-$ is defined as $C(S)^- = \{C(s_1), C(s_3)\}$. Alg. 1 illustrates the computation of Sim.

Similarity Function. Now, we introduce $SetSim(X_1, X_2)$ to compute the similarity between two sets of instances X_1 and X_2 based on their sets of features $F(X_1)$ and $F(X_2)$:

$$SetSim(X_1, X_2) = FSSim(F(X_1), F(X_2)) \tag{3.8}$$

where $FSSim(F(X_1), F(X_2))$ is a function capturing the similarity between $F(X_1)$ and $F(X_2)$.

Early work such as Tversky's [55] shows that the similarity of a pair of items depends both on their commonalities and differences. This intuition is exploited by similarity functions used for instance matching, which like Jaccard similarity, gives the *same weight* to commonalities and differences.

We depart from the equal-weight strategy to give a *greater emphasis on commonalities*. This is because the goal of class-based matching is to find whether some instances match

Algorithm 1 SimScores($C(S)$).

```
1: scores ← ∅
2: for c(s) ∈ C(S) do
3:    C(S)⁻ ← C(S) \ C(s)
4:    score_c(s) ← ∅
5:    for t ∈ C(s) do
6:       score_t ← 0
7:       for c(s)′ ∈ C(S)⁻ do
8:          score_t ← score_t + SetSim({t},C(s)′) / |c(s)′|
9:       end for
10:      score_c(s) ← score_c(s) ∪ score_t
11:   end for
12:   scores ← scores ∪ score_c(s)
13: end for
14: maxscore ← max(scores)
15: for score_c(s) ∈ scores do
16:   for i in 1..|score_c(s)| do
17:      score_c(s)[i] ← score_c(s)[i] / maxscore
18:   end for
19: end for
20: return scores
```

a class, which by our definition, is the case when they share many features with that class. For deciding whether an instance belongs to a class or not, the common features are thus, by definition, more crucial. Not only that, the special treatment of common features also makes sense when considering that common features are more scarce. That is, the number of features shared by all instances in a class is typically much smaller than features that are not.

We propose the following function to support this intuition:

$$FSSim(f_1, f_2) = \begin{cases} 0 \text{ if } |f_1 \cap f_2| = 0 \\ |f_1 \cap f_2| - (\frac{|f_1 - f_2| + |f_2 - f_1|}{2|f_1 \cup f_2|}) \quad \text{otherwise} \end{cases} \tag{3.9}$$

where f_1 and f_2 stand for $F(X_1)$ and $F(X_2)$, respectively. $FSSim(f_1, f_2)$ only considers f_1 and f_2 to be similar when there exist some commonalities (i.e. $FSSim(f_1, f_2)$=0 if $|f_1 \cap f_2| = 0$). The first term $|f_1 \cap f_2|$ has a much larger influence, capturing commonalities as the number of overlaps between f_1 and f_2, which is always larger than 1. The second term $(\frac{|f_1 - f_2| + |f_2 - f_1|}{2|f_1 \cup f_2|})$, capturing the differences, is always smaller than 1. In fact, given f_j and f_k that have n and $n-1$ features in common with f_i, respectively, $FSSim$ always returns a higher score for f_j.

For example, assuming $f_1 = F(\{\texttt{db:Belmont\_California}\})$, $f_2 = F(\{\texttt{db:Belmont\_France}\})$ and $f_3 = F(C(\texttt{nyt:5555}))$; then, $FSSim(f_1, f_3) = 3.65$, while $FSSim(f_2, f_3) = 1.5$.

The scores reflect the fact that f_1 has 4 features in common with f_3, while f_2 has only 2.

Notice that $FSSim$ does not capture any class semantics but is a set similarity function tailored towards the commonalities, for supporting the intuition discussed before. However, the class semantics is inferred as a result of applying this similarity computation as performed in our approach: the instances found by CBM form a class that corresponds to the class of interest.

The bias towards commonalities is captured by the following theorem, which does not hold for the Jaccard function (see Appendix A):

Theorem 3.7. *If* $|f_i \cap f_j| > |f_i \cap f_k|$ *then* $FSSim(f_i, f_j) > FSSim(f_i, f_k)$.

3.5.2 Reducing the Number of Comparisons

In order to compute a score for every instance in each candidate set $C(s) \in C(S)$, our class-based matching approach requires a maximum of $|C(S)| \times |C|$ comparisons. Since $|C(S)|$ can be large, we propose to reduce the number of comparisons by reducing $C(S)$ to a minimal subset $C(S)^*$ such that the feature distribution of $C(S)^*$ differs only within an error margin ϵ from the feature distribution of $C(S)$. Then, $C(S)^*$ is used in the line 3 of Alg. 1 instead of $C(S)$, i.e., $C(S)^- = C(S)^* \setminus C(s)$. We define the feature set and the distribution over elements in that set as follows:

Definition 3.8 (Feature Set)**.** The feature set of $C(S)$ is $F(C(S)) = \bigcup_{C(s) \in C(S)} F(C(s))$.

Definition 3.9 (Feature Distribution)**.** A distribution over the feature set $X = F(C(S))$, denoted by $Pr(X)$, assigns a probability $p(x)$ to every feature x, i.e. the probability of observing a feature x through the repeated sampling of features from X:

$$p(x) = Pr\{X = x\} = \frac{\sum_{C(s) \in C(S)} |\{x\} \cap F(C(s))|}{|F(C(S))| \times |C(S)|}$$

where

1. $p(x) \geq 0$ for all $x \in X$ and
2. $\sum_{x \in X} p(x) = 1$.

In the ideal case, $C(S)^*$ contains a much smaller amount of candidate sets compare to $C(S)$, i.e. $|C(S)^*| << |C(S)|$, while carrying the same amount of information such that the similarity scores computed for $C(S)^*$ and $C(S)$ are the same. In order to capture the

differences in the provided information content, we use the *z-test*, which is a standard method for analyzing the similarity/difference between the distribution of a sample and the distribution of the original population:

$$z\text{-}test = \frac{(\mu(sample) - \mu(population))}{(\frac{\sigma(population)}{\sqrt{size(sample)}})}$$

where $\mu(\cdot)$, $\sigma(\cdot)$ and $size(\cdot)$ denote the mean, the standard deviation and the size, and $population = Pr(F(C(S)))$ and $sample = Pr(F(C(S)^*))$.

A brute force algorithm to solve this problem is to enumerate all possible subsets of $C(S)$, i.e., its power set $2^{|C(S)|}$. Then, for each set in $2^{|C(S)|}$, it picks the minimal set $C(S)^*$ that has a distribution equivalent to the one of $C(S)$. In the worse case, this algorithm takes $O(2^{|C(S)|})$ verifications to find $C(S)^*$, which is prohibitive even for small $C(S)$.

We note the attempt to find an optimal solution to this problem may go against our goal. We need to find the set $C(S)^* \subseteq C(S)$ at very low cost so that the time spent is smaller than the gain that can be achieved by using $C(S)^*$ instead of $C(S)$. We thus use an efficient greedy algorithm that exploits the following intuition: a sample is more similar to its population when it contains more data from the population. Without enumerating and evaluating each subset, it iteratively extracts and adds a subset $C(s) \in C(S)$ to the sample $C(S)^*$ until the *z-test* between $Pr(F(C(S)))$ and $Pr(F(C(S)^*)))$ approaches the confidence value commonly used in the literature,[4] or all $C(s) \in C(S)$ is added to $C(S)^*$. For faster convergence, only features that occur more than once in the data are considered in $F(C(S))$.

The procedure to obtain $C(S)^*$ is summarized in Alg. 2. It takes $O(|C(S)|)$, in the worse case. In Sec. 3.6, we compare the time performance and accuracy of Alg. 1 with and without this procedure.

3.5.3 Selecting the Threshold

As discussed, the Top-1 approach can be used when the datasets are duplicate-free. In all other cases, a threshold selection method should be employed. Then, only instances with similarity score above the computed threshold δ are selected as matches. State-of-the-art methods [47, 56] are supervised, relying on training data to find the best threshold. We propose an unsupervised method, which only uses statistics that can be derived from the computed scores. We cast the problem of threshold selection as the one of finding the statistical outliers among the similarity scores. In particular, we use

[4] which, under our assumption of normal distribution, is in $[-1.96, 1.96]$

Algorithm 2 CandidateSetsReduction($C(S)$).

1: $C(S)^* \leftarrow \emptyset$
2: $\mu \leftarrow mean(p(C(S)))$
3: $\sigma \leftarrow stdv(p(C(S)))$
4: **for all** $C(s) \in C(S)$ **do**
5: $\quad C(S)^* \leftarrow C(S)^* \cup C(s)$
6: $\quad n \leftarrow |C(S)^*|$
7: $\quad M \leftarrow mean(p(C(S)^*))$
8: $\quad SE \leftarrow \frac{\sigma}{\sqrt{n}}$
9: $\quad z \leftarrow \frac{(M-\mu)}{SE}$
10: $\quad$ **if** z is in the confidence interval **then**
11: $\quad\quad$ **return** $C(S)^*$
12: $\quad$ **end if**
13: **end for**
14: **return** $C(S)^*$

two bags of scores, one containing only the maximum scores and the other containing all scores.

Definition 3.10 (Bag of Scores)**.** Given the candidates C and $C(S)$, the bag of *all scores* contains a score for every $t \in C$, i.e., $Scores_{all} = \{Sim(t, C(S)) \ | t \in C\}$. The bag of *maximum scores* contains a score for every $C(s) \in C(S)$, i.e., $Scores_{max} = \{MaxScore(C(s), C(S)) | C(s) \in C(S)\}$.

The maximum scores constitute the starting point for threshold selection. Intuitively speaking, two cases can be distinguished: First, (1) we have maximum scores that all are close to 1, and differences among them are small. (2) In the second case, there are large variations among scores. Some of them are low, approaching 0.

Note the first case corresponds to the setting where correct matches are easy to find, i.e., at least one candidate with score close to 1 could be found for every source instance. In this case, δ is simply defined based on the minimum score in $Score_{max}$. In this way, all candidates with score in $Score_{max}$ are selected. This strategy works for this "easy setting" because due to the use of set-based similarity in class-based matching, score differences among correct matches tend to be small while differences between correct and incorrect ones are much larger. Thus, incorrect matches typically have scores much lower than the minimum score in $Score_{max}$.

In the second "harder setting", "bad" candidates were detected, i.e., those with low scores in $Score_{max}$. This indicates that for some source instances, no correct candidates exist or could be found. However, we cannot use the minimum score as before to filter these "bad" candidates. It could be too low, or generally, not precise enough to separate correct from incorrect matches. To find δ in this case, we propose to detect outlier

scores. For finding outliers more precisely, we use the bag of all scores, $Score_{all}$, instead of $Score_{max}$. Intuitively, candidates that have an outlier score share fewer features with the class of interest, thus can be regarded as incorrect.

As a mechanism to implement the ideas above, we propose to use a method based on the Chauvenet's criterion [57], a statistical technique for outlier detection.

Definition 3.11 (Chauvenet's Criterion)**.** Given the mean μ and the standard deviation σ of the scores in $Score_{all}$, a score $x \in Score_{all}$ is an outlier if $Chauvenet(x) < c_1$, where

$$Chauvenet(x) = p(\frac{\mu - x}{\sigma}) \times |Score_{all}|,$$

c_1 is a confidence level[5] and $p(\frac{\mu-x}{\sigma})$ is the probability[6] of observing a data point x that is $\frac{\mu-x}{\sigma}$ times standard deviations away from the mean.

According to the Chauvenet's criterion, there are no outliers when $\sigma < c_2$, another confidence level that is typically set close to 0.[7]

Our procedure for threshold selection first extracts the maximum score of each candidate set $C(s) \in C(S)$ to form $Score_{max}$. When there are no outliers according to the Chauvenet's criterion, we set the threshold as the minimum score in $Score_{max}$. Otherwise, we iteratively apply the Chauvenet's criterion over $Score_{all}$ until no further outliers can be detected: in every iteration, if outliers are found and δ is the highest score among all outliers, we remove all scores that are smaller than δ from $Score_{all}$; this pruned bag of scores is then used in the next iteration. The maximum δ found in this process is used as the threshold. Alg. 3 describes this algorithm.

For example, for the scores in Fig. 3.4, the list of maximum scores $Score_{max}$={0.98, 0.23, 1.0} has a standard deviation much higher than the confidence level c_2; therefore, the algorithm is applicable. Considering all scores $Score_{all}$= {0.98, 0.5, 0.33, 0.07, 0.23, 0.12, 0.22, 1.0, 0.68, 0.24 }, this algorithm would select as threshold $\delta = 0.68$; therefore, all instances with scores smaller than 0.68 would be rejected as a correct match. Notice that 0.68 is much higher than 0.22, the minimal of $Score_{max}$.

3.6 Evaluation

Our experiments are based on the OAEI 2010 and 2011 instance-matching track. We observed that SERIMI with the proposed candidate set reduction algorithm was 20%

[5] Typically, it is set to 0.5 when using Chauvenet's criterion.
[6] We assume a normal distribution.
[7] In literature, $\sigma < 0.011$ is typically used.

Algorithm 3 ThresholdBasedSelection(C).

```
1: Y ← getMaxScores(C)
2: L ← getAllScores(C)
3: δ ← Array
4: if Y.standardDeviation < c2 then
5:    return Y.min
6: end if
7: for all x ∈ L do
8:    if L.mean − x < 0 then
9:       continue;
10:   end if
11:   if chauvenet(L, x) then
12:      C′ ← remove all scores ≤ x from C
13:      δ.add(x)
14:      δ.add(ThresholdBasedSelection(C′))
15:      return δ.max
16:   end if
17: end for
18: return 0
```

faster than SERIMI without it. Also, class-based matching was useful and complementary to direct matching. For OAEI 2010, this combination increased average F1 result of the second best by 0.21; and, for OAEI 2011 data, SERIMI improves the results of recently proposed approaches, *PARIS* [37] and *SIFI-Hill* [47], by 0.44 and 0.09, respectively. Compared to the best system participated at OAEI 2011, SERIMI achieved the same performance. However, as opposed to that, SERIMI does not assume domain knowledge and manually engineered mappings.

Evaluation Metrics. We used the standard F1 to measure the result accuracy (also employed by OAEI). $F1 = 2 \times \frac{Recall \times Precision}{Recall + Precision}$ is the harmonic mean between precision (proportion of correct matches among matches found) and recall (proportion of matches found among all actual matches). To compute F1, the provided reference mappings were used as the ground truth.

Data. We used all the OAEI 2010 data employed by participants, which include the life science (LS) collection containing DBPedia, Sider, Drugbank, Dailymed, Tcm and Diseasome and the Person-Restaurant (PR) dataset. From OAEI 2011, the datasets used were New York Times (Nyt), DBPedia, Geonames and Freebase. Given a pair of datasets, the task was to match instances in one dataset to instances in the other. The source class of instances for each dataset was defined by the OAEI. Detailed information can be found in their website [8]. Table 3.2 and 3.3 show some relevant statistics related to the datasets and matching tasks, respectively.

[8] http://oaei.ontologymatching.org

TABLE 3.2: Number of triples in each dataset.

Dataset	Triples	Dataset	Triples
Nyt	350.349	Person11	9.000
Freebase	3.554.824	Person12	7.270
DBPedia	>10.000.000	Person21	10.800
Geonames	>10.000.000	Person22	5.944
Sider	96.204	Rest1	1.130
Tcm	111.021	Rest2	7.520
Dailymed	131.068	Drugbank	507.500
Diseasome	69.545	-	-

TABLE 3.3: Dataset pairs representing matching tasks, number of comparable predicates (CP) for every task, number of correct matches (Match), number of candidate matches obtained from candidate selection (Cand), mean (MEAN) and standard deviation (STDV) of the number of candidates per instance.

Dataset Pairs	CP	Match	Cand	MEAN	STDV
Nyt-DB-Corp	3	1965	3839	2.0	2.01
Nyt-DB-Geo	4	1920	9246	4.87	7.9
Nyt-DB-Per	5	4977	7937	1.61	1.02
Nyt-Freebase-Corp	2	3044	3398	1.15	0.37
Nyt-Freebase-Geo	3	1920	2234	1.19	0.43
Nyt-Freebase-Per	3	4979	5090	1.04	0.19
Nyt-Geonames	4	1789	10782	6.18	9.21
Dailymed-Sider	8	1592	1592	1.0	0.03
Diseasome-Sider	4	163	163	1.0	0.08
Drugbank-Sider	8	284	284	1.0	0.06
Sider-Dailymed	2	1634	1915	2.93	2.43
Sider-DB-Drugs	2	734	742	1.05	0.22
Sider-DB-SideEffect	2	775	960	1.25	0.56
Sider-Diseasome	4	173	192	1.2	0.57
Sider-Drugbank	8	1140	1260	1.04	0.21
Sider-Tcm	2	171	171	1.0	0.08
Person11-Person12	6	500	1501	3.23	2.28
Person21-Person22	6	400	476	5.06	3.2
Rest1-Rest2	2	112	117	1.06	0.5

Systems. All computed results were done using an Intel Core 2 Duo, 2.4 GHz, 4 GB RAM, using a FUJITSU MHZ2250BH FFS G1 248 GB hard disk. The SERIMI implementation used in these experiments is available for download[9] at GitHub. It was implemented in Ruby. Except for SIFI and PARIS, we copied all available results as published in the OAEI benchmarks. We used the available authors implementation[10] for PARIS, and the best effort implementation in Java for SIFI-Hill (SIFI).

[9] https://github.com/samuraraujo/SERIMI-RDF-Interlinking
[10] http://webdam.inria.fr/paris/

3.6.1 Task Analysis

The suitability of direct matching and class-based matching for a task is related to the complexity of the matching task itself. So far, there is no method that suits all kinds of matching tasks, because data are imperfect in this heterogeneous setting. As we will show, the widely employed assumption that attributes between datasets largely overlaps is not true for all matching tasks, or for all instances within a matching task. We observed the accuracy of each matching technique largely depends on the distribution of the predicates and values in the source and target dataset. In order to obtain a better understanding of how these distributions affect the accuracy of a matching technique, below we propose the use of coverage (Cov) and discriminative power (Disc) as measures for analyzing the task complexity.

$$Cov(p, S, G) = \frac{|\{s|\langle s, p, o\rangle \in G \wedge s \in S\}|}{|S|} \tag{3.10}$$

$$Disc(p, S, G) = \frac{|\{o|\langle s, p, o\rangle \in G \wedge s \in S\}|}{|\{t|t = \langle s, p, o\rangle \in G \wedge s \in S\}|} \tag{3.11}$$

where S is the given set of instances in the dataset G.

The coverage of a predicate p measures the number of instances in S that p occurs. A predicate p with low coverage indicates that p occurs in a few instances; therefore, when utilizing values of p for finding matches, we may miss some candidates. The discriminative power measures the diversity of predicate values. A predicate p has low discriminative power when many instances have the same values for p; therefore, using values of p for matching, results in larger candidate sets. Consequently, datasets with many predicates that have low coverage and low discriminative power are harder to match.

Using these two measures, we introduce a task complexity measure TC that defines the complexity of matching a set of instances S with T, where $T = \bigcup_{c \in C(S)} c$. First, we introduce the *predicate complexity measure* (PCM) that measures the complexity of matching a set of instances X based on coverage and discriminative power of a set of predicates P in G.

$$PCM(P, X, G) = \frac{\sum_{a \in P} Cov(a, X, G) + Disc(a, X, G)}{2|P|} \tag{3.12}$$

The size of the candidates sets in $C(S)$ is also an indication of complexity because sets with more candidates may have more ambiguous candidates to filter out. Therefore, we

define $Card(S)$. Smaller values for $Card(S)$ indicate that $C(S)$ has bigger candidate sets.

$$Card(S) = \frac{|C(S)|}{\sum_{c \in C(S)} |c|} \tag{3.13}$$

Finally, we introduce TC, defined as:

$$TC = 1 - PCM(P_s, S, G_s) \times PCM(P_t, T, G_t) \times Card(S) \tag{3.14}$$

where TC is a value in the interval [0,1], where 0 is less complex and 1 more complex. Table 3.3 shows the characteristics of each matching task. Fig. 3.5 shows the tasks ordered by TC. With respect to that, Nyt-Geonames is the most complex task, which on average has around six candidate matches per instance. In this table, some tasks are easier tasks because most of the candidate sets contain only correct matches, or one instance per candidate set (e.g. Sider-Tcm).

Fig. 3.6 shows the coverage and discriminative power of predicates in the target datasets. In all these datasets, there exist at least one predicate with 100% coverage (e.g. `drugbank:brandName`, `freebase:name`). However, only in some cases, their discriminative power is maximal (e.g. `drugbank:brandName`). The DBPedia, Geonames and Freebase datasets seem to be the hardest to match, as both coverage and discriminative power of their predicates are the lowest. In these cases, many predicates have to be used, which is only possible when there are many corresponding predicates in the source. Contrarily, the higher the coverage, the easier is the task because more instances can be covered with fewer predicates (the discriminative power of source predicates is, however, irrelevant because only target predicate values are used for finding matches). Fig. 3.7 shows predicates in the source datasets that are comparable to target predicates, and their coverage. It indicates there are always some comparable predicates that can be used (Table 3.3 explicitly shows the number of comparable predicates for every task), and that their coverage is always maximal (except for Nyt). In summary, comparable predicates exist for all the given tasks. However, direct matching is harder for some tasks such as Nyt-Geonames and Nyt-DB-Geo as they require using several predicates due to low coverage and discriminative power of target predicates. As the coverage is different for different target instances in those tasks, direct matching may not achieve its full performance due to the lack of comparable predicates at instance level.

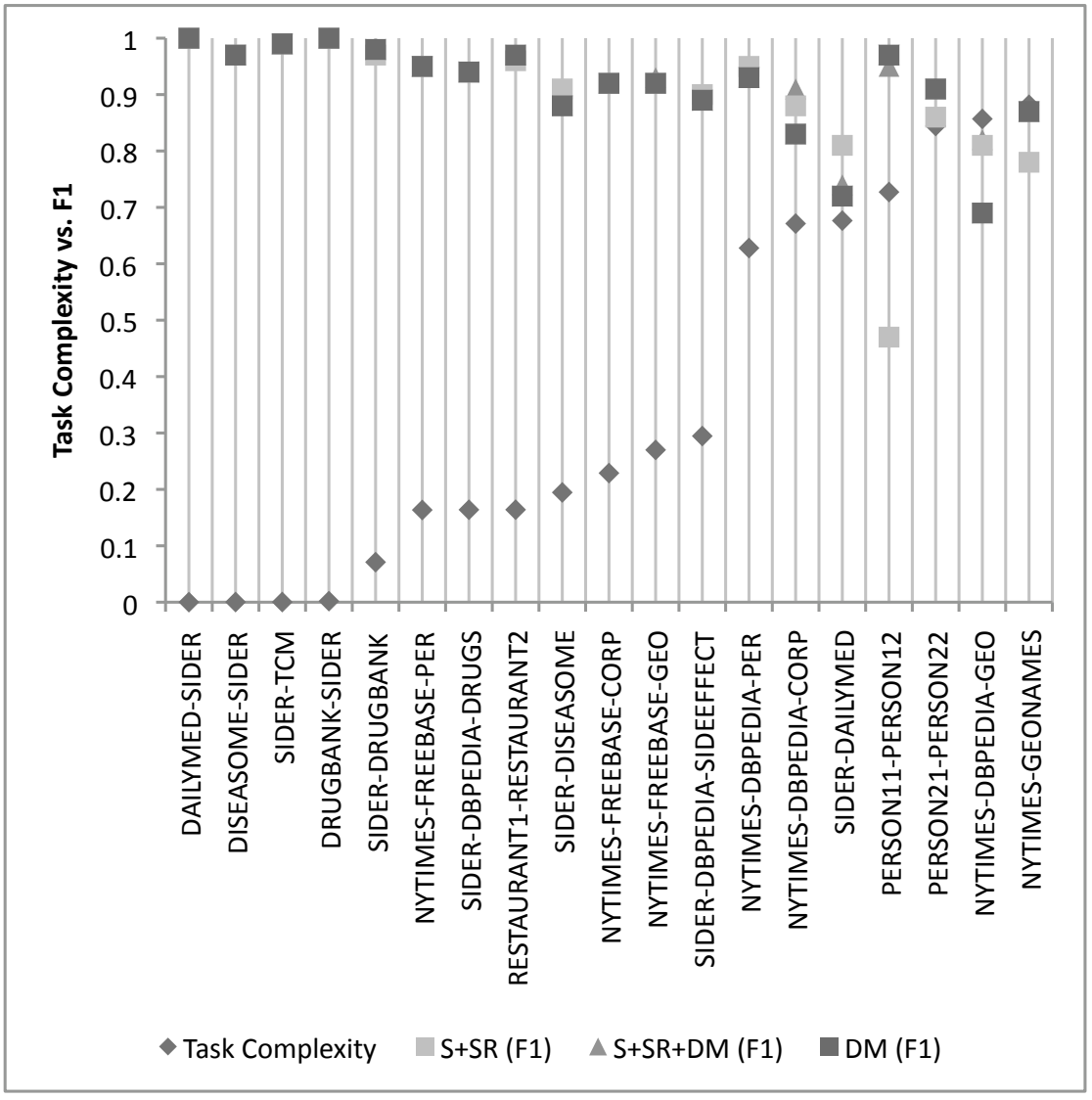

FIGURE 3.5: F1 for tasks with increasing complexity.

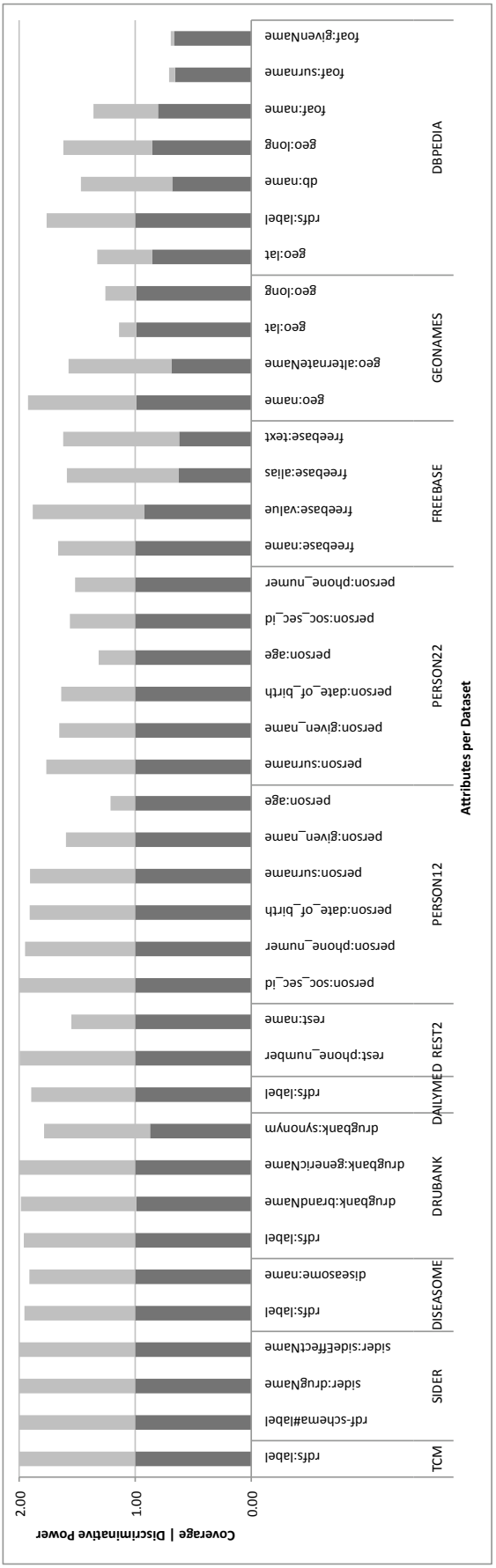

FIGURE 3.6: Coverage and discriminative power of predicates in the target datasets.

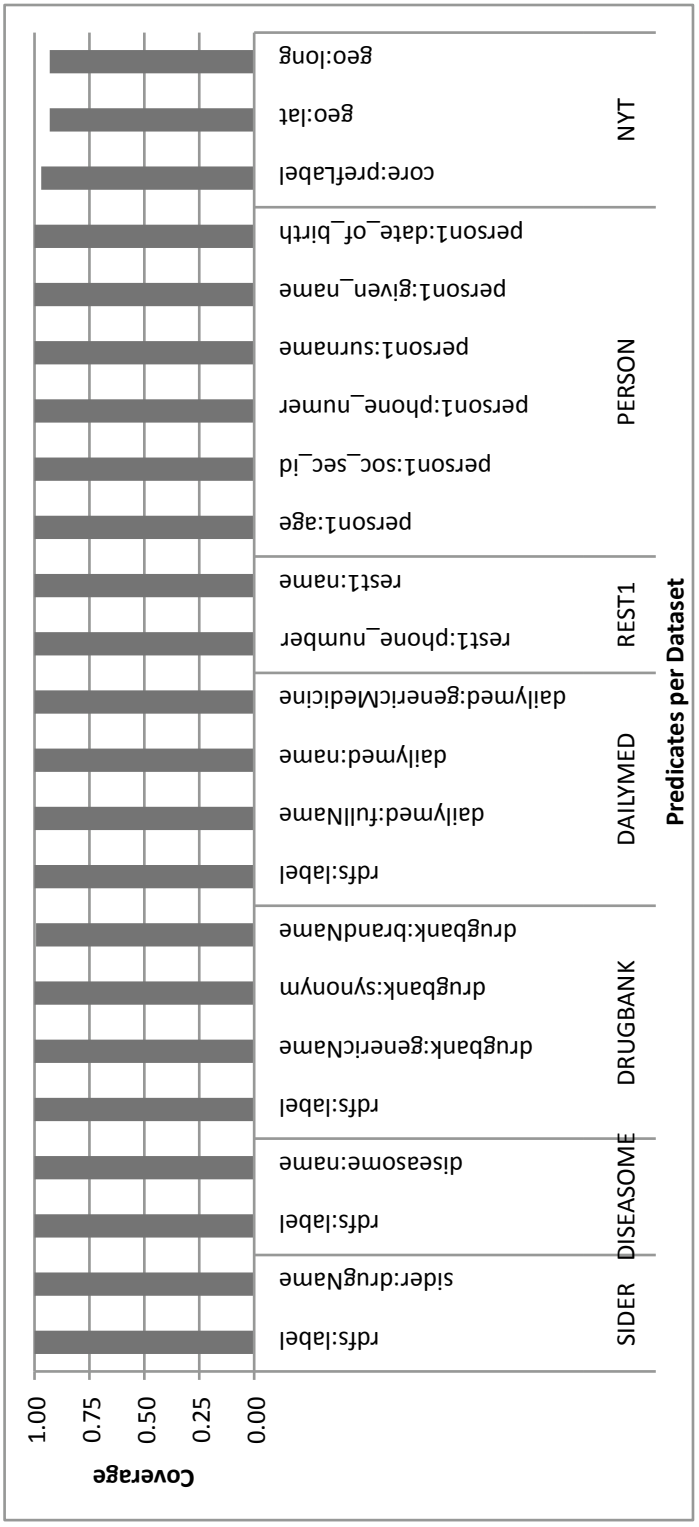

FIGURE 3.7: Coverage of predicates in the sources.

3.6.2 SERIMI Configurations

We evaluated 5 different configurations of SERIMI: (1) We evaluated SERIMI's performance without and with candidate set reduction (algorithm in Sec. 3.5.2), referred to as *S* and *S+SR*, respectively. (2) We removed different features proposed for CBM, namely predicates (S+SR-P), datatype properties (S+SR-D), object properties (S+SR-O) and tuples (S+SR-T). (3) We evaluated SERIMI's performance with the top-1 approach (S+SR+TOP1) and the threshold approach (S+SR+TH). (4) Further, direct matching is used (DM), which is compared with SERIMI's performance (class-based matching) combined with direct matching (S+SR+DM). (5) Finally, S+SR+DM+J uses Jaccard instead of FSSim (Eq. 3.9). Except for S+SR+TOP1 and S+SR+TH, top-1 was used instead of the threshold for matching tasks with one-to-one matching. We measured time efficiency and result accuracy for every configuration, using all mentioned collections in OAEI 2010 and 2011. The results are shown in Table 3.4 and Table 3.5, respectively.

Candidate Set Reduction. We observed that with candidate set reduction, SERIMI is 20% faster (average performance of S is 61s vs. 49s for S+SR). The number of candidate sets used in class-based matching could be considerably reduced. Consequently S+SR performed a much smaller number of comparisons. S+SR did not compromise accuracy as average results for S and S+SR were almost the same (F1 of 0.89 vs. 0.9).

Feature Removal. We could see that the performance improvement resulting from using less features (S+SR vs. S+SR-P, -D, -O and -T) is consistent but small in most cases. Removing predicates (S+SR-P) has the largest impact, where performance increased by 20%. This type of features represented a large part of all features used. Hence, processing was much faster without them. Removing features, however, also had a small but consistently negative impact on the accuracy. S+SR-P had the greatest impact on efficiency as well as accuracy; without predicates, F1 is 0.88 (a 0.02 loss in F1). In general, the results suggest that all proposed features are useful as they contributed to higher accuracy.

Top-1 vs. Threshold. There were no significant differences in time between the top-1 and the threshold approach (S+SR+TOP1 and S+SR+TH performances were similar). This suggests that selecting the threshold using the method in Sec. 3.5.3 requires little effort and can be done very efficiently. In terms of accuracy, S+SR+TOP1 had better average performance (86% F1) than S+SR+TH (84% F1). More specifically, S+SR+TOP1 yielded better results for tasks with one-to-one mappings. However, S+SR+TOP1 exhibited lower performance than S+SR+TH in two cases (50% F1 for Person21-Person22 and 56% F1 for Sider-Dailymed, compared to 86% and 81%, respectively), in which one-to-many mappings were needed.

Direct Matching vs. Class-based Matching. The DM (20s) approach was the fastest, followed by S+SR (50s), S+SR+DM (55s) and S (61s). Class-based matching as performed by S was expensive, requiring a much larger number of comparisons than direct matching (DM). Using candidate set reduction (S+SR), performance could be improved; S+SR is only 2.5 times slower than DM. Their combination (S+SR+DM) is slightly slower than S+SR. In return, S+SR+DM achieved the best F1 performance (93%). That is, SERIMI achieved the highest accuracy when direct and class-based matching are combined. S+SR+DM improved upon S+SR because DM could reinforce the similarity between instances when there was a direct overlap between the source and target. In some cases, such as Nyt-DB-Geo, S+SR achieved much higher F1 than DM (81% vs. 69%). The combination of the two, S+SR+DM, could leverage evidences used by both approaches to further improve the results (82%). While this simple combination led to better results on average, there was one exception where DM yielded better performance (Person11-Person12), and several cases in which S+SR produced better results (Sider-Dailymed, Sider-DB- SideEffect, Sider-DIASEASOME).

Particularly, S and S+SR performed poorly in Person11-Person12 (49% and 47%, respectively) because features of the candidate instances are very similar (e.g. they all contain phone, address and are of the type Person). Due to this, CBM produced similar scores for all candidates, which were not sufficiently distinct to separate the correct matches from the incorrect ones. For this task, DM performed better because the overlap between the source and target instances is sufficiently high to identify the correct matches.

Jaccard Similarity vs. Set-based Similarity. Observe also that the use of Jaccard in S+SR+DM+J as set similarity decreased the average F1 from 93% to 87%. This confirms our intuition that the commonalities are more relevant than the differences to define similarity in our problem setting. Regarding performance, S+SR+DM+J (53s) was slightly better than S+SR+DM (54s), in average.

Task Complexity. Fig. 3.8 shows the connection between time performances for S+SR, S+SR+DM and DM and the number of triples in the candidate sets, which captures the amount of data that has to be processed. Clearly, more time was needed when more candidates and data have to be processed. Time performance for all 3 configurations increased quite linearly with a larger amount of data. To assess the complexity from the viewpoint of accuracy, we used the TC measure discussed before. Fig. 3.5 shows the connection between F1 performances for S+SR, S+SR+DM and DM and TC. We observed there was a trend between complexity and F1: F1 decreased as complexity increased. Interestingly, we could see many cases, including Person21-Person22 and Nyt-DB-Geo, where S+SR and DM are complementary, i.e. S+SR had a higher performance

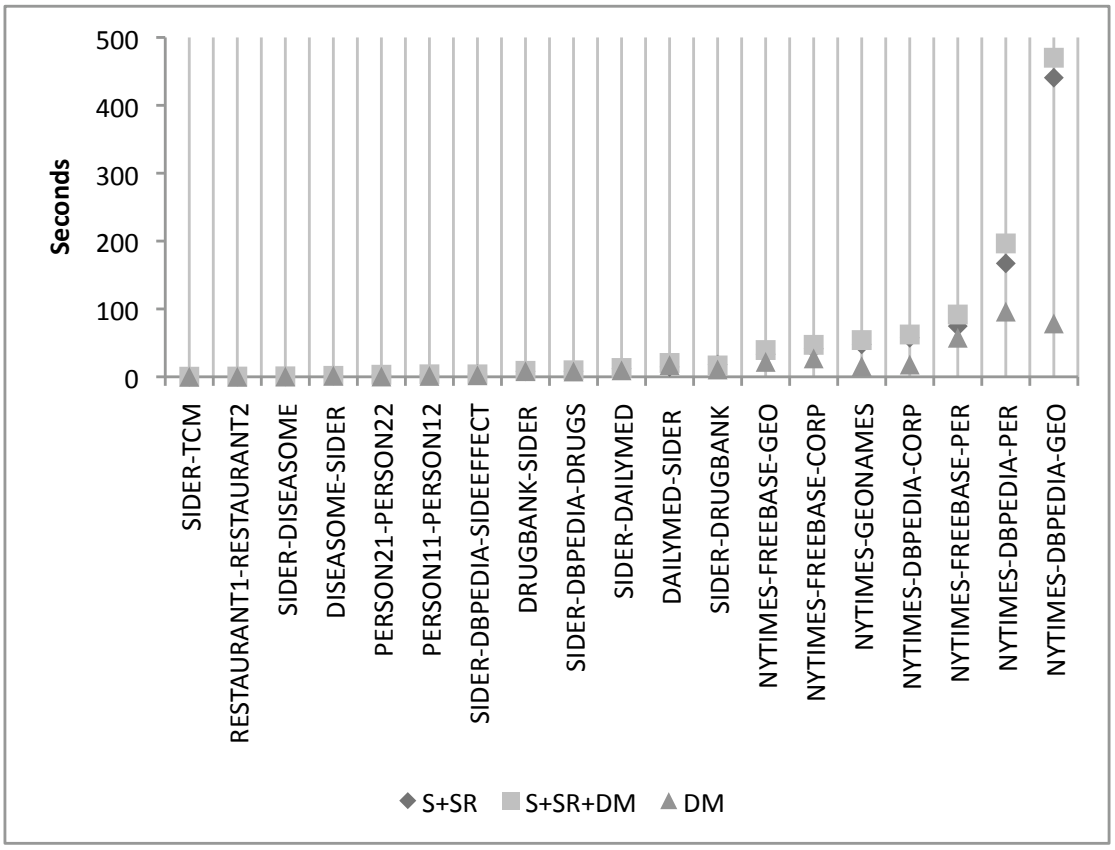

FIGURE 3.8: Time performance; tasks are ordered according to the number of triples in the candidate sets.

when DM had a lower performance, and vice-versa. S+SR+DM was most helpful in these cases as it could leverage the complementary nature of these two approaches to improve the results.

Concluding, the highest accuracy is achieved by combining class-based matching with direct matching. Further, candidate set reduction helps to improve time efficiency. In the following experiments, we will use S+SR+DM, in combination with the top-1 approach where there is an one-to-one mapping or the threshold approach otherwise.

TABLE 3.4: Time performance for different SERIMI configurations, in seconds.

Datasets	S	S+SR	S+SR+DM	S+SR+DM+J	DM	S+SR+TH	S+SR+TOP1	S+SR-P	S+SR-D	S+SR-O	S+SR-T
Dailymed-Sider	22.6	14.57	20.41	29.87	17.34	14.15	14.15	15.33	14.44	14.03	13.93
Diseasome-Sider	1.75	1.38	1.45	1.86	1.71	1.46	1.37	1.36	1.41	1.44	1.36
Drugbank-Sider	8.85	8.12	8.84	10.79	8.33	7.79	7.64	7.67	8.62	7.84	7.56
Nytimes-DB-Corp	64.08	57.52	62.37	73.68	18.03	56.34	58.62	48.06	55.39	52.76	49.74
Nytimes-DB-Geo	606.43	440.71	470.12	435.19	78.63	441.12	437.56	365.62	421.33	430.98	413.31
Nytimes-DB-Per	159.13	167.14	196.53	190.75	96.07	172.58	167.27	145.19	163.92	163.29	162.24
Nytimes-Freebase-Corp	47.43	41.39	47.11	44.86	27.15	40.34	47.19	37.92	37.79	38.97	38.84
Nytimes-Freebase-Geo	33.41	33.34	39.44	38.81	21.99	32.25	35.93	28.38	31.91	34.59	34.05
Nytimes-Freebase-Per	78.76	74.68	91.79	87.95	57.56	75.29	73.65	70.04	77.12	71.94	72.7
Nytimes-Geonames	73.75	47.85	54.13	45.02	15.72	51.51	47.95	35.67	46.28	43.29	35.45
Person11-Person12	3.29	3.11	3.7	3.21	1.34	3.04	3.26	2.58	2.86	2.8	2.45
Person21-Person22	2.73	2.86	3.08	2.38	0.47	2.86	2.84	2.28	2.46	2.51	1.79
Rest1-Rest2	0.32	0.36	0.38	0.31	0.14	0.33	0.34	0.27	0.33	0.29	0.27
Sider-Dailymed	20.53	11.92	13.02	12.72	9.58	12.87	11.64	11.3	12.24	12.72	9.99
Sider-DB-Drugs	9.52	8.3	9.63	8.81	7.89	8.35	8.05	7.55	8.04	7.64	8.51
Sider-DB-SideEffect	4.37	4.1	3.63	3.38	2.3	3.5	3.35	2.42	2.72	2.68	2.67
Sider-Diseasome	1.05	0.56	0.71	0.67	0.48	0.54	0.55	0.48	0.54	0.56	0.52
Sider-Drugbank	22.77	18.05	16.76	17.53	10.84	14.42	14.09	12.94	13.41	14.24	12.99
Sider-Tcm	0.41	0.14	0.17	0.19	0.15	0.14	0.14	0.15	0.13	0.13	0.13
AVERAGE	61.11	49.27	54.91	53.05	19.77	49.41	49.24	41.85	47.42	47.51	45.71

TABLE 3.5: F1 performance for different SERIMI configurations.

Datasets	S	S+SR	S+SR+DM	S+SR+DM+J	DM	S+SR+TH	S+SR+TOP1	S+SR-P	S+SR-D	S+SR-O	S+SR-T
Dailymed-Sider	1.0	1.0	1.0	1.0	1.0	0.99	1.0	1.0	1.0	1.0	1.0
Diseasome-Sider	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Drugbank-Sider	1.0	1.0	1.0	1.0	1.0	0.98	1.0	1.0	1.0	1.0	1.0
Nytimes-DB-Corp	0.88	0.88	0.91	0.85	0.83	0.78	0.88	0.87	0.88	0.88	0.88
Nytimes-DB-Geo	0.81	0.81	0.82	0.63	0.69	0.36	0.81	0.79	0.81	0.81	0.81
Nytimes-DB-Per	0.95	0.95	0.95	0.94	0.93	0.91	0.95	0.95	0.95	0.95	0.95
Nytimes-Freebase-Corp	0.92	0.92	0.92	0.84	0.92	0.88	0.92	0.86	0.92	0.92	0.92
Nytimes-Freebase-Geo	0.92	0.92	0.93	0.83	0.92	0.87	0.92	0.88	0.92	0.93	0.93
Nytimes-Freebase-Per	0.95	0.95	0.95	0.93	0.95	0.94	0.95	0.95	0.95	0.95	0.95
Nytimes-Geonames	0.78	0.78	0.87	0.49	0.87	0.4	0.78	0.64	0.78	0.78	0.78
Person11-Person12	0.47	0.47	0.95	0.95	0.97	0.49	0.47	0.46	0.48	0.46	0.46
Person21-Person22	0.86	0.86	0.91	0.91	0.91	0.86	0.5	0.86	0.86	0.86	0.86
Rest1-Rest2	0.96	0.96	0.97	0.97	0.97	0.94	0.96	0.96	0.96	0.96	0.96
Sider-Dailymed	0.83	0.81	0.74	0.55	0.72	0.81	0.56	0.73	0.8	0.79	0.79
Sider-DB-Drugs	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
Sider-DB-SideEffect	0.9	0.9	0.89	0.89	0.89	0.89	0.9	0.9	0.9	0.9	0.9
Sider-Diseasome	0.91	0.91	0.89	0.9	0.88	0.91	0.91	0.92	0.91	0.91	0.91
Sider-Drugbank	0.97	0.97	0.98	0.99	0.98	0.96	0.97	0.97	0.97	0.97	0.97
Sider-Tcm	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
AVERAGE	0.90	0.89	0.93	0.87	0.91	0.84	0.86	0.88	0.89	0.89	0.89

3.6.3 SERIMI vs. Alternative Approaches

We compared SERIMI with state-of-the-art approaches. We carefully selected in the literature systems that reported the best performance in the benchmark that they participated. Those systems represents a large number of approaches used for instance matching.

We compared SERIMI with *RIMOM* and *ObjectCoref2010* (OC2010) using the data and results of OAEI 2010 [46]. To ensure the validity of this evaluation, we also included recently published results for ObjectCoref [58], called ObjectCoref2012 (OC2012). Using OAEI 2011 data and published results [26], we compared SERIMI with *AgreementMaker* (AM) and *Zhishi.links* (Zhi). Using the same data, we also compared SERIMI with the latest state-of-the-art approaches for instance matching, which did not participate at OAEI: *PARIS* [37] and *SIFI-Hill* [47].

OAEI 2010. Table 3.6 shows results for OAEI 2010. Missing values in the table indicates that the results were not published by the authors at OAEI. On average, SERIMI largely outperformed both systems. As shown in Table 3.6, SERIMI (93% F1) largely outperformed RIMON (72% F1) on average. SERIMI achieved considerable performance gain for the life science collection. Here, CBM played an important role because source and target instances often belong to different classes. In Sider-Dailymed for instance, there were instances of the types Drug and Ingredient sharing the same name that were incorrectly identified as candidate matches; these false positives were rejected by SERIMI thanks to CBM.

TABLE 3.6: F1 performance for SERIMI, OC2010, RIMON, OC2012 over OAEI 2010 data; some results were not available for OC2010, RIMON OC2012.

Datasets	SERIMI	OC2010	RIMON	OC2012
Sider-Dailymed	**0.74**	-	0.62	-
Sider-Diseasome	**0.89**	-	0.45	-
Sider-Drugbank	**0.98**	-	0.50	-
Sider-Tcm	**0.99**	-	0.79	-
Dailymed-Sider	**1.0**	0.70	0.62	-
Diseasome-Sider	**0.97**	0.74	-	-
Drugbank-Sider	**1.0**	0.46	-	-
Person11-Person12	0.95	**1.0**	**1.0**	**1.0**
Person21-Person22	0.91	0.95	**0.97**	0.95
Restaurant1-Rest.2	**0.97**	0.73	0.81	0.89
Average (OC2010)	**0.97**	0.76	-	-
Average (RIMON)	**0.93**	-	0.72	-
Average (OC2012)	**0.97**	-	-	0.95

TABLE 3.7: F1 performance for OAEI 2011.

Datasets	SERIMI	AM	Zhi	SIFI	PARIS
Nyt-DB-Corp	**0.91**	0.74	**0.91**	0.84	0.65
Nyt-DB-Geo	0.82	0.69	**0.92**	0.82	0.03
Nyt-DB-Per	0.95	0.88	0.97	**0.98**	0.06
Nyt-Freebase-Corp	**0.92**	0.80	0.87	0.80	0.82
Nyt-Freebase-Geo	**0.92**	0.85	0.88	0.64	0.60
Nyt-Freebase-Per	0.95	0.96	0.93	**0.97**	0.66
Nyt-Geonames	0.87	0.85	**0.91**	0.72	0.46
Average	**0.91**	0.82	**0.91**	0.82	0.47

SERIMI was outperformed by OC2010 and RIMON in the Person collection. One reason is that this data involves artificially generated spelling mistakes. OC2010 and RIMON employed special direct matching strategies to deal with that. More importantly, SERIMI could not yield better results because CBM has limited impact when all candidates belong to the same class and the data schema is well-defined. In this scenario, all instances belong to the class Person and the source and target schema completely overlap. Thus, instances did not greatly vary in terms of class related information.

Also compared to OC2012, which only published results for the easiest matching tasks, SERIMI achieved better average performance (97% F1).

OAEI 2011 As shown in Table 3.7, SERIMI had the same average performance as Zhi. In particular, Zhi performed better in tasks involving the location datasets (DB-Geo and GeoNAMES) because as opposed to SERIMI, it made use of domain knowledge and location-specific similarity functions. SERIMI largely outperformed SIFI (91% vs. 82%). SIFI had slightly better performance than SERIMI for Nyt-DB-Per and Nyt-Freebase-Per. With these tasks, SIFI was able to obtain more fine-tuned thresholds, which led to better results. As opposed to SIFI, which relies on training data for this threshold tuning, SERIMI is completely unsupervised. For SIFI, we used 10% of the OAEI ground truth as positive examples, and 10% of wrong alignments in the candidate sets as negative examples. PARIS obtained average performance of 47% F1, which was considerably worse than SERIMI. PARIS used both schema- and data-level features for matching. However, it only employed exact matching, i.e. it considers instances as matches when their features exactly match. In PARIS's authors experiments, good results could be achieved because exact matching was sufficient for the tasks involved. With the tasks studied here, exact matching led to very low performance.

Overall, the results show that SERIMI achieved the best accuracy results. Further, there is room for improvement as SERIMI so far neither uses training data nor exploits domain knowledge. Training data, for instance, could be exploited to fine tune the threshold (as implemented by SIFI).

3.7 Related Work

Instance matching across datasets involves the use of similarity functions, thresholds and comparable attributes. They are captured by a matching scheme. While the majority of approaches use a flat representation of instances based on attribute values, other features might be applied. We will discuss existing approaches along these dimensions of features, similarity functions and matching schemes.

Matching Features. Instance features are derived from flat attributes, structure information (e.g. relations between RDF resources) [3, 59, 60] or semantic information extracted from ontologies. ObjectCoref [35] for instance, exploits the semantics of OWL properties such as *owl:InverseFunctionalProperty* and *owl:FunctionalProperty*. Also, the combination of instance-level and schema-level features have been explored by PARIS [37], which jointly solve the problem of instance and schema matching.

SERIMI targets the heterogeneous scenario, where no structure, semantic or schema information is available in the worst case. It is based on a simple *flat representation*, where instances are captured as a set of attribute values. This representation is employed for single instances as well as for class of instances, which are needed for CBM.

Similarity Functions. The choice of similarity functions depends on the nature of the features. For strings, character-based (e.g. Jaro, Q-grams), token-based (e.g. SoftTF-IDF, Jaccard) and document-based functions (e.g. cosine similarity) were used [61]. In addition to using syntactic information, special similarity functions have also been proposed to exploit different kinds of (lexical) semantic relatedness [62, 63].

Also along this dimension, we pursued a simple approach where only tokens are employed. However, for our new problem of CBM, which involves comparing sets of instances, we propose a *set-based similarity function* that take the token overlaps between sets into account.

Matching Schemes. With approaches relying on a flat representation of instances, i.e., attribute values, the employed schemes contain the similarity functions, thresholds and comparable attributes. Comparable attributes are either computed via automatic schema matching or assumed to be manually defined by experts [45]. Then, techniques with different degrees of supervision are employed for learning the scheme. Knofuss+GA[64] is an unsupervised approach that employs a genetic algorithm for learning. SIFI [47], AdaBoost [39] and OPTrees [56] represent supervised approaches that learn the schemes from a given set of examples. Others approaches such as Zhishi.links [45], RIMON [65] and Song et. al [52] assume matching schemes that for the most part,

were manually engineered, i.e., the similarity functions and thresholds were defined manually. They focus on the problem of learning the best comparable attributes.

The above solutions focus on direct matching. As oppose to that, class-based matching does not rely on a complex scheme. It uses a special similarity function we specifically design for this matching task. The problem of finding the threshold is cast as the one of detecting outliers, for which we propose an unsupervised solution.

Overall, our solution can be characterized as an unsupervised, simple, yet effective solution, which employs a novel class-oriented similarity function, matching technique and threshold selection method to exploit the space of class-related features never studied before.

There are other systems tackling the same problem. For instance, Linda [66] is an entity matching system for web scale that was evaluated over a small subset of the datasets that we consider here. The reported results have a lower accuracy compared to the systems used here.

3.8 Conclusion

In this work, we propose an unsupervised instance matching approach that combines direct-based matching with a novel class-based matching technique to infer Sameas relation over heterogeneous data. We evaluated our method using two public benchmarks: OAEI 2010 and 2011. The results show that we achieved good and competitive results compared to representative systems focused on instance matching over heterogeneous data.

Chapter 4

Efficient and Effective On-the-fly Candidate Selection over Sparql Endpoints

Instance matching is the problem of finding instances that refer to the same real-word entity. This task is a crucial step towards Web data integration. It is challenging due to the heterogeneous nature of Web data that often is only accessible through remote data endpoints. In this work, we drop the assumption that data is available in advance, and cast instance matching as the problem of answering queries over remote endpoints. In particular, we propose *instance-specific matching schemes* that for every instance use several queries to retrieve the various heterogeneous candidate matches *on-the-fly* in a pay-as-you-go fashion. As the number of these candidate selection queries might be large and the cost of executing them over remote endpoints is high, we propose a *heuristic-based search optimization framework* that is used to prune non-optimal queries. We show that compared to two baselines, the proposed solution not only yields higher quality results but also better runtime performance.

4.1 Introduction

Interlinking datasets is a crucial step towards Web data integration. A prominent initiative towards this goal is the Linking Open Data project (Linked Data),[1] which integrates publicly available datasets by establishing "same-as" links between their instances that represent the same real-world entity. Effectively establishing these links is a challenging task due to data *heterogeneity*. Especially in this Web of data scenario, instances are not only different at the data level (variations in names, values etc.) but also, are described using schemas that largely vary. The following example illustrates this, showing two pairs of instances representing the same entity from two datasets that are described using different schema attributes:

```
SIDER dataset:
<sider:12312> <label> "Morphine"
<sider:12312> <type> <sider:Drug>
<sider:43434> <title> "Eosinophilic Pneumonia"
<sider:43434> <type> <sider:Drug>
```

```
DRUGBANK dataset:
<drugbank:DB00295> <drugname> "Morphine Sulphate"
<drugbank:DB00295> <synonym> "Morphine"
<drugbank:DB00295> <type> <drugbank:Drug>
<drugbank:DB00295> <affectedOrganism> "Mammals"
<drugbank:DB00001> <drugname> "Morphine"
<drugbank:DB00001> <type> <drugbank:Ingredient>
<drugbank:DB00494> <drugname> "Eosinophilic Pneumonia"
<drugbank:DB00494> <type> <drugbank:Drug>
<drugbank:DB00494> <affectedOrganism> "Mammals"
```

This linking problem has been studied widely under various labels, such as entity resolution, linking discovery, record linkage and *instance matching* [6, 31, 58, 67, 68]. It is typically solved in two steps, namely to find candidate matches first using a relatively simple but quick matching technique, called *candidate selection* (also referred to as blocking in the offline scenario) [2, 27, 48], and followed by a refinement step using a more advanced but also more expensive technique, the actual *instance matching*. Different techniques for learning *instance matching schemes* exist that capture weighted combinations of similarity attributes, measures and thresholds to be used for computing and selecting the resulting matches [38, 69]. A *candidate selection scheme* is typically more simple, e.g. uses equal weights for attributes and binary similarity that indicates true or false (instead of a degree of similarity). Since instance matching in this scenario has to be performed across heterogeneous schemas, it actually also involves schema matching [70] to find some attributes from one source dataset are similar to some others in the target datasets such that their values can be compared and used to find matches.

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

Given `label` and `drugname` are found to be a pair of comparable attributes, a candidate selection scheme for the example is $\langle label, drugname \rangle$. For example, for the given source instance `12312` with "Morphine" as `label`, this scheme can be employed to find in the target dataset that `DB00295` is a possible candidate, because its `drugname` also contains the key value "Morphine". However, using this as a *general scheme for all instances* is problematic in this heterogeneous case: for example, the same scheme does not apply to the instance `43434` because it does not have a `label` attribute. Solving this requires learning several schemes and executing them against all instances in multiple runs. For this example, the other scheme needed would be $\langle title, drugname \rangle$. In this work, we propose *instance-specific schemes* to address candidate selection over heterogeneous datasets, which are not applied to all but a particular instance.

A challenging aspect in Linked Data integration is the distributed nature of data access, usually through *SPARQL endpoints* [71–74]. Existing candidate selection and instance matching techniques tailored to the offline scenario assume that data is available locally (and can be indexed for efficient processing). In the Linked Data setting, this requires downloading complete datasets from endpoints and also, periodically crawling data updates to perform incremental data integration. In many cases however, a copy of the entire dataset is not available for download, and, the data is only accessible by querying a SPARQL endpoint. While SPARQL queries can be used to selectively retrieve some data, retrieving the entire dataset often hits time limits imposed by the endpoint providers [75, 76]. Further, the target data to be interlinked may represent only a small fraction of the entire target dataset. Downloading the whole dataset in these cases is rather inefficient.

Instead of offline data integration, we propose to study the problem of *on-the-fly candidate selection over SPARQL endpoints*. We cast the problem of candidate selection as querying over remote SPARQL endpoints for possible candidate matches for a given instance. For example, candidates for `12312` can be retrieved using the following SPARQL[2] queries over the Drugbank data endpoint:

```
select ?s where {?s ?p "Morphine"}
select ?s where {?s ?p "Morphine"@en}
select ?s where {?s <drugname> "Morphine"}
select ?s where {?s <synonym> "Morphine"}
```

As the number of candidate queries might be large and the cost of executing them over remote endpoints may be high, the problem tackled in this work is to find instance-specific schemes (i.e. queries) that not only deliver high quality candidates, but also, can be executed efficiently. For example, although all queries above can retrieve candidates

[2]We use a simplified SPARQL syntax for the sake of presentation.

for `12312`, the first query may run faster than the others, while the last two queries are more selective and may be more precise. For a large number of instances, the choice of the most efficient query makes a big different in the quality of the candidates selected and the overall execution time. The list of issues regarding the trade-off between quality and efficiency is broad and will be discussed throughout this chapter.

Our research advances the state-of-the-art by (1) considering schemes that are instance-specific, (2) conceiving on-the-fly candidate selection as a querying problem, (3) and optimizing schemes not only for effectiveness but also efficiency.

Contributions. To this end, we provide the following contributions. (1) We show how on-the-fly candidate selection can be solved by *learning queries* for every instance to establish an instance-specific candidate selection scheme. As queries, we not only consider attributes but also class-related information learned on-the-fly from data obtained at query time that is treated as training examples. (2) Targeting both precision and recall requires dealing with a large number of candidate queries. To improve efficiency, we propose a *heuristic-based search optimization framework* that aims to select and execute only a small number of queries discovered to be not only more quality- but also time-optimal. (3) We evaluated the proposed approach, called *Sonda*, using the Ontology Alignment Evaluation Initiative (OAEI) 2010 and 2011 instance matching benchmarks. We adapted existing works to this on-the-fly setting to obtain non-trivial baselines to compare with Sonda. Compared to these baselines, Sonda took 11 minutes and achieved an F1 measure of 85% on average, while the best baseline, S-based required 14 minutes for an F1 measure of 73%. Also, we used Sonda, as the candidate selection strategy, in combination with an instance matcher, SERIMI [25], to refine candidates. Using Sonda, the results of this matcher greatly improved, which we attribute to the improved accuracy in the first stage of the instance matching pipeline. The final matches produced by Sonda in combination with this matcher were about 10% better in terms of F1, compared against the best results reported by the benchmark.

Outline. This chapter is organized as follows: we present the problem in Sec. 4.2. In Sec. 4.3, we elaborate on the technique for building candidate selection queries. The refinement and optimal execution of these queries to retrieve candidates are discussed in Sec. 4.4. Sec. 4.5 presents the experimental results, followed by related work and conclusions in Sec. 4.6 and 4.7, respectively.

4.2 Overview

We focus on RDF and other types of data that can be modeled as graphs. Let $\mathbf{G}$ denote the set of all data graphs. Then, every dataset is conceived as a graph $G \in \mathbf{G}$ comprising a set of triples:

Definition 4.1 (Data Model)**.** A dataset is a graph G formed by a set of triples (s, p, o) where $s \in U$ (called subject), $p \in U$ (predicate) and $o \in U \cup L$ (object), and U and L denote the sets of Uniform Resource Identifiers (URIs) and literals, respectively. Each literal $l \in L$ is conceived as a bag of tokens, $l = \{t_1, \ldots, t_i, \ldots, t_n\}$, drawn from a vocabulary V, i.e. $t_i \in V$.

With respect to this model, *instances* are resources, i.e. URIs, that appear at the subject position of triples. An instance representation can be obtained from the data graph as follows:

Definition 4.2 (Instance Representation)**.** The instance representation is a function $I : U \times \mathbf{G} \rightarrow \mathbf{G}$, which given an instance $s \in U$ and a graph $G \in \mathbf{G}$, maps s to a set of triples in which it appears as the subject, i.e. $I(s, G) = \{(s, p, o) | (s, p, o) \in G\}$.

We also use $P(s, G) = \{p | (s, p, o) \in G\}$ and $O(s, p, G) = \{o | (s, p, o) \in G\}$ to denote the set of predicates associated with an instance s, also called *attributes*, and the set of objects associated with s via p, called *attribute values*, respectively.

4.2.1 Problem - Find Candidate Matches

Instance matching is the problem of finding different instance representations that refer to the same real-world entity:

Definition 4.3 (Instance Matching)**.** Given a set of source instances S, a set of target instances T, and all (s, t) in $S \times T$, instance matching is the problem of finding all pairs of instances (s, t) such that $sim(s, t) > \alpha$, where $sim(s, t)$ is a similarity function, $sim : U \times U \rightarrow \mathbf{R}^+$, which for the given instances $s, t \in U$, returns a number representing a degree of similarity between s and t.

Typically, $sim(s, t)$ is captured by an *instance matching scheme* that is actually a weighted combination of similarity functions defined over the instances' attribute values, i.e. $sim(s, t) = \sum_{p \in P(s, G_S) \cap P(t, G_T)} w_p \cdot sim(O(s, p, G_S), O(t, p, G_T)) > \alpha$, where G_S is the source dataset containing s and G_T the target dataset containing t. When the overall similarity $sim(s, t)$ exceeds the threshold α, the instances s and t form a *match*. Then, whether this match computed by the algorithm is indeed correct or

not, i.e. refer to the same real-world entity or not, is verified by the ground truth. In the heterogeneous setting, the datasets G_S and G_T may exhibit differences in their schemas such that instead of using the same attribute p that is both in G_S and G_T, pairs of comparable attributes $\langle p_S, p_T \rangle$ have to be found. Let P_{ST} be the set of all comparable pairs of attributes in G_S and G_T. Then, the extended scheme $sim(s,t) = \sum_{\langle p_S,p_T\rangle \in P_{ST}} w_{\langle p_S,p_T\rangle} \cdot sim(O(s,p_S,G_S), O(t,p_T,G_T)) > \alpha$ can be defined for this setting to capture that the values of the source attribute p_S shall be compared with those of the target attribute p_T. Accordingly, instance matching entails the main sub problems of (A) finding pairs of comparable attributes P_{ST} (schema matching) as well as (B) choosing and (C) weighting them, and determining the (D) similarity functions (e.g. Edit Distance, Jaccard) and (E) the threshold α.

To avoid the $|S| \times |T|$ instance comparisons required for this task, solving the instance matching problem is often preceded by a *candidate selection* step, which entails a subset of the sub problems mentioned above. Given an instance $s \in S$, candidate selection is the problem of quickly selecting a reduced set $T' \subset T$ of possible *candidates matches* for s:

Definition 4.4 (Candidate Selection)**.** Given an instance $s \in S$, the target instances T, candidate selection is the problem of finding $T' \subset T$ such that $\forall t \in T'$, $sim_b(s,t) = true$, where the similarity function here is defined as $sim_b : L \times L \rightarrow \{true, false\}$, i.e. it returns $true$ if the given values are similar, or $false$ otherwise.

Analogous to instance matching, $sim_b(s,t)$ here is actually evaluated using a single scheme, called the *candidate selection scheme*, which is a conjunction of similarity functions, i.e. $sim_b(s,t) = \bigwedge_{\langle p_S,p_T\rangle \in P_{ST}} sim_b(O(s,p_S,G_S), O(t,p_T,G_T))$. In this context, $\langle p_S, p_T \rangle$ is has also been referred to as blocking key, or blocking key pair in a setting with heterogeneous datasets. Note the difference to instance matching is that here, a relaxed notion of similarity is used. Instead of a degree of similarity, we only need to know if instances are similar or not. This eliminates the sub problem of finding the threshold. Also, the similarity function is assumed to be defined manually and all attributes are of equal importance such that finding weights is no longer a problem. As a result, candidate selection targets only two of the five main sub problems above, namely (A) finding comparable attribute pairs and (B) choosing the most selective ones to be included in the scheme.

This chapter deals with the *candidate selection problem over remote endpoints*. In particular, we focus on the *on-the-fly* pay-as-you-go setting where we are interested in finding candidate matches for all instances I in G_s, or a subset S of I, where both the tasks of learning how to determine candidate matches, i.e. the *scheme*, and retrieving them, have

to be performed online (over remote endpoints). Without loss of generality, we assume that the instances in G_s can be grouped into sets of source instances S that belong to the same class of interest Z, i.e. $\forall(s \in S).(s, \texttt{rdf:type}, Z) \in G_S$, where `rdf:type` is an attribute predefined in RDF representing that Z is a class of s. A possible goal is to find target matches T' in G_T for source instances S in G_S that belong to the class $Drug$, i.e. $\forall(s \in S).(s, \texttt{rdf:type}, Drug) \in G_S$. We do so because class information expressed as RDF properties (e.g. `rdf:type`) can be used to produce more selective queries. Those queries result in better quality candidate selection. However, class information is not necessary for the method to work. As we will show, dropping this assumption results in only a small loss of quality.

4.2.2 Existing Solutions

State-of-the-art instance matching / candidate selection methods are based on supervised learning, leveraging training data to evaluate the errors and refine the learned candidate schemes. Optimal schemes found are those which maximize the coverage of positive examples while avoiding negative examples [56]. However, the required training data has to be preprocessed off-line.

In this work, we focus on solutions for on-the-fly candidate selection over possibly remote endpoints, for which the availability of instance-level training data cannot be assumed. A previously proposed unsupervised approach [38], called *S-based*, assumes precomputed schema mappings such that the comparable attribute pairs are known. Then, it chooses them based on their coverage and discriminability, two metrics derived directly from the data reflecting the number of instances a given attribute can be applied to and how well it distinguishes them. An alternative method to that, referred to as S-agnostic, has been proposed in [77] as a schema-agnostic unsupervised approach to candidate selection. It does not use attributes for matching but treats instances simply as bags of value tokens. Instances sharing the same token (in any attribute) are placed in one candidate set, i.e. the similarity function is based on value token overlap. This approach does not require any effort for learning the scheme and is particularly suited when there is a lack of schema overlap such that only few or no comparable attributes exist. We will discuss how S-agnostic and S-based can be adapted for on-the-fly candidate selection and used as non-trivial baselines.

4.2.3 Sonda

The primary distinguishing feature of Sonda lies in the granularity of the learned scheme. Instead of using a single scheme for all instances, Sonda optimizes a scheme for every

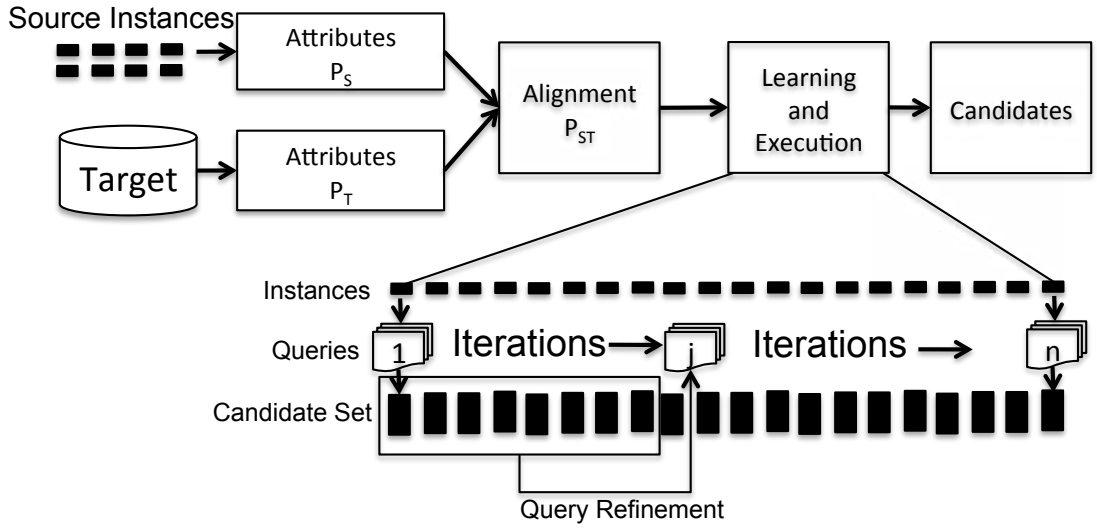

FIGURE 4.1: The process of learning queries and executing them.

individual instance. More precisely, candidate selection is formulated as a querying problem where for every instance, the learned scheme is a set of candidate selection queries.

Further, an aspect neglected so far is time efficiency. Previous work on finding the scheme as discussed above focuses on the quality of matches. We consider time as an additional optimization objective such that optimal schemes (queries in Sonda) are those that yield high quality candidates and can be executed efficiently. Moreover, not only the execution but also the learning of schemes should be time efficient.

The overall process of learning and executing queries is illustrated in Fig. 4.1. The inputs include a set of instances from the source dataset. Selecting candidates from the target dataset consists of four main tasks. Firstly, (1) comparable attributes are determined (2) and then iteratively for every source instance, queries are constructed using the comparable attributes and information in the instance representation. (3) The efficient and effective queries are selected and executed to retrieve candidates. The selection of queries is performed through heuristic-based search optimization [78], which for every instance, decides which queries to be used for execution and when to move on to the next iteration (next instance). (4) During this iterative process, the retrieved candidates are treated as additional information (i.e. training example) for learning queries more refined than those that could be derived from comparable attributes in step 2.

4.3 Learning Queries

We use star-shaped conjunctive queries that constitute a fragment of the Basic Graph Pattern (BGP) feature of SPARQL [79].

Definition 4.5 (Query, Query Component)**.** A star-shaped query $Q(x)$ is a conjunction of query predicates defined as $Q(x) = (x).(x, p_1, o_1) \wedge \ldots \wedge (x, p_n, o_n)$ where x is the only (free) variable and all p_i and o_i are constants, for $1 \leq i \leq n$. A *query component* is a query with one predicate, i.e. $(x).(x, p, o)$.

Intuitively, such a query can be conceived as triple patterns which together form a star-shaped pattern around the variable node x. Bindings to x are instances. In the definition above, we do not consider the unbound variable x on the object position because candidate instances are subject instances that contain a specific literal value as a predicate value. For example, the following query retrieves all instances of the type *Drug* that have "Eosinophilic Pneumonia" as *drugname*:

$$(x).(x, drugname, Eosinophilic\ Pneumonia) \wedge (x, type, Drug).$$

Equivalently in SPARQL, this query can be written as follows:

```
SELECT ?x WHERE {?x <drugname> "Eosinophilic Pneumonia" .
?x type Drug .}
```

We note that besides exact value matching, which in this case returns instances of the type *Drug*, with attribute *drugname* equal to "Eosinophilic Pneumonia", most data endpoints also support fuzzy matching. Using SPARQL endpoints, we consider using query types EXACT, as well as EXACT_LANG, LIKE, AND and OR, as follows:

```
EXACT: SELECT  ?s
WHERE {?s label ‘‘Eosinophilic Pneumonia’’ }

EXACT_LANG: SELECT  ?s
WHERE {?s label ‘‘Eosinophilic Pneumonia’’@en }

LIKE: SELECT  ?s  WHERE {?s label ?o FILTER
regex(?o,‘‘Eosinophilic Pneumonia’’) }

AND: SELECT  ?s  WHERE {?s label ?o FILTER
regex(?o,‘‘Eosinophilic’’)&&regex(?o,‘‘Pneumonia’’)}

OR: SELECT  ?s  WHERE {?s label ?o FILTER
regex(?o,‘‘Eosinophilic’’)||regex(?o,‘‘Pneumonia’’)}
```

The EXACT_LANG query returns instances that match exactly on label value, with the additional constraint that the label value should be annotated with tag *@en* to

indicate that the value is in English. The second LIKE query retrieves instances with a *label* value containing the string "Eosinophilic Pneumonia". Results for the third AND (fourth OR) query have a value containing the string "Eosinophilic" and (or) the string "Pneumonia". Note that these query types represent different similarity functions. While EXACT corresponds to exact value matching, OR corresponds to value token overlap. The latter is used to implement the S-based and S-agnostic baselines. These queries implement the function $sim_b(\cdot,\cdot)$ as required by candidate selection, i.e. return results that match. They are however not designed for instance matching, which requires further filtering based on a degree of similarity.

We learn queries that consist of a small but fixed number of components. Multiple queries may be used per instance, to achieve high recall - they avoid missing positive matches when they do exist. To achieve high precision, only the results of the "best" queries are taken into account. In particular, we focus on queries that are composed of at most one *attribute component* and one *class component*. The attribute component is a keyword query based on a discriminative key that is used to select candidates (e.g. $(x).(x, drugname, \text{“}Eosinophilic\ Pneumonia\text{”})$), while the class component is a query that selects a class of instances (e.g. $(x).(x, type, Drug)$). When combined, the attribute component selects candidates that share a similar key to the source instance and the class component prune those candidates that do not belong to the class of interest. Attribute components are learned from the set of comparable attributes that were obtained prior to the iterative selection process while class components are learned from positive and negative matches that are obtained during the process.

We now describe the main steps involved in building queries.

4.3.1 Finding Comparable Key Pairs

Finding comparable key pairs requires finding discriminative attributes in the source and the target, P_S and P_T, and constructing pairs of comparable attributes, P_{ST}. To generate P_S and P_T, we use the approach proposed by Song. et al. [38], which selects attributes based on their discriminative power and coverage.

Instead of assuming the data to be available offline, we apply this approach over a sample of data obtained by querying the source and target endpoints. Sampling is crucial because retrieving all source and target data needed for learning is expensive.

To obtain the source data sample, we randomly select X% (1% in the experiment) from the given set of source instances S, i.e. $S' \subset S$, and retrieve their representations,

$I(S') = \bigcup_{s \in S'} I(s, G_S)$, from the source endpoint. Then, we apply Song et al.'s approach over the triples in $I(S')$ to obtain the keys P_S.

To obtain the target keys P_T, we sample from the target dataset. Note the source data sample captures only data related to the given source instances $S' \subset S$. Likewise, we propose to select a sample from the target that is related to S'. In particular, we select all target instances that match attribute value of some instances in S'. For this, we use queries of the form $(x).(x, p, o_S)$ to retrieve data from the target endpoint, where x and p are variables and o_S is a value in a triple $(s_S, p_S, o_S) \in I(S')$. For example, for $o_S =$ "*Eosinophilic Pneumonia*", the query SELECT ?s WHERE {?s ?p "Eosinophilic Pneumonia"} would be used to obtain target instances[3]. We construct queries for all values in S', obtain the union of their results, and then apply Song et al.'s approach over the resulting sample to obtain P_T.

Finally, we obtain P_{ST} by applying an instance-based schema matching technique [70] over P_S and P_T. We observed in the experiments that using this "focused" sampling based on $S' \subset S$ provides sufficient information to find P_{ST}, even using a small sample of 1%, resulting in good time performance.

4.3.2 Constructing Attribute Components

Next, we use information in the instance representation of source instance s and the computed comparable attributes P_{ST} to construct a discriminative query called attribute component, which finds matching instances in the target. The attribute components for s are directly derived from attributes in the pairs P_{ST}, the attributes $P(s, G_S)$ belonging to the source dataset, and their values. For every $p_S \in P(s, G_S)$, we find in P_{ST} the attribute p_T that is comparable to p_S. If exists, we construct the component $(x).(x, p_T, o_S)$, where x is a variable, p_T and o_S are constants, and $o_S \in O(s, p_S, G_S)$. Alg. 4 describes this procedure.

As an example, we have the source attributes $P_S = \{label, title\}$, which match all the attributes in $P_T = \{drugname, synonym\}$ such that $P_{ST} = \{\langle label, drugname\rangle, \langle label, synonym\rangle, \langle title, drugname\rangle, \langle title, synonym\rangle\}$. For the instance 12312, we have two attributes, $P(12312, Sider) = \{label, type\}$. The attributes comparable to *label* are the attributes *drugname* and *synonym*, and the value of the attribute *label* is "$Morphine$"; thus as attribute components, we have $(x).(x, drugname, \text{"}Morphine\text{"})$ and $(x).(x, synonym, \text{"}Morphine\text{"})$.

[3] We use OR queries in the experiment.

Algorithm 4 AttributeComponentQueries(S, G_S, G_T).

$P_S \leftarrow$ FindSourceDiscriminatePredicates(S, G_S)
$P_T \leftarrow$ FindTargetDiscriminatePredicates(S, G_S, G_T)
$P_{ST} \leftarrow$ Align(P_S, P_T)
$Q_S \leftarrow \emptyset$
for all $s \in S$ **do**
  $Q_s \leftarrow \emptyset$
  for all $< p_S, p_T > \in P_{ST} \wedge p_S \in P(s, G_s)$ **do**
    for all $o_S \in O(s, p_S, G_S)$ **do**
      $Q_s \leftarrow Q_s \cup (x).(x, p_T, o_S)$
    end for
  end for
  $Q_S \leftarrow Q_S \cup Q_s$
end for
return Q_S

Without loss of generality and mainly for presentation purpose, we assume in this chapter $|O(s, p_S, G_S)| = 1$. Then, the maximal number of attribute components that is formed for s, hence the space complexity of the procedure above, is bounded by $|P_{ST}| \times |M| \times |S|$, where M is the set of all query types (e.g., EXACT, OR). The number of queries for an instance s is equal to $|P_{ST}| \times |M|$ if $\forall \langle p_S, p_T \rangle \in P_{ST}, p_S \in P(s, G_S)$.

4.3.3 Learning Class Components

Given the class of interest Z, a class component is used in addition to an attribute component to prune candidates that do not belong to Z. It is of the form $(x).(x, p_T, o_T)$ where x is the only variable and p_T and o_T are constants derived from attributes and values in the target dataset. For instance, the class component $(x).(x, type, Drug)$ selects instances of type *Drug*. Combining this class component to an attribute query, e.g. $(x).(x, drugname, \text{“}Morphine\text{”}).(x, type, Drug)$, would retrieve instances of the type *Drug* with *drugname* “Morphine”. Class components might be constructed using attributes other than *type*, e.g. $(x).(x, affectedOrganism, \text{“}Mammals\text{”})$ selects the “class” of all those instances that have an attribute *affectedOrganism* with the value “Mammals”.

While attribute components are constructed prior to the actual candidate selection process, class components are learned during the process. This is because we infer the latter after a few sets of candidates have been retrieved, i.e. at a specific iteration i, we treat the candidate sets generated during all iterations previous to i as training data. This is the Query Refinement step illustrated in Fig. 4.1.

More precisely, an instance matcher is used to determine positive and negative matches among these candidates, which then serve as training examples. From these positive and negative examples, the best class components are computed, which are attribute value pairs that occur in all positive matches but in no negative ones. That is, class components $(x).(x, p_T, o_T)$ are constructed if $\forall r \in \mu(x), r \in \Delta^+$ and $r \notin \Delta^-$, where Δ^+ and Δ^- are the two sets of positive and negative matches, respectively, and $\mu(x)$ denotes the result bindings obtained for x when executing the class component as a query. For example, for $\Delta^+ = \{DB00295, DB00494\}$ and $\Delta^- = \{DB00001\}$, a class component constructed is the query $(x).(x, type, Drug)$ because, both $DB00295$ and $DB00494$ are results of that query while $DB00001$ is not.

Given the finite sets Δ^+ and Δ^-, to build class components, we obtain the set of attribute value pairs $\Lambda^+ = \{(p_T, o_T) | (x, p_T, o_T) \in I(x, G_T) \wedge x \in \Delta^+\}$ and $\Lambda^- = \{(p_T, o_T) | (x, p_T, o_T) \in I(x, G_T) \wedge x \in \Delta^-\}$. Then, we compute the difference $\Lambda = \Lambda^+ - \Lambda^-$ to capture target attribute value pairs that occur in the positive matches but in no negative ones. Finally, we use a greedy set-cover based algorithm [80] to select the minimum set $\Lambda^m \in 2^\Lambda$ that covers all positive matches in Δ^+. Each $(p_T, o_T) \in \Lambda^m$ represents a class component $(x).(x, p_T, o_T)$.

To compute Λ^m, the attributes values pairs in Λ are ordered by their decreasing frequency in the positive matches in Δ^+. Then, the first element in this sorted list is removed from Λ, added to Λ^m and the positive matches that it covers are removed from Δ^+. This process is repeated continuously until $\Delta^+ = \emptyset$ or all elements in Λ are added to Λ^m. Particularly, we only consider elements in Λ that occur in more than one positive match, which avoids generating a class component that occurs only for very little percentage of instances. In practice, this reduces Λ to a smaller set of attribute values (a few dozens); consequently, allowing Λ^m to be computed quite efficiently. This computation is bound to $O(|\Lambda| \times |\Delta^+|)$ because we need to pass through all Δ^+ matches to compute the frequency of each element in Λ.

The worst case number of class components is $|\Lambda^m| = |\Lambda|$. However, the number of actual components that cover all examples is usually much smaller, i.e. $|\Lambda^m| \ll |\Lambda|$, limiting the number of queries that have to be issued. In the current example, this algorithm would select for Λ^m only one out of the seven attribute pairs in Λ, i.e., $\Lambda^m = \{(type, Drug)\}$.

4.4 Executing Optimal Queries

Using the method described in the previous section, a set of queries Q_s is learned for each instance $s \in S$. The maximum number of queries in Q_s corresponds to all combinations

of attribute components, class components and query types, i.e. $|Q_s| = (|P_{ST}| \times |M|) \cdot |\Lambda|$. In this section, from the total search space of $|Q_S|$ queries, which contains one set Q_s for every $s \in S$, we propose to find and, whenever possible, execute only those that are optimal, $Q_s^* \subseteq Q_s$, for every $s \in S$. We consider a query as *optimal*[4], $q_s^* \in Q_s^*$, if it retrieves only correct matches for s (*quality-optimal*), and also yields the fastest execution time compared to the other queries $q \in Q_s$ (*time-optimal*). Note that the quality criterion here only captures precision, as q_s^* is not required to retrieve all correct matches. We focus on *precision-oriented* queries, while recall is accounted for by the use of several queries in Q_s^* (discussed later).

Clearly, this notion of optimality implies that an optimal query can only be determined after all queries in Q_s are evaluated and their results are inspected for correctness and execution time. This is expensive in the on-the-fly setting, and the actual purpose of the optimization proposed here is in fact to avoid executing and evaluating all queries. To this end, we propose the use of estimates instead of the actual observed quantity for time and correctness for selecting optimal queries.

In this section, we firstly discuss how to compute these estimates, then elaborate on how they can be used to determine optimal queries for a given instance s, and finally, present the whole iterative process that finds and executes queries for all instances $s \in S$.

4.4.1 Estimating Metrics for Query Optimality

For every $s \in S$, we propose to obtain estimates for the performance, i.e. cardinality and execution time, of the queries Q_s using the performance of similar queries previously evaluated for some other instances in S. We introduce the notion of query pattern to capture similar queries:

Definition 4.6 (Query Pattern)**.** A query pattern of a query q is defined as $C(q) = (type, cc, p)$, where *type* is the query type (EXACT, OR etc.), *cc* is its class component and p is the attribute of its attribute component.

Intuitively, a query pattern captures elements of a query that do not vary for some instances. For example, the query pattern for the query $q = (x).(x, drugname, \text{“}Morphine\text{”})$ is $C(q) = (EXACT, null, drugname)$. Based on this notion, we consider queries as similar when they share the same pattern. Based on similar queries, estimates for a given query can obtained as follows:

[4]Optimal here is not use in the theoretical sense but simply to denote maximality w.r.t. our objectives of precision and time; and later defined as maximality w.r.t. our heuristics.

Definition 4.7 (Estimated Execution Time)**.** Given the time of q, $Time(q)$, and a set of previously evaluated queries Q and $\forall x \in Q, C(x) = C(q)$, the estimated execution time of q is $Time(q) = \frac{\sum_{x \in Q} Time(x)}{|Q|}$.

Definition 4.8 (Estimated Cardinality)**.** Given the cardinality of q, $Card(q)$, and a set of previously evaluated queries Q and $\forall x \in Q, C(x) = C(q)$, the estimated cardinality of q is $Card(q) = \frac{\sum_{x \in Q} Card(x)}{|Q|}$.

In order to assess the *time* aspect of optimality introduced above, we simply use the estimated time. Further, since there might be not one but a set of optimal queries Q_s^*, minimizing the number of queries needed for an instance s is another mean to improve the time efficiency of the overall process. To optimize for the *quality* aspect of optimality, we propose the use of estimated cardinality as a heuristic to reflect the correctness of query results, because queries that retrieve less instances include less incorrect matches. This aggressive heuristic geared towards precision is designed to support our precision-oriented quality definition. Putting these together results in the following optimization problem that has three goals, namely evaluating (1) a minimum number of queries for every instance, which according to estimations, have (2) fastest total execution time and (3) lowest total cardinality:

Definition 4.9 (Optimization Goal)**.** Let S be the set of all instances and Q_s be the set of queries evaluated for $s \in S$, and $Card(q)$ and $Time(q)$ the estimated cardinality and execution time of q, respectively, then the optimization goals are

$$(1)\ \min \sum_{s \in S} |Q_s|, (2)\ \min \sum_{q \in Q_s} Time(q), (3)\ \min \sum_{q \in Q_s} Card(q)$$

subject to $\sum_{s \in S} |Q_s| > 0$ and $\sum_{q \in Q_s} Card(q) > 0$.

4.4.2 Optimal Queries for One Instance

Given one instance s and the queries Q_s, we aim to achieve the optimization goals above by selecting fast queries with low cardinality to execute, while skipping other queries. Note that achieving these goals does not guarantee that the executed queries are strictly optimal w.r.t. both time and quality as introduced before. The proposed optimization goals account for the fact that since these two aspects are not always complementary, queries that strictly satisfy both may not exist, i.e. there might be no queries that are both the fastest and have lowest cardinality. The goals (2+3) can be reached simply by minimizing the total time and cardinality of all the executed queries. In the following, we use "optimal" simply to express that queries are optimal w.r.t. these relaxed optimization goals.

To identify fast queries with low cardinality, we use the estimates above as heuristics to induce an ordering of queries:

Definition 4.10 (Time-Based Heuristic)**.** Given two queries $q_i, q_j \in Q_s$, q_i is more time-optimal than q_j if $Time(q_i) < Time(q_j)$.

Definition 4.11 (Cardinality-Based Heuristic)**.** Given two queries $q_i, q_j \in Q_s$, q_i is more cardinality-optimal than q_j if $Card(q_i) < Card(q_j)$.

Now it is straightforward to verify that by executing queries according to either of these two induced orders, we optimize either w.r.t. goal (2) or goal (3). Since these goals are not complementary, we choose in the experiment to use time for ordering queries to optimize towards goal (3) first, i.e. execute the fastest queries first. To tackle goals (1+2), we skip queries in this time-ordered list that are redundant to or have a higher cardinality than the previous query, using the cardinality-based heuristic and the following one:

Definition 4.12 (Redundancy-based Heuristic)**.** Given the ordered list of queries Q_s, a query $q_i \in Q_s$ should be evaluated only if its results do not overlap with the ones of the preceding query $q_{i-1} \in Q_s$, i.e. when $\mu(q_{i-1}) \cap \mu(q_i) = \emptyset \wedge \mu(q_{i-1}) \neq \emptyset \wedge \mu(q_i) \neq \emptyset$.

Note that our quality metric is precision-oriented while for increasing recall, we employ several queries. However, increasing recall is only possible by adding queries that retrieve results not already captured by the queries in the current set. The intuition behind this heuristic is to focus only on queries that produce new results. If the results of q_i and q_{i-1} overlap, then either of the two is redundant. However, implementing this heuristic actually requires evaluating both queries to identify if their result overlap, while the aim is to skip the second one.

Learning to Skip Queries. We propose the use of a simple generative classifier [81, 82] to achieve this skipping. We use a Naive Bayes classifier [83] for this in the experiment (Alg. 5, 7 and 8). This classifier learns to predict if a query q_i should be evaluated or not. According to the proposed heuristics, this should be done when q_i and its previous query q_{i-1} satisfy the condition $Card(q_{i-1}) > Card(q_i) \vee q_{i-1} \cap q_i = \emptyset$ (Alg. 7, lines 7-8). This condition is treated as a class label, whereas training data points are derived from the pairs of consecutive queries (q_{j-1}, q_j), which have been executed in previous runs. The features of these data points include the query patterns $C(q_{j-1})$ and $C(q_j)$, indicating the types of queries (q_{j-1}, q_j) belong too. As a result, at prediction time (Alg. 8, line 3), the classifier yields true for a given data point formed by q_{i-1} and q_i when this condition was satisfied for a representative number of evaluated query pairs that exhibit the same pattern, i.e. when $C(q_{i-1}) = C(q_{j-1}) \wedge C(q_i) = C(q_j)$ for a large number of previous pairs (q_{j-1}, q_j).

Candidate Selection Procedure for One Instance. Putting these together, the process for finding and executing optimal queries for an instance s is as follows. First, we sort all queries Q_s according to their estimated time. Then, we partition the queries into subsets $Q_s(p_t) \subseteq Q_s$, each comprising queries that have the same attribute p in their attribute component, but we preserve Q_s. We execute the queries in Q_s according to the sorting order and skip some redundant queries during this execution. This process is repeated until reaching a *termination condition*, i.e. when the query $q_s^* \in Q_s(p_t)$ with the lowest possible observed cardinality has been reached, i.e. observed $card(q_s) = 1$, or when all queries $q_s \in Q_s$ have been evaluated (Alg. 7 and Alg. 8, line 3) Note that as we use cardinality as a heuristic for quality, q_s^+ can be seen as the most *quality-optimal* query. While many more queries might be executed during the process, results are only taken into account for this most quality-optimal query. The result for each such $q_s^+ \in Q_s(p_t)$ are finally aggregated over all the subsets $Q_s(p_t) \subseteq Q_s$ to obtain the candidate set for s (Alg. 6, 7, and 8, lines 7, 12, 8; respectively). In summary, while this process does not guarantee that only optimal queries are executed, it helps to prioritize (time-)optimal queries during the execution and to terminate early when the most quality-optimal queries q_s^+ have been found. Note we use q_s^+ instead of q_s^* to make clear that q_s^+ is only quality-optimal w.r.t. our heuristics, and also, due to possible conflict between time and quality, it cannot be guaranteed to be time-optimal.

Precision vs. Recall We use a precision-oriented quality measure and a corresponding heuristic that prefers low cardinality queries to achieve this. Low cardinality queries retrieve fewer instances, hence include fewer incorrect matches but also, might fail to produce some correct matches. However, as we use not one but several queries for every instance, correct matches missed by one query are often accounted for by the other queries. Aiming at higher recall, we prioritize queries that complement each other. This is supported by the redundancy-based heuristic, as well as the grouping of queries in the complementary sets $Q_s(p_t) \subseteq Q_s$ discussed above. This grouping results in sets of queries with different attribute components, which select different sets of candidates, hence increasing the coverage of possible candidates.

4.4.3 Optimization Process for All Instances

As discussed previously, computing the metrics to estimate optimality as well as learning to skip queries required information from "some" previous runs. Here, we describe the iterative process in detail, where each iteration is concerned with processing one particular instance, and previous runs refer to the previous iterations that have been performed for different instances.

Algorithm 5 CandidateSelection(S, G_S, G_T, β, n).

1: $Q_S \leftarrow$ AttributeComponentQueries(S, G_S, G_T)
2: $\Gamma \leftarrow$ NaiveBayesClassifier.new()
3: $(T_S, Q_S, S) \leftarrow$ SortingPhase(S, Q_S, β)
4: $(\Delta^+, \Delta^-) \leftarrow$ matcher(T_S)
5: $K \leftarrow$ ClassComponentsQueries(Δ^+, Δ^-)
6: $Q_S \leftarrow$ UpdateQueries(Q_S, K)
7: $(T, S) \leftarrow$ LearningPhase(S, Q_S, Γ, n)
8: $T_S \leftarrow T_S \cup T$
9: $T_S \leftarrow T_S \cup$ PredictingPhase(S, Q_S, Γ)
10: **return** T_S

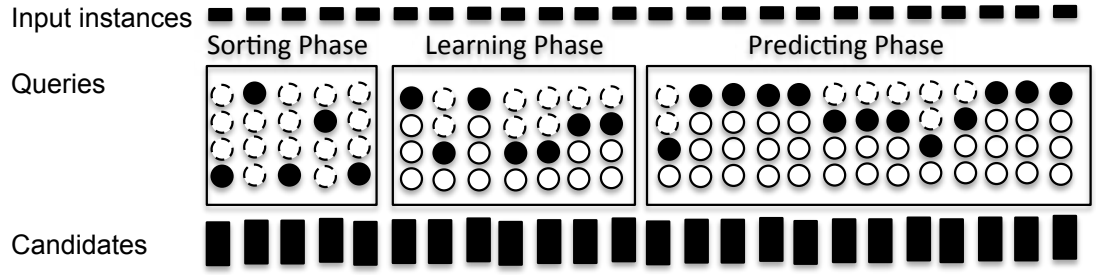

FIGURE 4.2: All queries are evaluated in the Sorting phase (black and dashed circles stand for optimal and "unnecessary" queries, respectively), while fewer queries are evaluated in the Learning and Predicting phases (white circles denote unevaluated queries).

The overall procedure is presented Alg. 5, which has three distinct phases applied to different subsets of source instances, namely *Sorting*, *Learning* and *Predicting* phases. Fig. 4.2 illustrates these three phases as well as the overall process. Clearly, these phases differ in the number of queries evaluated for each instance because in the beginning, the optimization is not effective as information necessary for estimation and learning have to be acquired first.

Sorting. This phase covers β% of the instances (1% in the experiment, denoted by B in Alg. 6). During this phase, all queries are evaluated to obtain the time and cardinality estimations. That is, for every s covered in this phase, queries are randomly selected and executed, and no skipping of queries is possible. At the end of this phase, the time-order of the queries is determined.

Learning. During both the Learning (see Alg. 7) and Predicting phases, for every instance s and every $Q_s(p_t) \subseteq Q_s$, queries are executed according to the determined time-order until the termination condition is reached, i.e. until q_s^* with the lowest possible observed cardinality has been reached, i.e. observed $card(q) = 1$, or when all queries in Q_s have been evaluated. Information collected previously as well as during this phase is exploited to learn the classifier for skipping queries as discussed in the previous section.

Algorithm 6 SortingPhase(S, Q_S, β).

1: $B \leftarrow$ sample(β, S)
2: **for all** $s \in B$ **do**
3: **for all** $q_i \in Q_s \in Q_S$ **do**
4: $eval(q_i)$
5: **end for**
6: $S \leftarrow S - s$
7: **for all** $Q_s(p_t) \subseteq Q_s$ **do**
8: $T_s \leftarrow T_s \cup \mu(q_s^+ \in Q_s(p_t))$
9: **end for**
10: **end for**
11: $Q_S \leftarrow$ SortByTimeEstimation(Q_S)
12: **return** T_s, Q_S, S

The learning is performed until convergence, i.e. until the prediction becomes more stable as more and more data is exploited during the process (Alg. 6, line 9). In particular, we test the prediction quality to stop this process when prediction errors do not change for n instances (3 in the experiment).

Algorithm 7 LearningPhase(S, Q_S, Γ,n).

1: **for all** $s \in S$ **do**
2: **for all** $q_i \in Q_s \in Q_S$ **do**
3: **if** $\exists q \in Q_s(p_t) \wedge q_i \in Q_s(p_t) \wedge Card(q) = 1$ **then**
4: next
5: **end if**
6: $eval(q_i)$
7: $Y \leftarrow Card(q_{i-1}) > Card(q_i) \vee q_{i-1} \cap q_i = \emptyset$
8: $\Gamma.addExample(datapoint, Y)$
9: $S \leftarrow S - s$
10: stop if $\Gamma.converge(n)$
11: **end for**
12: **for all** $Q_s(p_t) \subseteq Q_s$ **do**
13: $T_s \leftarrow T_s \cup \mu(q_s^+ \in Q_s(p_t))$
14: **end for**
15: **end for**
16: **return** T_s, S

As an additional step, queries are updated with class components before the Learning phase (Alg. 5, lines 4-6). To obtain the examples Δ^+ and Δ^-, the candidates obtained so far serve as input to an instance matcher that outputs the examples. Then, using Δ^+ and Δ^-, the class components are learned, as discussed in Section 3.3, and the queries are updated accordingly.

Predicting. Finally, the classifier learned in the previous phase is used to skip queries in this phase (see Alg. 8), i.e. given q_{i-1} and q_i, it executes q_i when the classifier yields true for the pair (q_{i-1}, q_i).

Algorithm 8 PredictingPhase(S, Q_S, Γ).

for all $s \in S$ **do**
  for all $q_i \in Q_s \in Q_S$ **do**
    if $(\exists q \in Q_s(p_t) \wedge q_i \in Q_s(p_t) \wedge Card(q) = 1) \vee \Gamma.predict(datapoint) = false$ **then**
      next
    end if
    $eval(q_i)$
  end for
  for all $Q_s(p_t) \subseteq Q_s$ **do**
    $T_s \leftarrow T_s \cup \mu(q_s^+ \in Q_s(p_t))$
  end for
end for
return T_s

4.5 Evaluation

Currently, there exist no solutions that can be directly applied to our on-the-fly integration problem. To compare Sonda, we designed two best-effort non-trivial baselines. They are based on S-based and S-agnostic, two recent candidate selection approaches we have adapted to the on-the-fly setting. Although we will refer to those baselines as S-based and S-agnostic, improvements reported in this chapter do not refer to the original systems (which cannot be directly compared to Sonda) but the baselines.

Summarizing the experiments discussed in detail below, Sonda took 11 minutes (37m, with real Web endpoints) and achieved an average effectiveness of 85%, measured by F1. The best baseline, S-based, took 14m (82m, Web endpoints) and achieved 73% F1. To consider the effect of candidate selection on instance matching, we run SERIMI on top of Sonda's candidates and compared the results with those reported for the OAEI benchmark. As an average over all datasets, the best system was Sonda+SERIMI, and it resulted in 13% F1 improvement over SERIMI, indicating that Sonda effectively preserved the correct candidates and also reduced ambiguity (incorrect candidates), helping the matcher to achieve higher quality results.

Datasets. We relied on the datasets and ground truth published by OAEI [84]. We used the life science (LS) collection (which includes Sider, Drugbank, Dailymed TCM, and Diseasome) and the Person-Restaurant (PR) from the 2010 collection and all datasets from the 2011 collection. The matching tasks are cross-dataset tasks, which always involve a pair of datasets. One is the source while the other is treated as the target.

Metrics. For assessing candidate selection results, we employed standard metrics, namely Reduction Ratio (RR), Pair-wise Completeness (PC) and F1. Basically, high RR means that the candidate selection algorithm helps to focus on a smaller number of

candidates, while high PC means that it preserves more of the correct candidates. More precisely, RR captures the reduction in the number of all possible candidate pairs that have to be considered for matching. A normalized version of RR can be used, when the number of all possible candidate pairs is large [38]. We also use normalization, where instead of considering the reduction in the number of candidate pairs, we consider the reduction in the number of candidates. Besides these metrics, we also count the average number of queries evaluated per instance as well as the time needed. For assessing the instance matching results, we used the standard metrics Precision, Recall and F1.

Systems. Our system Sonda and the results of this experiments are available for download[5] at GitHub. It was implemented in Ruby and the queries were implemented as SPARQL queries issued over remote SPARQL endpoints. For the OAEI datasets, we could find a SPARQL endpoint[6] on the Web, which serves DBpedia (the largest out of all given datasets). This endpoint runs the OpenLink Virtuoso Universal Server version 06.04.3132 on Linux, using 4 server processes. For all other datasets, we employed the OpenLink Virtuoso Universal Server Version 6.1.5.3127 as a SPARQL endpoint, and run it on a server in our controlled environment with Intel Core 2 Duo, 2.4 GHz, 4 GB RAM, using a FUJITSU MHZ2250BH FFS G1 248 GB hard disk. We load these datasets into Virtuoso, creating the default S-P-O index and an inverted index as supported by Virtuoso that was used to support LIKE, AND and OR queries.

We used two configurations. In the *Web* configuration, the DBpedia endpoint on the Web is used while in the *controlled* one, all datasets are managed using our controlled endpoint. The presented values are averages over five runs. For performance reasons, remote data endpoints stop processing according to a manually set query timeout. The one we used supports a query limit. A *query limit* of 100 for instance indicates that the endpoint should stop processing after 100 number of results have been retrieved. To evaluate the effect of *class components*, we considered two versions of Sonda, namely without (Sonda-A) and with class components (Sonda-C).

For comparison, we modified the *S-agnostic* [50] and *S-based* [38] approaches by translating their schemes to queries that are processed against endpoints. S-agnostic's scheme consists of all value tokens while S-based uses values of discriminative attributes. Accordingly, OR queries are created to consider all value tokens used by S-agnostic. These queries are executed sequentially and their results are aggregated to produce the candidate set. As discussed, our approach and S-based use discriminative and comparable attributes for candidate selection. To achieve this in the on-the-fly setting, we use the sampling procedure presented in Section 3.1 for both approaches. S-based applies an

[5] https://github.com/samuraraujo/Sonda
[6] http://dbpedia.org/sparql

additional similarity function to further prune incorrect candidates retrieved from these queries. For comparison purposes, we apply this strategy to all approaches, using the same similarity function.

In summary, S-agnostic uses only value tokens while S-based additionally, employs attributes (focusing on discriminative ones). Sonda-A extends S-based, considering 4 more query types and furthermore, implements the heuristic-based search optimization. Sonda-C extends Sonda-A with class components.

TABLE 4.1: Results of the three systems over all pairs of datasets, where Queries denotes the total number of queries issued by the system, Queries/Instance (Q/I) denotes the amount of queries evaluated per instance, and Learning(s) and Search(s) stands for the time needed for learning queries and executing them, respectively.

Task	Systems	Queries	Q/I	Learning(s)	Search(s)	Overall Time (s) (Controlled/Web)	RR(%)	PC(%)	F1(%)
1 DAILYMED-SIDER	Sonda-A	40	1.63	27	**220**	**248**	**100.0**	97.86	98.92
	Sonda-C	40	1.88	30	228	258	**100.0**	**98.43**	**99.21**
	S-based	8	8.0	25	1158	1183	99.37	97.99	98.67
	S-agnostic	4	4.0	**15**	638	653	99.37	97.99	98.67
2 DISEASOME-SIDER	Sonda-A	20	2.77	8	16	24	**100.0**	**94.77**	**97.31**
	Sonda-C	20	2.47	9	**14**	**23**	**100.0**	**94.77**	**97.31**
	S-based	4	4.0	6	50	56	99.38	92.44	95.78
	S-agnostic	2	2.0	**2**	27	29	**100.0**	91.86	95.76
3 DRUGBANK-SIDER	Sonda-A	40	1.49	57	**36**	**93**	**100.0**	**99.65**	**99.82**
	Sonda-C	80	7.16	57	155	212	**100.0**	98.23	99.11
	S-based	8	8.0	51	265	316	95.92	**99.65**	97.75
	S-agnostic	29	29.0	**25**	249	274	95.59	**99.65**	97.58
4 NYTIMES-DBPEDIA(CORP)	Sonda-A	15	3.28	204	1907	2111 / (5498)	47.6	**91.3**	62.57
	Sonda-C	30	3.33	160	4116	4276 / (2621)	**87.63**	88.96	**88.29**
	S-based	3	3.0	107	1809	1916 / (5362)	41.04	56.39	47.51
	S-agnostic	4	4.0	**18**	**1772**	**1790** / (**1864**)	13.93	33.54	19.68
5 NYTIMES-DBPEDIA(GEO)	Sonda-A	15	3.2	320	2813	3133 / (2640)	22.76	**81.09**	35.54
	Sonda-C	15	1.33	143	**522**	**665** / (3386)	**62.52**	75.52	**68.41**
	S-based	3	3.0	527	2288	2815 / (10145)	22.72	42.66	29.65
	S-agnostic	5	5.0	**10**	2770	2780 / (**2047**)	6.26	31.35	10.44
6 NYTIMES-FREEBASE(CORP)	Sonda-A	15	1.68	38	**490**	**527**	98.88	87.88	93.06
	Sonda-C	45	2.92	43	641	684	**99.24**	**89.85**	**94.31**
	S-based	3	3.0	24	1262	1286	86.6	69.97	77.4
	S-agnostic	4	4.0	**13**	588	601	81.1	42.35	55.64
7 NYTIMES-FREEBASE(GEO)	Sonda-A	15	2.57	38	621	659	**97.62**	87.71	92.4
	Sonda-C	15	2.29	36	597	633	97.56	**88.13**	**92.6**
	S-based	3	3.0	31	866	897	79.88	56.51	66.19
	S-agnostic	5	5.0	**12**	**499**	**511**	78.11	45.78	57.73
8 NYTIMES-FREEBASE(PER)	Sonda-A	15	2.1	42	**1154**	**1196**	**95.49**	95.58	**95.54**
	Sonda-C	30	3.09	44	1349	1393	93.7	**96.04**	94.86
	S-based	3	3.0	38	3181	3219	93.63	73.55	82.39
	S-agnostic	4	4.0	**25**	1217	1242	95.32	29.54	45.11
9 NYTIMES-GEONAMES	Sonda-A	15	3.33	130	1826	1956	16.19	**90.5**	**27.47**
	Sonda-C	15	3.33	128	2025	2153	16.19	**90.5**	**27.47**
	S-based	3	3.0	39	1194	1233	27.54	23.59	25.41
	S-agnostic	5	5.0	**11**	**552**	**563**	**28.45**	19.62	23.23

Task	Systems	Queries	Q/I	Learning(s)	Search(s)	Overall Time (s) (Controlled/Web)	RR(%)	PC(%)	F1(%)
10 PERSON11-PERSON12	Sonda-A	15	1.27	11	**27**	**38**	**99.57**	95.63	**97.56**
	Sonda-C	15	1.26	11	27	38	**99.57**	95.63	**97.56**
	S-based	3	3.0	9	63	72	78.68	96.25	86.58
	S-agnostic	4	4.0	**4**	68	73	30.98	**100.0**	47.3
11 PERSON21-PERSON22	Sonda-A	20	7.16	6	20	26	**44.59**	58.47	50.6
	Sonda-C	20	6.24	6	18	24	44.3	59.32	**50.72**
	S-based	4	4.0	4	16	18	25.19	87.29	39.09
	S-agnostic	4	4.0	**0.5**	**11**	**12**	21.88	**100.0**	35.9
12 RESTAURANT1-RESTAURANT2	Sonda-A	10	1.83	5	8	14	**98.25**	**98.23**	**98.24**
	Sonda-C	10	1.82	5	**8**	**13**	**98.25**	**98.23**	**98.24**
	S-based	2	2.0	3	19	22	94.02	96.46	95.22
	S-agnostic	3	3.0	**0.7**	18	19	94.02	96.46	95.22
13 SIDER-DAILYMED	Sonda-A	35	3.88	177	**140**	**318**	34.96	64.67	45.38
	Sonda-C	105	11.55	180	315	495	**40.33**	57.07	47.26
	S-based	7	7.0	174	583	758	25.05	**83.83**	38.57
	S-agnostic	3	3.0	**33**	7781	7814	36.02	75.63	**48.8**
14 SIDER-DBPEDIA(DRUGS)	Sonda-A	10	1.52	46	**113**	**159** / **(317)**	98.06	92.1	94.99
	Sonda-C	30	2.28	41	199	240 / (811)	**99.72**	**94.69**	**97.14**
	S-based	2	2.0	39	146	185 / (515)	77.56	93.04	84.6
	S-agnostic	3	3.0	**35**	156	191 / (1122)	69.18	93.19	79.41
15 SIDER-DBPEDIA(SIDEEFFECT)	Sonda-A	20	1.58	33	**303**	**337** / **(582)**	76.69	89.29	82.51
	Sonda-C	40	4.66	21	508	530 / (2002)	**94.44**	**90.58**	**92.47**
	S-based	4	4.0	24	543	567 / (3629)	77.42	71.74	74.47
	S-agnostic	3	3.0	**5**	401	405 / (2236)	56.85	65.42	60.83
16 SIDER-DISEASOME	Sonda-A	20	4.52	10	**43**	53	83.33	**94.77**	88.68
	Sonda-C	20	4.53	10	47	57	83.33	**94.77**	88.68
	S-based	4	4.0	8	92	99	86.19	91.86	88.93
	S-agnostic	3	3.0	**1**	47	**48**	**90.17**	88.95	**89.56**
17 SIDER-DRUGBANK	Sonda-A	25	3.59	53	270	322	95.67	98.02	96.83
	Sonda-C	25	2.38	55	**154**	**209**	**98.25**	97.44	**97.84**
	S-based	5	5.0	50	471	521	94.22	**99.3**	96.69
	S-agnostic	3	3.0	**41**	338	379	97.11	97.79	97.45
18 SIDER-TCM	Sonda-A	10	1.68	8	14	22	**100.0**	**98.25**	**99.12**
	Sonda-C	10	1.61	8	**12**	**20**	**100.0**	**98.25**	**99.12**
	S-based	2	2.0	5	23	28	**100.0**	95.32	97.6
	S-agnostic	3	3.0	**1**	24	25	**100.0**	95.32	97.6
Average	Sonda-A	19.72	2.73	67	**557**	**624** / (2261)	78.31	**89.77**	80.92
	Sonda-C	31.39	3.56	55	608	662 / (2205)	**84.17**	89.25	**85.03**
	S-based	3.94	3.94	65	779	844 / (4905)	72.47	79.32	73.47
	S-agnostic	5.06	5.06	**14**	953	967 / (**1793**)	66.35	72.47	64.22

4.5.1 Candidate Selection Results

Table 1 shows an overview of the results. Compared to the baseline approaches over all 18 matching tasks, Sonda-A and Sonda-C could improve F1 score in 16 and 17 of the tasks, respectively. Average F1 values for Sonda-A, Sonda-C, S-based and S-agnostic are 80% and 85%, 73% and 64%, respectively. This translates into a 14% improvement that Sonda-C could achieve over the best baseline, S-based. Average time performance of Sonda-A, Sonda-C, S-based and S-agnostic are 10m, 11m, 14m and 16m, respectively. Thus, Sonda-A and Sonda-C were 34% and 22% faster than the fastest baseline, S-based, respectively. Since higher quality results often require more processing time, we also look at time performance results in the light of result quality. In particular, we look at the matching tasks for which the results quality were comparable among the systems (when differences in PC and F1 were $< 5\%$). For these tasks, Sonda-A and Sonda-C were over 45% faster than the fastest baseline, S-agnostic.

Task Complexity. Differences in F1 values obtained for different tasks indicate their varying levels of complexity. Sonda-A and Sonda-C consistently outperformed the baselines over all tasks (with one exception, task 16, where results were comparable).

Large improvements could be achieved for tasks 4-8, especially the two tasks that involved DBpedia. These tasks involve large datasets and thus, capture a larger amount of possible candidates that have to be considered for every instance. Sonda was more effective in dealing with this ambiguity. In particular, it was more effective both in finding the correct candidates and reducing the number of candidates as indicated by average PC and RR, respectively. Higher PC could be achieved because more query types were considered, thus incorporating a larger space of candidates. This however, does not come at the expense of RR. While S-based and S-agnostic use all their query results as candidates, Sonda selectively chooses the best queries and utilizes only their results as the candidate set.

There are 4 problematic tasks where F1 values were < 0.7 (tasks 5, 9, 11 and 13). Particularly difficult was task 9, which involves Geo Names. This dataset contains many instances with the same labels with only few additional information to disambiguate them. The strategies used by OAEI matching systems to deal with this task is to manually encode and exploit geo- and location-specific knowledge in the form of rules (which were not used by our systems). Task 11 involves an artificial dataset where syntax mistakes were added to produce string level ambiguity. Sonda's PC values (58.47 and 59.32, respectively) were lower than those achieved by the baselines. Here we can clearly see the strategy of aggregating all queries results works well, while the heuristics

used by Sonda's to choose only the best ones may compromise PC. However, it has a positive effect on RR, resulting in higher F1 also for this task.

Attribute Components. On average, systems using attribute components, Sonda-C, Sonda-A and S-based, are more effective than S-agnostic, which dismisses attribute information and used value tokens only. In terms of F1, their values are 85%, 81% and 73%, respectively, compared to 64%.

Class Components. The effect of the class component can be seen in differences between Sonda-C and Sonda-A. The former achieved a higher RR higher and comparable PC, thus indicating the class component has the positive effect of reducing the number of incorrect candidates. Especially for tasks 4 and 5 that involve DBpedia, improvements in RR were large (from 47.6 to 87.63 and 22.76 to 62.52, respectively). The class component has a stronger effect here because this dataset simply captures more candidate results, thus there is potentially also a higher number of incorrect results that could be pruned.

Processing Cost. As captured by Table 1, overall processing cost can be decomposed into learning and execution times. While learning is essential to produce the queries capturing different candidates, execution is needed to retrieve them. Thus, both steps are crucial for result quality. However, while the baselines execute all the learned queries, Sonda does not. This has a large impact on performance as we can see that learning is relatively cheap, which makes up only 7% of total time, on average. Due to the sampling we performed, the number of queries needed to retrieve data during learning is substantially smaller. Although Sonda-A and Sonda-C were 4.7x and 3.8x slower than S-agnostic during learning, the fastest approach that simply maps value tokens to queries, they were 41% and 36% faster than S-agnostic when considering the whole process. Thus, the results show that although Sonda invested more time to learn the queries (which are needed to achieve the better results), the optimization could reduce the time in executing the queries. While we focus the discussion on times achieved for the controlled configuration because values were more stable, Table 1 also shows the differences between the controlled and Web configurations. Compared to its controlled version, Sonda-A, Sonda-C, S-based and S-agnostic were 3.6x, 3.3x, 5.8x and 1.9x slower, respectively. This suggests that delays caused by the external DBpedia endpoint have the largest negative impact on S-based. Accordingly, the performance improvement Sonda could achieve over S-based is larger in the Web setting. S-agnostic yields best time performance here because as opposed to the other systems, it does not require learning and thus, does not have retrieve data samples from the Web endpoints.

Number of Queries. This connection between cost and quality can be more clearly seen in the number of queries and F1. Sonda-C, Sonda-A, S-based and S-agnostic considered (during learning) on average 31, 20, 4 and 5 queries, which translates to F1 values

of 85%, 81%, 73% and 64%, respectively. In all cases, Sonda achieves a considerable reduction in the number of queries evaluated per instance (during execution). In some cases (e.g. tasks 10, 14 and 18), it performed close to one query per instance.

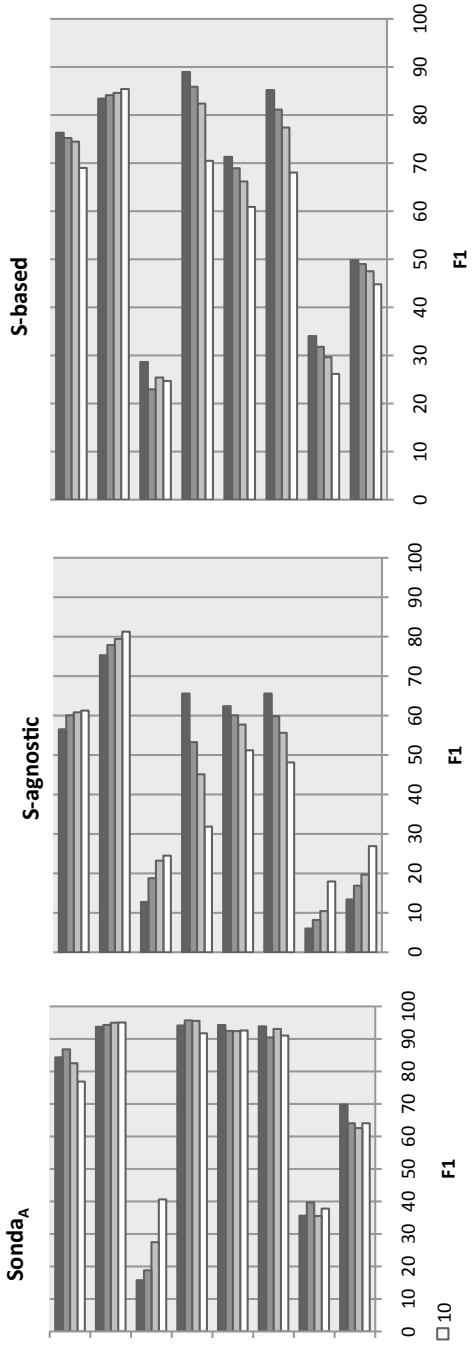

FIGURE 4.3: F1 for Sonda-A, S-agnostic and S-based for query limits 10, 30, 50 and 100.

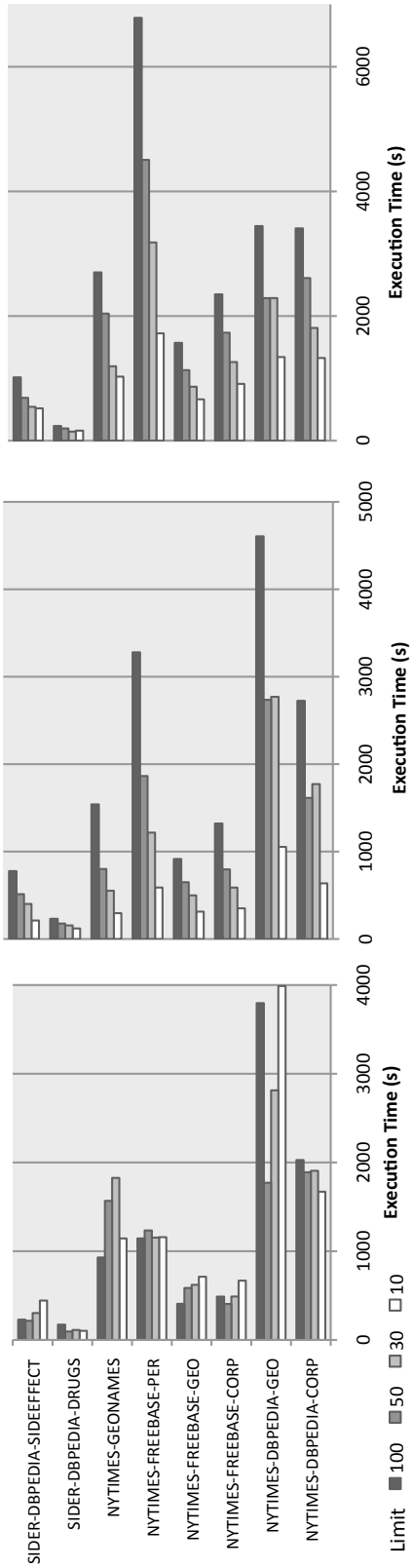

FIGURE 4.4: Execution time for Sonda-A, S-agnostic and S-based for query limits 10, 30, 50 and 100.

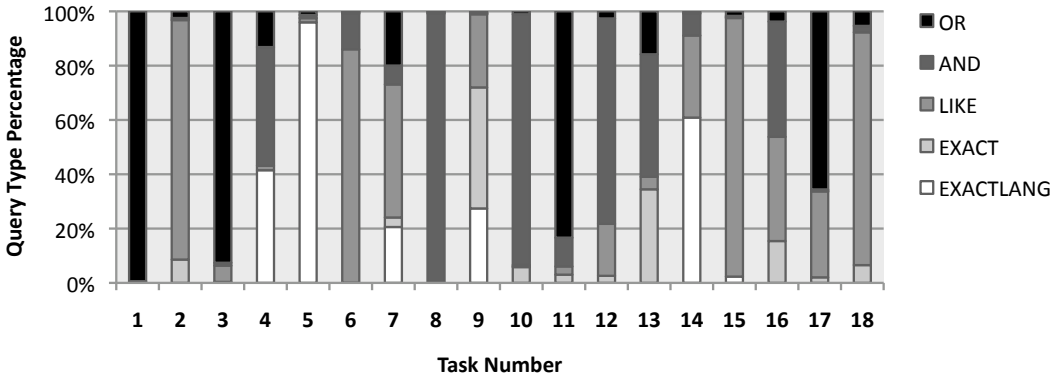

FIGURE 4.5: Percentages of query types executed by Sonda-A per task.

Number of Results. Also, quality (F1) is related to the number of results retrieved by the queries. Fig. 4.3 and Fig. 4.4 show the effect of query limit on Sonda-A, S-agnostic and S-based. Four query limits were used: 10, 30, 50, 100 elements. We can see that only F1 values for S-based improved consistently with increasing query limit, while execution times for both S-based and S-agnostic were higher with increasing query limit. This effect on time however, could not be observed for Sonda-A because due to the optimization, a small value for query limit sometimes resulted in a greater number of queries that have to be executed. We observed that while PC consistently improved with increasing limit (more results are incorporated), RR sometimes got worse with increase limit (because more results also include more negative matches). Thus, increasing query limit has a mixed impact on F1, while for the baselines, it unambiguously resulted in higher processing cost.

Query Types. Fig. 4.5 shows for each task the percentage of query types executed to find the optimal candidate set. It illustrates that to produce non-empty candidate sets, all query types were considered useful by Sonda-A.

4.5.2 Instance Matching Results

To produce final instance matches for the OAEI matching tasks, we refined the candidates produced by Sonda using SERIMI [7] (Sonda+SERIMI). Table 2 shows the OAEI 2010 results for Sonda+SERIMI, SERIMI (without Sonda) and the other systems, RIMON [65] and ObjectCoref [86]. When considering only datasets supported by ObjectCoref, we can see that ObjectCoref was second best (see Average-ObjectCoref), while over the datasets used by Rimon, SERIMI was second best (see Average-Rimon). Sonda+SERIMI was best over all three combinations of datasets. As an average, Sonda+SERIMI resulted in 13% average improvement over SERIMI. Thus, these results

[7] The SERIMI' version considered on this experiment is the one published at [85]

TABLE 4.2: Sonda+SERIMI compared to other OAEI 2010 published results.

Task	Sonda+SERIMI	SERIMI	ObjectCoref	Rimon
Sider-Dailymed	0.63	**0.66**	-	0.62
Sider-Diseasome	**0.91**	0.87	-	0.45
Sider-Drugbank	0.95	**0.97**	-	0.50
Sider-TCM	**0.99**	0.97	-	0.79
Dailymed-Sider	**1.0**	0.67	0.70	0.62
Drugbank-Sider	**1.0**	0.48	0.46	-
Diseasome-Sider	**0.97**	0.87	0.74	-
Person11-Person12	0.97	**1.00**	0.99	**1.00**
Person21-Person22	0.43	0.46	0.95	**0.97**
Restaurant1-Rest.2	**0.98**	0.77	0.88	0.81
Average-All	**0.88** (+13%)	0.77	-	-
Average-ObjectCoref	**0.89** (+12%)	0.71	0.79	-
Average-Rimon	**0.86** (+9%)	0.80	-	0.71

suggest that Sonda was effective in preserving correct matches and reducing ambiguities (incorrect matches). This facilitates the instance matching task, enabling SERIMI to produce higher quality results.

4.5.3 Utility of the Approach

Finally, in this section, we investigate when it is more efficient to query all instances from the remote endpoint and to perform candidate selection locally instead of executing instance-specific queries against the remote endpoint. This is the case when:

$$\sum_{s \in S} \sum_{q_s \in Q_s} Time(q_s) < \sum_{q_t \in Q_T} Time(q_t) \tag{4.1}$$

where Q_s is a set of instance-specific queries for source instance $s \in S$ and Q_T is a set of queries that retrieve all instances in the target dataset. If we consider t_S and t_T as estimated average time of the queries in Q_s and Q_T, respectively; then Eq. 4.1 can be approximated by:

$$|S| \times |Q_s| \times t_S < |Q_T| \times t_T \tag{4.2}$$

The most straightforward query that retrieves the entire content of an endpoint is the SPARQL query *"select * where {?s ?p ?o}"*. In this case, $|Q_T| = 1$, and consequently, the inequality in Eq. 4.2 would depend entirely on the time t_T. For a relatively small dataset with a few thousand triples, this query can be reasonably fast. For large datasets, it timeouts, because in practice endpoints impose a limit on the time spent on processing a

query (or number of triples that can be retrieved by a query). Instead, the query *"select * where {?s ?p ?o} limit X offset Y"* can be considered, where X represents the number of instances retrieved and $Y = X \times i$ where $i = [0, \frac{|T|}{X}]$. In this case, $|Q_T| = \frac{|T|}{X}$.

To obtain some numbers for Eq. 4.2 using real-world data, we selected the NYTimes dataset from the OAEI benchmark, as an example. It contains 350.000 triples in total, which was loaded using the same system configuration discussed before. We considered one attribute component and 5 query types; consequently, $|Q_s| = 5$. We obtained $t_S = 0.02s$ for these 5 query types. Assuming a limit of 1000 (i.e., $|Q_T| = \frac{350.000}{1000} = 350$), we obtained $t_T = 2.65s$. With these values, Eq. 4.2 can be reduced to $|S| < 9.275$, i.e., for this example, our method is more efficient than the alternative method of retrieving all instances, when S contains less than 9.275 instances.

We note this is the case for all source datasets in the OAEI benchmark, i.e. they have no more than 5000 instances. Moreover, most of the target datasets in this benchmark contains millions of triples. It is clear that it is not efficient to download millions of triples when only a few thousand of them might be relevant for the integration task. These results suggest that in real-world datasets and integration tasks, downloading all data is not always needed. Our on-the-fly matching solution that selects only necessary candidates helps to improve performance, especially when the number of instances to be matched is small.

4.6 Related Work

Candidate selection (blocking) techniques [48] aim at more efficient instance matching by reducing the number of similarity comparisons between instances. As blocking keys, the set of all tokens that can be extracted from the instance data has been used [77]. This approach may yield too many incorrect candidates (as shown in our experiment). The authors tackled this problem by extracting several such profiles and combining them [29]. In principle, this is similar to using several attributes as keys. There are works, which have shown that the most discriminative keys can be selected by considering their discriminative power and coverage [38].

For *matching*, various learning-based approaches exist that can be further distinguished in terms of training data and degree of supervision, respectively (i.e. supervised, semi-supervised, unsupervised [35, 38, 56, 87]). These approaches focus on learning instance matching schemes that are optimized solely towards result quality. There are also works that address the efficiency of learning. In particular, the efficiency of learning has been also studied at query time, where the goal is to minimize the amount of data needed for

learning [88]. The authors perform collective matching, which improves the quality of matches by considering the similarities of nodes related to these matches (the collective). Techniques have been proposed to reduce the number of related nodes that have to be considered [88]. In this work, we show how the number of queries can be minimized to efficiently compute the matches. The proposed technique for retrieving related nodes [88] can be applied in addition (for collective matching). Furthermore, we also optimize the queries towards execution efficiency.

In *On-the-fly integration* scenario, prior research on similarity joins and efficient indexes can be exploited for faster execution [89]. They are complementary to our work. We aim to select efficient queries while these works can be exploited (and implemented by the endpoints) to further optimize the processing of these queries. In principle, SERIMI [25] is able to perform instance matching over SPARQL endpoints. It focuses on refining results using a class-based strategy for pruning those, that do not belong to the class of interest. We take this idea to design class components. However, while SERIMI uses a set of instances as an instance-based class representation, we learn the class component (an explicit query-based representation of class) from data. Further, SERIMI focuses on the quality results but does not pay attention to the cost of obtaining these results, i.e. execution efficiency. Also focusing on result quality, it has been shown that through on-the-fly instance matching, entity search results can be improved [90]. As opposed to these works, we propose to optimize for both result quality and execution efficiency.

4.7 Conclusions

We proposed a candidate selection approach that operates by querying remote data endpoints in the Linked Data. Our method focuses on optimizing the quality of the results as well as optimizing the execution time to obtain them. To achieve high quality results, we learn from the data, candidate selection schemes that are used to build effective instance-specific queries. To achieve time performance, we employ a heuristic-based search algorithm that learns to efficiently execute those queries. We evaluate our approach over two baseline, using two benchmark matching task, OAEI 2010 and 2011. The results indicate that the use of schema information in the queries improves the quality of the results and the overall execution time (because limit the scope where the queries are computed) considerably. Overall, compared to the best baseline, Sonda was 34% faster and improved the quality by 13%, measured in terms of F1.

Chapter 5

Learning Edit-Distance Based String Transformation Rules From Examples

The task of transforming a string from a source format into a target format is encountered in many information-processing tasks. Consider the task of transforming a list of names in the form *"firstname lastname"* (e.g. "Michael Jackson") into the target form *"lastname first letter of the firstname"* (e.g. "Jackson M"). In many domains, identifying an appropriate set of operations that transforms one string to another is challenging, as the space of possible transformations is large. In this work, we investigate the problem of learning string transformation rules from pairs of example strings. We propose a solid way to design these rules based on only four string operations: permutations, insertions, deletions and updates. Additionally, we propose an efficient algorithm to learn such rules, implemented as a combination of variations of well-known string manipulation algorithms. The proposed algorithm has the following desirable properties: it can express any string transformation; it can produce transformation rules that correctly transform a large part of the data, even when a limited number of training examples is provided; it is linear w.r.t the training sample size, which allows it to scale for large tasks; and it easy to understand and implement. We demonstrate the ability of our algorithm over real-world transformation tasks. The results indicate that the algorithm learns transformation rules that are generalizable for a broader range of strings to be transformed, using very few examples. The algorithm is especially useful for spreadsheets and data cleaning tool developers that want to support their end-users on string transformation related tasks.

TABLE 5.1: Examples of String Transformations

i	$u_i \rightarrow v_i$	$s_i \rightarrow t_i$
1	25/12/2003 → 2003-12-25	12/11/2032 → 2032-11-12
2	Aug 06, 2013 → 06/08/13	Aug 20, 2017 → 20/08/17
3	Desoxyn (Tablet) → DESOXYN	Cataflan (For Injection) → CATAFLAN
4	Glycine → <tag>GLYCINE</tag>	Taurines → <tag>TAURINES</tag>
5	Oakland (Calif) → Oakland, California	Dionisio (Calif) → Dionisio, California
6	regulation → regulate	legislation → legislate
7	regulate → regulation	legislate → legislation
8	studies → study	dummies → dummy
9	Durbin, Richard J → Durbin	Amaren, Maro F → Amaren
10	Deutch, John M → John_M._Deutch	Kalikow, Peter S → Peter_S._Kalikow
11	maria joana → maria_joana	brunela averedo → brunela_averedo
12	michael j. jackson → Jackson, Michael J.	jerry l. lewis → Lewis, Jerry L.
13	Perle, La → La Perle	Circulo Magico, El → El Circulo Magico
14	microsoft corp. → Microsoft Corporation	oracle corp. → Oracle Corporation
15	microsoft → MICROSOFT	oracle → ORACLE

5.1 Introduction

The task of transforming a string from a source format into a target format is encountered in many information-processing tasks [91, 92]. Consider the task of transforming a list of names in the form *"firstname lastname"* (e.g. "Michael Jackson") into the target form *"lastname first letter of the firstname"* (e.g. "Jackson M"); or the task to transform a list of dates in the form "31/12/2009" into the form "12-31-2009". These types of string transformations are performed by hundreds of millions of end-users of spreadsheets and data cleaning tools every day. Microsoft has recently implemented a string transformation functionality [93] in the Excel Spreadsheet 2013 [94] and Google has launched Google Refine[1], both aiming to support end-users (specially non-programmers) on data transformation related tasks (e.g. data cleaning).

Recently, Gulwani et. al [94] introduced an effective way to do string transformations. Their method allows end-users to provide an example of a transformation that they want to obtain, and then they can generalize the transformation to the rest of the users' data. This *Programming by Example strategy* (PBE) [95] allows non-programmers (e.g. data analysts, business analysts) to perform transformations that before only expert programmers could perform using shell scripts combined with Awk[2], regular expressions or other advanced programming techniques. PBE avoids repetitive manual labour and allows non-professional programmers to perform complex string transformation tasks.

[1] http://code.google.com/p/google-refine/
[2] https://en.wikipedia.org/wiki/AWK

Although PBE tools are undoubtedly useful, to design these tools is a challenging task. Particularly, it becomes challenging because users want to provide as few example transformations as possible and to produce rules, from this limited set of examples, that generalize for the rest of the data is not trivial.

Existing methods, discussed in Sec. 5.6, either focus on a small set of string transformations or require a large amount of examples to guarantee accurate transformations. For instance, in the well-known area of record matching, some authors [51] have proposed to use string transformations in the process of matching of records. Basically, these methods concern the problem of learning one-to-one mapping rules from pairs of matching examples, i.e., they learn tokens that map to each other. Then they can use these mappings to improve the matching of records. Because they do not learn the character replacements that lead a string into another, these learned transformation rules can only transform what they have observed in the set of examples, i.e., the learned rules generalize poorly. For instance, they can learn $North \rightarrow N$, but they cannot transform "*South*" into "*S*".

Other authors [96] looked into the problem of finding mappings between synonyms, abbreviations and acronyms, which are types of transformations that, usually, cannot be generalized for other strings. For example, a rule that transforms $New\ York \rightarrow Big\ Apple$ is specific for the string "$New\ York$" and cannot be applied in any other string. This problem (that can also be described as learning a dictionary) is considered orthogonal to the problem of learning string transformation rules of interest here.

Closely related to our problem is the work of Okazaki et al. [97]. They tackle the problem of word lemmatization (e.g. $studying \rightarrow study$) and spelling correction (e.g. $vapour \rightarrow vapor$) by learning transformations to be applied on the strings. Their work limits the transformations to a single substring replacement in the source string. Although quite effective for the transformation task that they designed, their method requires a large number of examples to learn the transformation rules. Apart from the works mentioned so far, only few works exist tackling the same problem. We discuss them in more details, in Section 5.6.

This chapter presents a general string transformation algorithm that learns transformation rules from a few given examples. Given a pair of strings (u, v), this work tackles the problem of learning a *transformation rule* (for short, *a rule*), which transforms u into v, so that this learned rule can be used to transform an unseen string s into t, where s is in the same form as u. For example, a rule learned from the pair of strings ("31/12/2009", "12-31-2009") should also transform "01/04/2012" into "04-01-2012" . Particularly, a transformation rule is considered as a set of character permutations, insertions, deletions

and updates that takes place in u to transform it into v. We will use the notation $u \rightarrow v$ to denote a transformation of a string u into v (e.g. $Michael\ Jackson \rightarrow Michael\ J.$).

Table 5.1 shows a few relevant examples of transformations addressed in this chapter. They are real-world use cases drawn from data cleaning and spreadsheet processing literature, representing user questions in data cleaning forums and discussion lists.

The method proposed in this chapter, called *STransformer*, differs from the state-of-the-art string transformation methods in two ways. First, it learns an edit-distance based transformation rule (i.e. a set of character permutations, insertions, deletions and updates) from a single example (u, v), which is the most general set of basic operations that can transform $u \rightarrow v$. The learned rule can not only transform $u \rightarrow v$ but, also, an unseen string s similar to u into its desired target form t. Second, it focuses on string transformations that change the formatting of a string into another. For example, in Table 5.1, a rule that transforms $u_i \rightarrow v_i$ must also transform $s_i \rightarrow t_i$.

The proposed method requires only a few positive examples to learn transformation rules, which are user-provided examples of valid transformations $u \rightarrow v$. The basic idea is to learn a rule for each example transformation that the user has to perform. Next, the learned rules are used to transform the remaining strings that the user has at hand; for each string to be transformed, STransformer selects the rule that is likely to transform it correctly. We assume that the user can provide a compilation of examples of the desired transformations. We do so, because usually the user has a specific need, over a specific dataset, and examples are not available beforehand and cannot be generated or selected by automatic means. The user can potentially participate in the process providing more examples when she observes incorrect transformations being produced. In this work, we assume that the examples are given, and we focus on learning the transformation rules.

5.1.1 Overview and Contributions

Basically, a string can be transformed into another, by eliminating the differences between them, i.e., the parts of a string that are not parts of the other and vice-versa. To do so, the basic edit-distance operations of character permutations, insertions, deletions and updates can be applied. To find a set of operations (a transformation rule) that generates a transformation $u \rightarrow v$ can be trivial (e.g. delete all characters in u and add all characters of v); to find a set of operations that transforms a large number of string correctly is, however, a challenge. The problem involves not just selecting the best set of operations to compose a rule but also how to express these operations as generally as possible. To this end, we propose a set of primitive string operations that can precisely describe a transformation $u \rightarrow v$ but can also be used to transform a large number

of unseen strings similar to u, correctly. To make the problem linear w.r.t the size of the learning sample, each rule is learned independently for each pair of example strings (u, v). As this learning approach leads to many learned rules, a method is proposed to select, automatically, the likely probable rule to transform correctly an unseen string s.

The problem of learning a general transformation rule is formalized in Section 5.2. Any transformation can be expressed with four basic string operations. We propose a generalization of the four basic string operations so that a transformation rule learned from a single example can transform other strings with similar features. As these operations include characters permutation as well, in Section 5.3 we provide a linear time algorithm to find the best permutation of characters of u for permutation-based transformations (e.g. 01/04/2012 $\rightarrow$ 04/01/2012). In Section 5.4, we describe an algorithm to select a learned rule to transform a given unseen string s.

We conduct a detailed empirical investigation of our algorithm (Section 5.5) using real-world string transformation scenarios. For example, in one of these scenarios the user faces the task of putting book titles in a standard format, given a dataset of available titles described in a non-standard format, i.e., transformation like *Art of Science, The* $\rightarrow$ *The Art of Science*. We study various aspects of the algorithm including the generality of learned rules and the linearity w.r.t. the input size. Particularly, we investigate the accuracy of the algorithm in the real-world setting where a very limited set of examples is provided. We do so because in the real world transformation tasks, the user wants to obtain the correct string transformations providing the minimal number of examples as possible. Also, we compare the proposed algorithm, *STransformer*, to the state-of-the-art string transformation algorithm implemented in Microsoft Excel 2013, called *FlashFill*. The results show that STransformer is 30% more accurate than FlashFill, on average. Additionally, STransfomer accuracy dependents less upon a particular example, which is an important property, given that the variety of available examples can be large. Finally, we discuss related work in Section 5.6 and conclude the chapter in Section 5.7.

5.2 Learning Transformations

This section formulates our problem. The input for the transformation-learning problem is a set of N positive string transformation examples $\Lambda^+ = \{(u_i, v_i) : i \in [1, N]\}$, where (u_i, v_i) are pairs of strings from an arbitrary domain; for example, organization names or log file entries. Our high-level goal is to learn transformations from these examples that can be applied to new instances of our problem where the desired output is not

yet available. First we introduce basic definitions, and then we formulate the learning problem.

5.2.1 Preliminary Definitions

Definition 5.1 (Characters)**.** A character a belongs to an alphabet Σ. A distance between two characters a and b is denoted as $\Delta(a, b) = n$, where $n \in \mathbb{Z}$. Also, a character a belongs to a class of characters denoted as $C(a) = \mathbf{e}$, where $\mathbf{e} \in E^{\Sigma}$, the set of all regular expression for the alphabet Σ.

Definition 5.2 (Strings)**.** A string $u \in \Sigma^*$. The set of all substrings of u is denoted as u^*. The length of u is denoted as $|u| = n$. An empty string is denoted as λ, where $|\lambda| = 0$. A character at position i in a string u is denoted as $u[i]$. A substring of u is denoted as $u[i..j]$, where $0 \leq i \leq j \leq |u|$. The u^c is a string where $\forall i, u^c[i] = C(u[i])$.

Without loss of generality, unless noted otherwise, we assume the ISO-8859-1[3] character set as the alphabet Σ and the distance $\Delta(a, b)$ as the difference of the decimal representation of a and b in the ISO-8859-1 character set table. For example, the distance $\Delta(\text{“}a\text{”}, \text{“}A\text{”}) = 97 - 65 = 32$, while $\Delta(\text{“}A\text{”}, \text{“}a\text{”}) = 65 - 97 = -32$. We defined the following specific classes of characters: lowercase letter (**l**), uppercase letter (**u**), digit (**d**), space (**s**), punctuation (**p**), lowercase accented letter (**a**), uppercase accented letter (**b**), and any other character (**f**). We denote as **z** the class of delimiter characters, which unless noted otherwise, we consider existing in the beginning and end of every string.

We defined four basic string operations: permutation, insertion, deletion and update such as:

Definition 5.3 (Permutations)**.** The function permutation $P(u, i, j) : \Sigma^* \times \mathbb{N} \times \mathbb{N} \rightarrow \Sigma^*$; permutes a character $u[i]$ with $u[j]$ in u.

Definition 5.4 (Insertions)**.** The function insertion $I(v, u, i,) : \Sigma^* \times \Sigma^* \times \mathbb{N} \rightarrow \Sigma^*$; inserts a string v in the position i of u.

Definition 5.5 (Deletions)**.** The function deletion $D(u, i) : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^*$; deletes the character in the position i of u.

Definition 5.6 (Updates)**.** The function update $U(a, u, i) : \Sigma \times \Sigma^* \times \mathbb{N} \rightarrow \Sigma^*$; replace the character in the position i of u by a.

The last three operations recall the operations used in the Levenshtein edit distance [98], a string metric for measuring the difference between two strings. However, in this

[3] http://en.wikipedia.org/wiki/ISO_8859-1

chapter, these operations will be used to determine a string transformation instead of measuring the difference between two strings. Moreover, an additional permutation operation is considered, to describe a transformation represented by the permutation of characters in a string (e.g. $Michael\ Jackson \rightarrow Jackson\ Michael$).

Notice that string permutations and updates can be represented by insertions and deletions, but these four operations better represent a transformation $u \rightarrow v$ that consists of eliminating differences between u and v, i.e., first, if $|u| = |v|$ and they have the same characters but $u \neq v$, then characters in u have to be permuted to transform it into v; second, if $|u| > |v|$, it means that characters in u should be deleted; third, if $|u| < |v|$, it means that characters from v have to be inserted in u; and finally, if $|u| = |v|$ and set of characters of u differs from the set of characters from v, it means that characters in u have to be replaced by a character from v. For example, a transformation rule to produce the transformation $michael\ jackson \rightarrow J.\ Michael$ would require all four types of string operations. It would require to permute *michael* with *jackson* (*jackson michael*), to delete "akson" (*j michael*), to insert "." after "j" (*j. michael*), and to update "j" with "J" and "m" with "M" (*J. Michael*).

A string operation t (permutation, insertion, deletion or update) over a string u that results in a string u_1 is denoted as $u \mapsto^t u_1$.

5.2.2 Transformation Rules

The set of transformations that we consider in this work is based on these four string operations. Basically, a transformation $u \rightarrow v$, is seen as a set of permutations, insertions, deletions and updates that have to be performed in u to transform it into v. A *transformation rule* is a chain of string operations. A *transformation model* is a set of transformation rules learned from a set of examples Λ^+, denoted by $\Omega(\Lambda^+)$.

Definition 5.7 (Transformation Rule)**.** Given a pair of strings (u, v), a transformation rule is a chain of string operations $T = \{t_1, \ldots, t_n\}$ that transforms u to v. It is denoted as $u \rightarrow^T v$, where $\rightarrow^T$ can be expanded to $u \mapsto^{t_1} u_i \mapsto^{t_2} \ldots \mapsto^{t_n} v$.

For example, the transformation 12/01/2009 $\rightarrow$ 12/01/09 can be achieved using the specific transformation rule 12/01/2009 $\rightarrow^D$ 12/01/09, where $D = \{D(u, 7), D(u_2, 7)\}$ is a chain of deletion operations. This transformation can also be expressed in terms of its operations; i.e., 12/01/2009 $\mapsto^{D(12/01/2009,7)}$ 12/01/009 $\mapsto^{D(12/01/009,7)}$ 12/01/09.

Different transformations require different types of transformation rules. For example, the four transformations *michael jackson* $\rightarrow$ *jackson michael*, *michael jackson* $\rightarrow$

michael (*jackson*), *michael jackson* $\rightarrow$ *michael j*, *michael jackson* $\rightarrow$ *Michael Jackson*, require permutation, insertion, deletion and update transformation rules, respectively. More complex transformations require a concatenation of transformation rules.

In this work, we are particularly interested in complex transformation rules in the form $u \rightarrow^P u_1 \rightarrow^I u_2 \rightarrow^D u_3 \rightarrow^U v$, where P, I, D and U, are sequences of permutation, insertion, deletion and update operations; respectively. We denote such a transformation rule as $\Psi_{u,v}$. Notice this chain of transformations can represent any string transformation.

Universe of permutations: The purpose of using permutation in our rules is to capture permutation of tokens in a string (e.g. *michael jackson* vs. *jackson michael*, or 12/01/09 vs. 09/12/01). As these permutations usually involve a separator character (e.g. space, slash), we will consider only permutations of substrings (tokens) that are separated by a character a (separator) of u. Such a subset of permutations captures permutations on the majority of the human readable strings and drastically reduces the problem search space.

Universe of insertions: When transforming a string u to v, it only makes sense to consider insertions that insert a substring $x \in v^*$ into u, because the insertion of any substring $y \notin v^*$ would require further an update or deletion in u to obtain v. For that reason, we restrict our insertions I to the set of insertions that insert $x \in v^*$ into u. Notice that an insertion is only performed if $|u| < |v|$.

Universe of deletions: Deletions always occur after insertions in a transformation rule. Their purpose is to reduce the length of $|u|$ to $|v|$, if $|u| > |v|$.

Universe of updates: In the chain of transformations that we propose, updates only occur in the last step. At that point, $|u| = |v|$. We only consider updates of a character $u[i]$ with its corresponding character $v[i]$. This set of updates always transforms u into v.

5.2.3 Generalization of Transformation Rules

Our ultimate and most important goal is to transform an unseen string w, using the transformation rules learned from Λ^+. However, a transformation rule learned for a single example $(u, v) \in \Lambda^+$ is very specific and would, if at all, only generalize to a limited set of strings highly similar to u. For example, a transformation rule $12.00 \rightarrow^\Psi 12,00$ can be created based on replacing the "." by a ",": a rule Ψ composed of $P = I = D = \emptyset$ and $U = U(\text{“,”}, 12.00, 3)$. Such a rule can generate a correct transformation for $w =$ "$32,03$", i.e., $32.03 \rightarrow^\Psi 32,03$, but when applied to $w =$ "145.00", the transformation

$145.00 \rightarrow^{\Psi} 14, .00$ results in an incorrect string. Any human would proper understand from the example that the intended transformation is $145.00 \rightarrow^{\Psi} 145, 00$.

To achieve this goal, a generalization of the basic string operations is proposed, so that a larger range of strings can be transformed correctly by the same transformation rule. Consequently, fewer examples will be needed to cover a larger number of strings that the user may want to transform. We denote as $\Psi^{G}_{u,v}$, a generalization of a transformation rule $\Psi_{u,v}$. Mainly, they differ on the level of abstraction that their operations are defined. Intuitively, to make a rule general, covering the highest number of strings possible, it makes sense to consider information about all strings to be transformed. However, to make this problem feasible, only information in the string (u_i, v_i) is used to produce a rule $\Psi^{G}_{u_i,v_i}$. Although it looks like a limitation, such strategy is very effective (as we will show empirically) and avoids an exponential problem of combining information of many strings in this process of generalization.

Two elements in a rule can be generalized: the position in the string where an operation takes place and the character(s) part of the operation. We propose to infer the position as a function of a character in the string; and a character itself to be abstracted to its class (e.g. 7 can be seen as a digit). For example, the transformation $Michael \rightarrow M$ could be expressed as: remove all characters after the first uppercase letter. This rule is clearly more general than a rule that specifies a chain of deletion operations that transforms "Michael" to "M". Consequently, it could also be used to transform $Elvis \rightarrow E$. Therefore, we define the new notion of *relative position* in a string, to allow us to express more general transformations.

Definition 5.8 (Relative position)**.** A relative position function $R_u(k, e, j) : \mathbb{N} \times E^{\Sigma} \times \mathbb{Z} \rightarrow \mathbb{N}$ retrieves a position i in a string u^c of u that is j positions distant from the k-th occurrence of a substring e in u^c.

We define e as an *n-gram* (a string of size n) in u^c. The set of all relative positions in u is denoted as R^u.

Using relative position, the "/" between day and month in the string $u =$ "12/01/09", can be represented as $R_u(1, \mathbf{p}, 0) = 3$, i.e., the position of the first punctuation ($\mathbf{p}$). As well as by $R_u(3, \mathbf{d}, -1) = 3$, i.e., the position before the third digit ($\mathbf{d}$).

Using relative position, we introduce generalizations of the four basic operations introduced above: generalized permutation, insertion, deletion and update.

Generalized Permutation. A permutation rule $P = \{p_1, \ldots, p_n\}$ is generalized into a permutation rule $P^{G}_{a} = \{s^{G}_{1}, \ldots, s^{G}_{m}\}$, where $m \leq n$, a is a character (separator) and

$\rightarrow^{P_a^G}$ is a chain of concatenations of the type $s_1^G + a + s_2^G + a + \cdots + a + s_m^G$, where s_i^G is selection operation defined below:

Definition 5.9 (Selection)**.** The function selection $S(u, r_1, r_2) : \Sigma^* \times R^u \times R^u \rightarrow \Sigma^*$; selects the substring $u[r_1..r_2]$.

For example, the transformation 30/05/09 → 09/30/05 could be done with the generalized permutation rule $P_/^G = \{S(u, R_u(2, \mathbf{p}, 1), R_u(2, \mathbf{z}, -1)), S(u, R_u(1, \mathbf{d}, 0), R_u(2, \mathbf{d}, 0)), S(u, R_u(3, \mathbf{d}, 0), R_u(4, \mathbf{d}, 0)\}$, where $u =$ "30/05/09". Notice that **z** denotes an unprintable delimiter character in the beginning and end of all strings. The three selection operations in $P_/^G$ select the substrings "09", "30" and "05"; respectively. The chain of concatenations "09" + "/" + "30" +"/" + "05" results in the string "09/30/05". Notice that the previous rule can be used to transform any date in the source format into the target format, including a date such as "12/20/2009", where the year is represented with four digits.

Generalized Insertion. An insertion rule $I = \{I_1, \ldots, I_n\}$ is generalized into $I^G = \{I_1^G, \ldots, I_n^G\}$, by making the positions of the insertions I_i relative.

For example, to transform $Microsoft\ Corp \rightarrow Microsoft\ Corporation$, the operation $I(\text{"}oration\text{"}, u, 14)$ is required, where $u =$ "$Microsoft\ Corp$". The position 14 can also be represented as $R_u(2, \mathbf{z}, -1)$, where **z** is the set of delimiter characters. By making the position relative, i.e. $I(\text{"}oration\text{"}, u, R_u(2, \mathbf{z}, -1))$, this rule would also transform *Apple Corp* → *Apple Corporation*.

Generalized Deletion. A deletion rule $D = \{D_1, \ldots, D_n\}$ is generalized into a deletion rule $D^G = \{D_1^G, \ldots, D_m^G\}$, where $m \leq n$, $\rightarrow^{D^G}$ is a chain of operations D_i^G defined below:

Definition 5.10 (Deletion)**.** The function deletion $D^G(u, r_1, r_2) : \Sigma^* \times R^u \times R^u \rightarrow \Sigma^*$; deletes from u the substring $u[r_1..r_2]$.

For example, *James Brown* → *J Brown* can be transformed by deleting the characters between the first uppercase character and the subsequent space, i.e., by the generalized deletion rule $D^G(u, R_u(1, \mathbf{u}, 1), R_u(1, \mathbf{s}, -1))$, where $u =$ "$James\ Brown$". The relative positions $r_1 = R_u(1, \mathbf{u}, 1) = 2$ and $r_2 = R_u(1, \mathbf{s}, -1) = 5$ define the substring $u[2..5] =$ "*ames*", which is the substring to be deleted from u.

Generalized Update. An update rule $U = \{U_1, \ldots, U_n\}$ is generalized into an update rule $U^G = \{U_1^G, \ldots, U_m^G\}$, where $m \leq n$, $\rightarrow^{U^G}$ is a chain of operations U_i^G defined below:

Definition 5.11 (Update)**.** The function update $U^G(u, r, e, d) : \Sigma^* \times R^u \times E^\Sigma \times \mathbb{Z} \rightarrow \Sigma^*$; applies a distance d to each character of the substring resultant from the first match of the regular expression e in the substring $u[r..|u|]$.

For example, the transformation $VICTOR\ HUGO \rightarrow Victor\ Hugo$ can be achieved with the update rule $U^G = \{U_1^G(u, R_u(1, \mathbf{u}, 1), \mathbf{u}^+, -32), U_2^G(u, R_u(1, \mathbf{s}, 2), \mathbf{u}^+, -32)\}$, i.e., $VICTOR\ HUGO \rightarrow^{U_1^G} Victor\ HUGO \rightarrow^{U_2^G} Victor\ Hugo$, where $u =$ "$VICTOR\ HUGO$" [4].

Concluding, we are particularly interested in learning the transformation rules $\Psi_{u,v}^G$ of the form $u \rightarrow^{P^G} u_1 \rightarrow^{I^G} u_2 \rightarrow^{D^G} u_3 \rightarrow^{U^G} v$. Section 5.3 introduces our algorithms to learn the components of $\Psi_{u,v}^G$: P^G, I^G, D^G and U^G.

5.2.4 Learning Problem

This section defines the *validity* and *coverage* of a transformation rule $\Psi_{u,v}^G$ for a transformation $x \rightarrow y$; and, finally, formalizes the string transformation-learning problem.

Definition 5.12. Given a pair of strings (x, y), the validity of a transformation rule $\Psi_{u,v}^G$ is defined as:

$$Validity(\Psi_{u,v}^G, x, y) = \begin{cases} 1 & \text{if } x \rightarrow^{\Psi_{u,v}^G} y \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The coverage of $\Psi_{u,v}^G$, given a set of examples Λ^+ is defined as:

$$Cov(\Psi_{u,v}^G, \Lambda^+) = \sum_{(x,y)\in\Lambda^+} Validity(\Psi_{u,v}^G, x, y) \tag{5.2}$$

The string transformation-learning problem is divided into two sub problems. The first sub problem concerns the problem of learning a transformation rule from a pair of strings (u, v). The second sub problem concerns the problem of selecting a rule from a set of learned rules to transform an unseen string w. We now formally state our first learning problem.

Definition 5.13. Given a pair of strings (u, v) and a set of positive examples Λ^+, find the transformation rule $\Psi_{u,v}^G$ that maximizes its coverage over Λ^+.

We now state our second and last learning problem.

Definition 5.14. Given a transformation model $\Omega(\Lambda^+)$ and a pair of strings (x, y), find a transformation rule $\Psi_{u,v}^G \in \Omega(\Lambda^+)$, such that $Validity(\psi_{u,v}^G, x, y) = 1$.

Assuming that we have learned a transformation model $\Omega(\Lambda^+)$, this second learning problem is the problem of selecting from $\Omega(\Lambda^+)$ a rule that can possible transform an

[4] Adding -32 to each character in "$ICTOR$" produces its correspondent lower case version "*ictor*".

unseen string x correctly. The algorithm that solves the first learning problem is called the *rule learner*, while the one that solves the second learning problem is called the *rule selector*. They are described in Section 5.3 and 5.4; respectively.

5.3 Rule Learner Algorithm

This section describes our algorithm to produce a transformation rule $\Psi^G_{u,v}$ from (u, v).

5.3.1 Rule Learning

The rule learner algorithm is composed of four parts, corresponding to the four basic string operations permutation, insertion, deletion and update. Intuitively, the algorithm learns possible permutations in u, then on the permuted string u_p it identifies insertions and deletions to transform it to u_{i_d}, such that $|u_{i\_d}| = |v|$; and finally, in the resultant string u_{i_d}, it learns the character replacements necessary to transform u_{i_d} into v.

Firstly, we describe the algorithm to find a relative position in u, to be used in the string operations.

5.3.2 Relative Position Algorithm

Alg. 9 obtains a relative position $R_u(k, e, j)$ of $u[i]$ in the string u. The purpose of the algorithm is to select the most general relative position as possible, i.e., the one that is likely to represent the same absolute position over a set of similar strings. Practically, the most general relative position is also the most discriminative one, with the smallest offset j. In other words, we identify e that is the least frequent in u^c while closest to the absolute position i.

In order to compute the relative position $R_u(k, e, j)$ of a position i of u, all substrings e in u^c have to be considered. Let E_n be the bag of all n-grams of u^c, i.e., $E_n = \{e_p | \forall p, 0 < p < |u^c| - n : e_p = u^c[p..p + n]\}$, where $1 \leq n \leq |u^c|$, and let $f(x)$ be a function that gives the frequency of a substring x in u^c; then, for a specific $e \in E_n$, $k = f(e)$.

For example, consider the string $u =$ "$Noia, La$". Assume E_2, the bag of bigrams, of $u^c =$ **zulllpsulz**, i.e., $E_2 =$ {**zu, ul, ll, ll, lp, ps, su, ul, lz**}. The frequency of each distinct character class $a \in u^c$ is $f(\mathbf{u}) = 2$, $f(\mathbf{l}) = 4$, $f(\mathbf{p}) = 1$ and $f(\mathbf{s}) = 1$. Particularly, $f(\mathbf{z}) = 0$, by definition. While the frequency of each bigram in E_2 is: $f(\mathbf{zu}) = 1$, $f(\mathbf{ul}) = 2$, $f(\mathbf{ll}) = 2$, $f(\mathbf{lp}) = 1$, $f(\mathbf{ps}) = 1$, $f(\mathbf{su}) = 1$ and $f(\mathbf{lz}) = 1$.

The most discriminative $e \in E_n$ is the one with the smallest k (frequency) and smallest absolute distance j to i. This discriminative score can be computed by calculating the average frequency of the characters in a substring $w_e \in u^{c^*}$ that starts in e and ends in i, (or vice-versa, if $i < p$).

Take for example the position $i = 7$. In the previous example, $u^c[7] = \mathbf{u}$. For the first bigram $e_1 = \mathbf{zu}$, the substring w_{e_1} would be $w_{zu} = \mathbf{zulllpsu}$, and for the bigram $e_9 = \mathbf{lz}$, $w_{lz} = \mathbf{ulz}$. The example is illustrated in Fig. 5.1.

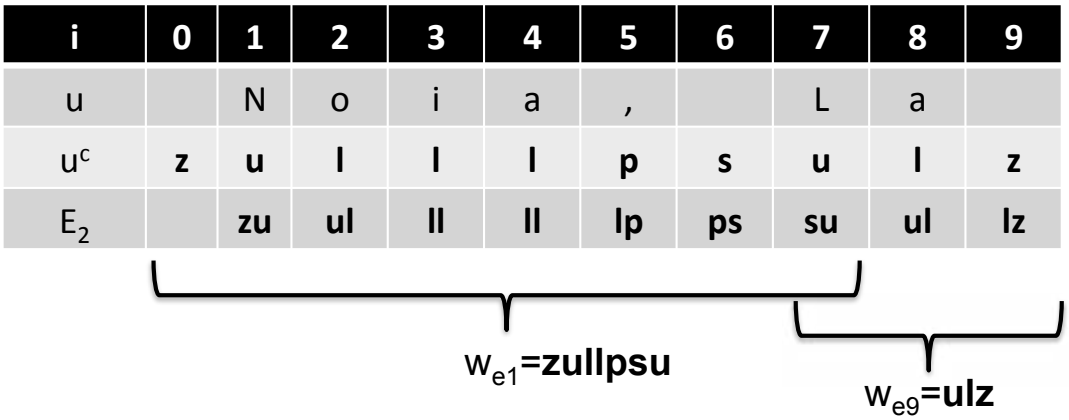

FIGURE 5.1: String u = "$Noia, La$", u^c = **zulllpsulz**, $E_2(u^c)$ = {**zu, ul, ll, ll, lp, ps, su, ul, lz**}, w_{e_1} = **zulllpsu** and w_{e_9} = **ulz**.

The best e will be the one with the smallest average frequency in w_e. It is obtained by solving the minimization described in Eq. 5.3:

$$\arg\min_{e \in E_n} F(e) \tag{5.3}$$

$$F(e) = \sqrt[f(e)]{Mean(e)} \tag{5.4}$$

$$Mean(e) = \sqrt[m]{\frac{\sum_{a \in w_e} g(a)^m}{m}} \tag{5.5}$$

where $m = |w_e|$, a is a character in w_e and $g(a) = \frac{f(a)}{|u^c|}$. Notice that $Mean(e)$ is the *power mean* over the frequency $f(a)$ of character a in u^c. Intuitively, the power mean gives a higher score to longer strings w_e, i.e., e that are far from $u^c[i]$. Consequently, e that are closer to i will have a smaller score. In Eq. 5.4, the root $f(e)$ is used to increase the score of e that are more frequent. Therefore, e that are less frequent in u^c and closer to i will be selected.

In the current example, as $|u^c| = 10$, we obtain the following scores $F(e)$ for each $e \in E_2$: $F(\mathbf{zu}) = 0.353$, $F(\mathbf{ul}) = 0.354$, $F(\mathbf{ll}) = 0.356$, $F(\mathbf{ll}) = 0.578$, $F(\mathbf{lp}) = 0.287$, $F(\mathbf{ps}) = 0.149$, $F(\mathbf{su}) = 0.158$, $F(\mathbf{ll}) = 0.562$ and $F(\mathbf{lz}) = 0.288$. $F(\mathbf{ps}) = 0.149$ is the smallest score, therefore, $e = \mathbf{ps}$ is selected. Then, the relative position for $i = 7$ is defined as $R_u(1, \mathbf{ps}, 1)$.

Algorithm 9 RelativePosition(u, i, n).

for all p in $0..|u| - n$ **do**
  $e \leftarrow u^c[p..p+n]$
  $E_n \leftarrow E_n \cup e$
  if $p \leq i - n$ **then**
    $w_e \leftarrow u^c[p..i]$
  else if $p > i - n$ and $p \leq i$ **then**
    $w_e \leftarrow u^c[p..p+n]$
  else
    $w_e \leftarrow u^c[i..p+n]$
  end if
  $sum[w_e] \leftarrow 0$
  for all a in w_e **do**
    $sum[w_e] \leftarrow sum[w_e] + (f(a)/|u^c|)^m$
  end for
  $m \leftarrow |w_e|$
  $Mean[e] \leftarrow \sqrt[m]{(sum[w_e]/m)}$
  $F[e] \leftarrow \sqrt[f(e)]{Mean[e]}$
end for
$e_p \leftarrow \arg\min_{e_p \in E_n} F[e_p]$
return $[f(e_p), e_p, i - p]$

This algorithm is bound by $O(|u|^2)$ for $n = 1$, and $O(1)$ for $n = |u|$. We observed empirically that this algorithm produces the most discriminative relative position with high likelihood when $n = 2$. Notice that for $n = |u|$, $j = i$ in $R_u(k, e, j)$, i.e. j is the value of the absolute position i.

5.3.3 Permutation Rule Learner

To learn permutations in u, we have to select among all permutations of tokens in u (separated by a character c), the permutation that the concatenation of its tokens forms a string that is closer to the target string v. For example, for the transformation *John M. Lewis* $\rightarrow$ *John, Lewis M.*, considering that the tokens are separated by a space (c is a space character), there are six possible permutations for the tokens [“*John*”, “*M.*”, “*Lewis*”]: {[“*John*”, “*Lewis*”, “*M.*”], [“*M.*”, “*John*”, “*Lewis*”], [“*M.*”, “*Lewis*”, “*John*”], [“*Lewis*”, “*John*”, “*M.*”], [“*Lewis*”, “*M.*”, “*John*”]}. The permutation [“*John*”, “*Lewis*”, “*M.*”] is the most similar to the string $v =$ *John, Lewis M.*.

As we do not know the separator c in advance, all characters in u should be considered as possible separators. So, to learn permutations in u (delimiters are ignored), for each distinct character c in u, we build a set of tokens $t_c = \{u_1, \ldots, u_n\}$, where $u = u_{t_c} = u_1 + c + \cdots + c + u_n$. Let Q^{t_c} be the set of all $n!$ permutations of tokens in t_c, where $n = |t_c|$. From the union $Q = \bigcup_{c \in u} Q^{t_c}$, we select the permutation $q_i \in Q$ that generates a string u_{q_i} with the highest similarity to v, i.e. we solve the problem:

$$\underset{q_i \in Q}{\operatorname{argmax}}\, sim(u_{q_i}, v) \tag{5.6}$$

The well-known *Levenshtein edit-distance*[5] is used as $sim(\cdot, \cdot)$.

For example, for the transformation *John M. Lewis* $\rightarrow$ *John, Lewis M.*, considering $c =$ " " (space), the token set is $t_c = \{$"*John*", "*M.*", "*Lewis*"$\}$ and $Q^{t_c} = \{$["*John*","*M.*", "*Lewis*"], ["*John*", "*Lewis*", "*M.*"], ["*M.*", "*John*", "*Lewis*"], ["*M.*", "*Lewis*", "*John*"], ["*Lewis*", "*John*", "*M.*"], ["*Lewis*", "*M.*", "*John*"]$\}$. Among all possible Q^{t_c}, the permutation $q_i =$ ["*John*", "*Lewis*", "*M.*"], which forms the string "John Lewis M.", has the highest similarity to the target string. Notice that the complete set Q would be the union of sets Q^{t_c}, where c would be any of the characters $\{$"*J*", "*o*", "*h*", "*n*", " ", "*M*", ".", "*L*", "*e*", "*w*", "*i*", "*s*"$\}$.

The generation of all permutations of t_c becomes prohibitive when t_c has too many tokens. As we do not need to list all permutations but only to find the best one, the best permutation of t_c can be efficiently found by sorting the tokens in t_c, using sim as criteria of ordering the tokens. Alg. 10 specifies this sorting process, explained below.

Starting from a set of tokens $t_c = \{u_1, \ldots, u_n\}$, the algorithm (Alg. 10) uses Selection Sort[6] to sort the tokens, where sim is the criteria of ordering. Basically, in this algorithm two tokens i and j in t_c are permuted if $sim(w_{t_c}, v) > sim(u_{t_c}, v)$, where i and j are permuted in w_{t_c} and i and j are not permuted in u_{t_c}. The final sorted t_c is included in Q. Notice that Q will have a maximum of m elements, where m is the number of distinct characters in the string u.

The Sorting algorithm takes $O(n^2)$ comparisons to sort $n = |t_c|$ tokens in t_c. In the worse case, when all characters in u are distinct (i.e., $m = |u|$), the algorithm to solve Eq. 5.6 is bound by $O(m.n^2)$ because it runs the sorting algorithm m times, once for each possible separator in u. As Q contains the best permutation of each separator, the

[5] Notice that set-based similarity functions (e.g., Jaccard) cannot be used as $sim(\cdot, \cdot)$ because they ignore the tokens order in u_{q_i} and v.

[6] Any sorting algorithm can be used to sort the tokens. We used Selection Sort because in this problem the number of tokens are relatively small and it performs satisfactorily.

Algorithm 10 Sorting(t_c, v).

return t_c if $|t_c| = 1$
$u_{t_c} \leftarrow concatenate(t_c)$
$uscore \leftarrow sim(u_{t_c}, v)$
$i \leftarrow j \leftarrow max \leftarrow 0$
for all i in $0..|t_c| - 1$ **do**
  $max \leftarrow j$
  for all j in $i + 1..|t_c|$ **do**
    $tmp \leftarrow t_c$
    $tmp[j], tmp[max] \leftarrow tmp[max], tmp[j]$
    $w_{t_c} \leftarrow concatenate(tmp)$
    $wscore \leftarrow sim(w_{t_c}, v)$
    if $wscore > uscore$ **then**
      $uscore \leftarrow wscore$
      $max \leftarrow i$
      $t_c[i], t_c[max] \leftarrow t_c[max], t_c[i]$
    end if
  end for
end for
return t_c

overall best permutation in Q can be trivially obtained in $O(m)$. Notice that excluding letters as separators decreases the size of m, substantially.

Lemma 5.15. *The Selection Sort is correct* [7].

Theorem 5.16. *The procedure Sorting*(t_c, v) *using Levenshtein edit distance as sim finds the permutation of* t_c *that maximizes* $sim(u_{t_c}, v)$*, i.e., it solves the problem:*

$$\underset{q \in Q^{t_c}}{\operatorname{argmax}}\, sim(u_q, v) \tag{5.7}$$

Proof. The Levenshtein edit-distance measures the number of operations to transform a string into another. Following the Lemma 5.15, at any iteration, $t_c[i]$ is only permuted to $t_c[j]$ if the permutation reduces the number of operations to transform $u_{t_c} \rightarrow v$. Consequently, the final sorted t_c contains the permutation that reduces the highest number of edit-distance operations in $u_{t_c} \rightarrow v$. In other words, the sorted t_c maximizes $sim(u_{t_c}, v)$. □

Finally, the permutation $q_i \in Q$ with highest similarity to v is expressed as a transformation rule as follows. For each token $u_k \in q_i$, a selection operation $S(u, r_1, r_2)$ is created, where r_1 is the relative position of the first character of u_k in u, and r_2 is the relative position of the last character of u_k in u . Then, the set of all selection operations forms a permutation rule P_c^G.

[7] The proof of the Lemma 5.15 is available in [99].

For instance, in the current example, considering $c =$" ", then, P_c^G would contain three selection operations, one for each token in $t_c = [$"*John*", "*Lewis*", "*M.*"], i.e., $P_c^G = \{S(u, R_u(1, \mathbf{zu}, 1), R_u(1, \mathbf{ls}, 0)), S(u, R_u(2, \mathbf{su}, 1), R_u(1, \mathbf{lz}, 0)), S(u, R_u(1, \mathbf{su}, 1), R_u(1, \mathbf{ps}, 0))\}$. As $u =$ "*John M. Lewis*", the first selection would select "John", the second would select "Lewis" and the third "M.". By definition, the permutation operation concatenates all selected strings by the character c, which in this case would produce $u_p =$ "*John Lewis M.*". In this example an additional insertion of a conman after "John" would be required to fully transform *John M. Lewis* $\rightarrow$ *John, Lewis M.*. Notice that the same permutation rule could be used to transform *Michael B. White* $\rightarrow$ *Michael, White B.*.

5.3.4 Insertions and Deletions Rule Learner

To learn the insertions and deletions in u_p, we propose an algorithm based on the known *longest common substring algorithm (LCS)*[8] [101]. The purpose of our algorithm is to find all possible alignments of characters between u_p and v. As the characters in u have the best permutation at this stage, if $u_p[i] = v[j]$ and $u_p[h] = v[k]$, then $i > h$ and $j > k$ or $i < h$ and $j < k$. If u_p and v are represented in a matrix as shown in Fig. 5.2; basically, characters in u_p that do not align to v (zero columns) have to be deleted from u_p to transform it into v; and characters in v that does not align to u_p (zero rows) have to be inserted in u_p to transform it into v.

	$	A	u	g		0	6	,		2	0	1	3	$
$	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
/	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
/	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
$	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0

FIGURE 5.2: All common substrings between $u_p =$ "*Aug* 06, 2013" and $v =$ "06/08/13".

For example, Fig. 5.2 shows a matrix representing all character alignments between $u_p =$ "*Aug* 06, 2013" and $v =$ "06/08/13". The characters that align are represented

[8] Not to be confused with longest common **subsequence** problem [100].

as 1 and the characters that do not align are represented as zero. In this example, the substrings {"*Aug*", ", 2"} (zero columns) have to be deleted from u_p; and {"/", "8/"} (zero rows) have to be inserted in u_p. The insertion and deletion rule can be represented as: $D^G = \{D_1^G(u_p, R_{u_p}(1, \mathbf{zu}, 1), R_{u_p}(1, \mathbf{zu}, 4)),\ D_2^G(u_p, R_{u_p}(1, \mathbf{ps}, 0), R_{u_p}(1, \mathbf{ps}, 2))\}$ and $I^G = \{I_1^G(\text{"/"}, u_p, R_{u_p}(1, \mathbf{ps}, 0)),\ I_2^G(\text{"8/"}, u_p, R_{u_p}(1, \mathbf{ps}, 4))\}$, respectively.

Given a matrix K of $m \times n$ where $m = |u_p|$ and $n = |v|$, the LCS finds the longest diagonal in the matrix where $u_p[i] = v[j]$, $1 \leq i \leq m$ and $1 \leq j \leq n$. We consider $u_p[i]$ equal to $v[j]$, if $u_p[i] = v[j]$ or $lowercase(u_p[i]) = lowercase(v[j])$.

Our algorithm starts looking for the longest common substring *(lcs)* between u_p and v, using LCS. Once the *lcs* $u_p[r..s] = v[x..y]$ is determined, the LCS is applied recursively over the upper and lower matrix $K_U = [1, 1, r, x]$ and $K_L = [s, y, m, n]$, respectively. This step makes this algorithm different from the LCS because it will guarantee, after all character are matched, that if $u_p[i] = v[j]$ and $u_p[h] = v[k]$, then $i > h$ and $j > k$ or $i < h$ and $j < k$. After all *lcs* are determined, the lines that are not part of a *lcs* represent the characters in v that have to be inserted in u_p and the columns the characters that have to be deleted from u_p. This algorithm is bound by the upper bound of the original LCS, which is $O(n.m)$.

Alg. 9 is used to find the relative position for the insertions and deletions. The set of insertions I^G forms a transformation rule $u_p \rightarrow^{I^G} u_i$, and the set of deletions D^G forms a transformation rule $u_i \rightarrow^{D^G} u_{i\_d}$.

5.3.5 Update Rule Learner

To learn the update rules, for each $u_{i\_d}[i]$ in u_{i_d}, a rule $U_i^G(u, r_i, e_i, d_i)$ is created, where the relative position r_i of i is found using Alg. 9, the distance $d_i = \Delta(u_{i\_d}[i], v[i])$, and the RE $e_i = u^c[i]$. Any consecutive set of rules $U_i^G, \ldots, U_j^G$ that have the same distance and the same e_i, are merged in one U_i^G, where e_i is defined to e_i^+, the *Kleene Plus* [102] of e_i. This produces a set of update rules U^G, such that $u_{i\_d} \rightarrow^{U^G} v$. This algorithm is bound by $O(|v|)$, due that it has to walk once through the string u_{i_d} to compute the distance Δ to v.

For example, for *jack_w.* $\rightarrow$ *Jack W.*, we have $u^c =$ **llllplp** and $v^c =$ **ulllsup**. The algorithm produces 7 rules (ignoring the delimiters, $|u|$=7), i.e., $U_1^G(u, R(1, \mathbf{l}, 0), \mathbf{l}, -32)$, $U_2^G(u, R(1, \mathbf{p}, -3), \mathbf{l}, 0)$, $U_3^G(u, R(1, \mathbf{p}, -2), \mathbf{l}, 0)$, $U_4^G(u, R(1, \mathbf{p}, -1), \mathbf{l}, 0)$, $U_5^G(u, R(1, \mathbf{p}, 0), \mathbf{p}, 63)$, $U_6^G(u, R(1, \mathbf{p}, 1), \mathbf{l}, -32)$, $U_7^G(u, R(2, \mathbf{p}, 0), \mathbf{p}, 0)$. The consecutive rules U_2^G, U_3^G and U_4^G have the same distance and RE, then they are replaced by the rule $U_2^G(u, R(1, \mathbf{p}, -3), \mathbf{l}^+, 0)$. Such a set of rules can transform any string in the form $\mathbf{ll}^+\mathbf{plp}$ into $\mathbf{ul}^+\mathbf{sup}$.

5.3.6 Discussion

Table 5.1 shows a large variety of examples of string transformations: $u_i \rightarrow v_i$ and $s_i \rightarrow t_i$. The algorithm just described can learn a rule using as example a transformation rule $\Psi^G_{u_i,v_i}$ and produce a correct transformation for any string s_i in this table (i.e. $s_i \rightarrow^{\Psi^G_{u_i,v_i}} t_i$). Although some of these transformations could be easily expressed by programmers, non-programmers do not have the skills to express it. The proposed method allows them to perform a large range of non-trivial transformations by simply providing examples.

As a design decision, the rules are learned and applied independently of each other. This avoids the exponential problem of searching for the combination of operations that maximize the coverage over all examples. Obviously, it is impossible to learn a rule with maximal $Cov(\Psi^G_{u,v}, \Lambda^+)$ for any arbitrary string pair (u, v) and Λ^+, considering only information in (u, v). However, in Section 5.5, we will show that the rule learner algorithm produces high coverage rules over real-world transformation tasks, which can be properly selected to transform an unseen string s, correctly.

5.4 Rule Selector Method

In this section, we describe our method to tackle the second learning problem.

Given a transformation model $\Omega(\Lambda^+)$ and a pair of strings (x, y), we model the problem of finding a transformation rule $\Psi^G_{u,v} \in \Omega(\Lambda^+)$, such that $Validity(\psi^G_{u,v}, x, y) = 1$, as a classification problem. Our training data are pairs of string $(u_i, v_i) \in \Lambda^+$ (observations) and rules in $\psi^G_{u_i,v_i} \in \Omega(\Lambda^+)$ (categories). Then given a new string x, the task is to assign a specific rule (category) to it, based on the features extracted from x. We use a *Naive Bayes Classifier* as classifier.

During the training phase, where u and v are available, the set of trigrams (3-grams) of the strings u, u^c, v and v^c, and the frequency of the trigrams of u^c were used as features. Precisely, the set of features can be represented as: $(Trigram(u) - Trigram(v)) \cup (Trigram(u^c) - Trigram(v^c)) \cup freq(Trigram(u^c))$.

A frequency was represented by concatenating the trigram with its frequency value, e.g., for a trigram **ull** with $f(\textbf{ull}) = 2$, the feature was represented as **ull2**.

During the classification phase, where only the string x to be transformed is available, the set of trigrams of the string x and x^c, and the frequency of the trigrams of x^c were used as features. Precisely, this set of features can be represented as: $Trigram(x) \cup Trigram(x^c) \cup freq(Trigram(x^c))$.

Notice that the difference of the set of trigrams used above gives exactly the trigrams that change during a transformation $u \rightarrow v$. During the classification phase, if a string x shares these features (trigrams) that represent a rule m, it is likely that m is the best candidate rule to transform x correctly.

As any other classification task, we assume the user can collect a representative and discriminative training sample to obtain a satisfactory performance of this method. We acknowledge that for specific transformation tasks other machine learning approaches may perform better. However, in Section 5.5, we will show that the effectiveness of the Naive Bayes Classifier is sufficient for our purpose, on average.

5.5 Evaluation

In this section, we investigate empirically three aspects of our method: the coverage of the rules produced by the rule learner, the accuracy of the rule selector, and the learning time of the rule learner. In the end of this section, we compare our algorithm, namely *STransformer*, to a state-of-the-art string transformation method.

5.5.1 Data

In this investigation, four datasets were constructed based on real-world string transformation use cases drawn from data cleaning and spreadsheet processing literature. As we will discuss, these four scenarios show the power of STransformer, which can solve different transformation tasks, requiring a very limited set of examples.

Abbreviations Use Case. Ana is a secretary of an institute and she has a catalog with a few thousands organizations names that need to be converted into their abbreviations. The list contains universities, society groups among others entities. To achieve her task, she copies the names of the organizations into her text editor and tries to build the abbreviations manually; e.g., $Youth\ Hostels\ Association \rightarrow YHA$, $University\ of\ New\ Hampshire \rightarrow UNH$. After transforming a few abbreviations, she realizes that her manual process is not productive and she would like to automate the task. To that, she decides to use STransformer (a feature available on her text editor). She inputs the transformations she has done so far as examples and STransformer transforms the rest of the data.

To simulate this task, we used an open online catalog[9] with 2034 organization names and their abbreviations. The overall task was to learn transformation rules from a few

[9] http://www.betweenthelakes.com/pdfs/organizations.pdf

examples that could generate the abbreviations to the full organizations names, Then, the learned rules were used to transform all organizations names into their abbreviations. Table 5.5 shows the examples selected for this task.

Book Titles Use Case. John is a librarian in charge of publishing a catalog of books acquired by the National Library. He obtained, from the IT department, a list of book titles. He observes the titles of the books have the article shifted to the end of the sentence, e.g. "Cloud, The" instead of "The Cloud". Consequently, he needs to fix them, i.e., he needs to apply a transformation in the titles of the form: *Cloud, The* → *The Cloud*. He needs the task done as soon as possible, and he cannot count upon the IT department. Using STransformer, he can prepare his own data, providing a few manual examples of the desired titles to input to STransformer, without any programming skills.

To simulate this task, we used book titles from the *Book Crossing dataset* [103], which contains 51690 titles from books records where the titles start with an article (e.g. "The", "La", "El", "An"). Then, we shifted the article to the end of the sentence, after inserting a comma and a space. Consequently, the task was to put the title in the original form, i.e., to shift the article to the beginning of the sentence and to remove the additional comma and space. As in the previous scenario, rules were learned from a limited given set of examples (shown in Table 5.6), and then used to transform all titles in the correct conventional form, i.e. *Cloud, The* → *The Cloud*.

Songs Use Case. Mary is a fan of the band R.E.M. She is building a blog and she would like to list all songs of R.E.M at her blog's home page. She finds on Wikipedia 184 R.E.M 's songs. She copies the songs into her editor but she realizes the songs contain also the song duration, which she would like to remove, e.g., *"Shiny Happy People" - 3:44*. To obtain only the title of the songs, she decides to remove the song's duration manually. She expends 3 seconds per songs, taking in total 9 minutes to convert all 184 songs. Later, she learns about STransformer and realized that the task could have been done much faster.

We simulate this use case collecting 184 songs from Wikipedia, and we simulate the task of learning rules that could extract the song titles from the copied text, i.e. "*Shiny Happy People*" 3:44 → *Shiny Happy People*. The task can be achieved with two examples shown in Table 5.7.

Dates Transformation Use Cases. Bob is an weather researcher studying temperatures of in the surroundings of an industrial zone. In his measurements, he uses sensors from two manufactures that output data in different formats. Particularly, one kind of sensor outputs the dates of the measurements in format *"month day, year"*, where *month* is represented by its abbreviated name (e.g. *Jan* 02, 2013). While the other

kind, outputs in the format *"day/month/year"*, where the *month* is represented by its decimal representation (e.g. 02/01/13). Bob would like to transform the output the of first sensor into the format of the second, i.e., $Jan\ 02,\ 2013 \rightarrow 02/01/13$, so he can build an homogeneous report. To that, he uses the STransformer, providing examples of the dates that he needs to transform.

To simulate this task, we used an artificial dataset containing 366 dates in format *"month day, year"*. This dataset is quite homogeneous requiring exactly 12 rules to transform all strings, which map to the twelve-month names and their equivalent decimal representations. Contrarily, all other datasets evaluated are heterogeneous, i.e., there is no logical pattern or obvious regularity that can explain their data beforehand. Particularly, we selected this Dates dataset to show that when there is regularity in the data, the algorithm can learn it with 100% accuracy. The 12 dates used as examples are shown in Table 5.8.

We manually constructed the ground truth for all strings in all datasets. To ensure reproducibility of our results both datasets and the implementation of the proposed algorithm are available for download[10].

5.5.2 Evaluation Metric

To assess the quality of the rule learner algorithm (i.e. the rule coverage), we used the notion of *maximal coverage*, which is the minimal number of example transformations that have to be learned to transform all strings correctly. We evaluated the rule learner with three different configurations of n-grams (E_n) in relative position algorithm (Alg. 9): E_1 (1-gram), E_2 (2-grams) and E_3 (3-grams).

To assess the quality of the rule selector algorithm, the *accuracy measure* was used. It is defined below:

$$accuracy = \frac{\#correct\ transformations}{\#string\ pairs\ in\ the\ ground\ truth} \tag{5.8}$$

Where *#correct transformations* stand for the number of transformations that the rule selector produces correctly; and *#string pairs in the ground truth* stands for the total number of strings that have to be transformed.

[10] https://github.com/samuraraujo/StringTransformation

5.5.3 Rule Coverage

Table 5.2 shows the maximum coverage per task. It indicates that indeed the rule learner algorithm would need a relatively small number of examples to transform all strings in the ground truth correctly, on average. For the Books, Songs and Dates datasets, E_2 requires 157, 2 and 12 examples, respectively. This equates to 0.29%, 1% and 3% of the data, respectively.

TABLE 5.2: Maximal Coverage Per Task

	Abbreviations	Books	Songs	Dates
E_1	439	132	3	12
E_2	305	157	2	12
E_3	326	249	6	12

In total, 305 (for E_2) examples are necessary to obtain maximal coverage in the Abbreviations dataset. This equates to 15% of the data. Although this is a relative large number of examples in our context, a small set of rules (precisely, 7 rules) covers 78% of the strings (i.e., 1577 out of 2034 strings), as can be observed in Table 5.3. It indicates that given the right seven examples, STransformer can learn seven rules that transform 78% of this dataset, correctly. This is quite satisfactory coverage considering that there are precisely 244 cases (12% of the data) that can only be transformed by completely distinct rules, i.e., no general rule could transform them. Examples of these cases are: $Zeta\ Psi \rightarrow ZPsi$ and $Congregation\ of\ the\ Holy\ Ghost \rightarrow CSSp$. Consequently, to achieve 100% coverage with seven or fewer rules is not truly possible in this data due to the lack of regularity in the data. In practice, no method can learn rules from the other available examples that can transform these 244 cases, correctly. Although, this is a very heterogeneous dataset with distinct forms of abbreviating the organizations names, for the cases where there is regularity, the algorithm learns them with acceptable coverage. Notice that if we exclude these 244 outlier cases, then the coverage would be 88%.

TABLE 5.3: The first 7 rules with the highest coverage for the Abbreviations dataset using E_2.

Rule	Covered Examples	Percentage of Data	Cumulative %
1	752	37.0%	37.0%
2	430	21.2%	58.2%
3	233	11.5%	69.6%
4	85	4.2%	73.8%
5	37	1.8%	75.6%
6	22	1.1%	76.7%
7	18	0.8%	77.6%
Total	1577	77.6%	77.6%

Similarly to the Abbreviations dataset, in the Books dataset, many examples, in total 157 (for E_2), are necessary to obtain maximal coverage. However, as observed in Table 5.4, a very small set of rules (precisely, 11 rules) covers a large number of strings (i.e., 51282 strings or 99% of the strings). It confirms that the algorithm is effective in learning transformation rules from a few examples, provided that some regularity is presented in the data.

TABLE 5.4: The first 11 rules with the highest coverage for the Books dataset using E_2.

Rule	Covered Examples	Percentage of Data	Cumulative %
1	37854	73.2%	73.2%
2	4026	7.8%	81.0%
3	2113	4.1%	85.1%
4	1952	3.8%	88.9%
5	1766	3.4%	92.3%
6	1632	3.2%	95.5%
7	570	1.1%	96.6%
8	368	0.7%	97.3%
9	346	0.7%	98.0%
10	331	0.6%	98.6%
11	324	0.6%	99.2%
Total	51282	99.2%	99.2%

In the case of the Songs dataset, the algorithm (with E_2) needs only 2 examples. This shows that it can capture the regularity in the data quite precisely.

In all configurations (E_1, E_2 and E_3), STransformer performs optimally for the Dates dataset, requiring exactly 12 examples, to capture the 12 distinct patterns of dates in the ground truth, i.e., the twelve-month names and their equivalent decimal representations.

Concluding, the results show that the rule learner is able to generate rules with acceptable coverage in a variety of datasets. To obtain the best performance, we recommend using E_2 instead of E_1, because E_2 is more discriminative than E_1, even though slightly more examples were necessary in the Books case.

5.5.4 Rule Selector Accuracy

To verify the rule selector accuracy, examples with maximal coverage were selected for each task; except for the Abbreviations and Books tasks, where only 7 examples with 78% coverage and 11 examples with 99% coverage were selected, respectively. As these examples produce rules that accumulate 100% coverage (78% for the Abbreviations examples and 99% for the Books examples), we generated rules using the selected examples, and then we verified the accuracy of the rule selector in selecting a rule that transforms

correctly a new string s. Table 5.5, 5.6, 5.7 and 5.8 show the examples used to build the rules for the Abbreviations, Books, Songs, and Dates transformation tasks; respectively.

These examples were manually selected, by drawing from the set of examples a few exemplars with evident difference in their features (precisely, strings with different u^c trigrams). Given that the majority of the example strings are quite similar, applying a random selection of examples does not make sense in this setting, because likely we would select examples that produce the same rule; consequently, resulting in low coverage and accuracy. To such an approach to be effective, we would have to consider a large number of examples, which goes against our goal here.

TABLE 5.5: Abbreviations Examples

	u → v
1	American Baptist Foreign Missionary Society→ABFMS
2	American Kennel Club Canine Health Foundation→AKCCHF
3	Congressional Medal of Honor Society→CMHS
4	American Society of Agricultural and Biological Engineers→ASABE
5	Alcoholics Anonymous→AA
6	Army Against War and Fascism→AAWF
7	Army Air Corps→AAC

TABLE 5.6: Book Titles Examples

	u → v
1	Perle, La → La Perle
2	Kill, A → A Kill
3	Angel, The → The Angel
4	Solstice,The → The Solstice
5	Solstice ,The → The Solstice
6	LONG SECRET, THE → THE LONG SECRET
7	CHOCOLATE TOUCH,THE → THE CHOCOLATE TOUCH
8	Hunny, Funny, Sunny Day, A → A Hunny, Funny, Sunny Day
9	street bible, the → the street bible
10	mummies of Urumchi, The → The mummies of Urumchi
11	Mummies of Urumchi, The → The Mummies of Urumchi

TABLE 5.7: Song Examples

	u → v
1	“New Test Leper” - 5:26 → Sitting
2	“Radio Song” (feat. KRS-One) - 4:12 → Radio Song

Table 5.9 shows the accuracy of rule selector for each transformation task using the examples described previously. For the Abbreviations, Books, Songs and Dates task, the accuracy was 74%, 83%, 96%, 100%; respectively. Although, the naive classifier is very sensitive w.r.t the quantity and quality of the examples [104], particularly when a few examples are provided, it performed satisfactorily with high accuracy, on average.

TABLE 5.8: Dates Examples

	u → v
1	Jan 27, 2013 → 27/01/13
2	Feb 27, 2013 → 27/02/13
3	Mar 27, 2013 → 27/03/13
4	Apr 27, 2013 → 27/04/13
5	May 27, 2013 → 27/05/13
6	Jun 27, 2013 → 27/06/13
7	Jul 27, 2013 → 27/07/13
8	Aug 27, 2013 → 27/08/13
9	Sep 27, 2013 → 27/09/13
10	Oct 27, 2013 → 27/10/13
11	Nov 27, 2013 → 27/11/13
12	Dec 27, 2013 → 27/12/13

This is due to the quality of the examples provided, which were manually selected in this investigation. Particularly, in the Books task, the accuracy (83%) was a bit lower than expected. Given that selected Books examples have high coverage, features that were insufficiently discriminative can explain this accuracy. The Abbreviations task had the lowest accuracy (74%); however, the coverage of the examples was only 78%, for this task. It means that its relative accuracy, w.r.t its coverage, was 95%.

Overall, we observed that the rule learner indeed learned rules that were quite general, most of the incorrect transformations observed were due to the rule selector that selected incorrect rules. Concluding, the results demonstrate the feasibility of STransformer for general string transformations tasks. Users without programming knowledge can produce useful string transformations by simply supplying a set of examples. Future extensions could engineer better features to improve the effectiveness of rule selector even further, e.g., using the techniques of [105, 106].

TABLE 5.9: Accuracy of the Rule Algorithm With E_2

Setup	Abbreviations	Books	Songs	Dates
E_2	**74%**	**83%**	**96%**	**100%**

5.5.5 Runtime Cost

Now, we show the linearity of the rule learner algorithm. For this evaluation, we used an Intel Core 2 Duo, 2.4 GHz, 4 GB RAM, using a FUJITSU MHZ2250BH FFS G1 248 GB hard disk. As noted in Section 3, the algorithm is linear which allows it to scale well with the number of examples, considering transformation tasks where a large number of examples is available and necessary. We empirically study the performance of the learner algorithm with an increasing number of input examples. We use subsets

of increasing cardinality drawn from the Books dataset. Fig. 5.3 shows the running time of four runs for various sample sizes. We observe a linear increase in running times as the number of input examples grows, as expected from our analysis in Section 3.

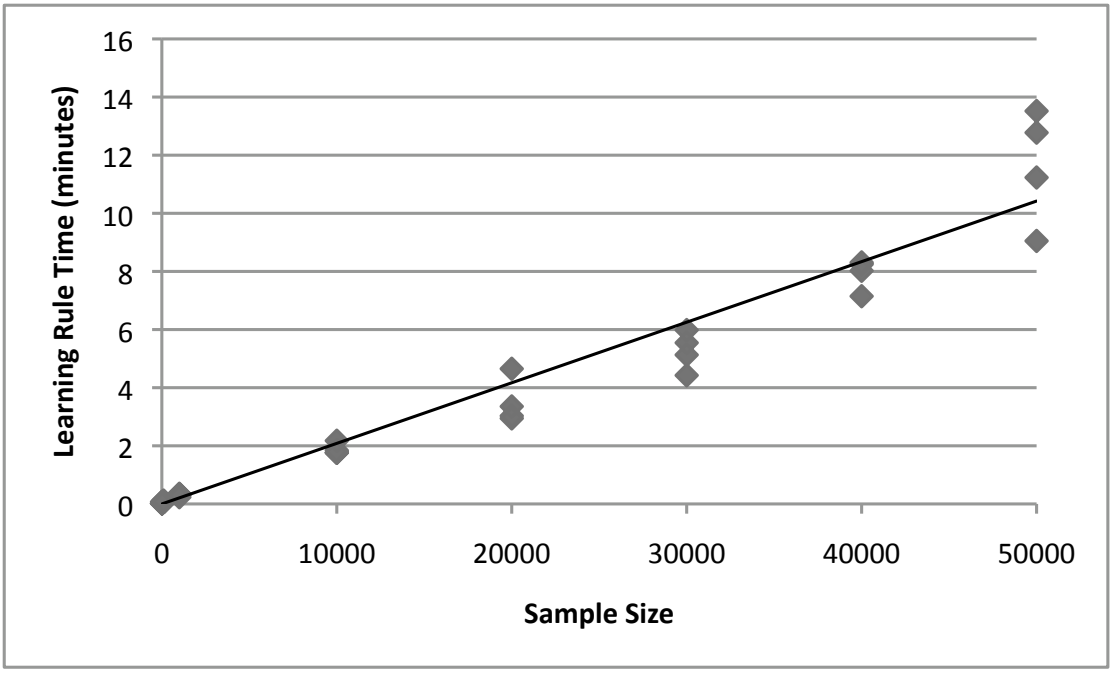

FIGURE 5.3: Learning time varying the sample size for the Books dataset. We considered 4 runs for each sample size.

5.5.6 Performance Comparison

Finally, we compare the STransformer to the state-of-the-art string transformation method proposed in [93, 94] (namely, *FlashFill*). Their algorithm is implemented in the Microsoft Excel 2013, available as a command in the Excel's toolbar. Particularly, we used Excel 2013 version 15.0.4505.1001 in this evaluation.

We measured the accuracy of both systems in solving 4 tasks: Abbreviations, Books, Songs and Dates, which use the same datasets described in the previous evaluations. For each task, we randomly selected a single example from a list of examples that have regular features (e.g., common trigrams) in the data, so that this example transformation had a clear pattern that could be learned. In both systems, a rule was learned from the selected example, and then used to transform the rest of the data.

Particularly, for STransformer, a single rule was learned from this single example using the rule learner and then used to transform the rest of the data. In the FlashFill, this single example was input into Excel's interface as the example of the desired transformation. Then, we apply the FlashFill Excel command to the rest of the data. Finally,

we measured the accuracy of each system for each task. We repeated this process five times for different examples and computed the average accuracy. Table 5.10 shows the results.

TABLE 5.10: Average accuracy per system.

System	Abbreviations	Books	Songs	Dates
STransformer - 1	**39%**	**72%**	**92%**	**8%**
STransformer - 2	39%	**72%**	**92%**	**8%**
STransformer - 3	39%	72%	**92%**	**8%**
STransformer - 4	39%	72%	**92%**	**8%**
STransformer - 5	39%	**72%**	**92%**	**8%**
STransformer (Average)	39%	**72%**	**92%**	**8%**
FlashFill - 1	2%	9%	83%	1%
FlashFill - 2	**74%**	9%	23%	1%
FlashFill - 3	**74%**	**98%**	3%	1%
FlashFill - 4	**74%**	**98%**	83%	1%
FlashFill - 5	**74%**	18%	83%	0.8%
FlashFill (Average)	**60%**	46%	55%	1%

In this evaluation, some tasks reported low accuracy (e.g. Dates) because a single rule could not cover 100% of the universe of transformations; however, the results clearly reflects the ability of each system in producing general rules from a single example. Comparatively, in practice, from a single example, STransformer can transform a larger portion of the data correctly than FlashFill.

Considering the average accuracy among all tasks, STransformer's accuracy was superior to FlashFill's accuracy. Particularly, STransformer's average accuracy per task was superior in the Books (72%), Songs (92%) and Dates (8%) tasks, compared to FlashFill's average accuracy in the Books (46%), Songs (55%) and Dates (1%) tasks. FlashFill's average accuracy in the Abbreviations task was 60% while STransformer's average accuracy in this task was 39%. The results indicate that in some individual runs FlashFill had a better performance than STransformer (e.g. FlashFill - 3, in Abbreviations and Books tasks.); however, STransformer's accuracy was stable, i.e., it did not vary for different examples (different runs), contrarily to FlashFill's accuracy that varied a lot in all tasks for different examples. It indicates that FlashFill depends on very specific examples to produce very good accuracy, contrarily to STransformer that produces equally good accuracy for arbitrary examples.

These results indicate that STransformer produces more general rules than FlashFill, on average. Also, the results expose a strong characteristic of STransformer: it is robust, i.e., its accuracy is less impacted by different examples than FlashFill. This is a desirable property, given that the universe of examples available is diverse in these types of tasks.

Concluding, the edit-distance based transformation rules generalize much better in real data than the grammar-based string transformation approach proposed by FlashFill. STransformer has less than 2000 lines of code[11] and can be easily be integrated into PBE interfaces, such as Microsoft Excel 2013, or into data cleaning tools.

5.6 Related Work

In this section, we discuss the related work and other methods addressing string transformations.

Learning Association Rules. Arasu et al. [51] studied the problem of learning a set of transformation rules given a set of examples matches. In their problem, they assume that a transformation rule maps a sequence of tokens to another sequence of tokens (e.g. $1st\ Ave. \rightarrow First\ Avenue$). These mappings or associations are then used to transform a string into another. A limitation of their approach is that rules cannot be applied over unseen tokens. For instance, the rule $North \rightarrow N$, cannot be used to transform "South" into "S". Moreover, their algorithm needs a large number of examples to generate useful rules. Importantly, they showed that string transformation can be used to normalize strings in record matching tasks, which improves the quality of the matches because it reduces the dissimilarity among the strings. Michelson et al. [96] studied the problem of heterogeneous transformations, which are translations between strings that are not characterized by a single function. E.g. abbreviation, synonyms and acronyms. Addressing the problem of record linkage, Patro et al. [107] proposed an automatic method to extract top-k high quality transformation rules given a set of possibly co-referent record pairs. Tejada et al. [108] addressed a similar problem. Although relevant, these transformations are complementary to the class of transformations that we looked at in our work. We looked into transformations that change the formatting of a string, instead of a mapping based transformations.

Learning Candidate Transformations. Okazaki et al. [97] studied the problem of generating candidate strings to which a given string s is likely to be transformed. They propose a supervised approach that uses sub-string substitution rules as features and score them using an L1 regularized logistic regression model. Then their model selects the best target string t based on the probability of a string t be a transformation to s. As they use a discriminative model, they required a large number of both positive and negative examples [109]. Moreover, the authors state that their model cannot handle changes at phrase/term level, e.g., "Michael Jackson" and "Jackson Michael", which we propose to address in our work.

[11] https://github.com/samuraraujo/StringTransformation

Learning String Transformations from Examples. Gulwani [93] proposed a grammar-based string transformation language to express syntactic transformations. Their method aims to synthesize a desired program including loops and conditions, which together with other functions can express a transformation. Although they designed an efficient algorithm, their method has many performance issues due to the exponential space of transformations that they have to explore. As stated by the author, their algorithm works in practice, but it is not guaranteed to work for all cases. Particularly, we observed in the experiments that their method cannot process strings larger than 255 characters. This method was extended by Singh et al. [110] to support semantic based transformations. Recently, Wu et al. [111] also proposed a gram-based string transformation learner. Compared to these systems, our approach is simpler and can express all transformations listed in their papers, when the right number of examples is given.

String transformations have been studied in many other domains, as well. For instance, Satta et al. [112], introduce an original data structure and efficient algorithms that learn some families of transformations that are relevant for part-of-speech tagging and phonological rule systems. Potter's Wheel [113] is a system that proposes an interactively transformation strategy for data cleaning. They show the evident need of the user interaction in some transformation tasks. Our algorithm can be easily integrated into more complex transformation workflows, as in this process proposed by Potter's Wheel.

5.7 Conclusions

We have presented a novel algorithm to learn string transformation rules from examples. The algorithm is especially useful for non-programmers that in preparation of their data analysis expend a considerable effort on string transformations (e.g. data cleaning). Here, it is presented as a standalone algorithm that can be integrate into data processing tools that support the programming-by-example paradigm, such as Microsoft Excel. The empirical investigation indicates this algorithm can learn transformation rules that generalize for a large number of strings, even when a limited number of training examples is given. Additionally, the comparison against a state-of-the-art string transformation algorithm shows 30% improvement in accuracy (on average), indicating that the proposed algorithm is more effective in learning a transformation from a single example, in the majority of the cases.

As future research, we will investigate alternative machine learning approaches to select the rules when a limited set of features and examples are available. The rule selector proposed in this work is satisfactory and ready to be deployed in real applications; however, it may be improved by incorporating feature selection state-of-the-art techniques.

Overall, the results achieved in this work can facilitate the data processing tasks of millions of non-programmers (and programmers) that need to do string transformations, in a daily basis.

Chapter 6

Exercises on Knowledge Based Acceleration

So far, this thesis has focused on the problem of interlinking resources. This chapter investigates the use of interlinked data in an information filtering task, organized by the information retrieval community at the Text Retrieval Conference (TREC). The Knowledge Base Acceleration task 2012 (TREC-KBA) aims to identify documents from a stream corpora that brings new information about a Wikipedia entity. Given a Wikipedia entity, the task consists of filtering this stream to find documents that are central to this entity. Informally, a central document would be cited on the Wikipedia page, and may trigger an update thereof. We used unsupervised and supervised approaches as retrieval models to detect the document centrality. The unsupervised approach filters the stream based on keywords extracted from DBpedia.org representation of the given target entities. As supervised approaches, we employed two distinct approaches. The first method learns a language model from documents annotated as central, while the second learns relevant discriminative keywords that occur uniquely on the documents annotated as central. In the supervised cases, both the language model and the set of learned keywords are used to rank the documents in the stream corpora. Using those simple approaches, we obtained good results compared to the other participants; however, overall, there is a huge space for improvement in this task.

6.1 Introduction

This chapter describes the retrieval models used and the results obtained in the Knowledge Base Acceleration track in TREC 2012 [114] (TREC-KBA). The TREC-KBA focuses on a single task: filter a time-ordered corpus for documents that are highly relevant to a predefined list of Wikipedia entities. A successful KBA system must do more than resolve the meaning of entity mentions by linking documents to the KB: it must also distinguish centrally relevant documents that are worth citing in the entitys Wikipedia article. Relevance can be interpreted as a fact about an entity that is not described in its respective Wikipedia article, yet. While centrality can be interpreted as a fact that is so important that should be mentioned in the entity's page. Consequently, the task requires the documents to be ranked according to their relevancy for an specific Wikipedia entity.

The total number of documents to be processed and ranked amounts to approximately 400 M, consuming 9 TB of disk space. A sample of documents in these corpora that mentioned a Wikipedia entity has been annotated as garbage, neutral, relevant or central, by human judgment. Particularly, we focused on detecting the last two cases, i.e., we try to detect documents that were central and/or relevant for an entity.

Part of the annotated documents have been provided as training data. We used unsupervised and supervised approaches as retrieval models to detect a document's relevancy and centrality. The unsupervised method (*called Disambiguator*) relies upon keywords extracted from the DBpedia.org representation of the given target entities; as supervised approaches we employed two approaches, one that learned a language model of documents annotated as central (*called LanguageModel*); and another that learned relevant discriminative keywords that occurs uniquely on the documents annotated as central (*called Learning16000*).

Surprisingly, the simplest unsupervised approach, the Disambiguator, performed considerably better than the supervised approaches (LanguageModel and Learning16000). They obtained 0.32, 0.13 and 0.31 F1; respectively. Overall, the Disambiguator did extremely well, obtaining the TOP-3 best result as reported at TREC-KBA 2012 conference.

In the remainder of this chapter, we discuss each of the approaches in details. Section 6.2 introduces the TREC-KBA task. Section 6.3 introduces our approaches. Section 6.4 describes the evaluation and the results obtained. And finally, Section 6.5 concludes this work.

6.2 TREC-KBA Task Overview

The Knowledge Base Acceleration track in TREC 2012 focused on a single task: filter a time-ordered corpus for documents that are highly relevant to a predefined list of entities. It goal is to help accelerate the construction and maintenance of large KBs. This section describes details of the TREC-KBA task.

6.2.1 Data Overview

Corpora overview. For TREC 2012, 29 Wikipedia entities were selected: 27 people and 2 organizations (vide Table 6.1). A stream corpus has been constructed spanning 4,973 consecutive hours from October 2011 through April 2012. It contains over 400M documents augmented with automatic named entity tagging for the $\tilde{4}0\%$ of the documents that has been identified as English. Each document has a timestamp that places it in the stream. This collection has a total size of 9 TB.

Table 6.1: Entities Names

Aharon_Barak	Alex_Kapranos
Alexander_McCall_Smith	Annie_Laurie_Gaylor
Charlie_Savage	Darren_Rowse
Douglas_Carswell	Frederick_M._Lawrence
Ikuhisa_Minowa	James_McCartney
Jim_Steyer	Lisa_Bloom
Lovebug_Starski	Mario_Garnero
Masaru_Emoto	Nassim_Nicholas_Taleb
Rodrigo_Pimentel	Roustam_Tariko
Ruth_Rendell	Satoshi_Ishii
Vladimir_Potanin	William_Cohen
William_D._Cohan	William_H._Gates,_Sr
Basic_Element_(company)	Basic_Element_(music_group)
Bill_Coen	Boris_Berezovsky_(businessman)
Boris_Berezovsky_(pianist)	

Relevancy Judgment. The 29 target entities were mentioned infrequently enough in the corpus that NIST[1] assessors could judge the relevance of most of the mentioning documents (91%). Judgments for documents from before January 2012 were provided to TREC teams as training data for filtering documents from the remaining hours. The results were evaluated against the assessor generated list of citation-worthy documents.

[1] http://www.nist.gov/

6.2.2 TREC-KBA Baseline

TREC-KBA organizers provided a simple reference baseline. This baseline assigns a score in the range [0, 1000] to each document based on a set of filtering tokens. A threshold is used to select the documents that are relevant+central, or central. In this baseline, the score of a document is computed using the equation below:

$$score = \underset{t \in T}{\text{argmax}} \frac{length(t)}{length(entity\_name)} \times 1000 \tag{6.1}$$

where *entity_name* is the name of the entity, as listed in Table 6.1, and T is a set of tokens (an *entity_name* split by "_"), including the *entity_name* itself.

6.3 Approaches

This section describes three the approaches that we have applied to this challenge. The first approach, Prefix-Suffix Learning, namely *Learning16000*, focuses on precision, penalizing the recall. The second, namely *Disambiguator*, focuses on recall, with minimal improvement on precision over the Baseline. The third and last approach, namely *Language Model*, focuses on balancing the precision and recall.

Firstly, we introduce an *entity representation*, which basically is a set of strings used to represent a specific Wikipedia entity. This representation will used later in the three approaches presented.

6.3.1 Entity Representation

Given an entity $e \in E$(the 29 given Wikipedia entities), we represent e as a set of string V_e, defined below:

Definition 6.1 (Entity Representation)**.** The entity representation V_e of an entity e is constructed by the results of the SPARQL[2] query q over the DBPedia[3] representation of an entity e, where q is defined such as:

```
SELECT distinct ?o  WHERE
{ e <http://www.w3.org/2000/01/rdf-schema#label> ?o  .}

UNION
```

[2]http://en.wikipedia.org/wiki/SPARQL
[3]http://dbpedia.org

```
SELECT distinct ?o  WHERE  {
e <http://xmlns.com/foaf/0.1/name> ?o  .}
```

In other words, the set V_e is the set of `labels` and `names` of the entity e in DBPedia, which is a RDF [4] version of Wikipedia. For example, the entity representation for the entity Nassim Taleb [5] is: $V_{NassimTaleb} = \{Nassim\ Nicholas\ Taleb,\ Nassim\ Taleb\}$.

6.3.2 Prefix-Suffix Learning Approach

The purpose of this approach is to focus on the precision of the annotations, penalizing the recall. Basically, the precision can be controlled by identifying central documents using tokens that occur uniquely in documents previously annotated as central. In that way, the number of documents classified as central is reduced, potentially increasing the precision, but consequently penalizing the recall. This intuition that central documents are likely to share the same set of discriminative tokens is exploited as follows.

For each entity e, we learned a set of strings S_e of the form uv and vw that occur in the documents annotated as central (denoted as Δ^+) and does not occur in document annotated as relevant, neutral or garbage (denoted as Δ^-); where $v \in V_e$, u is prefix of v of size K and w is a suffix of v of size K (in characters). We vary K in the interval $[1, .., K]$, considering at most $2K$ different strings per unique $v \in V_e$.

Example: Consider the entity Nassim Taleb, its set $V_{NassimTaleb} = \{Nassim\ Nicholas\ Taleb,\ Nassim\ Taleb\}$ and the news document below:

"Among the people we reached out to while reporting this weeks cover story on Rich Marin ***was Nassim Taleb. No****t only is he a well-known talking head (and presumably accessible), but he also got his start at Bankers Trust, just like Mr. Marin."*

For $K = 4$, the set $S_{NassimTaleb} = \{$ *Nassim Taleb, s Nassim Taleb, as Nassim Taleb, was Nassim Taleb, Nassim Taleb., Nassim Taleb. , Nassim Taleb. N, Nassim Taleb. No*$\}$

The algorithm to learn the set of strings S_e is described in the Alg. 11. In the run that we submitted, we used $K = 10$.

During the annotation phase, an arbitrary document D was annotated as central for e if any string in S_e occur in D. In this approach a score of 1000 was assigned to the document in the mentioned case, or zero otherwise.

[4] Resource Description Framework - http://www.w3.org/RDF/
[5] In DBPedia: http://dbpedia.org/page/Nassim_Nicholas_Taleb

Algorithm 11 LearningPrefixSuffixStrings(e, Δ^+,Δ^-, K).

$S_e \leftarrow \emptyset$
$V_e \leftarrow QueryDBPediaLabels(e)$
for all $v \in V_e$ **do**
  for all $d \in \Delta^+$ **do**
    if $d.contains(v)$ **then**
      $S_e \leftarrow S_e \cup prefix(v, d, K)$
      $S_e \leftarrow S_e \cup suffix(v, d, K)$
    end if
  end for
  for all $d \in \Delta^-$ **do**
    if $d.contains(v)$ **then**
      $N_e \leftarrow N_e \cup prefix(v, d, K)$
      $N_e \leftarrow N_e \cup suffix(v, d, K)$
    end if
  end for
end for
return $(S_e - N_e)$

6.3.3 Disambiguator Approach

This approach aims to produce high recall and improve the precision over the Baseline by solving only the cases where the source entities are ambiguous.

Definition 6.2 (Ambiguous Entities)**.** Given two distinct entities e and f, they are ambiguous if the intersection of their entity representations is non-empty, i.e., $V_e \cap V_f \neq \emptyset$.

Basically, to ensure high recall, we select documents using the strings in the entity representation V_e, and then to improve precision over this initial selection, we filter out the documents using an extended entity representation T_e that we describe next.

When two or more entities are ambiguous, it is impossible to decide exactly the entity referred to by only considering strings in V_e. In TREC-KBA 2012, among the 29 entities provided, at least four entities were ambiguous, as observed in the query set:

```
Boris_Berezovsky_(businessman);
Boris_Berezovsky_(pianist);
Basic_Element_(company);
Basic_Element_(music_group).
```

For example, the string *Boris Berezovsky* may refer to a pianist or a businessman. Without context information, we cannot decide which one it refers to.

In order to decide which one of the ambiguous entity an document mention, we contextualize an entity representation using type information extracted from DBPedia representation of this entity. An t*ypified entity representation* is defined such as:

Definition 6.3 (Typified Entity Representation)**.** A typified entity representation T_e of an entity e is the results of the SPARQL query below:

```
SELECT distinct ?c   WHERE  {
e <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o  .
?o <http://www.w3.org/2000/01/rdf-schema#label> ?c. }

UNION

SELECT distinct ?c  WHERE  {
e <http://purl.org/dc/terms/subject> ?o  .
?o <http://www.w3.org/2000/01/rdf-schema#label> ?c. }

UNION

SELECT distinct ?c   WHERE  {
e <http://purl.org/dc/terms/subject> ?z .
?z <http://www.w3.org/2004/02/skos/core#broader> ?o  .
?o <http://www.w3.org/2000/01/rdf-schema#label> ?c. }
```

In other words, this query retrieves a set of strings representing the type of an entity e in its DBPedia representation. The properties `type` and `subject` above define the type of a specific DBPedia representation.

For example, considering the entity *Boris_Berezovsky_ (businessman)*, the instantiation of the query above would be:

```
SELECT distinct ?c   WHERE  {
<http://dbpedia.org/page/Boris_Berezovsky_(pianist)>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o  .
?o <http://www.w3.org/2000/01/rdf-schema#label> ?c. }

UNION

SELECT distinct ?c  WHERE  {
<http://dbpedia.org/page/Boris_Berezovsky_(pianist)>
<http://purl.org/dc/terms/subject> ?o  .
?o <http://www.w3.org/2000/01/rdf-schema#label> ?c. }

UNION

SELECT distinct ?c   WHERE  {
<http://dbpedia.org/page/Boris_Berezovsky_(pianist)>
<http://purl.org/dc/terms/subject> ?z .
?z <http://www.w3.org/2004/02/skos/core#broader> ?o  .
?o <http://www.w3.org/2000/01/rdf-schema#label> ?c. }
```

Then $T_{Boris\_Berezovsky\_(pianist)} = \{pianist, musician, russian\}$.

Given an entity e, the algorithm process the news stream sequentially and for all document D that contains a string $v \in V_e$, we produce a *Jaccard score* between T_e and d_e, where the entity context d_e and *Jaccard* is defined such as:

Definition 6.4 (Entity Context)**.** A entity context d_e of an entity e is a set of tokens of a string uvw in D, where $v \in V_e$, and u and w have at most a length L. In the run that we submitted, we set $L = 400$.

Definition 6.5 (Jaccard Score)**.** Given two sets A and B, the Jaccard score of these sets are:

$$Jaccard = \frac{(A \cap B)}{(A \cup B)}$$

A document D is considered central for an entity e if:

$$Jaccard(T_e, d_e) > 0 \wedge x \in E : x \neq e, Jaccard(T_e, d_e) > Jaccard(T_x, d_e)$$

Alg. 12 describes the process of annotating central documents using this method.

Algorithm 12 DisambiguatorCentralAnnotator(E, D, L).

$CENTRAL \leftarrow \emptyset$
$MATRIX \leftarrow newMatrix(D, E)$
for all $e \in E$ **do**
  $V_e \leftarrow QueryDBPediaLabels(e)$
  $T_e \leftarrow QueryDBPediaTypes(e)$
  for all $v \in V_e$ **do**
    for all $d \in D$ **do**
      if $d.contains(v)$ **then**
        $d_e \leftarrow context(d, e, L)$
        $MATRIX[d][e] \leftarrow MATRIX[d][e] + Jaccard(T_e, d_e)$
      end if
    end for
  end for
end for
for all $d \in D$ **do**
  if $MATRIX[d].max > 0$ **then**
    $CENTRAL \leftarrow CENTRAL \cup [d, MATRIX[d].maxEntity]$
  end if
end for
return $CENTRAL$

6.3.4 Language Model Approach

This approach focuses on balancing precision and recall. To do so, we build a *statistical language model* for each entity $e \in E$ using the training documents annotated as central. Then, we score the documents based on the *perplexity measure* between the entity language model and the document text. We normalize the values between [0, 1000]. This measure may produce some documents with low score; however, we consider all those documents with score > 0 as central. Below we detail this approach.

Definition 6.6 (Entity Language Model)**.** Given an entity e and a corpora Δ^+ of documents annotated as central, we build a statistical language model, a trigram model, LM_e for e over $\bigcup_{d \in \Delta_e^+} d$, where $\Delta_e^+ = \{d | d \in \Delta^+ \ \wedge v \in V_e \wedge d = uvw\}$

In other words, for each entity e we build a trigram language model LM_e over an aggregated document containing all document annotated as central for the entity e.

We have used the *Kyoto Language Modeling Toolkit (Kylm)*[6] to build this language model. The only parameter set in this api was `smoothuni` (it avoids zero probabilities when trigrams do not occur). Alg. 13 describes the process of creating the language model.

Algorithm 13 LanguageModelTraining(E, Δ^+).

```
LM ← ∅
for all e ∈ E do
  V_e ← QueryDBPediaLabels(e)
  for all v ∈ V_e do
    for all d ∈ Δ+ do
      if d.contains(v) then
        Δ_e+ ← Δ_e+ ∪ d
      end if
    end for
  end for
  LM_e ← NGramModel(Δ_e+, 3) #Trigram
  LM ← LM ∪ LM_e
end for
return LM
```

In order to annotate documents in the test corpus as central, we compute the perplexity between a new document and each specific language model.

[6] http://www.phontron.com/kylm/

The perplexity is based on entropy, where the entropy gives a measure of how likely the ngram model is to have generated the test data. Entropy is defined (for a sliding-window type ngram) as:

$$H = -\frac{1}{Q}\sum_{i=1}^{Q} logP(w_i|w_{i-1}, w_{i-2}, ...w_{i-N+1})$$

where Q is the number of words of test data and N is the order of the ngram model. Perplexity is a more intuitive measure, defined as:

$$perplexity = 2^H$$

The perplexity of an ngram model with vocabulary size W varies between 1 and W. Low perplexity indicates a more predictable language. All documents with perplexity < 100 were consider as central.

Alg. 14 describes this process.

Algorithm 14 LanguageModelCentralAnnotator(E, D, LM).

```
CENTRAL ← newMatrix(D, E)
for all e ∈ E do
  V_e ← QueryDBPediaLabels(e)
  for all v ∈ V_e do
    for all d ∈ D do
      if d.contains(v) then
        p ← perplexity(LM_e, d)
        if p < 100 then
          CENTRAL[d][e] ← p
        end if
      end if
    end for
  end for
end for
return CENTRAL
```

6.4 Evaluations and Discussions

This section discusses the results that we obtained in TREC-KBA challenge. Table 6.2 shows the average precision, recall and F1 over all entities for each approached that we evaluated. $F1 = 2 \times \frac{precision \times recall}{precision + recall}$ is the harmonic mean between precision (proportion of correct annotations among annotated documents) and recall (proportion of documents

annotated among all actual documents). To compute F1, the ranking that was submitted to TREC-KBA was ignored, we use as ground truth all annotated documents provided by TREC-KBA.

TABLE 6.2: Precision (P), Recall (R) and F1 for each evaluated approach, w.r.t central documents.

	P	R	F1
Baseline	0.21	0.81	0.30
Disambiguator	0.23	**0.82**	**0.32**
LanguageModel	0.18	0.22	0.13
Learning16000	**0.36**	0.35	0.31

The Disambiguator obtained the best evaluation results when considering F1 (0.32), which was slightly better than the Baseline (0.30 F1). Although, only a small improvement was observed, the Disambiguator increased both the precision (0.23) and recall (0.82) w.r.t the Baseline (0.21 and 0.81). We observed that this improvement can be attributed to the use of DBPedia labels in the entity representations. It confirms our intuition that those labels are more representative to the entities than the strings used in the Baseline (listed on Table 6.1).

The LanguageModel had the worse performance (0.13 F1). The central documents are so diverse that there is no model that can describe successfully all the central documents of an entity. Basically, central documents are highly similar to only few other central documents. Consequently, building a language model for each individual central document, and then ranking the documents based on those individual models may produce a better result. An additional challenge in this approach would be to make it efficient, considering the space of search would increase drastically. As such, the entity-centric approach offers clear advantages over the traditional language-oriented approach.

The Learning16000 approach had a similar F1 as Disambiguator, 0.31 and 0.32; respectively. As expected, Learning16000 had a better precision (0.36) than Disambiguator 0.23, and, as may be expected given the precision-recall trade-off usually observed, a lower recall (0.35 and 0.82; respectively). Such a low average recall and better precision can be attributed to more selective keywords used by Learning16000.

Overall, the Disambiguator had the third best performance on F1 among all TREC-KBA submissions. The best approach, namely *UDInfoKBA_WIKI1* (see [114] for details), obtained 0.42. Surprisingly, this approach used an approach quite similar to that of the Disambiguator. Instead of using the DBPedia type to give context to an entity, they considered inner links in the entity's Wikipedia page; then they ranked the documents according to the number of occurrence of the entities referred by those inner links. Although they obtained a greater performance than our best performance (0.32 F1

Disambiguator), they still obtained a relative low performance, indicating that there is a lot of space for improvement in this task.

In general, we observed that it was quite hard to improve precision while preserving recall, when using only information in the content of the documents. A possible reason for this fact may be that the human annotators have used criteria to judge the centrality of a document based on non-content based features. For example, the annotators may have selected the first news item in a day, or news from authority sources like BBC, giving priority to a specific source.

Despite of all results, clearly, the simple and blind application of state-of-the-art content filtering techniques on this task is not appropriate. To improve the accuracy on this task, the human annotator behavior, i.e. the criteria that they used to define centrality, will need to be better understood, so that such a behavior can be better reproduced in computer programs.

6.5 Conclusion

Concluding, we have participated in TREC-KBA 2012 where we obtained good results relative to the other 10 participants. We used a content based approach to determine whether a document carries new information about an entity. Entity-centric approaches seems adequate to achieve reasonable effectiveness, for example when comparing to a more classic approach based on statistical language modeling. The results indicate that human assessors may also judge the centrality of a document using information that were not only content based. We argue that in order to obtain a significant improvement on this task in the next years, the criteria used by the human assessors to annotated central documents have to the better defined.

Chapter 7

Conclusions

This chapter reflects on the research questions posed in Chapter 1 and discusses the status of each with respect to the results that we obtained. Further, we discuss future research directions.

7.1 Research Questions

7.1.1 Towards Self-Linking Linked Data

The vision of a Self-Linking Linked Data introduced in Chapter 2 seems feasible by deploying results of Chapter 3 and Chapter 4. The components described in this thesis can be immediately deployed to attain a limited form of self-linking behavior. However, we acknowledge that many research questions that need to be answered before we have the proposed self-linking behavior implemented in the Linked Data. In particular, a community effort would be needed to integrate these components in the standard Linked Data tool suite.

7.1.2 SERIMI: Class-based Matching for Instance Matching Across Heterogeneous Datasets

How can we obtain correct matches for a set of source instances when there is no overlapping between the source and target schemas? (Chapter 3)

Firstly, it is important to mention that the cases where there is limited overlapping between schemas occur in the real-world matching tasks. In Chapter 2, we discussed a scenario where only an overlap in the entity's labels occurs, and we showed the proposed

method can satisfactorily solve the instance matching problem in this case. Particularly in this scenario, there were no properties in the source data besides the label of the entities (band's names); consequently, no overlapping of properties (in the schema) could exist to the target dataset (MusicBrainz). We have observed only marginal schema overlap between the source and target datasets on the OAEI 2011 benchmark, especially when considering the New York Times collection. In the person and organization datasets of this collection, only a label and a type property overlapped with the target schemas. This shows that in the reference benchmark in the field, the lack of schema overlap also exists, and it is not an isolated case. These cases help to enforce that the problem that we tackled in Chapter 3 is indeed a relevant problem.

We have shown that it is possible to match instances when the schemas do not overlap, by using newly created method of class-based matching. We observed that this method is more effective when there are instances in the target dataset that share the same or similar label, such as we observed in Geonames and DBPedia. In those cases, and when there is not schema overlap between source and target datasets, the class-based matching is the approach to integrate resources.

The results lead us to conclude that class-based matching should be combined with direct matching to obtain further improvements in instance matching performance (accuracy). Direct matching performs better than class-based matching when there is enough schema overlap in the data, i.e., when the predicates that overlap can identify the correct target matching instance for a source instance. Class-based matching and direct matching complement each other, because no method exists that will perform optimally in all matching tasks. In the future, these two approaches should be combined with newcomer approaches to cover a larger set of matching tasks.

7.1.3 Efficient and Effective On-the-fly Candidate Selection over Sparql Endpoints

How can we obtain candidate matches for a set of source instance in an effective and time efficiency way, by querying a target remote endpoint? (Chapter 4)

The querying solution that we propose in Chapter 4 is on average 10 times faster than the straightforward (but too naive) solution that we considered to this problem in the beginning of our research. This can be observed when comparing the two implementations of SERIMI in GitHub, one using the discussed method[1] and the other a naive

[1] https://github.com/samuraraujo/SondaSerimi

querying approach[2]. The gain in efficiency compared to the alternative approaches discussed in Chapter 4 is not as big as the gain observed in these two implementations, but the performance increase is still considerable. The experimental results indicate that indeed it is possible to obtain candidate matches via querying. As we showed, for the cases proposed in the OAEI benchmark, even if integration is performed only once, the proposed strategy is more efficient than today's default to download the data and process it locally. We have tested this approach in many other real-world scenarios, and it produced comparable results to the one that we discussed in this thesis. The proposed strategy is therefore considered a viable alternative to instance matching over Linked Data.

The utility of our method is dependent on the throughput of the remote endpoints, which can be calculated beforehand. In general, our method should be preferred when we consider integration of a few thousands third-party resources to online remote endpoints with reasonable response time. On our studies on DBPedia and Geonames, two central hubs in the LOD network [115–117] it performed faster, with satisfactory accuracy, than approaches that aims to download and index these datasets for further processing.

We acknowledge that there is a long path ahead of research in this field. For instance, query engines could be optimized to answer to specific matching queries (e.g., those that includes geo similarity), by using specific query operators (e.g. geo-like) and tuning their internal index structures to this end. A related direction to explore is to use the algorithm design on Chapter 5 (discussed below) to learning the correct formatting of string in the target dataset. This would allow to build exact queries as opposed to approximate queries. Exact queries are more precise and more efficient to compute, as we observed in our experiment. However, a deep investigation of this problem is necessary to draw an a convincing conclusion.

7.1.4 Learning Edit-Distance Based String Transformation Rules From Examples

How can we learn string transformation rules from a limited set of examples that can correctly transform a large amount of unseen strings similar to the examples? (Chapter 5)

The results obtained in Chapter 5 demonstrate how a learning approach can infer rules with high coverage from only few unique examples. Partially, the good results presented are due to the distribution of the data in the datasets that we evaluated; and partially the good results are due to the novel algorithm that we proposed. The method works

[2]https://github.com/samuraraujo/SERIMI-RDF-Interlinking

so well because most of the human readable strings, or strings produced by human language, are quite homogeneous. No algorithm can transform data if there is no regularity (pattern) between the data and the given example transformations used to learn the rules. We expect that it will be hard to improve the current coverage of the rule leaner, without drastically compromising its efficiency. In contrary, the rule selector can easily be improved upon, when applying state-of-the-art machine learning techniques to select a specific rule from the set of learned ones.

Concluding, the proposed method can learn string transformation rules from a limited set of examples that can correctly transform a large amount of unseen strings similar to the examples, assuming that the examples are representatives. The algorithm proposed can be easily incorporated into information processing tools such as spreadsheets and data cleaning engines. Microsoft has recently released a version of its spreadsheet tool (Excel) implementing an algorithm to support the tasks discussed in our work. The proposed algorithm can be easily incorporated into open source related tools to provide competitive functionality. Among its applications, we have used this algorithm to clean and normalize data on databases, to extract information from HTML tables and XML files, and to modify text on repetitive string transformation tasks.

7.1.5 Exercises on Knowledge Based Acceleration

How can we design a model of centrality for news documents? (Chapter 6)

The study done in this work indicates that the model of centrality is very subjective to the explicit decisions done by the human annotators. The use of information retrieval techniques or semantics on the modeling of centrality produces marginal improvement to an elementary baseline because it does not capture the decisions of the human annotators.

A significant improvement in this area would require better understanding of the human annotators judgment of centrality. It seems reasonable that new facts that appear for the first time in the news about an entity is relevant for this entity. To determine whether a new fact is central or important for an entity, and worth to be mentioned in Wikipedia, requires more discussion. Concrete questions that may help to understand this issue are: what class of facts are worth mentioning in Wikipedia? Daily habits of an entity are important? Is there any temporal aspect of a fact that can determine its importance? Is there any meta-property (e.g. time, space) about a fact that can be used to determine its importance? Whether the answers to these questions lead to decisions that converge to an human annotator's decisions is also an interesting subject of study. It is hard to

outline whether the issue of determine centrality of a fact is a decision theory discipline or a computer science one.

7.2 Future Research

In this thesis, we studied a variety of topics mostly focused on instance matching of heterogeneous and distributed data. Within each research theme, several perspectives are not fully addressed, and or open issues are worth further investigation.

We start with Chapter 2. The self-linking architecture we proposed requires further research in many areas. Research on federated querying aiming to improve the discovery of datasets by the self-linking engines is necessary. Also, research on self-linking policies is a brand new area to be explored. Unsupervised approaches for instance matching should be reviewed when considering its application in Linked Data. Progress on indexing should be conducted aiming to support the self-linking behavior. For example, as matching queries are quite specific, an index structure could be designed to support a quicker evaluation of these types of queries. Finally, research on SPARQL language and protocol should investigate new primitives to support signaling between datasets in the Linked Data specifically focusing on the self-linking behavior.

In Chapter 3, our results on combining class-based matching and direct matching are preliminary. The results that we obtained rise a new research question: *How can we select the best matching strategy for a matching task?* The benefit of selecting the most appropriate matching strategy is to have gain in efficiency, avoiding unnecessary computation. Efficiency is another area of study, given that at a large scale may make sense to consider pruning strategies. For example, avoiding computing scores for candidate instances that are identified as false-positive, early in the process.

In Chapter 4, we concluded that exact queries are faster than other queries but do not work in all cases, mainly due to variations on the string formatting between the source and target instances. This rises a new question: *How could we formulate exact instance matching queries?* The challenge is two-fold. First, find a function that transform a source string (instance label) to the correct target format is problematic. As data are heterogeneous many formats may exist; consequently, how to select the correct one is an issue to be investigated. Second, there is no guarantee that the time to learn the transformation functions plus the time to execute the exact queries will be faster than the time to execute the method proposed in Chapter 4. The efficiency of this new approach would need further investigation. Due to the expected improvements that it may bring, this is definitely an interesting direction to be explored.

In Chapter 5, we obtained promising results using the proposed algorithm. Considering that this algorithm has a large range of applications, the identification of a rule selector method that performs optimally to a specific data collection is an interesting research area.

In Chapter 6, we studied the particular problem proposed by Knowledge Based Acceleration of TREC 2012. The research area of building a model of centrality for a new stream is still in its childhood. It would be interesting to investigate by interviewing human annotators how they define centrality. Further investigation of a centrality model based on these reported aspects would be of great interest for this community.

Overall, the results in this thesis merit further investigation on how to adapt the proposed components into the Linked Data. Also, our findings prompt the question whether instance matching has a different nature in Linked Data settings. Our studies and results indicates that it does, which might stimulate further research into the purpose of instance matching for the Semantic Web.

This thesis sheds more light on instance matching on heterogeneous and distributed data, broadens its relevance for Linked Data and opens up a range of topics for future research.

Appendix A

Jaccard Vs. FSSim

Amos Tversky [55] proposed the *ratio model* as a measure of similarity between two sets A and B. The ratio model is given below:

$$S(A,B) = \frac{|A \cap B|}{|A \cap B| + \alpha|A - B| + \beta|B - A|}, \alpha, \beta \geq 0 \tag{A.1}$$

The parameters α and β balance the weight of the differences in the equation. This equation normalizes the similarity so that S is between 0 and 1. We can show that this model generalizes several set-theoretical models of similarity proposed in literature. If $\alpha = \beta = 1$ it reduces to the Jaccard coefficient, i.e. $Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|}$:

Proof. Replacing $\alpha = \beta = 1$ in E.q. A.1, we have $S(A,B) = \frac{|A \cap B|}{|A \cap B| + |A - B| + |B - A|}$. Then, using the identity $|AUB| = |A \cap B| + |A - B| + |B - A|$, we have: $\frac{|A \cap B|}{|A \cap B| + |A - B| + |B - A|} = \frac{|A \cap B|}{|A \cup B|} = Jaccard(A,B)$ □

It is easy to show that when FSSim is replaced by Jaccard it violates the Theorem 3.7.

Proof. By counterexample: Let $|A \cap B| = 20$ and $|A \cap C| = 10$, and let $|A \cup B| = 40$ and $|A \cup C| = 20$. Then $|A \cap B| > |A \cap C|$ but $\frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap C|}{|A \cup C|} \Rightarrow \frac{20}{40} = \frac{10}{20} \Rightarrow Jaccard(A,B) = Jaccard(A,C)$. □

Now we proof that Theorem 3.7 is valid for FSSim(A,B).

Lemma A.1. *If* $|A \cap B| > 0$ *then* $\frac{|A-B|+|B-A|}{2|A \cup B|} < 1$

Proof. Proof of Lemma A.1: If $|A \cap B| > 0$ then $|A - B| + |B - A| < |A \cap B| + |A - B| + |B - A| < 2(|A \cap B| + |A - B| + |B - A|)$. Applying identity mentioned in Proof A, we have: $|A - B| + |B - A| < 2|A \cup B| \Rightarrow \frac{|A-B|+|B-A|}{2|A \cup B|} < 1$ □

Proof. Proof of Theorem 3.7: If $|A \cap B| > |A \cap C|$ then $|A \cap B| > 0$. Let a positive integer $\delta < 1$ and $\omega < 1$, then $|A \cap B| > |A \cap C| + (\delta - \omega) \Rightarrow |A \cap B| - \delta > |A \cap C| - \omega$. By Lemma A.1 $\delta = \frac{|A-B|+|B-A|}{2|A \cup B|} < 1$ and $\omega = \frac{|A-C|+|C-A|}{2|A \cup C|} < 1$, then $|A \cap B| - \frac{|A-B|+|B-A|}{2|A \cup B|} > |A \cap C| - \frac{|A-C|+|C-A|}{2|A \cup C|} \Rightarrow FSSim(A, B) > FSSim(A, C)$. □

Bibliography

[1] Carina F. Dorneles, Rodrigo Gonçalves, and Ronaldo dos Santos Mello. Approximate data instance matching: a survey. *Knowl. Inf. Syst.*, 27(1):1–21, 2011.

[2] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[3] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *J. Data Semantics IV*, pages 146–171. 2005.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: Scientific American. *Scientific American*, May 2001. URL http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=1&catID=2.

[5] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[6] Peter Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-centric systems and applications. Springer, 2012. ISBN 978-3-642-31163-5.

[7] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, 2010.

[8] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[9] Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.

[10] AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.

[11] Nikolaos Konstantinou, Dimitrios-Emmanuel Spanos, and Nikolas Mitrou. Ontology and database mapping: A survey of current implementations and future directions. *J. Web Eng.*, 7(1):1–24, 2008.

[12] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176, 2013.

[13] André Freitas, Edward Curry, João Gabriel Oliveira, and Seán O'Riain. Querying heterogeneous datasets on the linked data web: Challenges, approaches, and trends. *IEEE Internet Computing*, 16(1):24–33, 2012.

[14] Tim Berners-Lee, J. Hollenbach, Kanghao Lu, J. Presbrey, Eric Prud'hommeaux, and Monica M. C. Schraefel. Tabulator redux: Browsing and writing linked data. In *LDOW*, 2008.

[15] Mathieu d'Aquin, Marta Sabou, Enrico Motta, Sofia Angeletou, Laurian Gridinoc, Vanessa Lopez, and Fouad Zablith. What can be done with the semantic web? an overview watson-based applications. In *SWAP*, 2008.

[16] Samur Araújo and Daniel Schwabe. Explorator: A tool for exploring rdf data through direct manipulation. In *LDOW*, 2009.

[17] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and browsing linked data with swse: The semantic web search engine. *J. Web Sem.*, 9(4):365–401, 2011.

[18] Harry Halpin, Patrick J. Hayes, James P. McCusker, Deborah L. McGuinness, and Henry S. Thompson. When owl: sameas isn't the same: An analysis of identity in linked data. In *International Semantic Web Conference (1)*, pages 305–320, 2010.

[19] Bo Hu and Glenn Svensson. A case study of linked enterprise data. In *International Semantic Web Conference (2)*, pages 129–144, 2010.

[20] Ian Millard, Hugh Glaser, Manuel Salvadores, and Nigel Shadbolt. Consuming multiple linked data sources: Challenges and experiences. In *COLD*, 2010.

[21] Aidan Hogan. Integrating linked data through rdfs and owl: Some lessons learnt. In *RR*, pages 250–256, 2011.

[22] Prateek Jain, Pascal Hitzler, Peter Z. Yeh, Kunal Verma, and Amit P. Sheth. Linked data is merely more data. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.

[23] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *J. Web Sem.*, 14:14–44, 2012.

[24] Giovanni Tummarello and Renaud Delbru. Publishing data that links itself: A conjecture. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.

[25] Samur Araújo, Duc Tran, Arjen DeVries, Jan Hidders, and Daniel Schwabe. Serimi: Class-based disambiguation for effective instance matching over heterogeneous web data. In *WebDB*, pages 25–30, 2012.

[26] Jerome Euzenat, Alfio Ferrara, Willem Robert van Hage, Laura Hollink, Christian Meilicke, Andriy Nikolov, Dominique Ritze, François Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, and Cássia Trojahn dos Santos. Results of the ontology alignment evaluation initiative 2011. In *OM*, 2011.

[27] Matthew Michelson and Craig A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[28] Robert Isele, Anja Jentzsch, and Christian Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *WebDB*, 2011.

[29] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.

[30] Ivan P. Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):pp. 1183–1210, 1969. ISSN 01621459. URL http://www.jstor.org/stable/2286061.

[31] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - a link discovery framework for the web of data. In *LDOW*, 2009.

[32] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag. Executing sparql queries over the web of linked data. In *International Semantic Web Conference*, pages 293–309, 2009.

[33] Olaf Görlitz and Steffen Staab. Federated data management and query optimization for linked open data. In *New Directions in Web Data Management 1*, pages 109–137. 2011.

[34] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference (1)*, pages 601–616, 2011.

[35] Wei Hu, Yuzhong Qu, and Xingzhi Sun. Bootstrapping object coreferencing on the semantic web. *J. Comput. Sci. Technol.*, 26(4):663–675, 2011.

[36] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. Refining instance coreferencing results using belief propagation. In *ASWC*, pages 405–419, 2008.

[37] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[38] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *ISWC*, pages 649–664, 2011.

[39] Shu Rong, Xing Niu, Evan Wei Xiang, Haofen Wang, Qiang Yang, and Yong Yu. A machine learning approach for instance matching based on similarity metrics. In *International Semantic Web Conference (1)*, pages 460–475, 2012.

[40] Andreas Schultz, Andrea Matteini, Robert Isele, Christian Bizer, and Christian Becker. Ldif - linked data integration framework. In *COLD*, 2011.

[41] Robert Isele and Christian Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.

[42] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets. In *LDOW*, 2009.

[43] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in owl-lite. In *ECAI*, pages 333–337, 2004.

[44] Luiz André P. Paes Leme, Marco A. Casanova, Karin Koogan Breitman, and Antonio L. Furtado. Instance-based owl schema matching. In *ICEIS*, pages 14–26, 2009.

[45] Xing Niu, Shu Rong, Yunlong Zhang, and Haofen Wang. Zhishi.links results for oaei 2011. In *OM*, 2011.

[46] Jrme Euzenat, Alfio Ferrara, Christian Meilicke, Andriy Nikolov, Juan Pane, Franois Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Svb-Zamazal, Vojtech Svtek, and Cssia Trojahn dos Santos. Results of the ontology alignment evaluation initiative 2010. In Pavel Shvaiko, Jrme Euzenat, Fausto Giunchiglia, Heiner Stuckenschmidt, Ming Mao, and Isabel Cruz, editors, *Proc. 5th ISWC workshop on ontology matching (OM), Shanghai (CN)*, pages 85–117, 2010. URL http://oaei.ontologymatching.org/2010/results/oaei2010.pdf.

[47] Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. Entity matching: How similar is similar. *PVLDB*, 4(10):622–633, 2011.

[48] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. pages 127–138, 1995.

[49] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[50] George Papadakis and Wolfgang Nejdl. Efficient entity resolution methods for heterogeneous information spaces. In *ICDE Workshops*, pages 304–307, 2011.

[51] Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. Learning string transformations from examples. *PVLDB*, 2(1):514–525, 2009.

[52] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *International Semantic Web Conference (1)*, pages 649–664, 2011.

[53] Yongtao Ma and Thanh Tran. Typifier: Inferring the type semantics of structured data. In *ICDE*, 2013.

[54] Bahram Alidaee, Fred Glover, Gary A. Kochenberger, and Haibo Wang. Solving the maximum edge weight clique problem via unconstrained quadratic programming. *European Journal of Operational Research*, 181(2):592–597, 2007.

[55] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, July 1977. URL http://dx.doi.org/10.1037/0033-295X.84.4.327.

[56] Surajit Chaudhuri, Bee-Chung Chen, Venkatesh Ganti, and Raghav Kaushik. Example-driven design of efficient record matching queries. In *VLDB*, pages 327–338, 2007.

[57] William Chauvenet. *A Manual of Spherical and Practical Astronomy V.II*. Dover, 1960.

[58] Wei Hu, Jianfeng Chen, and Yuzhong Qu. A self-training approach for resolving object coreference on the semantic web. In *WWW*, pages 87–96, 2011.

[59] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[60] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *International Semantic Web Conference*, pages 650–665, 2009.

[61] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[62] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[63] Xianpei Han and Jun Zhao. Structural semantic relatedness: A knowledge-based method to named entity disambiguation. In *ACL*, pages 50–59, 2010.

[64] Andriy Nikolov, Mathieu d'Aquin, and Enrico Motta. Unsupervised learning of link discovery configuration. In *ESWC*, pages 119–133, 2012.

[65] Zhichun Wang, Xiao Zhang, Lei Hou, Yue Zhao, Juanzi Li, Yu Qi, and Jie Tang. Rimom results for oaei 2010. In *OM*, 2010.

[66] Christoph Böhm, Gerard de Melo, Felix Naumann, and Gerhard Weikum. Linda: distributed web-of-data-scale entity matching. In *CIKM*, pages 2104–2108, 2012.

[67] Afraz Jaffri, Hugh Glaser, and Ian Millard. Uri disambiguation in the context of linked data. In *LDOW*, 2008.

[68] Alfio Ferrara, Andriy Nikolov, and François Scharffe. Data linking for the semantic web. *Int. J. Semantic Web Inf. Syst.*, 7(3):46–76, 2011.

[69] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer, and Hermann de Meer. idmesh: graph-based disambiguation of linked data. In *WWW*, pages 591–600, 2009.

[70] Avigdor Gal. *Uncertain Schema Matching*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[71] Bijan Parsia. Querying the web with sparql. In *Reasoning Web*, pages 53–67, 2006.

[72] Cristian Pérez de Laborda and Stefan Conrad. Bringing relational data into the semanticweb using sparql and relational.owl. In *ICDE Workshops*, page 55, 2006.

[73] Sebastian Dietzold and Sören Auer. Integrating sparql endpoints into directory services. In *SFSW*, 2007.

[74] Walter Corno, Francesco Corcoglioniti, Irene Celino, and Emanuele Della Valle. Exposing heterogeneous data sources as sparql endpoints through an object-oriented abstraction. In *ASWC*, pages 434–448, 2008.

[75] Olaf Görlitz and Steffen Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD*, 2011.

[76] Gabriela Montoya, Maria-Esther Vidal, Óscar Corcho, Edna Ruckhaus, and Carlos Buil Aranda. Benchmarking federated sparql query engines: Are existing testbeds enough? In *International Semantic Web Conference (2)*, pages 313–324, 2012.

[77] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, pages 535–544, 2011.

[78] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. ISBN 978-0-13-207148-2.

[79] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), 2009.

[80] Robert D. Carr, Srinivas Doddi, Goran Konjevod, and Madhav V. Marathe. On the red-blue set cover problem. In *SODA*, pages 345–353, 2000.

[81] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999. ISBN 1-55860-552-5.

[82] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2001.

[83] David J. Hand and Keming Yu. Idiot's bayesnot so stupid after all? *International Statistical Review*, 69(3):385–398, 2001. ISSN 1751-5823. doi: 10.1111/j.1751-5823.2001.tb00465.x. URL http://dx.doi.org/10.1111/j.1751-5823.2001.tb00465.x.

[84] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn dos Santos. Ontology alignment evaluation initiative: Six years of experience. *J. Data Semantics*, 15:158–192, 2011.

[85] Samur Araujo, Thanh Tran, Arjen P. de Vries, Jan Hidders, and Daniel Schwabe. Serimi: Class-based disambiguation for effective instance matching over heterogeneous web data. 2012.

[86] Wei Hu, Jianfeng Chen, Gong Cheng, and Yuzhong Qu. Objectcoref & falcon-ao: results for oaei 2010. In *OM*, 2010.

[87] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. Rimom: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.*, 21(8):1218–1232, 2009.

[88] Indrajit Bhattacharya and Lise Getoor. Query-time entity resolution. *J. Artif. Intell. Res. (JAIR)*, 30:621–657, 2007.

[89] Ahmed Metwally and Christos Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *PVLDB*, 5(8): 704–715, 2012.

[90] Daniel M. Herzig and Thanh Tran. Heterogeneous web data search using relevance-based on the fly data integration. In *WWW*, pages 141–150, 2012.

[91] Usama M. Fayyad. Data mining and knowledge discovery: Making sense out of data. *IEEE Expert*, 11(5):20–25, 1996.

[92] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[93] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, pages 317–330, 2011.

[94] Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, 2012.

[95] Tessa Lau. Why programming-by-demonstration systems fail: Lessons learned for usable ai. *AI Magazine*, 30(4):65–67, 2009.

[96] Matthew Michelson and Craig A. Knoblock. Mining the heterogeneous transformations between data sources to aid record linkage. In *IC-AI*, pages 422–428, 2009.

[97] Naoaki Okazaki, Yoshimasa Tsuruoka, Sophia Ananiadou, and Jun ichi Tsujii. A discriminative candidate generator for string transformations. In *EMNLP*, pages 447–456, 2008.

[98] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[99] *Design and Analysis of Algorithms*. Pearson Education Canada, 2009. ISBN 9788177585957. URL http://books.google.nl/books?id=RxvrDwBUzcIC.

[100] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009. ISBN 978-0-262-03384-8.

[101] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. ISBN 0-521-58519-8.

[102] Peter Linz. *An introduction to formal languages and automata (4. ed.)*. Jones and Bartlett Publishers, 2006. ISBN 978-0-7637-3798-6.

[103] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM. ISBN 1-59593-046-9. doi: 10.1145/1060745.1060754. URL http://doi.acm.org/10.1145/1060745.1060754.

[104] Jie Cheng and Russell Greiner. Comparing bayesian network classifiers. In *UAI*, pages 101–108, 1999.

[105] L.C. Molina, L. Belanche, and A. Nebot. Feature selection algorithms: a survey and experimental evaluation. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 306–313, 2002. doi: 10.1109/ICDM.2002.1183917.

[106] Chotirat (Ann) Ratanamahatana and Dimitrios Gunopulos. Feature selection for the naive bayesian classifier using decision trees. *Applied Artificial Intelligence*, 17 (5-6):475–487, 2003.

[107] Sunanda Patro and Wei Wang. Learning top-k transformation rules. In *DEXA (1)*, pages 172–186, 2011.

[108] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.

[109] Guillaume Bouchard and Bill Triggs. The trade-off between generative and discriminative classifiers. In *Proceedings in Computational Statistics, 16th Symposium of IASC*, pages 721–728. Physica-Verlag, 2004.

[110] Rishabh Singh and Sumit Gulwani. Learning semantic string transformations from examples. *PVLDB*, 5(8):740–751, 2012.

[111] Bo Wu, Pedro A. Szekely, and Craig A. Knoblock. Learning transformation rules by examples. In *AAAI*, 2012.

[112] Giorgio Satta and John C. Henderson. String transformation learning. In Philip R. Cohen and Wolfgang Wahlster, editors, *ACL*, pages 444–451. Morgan Kaufmann

Publishers / ACL, 1997. URL http://dblp.uni-trier.de/db/conf/acl/acl97.html#SattaH97.

[113] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.

[114] J. R. Frank, M. Kleiman-Weiner, D. A. Roberts, F. Niu, C. Zhang, C. Re, and I. Soboroff. Building an Entity-Centric Stream Filtering Test Collection for TREC 2012. In *Proceedings of the Text REtrieval Conference (TREC)*, 2012. URL http://trec.nist.gov/pubs/trec21/papers/KBA.OVERVIEW.pdf.

[115] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[116] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - a crystallization point for the web of data. *J. Web Sem.*, 7(3):154–165, 2009.

[117] Georgi Kobilarov, Tom Scott, Yves Raimond, Silver Oliver, Chris Sizemore, Michael Smethurst, Christian Bizer, and Robert Lee. Media meets semantic web - how the bbc uses dbpedia and linked data to make connections. In *ESWC*, pages 723–737, 2009.

SIKS Dissertations

SIKS Dissertatiereeks

==== 1998 ====

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

==== 1999 ====

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

==== 2000 ====

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

==== 2001 ====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

==== 2002 ====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

==== 2003 ====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

==== 2004 ====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politile gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

==== 2005 ====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM)
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

==== 2006 ====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

==== 2007 ====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramrez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

==== 2008 ====

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

==== 2009 ====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU)
and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers (UvT)
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

==== 2010 ====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to
Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Withdrawn

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

==== 2011 ====

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP)
: Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

==== 2012 ====

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN)
A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN)
Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN)
A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU)
Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT)
Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT)
Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU)
Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU)
Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU)
On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT)
A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA)
Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA)
Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE)
Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM)
Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD)
Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD)
Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

==== 2013 ====

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati(TUD)
Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT)
Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves(CWI)
The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT)
Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk(VU)
Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN)
Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige(UvT)
A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras(TUD)
Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian(VU)
Knowledge-driven Migration to Services

2013-13 Mohammad Safiri(UT)
Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)
Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU)
Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD)
Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA)
Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)
Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic

2013-23 Patricio de Alencar Silva(UvT)
Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)
Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT)
Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT)
When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT)
Listening Heads

2013-30 Joyce Nakatumba (TUE)
Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT)
Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN)
Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

2013-33 Qi Gao (TUD)
User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT)
Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA)
Minimal Mobile Human Computer Interaction Promotor: Prof. dr. L. Hardman (CWI/UVA)

2013-36 Than Lam Hoang (TUe)
Pattern Mining in Data Streams

2013-37 Dirk Brner (OUN)
Ambient Learning Displays

2013-38 Eelco den Heijer (VU)
Autonomous Evolutionary Art

2013-39 Joop de Jong (TUD)
A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM)
Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UVA)
Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

2013-42 Lon Planken (TUD)
Algorithms for Simple Temporal Reasoning

2013-43 Marc Bron (UVA)
Exploration and Contextualization through Interaction and Concepts

==== 2014 ====

2014-01 Nicola Barile (UU)
Studies in Learning Monotone Models from Data

2014-02 Fiona Tuliyano (RUN)
Combining System Dynamics with a Domain Modeling Method

2014-03 Sergio Raul Duarte Torres (UT)
Information Retrieval for Children: Search Behavior and Solutions

2014-04 Hanna Jochmann-Mannak (UT)
Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijsen (UU)
Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU)
Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE)
Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD)
Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT)
Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language