Acknowledgements

This dissertation would not be possible without the constant encouragement, high standards, equanimity and constructive feedback of my adviser, Dan Miranker. Since my very first semester as a graduate student, his support has been constant. I look up to him as both a person and teacher, and I can only hope that I do his mentorship justice as I move forward in my young career as a scientist.

I also thank the students (both current and former) in Dan's group, namely Aibo Tian, Juan Sequeda and Lee Thompson for all their help over the years. In particular, Juan's work, his entrepreneurship and his ability to bring people together has always been an inspiration. Aibo's early investment in my development, by allowing me to participate in his projects and always answering my questions patiently, gave me some crucial insights into our field, and academic publishing in general. I also thank my committee members for being gracious and constructive. In particular, I want to thank Risto Miikkulainen for always being there when I needed him, and Ray Mooney for investing his time and thoughts into my dissertation and contributions. I also wish to thank Elaine Rich, under whom I had two very satisfying semesters as a teaching assistant.

Most importantly, without the prayers and best wishes of my family, my journey to an advanced degree would never have commenced. My mother is my bedrock, and her love towards me has been truly unconditional. Without my grandfather's faith in my abilities, I would never have had the confidence to pursue a career in engineering or science. My sister, Divya, is my guardian angel, a role that I consider sacred.

Abstract

Resource Description Framework (RDF) is a graph-based data model used to publish data as a Web of Linked Data. RDF is an emergent foundation for large-scale data integration, the problem of providing a unified view over multiple data sources. An Entity Name System (ENS) is a thesaurus for entities, and is a crucial component in a data integration architecture. Populating a Linked Data ENS is equivalent to solving an Artificial Intelligence problem called instance matching, which concerns identifying pairs of entities referring to the same underlying entity.

This dissertation presents an instance matcher with four properties, namely automation, heterogeneity, scalability and domain independence. Automation is addressed by employing inexpensive but well-performing heuristics to automatically generate a training set, which is employed by other machine learning algorithms in the pipeline. Data-driven alignment algorithms are adapted to deal with structural heterogeneity in RDF graphs. Domain independence is established by actively avoiding prior assumptions about input domains, and through evaluations on ten RDF test cases. The full system is scaled by implementing it on cloud infrastructure using MapReduce algorithms.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Resource Description Framework (RDF) is a graph-based data model used widely to represent and publish structured data (Klyne & Carroll, 2006). The structure in RDF data can be conveniently visualized using *directed labeled graphs*.

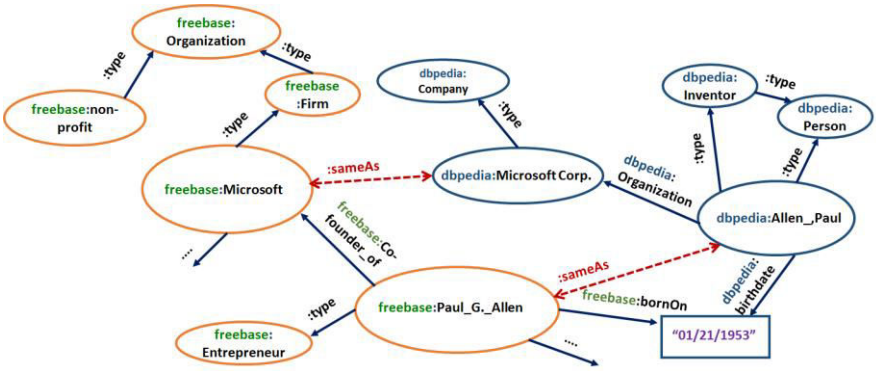


Figure 1.1: *An illustrative*[1] *example of an interlinked RDF graph fragment derived from Freebase and DBpedia, two real-world knowledge bases.* Successful instance matching, defined in the thesis statement, would output the dashed *:sameAs* declarations connecting equivalent entities.

Figure 1.1 illustrates a running example that will be used throughout this dissertation. Nodes in the graph represent entities (e.g. the node with ID *dbpedia:Allen_Paul* represents the entity *Paul Allen* in the knowledge base DBpedia) and edges represent either attributes of an entity (e.g. *"01/21/1953"* is the birthdate of *Paul Allen*) or relationships between two entities (e.g. *Paul Allen* is the co-founder of the company entity *Microsoft*).

[1] The fragment may differ from the actual data in the current versions of Freebase and DBpedia.

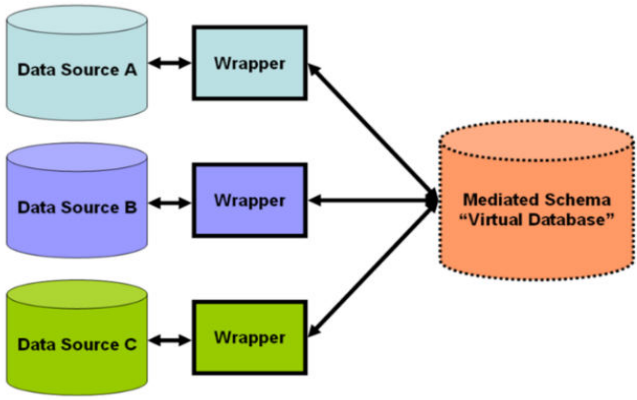


Figure 1.2: *Abstract illustration of the data integration problem.*

Data integration is the problem of providing a unified interface over multiple data sources to an application or end user (Lenzerini, 2002; Doan, Halevy & Ives, 2012). The unification may be virtual or accomplished via wrappers, illustrated in Figure 1.2.

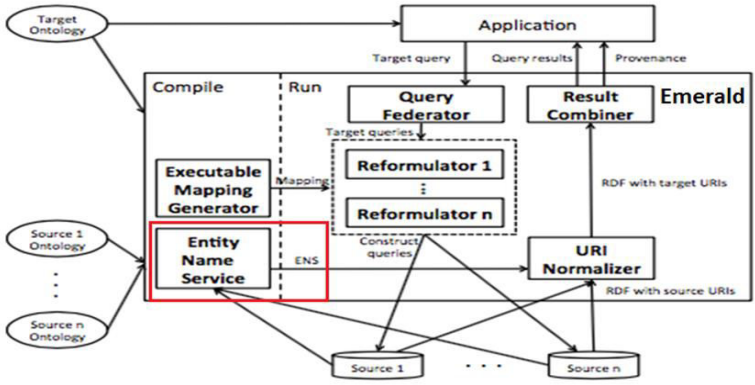


Figure 1.3: *The Emerald data integration system.* The Entity Name Service is a front-facing component that exposes an Entity Name System (ENS).

Data integration has numerous applications (Halevy, Rajaraman & Ordille, 2006), and continues to be an active area of research (Doan, Halevy & Ives, 2012). A full data integration system is complex and requires collaboration across several research areas. Figure 1.3 illustrates the schematic

of *Emerald*, a data integration system currently being developed at the RiBS[2] research group at the University of Texas at Austin. The system is designed for *Linked Data*, which is RDF data that has been published using a set of four subsequently defined principles (Bizer, Heath & Berners-Lee, 2009). In order to complete the system, a component called an *Entity Name System* or ENS must be populated and exposed through an Entity Name *Service* (the red box in Figure 1.3). An ENS is used to serve *instance matching* needs across source databases, and is described further in Section 1.2. This dissertation addresses the population of a Linked Data ENS.

1.1 Linked Data

The scope of data integration has grown in concert with the publishing of new RDF data on the Web (Noy, 2004). Four principles, known as Linked Data principles, are used to stipulate the manner in which such data is published (Bizer et al., 2009).

The first, most basic, Linked Data principle states that entities should be identified using Uniform Resource Identifiers (URIs). The second principle states that URIs should be HTTP-dereferencable, so that they can be accessed using standard Web protocols.

The two principles above do not mention RDF *per se*. The third principle establishes this connection by stipulating that, when dereferenced, a URI should provide useful information about the entity using open standards. RDF is an[3] example of an open standard that has proven to be dominant in the Linked Data community (Bizer et al., 2009; Schmachtenberg, Bizer & Paulheim, 2014). RDF is formally defined in Chapter 2.

The fourth principle, which is of primary concern in this dissertation, is that data should not exist in silos, but be linked to existing datasets. Coupled with the Open Data movement, the fourth principle has had tremendous impact (Auer et al., 2007). For example, consider *Linked Open Data*[4] (LOD), which is the collection of RDF datasets published under an open license (Bizer

---

2 Research in Bioinformatics and Semantic Web.

3 A second example would be the SPARQL query language, which can be used to match patterns given RDF graph inputs. In this dissertation, only the RDF standard will be of interest.

4 http://linkeddata.org/

et al., 2009). According to a recently published study, the LOD cloud currently contains many billions of triples in over 1000 individually published datasets (Schmachtenberg et al., 2014). LOD continues to grow in both variety and volume, and has invited significant research interest in the previous decade.

1.2 An Entity Name System

On the LOD cloud, two nodes may refer to the same *underlying* entity, despite having different names or identifiers. For example, the company *Microsoft* is referred to using two different names (and syntactic IDs) in the two graph fragments in Figure 1.1. For simplicity, such pairs of nodes are referred to as being *equivalent*. Pairs of equivalent entities may be found either within an individual data source, or across data sources.

In Linked Data applications, *instance matching* is defined as the algorithmic problem of finding pairs of equivalent entities (Ferrara, Nikolov & Scharffe, 2013), and then linking them using a special *:sameAs* property, indicated in Figure 1.1. An important application is fulfilling the fourth Linked Data principle. Empirical studies have shown that publishers of Linked Data sources overwhelmingly prefer *:sameAs* links (over other arbitrary links) to establish connections to published data (Schmachtenberg et al., 2014).

More generally, instance matching is known to occur in structured, semi-structured and even unstructured data communities, typically under a plethora of different names. Example names include *entity resolution* (Benjelloun et al., 2009), *deduplication* (Elmagarmid, Ipeirotis & Verykios, 2007), *record linkage* (Elfeky, Verykios & Elmagarmid, 2002), *entity linking* (Moro, Raganato & Navigli, 2014), *co-reference resolution* (McCarthy & Lehnert, 1995), *link discovery* (Ferrara et al., 2013b), the *merge-purge* problem (Hernández & Stolfo, 1995), discovering *entity synonyms* (Chakrabarti, Chaudhuri, Cheng & Xin, 2012), and *hardening soft databases* (Cohen, Kautz & McAllester, 2000). In the rest of the dissertation, the term *instance matching* is uniformly adopted to maintain consistency.

Instance matching is a vital component of data integration, as it is required to populate an Entity Name System or ENS. Earlier, an ENS was defined as a thesaurus for entities and the primary means of serving instance matching needs across data sources (Bouquet & Molinari, 2013).

Example 1.1: Consider again the *Emerald* architecture in Figure 1.3. Suppose the architecture is being employed for an e-commerce application. An e-commerce company (e.g. Amazon) would have a user-facing *target ontology* that provides a unified interface over products, sellers and marketplaces, in order to support applications like faceted search. The actual data could be located in multiple sources, owned either by Amazon or multiple third-party sellers. When a user searches for a particular product (e.g. *Burt's Bees Baby Oil*), the probability is high, owing partly to the dynamic nature of e-commerce offerings, that the product will show up multiple times in the sources. When querying the multiple data sources, an ENS is used for ensuring that the system treats the various mentions of *Burt's Bees Baby Oil* as equivalent. This treatment not only affects the user-experience (by displaying a unique entity only once), but is also vital for correct query answering and aggregation (e.g. calculating the minimum price, and all qualified sellers for the product).

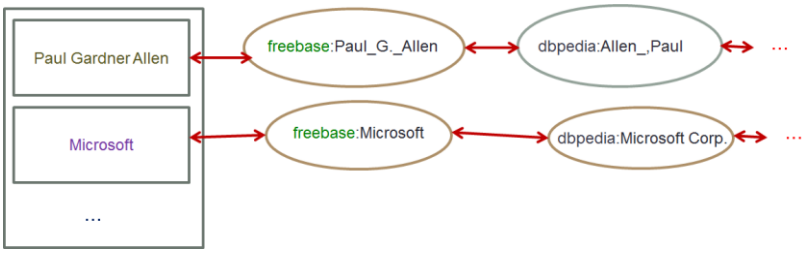


Figure 1.4: *Population of a Linked Data Entity Name System.*

Figure 1.4 shows an ENS that was populated over the entities in Figure 1.1. To correctly populate an ENS, the *:sameAs* edges, or synonyms, between equivalent entities must be located. A previous study estimated that LOD contains many such pairs of equivalent entities that are unidentified (Papadakis, Demartini, Fankhauser & Kärger, 2010). A recent study lent credence to this finding by showing that, despite its growth, LOD is sparse in inter-dataset edges (Schmachtenberg et al., 2014). The vocabulary used by LOD sources are also varied, spanning many different use-cases. In addition, data sources are effectively *schema-free*, meaning that they have little useful metadata associated with them.

1.3 Research Question and Thesis

These findings, together with the ongoing growth of LOD, show that populating a Linked Data ENS is a challenging problem. The specific research question addressed by the thesis can be stated as follows: *what requirements must an instance matcher fulfill in order to populate a Linked Data Entity Name System, and how can it be built?*

Four intuitive requirements are stated. First, the size and growth in LOD data sources suggests that building a feasible instance matcher requires devising solutions that meet requirements of *elastic scalability*, preferably requiring computational resources that increase only linearly in the size of the data.

A second requirement, indicated by the noise and variety in LOD sources, is that of heterogeneity. *Type heterogeneity*, and its complexities, can be understood by referring again to the example in Figure 1.1. The entity *Microsoft* has type *Firm* in the data source Freebase, and type *Company* in the data source DBpedia. On the other hand, *Paul Allen* has type *Entrepreneur* in Freebase, and multiple types (*Inventor* and *Person*) in DBpedia. The problem is further compounded by potential noise in type annotations, and by the presence of a type hierarchy. Such a hierarchy is evident in Figure 1.1. In the Freebase knowledge base, for example, *Firm* is a type of *Organization*. In the Freebase type hierarchy, *Firm* is denoted as a *sub-type* of *Organization*, and *Organization* is denoted as a *super-type* of *Firm*.

To avoid noisy conclusions and wasted comparisons in the presence of many types, a good instance matcher would have to correctly align the pairs of types and then only compare pairs of instances conforming to these types. Not deducing a correct alignment leads to sub-optimal instance matching outputs: for example, the two instances of *Paul Allen* in Figure 1.1 may not get compared (and declared equivalent) in an instance matching pipeline, if neither *dbpedia:Inventor* nor *dbpedia:Person* is aligned with *freebase:Entrepreneur* by a type alignment algorithm.

For similar reasons, the problem of *property heterogeneity* (the matching of property or edge labels) arises once types are aligned. Considering Figure 1.1, an instance matcher would have to deduce that *freebase:co-founder_of* and *dbpedia:organization* are properties that should be aligned for the purposes of instance matching. Intuitively, such an

alignment is necessary in order to extract structural features from the input data.

Given the expense of domain expertise, a third requirement is that a feasible solution should exhibit a high degree of *automation*. This requirement can be met by a non-adaptive system (e.g. by using a fixed similarity metric in an appropriate feature space), but such a system would have little success against the challenges that real-world instance matching applications are known to present (Elmagarmid et al., 2007).

If the system is *adaptive*, the automation requirement can hypothetically be fulfilled by (1) algorithms that rely on self-training (i.e. generating their own training examples), (2) algorithms that are inherently unsupervised (e.g. clustering), or (3) algorithms that use prior results or distant supervision, possibly through a process of transfer learning.

Interestingly, each of the three avenues presented above has found its utility in various applications. In the natural language processing community, (3) seems to be particularly favored, owing to the high quality and completeness of Wikipedia (Cucerzan, 2007), and in more traditional data mining, (2) is favored (Bhattacharya & Getoor, 2006). In data integration, (1) is emerging as the technique of choice (Christen, 2008b; Ma, 2014; Kejriwal & Miranker, 2015c). In Chapter 3, where related work is reviewed, arguments are provided against using either (2) or (3) for populating an ENS in a data integration system.

Given the many *domains*[5] in LOD, a fourth requirement is that the system must also be *domain-independent*, rather than being tuned to the specific needs of individual domains like biomedicine or social media. The issue of domain-independence is largely empirical and is related to, but different from, that of heterogeneity. For example, biomedical datasets contain many different types, which would have to be aligned before instance matching. An instance matcher that performs this task adequately addresses type heterogeneity. If the instance matcher is fine-tuned to the specific needs of the biomedical domain, and fails on other domains, it will not meet the

[5] The *practical* definition of a domain in Linked Data is that a domain is a collection of *related types*. More formally, a domain tends to be defined by a hand-crafted ontology such as the *Gene* ontology (Ashburner et al., 2000). In the Relational Database setting, this is akin to a namespace in which schemas are declared.

domain-independence requirement. Such a matcher relies on too much prior knowledge about the domain to be useful on another domain[6].

Grouping these observations together leads to the following thesis statement in response to the research question stated earlier: *Given the current state of Linked Open Data, a feasible instance matcher must simultaneously fulfill the four requirements of domain-independence, automation, scalability and heterogeneity, referred to henceforth as the DASH[7] requirements*.

1.4 Dissertation

Given the thesis statement, it is natural to investigate if such a system already exists, or can be built with minimal modifications to an existing system. *A priori*, the probability of finding such a system seems to be quite high, since instance matching has been investigated as an Artificial Intelligence problem for over 50 years (Newcombe, Kennedy, Axford & James, 1959). Such an impression would be misleading for two reasons.

First, the growth in data over the last decade has been enormous (Dong & Srivastava, 2012). Any solution that is not amenable to elastic scaling is not likely to be useful on LOD. Unfortunately, most instance matchers proposed in the literature are inherently serial, as argued in Chapter 3.

Second, the condition of simultaneity in the thesis statement indicates that researchers can no longer afford to decouple individual DASH requirements. Problems with such divide-and-conquer approaches can be illustrated with a simple example. Much of the prior instance matching research assumes that the problem of *schema matching* has been perfectly solved before the data is input to an instance matcher (Elmagarmid et al., 2007). The implication is that the sources to be linked have entities belonging to a single type and that the properties describing those types have been *homogenized* in a pre-processing step. A cursory survey of the literature shows that schema matching itself is a difficult problem that continues to be actively

---

6 This also explains why the issue is *empirical*. Since prior knowledge can lead to an unintentional bias in system design, a convincing way of establishing domain independence is by using a single development dataset but conducting multi-domain evaluations (Chapter 5).

7 The acronym is intended as a mnemonic device, and does *not* imply an ordering among the requirements.

researched (Bellahsene, Bonifati & Rahm, 2011). In terms of the DASH requirements, the systems ignore heterogeneity. Extending such systems to account for either type or property heterogeneity is non-trivial.

Concerning implementation, the emergence of cloud services and dataspaces (Jeffery, Franklin & Halevy, 2008) implies that it would be an added boon for the system to be accessed as an on-demand service over the Internet. Although not a requirement *per se*, such a service would have enormous pragmatic benefits for efforts besides data integration that require instance matching as a vital precondition. Three prominent utilities are semantic search (Bouquet & Molinari, 2013), knowledge base population (Dredze, McNamee, Rao, Gerber & Finin, 2010) and knowledge graph identification (Pujara, Miao, Getoor & Cohen, 2013). Similar to data integration, these are broad areas of research[8], but often require solutions to instance matching to be fully functional.

A cloud implementation is challenging both because costs must be kept low, and the implementation can only rely on a standard set of clusters and services. Given the novelty of the cloud, devising such an implementation for instance matching is a relatively open problem.

1.5 Contributions

The primary output of this dissertation is an instance matcher that putatively fulfills the DASH requirements motivated in the thesis statement. A high-level schematic of the instance matcher is illustrated in Figure 1.5. Without loss of generality, two *multi-type* RDF graphs, serialized appropriately[9], are assumed as inputs to the system. The output is a set of *:sameAs* links between entities across all the types. For example, if the two RDF graph fragments in Figure 1.1 are input to the system, the expected output from an ideal execution are the illustrated *:sameAs* links in Figure 1.1.

[8] For example, knowledge base population (KBP) usually requires an *information extraction* step in order to extract structured information from unstructured data sources. This structured information can be organized in RDF (Alani et al., 2003), and be input to the system developed in this dissertation.

[9] Possible RDF serializations are covered in Section 2.1.3.

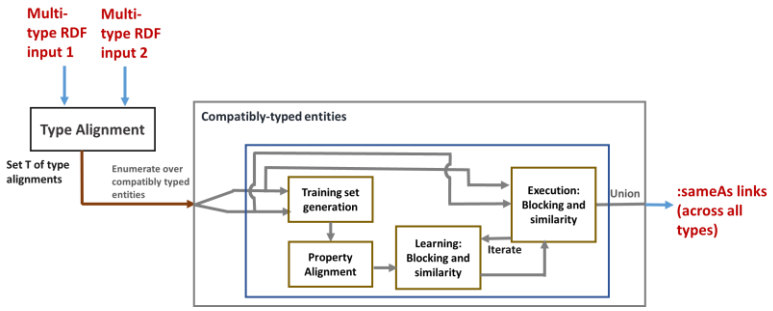


Figure 1.5: *A schematic of the instance matcher presented in the dissertation*. The matcher is designed to fulfill the four DASH requirements of domain-independence, automation, scalability and heterogeneity.

The rationale behind the two-input assumption is stated as follows. If a single graph is provided and needs to be deduplicated, the algorithms described in this dissertation can be modified in a straightforward fashion with the constraint that two instances with the same syntactic ID should never be paired. The assumption also suggests that the main motivation is in finding links between the graphs, not in discovering additional links within individual graphs. If this latter task is of interest, each graph should first be deduplicated, using the system, as a preliminary step.

In a full execution of the schematic in Figure 1.5, the first step is to resolve type heterogeneity by performing *type alignment*, defined as the problem of determining (in a sense that will be made more precise in Chapter 4) pairs of types that are closely *semantically related* to each other. In Figure 1.1, two such alignments exist (between *dbpedia:Inventor* and *freebase:Entrepreneur*, and also *dbpedia:Company* and *freebase:Firm*). The notion of semantic relatedness is analogous to that of *relevance* measures in the *Information Retrieval* (IR) community (Salton & McGill, 1986). In Chapter 4, it is shown that simple, unsupervised techniques inspired by IR algorithms work well for type alignment. Such alignments are akin to constraining the scope of the problem: later algorithms only process *compatibly typed* entities from the overall graph inputs.

A core contribution of this dissertation is an unsupervised algorithm called a *training set generator* (TSG), which uses a combination of fast, intuitive heuristics to output a (possibly noisy) seed training set that can be used to bootstrap the learning process for finer-grained tasks. As detailed in

Chapter 3, addressing the automation requirement turns out to be a challenging bottleneck in instance matching architectures. Specifically, *training examples* are difficult to locate owing both to data sparsity, as well as the dynamic, metadata-poor nature of Linked Open Data (Papadakis et al., 2013). The approach and its empirical viability are both detailed in Chapter 5. To the best of our knowledge, this dissertation presents the first RDF-based TSG that can be usefully employed to execute an entire instance matching pipeline in a completely unsupervised fashion. In Chapter 8, we illustrate a MapReduce-based implementation of the algorithm as an auxiliary contribution.

Once generated, the training examples are used for resolving property heterogeneity by adaptively generating a set of *property alignments*. In principle, property alignment is similar to type alignment, but turns out to be a finer-grained problem with some specific requirements[10] that must be fulfilled in order for later steps in the pipeline to be successful. Simple approaches, and even a relatively sophisticated baseline from the schema matching literature, are shown to fall short empirically of fulfilling these requirements (Section 6.2).

As a second core contribution, we present a parameter-free property alignment algorithm in Chapter 6 that uses an intuitive approach to fulfill the requirements in a domain-independent fashion. The property alignments thus output are used by later machine learning algorithms to extract *structural features* that prove useful in discriminating duplicates from non-duplicates.

The next step, *blocking*, is a preprocessing step that uses a function, called a blocking key, to cluster approximately similar (and compatibly typed) entities into overlapping blocks. Only entities that share a block are paired and considered candidates for further comparison in a *similarity* step (Christen, 2012b). This is in contrast to a naïve one-step similarity approach that compares every entity in one dataset with every entity in the other dataset and entails quadratic complexity. Traditionally, the similarity step was more heavily researched than blocking, with current state-of-the-art work framing the problem as binary machine learning classification (Elmagarmid et al., 2007; Köpcke, Thor & Rahm, 2010).

---

[10] One of these requirements, discussed in Chapter 6, is that the property alignment must have high *recall* (with respect to a manually determined ground-truth) in order for later steps in the pipeline in Figure 1.5 to execute successfully.

In the last ten years, blocking has become an intensely studied problem, in part because of the growth of large, heterogeneous datasets (Christen, 2012b; Papadakis et al., 2013). In the Relational Database (RDB) community, the problem of adaptively learning blocking keys from training data is well-studied. A particular class of blocking keys, called *Disjunctive Normal Form* (DNF) blocking keys, is known to have some excellent theoretical and empirical properties (Michelson & Knoblock, 2006; Bilenko, Kamath & Mooney, 2006).

As a third core contribution, we present both formalism and a learning algorithm for learning and executing DNF blocking keys on heterogeneous RDF data in Chapter 7. Prior to the work described in this dissertation, DNF blocking keys could only be learned and executed on homogeneous RDBs. We show that the DNF blocking keys learned using the generated training set and the property alignment as inputs often outcompete a state-of-the-art RDF blocking algorithm. In Chapter 7, the features and specific machine learning methodology used for the similarity step are also detailed.

Upon execution of blocking and similarity, the *:sameAs* links output by the overall system in Figure 1.5 are collected and can be processed further by upstream applications. In the context of this dissertation, the assumption is that the *:sameAs* links will be used for populating a Linked Data Entity Name System. Depending on the architecture of the data integration system, the ENS may be physically materialized, or accessed virtually through a set of Application Programming Interfaces (APIs). In the *Emerald* architecture illustrated in Figure 1.3, the Entity Name Service is an example of such an API. Although the engineering details behind constructing an ENS are important, the key assumption made by all implementations is the availability of a set of *:sameAs* links. The scope of this dissertation is limited to the system illustrated in Figure 1.5.

Finally, in support of the scalability desiderata discussed in Sections 1.3 and 1.4, the system in Figure 1.5 is implemented both serially and in MapReduce. In Chapter 8, we illustrate MapReduce-based implementations of the algorithms described in Chapters 4-7. These implementations are executed in public cloud infrastructure (HDInsight clusters on Microsoft Azure) using modest resources, and are found to scale even for datasets with millions of entities.

Chapter 2: Background

Instance matching has been researched for at least 50 years in both the structured and unstructured data communities in a variety of methodological contexts (e.g. rule-based vs. statistical approaches) (Newcombe, Kennedy, Axford & James, 1959; Elmagarmid, Ipeirotis & Verykios, 2007). An exhaustive treatment of this research is beyond the scope of this dissertation. Instead, this chapter is limited to two goals. First is a synthesis of common trends that have emerged over the last 50 years. Despite the diversity of research, there is widespread consensus on a number of issues, including the abstract workflow of an instance matcher. The second goal is an exposition of important differences that have emerged over 50 years. As will be shown, these differences tend to be algorithmic, rather than conceptual, and represent a natural evolution of the field over 50 years.

Note that specific systems are *not* critiqued in this chapter. Chapter 3 is exclusively dedicated to discussing related work from the lens of the thesis requirements. Instead, the motivation is to provide a contextual background for the rest of the dissertation.

2.1 Structured Data Models

In this dissertation, the primary data model is assumed to be the structured *Resource Description Framework* (RDF) model (Klyne & Carroll, 2006). The *Relational Database* (RDB) model is also important for historical reasons, given that much of the instance matching literature has traditionally been confined to the RDB community (Elmagarmid et al., 2007). In multiple contexts, research in the RDB community has been productively utilized to solve a compatible problem (e.g. query optimization) on RDF graphs (Angles & Gutierrez, 2005; Sequeda & Miranker, 2013; Sahoo et al., 2009). This suggests that an interesting synergy exists between the two models, and neglecting the RDB model risks not making good use of this synergy.

2.1.1 Resource Description Framework (RDF)

Resource Description Framework (RDF) is a graph-based data model. An RDF graph comprises a *set* of triples.

Definition 2.1 **(RDF triple)** Given three disjoint sets of I, B and L, of Internationalized Resource Identifiers (IRIs), abstract identifiers and literals respectively, a *triple* in the Resource Description Framework (RDF) data model is a 3-element tuple $(subject, property^{11}, object)$, where $subject \in I \cup B$, $property \in$ I and $object \in I \cup B \cup L$. The triple is referred to as an *RDF triple*.

Visually, a triple represents an edge in a directed, labeled graph. Per the first Linked Data principle (Section 1.1), all IRIs used in RDF data sources published as Linked Data must be Uniform Resource Identifiers (URIs), a strict subset of IRIs (Bizer, Heath & Berners-Lee, 2009). In particular, abstract identifiers are not used for representing Linked Data. In the rest of the dissertation, all non-literals in an RDF graph are necessarily assumed to be URIs.

In Chapter 1, Figure 1.1 illustrated an example of two interlinked RDF graph fragments sourced from *DBpedia* and *Freebase*. The following conventions are adopted in visualizing the data. First, oval nodes are used to represent URI subjects and objects, while rectangular nodes are used to represent literal objects. By convention, URI elements are represented using a prefix followed by a colon and an identifying string. The prefix is typically used to represent the *namespace* or vocabulary of the entity. In this dissertation, the prefix is used to indicate the *source* containing the entity of discourse. For example, the URI *freebase:Microsoft* in Figure 1.1 indicates that the entity is from Freebase. Empty prefix strings indicate a globally applicable vocabulary. Such strings are typically used for representing properties with special semantics (e.g. *:sameAs* and *:type* in Figure 1.1).

RDF is the data model used for publishing Linked Data, but in the full Semantic Web technology stack, it is also the basis for representing *RDF Schema* (RDFS), and the *Web Ontology Language* (OWL) (Allemang & Hendler, 2011). The RDFS standard is described as a semantic extension of RDF. It provides a convenient data modeling vocabular*y* that can be used to

---

[11] *Predicate* is sometimes used in place of *property*; *propert*y is uniformly used in this dissertation.

publish a metadata-level schema for an RDF dataset. OWL is a semantic markup language primarily used for representing and publishing ontologie*s* (McGuinness & Harmelen, 2004). Ontologies contain more detailed metadata information (e.g. functional constraints) than simple RDFS schema. OWL also provides various reasoning capabilities over ontologies.

As the principal concern is the RDF data itself, and not the associated metadata, the full details of both RDFS and OWL are beyond the scope of this dissertation. In Linked Open Data, both missing and shallow schemas are common (Schmachtenberg, Bizer & Paulheim, 2014). Namely, Linked Data is *roughly schema-free*[12], meaning that any feasible algorithm, whether for instance matching or not, cannot rely on a detailed metadata-level information set (Papadakis, Ioannou, Palpanas, Niederée & Nejdl, 2013).

2.1.2 Relational Database (RDB) Model

The Relational Database (RDB) model is a highly structured tabular model with constraints and specifications formally based on first-order logic (Codd, 1970). The model is accompanied by a Relational Algebra that forms the underlying basis for expressive query[13] languages like the Structured Query Language (SQL) (Date & Darwen, 1993).

Formally, the schema S' of an RDB can be defined as a set of relation names. Each name is associated with a list of attributes. An RDB instance S associates, with each relation name $R' \in S'$, a set R of records. Although technically a set, R can be visualized as a table, and each of its attributes can be visualized as a column in the table. The visualization can be extended by imagining an RDB as consisting of a set of tables, with directed edges indicating constraints (e.g. foreign keys) both within and between tables. A more formal visualization can be achieved through logical *Entity Relationship* models (Chen, 1976).

---

12 Some schema information is usually available; hence, the qualification *roughly*. For example, the domain (e.g. social media) of a dataset is typically known, and type declarations (e.g. *freebase:Microsoft :type freebase:Firm* in Figure 1.1) are often available for many LOD entities.

13 It may seem strange that querying was not described in the context of RDF. While a SQL-like language, SPARQL (Quilitz & Leser, 2008), exists for querying RDF data, there is no explicit requirement (per Linked Data principles) to provide a SPARQL-processing endpoint over published data.

2.1.3 Serializing RDF Data

For many problems, research on tabular data models (of which RDBs are the most noted example) far predates the relatively recent RDF data model. While the two kinds of models are visualized rather differently, there is precedence to believe that research on a particular problem in the tabular community provides insight on a similar problem in the RDF community. As an example, the Ultrawrap system uses RDB query optimizers to optimize SPARQL queries issued over RDB sources (Sequeda & Miranker, 2013). There is also a standard for mapping RDBs to RDF sources (Sahoo et al., 2009).

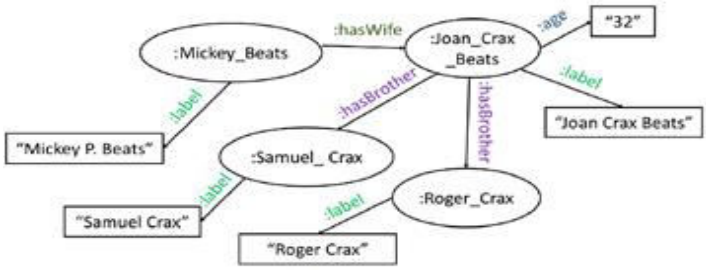
Concerning instance matching, this close connection can be exploited by appropriately *serializing* RDF data. Figure 2.1 shows four equivalent ways of representing a small RDF dataset describing members of a family.

The first of these (Figure 2.1a) is a *directed, labeled graph* representation, similar to the example in Figure 1.1. In practice, this representation is used only for visualization purposes.

Figure 2.1b represents the graph as a *set of triples*, and conforms closely to Definition 2.1. Despite not being the most space-efficient representation, the set-of-triples format is widely used for publishing RDF graphs on the Web[14]. This is because it is simple to parse, and can be easily distributed.

The two representations central to this dissertation are the *logical property table representation* and the *NoSQL representation*. Figure 2.1c illustrates a logical property table serialization of RDF data (Kejriwal & Miranker, 2014; 2015a). In logical form, the tables can be thought of as *loosely structured* (Kejriwal & Miranker, 2014). Each unique URI that occurs at least once as a subject (in an RDF triple) in the set-of-triples representation has its own row in the property table. Cells in the table can contain multiple values (using a reserved delimiter, such as semicolon in Figure 2.1c) or no values (indicated through a reserved keyword, such as *null* in Figure 2.1c).

[14] For example, the Freebase dump (over 400 GB in uncompressed form) is available only in this format.



(a) Directed, labeled graph

```
:Mickey_Beats :hasWife :Joan_Crax_Beats
:Mickey_Beats :label "Mickey P. Beats"
:Joan_Crax_Beats :label "Joan Crax Beats"
:Joan_Crax_Beats :age "32"
:Joan_Crax_Beats :hasBrother :Samuel_Crax
:Joan_Crax_Beats :hasBrother :Roger_Crax
:Samuel_Crax :label "Samuel Crax"
:Roger_Crax :label "Roger Crax"
```

(b) Set of triples

Subject	:label	:hasWife	:hasBrother	:age
:Mickey_Beats	"Mickey P. Beats"	:Joan_Crax _Beats	null	null
:Joan_Crax_Beats	"Joan Crax Beats"	null	:Roger_Crax; :Samuel_Crax	"32"
:Samuel_Crax	"Samuel Crax"	null	null	Null
:Roger_Crax	"Roger Crax"	null	null	null

← Property Schema

(c) Logical property table

(d) Set of sets of <key, value-set> pairs

Figure 2.1: *Four equivalent representations of an RDF data source describing members of a family.*

Formally, the table is described by a *property schema*, which maps each property in the set-of-triples representation to its own column. The $subject$ column serves as the key of the table. The logical property table is preferable for serial solutions, since all rows must be able to access the property schema.

Previously, property tables had been proposed as *physical* data structures for storing and querying RDF data efficiently (Carroll et al., 2004). The goal of such data structures is to use underlying Relational Database architectural principles for similar operations on RDF graphs.

Figure 2.1d illustrates a NoSQL representation of the RDF data. Each entity is now represented as a set of $< key, value - set >$ pairs. The representation is similar to that of the logical property table, but the information set of an entity is now self-contained. This is because the labels of the relevant properties are included as keys in each set, independent of their inclusion in other sets. This makes the representation particularly suitable for parallel and distributed processing, as a set of $< key - value\ set >$ pairs can be encoded and parsed in a self-contained XML/JSON-like format, enabling independent distributing and processing of entities.

The argument in favor of using the two representations illustrated in Figures 2.1c and 2.1d is that many existing tabular instance matchers do not rely *heavily* on structural information (e.g. constraints), and would be robust, with few modifications, to the occasional missing[15] value, redundancy or functional violation in a tabular dataset (Elmagarmid et al., 2007). In principle, such systems are also applicable to tabularly serialized RDF data. In practice, the loose structure can cause unanticipated problems, as discussed in later chapters.

This section concludes with a note on *unstructured* data, typically assumed to be free text represented in natural language. Instance matching in the Natural Language Processing (NLP) community is primarily referred to as *co-reference resolution* (McCarthy & Lehnert, 1995). Because of the special nature of natural language compared to more structured data, a co-reference resolution pipeline involves steps that are inapplicable to structured instance matching (e.g. syntactic parsing). For this reason, the two communities have diverged in their techniques (Elmagarmid et al., 2007; Christen, 2012a). Co-

[15] Such robustness is known to be important in real-world instance matching. For example, it is unlikely that, in the case of a customer database, there is a value for every attribute of a customer.

reference resolution is not considered further in this dissertation. An NLP application that is more relevant to the problem studied in this dissertation is knowledge base population, which was briefly discussed (in Section 1.4) as a potential non-data integration application for the system in Figure 1.5.

2.2 Instance Matching

Even in early research, the quadratic complexity of pairwise instance matching was well recognized (Newcombe et al., 1959). Given two data sources D_1 and D_2, represented generically as sets of entities, a naïve instance matcher would evaluate all possible entity pairs. Assuming constant cost per evaluation, the run-time would be $O(|D_1||D_2|)$.

In the rest of this discussion, for two input graphs D_1 and D_2, an entity pair (e_1, e_2) is denoted as *bilateral iff* $e_1 \in D_1$ and $e_2 \in D_2$. Given a collection of entities from $D_1 \cup D_2$, two entities e_1 and e_2 are said to be *bilaterally paired iff* (e_1, e_2) is bilateral.

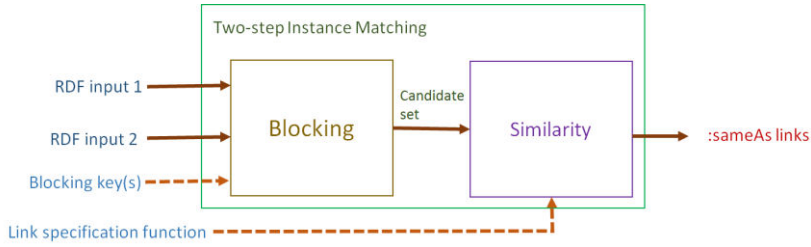


Figure 2.2: *The two-step instance matching workflow.*

To mitigate the quadratic complexity of generating all possible bilateral pairs, a two-step approach is adopted, as illustrated in Figure 2.2 (Christen, 2012a). The first step, *blocking*, uses a many-many function called a *blocking key* to cluster approximately similar entities into overlapping blocks (Christen 2012b). Only entities sharing a block are bilaterally paired and become candidates for further evaluation by a *link specification function* in the *similarity* step (Volz, Bizer, Gaedke & Kobilarov, 2009). The link specification function may be either Boolean or probabilistic, and is used to indicate whether a candidate entity pair represents the same underlying entity.

In the majority of instance matching systems, D_1 and D_2 are assumed to be *structurally homogeneous*[16] (Elmagarmid et al., 2007; Christen, 2012a). That is, they are assumed to contain entities of the same type (e.g. *Firm*), and are described by the same property schema (Figure 2.1c). An important special application of structural homogeneity is *deduplication*, whereby matching entities in a single dataset must be found. In the rest of this section (and in Figure 2.2), structural homogeneity between input data sources is assumed. In Section 2.3, the model is extended to include structural heterogeneity.

2.2.1 Blocking Step

Following the intuitions described earlier, a blocking key is defined as follows.

Definition 2.2 **(Blocking key)** Given a data source D represented as a set of entities, a blocking key K is a *many-many* function that takes an entity from D as input and returns a non-empty set of literals, referred to as the *blocking key values* (BKVs) of the entity.

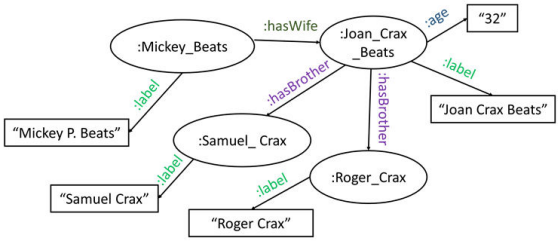
Let $K(e)$ denote the set of BKVs assigned to the entity $e \in D$ by the blocking key K. Given two data sources D_1 and D_2, two blocking keys K_1 and K_2 can be defined using Definition 2.2. Multiple definitions are typically used only when D_1 and D_2 are heterogeneous. At present, a single key (i.e. $K_1 = K_2 = K$), applicable to two structurally homogeneous input data sources, D_1 and D_2, is assumed. Without loss of generality, the literals in Definition 2.2 are assumed to be strings.

Example 2.1 **(Blocking homogeneous datasets)***:* Figure 2.3 illustrates two structurally homogeneous RDF graph fragments describing people. An example of a good blocking key K applicable to the two datasets is $K = Tokens(:label) \cup Identity(:age)$. Applied on an entity e from either dataset, K would return a set of BKVs that contains the tokens in an entity's label, as well as a single string for the age. For example, when applied to the entity $:Joan\_Crax\_Beats$ from the dataset in Figure 2.3a, the output

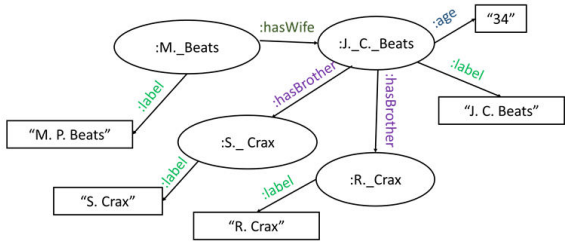
16 The phrase *structural homogeneity* was introduced by Elmagarmid et al. (2007) in their survey of instance matching.

(set of BKVs) returned[17] by the blocking key would be {"$Joan$","$Crax$","$Beats$","32"}. Similarly, when applied to the entity :$J.\_C.\_Beats$ from Figure 2.3b, the output returned would be the set {"J","C","$Beats$","34"}. Since the two BKV sets have a common BKV ("$Beats$"), the entities referenced by the URIs :$J.\_C.\_Beats$ and :$Joan\_Crax\_Beats$ share a block.

Given the single blocking key K, a candidate set C of bilateral entity pairs can be generated by a *blocking method* using the BKVs of the entities. Three prominent blocking methods are described next, followed by the learning of blocking keys.



(a)



(b)

Figure 2.3: *Entities from two structurally homogeneous RDF graph fragments used in Example 2.1.* (a) was introduced earlier in Figure 2.1, while (b) is a second dataset that has the same property schema as (a).

17 This example assumes that a practical implementation of the $Tokens$ function includes a sufficiently expressive set of delimiters. For $Tokens$ to work as expected in the example, the set must include the *whitespace* and *period* delimiters.

Traditional Blocking

Given a blocking key K, an obvious solution is to generate the candidate set C as the set $\{(e_i, e_j) \mid e_i \in D_1 \wedge e_j \in D_2 \wedge K(e_i) \cap K(e_j) \neq \{\}\}$. In other words, e_i and e_j are bilaterally paired *iff* they were assigned a common BKV. The definition of C as a set further implies that e_i and e_j may share multiple blocking key values.

A problem with this so-called *traditional blocking*[18] approach is that of *data skew* (Christen, 2012b). Consider, for example, two entities from a *People* database that are blocked based on the tokens in their last names. Last name frequencies in many countries tend to exhibit skew for some values (e.g. *Smith* in English-speaking countries). A consequence of the skew is that the run-time of the blocking method ends up being roughly proportional to the number of pairs generated by the largest block.

Despite this problem, traditional blocking is often the first line of attack in practical systems (Christen, 2012b; Sadosky, Shrivastava, Price & Steorts, 2015). In recent years, researchers have modified traditional blocking to handle the large blocks that result from skew (Papadakis et al., 2013). A simple method that is easy to implement and difficult to outperform is *block purging*. The premise of the method is that, with a sufficiently expressive blocking key, blocks that are too large can be safely ignored. Such blocks are most likely indexed by BKVs that are equivalent to stop-words. The algorithm takes a purging threshold as an input parameter, and discards all blocks that have more bilateral pairs than this threshold (Papadakis et al., 2013). The threshold may be learned from the data, and is also empirically robust to good default values (Kejriwal & Miranker, 2015c), as investigated in Chapter 7.

Sorted Neighborhood

Another influential blocking method that was fundamentally designed to *guarantee* a bound on the size of the candidate set is the *Sorted Neighborhood* (SN) method, also known as *merge-purge* (Hernández & Stolfo, 1995). The

---

[18] Also called *hash-based blocking* when the blocking key is explicitly constrained to return at most one blocking key value for an input entity. In some papers, hash-based blocking is considered the same as traditional blocking (Christen, 2012b).

algorithm, based on equational theory, works as follows. First, a single blocking key value (BKV) is generated for each entity using a many-one blocking key. Next, the BKVs are used as sorting keys to impose an ordering on the entities. Finally, a window of constant size w is slid over the sorted list. All entities sharing a window are added to the candidate set (Figure 2.4).

ID	First Name	Last Name	Zip	BKV
1	Cathy	Ransom	77111	CR7
2	Catherine	Ridley	77093	CR7
3	Cathy	Ridley	77093	CR7
4	John	Rogers	78751	JR7
5	J.	Rogers	78732	JR7
6	John	Ridley	77093	JR7
7	John	Ridley Sr.	77093	JRS7

Final Candidate Set ($w = 3$):

{(1,2), (2,3), (1,3), (2,4), (3,4), (3,5), (4,5), (4,6), (5,6), (5,7), (6,7)}

Figure 2.4: *Illustration of Sorted Neighborhood for a tabular dataset.* Assuming a sliding window of size 3 ($w = 3$), the final candidate set, generated after the method has terminated, contains eleven pairs of records (referred to by their IDs).

Example 2.2 **(Sorted Neighborhood)***:* Figure 2.4 illustrates a small tabular database describing people. A single blocking key value (BKV) is generated for each entity by concatenating (in order) the initials of tokens present in the first and last names, as well as the first digit of the zip code. The records in Figure 2.4 are sorted using the BKVs as sorting keys. Assuming a sliding window w of size 3, record pairs $(1,2), (1,3)$ and $(2,3)$ are added to the (initially empty) candidate set C in the first sliding iteration, since the records with IDs 1, 2 and 3 share the first window. The window slides forward by one record, and in the second iteration, new record pairs $(2,4)$ and $(3,4)$ are added to C. The method terminates when, at the end of the fifth iteration, the window cannot slide any further.

The sliding window has two implications for candidate set generation. First, entities that do not have the same blocking key value may still get paired. An example of such a pair in Figure 2.4 is $(3,4)$. Second, some entities with the same blocking key value may not get paired. For example, if the sliding window parameter w had been 2 instead of 3 in Figure 2.4, the pair $(1,3)$ would not have been added to the candidate set, despite the two records having the same BKV $CR7$.

Assuming that the window size w is much smaller than the total number of entities[19], Sorted Neighborhood has time and space complexity that is linear in the size of the data. For this reason, it has endured as a popular blocking technique in the instance matching community. Numerous variations now exist (Christen, 2012b). The main differences between these versions and the original version are input data types (e.g. XML Sorted Neighborhood vs. Relational), and various ways of tuning the sliding window parameter (e.g. adaptive vs. constant) for maximal performance (Puhlmann, Weis & Naumann, 2006; Yan, Lee, Kan & Giles, 2007). A major trend has been the proposal of SN algorithms that run on distributed architectures (Kolb, Thor & Rahm, 2012; Ma & Yang, 2015).

A disadvantage of SN algorithms is that they rely on a single-valued blocking key. The authors of the original SN algorithm recognized this as a serious limitation and proposed *multi-pass* SN, whereby multiple blocking keys could be used to improve coverage (Hernández & Stolfo, 1998). For a constant number of passes, the run-time of the original method is not affected asymptotically. Practical scaling is achieved by limiting the number of passes to the number of cores in the processor.

Even in multi-pass SN, each blocking key still remains single-valued[20], precluding the use of expressive blocking keys or even simple token-based set similarity measures that have high redundancy. Extending SN to account for heterogeneous data sources is also non-trivial (Kejriwal & Miranker, 2015d). For this reason, the application of Sorted Neighborhood in Linked Data instance matchers is limited. Instead, the use of an expressive blocking key, combined with a simple blocking method such as traditional blocking with block purging, has gained traction (Papadakis et al., 2013; Kejriwal & Miranker, 2015c), with one possible implementation detailed in Chapter 7.

---

[19] A reasonable assumption, since a window size of $w < 10$ was found to be empirically sufficient (Hernández & Stolfo, 1998).

[20] A second problem occurs when many entities are assigned the same BKV. Given a link specification function and $w \geq 2$, determining an optimal ordering (i.e. contributing maximum duplicate pairs to the candidate set) of same-BKV entities is NP-hard.

Canopies

Clustering methods such as *Canopies* have also been successfully applied to blocking in the context of Relational Databases (McCallum, Nigam & Ungar, 2000; Baxter, Christen & Churches, 2003). The basic algorithm takes a distance function and two threshold parameters $tight \geq 0$ and $loose \geq tight$, and operates in the following way for *deduplication*. First, a seed entity e is randomly chosen from the dataset. All entities that have distance less than $loose$ are assigned to the *canopy* represented by e. Among these entities, the entities with distance less than $tight$ (from the seed entity) are removed from the dataset and not considered further. Another seed entity is now chosen from all entities still in the dataset, and the process continues till all points have been assigned to at least one canopy. The method can be extended to two input data sources by using *every* entity in the smaller data source as a seed entity. The extension has the additional advantage of rendering the original randomized algorithm, deterministic.

Note that, unlike Sorted Neighborhood, *Canopies* does not rely on a blocking key, and instead takes a distance function as input. For this reason, at least one work has referred to it as an *instance-based blocking* method, and distinguished it from *feature-based blocking* methods such as Sorted Neighborhood (Ma, 2014).

Similar to Sorted Neighborhood, several variants of *Canopies* have been proposed over the years, but the basic framework continues to be popular (Christen, 2012b). For example, a nearest-neighbors method could be used for clustering entities, rather than a threshold-based method. In yet another variant, a blocking key can be used to first generate a set of BKVs for each entity, and *Canopies* can then be executed by performing distance computations on the BKV sets of entities, rather than directly on the entities themselves (Christen, 2012b). Because this variant relies on a blocking key, it can no longer be considered an instance-based blocking method.

In the *Canopies* framework, each canopy represents a block. The method has been found to work well with a number of token-based set similarity measures, including *Jaccard* and *cosine similarity* (Baxter et al., 2003). Till recently, when we employed it as a baseline, its performance was not evaluated on loosely structured RDF data. Those evaluations are presented and discussed in Chapter 7.

Learning Blocking Keys

Historically, the blocking key was assumed as a given, typically hand-crafted by a domain expert. In Chapter 1, automation was described as a DASH requirement for populating a Linked Data Entity Name System. In the context of blocking, the need for adaptively learning a blocking key from training data was first addressed by two independent papers (Bilenko, Kamath & Mooney, 2006; Michelson & Knoblock, 2006). Both papers defined a particular class of blocking keys known as *Disjunctive Normal Form* (DNF) blocking keys, and showed that they exhibited excellent empirical performance.

A DNF blocking key can be constructed by starting with a set of *indexing functions* that take a primitive data type as input and return a set of primitives as output[21]. Without loss of generality, *String* is assumed as the only available primitive data type.

Example 2.3 **(Indexing function)***:* An example of an indexing function introduced earlier in Example 2.1 is $Tokens$, which relies on delimiters to tokenize a string (e.g. *"*$J.C.Beats$*"*) into a set of strings (e.g. {"J","C","$Beats$"}).

A *blocking predicate* b_{prop} on entities is now defined by pairing an indexing function h with a property $prop$, and adopting the following semantics[22]. The logical predicate $b_{prop}\ (e_1, e_2)$ is satisfied *iff* the intersection $h_{prop}\ (e_1) \cap h_{prop}\ (e_2)$ is non-empty, where $h_{prop}(e)$ is defined as the set obtained by applying h on the object value of e for property $prop$. Typically, the predicate b *mnemonically* indicates the underlying indexing function h. In an abuse of notation, the property is included in parenthesis. Example 2.4 implements these ideas in practice.

Example 2.4 **(Blocking predicate)** An example of a blocking predicate is $CommonToken(:label)$. $CommonToken$ indicates that $Tokens$ is the underlying indexing function, while $:label$ is the underlying property used by the predicate. Considering the data in Figure 2.3, $CommonToken(:label)$ is satisfied (i.e. returns $True$) for the entity pair

---

21 As in the rest of this section, structural homogeneity is assumed.

22 In the original paper, the predicate b was referred to as a *general blocking predicate*; once paired with a property $prop$, the resulting predicate b_{prop} was referred to as a *specific blocking predicate* (Bilenko, Kamath & Mooney, 2006).

$(:Joan\_Crax\_Beats, :J.\_C.\_Beats)$ since the two entities share a common token ("$Beats$") in their labels.

A Boolean expression, called a *blocking scheme*, can be formed by using these predicates as atoms (Bilenko et al., 2006). For well-defined semantics, negated atoms are disallowed. The expression can be canonically represented in Disjunctive Normal Form (DNF); hence, the blocking scheme is called a DNF blocking scheme. Similar to the blocking predicates, a DNF blocking scheme[23] takes a pair of entities as input. The mnemonic considerations earlier stated also apply.

Example 2.5 **(DNF Blocking Scheme)** Assuming the structurally homogeneous input data sources in Figure 2.3, and continuing the running example in Examples 2.3 and 2.4, an example of a single DNF blocking scheme for both sources is the expression $CommonTokens(:label) \vee (HasExactMatch(:age) \wedge CommonSoundex(:label))$. Two new blocking predicates, named self-explanatorily, are introduced in the DNF expression: $HasExactMatch$ uses the identity function as its underlying indexing function, and returns $True$ if two strings exactly match, while $CommonSoundex$ uses a modified version of $Tokens$ as its indexing function and returns $True$ if two strings share at least one token that *sounds* the same (i.e. have the same *Soundex* encoding).

Logically, a bilateral entity pair is added to the candidate set if it satisfies the scheme. In practice, the indexing functions are used, in combination with a blocking method, to obtain near linear-time performance and avoid data skew[24].

DNF blocking keys offer several advantages over ad-hoc blocking keys and generic distance-based measures. In particular, they are *adaptive* and can be learned from a given training set of duplicates and non-duplicates (Bilenko et al., 2006; Michelson & Knoblock, 2006). Intuitively, learning such schemes can be phrased as an application of approximate *set-covering* solutions (Carr, Doddi, Konjevod & Marathe, 2000). Another advantage is that they are empirically robust, and are known to provide high coverage even

---

23 The difference between a blocking *scheme* and a blocking *key* is fairly pedantic (and unimportant for the discussion in this chapter). Formally, the former is a Boolean expression that takes a pair of entities as input, while the latter operates directly on an entity and yields a set of blocking key values.

24 For that reason, there is no guarantee that *every* bilateral entity pair satisfying the scheme will get added to the final candidate set. For example, if block purging is the underlying blocking method, blocks that are too large will get discarded.

in the presence of noise, with small cost in efficiency (Kejriwal & Miranker, 2013). This robustness is relied upon in Chapter 7, where algorithms for learning DNF blocking schemes from noisy training data are presented. Finally, the basic DNF blocking scheme described in this section can be extended to heterogeneous inputs, outlined in Chapter 7.

Locality Sensitive Hashing: An Alternative to Blocking

Locality Sensitive Hashing (LSH) is a method rapidly gaining in popularity for *approximately* solving the nearest neighbors problem, especially in high-dimensional spaces (Datar, Immorlica, Indyk & Mirrokni, 2004). In this section, a few key intuitions that guide LSH methods are presented, followed by the relevance of LSH to blocking.

Given a distance measure D, an LSH family of hash functions is typically defined by specifying two radii (say r, s with $r < s$) and two probabilities (say p_r, p_s with $p_r > p_s$). A condition for a family to be considered $(r, s, p_r, p_s) - sensitive$ is that any point u that falls within a ball of radius r of a point v, in the feature space on which D is defined, must have the same hash as v with probability at least p_r, per a probability distribution defined on the hash family. A second condition is that if u does *not* fall within the ball of radius s, the probability that the two points have the same hash is less than p_s (Datar et al., 2004).

LSH can be directly applied to the blocking problem in a manner not dissimilar to that of *Canopies* or any other unsupervised blocking method that relies on clustering. Similar to *Canopies*, a distance measure (e.g. *Jaccard*) would have to be assumed, and additionally, a hash family (e.g. *MinHash*) would have to be designed. Given the practical similarities of both methods within the instance matching context, and their theoretical connections to the nearest neighbors problem, it is reasonable to suppose that their success as blocking methods is interlinked. Both *Canopies* and LSH are used as baselines (albeit, for slightly different purposes) later in this dissertation.

Another intriguing application of LSH, suggested by its reasonable success in a large-scale ontology matching application (Duan, Fokoue, Hassanzadeh, Kementsietsidis, Srinivas & Ward, 2012), is in the similarity step of instance matching. The similarity step is described in Section 2.2.2,

where a possible way of using LSH for similarity is also covered. An LSH-based baseline is employed in the evaluations in Chapter 7.

2.2.2 Similarity Step

While the candidate set is expected to contain most, if not all, of the duplicate pairs in the database, it also contains many non-duplicates. A finer-grained *similarity function* (relative to the blocking key) is required to discriminate candidate non-duplicates from duplicates (Elmagarmid et al., 2007). In the Linked Data instance matching literature, this function is commonly denoted as a *link specification function* (Volz, Bizer, Gaedke & Kobilarov, 2009).

Definition 2.3 **(Link specification function)** Given two data sources D_1 and D_2, a link specification function is a Boolean function that takes as input, a bilateral pair of entities, and returns $True$ *iff* the input entity pair refers to the same underlying entity (i.e. is a *duplicate* pair) and returns $False$ otherwise.

The actual link specification function is typically unknown in real-world applications, and must be approximated. Unless otherwise indicated, the phrase 'link specification function' is henceforth assumed to mean the approximated function. The approximated function may be real-valued and return values in the range of [0,1], with a higher score indicating a higher probability of the input pair being a duplicate.

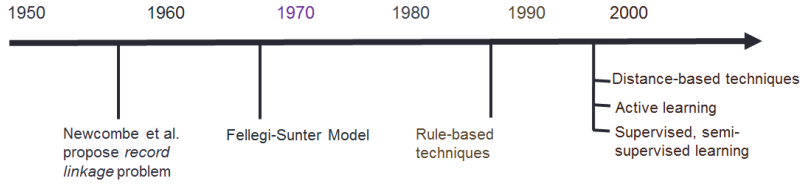


Figure 2.5: *A timeline illustrating the evolution of the similarity step.*

The application of the link specification function L to the pairs in the candidate set C is commonly referred to as the *similarity* step (Elmagarmid et al., 2007; Christen, 2012a). Typically, only the pairs labeled as duplicates (or scored above a certain threshold, if L is real-valued) are output. Due to the longevity of instance matching, numerous similarity techniques have been

researched. A good survey was provided by Elmagarmid et al. (2007), and by Christen in his text on data matching (2012a). Figure 2.5 illustrates the evolution of the field over 50 years.

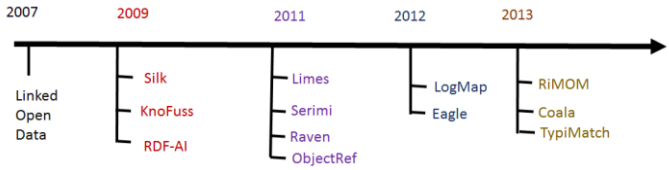


Figure 2.6: *The evolution of the similarity step in Linked Data research.*

In the early days, rule-based approaches were popular, but in the last decade, machine learning has emerged as the dominant paradigm for *learning* an approximate link specification function from a training set of duplicates (positive class) and non-duplicates (negative class). A similar evolution is already taking place in the Linked Data community, where rule-based approaches, such as Silk (Volz et al., 2009), still enjoy support but are being gradually supplanted by adaptive algorithms relying on machine learning techniques such as active learning (Ngomo, Lyko & Christen, 2013). A timeline is presented in Figure 2.6, along with examples of specific systems, several of which are reviewed in Chapter 3. Interestingly, many systems in Figure 2.6 are *hybrid*, and combine a variety of techniques. Some real-world advantages of hybrid algorithms (in the Linked Data community) are explored in Chapter 6.

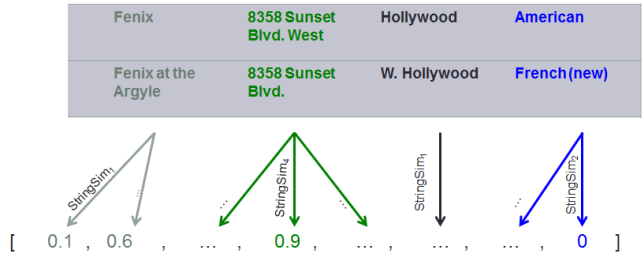


Figure 2.7: *Conversion of an entity pair into a numeric feature vector*. The two entities are represented in logical property table form (Figure 2.1c) and are from a real-world *Restaurants* dataset.

In machine learning-based instance matching, each entity pair in the candidate set is first converted to a numeric *feature vector*. Figure 2.7 illustrates the procedure for the structurally homogeneous case. In Figure 2.7, a candidate pair, comprising duplicate restaurants, is converted to a feature vector using a library of feature functions on each attribute. Given m features and n attributes, the feature vector has exactly mn elements.

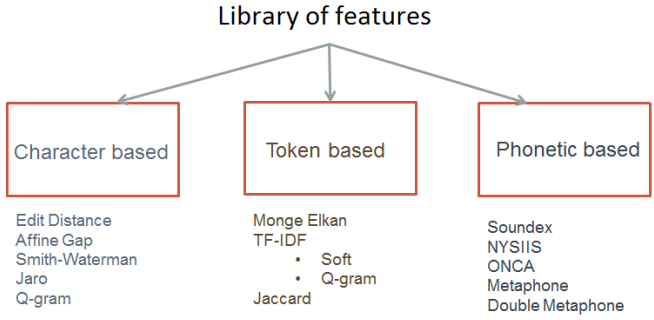


Figure 2.8: *Examples of popular features used by existing instance matchers*. Numeric features, used for computations on dates, currency and other numeric data, tend to be defined in an ad-hoc fashion and are not included in the figure.

Popular features that have been investigated in the instance matching literature include *string, token, numeric* and *phonetic* features (Elmagarmid et al., 2007). Figure 2.8 provides a non-exhaustive taxonomy; full details and a comprehensive evaluation may be found in the text by Christen (2012a).

An alternative[25] way of extracting features is by generating *hashes* using several well-known Locality Sensitive Hashing families (Kejriwal & Miranker, 2015c). According to this model of feature generation, the underlying link specification function can be modeled through a functional combination of various distance measures for which LSH-sensitive families exist. A validation of this model would be consequential as it significantly eases the burden of scalability[26], both in the blocking and similarity steps. In the most general case, the hashes would be used as features, and an appropriate

[25] We acknowledge our anonymous reviewers for this suggestion (Kejriwal & Miranker, 2015c).

[26] Mainly because LSH was designed for large-scale, high-dimensional nearest neighbors applications (Datar et al., 2004).

learner would be used for discovering an explicit functional combination (or rules) for class separation (Chapter 7).

A machine learning classifier is trained on positively and negatively labeled training samples, and is used to classify vectors in the candidate set. Several classifiers have been explored in the literature, with random forest, multilayer perceptron and Support Vector Machine (SVM) classifiers all found to perform reasonably well (Rong et al., 2012; Soru & Ngomo, 2014; Kejriwal & Miranker, 2015c).

Independence of Blocking and Similarity

This section concludes with a note on the *independence* of the blocking and similarity steps. As illustrated in Figure 2.2, the blocking step generates the candidate set, which is further processed in the similarity step. In practice, the two steps can interact: space can be conserved by not storing the candidate set explicitly but classifying pairs as they are generated. However, the assumption still holds that the decisions in the similarity step do not affect blocking.

This independence assumption has been challenged in a small number of applications in recent years (Whang, Menestrina, Koutrika, Theobald & Garcia-Molina, 2009; Papadakis et al., 2013). As just one example, a blocking technique called *comparisons propagation* proposes using the outcomes in the similarity step to estimate the usefulness of a block in real time (Papadakis et al., 2013). The premise is that if a block has produced too many non-duplicates, it is best to discard it rather than finish processing it. By this logic, the cost of processing the block outweighs the gain, at least in expectation.

While such techniques are appealing, their implementations have mostly been limited to serial architectures, owing to the need for continuous data-sharing between the similarity and block generating components (Whang et al., 2009; Papadakis et al., 2013). Experimentally, the benefits of such techniques over independent techniques like Sorted Neighborhood or traditional blocking (with skew-eliminating measures such as block purging) have not been established extensively enough to warrant widespread adoption. The two-step workflow, with both steps relatively independent, continues to be predominant in the vast majority of instance matching research (Köpcke & Rahm, 2010).

2.2.3 Evaluating Instance Matching

The independence of blocking and similarity suggests that the performance of each can be *controlled* for the other in experiments (Elmagarmid et al., 2007). In the last decade, in particular, both blocking and similarity have become increasingly complex. It is the norm, rather than the exception, to publish either on blocking or on similarity in an individual publication (Christen, 2012b). Despite some potential disadvantages, this methodology has resulted in the adoption of well-defined evaluation metrics for both blocking and similarity.

Evaluating Blocking

The primary goal of blocking is to scale the naïve one-step instance matcher that bilaterally pairs all entities with each other. Blocking accomplishes this goal by generating a smaller candidate set. If complexity reduction were the only goal, blocking could simply generate the empty set and obtain optimal performance. Such a blocking system would be useless because it would generate a candidate set with zero *duplicates coverage*.

Thus, duplicates coverage and candidate set reduction are the two goals that every blocker seeks to optimize (Hernández & Stolfo, 1995). To formalize these measures, let Ω be denoted as the set $D_1 \times D_2$; in other words, the exhaustive set of all bilateral pairs. Let Ω_M denote the subset of Ω that contains all (and only) matching entity pairs. Ω_M is designated as the ground-truth (equivalently, gold standard). As in previous sections, let C denote the candidate set generated by blocking. Using this notation, *Reduction Ratio (RR)* is defined below:

$$RR = 1 - \frac{|C|}{|\Omega|} \tag{2.1}$$

The higher the Reduction Ratio, the higher the complexity reduction achieved by blocking, relative to the exhaustive set (Christen, 2012b). Less commonly, RR can also be evaluated relative to the candidate set of a baseline blocking method (Papadakis et al., 2013). Note that, since RR has quadratic dependence, even small differences in RR can have an enormous impact in

terms of run-time. For example, if Ω contains 100 million pairs[27], and System 1 achieves an RR of 99.7%, while System 2 achieves 99.5%, their candidate sets would differ by 200,000 pairs.

In a similar vein, coverage, or *Pairs Completeness (PC)*, is defined below:

$$PC = \frac{|C \cap \Omega_M|}{|\Omega_M|} \tag{2.2}$$

Note that Pairs Completeness gives an upper bound on the *recall* metric that is used for evaluating overall duplicates coverage in the similarity step, as described in the subsequent section. For example, if PC is only 80%, meaning that 20% of the duplicate pairs did not get included in the candidate set, then recall on the full instance matching task will never exceed 80%.

There is typically a tradeoff between achieving high PC and RR. The tradeoff is achieved by tuning a relevant parameter[28]. There are two ways to represent this tradeoff. The first is a single-point estimate of the *F-Measure*, or harmonic mean, between a given PC and RR:

$$F - Measure = \frac{2 \times PC \times RR}{PC + RR} \tag{2.3}$$

A single-point estimate is only useful when it is not feasible to run the blocker for multiple parameter values. Otherwise, a more visual representation of the tradeoff can be achieved by plotting a curve of PC vs. RR for different values of the parameters.

Another tradeoff metric, *Pairs Quality* (PQ), is less commonly used than the F-Measure of PC and RR:

$$PQ = \frac{|C \cap \Omega_M|}{|C|} \tag{2.4}$$

Superficially, PQ seems to be a better measure of the tradeoff between PC and RR than the F-Measure estimate, which weighs RR and PC equally, despite the quadratic dependence of the former. In this vein, PQ has been described as a *precision* metric for blocking (Christen, 2012b). Intuitively, a

---

[27] By Linked Data standards, this is not an unreasonable number. It is easily achieved if both datasets contained 10,000 entities each.

[28] For example, the sliding window parameter w in Sorted Neighborhood (presented earlier in Section 2.2.1) can be increased to achieve higher PC, at the cost of lower RR (Hernández & Stolfo, 1995).

high PQ indicates that the generated blocks (and by virtue, the candidate set C) are dense in duplicate pairs.

In practice, PQ gives estimates that are difficult to interpret, and can be misleading. For example, suppose there were 1000 duplicates in the ground-truth, and C only contained 10 pairs, of which 8 represent duplicates. PQ, in this case, would be 80%. Assuming that the exhaustive set is large enough that RR is close to 100%, the F-Measure would still be less than 2% (since PC is less than 1%). The F-Measure result would be correctly interpreted as an indication that, for practical purposes, the blocking process has failed. The result indicated by PQ alone is clearly misleading, suggesting that, as a tradeoff measure, PQ should not[29] be substituted for the F-Measure of PC and RR. An alternative, proposed by at least one author but not used widely, is to compute and report the F-Measure of PQ and PC (Christen, 2012b).

Evaluating Similarity

Given a candidate set C, the similarity step uses a link specification function to partition C into sets C_D and C_{ND} of duplicates and non-duplicates respectively. The two metrics predominantly used for evaluating the similarity step, and by virtue, instance matching as a whole, are *precision* and *recall*:

$$Precision = \frac{|C_D \cap \Omega_M|}{|C_D|} \tag{2.5}$$

$$Recall = \frac{|C_D \cap \Omega_D|}{|\Omega_M|} \tag{2.6}$$

In other words, precision is the ratio of true positives to the sum of true positives and false positives, while recall is the ratio of true positives to the sum of true positives and false negatives. Similar to PC and RR defined earlier, there is a tradeoff between achieving high values for precision and recall. An F-Measure estimate can again be defined for a single-point estimate, but a better, more visual, interpretation is achieved by plotting a curve of precision vs. recall for multiple parameter values.

---

[29] This is not to say that PQ is not useful in its alternative interpretation as a precision measure. Even that interpretation is not without its problems: the true precision of an instance matcher is determined by the similarity step, not by blocking, which is only relevant as a computational preprocessing step.

Note that, since similarity is defined as a binary classification problem in the machine learning interpretation of instance matching, other measures such as *accuracy* can also be defined. One reason why they are not considered in the instance matching literature is because they also evaluate performance on the negative (i.e. non-duplicates) class, which is not of interest in instance matching (Elmagarmid et al., 2007). An alternative to a precision-recall curve is *Receiver Operating Characteristic (ROC)*, which plots true positives against false positives (Hanley & McNeil, 1982). Historically, and currently, precision-recall curves dominate ROC curves in the instance matching community (Menestrina, Whang & Garcia-Molina, 2010; Köpcke & Rahm, 2010; Köpcke, Thor & Rahm, 2010). In keeping with existing trends in the literature, precision-recall curves are favored over ROC curves for similarity evaluations in this dissertation.

2.3 Heterogeneity

In Section 2.2, *structural homogeneity* was explicitly assumed. The two implications of this assumption were that (1) entities in the two sources were compatibly typed[30], and that (2) the property schemas for the compatible types in question were identical in both sources. This section explores violations of these implications, and extends the basic model in Figure 2.2 to accommodate *type* and *property heterogeneity*.

2.3.1 Type Heterogeneity

Violation of the implication that all entities must have compatible types leads directly to the notion of *type heterogeneity*, namely, the presence of multiple types in the input RDF graphs.

In RDF graphs, type information is often published using a special property, denoted $:type$ in this section. Formally, let a *type declaration triple* be denoted as a triple that has the form $(subject, :type, object)$, where $object$ is a URI denoting a semantic *type* (equivalently, *class* or *concept*) (Ma,

30 It is incorrect to denote the entities as having the 'same' type. Two types can be compatible without being *equivalent*, as will be subsequently illustrated.

Tran & Bicer, 2013). Visually, any edge with the label $:type$, represents a type declaration triple in an RDF graph fragment.

Example 2.6: In Figure 2.9a, an example[31] of a type declaration triple is $(d1{:}\,Joseph\_Swan, :\,type, d1{:}\,Inventor)$. We denote the type $d1{:}\,Inventor$ as *containing* the instance $d1{:}\,Joseph\_Swan$; similarly, $d1{:}\,Joseph\_Swan$ is said to *have* type $d1{:}\,Inventor$. When the $subject$ in a type declaration triple is itself a type, it is said to be a *sub-type* of the $object$, which, by definition, must be a type; similarly, the $object$ is said to be a *super-type* of the $subject$. In Figure 2.9a, for example, $d1{:}\,Person$ is a super-type of $d1{:}\,Inventor$.

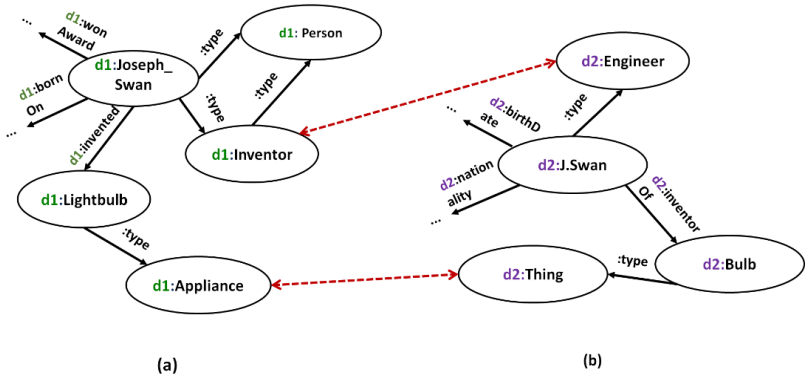


Figure 2.9: *Two RDF graph fragments illustrating type heterogeneity.* The dashed lines represent an alignment between semantically related (i.e. compatible) types.

The dashed lines between types in Figure 2.9 illustrate a *type alignment*, or a correspondence between *semantically related* types in two different datasets. Two entities, e_i and e_j, from the two datasets, are said to be *compatibly typed iff* there exists an alignment between two types, t_a and t_b, such that e_i and e_j have types t_a and t_b respectively[32].

In principle, type alignment is similar to blocking in that it seeks to avoid wasted comparisons 'for free'. Considering Figure 2.9, common sense

---

[31] It is a common misconception that Thomas Alva Edison 'invented' the incandescent lightbulb, in part, perhaps, due to a misleading advertising campaign known to have been launched by Edison's company at the time it was first patented and marketed in America.

[32] Note that an instance can have multiple types (e.g. *Joseph Swan* has two types in Figure 2.9a). A single alignment is sufficient for compatibility.

would indicate that it is futile comparing *appliances* (such as lightbulbs) to *people* in the hope of locating *:sameAs* links. Intuitively, a good type alignment algorithm addresses type heterogeneity by producing an alignment that maximizes the *efficiency* of overall instance matching (by restricting further processing to compatibly typed instances) but without sacrificing coverage of equivalent instances. This intuition will be formalized in Chapter 4.

In the Linked Data and Semantic Web communities, the problem of type alignment is related, but not identical, to the larger *ontology matching* problem (Euzenat & Shvaiko, 2007). Ontology matching only aligns two types if there is a well-defined relationship between them. In the literature, three such relationships that have been investigated most often are *subsumption, disjointness* and *equivalence* (Euzenat & Shvaiko, 2007). *Subsumption* implies that one type is a sub-type of the other, *disjointness* implies that the types do not overlap in terms of their instances, and - *equivalence* implies that both types are (semantically) identical. This last relationship has the same semantics as the *:sameAs* links in instance matching[33].

As Figure 2.9 illustrates, type alignments often do not have such well-defined semantics. For example, the only relationship between $d1\colon Inventor$ and $d2\colon Engineer$ is that they contain overlapping instances. At the same time, the types $d1\colon Person$ and $d2\colon Engineer$ have well-defined semantics (subsumption), but are not aligned. Given the data in Figure 2.9, aligning $d1\colon Person$ and $d2\colon Engineer$ would not be useful, since the equivalent mentions of *Joseph Swan* have already been covered by the alignment between $d1\colon Inventor$ and $d2\colon Engineer$. This ties directly into the intuition, earlier presented, that type alignment is more similar to blocking than to ontology matching.

The observations and intuitions noted above suggest that a good type alignment algorithm should be *data-driven* (like blocking), rather than *semantics-driven* (unlike ontology matching). This notion is revisited in Chapter 4, where a data-driven type alignment algorithm is presented and evaluated.

---

[33] For precisely this reason, many ontology matching researchers have also applied their innovations to instance matching (Euzenat & Shvaiko, 2007). Some notable examples are reviewed in Section 3.1.4.

2.3.2 Property Heterogeneity

Once the types are aligned, *property heterogeneity*[34] arises within *each* pair of aligned types. If the RDF graphs are represented as logical property tables, the problem bears a resemblance to *schema matching* in the Relational Database community (Rahm & Bernstein, 2001). Columns in one table need to be matched (possibly many-many) to columns in another table. An influential survey of schema matching was provided by Bellahsene, Bonifati and Rahm (2011). Similar to instance matching, machine learning methods have been successfully applied to schema matching in the previous decade (Bellahsene et al., 2011).

In the instance matching context, it is more appropriate to denote the matching process as *property alignment* rather than schema matching. One reason is that the property schema is only an abstraction for the purposes of data serialization. As earlier described, a formal RDF schema is defined by an extended vocabulary such as OWL or RDFS (McGuinness & Harmelen, 2004; Allemang & Hendler, 2011). A second reason is that, similar to type alignment, property alignment also relies on *semantic relatedness* to match properties. Indeed, many of the arguments presented about type alignment in the previous section also apply to property alignment. A successful property aligner is inherently data-driven, as detailed in Chapter 6, and should not assume standard semantics of subsumption or equivalence.

34 Property alignment is the second sub-problem within ontology matching. Relationships previously mentioned in the context of type alignment (equivalence, disjointness and subsumption) are similarly applicable to property alignment (Euzenat & Shvaiko, 2007).

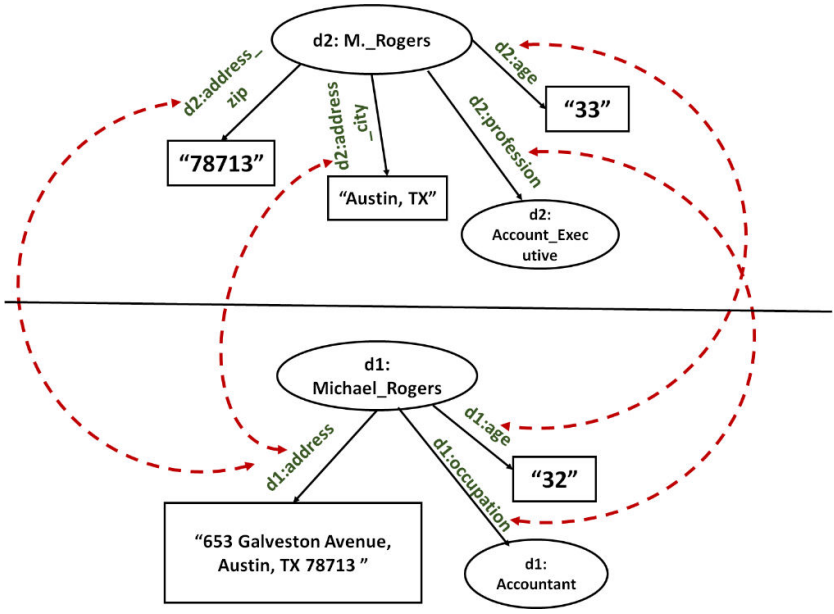


Figure 2.10: *An illustration of property alignment between the property schemas of two compatible types in two datasets.* The two compatible types, which could be *Executive* and *Businessman*, for example, are assumed to have been aligned by a type alignment procedure. Because the datasets are independent, the data values describing the entity *Michael Rogers* differ slightly.

Figure 2.10 illustrates a property alignment between two property schemas[35]. Visually, the alignment is represented through bi-directional dashed lines between edge labels. The alignment between the properties $d1{:}\,occupation$ and $d2{:}\,profession$ shows that an algorithm cannot rely solely on string matching. The alignment between the address properties shows that matches may occur in a many-one (and potentially, many-many) fashion. Collectively, these observations indicate that feasible property alignment should be robust, and is qualitatively more fine-grained than type alignment.

A computational motivation for property alignment can also be stated. As described in Section 2.2.2 (and illustrated in Figure 2.7), a machine learning-based similarity function would first convert an entity pair into a

[35] By the description in Section 2.1.3, the two datasets in Figure 2.8 have property schemas $\{d1{:}\,age, d1{:}\,occupation, d1{:}\,address\}$ and $\{d2{:}\,age, d2{:}\,profession, d2{:}\,address\_city, d2{:}\,address\_zip\}$ respectively.

numeric feature vector using a library of m feature functions. In Section 2.2.2, structural homogeneity was assumed, meaning that the property alignment is *trivial* (and one-one). Given n properties, the feature vector for any pair of entities would have exactly mn elements.

Suppose, instead, that the two datasets exhibited property heterogeneity, with their respective property schemas containing n_1 and n_2 properties each. To convert a bilateral pair of entities into a feature vector, a naïve feature generator (operating without the benefit of property alignment) would be forced to apply the m feature functions to *every* pair of properties, leading to a vector with mn_1n_2 elements. In most real-world datasets, the actual number of aligned property pairs would be small compared to n_1n_2. In Figure 2.10, for example, $n_1n_2 = 12$, but the number of property alignments (i.e. the number of dashed lines) is only 4. Even with a library of only 10 feature functions, the dimensionality of each feature vector is reduced from 120 to only 40. Computationally, this would benefit any machine learning classifier that is being trained on those feature vectors[36].

Given its qualitative and computational benefits, a good property alignment module is clearly an important component of a heterogeneous instance matching pipeline. Chapter 6 details the property alignment module developed for the purpose of this dissertation.

2.3.3 Extending the Two-Step Workflow

To conclude this section, the basic two-step workflow in Figure 2.2 can be extended to account for structural (i.e. type and property) heterogeneity by including type and property alignment modules in the workflow. These modules output sets of alignments that are then used in the blocking and similarity steps.

---

[36] There could also be potential qualitative benefits, since the *exhaustive* property alignment (of size n_1n_2) would likely generate many *irrelevant* features, impeding machine learning generalization.

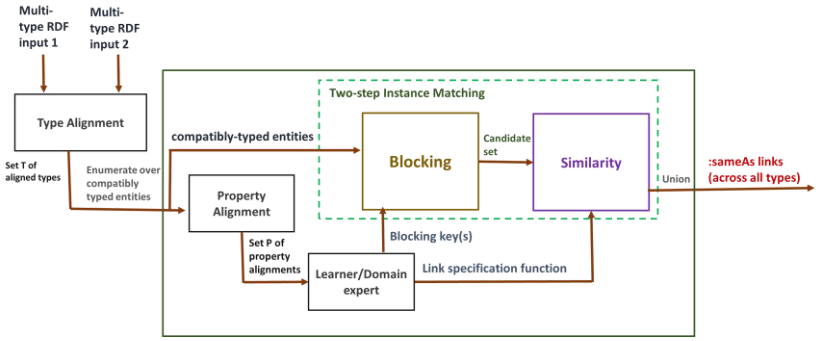


Figure 2.11: *A possible extension of the basic two-step instance matching workflow.* The inner dashed box is for illustrative purposes only. It is theoretically possible to reorder some of the components and obtain other versions of the extension (see text).

Figure 2.11 illustrates one possible extension of the workflow in Figure 2.2. Multi-type RDF graphs are input to a type alignment module, which outputs a set T of aligned types. For each such aligned pair, compatibly-typed entities are input[37] to both the property alignment module, as well as the blocking method. Using the property alignment P, a learner (e.g. a machine learning algorithm) or a domain-expert would output blocking keys[38] and a link specification function. At this point, the standard two-step workflow is executed (the dashed box in Figure 2.11), and the *:sameAs* links (between the compatibly-typed entities) are collected. In general, the outer box must be executed for each aligned type, highlighting the computational importance of having a good type alignment (discussed further in Chapter 4). The final output of the system is a union of all the sets of *:sameAs* links output by the individual executions.

[37] Graph-theoretically, these are *subgraphs* of the two multi-type graphs originally input to the system.

[38] Since the datasets are structurally heterogeneous, a blocking key can be defined for each input dataset (Section 2.2.1). In contrast, there is only one link specification function, since each entity pair is converted to a single numeric feature vector.

Note that this is not a definitive extension; other possible workflows can also be constructed. For example, if an instance-based blocking method (e.g. *Canopies*) is assumed, the learner in Figure 2.11 would only output a link specification function, and the blocking step (now completely schema-free) may be executed in parallel[39] with the property alignment module. The implementation of these modules is also unspecified, and would depend on the assumptions and use-cases of the overall system. In Chapter 3, various possibilities are discussed, based on a review of the literature; Chapter 1 illustrated and briefly described the schematic of the system developed in this dissertation.

2.4 Scalability

Preceding sections did not specify the actual *implementation* of the (basic or extended) model. Traditionally, instance matchers have been implemented and evaluated on *serial* machines (Elmagarmid et al., 2007; Christen, 2012a). Recent years have witnessed a surge in parallel and distributed instance matching research (reviewed in Chapter 3). In the next section, the motivation for a scalable instance matcher is first discussed, following which, the MapReduce paradigm, used for scaling the system developed in this dissertation, is reviewed.

2.4.1 Motivation

Given that the primary application of this dissertation is data integration in the Linked Open Data ecosystem, it is prudent to discuss the motivation behind scaling instance matching for such an application. This is because, at first glance, it is not obvious that scaling is even *required* for processing the largest datasets on Linked Open Data.

For the sake of discussion, let a dataset be putatively denoted as being *small-scale* if it contains 100,000 entities or fewer, *medium-scale* if it contains between 100,000 and 5 million entities, and *large-scale* otherwise. The numbers of types and properties are assumed to be non-trivial but still small

[39] A more extreme possibility is that both blocking and similarity are instance-based, in which case, property alignment, but not type alignment, can be safely eliminated from the workflow in Figure 2.11.

compared to the number of entities. In Freebase[40], for example, which is currently the world's largest known (and downloadable) *encyclopedic* knowledge base, the total number of types was found to be well within 5000, the number of properties (per type) to be in the hundreds (sometimes, tens), and the number of English entities to be in the millions. The English versions of the knowledge bases, DBpedia[41], Wikidata[42] and Yago[43], which are some of the most highly connected on Linked Open Data, are only medium-scale per the putative definition above (Suchanek, Abiteboul & Senellart, 2011). Other datasets (e.g. the New York Times[44]) contain far fewer entities; more generally, at the *tail-end* of the distribution, there are numerous small-scale datasets on Linked Open Data. Concerning domain-specific test cases, some of the largest datasets on Linked Open Data arise in the bioinformatics domain, but popular datasets (e.g. the Gene Ontology[45] dataset) still tend to fall within the medium-scale category.

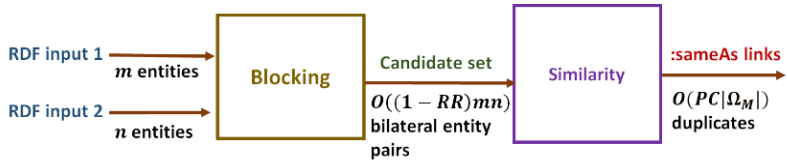


Figure 2.12: *An illustration of two-step instance matching from a complexity-theoretic perspective.* RR and PC stand for Reduction Ratio and Pairs Completeness respectively, and were described in Section 2.2.3. Ω_M is the ground-truth set of duplicate entity pairs (equivalently, the set of *true positives*).

One could make the argument that scalability should not be a major concern for such medium-scale datasets[46]. To refute such an argument, consider Figure 2.12, which illustrates the two-step instance matching

40 http://www.freebase.com/

41 http://dbpedia.org/services-resources/datasets/data-set-38/data-set-statistics

42 https://www.wikidata.org/wiki/Wikidata:Statistics/Wikipedia

43 http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/statistics/

44 http://data.nytimes.com/

45 http://www.ittc.ku.edu/chenlab/goal/stats.php. According to the latest statistics, the total number of annotated gene products (molecular functions, biological processes and cellular components) in the Gene Ontology still fall short of 3 million.

46 Historically, scalability was never the first priority in the vast majority of instance matching research. In 2013, Getoor and Machanavajjhala published an influential paper motivating a scalable version of the problem (Getoor & Machanavajjhala, 2013). Since then, the scope of scalable instance matching research has continued to expand.

workflow from a *complexity-theoretic* perspective. The inputs to the system are two RDF graphs containing m and n entities respectively. Problems of scale tend to emerge because of the intermediate output (the candidate set), which can easily be in the billions, as the example below illustrates.

Example 2.7: Consider two medium-scale inputs containing 1,000,000 entities each, and a state-of-the-art blocking method that is able to achieve a reduction ratio of 99.9%. The exhaustive set contains 1 trillion entity pairs, while the candidate set contains 1 billion entity pairs. Each of these 1 billion pairs must be converted to a feature vector and evaluated by a similarity function. Thus, even with extremely high reduction ratios, *medium-scale inputs* lead to *large-scale instance matching applications*.

In summary, the current state of Linked Open Data, which already contains many medium-scale datasets and continues to grow (Schmachtenberg et al., 2014), motivates research in scalable instance matching. In Chapter 8, scalability is revisited.

2.4.2 Implementation

Scalable systems may be implemented either on a customized architecture, or in a popular (typically open-source) parallel paradigm. Examples of the latter include the Message Passing Interface (MPI*)* and MapReduce (Gropp, Lusk, Doss & Skjellum, 1996; Dean & Ghemawat, 2008). Of these, the MPI paradigm is used widely in the supercomputing community for tasks that require high amounts of data sharing (e.g. large-scale matrix computations). In the instance matching, and other, communities, where computations can be distributed in a shared-nothing setting, the MapReduce paradigm has emerged as dominant (Kolb, Thor & Rahm, 2012b). A full treatment of MapReduce may be found in the work by Dean and Ghemawat (2008); for the sake of completeness, Appendix A contains a brief description.

The MapReduce model has found excellent support in the cloud, at the time of writing, with all major vendors offering elastic MapReduce services. Both proprietary (e.g. Google MapReduce) and open-source versions (e.g. Hadoop) of MapReduce are available (Dean & Ghemawat, 2008; White, 2012). More powerful (e.g. in-memory) variants of the basic MapReduce

model continue to be proposed, an influential example being *Spark* (Zaharia, Chowdhury, Franklin, Shenker & Stoica, 2010).

The scope of the thesis, in the context of scalability, is limited to the basic MapReduce model, with alternate implementations (e.g. in Spark) left for future work.

Chapter 3: Related Work

The goal of this chapter is to provide an overview of related work from the lens of the DASH requirements (domain-independence, automation, scalability and heterogeneity). This goal is pursued in the following way. First, Section 3.1 details influential domain-independent instance matchers that also non-trivially meet at least one of the other DASH requirements. In Section 3.2, these observations are synthesized, and some generalizations are derived. A takeaway from the discussion in Section 3.2 is that simultaneously meeting all four DASH requirements is conceptually problematic for many of the candidate systems.

3.1 Existing Domain-Independent Systems

At a broad level, most instance matchers can be divided into domain-independent systems and domain-specific systems (Ferraram, Nikolov & Scharffe, 2013). For example, the RKB co-reference resolution system uses resource equivalence lists that must be compiled in an ad-hoc fashion for each dataset (Jaffri, Glaser & Millard, 2008). Another example is GNAT, which is specifically designed for the music domain (Raimond, Sutton & Sandler, 2008). Another domain that has recently been of much interest is education (Dietze et al., 2013). It would not be misleading to portray *record linkage*, a version of instance matching in the tabular domain, as originally being a domain-specific application. In the earliest work that we are aware of, record linkage was specifically applied to the problem of obtaining reliable family statistics (Newcombe, Kennedy, Axford & James, 1959). The use of linkage techniques on census datasets was also a major motivation in the 1990s (Winkler, 1993; 1999).

System	Automation	Scalability	Heterogeneity
Winkler's Expectation Maximization (Winkler, 1993)	Yes		
Hierarchical Graphical Models (Ravikumar & Cohen, 2004)	Yes		
Latent Dirichlet Allocation (Bhattacharya & Getoor, 2006)	Yes		
Christen's Support Vector Machine (Christen, 2008b)	Yes		
RAVEN (Ngomo, Lehmann, Auer & Höffman, 2011)	Yes		Yes
SERIMI (Araujo, Hidders, Schwabe & De Vries, 2011)			Yes
PARIS (Suchanek, Abiteboul & Senellart, 2011)			Yes
FEBRL (Christen, 2008a)	Yes		
Dedoop (Kolb, Thor & Rahm, 2012)		Yes	
Locality Sensitive Hashing techniques (Kim & Lee, 2010)		Yes	
Transfer learning/schema-free features (Rong et al., 2012)	Yes		Yes
Genetic algorithms (Ngomo & Lyko, 2012)	Yes		
EUCLID (Ngomo & Lyko, 2013)	Yes		
COALA (Ngomo, Lyko & Christen, 2013)	Yes		
SILK algorithms (Volz, Bizer, Gaedke & Kobilarov, 2009; Isele, Jentzsch & Bizer, 2011)		Yes	
LIMES algorithms (Ngomo & Auer, 2011; Ngomo, 2011)		Yes	
SWOOSH algorithms (Benjelloun et al., 2009)		Yes	

Smart Joins (Vernica, Carey & Li, 2010; Metwally & Faloutsos, 2012; Das Sharma, He & Chaudhuri, 2014)		Yes	
Chaudhuri, Ganti & Motwani (2005)		Yes	

Table 3.1: *A list of domain-independent instance matchers.*

A key element of such systems is that their design process includes extensive prior knowledge about domain-specific processes. The tradeoff is a gain in performance on the domain in question. For some applications, such as census statistics and the medical domain, maximal performance is desirable, even at the cost of not being able to reuse the system beyond its intended application. On Linked Data, a flexible approach is prioritized, owing to the prevalence of numerous domains and stakeholders (Bizer, 2009). For this reason alone, such systems do not have a documented history of success in the general Linked Data ecosystem. They are not considered further in this discussion, as their motivations are tangential to those of this dissertation.

Much more interest has been generated in domain-independent systems, especially in recent years[47]. Some of these systems are listed in Table 3.1, along with an indication of which of the other DASH requirements they fulfill, and are reviewed subsequently.

3.1.1 Systems Addressing Automation

Since the early 2000s, machine learning has been actively applied to instance matching (Elmagarmid, Ipeirotis & Verykios, 2007). A machine learning-based instance matcher could adaptively learn good blocking and similarity functions from both the labeled training data (for supervised approaches), and the unlabeled data (for unsupervised, semi-supervised and clustering-based approaches). On the other hand, instance matchers that use a

[47] For the interested reader, accessible surveys and evaluations on much of what is covered in this chapter are provided by Köpcke et al. (2010), Elmagarmid et al. (2007), Christen (2012a), Winkler (1999) and Ferrara et al. (2013).

fixed set of heuristics on all data sources are non-adaptive, and by any pragmatic definition, the issue of automation trivially does not arise.

One of the earliest examples of an adaptive instance matcher, proposed by Winkler (1993), uses a variant of the Expectation Maximization (EM) algorithm (Dempster, Laird & Rubin, 1977). The *Fellegi-Sunter* model of record linkage is assumed (Fellegi & Sunter, 1969). In this model, candidate entity pairs are partitioned into three classes (matches, non-matches and *possible* matches) using two decision thresholds. The class of possible matches includes entity pairs that are too ambiguous for the similarity function to resolve into a match or non-match class. Such pairs require clerical review. A Bayesian argument shows that using two decision thresholds is optimal in the sense of minimizing possible matches for preset Type I and II error rates (Fellegi & Sunter, 1969).

Unfortunately, Winkler (2002) stipulated that the EM algorithm can only be successfully applied to instance matching if at least five empirical conditions are met. Elmagarmid et al. (2007) succinctly list these conditions, some of which are problematic for Linked Data. One such assumption is conditional independence of features. Another is that the match class is well-separated from the non-match class. In Chapter 7, EM is considered as a baseline, and the empirical performance is confirmed to be less than ideal on a Linked Data test suite where many of these conditions are arguably not met.

Ravikumar and Cohen (2004) use similar, but more robust, ideas by proposing *hierarchical graphical models* as a way of modeling the similarity of features through latent variables. The system is unsupervised, but assumes structural homogeneity and a serial architecture. A distance function[48] is also assumed to be provided. Empirically, the scope of the work was limited to Relational Database deduplication applications.

On a similar note, Bhattacharya and Getoor (2006) use *Latent Dirichlet Allocation* (LDA) for modeling latent commonalities between entities (Blei, Ng & Jordan, 2003). The main application of their work is in *collective classification*. A classic example arises in the co-authorship domain. Given a set of bibliographic works, two authors (on two independent works) are likely to be the same individual if they have similar co-authors. By modeling such relational information through latent variables, pairs of

[48] In the paper, *Soft-TFIDF* was used as the distance function (Ravikumar & Cohen, 2004).

individuals can be collectively disambiguated. While promising, the work has not been shown to be applicable to domains where relational issues don't arise. Similar to the work by Ravikumar and Cohen (2004), structural homogeneity and serial execution were both assumed in the original paper (Bhattacharya & Getoor, 2006).

Christen (2008b) adopts a different approach. First, a strong *weight-based heuristic* is used to sample training examples that are almost certainly matches or non-matches. Intuitively, the feature weights in such examples are nearly all 1.0 for matches (or 0.0 for non-matches). The method is predicated on locating such extreme-weighted samples to bootstrap the training process. Weights are assumed to only provide positive information, and features are assumed to be relatively independent. A classifier (SVM) is trained on the samples and used to label other feature vectors in the candidate set.

The method, along with other viable classifiers, a synthetic data generator and a user interface, is available in the FEBRL toolkit (Christen, 2008a). FEBRL was originally designed for biomedical record linkage, but can be applied to other domains. Heterogeneity is a major issue, since FEBRL is designed for structurally homogeneous applications. Empirically, only small benchmarks were used for actual evaluations.

Systems based on *active learning* have also been proposed, two good examples being RAVEN and COALA (Ngomo et al., 2011; 2013). Such systems do not require as many training examples as fully supervised systems such as MARLIN (Bilenko & Mooney, 2003), and deliver competitive performance. A major disadvantage is scalability, owing to the method being iterative and requiring continuous user participation. On a positive front, heterogeneity is less of an issue as these systems, unlike MARLIN, were designed for explicit Linked Data applications. In particular, RAVEN accommodates structural heterogeneity by modeling type and property alignments as an application of the *stable marriage* problem (Gusfield & Irving, 1989).

Genetic algorithms have also been extensively explored (Ngomo & Lyko, 2012), both in supervised and unsupervised versions. The unsupervised version relies on a measure known as a *pseudo F-Measure* (PFM). PFMs are heuristics that aim to approximate the actual F-Measure by analyzing the data, and are used as fitness functions in the genetic algorithms. A PFM can also be used to guide the unsupervised learning of a link specification function, as in

the deterministic EUCLID algorithm, which uses linear and Boolean classifiers (Ngomo & Lyko, 2013). Although promising, evaluations have shown that the correlation between various proposed PFMs and the actual F-Measure is tenuous (Ngomo & Lyko, 2013). With genetic approaches, the entire dataset has to be scanned over multiple iterations, and results are non-deterministic. In the original papers, EUCLID and the genetic algorithms also did not include solutions for type and property alignments, and were evaluated on small benchmarks (Ngomo & Lyko, 2012; 2013). Taken together, these observations indicate that these algorithms may not be suitable for large-scale Linked Data applications.

A promising solution that requires training data, but that can then be applied to other datasets with minimal supervision through *transfer learning* was proposed by Rong et al. (2012). This solution is also one of the few to favor both automation and heterogeneity, the latter by virtue of employing schema-free features. An example of a schema-free technique that was earlier introduced in Chapter 2 was *Canopies* (McCallum, Nigam & Ungar, 2000). Such techniques address heterogeneity in a brute-force fashion, by ignoring all structural information. In the case of Rong et al. (2012), features are extracted by jointly considering the information set of all properties (of a candidate instance pair). For example, a numeric parser is used to extract numeric information (e.g. dates) present in the properties. A problem with using such features is that noise can be introduced by extracting irrelevant information. Also, Rong et al. (2012) do not directly address *type* heterogeneity. Finally, while transfer learning has some advantages, it also degrades occasional performance. Determining when to use transfer learning is an ongoing area of research (Pan & Yang, 2010).

3.1.2 Systems Addressing Heterogeneity

Linked Data instance matchers are developed explicitly with heterogeneity in mind. The prevalence of numerous type and property definitions in Linked Data is well-documented (Schmachtenberg, Bizer & Paulheim, 2014). However, aligning the properties and types is assumed to be the responsibility of an *ontology matcher* that has been invoked *a priori* (Euzenat & Shvaiko, 2007). For example, the EUCLID algorithm explicitly invokes a property aligner, but the actual alignment algorithm is neither provided, nor is its performance or the contingent effects of wrong alignments

on subsequent instance matching, evaluated (Ngomo & Lyko, 2007). This is similar to an assumption often made in the record linkage community, where a *schema matcher* is assumed to have been invoked prior to the linkage itself (Hernández & Stolfo, 1998; Elmagarmid et al., 2007).

An important point is that, traditionally, property heterogeneity has received far greater attention in the instance matching community than type heterogeneity. The term *structural heterogeneity*, as used originally by Elmagarmid et al. (2007), did not include type heterogeneity. Schema matching systems have also placed more emphasis on aligning properties and columns (in the case of tabular databases) than on aligning types (Bilke &Naumann, 2005; Bellahsene, Bonifati & Rahm, 2011). In Table 3.1, a liberal definition of heterogeneity was adopted, and any system that addressed property heterogeneity was designated as fulfilling the heterogeneity requirement[49].

A notable exception is the RAVEN system (Ngomo et al., 2011), earlier described as an active learning-based instance matcher, which aligns types and properties by using a *stable marriage* sub-module (Gusfield & Irving, 1989). From an empirical perspective, the sub-module was not evaluated against schema matching systems such as *Dumas*, which have shown reasonably good performance on noisy test cases (Bilke & Naumann, 2005). The benchmarks used for evaluating RAVEN did not exhibit high heterogeneity. Thus, while RAVEN addresses heterogeneity in principle, it prioritizes automation through its active learning-based methodology. The scalability of the method on RDF data that does not fit in main memory is also not evident.

PARIS, proposed by Suchanek et al. (2011), performs heterogeneous instance matching within the framework of ontology matching[50]. PARIS is not adaptive, and models the instance matching problem probabilistically. The framework is iterative, requiring fixpoint computations for the model equations. Although evaluated on reasonably large datasets, the setup was

---

[49] In addition to erring on the side of caution, this decision is also justified on the grounds that type alignment, being an 'easy' problem, can almost always be included as an independent preprocessing module without significantly modifying the remainder of the system (Figure 2.11).

[50] The authors describe the framework as *holistic*, since Paris outputs alignments between instances, properties and types, much like RAVEN and the system developed in this dissertation (Suchanek et al., 2011).

serial and the authors made no claims about scaling the system to larger datasets, or a distributed implementation.

Finally, the SERIMI system (Araujo et al., 2011) addresses heterogeneity in a manner similar to Rong et al. (2012), namely, by considering schema-free techniques rather than actual alignment. SERIMI is not an adaptive system, and relies on a suite of similarity functions and collective heuristics to locate similar entities. It was also not implemented in a distributed, shared-nothing setting. It fulfills neither the automation nor scalability requirement.

3.1.3 Systems Addressing Scalability

There has been increased interest in scalable systems over the last five years, especially in the Semantic Web. The algorithms implemented in the SILK architecture, for example, can be processed efficiently over SPARQL endpoints over RDF Web data sources (Volz et al., 2009), as can algorithms implemented in the LIMES framework (Ngomo, 2011). In evaluations, LIMES was found to be much faster than SILK (Ngomo, 2011). The architectures are customized and are not implemented in a standard shared-nothing paradigm. Another disadvantage of both SILK and LIMES is that they require link specification functions to be explicitly specified. The systems are non-adaptive and do not fulfill the automation requirement. LIMES, in addition, requires the specification function to obey metric properties in order to execute efficiently.

More theoretical work includes the influential Swoosh algorithms, D-Swoosh and P-Swoosh, which are parallel and distributed versions of the original Swoosh algorithms (Benjelloun et al., 2009). These algorithms have some impressive theoretical properties, but similar to LIMES, require the specification function to obey some strong constraints. In particular, it is not evident that these constraints are obeyed by general-purpose machine learning classifiers, limiting the applicability of Swoosh. Swoosh does not contain direct provisions for addressing heterogeneity. The parallel architecture is customized, and cannot be elastically deployed in the cloud.

Locality Sensitive Hashing (LSH) has also emerged as a popular technique for implementing distance functions in a scalable fashion (Kim & Lee, 2010). An accessible introduction to LSH for the nearest-neighbors

problem is provided by Datar, Immorlica, Indyk and Mirrokni (2004). In the Semantic Web, LSH techniques for the *Jaccard* and a version of the *Cosine* distance function were first used by Duan et al. (2012) for ontology matching. LSH can accommodate heterogeneity by ignoring structure and treating instances as bag of tokens. An adaptive version of LSH (by using hashes as features and then employing Expectation Maximization) is considered as an unsupervised baseline in Chapter 7.

The *Dedoop* framework allows users a way to specify an instance matching workflow and efficiently execute it in MapReduce by using load balancing techniques (Kolb et al., 2012). There is a significant manual component involved. To the best of our knowledge, Dedoop is the only system that provides an end-to-end MapReduce-based instance matcher.

In earlier, but still influential, work by Chaudhuri et al. (2005), a scalable distance-based instance matching (referred to by the authors as *fuzzy duplicates identification*) system was implemented for Relational Databases using Microsoft SQL Server as backend infrastructure. Near-linear scalability was demonstrated for Relational datasets containing up to three million entities.

There are also examples where an architecture is purportedly scalable but has been evaluated only on small datasets or in simulated execution environments. For example, the D-Swoosh system was evaluated on datasets that had between 5,000-50,000 entities and the evaluations were conducted on emulated distributed environments (Benjelloun et al., 2007). FEBRL, earlier mentioned in the context of automation, was implemented using the Message Passing Interface (MPI) but only evaluated on small datasets on a single compute node (Christen, 2008a). Another system, proposed by Kirsten et al. (2010), was slightly more ambitious, and conducted evaluations on a match task of 114,000 entities. By Linked Data standards, this is quite small; the DBpedia dataset alone contains slightly over 3 million entities (Auer et al., 2007). The system also relies on the manual specification of a workflow, not unlike Dedoop, and the architecture is not shared-nothing (Kirsten et al., 2010). In contrast, the system developed in this dissertation is evaluated on datasets containing between 50,000 to 1.5 million entities (Chapter 8).

In other related work, *smart joins* implemented scalably in MapReduce have received enormous attention in the database literature (Vernica et al., 2010; Metwally & Faloutsos, 2012; Das Sharma et al., 2014).

A smart join algorithm assumes a set similarity function, typically with a threshold. The goal is to return all pairs of records that satisfy the thresholded similarity condition. Technically, this is different from the instance matching problem where the function is unknown (even to a human being) and has to be approximated. Thus, the smart join algorithms represent a different extreme where automation is explicitly disregarded and heterogeneity is accommodated only through the specification, but where strong scalability guarantees are available.

3.1.4 Other Systems

There are other systems that have delivered impressive performance and are used in a variety of contexts, but that are not discussed here (and are not listed in Table 3.1). The main reason is that these systems are either *domain-specific* or are *schema-based*[51]. Earlier, a brief discussion on, and some examples of, domain-specific instance matchers were provided. In this section, the discussion is restricted to schema-based systems.

Schema-based systems tend also to be *rule-based* (Leonardi et al., 2010). Many approaches were originally proposed as ontology matchers. Examples include RiMOM, LogMap, Asmov and ObjectCoref (Li, Tang, Li & Luo, 2009; Jiménez-Ruiz & Grau, 2011; Jean-Mary, Shironoshita & Kabuka, 2010; Hu, Qu & Sun, 2011). New systems continue to be proposed each year as part of the annual *Ontology Alignment Evaluation Initiative*[52] (Ferrara, Nikolov, Noessner & Scharffe, 2013). Another recent system, based on *rule-mining*, relied on the existence of inverse functional properties to learn good rules using a variant of the EM algorithm (Niu, Rong, Wang & Yu, 2012). The authors of that work proposed the system as a 'third choice' between a domain-specific and domain-independent system, since the rules were dataset-specific and required multiple EM iterations. Two other examples of Semantic Web data matchers that rely either on the specification of an ontology or a user workflow are KnoFuss and RDF-AI (Nikolov, Uren, Motta & De Roeck, 2008; Scharffe, Liu & Zhou, 2009).

---

51 A schema-based instance matcher is primarily characterized by its reliance on a declared ontology or schema.

52 http://oaei.ontologymatching.org/

While the performance of all these systems is impressive in the presence of ontologies and metadata, the larger part of the Linked Data ecosystem is known to contain only shallow meta-data (Schmachtenberg et al., 2014). Thus, such systems are better suited to more constrained Semantic Web applications, rather than Linked Data applications.

3.2 Discussion

The previous section listed existing systems, and their strengths and limitations. The goal of this section is to generalize the findings discussed in the previous section, and to note potential additions to a subset of described systems that could help fulfill the DASH requirements. Could the automated systems in Table 3.1, for instance, be re-implemented in a scalable way that allows them to address type and property heterogeneity? What are the barriers to such adaptations?

3.2.1 Automation vs. Scalability

A cursory scan of Table 3.1 shows that there is a dichotomy between domain-independent systems that fulfill the automation requirement and those that fulfill the scalability requirement. This is unlikely to be a coincidence. In particular, scalable systems tend to make strong assumptions. Locality Sensitive Hashing techniques, for example, assume that appropriate *hashing families* exist for the distance functions being approximated (Datar et al., 2004). In the instance matching (and also ontology matching) literature, the only functions for which LSH has been appropriately utilized are *Jaccard* and a version of the *Cosine* distance function (Duan et al., 2012). An extension to LSH techniques to accommodate the properties of machine learning classifiers is by no means straightforward. Another example of an architecture amenable to parallel and distributed algorithms, Swoosh, also imposes strong assumptions on the similarity function (Benjelloun et al., 2009).

It is also interesting to note that instance matchers implemented in a shared-nothing paradigm, such as MapReduce, tend to leave the burden of specification on the user. Dedoop, for example, requires the user to completely specify the workflow (Kolb et al., 2012). The same is true for LIMES and SILK (Ngomo, 2011; Volz et al., 2009), which are not implemented in

MapReduce, but require the user to specify the appropriate functions and parameters. Smart joins, for reasons discussed earlier, are considered conceptually disjoint from instance matching.

It is also possible to survey this issue from the opposite end of the spectrum. Automated systems, which mainly tend to be EM-based algorithms that iteratively refine a likelihood function by learning good parameters for latent variables, require multiple scans over the dataset, copious amounts of data sharing and an unspecified number of iterations before convergence (Ravikumar & Cohen, 2004; Bhattacharya & Getoor, 2006). In general, they are non-deterministic and may require multiple re-starts to avoid the pitfalls of local optima. As Winkler (1993) observed, various empirical conditions have to hold for such algorithms to be viable. Recent progress on this last issue has been promising but is not a settled matter at the time of writing (Rong et al., 2012).

A promising approach that is potentially amenable to a *fixed* number of approximately *linear-time* MapReduce jobs is the SVM-based proposal by Christen (2008b). In its present form, the proposal accommodates neither scalability nor heterogeneity. The latter problem can be dealt with, as described in the following section. It is less obvious how the system can automatically *and* scalably locate good seed examples to bootstrap the training process. Christen makes the assumption that seeds can be unambiguously located by seeking feature vectors with weights that are nearly all 0 or 1. With noisy data, this is almost never guaranteed. In empirical findings described in later chapters, feature vectors are often found to be sparse, even for duplicates[53]. If seeds can be located from such data using a fixed number of MapReduce jobs, automation and scalability requirements can be reconciled. A road map for this is provided in both Chapters 5 and 8. Once located, seeds can be used, in principle, for learning multiple functions.

3.2.2 Issues of Structural Heterogeneity

A traditional assumption in the instance matching community is that datasets have been *homogenized* prior to executing an instance matching workflow (Köpcke, Thor & Rahm, 2010). For this assumption to be validated,

[53] The principal reason for this is another DASH requirement, namely domain-independence. Domain-independence requires adopting a 'broad' feature-set, which results in sparsity on any one test set.

ontology matching must be performed *a priori* (Elmagarmid et al., 2007; Christen, 2012a; Ferrara et al., 2013b).

This assumption would not be problematic if ontology matching were a solved problem. In fact, research on them has been ongoing for many decades (Rahm & Bernstein, 2001; Euzenat & Shvaiko, 2007). In some cases, schema matching systems assume that instance matching has been solved *a priori* (Bilke & Naumann, 2005)[54]. The argument is that, despite the progress in both instance matching and schema matching, it is misleading to assume that either problem has been solved perfectly.

The question is largely empirical. Is it sufficient to use classic, relatively simple, approaches to address type and property heterogeneity in the broader context of instance matching? In Chapters 4 and 6, it is shown that while type heterogeneity is amenable to classic approaches, property heterogeneity is not. Insofar as the related work is concerned, only the RAVEN system *properly*[55] deals with heterogeneity, although empirical evaluations on this issue are limited (Ngomo et al., 2011). Other systems, like the one by Rong et al. (2012), address heterogeneity by using schema-free features that ignore structural properties altogether. An empirical argument against such approaches, for well-structured RDF graphs, is provided in Chapter 7.

Structural heterogeneity also impacts automation. In Chapter 1 (Section 1.3), one possibility mentioned for addressing automation was the use of distant supervision. Wikipedia, for example, can be used to acquire 'supervision' in named entity disambiguation applications (Cucerzan, 2007). It is not unreasonable to propose a similar utility for DBpedia (or a similar graph), which is structured using Wikipedia infoboxes.

Structural heterogeneity impedes this issue because, when input two data sources, a system would have to first find compatible types and properties between DBpedia and the two inputs to guide supervision[56]. Property heterogeneity presents a particularly formidable challenge owing to the schema-free nature of even well-curated Linked Data (such as DBpedia), as at least one recent study has shown (Arenas, Díaz, Fokoue, Kementsietsidis & Srinivas, 2014). Scale is also a barrier. Small data sources, which would

---

54 The *Dumas* schema matcher, used as a baseline in Chapter 6, uses duplicates to match columns.

55 That is, addresses heterogeneity through alignments, as opposed to ignoring structure.

56 We take it for granted that such types and properties exist.

normally require only serial processing, would now have to accommodate large, noisy and dynamic knowledge bases in the pipeline. In such scenarios, a *self-contained* solution is clearly more desirable.

In an instance matcher, the problem of structural heterogeneity is overcome by prepending alignment modules to the basic two-step workflow detailed in Chapter 2. In later chapters, viable alignment solutions are presented and evaluated. Recent progress on the structural heterogeneity issue has been promising, especially in the context of blocking (Papadakis, Ioannou, Palpanas, Niederée & Nejdl, 2013).

3.2.3 Issues of Unsupervised Blocking

Many of the systems earlier described had a strong focus on the *similarity* step of instance matching. An unfortunate consequence of the complexity of recent instance matching research is that researchers often ignore other aspects of instance matching, such as blocking, in their exclusive focus on similarity or scalability. For example, both Ravikumar and Cohen (2004), and Bhattacharya and Getoor (2006) use simple ad-hoc blocking keys in their experiments[57]. Scalable systems make more extreme assumptions. For example, Dedoop require both blocking and similarity steps to be precisely specified by a user as part of a workflow (Kolb et al., 2012).

Evaluations in Chapter 7 show that traditional techniques such as *Canopies* may not work well on heterogeneous RDF datasets (McCallum et al., 2000). In the Semantic Web, an unsupervised blocking scheme learner, by Song and Heflin (2011), was evaluated on small datasets and is not as expressive as DNF blocking schemes. In real-world instance matching, unsupervised blocking should not be assumed away, since it is still unsolved.

[57] For example, all records sharing a 4-gram character sequence were placed in the same block (Ravikumar & Cohen, 2004).

Chapter 4: Type Alignment

Type alignment is the first line of attack against structural heterogeneity. Relatively simple heuristics, executed in an unsupervised fashion, turn out to be adequate for solving this problem on real-world datasets. This chapter covers the type alignment module developed for the dissertation (Kejriwal & Miranker, 2014). Although we do not consider this module as constituting a *core* contribution in support of this dissertation, we detail it on account of its importance as the very first step in the pipeline in Figure 1.5.

4.1 Motivating Example and Preliminaries: A Review

In Chapter 2, type alignment was introduced as an extension to the basic two-step instance matching workflow. A type was defined in the context of a type declaration triple of the form $(entity, :type, type)$. Namely, the special property $:type$ is used to indicate that $entity$ has type $type$. $entity$ itself could be a type, in which case it has *super-type* $type$. A type hierarchy can be constructed in this way.

The sub-type and super-type relationships are examples of containment and subsumption respectively. *Equivalence* imposes a stronger condition. Using standard set semantics and notation, a type A is equivalent to another type B if A is both a super-type and sub-type of B. In a similar manner, disjointness can be defined.

In many ontology matching applications, the primary goal is to discover *equivalence* and *containment* relationships (Euzenat & Shvaiko, 2007). In instance matching, this goal is insufficient, and sometimes, unnecessary (Nikolov, Uren, Motta & De Roeck, 2009).

The running example in Figure 4.1 illustrates why type alignment is different from ontology matching, and motivates the development of a type alignment algorithm that is appropriate for an instance matching pipeline. In Figure 4.1, there are examples of type pairs not being aligned despite having well-defined semantics (e.g. *freebase:non-profit* is a sub-type of

dbpedia:Company); at the same time, one of the aligned pairs, *(dbpedia:Inventor, freebase:Entrepreneur)*, does not have well-defined semantics. Although rare in practical data integration applications, there is also no restriction on a type in one dataset being aligned with *multiple* types in the other dataset (Section 4.2).

Given sets of types[58] T and S from two input RDF graphs, a *type alignment* Θ is a subset of the Cartesian product $T \times S$, such that each pair in the type alignment comprises semantically related types. Type alignment has implications for both scalability and coverage, making it similar to blocking.

An intuitive explanation for the implication that type alignment affects both scalability and coverage can be stated. Let a bilateral entity pair (e_1, e_2) be denoted as being *covered* by a type alignment Θ *iff* there exists a pair $(t, s) \in \Theta$ such that e_1 and e_2 have types t and s respectively. Because e_1 and e_2 could be covered by multiple type pairs, the inclusion of all semantically related type pairs in Θ may not be beneficial. An example of a semantically related, but non-beneficial[59], type pair in Figure 4.1 is *(dbpedia:Person, freebase:Entrepreneur)*. A natural tradeoff between scalability and coverage is observed in terms of the type pairs included in Θ. For this reason, at least one paper refers to an instance matcher using both type alignment and standard blocking as employing a *blocking-within-blocking* strategy (Ma & Bicer, 2013).

58 Technically, these sets are extracted by scanning the type declaration triples.

59 This claim is accompanied by the caveat that, similar to blocking, the true benefit of type alignment must necessarily be proven through its *empirical* impact on the coverage and scalability of the overall instance matching problem (Section 4.4).

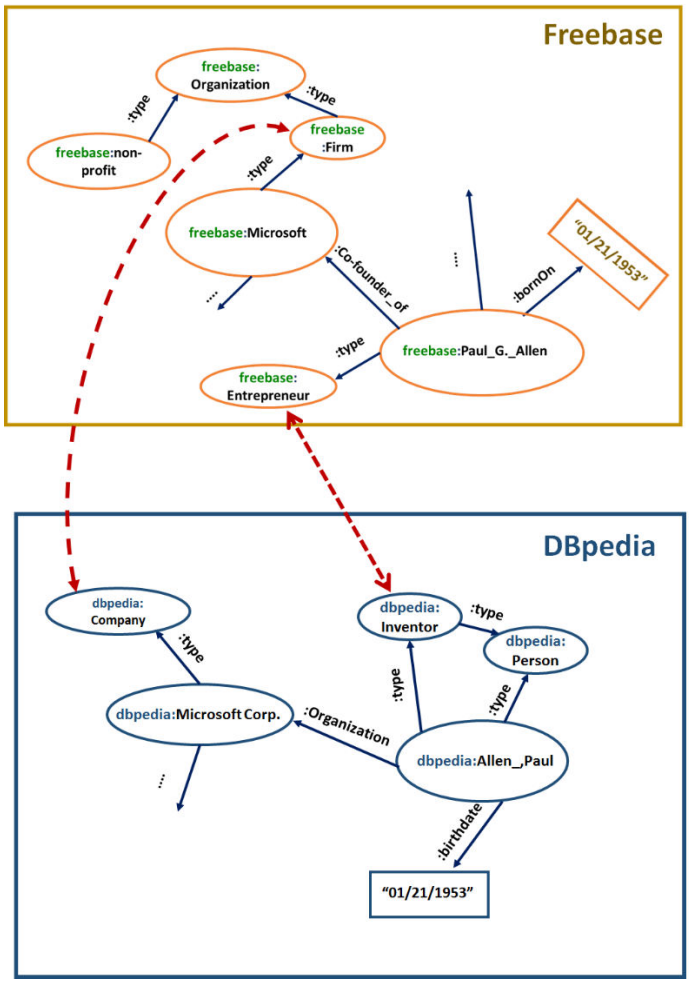


Figure 4.1: *The running example, illustrating the motivation behind type alignment.* An ideal type alignment system would take as input the two input graphs and output the dashed red lines shown in the figure. Note that, despite the visual similarity to *:sameAs* properties, these lines do not represent actual property declarations; hence, are unlabeled.

4.2 Applications of Type Alignment

The primary dissertation motivation is to populate a Linked Data Entity Name System for data integration. It is natural to question the extent of type heterogeneity in data sources common in domain-independent data

integration applications. Typically, heterogeneity in the data integration literature is assumed to mean property (or schema) heterogeneity, addressed *a priori* through a good ontology matcher (Elmagarmid, Ipeirotis & Verykios, 2007). In Chapter 3, when discussing which systems fulfilled heterogeneity requirements, explicit property alignment was considered to be the *de facto* criterion. The main reason for this, discussed at length in Chapter 2 and revisited in Chapter 6, is that property alignment is a finer-grained problem, known to affect both quality and scalability (Nikolov et al., 2009).

Two type alignment applications seem predominant in real-world Linked Data. The first application arises when each RDF graph input is actually a collection of graphs, none of which are interlinked. Each individual subgraph in the collection forms an undirected connected component, with a common type declaration linking all entities within the subgraph. The types are arranged in a flat hierarchy; hence, the types are not interlinked.

Strong evidence of this first application is found in *government data*, which has become a major contributor to Linked Open Data since the advent of the Open Government movement (Shadbolt et al., 2012). For example, in the United States, the Joint Committee on Taxation and the US Treasury both release data on budgetary and fiscal allocations annually. The schemas employed by both sources are different but stay homogeneous within each organization, undergoing minor changes every few years. Given a collection of annually released datasets over a period of several years, a type alignment system would match the files according to year. Some years may be missing in one of the directories; hence, the mapping may not cover all files. A good algorithm would detect the importance of numbers (especially dates) in performing the mapping, since the files are otherwise too similar. This application is used as a test case in Section 4.4.

A good analogy for the first application is to consider an input graph as a *directory*, and single-type subgraphs as *files* in the directory[60]. Given two such directories, the problem of type alignment is limited to finding a mapping between individual files. Some files may not have any corresponding matches, but if a match exists, it is typically one-one. The intuitive reason is that if two files, A_1 and A_2, in one directory match a file B in another directory, A_1 and A_2 are strongly related, a fact that is missing from the input. Although this is

[60] This is more than a convenient analogy. In the real-world cases that were collected for evaluations, the datasets were physically structured in this manner.

not a rigorous reason for anticipating a one-one mapping between two collections, real-world data tends to conform to it. To conclude the argument, if A_1 and A_2 are indeed strongly related, the first collection needs to undergo cleansing or deduplication before it can be correctly linked to another collection.

The first application is relatively trivial if the files belong to *different* domains. In such cases, there is much less scope for discovering false-positive mappings. If a DNF blocking scheme is used in the blocking step, the mapping is implicitly discovered by the DNF blocking scheme learner[61], and type alignment is unnecessary. One of the evaluations in Section 4.4 show that this is a distinct possibility; it is also relied upon in Chapters 5-7.

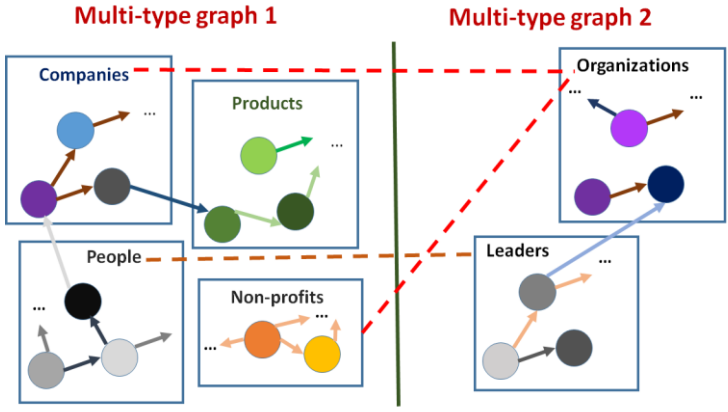


Figure 4.2: *Abstract depiction of the second type alignment application.*

The *second* type alignment application is unconstrained and arises when entities across types are interlinked (Figure 4.2). In this scenario, one-one mappings can no longer be assumed. A further complication arises when, in the case of many-many alignments, each alignment is not equally important. This makes the application similar to the problem of determining *query-document relevance* in information retrieval (Baeza-Yates & Ribeiro-Neto,

61 Functionally, a conjunction (or some disjunction of conjunctions) is learned for *each* domain, and absorbed into a *larger* DNF expression (returned as the final blocking scheme). Because the domains are different from each other, the conjunctions don't interfere, making the process indistinguishable from the one where a DNF blocking scheme is independently learned for each aligned type pair.

1999). For each type in one of the graphs, a ranked list of types in the second graph must be obtained.

While the first application is of direct concern to data integration, the second is only of tangential concern. One reason is that the second application arises mainly in the context of *encyclopedic* datasets such as DBpedia and Freebase. Although encyclopedic datasets are instrumental to the success of Linked Data (Bizer, 2009), often serving as a hub for connecting myriad datasets from different domains, their scope is too broad for them to serve as primary sources in a data integration application[62]. The focus in this chapter is on the first application, but the second application is relevant for type alignment *scalability* evaluations in Chapter 8.

It is reasonable to assume that type declaration triples are typically available for a given data source. Such an assumption must necessarily be based on a systematic analysis of published Linked Data. Tran and Bicer (2013) found that fewer than 10% of entities in the datasets under consideration in their work[63] lacked type information. While this is not a small number in terms of the absolute numbers of entities and triples involved, several approaches exist for deriving the missing type information. Tran and Bicer (2013) proposed a *typification* algorithm, which discovers latent type information through clustering. A simpler approach is to declare non-typed entities to have special type *other*. This approach is not unlike smoothing techniques applied in language models, which provide for rare words absent from the data used to derive model parameters (Zhai & Lafferty, 2001).

4.3 Approach

Given the nature of the type alignment problem and the preference for a data-driven algorithm, amenable to scaling in MapReduce, a *heuristics-based approach* is hypothesized to be an appropriate candidate. Some precedence for such approaches in the ontology matching domain already exists (Euzenat & Shvaiko, 2007), but are less favored than approaches based

62 Other anecdotal problems for not using encyclopedic datasets as primary data integration sources are quality, and the lack of control, since these datasets are themselves derived (e.g. from Wikipedia).

63 The considered datasets were mainly encyclopedic, suggesting that the missing type problem is more likely to occur in the second type alignment application. In the test cases gathered for the module in this dissertation, the problem was not encountered.

on structural features and logic. The drawbacks of schema-based approaches for the current problem were explained in Chapters 2 and 3.

Input: Two multi-type RDF graphs G_1 and G_2 with type sets T_1 and T_2 respectively

Output: Type Alignment Θ

Steps:

1. Initialize empty *type matrix* M of dimension $|T_1| \times |T_2|$
2. Use *similarity function* S to populate M
3. Apply *pair selection strategy* P on M to obtain set $\Theta \subseteq T_1 \times T_2$
4. Output Θ

Algorithm 4.1: *An abstract algorithm for type alignment.*

The pseudocode of an abstract algorithm is provided in Algorithm 4.1. The abstraction emerges from the similarity function and pair selection strategy, which are yet unspecified. Otherwise, algorithm execution is straightforward. First, the similarity function $S: T_1 \times T_2 \rightarrow [0,1]$ is used to populate a type matrix M, such that each element in the matrix represents a similarity score between two types. The pair selection strategy $P: M \rightarrow T_1 \times T_2$ derives the type alignment from the type matrix, and outputs it. In the present treatment, more complex scenarios (e.g. probabilistic type alignments) are not considered, as evaluations show such fine-grained distinctions to be unnecessary.

4.3.1 Possible Strategy Implementations

This section discusses possible implementations for the similarity function and pair selection strategy. The discussion is brief, and by no means exhaustive. The preference is for simple, robust approaches that can be employed by way of third-party black-box implementations. Though hard to objectively quantify, the use of transparent, openly available packages prove to be important assets in Linked Open Data domains.

Similarity function

A similarity function (within the context of Algorithm 4.1) is defined as a function that takes a pair of types (one from each graph) and returns a real value in the range of [0,1]. A similarity function is heavily influenced by the choice of argument representation. A good representation of a type is a *type document*, defined as a *bag* (or *multi-set*) of string tokens representing the *information set* of the type. In the literature, one suggested way of constructing a type document is to perform a multi-set union on the labels of all entities that have that type (Duan et al., 2012). This strategy is likely to fail if entity labels are opaque URIs (i.e. having syntactic relevance only) or are otherwise not indicative. The real-world encyclopedic graph, Freebase, falls within this category, but other cases also exist. In our work, the definition of a type document was extended to include both entity labels and also all $object$ strings that occur in a triple of the form $(subject, property, object)$ such that $subject$ is an entity having the type in question (Kejriwal & Miranker, 2014). Note that object may be a data value or a URI. The distinction is not made in the construction of the type document. An advantage of constructing the document in this way is that it is robust to large numbers of opaque strings, and the construction is scalable (Chapter 8).

Once constructed, any set similarity function can be applied to a pair of type documents. A distinction is made between *local* similarity functions that depend only on the two arguments, and *global* functions that require an additional information set, typically derived by collectively processing all documents. Local functions have an important scalability advantage over global functions, as they are far easier to handle in shared-nothing distributed settings. In serial settings, there is no particular advantage.

An example of a local set similarity function is the *Jaccard similarity function* that, for two bags (equivalently, type documents) S_1 and S_2, is defined as the ratio of the cardinality of the multi-set intersection of the bags and the cardinality of their multi-set union (Christen, 2012a):

$$Jaccard(S_1, S_2) = \frac{S_1 \cap S_2}{S_1 \cup S_2} \tag{4.1}$$

An example of a global similarity function is *Cosine similarity* between (appropriately normalized) *TFIDF*[64] weight vectors:

$$Cosine(S_1, S_2) = \sum_{q \in S_1 \cap S_2} w(S_1, q) w(S_2, q) \qquad (4.2)$$

A type document is 'cast' as a TFIDF weight vector by assigning a weight to each unique term in the domain. The weight depends on both the term frequency (TF) and the inverse document frequency (IDF). TF is given by the frequency of the term in the given document, while IDF is the inverse of the number of documents that the term appears in at least once. Specific formulae for computing TF-IDF weights will be provided in the next chapter, where they also play a significant role.

Computing IDF statistics requires the entire document collection to be scanned, making the similarity function global. A local version is also possible, if the IDF term is ignored and the term frequency vectors are appropriately normalized[65].

There are other reasons that, beyond locality, make ignoring IDF a good decision for this problem domain (Kejriwal & Miranker, 2014). One reason is that tokens in noisy entities may lead to undue noise in the IDF term. A possible way to address this problem is to divide the token weight in each vector by its total occurrence (in entities of all types) in the corresponding type document collection. This *vertical normalization* was found to work well in pilot serial experiments. Cosine similarity between vertically normalized TF vectors is still global, as the full collection must be scanned, but helps to compensate for IDF noise in small type document collections.

Pair selection strategy

Using a set similarity function, a type matrix can be populated, with each element in the matrix representing the similarity between two types. A pair selection strategy processes this matrix to output a potential type alignment. An obvious strategy for selecting an injective mapping to maximize the total sum of similarity scores is the *max. Hungarian* algorithm

---

64 Term Frequency-Inverse Document Frequency. The version of the TFIDF formula herein is primarily derived from the work by Cohen (2000).

65 An example normalization (denoted herein as an *L2-normalization*) is dividing each weight by the square root of the summed squares of all weights in the vector.

(Munkres, 1957). Among other applications, this algorithm has also been used for obtaining schema mappings from analogously defined *similarity matrices* (Bilke & Naumann, 2005). If $|T_1| = |T_2| = n$, the Hungarian algorithm runs in time $O(n^3)$; a similar result is obtained for unequal dimensions.

A more aggressive pair selection strategy, denoted *Greedy*, only picks elements that are the maximum in *both* their constituent row and column. This algorithm runs in linear time in the total number of type pairs ($O(|T_1||T_2|)$). It is aggressive because it prioritizes the precision of type alignment over the recall. In evaluations (Section 4.4), this strategy was found to effectively discover one-one mappings in instantaneous execution time66.

An intuition is provided on the scaling of Algorithm 4.1. Assuming that the number of types in either graph is not large[67], linear-time MapReduce jobs can be used to execute the similarity function on type document pairs and output the type matrix. The matrix, being small, can be shuffled to a *single* reducer, where the pair selection strategy is executed, and a type alignment is output (Chapter 8).

4.4 Evaluations

4.4.1 Test Cases

Table 4.1 describes the test cases used in these evaluations. All test cases are real-world and structurally heterogeneous.

66 Another intuitive strategy that also scales better than the Hungarian algorithm and is able to handle many-many mappings is a *threshold-based strategy*, which chooses any mapping that is above a pre-specified threshold. Along similar lines, a *ranking-based strategy* can also be defined. Threshold and ranking-based strategies prove important in Chapter 8, where the second type alignment application is evaluated in a large-scale setting.

67 A justification of this, based on analysis of encyclopedic graphs that are known to contain large numbers of types, is given in Chapter 8.

Name	Number of types	Number of entities	Number of true positive links
Case Law/Constitute (Colombia)	1/2	1204/2220	5577
Case Law/Constitute (Venezuela)	1/2	1503/1601	555
Joint Committee on Taxation (JCT) /US Treasury	5/5	1135/845	24,227

***Table 4.1**: Test cases used in type alignment evaluations.*

For a given country, *Case Law* consists of data that describe legal cases in that country. The *Constitute* datasets are derived from the Constitute project[68], and provide structured descriptions of a country's constitutions. In the tests considered here, only one of the Constitute datasets provided to the system should be linked to the corresponding Case Law dataset. The goal is to link *cases* in Case Law to the relevant *articles* in Constitute that were used in deciding that case. Such linkage problems are often referred to as *link discovery*, rather than instance matching, since the link may not necessarily have *:sameAs* semantics (Figure 4.2). The principles of non-adaptive link discovery systems are similar to those of non-adaptive instance matchers[69].

The third collection is more complex, and describes estimated US government budget data from 2009 to 2013. The data is provided independently by the Joint Committee on Taxation and the US Treasury. Data from a longer period may be found on a publicly accessible website[70]. The goal is to link budgetary allocations in the two sources that share the same function (e.g. health) in the *same* year. This test case also involves link discovery.

All datasets contain a wide variety of non-string data, such as numbers and dates (Figure 4.3). For this reason, they serve as good serial benchmarks

68 https://www.constituteproject.org/

69 Even when a component in such a system is adaptive, the difference arises if the system is also *unsupervised*. If manually labeled data is provided, the *semantics* of links have no bearing on statistical algorithms trained to detect them.

70 http://www.pewstates.org/research/reports/

for evaluating the first application of type alignment on real-world Linked Data.

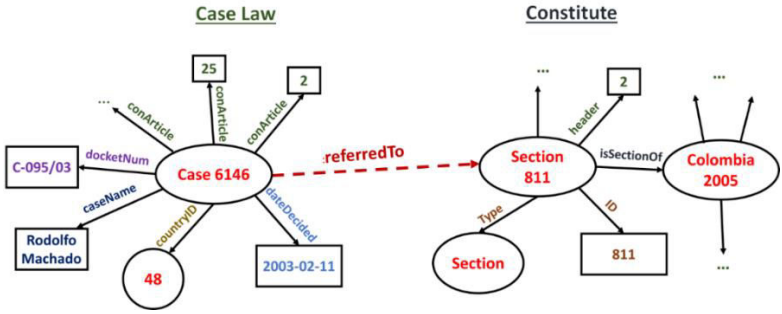


Figure 4.3: *Example of link between (Colombia) Case Law and Constitute.*

4.4.2 Metrics and Methodology

Within an instance matching application, a type alignment is successful if it leads to *efficiency* savings without a corresponding loss in *effectiveness*. A correct way to evaluate this is by using an underlying blocking algorithm as the baseline, and using the blocking metrics of *Pairs Completeness (PC)* and *Reduction Ratio (RR)* to respectively measure effectiveness and efficiency[71]. Namely, the blocking method is first executed on the two collections as if they were singly-typed graphs. A graph of PC vs. RR is plotted by varying blocking method parameters, described earlier in Chapter 2. Next, type alignment is performed on the collections and the same blocking method is executed on the instances covered by the type alignment. A graph of PC vs. RR is analogously plotted for this set.

Two blocking methods are considered to ensure that the assessment is generalizable. The first is the unsupervised *Canopies* (equivalently, *Canopy Clustering* or CC) method by McCallum, Nigam and Ungar (2000). The second is the extended (or heterogeneous) version of the *supervised* homogeneous Disjunctive Normal Form blocking scheme learner (DNF-BSL) briefly described in Chapter 2. The extended version is detailed in Chapter 7. The actual working of the blocking methods (and parameterization) is not

[71] Formulae for these metrics may be found in Section 2.2.3.

relevant for these evaluations[72], since (a) the parameters are varied over a range in order to generate a comprehensive PC-RR graph, and (b) tight controls are kept in place by ensuring that the only difference between the two blocking implementations is that one implementation includes type alignment as a preprocessing module, and the other does not.

Relatively simple methods are used for type alignment. Cosine similarity on vertically normalized TF vectors is used for the similarity function, and the *Greedy* method is used for the pair selection strategy. Note that the evaluations in this chapter are limited to a serial setting, where all datasets fit in main memory. The experiments were run on an Intel Core 2 Duo PC with 3 GB of memory and 2.4 GHz clock speed. All code was implemented in Java, and is freely available on the author's Github page[73].

4.4.3 Results and Discussion

Figure 4.4 illustrates the results on all three test cases. The heterogeneous DNF blocking scheme learner outperforms *Canopies* in all cases, but this is expected, since *Canopies* is unsupervised and explicitly attuned to the instance matching (rather than the generic link discovery) task. The DNF-BSL is trained on the links and is able to adapt to generic link discovery. The improvement does not represent a controlled result in the sense of one blocking method being universally better than the other.

The results indicate that type alignment, even with simple measures, significantly improves upon both blocking methods. There is one scenario when DNF blocking, by itself, seems to be sufficient. On the *Venezuela* test case, there is virtually no difference between the heterogeneous DNF-BSL baseline, and the method that employs type alignment. Thus, the success of type alignment must be interpreted with a caveat. Given enough training data, an adaptive DNF-BSL is able to compensate for type heterogeneity by virtue of searching in an expressive hypothesis space[74]. If this is not the case,

---

72 The issue was treated at length in the original publication (in Section 4.4; pg. 9). Briefly, *block purging* was used to control data skew in *Canopies*, and a fixed-size training set (300 positives and 300 negatives), along with a training method based on *bootstrap aggregating*, was used to train the DNF-BSL (Kejriwal & Miranker, 2014).

73 https://github.com/mayankkejriwal

74 The presumed reason for this adaptiveness occurring with *Venezuela* but not *Colombia* is that the relative proportion of training data was higher for *Venezuela* than *Colombia*, by virtue of fewer positive links in *Venezuela* (Table 4.1).

explicitly addressing type heterogeneity is in the best interest of a system designer.

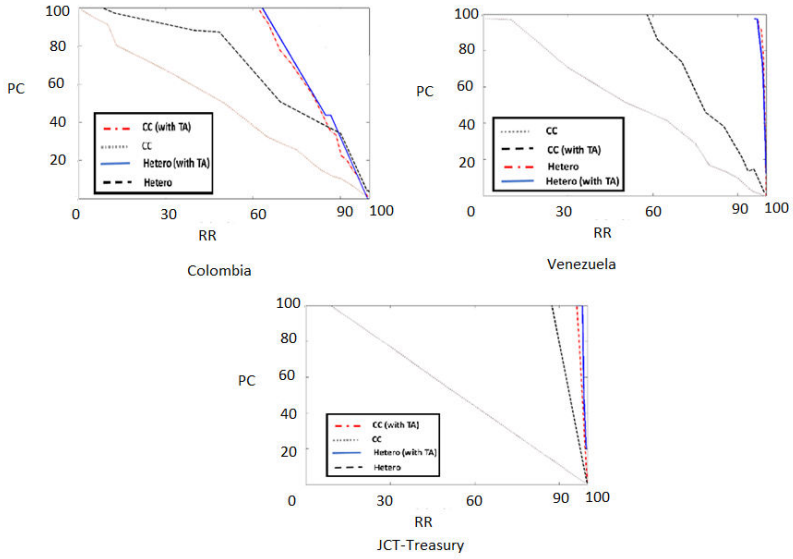


Figure 4.4: *Comparison of blocking techniques Canopy Clustering (CC) and Heterogeneous Blocking (Hetero), with and without type alignment (TA).* The relevant metrics are Reduction Ratio (RR) and Pairs Completeness (PC). For readability purposes, legend colors for Venezuela differ from the other two datasets.

A *post-hoc* analysis of the results showed that the alignments were *putatively* correct. As mentioned before, type alignment cannot rely on a human-annotated gold standard, as aligned types must exhibit semantic relatedness, only quantifiable by data-driven metrics (PC and RR in Figure 4.4). However, the test cases in these experiments have fairly standard semantics (on type alignment) and algorithm results agreed with putative judgments.

In auxiliary experiments, the benefits of type alignment, when interest is limited to only one type, are also evaluated. In the government finances test case, for example, it is likely that only matched instances from the most recent year are of interest. If type alignment is not performed, the only avenue is the baseline approach, followed by some manually defined filtering approach after the instance matcher terminates.

The benefit of type alignment is evaluated as follows. A candidate set is generated *only* for the type of interest[75], but the exhaustive set in the RR definition is the set of all entity pairs, regardless of type. Arguably, this is the correct way of evaluating type alignment for the scenario outlined above, since the baseline does not consider types at all. For comparison, a second, stricter definition of RR is also considered. Per this definition, the exhaustive set is the set of all entity pairs only for that type. One reason why this definition is useful is that, because PC and RR are always normalized to be between 0 and 1, a direct comparison is facilitated between the PC-RR tradeoffs of multi-type blocking and single-type blocking. A key research question here is whether type alignment has distributional implications for the PC-RR tradeoff[76].

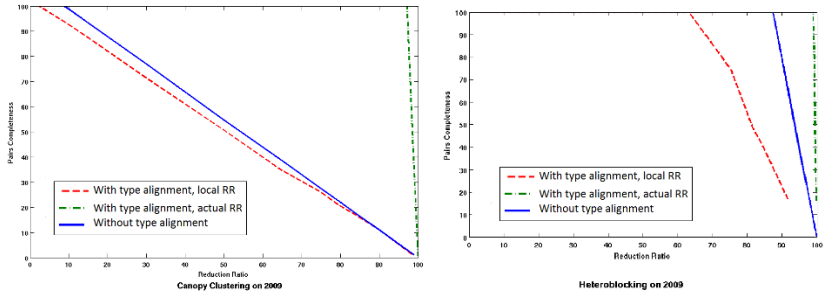


Figure 4.5: *Results of auxiliary experiments for Canopy Clustering and the heterogeneous DNF-BSL (Heteroblocking) for the single type 2009 in the US government test case.*

Figure 4.5 illustrates the results for the auxiliary experiment. *Local RR* uses the stricter definition of the exhaustive definition, while *actual RR* uses the first (and argued to be the correct) definition. The results show that, when the interest is bound to a single type, the benefits of type alignment are even greater. Concerning the aforementioned research question, distributional changes are minimal. For Canopy Clustering, the trends and slopes are similar. For the heterogeneous DNF-BSL, the similarity is less evident, especially near high PC values, but the trends become more similar for lower PC values. Note also that, although a comparison of trends between the *actual RR* scenario and

---

75 Recall that, in the first experiment, a union was performed on the candidate sets generated for the type alignment as a whole.

76 Another way of putting this is, 'does type alignment change the *shape* of the PC-RR curve for an adopted blocking method?'

the baseline (or the *local RR* scenario) is inapplicable, since entities from other types are completely ignored in the RR, the shapes of all curves are still quite similar, an encouraging sign that type alignment does not have unexpected impact on candidate set distribution.

Auxiliary experiments were also conducted for the other years in the US government test case, as well as for the Colombia and Venezuela test cases. The results were qualitatively similar and are not reproduced herein.

Chapter 5: Training Set Generation

Automation is one of the four goals in populating a Linked Data Entity Name System. In Chapter 3, it was shown that standard unsupervised techniques, such as iterative full-pass genetic algorithms and clustering algorithms, are often forced to assume away scalability (and in some cases, property heterogeneity) to function.

An approach by Christen (2008b) suggested an alternate approach, namely that of *training set generation*. The idea was that, if a good *seed* training set can be located through an appropriate set of assumptions, then the set could be used to bootstrap all adaptive processes in the overall instance matching schematic in Figure 1.5. Unfortunately, the approach developed by Christen relied on strong assumptions that do not hold in the majority of RDF test cases evaluated in the present set of experiments. In short, Christen's approach is unsupervised but is ill-suited to domain-independent, heterogeneous graphs. Other training set generators in the literature have also encountered empirical difficulties when evaluated on heterogeneous RDF graphs (Section 5.3.4)

The goal of this chapter[77] is to present a domain-independent training set generator (TSG) that is both unsupervised and performs well on RDF graph inputs. To the best of our knowledge, this is the first such TSG for heterogeneous graphs that yields outputs that can be viably employed in support of a full unsupervised execution of the instance matching pipeline in Figure 1.5. Because the TSG enables fulfilling automation, we consider it as the first (and primary) core contribution in support of this dissertation[78].

The generated training set may be noisy, since the underlying procedure relies on a set of heuristics and not human labeling effort. Learning procedures that rely on such training data must necessarily be robust to noise. Two such procedures, for property alignment (Chapter 6) and blocking key learning (Chapter 7), constitute the remaining core contributions in support of this dissertation.

---

[77] Some of the technical material in this chapter was previous published as part of an article in the Journal of Web Semantics (Kejriwal & Miranker, 2015c).

[78] Additionally, a MapReduce-based version of the algorithm is illustrated in Chapter 8, precluding the tension between automation and scalability that has been typical of other instance matchers (Section 3.2.1).

5.1 Intuition

For the purpose of this discussion, consider two people, *Michael Rogers* and *James Quinlan*, that are present in two different datasets *d1* and *d2*, belong to compatible types (e.g. *People* and *Individual*) and must be declared as being equivalent by a good instance matcher.

Figure 5.1 illustrates the RDF fragments describing the *information sets*[79] of these two entities. Assuming a set of heuristics that relies solely on token overlap and is agnostic to all other information, including structure and property labels, it is evident that such heuristics would output a higher score when comparing the information sets of *Michael Rogers* than the information sets of *James Quinlan*. For the *James Quinlan* entities, the zip codes differ by one, and the city *New York* (in the *d2* information set) has no equivalent phrase in the *d1* information set. The only token that is common, in fact, is *Quinlan*, a weak signal that would get ignored by any reasonably robust heuristic. This example can be made even more extreme by assuming *Quinlan* was mistyped as *Qinlan* in the second dataset. Such misspellings have a pattern (they *sound* the same) and are quite common in datasets that involved some form of manual data entry.

Assuming that the set of heuristics returns the pair of *Michael Rogers* entities as a training example, much can be learned from it, and others like it, in terms of a broad enough *fine-grained* feature-set that includes numeric, token, string and phonetic features. For example, initials and tokens (as opposed to an exact match) are important features in the labels of the entities, while the occupation is an unreliable indicator. Using other retrieved samples (not shown in Figure 5.1), a learner might conclude that, in conjunction with other features, an age is a reliable indicator. Zip codes are important but may differ by small margins. If *Quinlan* was misspelled as *Qinlan*, phonetic features would be particularly useful. Data-driven procedures can be executed on these heuristically located samples to reveal alignments between properties (Chapter 6). With the aid of these alignments and the feature-set mentioned above, blocking and similarity functions can be learned and executed (Chapter 7). This completes the execution of the pipeline in Figure 1.5.

---

79 Defined simply as a bag containing tokens (equivalently, 'words') that occur both in the entity URI itself, as well as the tokens in the object values and URIs that are one edge away from the entity.

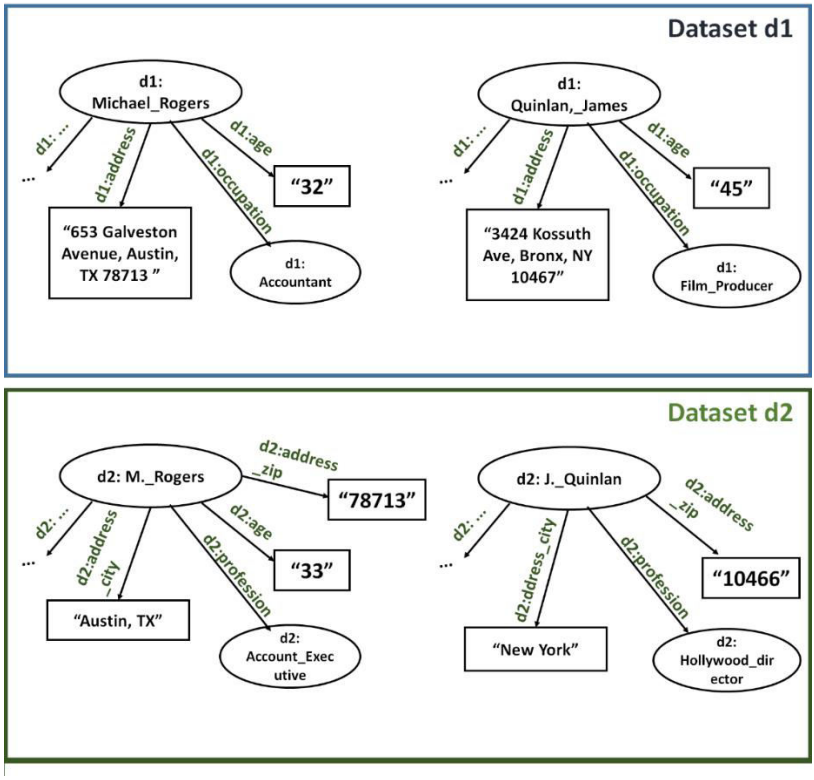


Figure 5.1: *An example illustrating the intuition behind the training set generator (TSG) detailed in this chapter.*

In short, the chief observation distilled from Figure 5.1 is that, in real-world data, not all training samples are *equally difficult*, and with a robust set of heuristics, easy samples can be located. In Section 5.2, we describe a combination of two heuristics that are used to locate such samples. Together, the two heuristics are found to significantly and consistently outperform a one-heuristic TSG in experimental evaluations (Section 5.3).

The easy samples generated by the TSG will be used to bootstrap a more expressive learning procedure (e.g. a machine learning classifier) that will, in turn, locate more difficult samples[80]. An important concern that arises

[80] There is a strong connection here between semi-supervised learning (and also EM), and the training set generator (TSG). In semi-supervised learning (Zhu & Goldberg, 2009), a seed labeled set is provided by a human. The TSG simply tries to automate this initial labeling effort by replacing human effort with

here is that the classifier could end up 're-learning' the heuristics that generated the examples in the first place (and thereby replicate the output of the TSG in a test phase). This concern is obviated by the use of the fine-grained feature-set briefly mentioned earlier (and detailed further in Section 7.1.1). Specifically, a change of feature representation ensures that the coarse heuristics cannot be *directly* learned by the machine learning classifier, which in turn, facilitates the discovery of additional 'harder' duplicates.

To ensure domain-independence, such features must not be designed for a specific dataset or domain. In practice, this issue can only be tested through independent evaluations on several different domains and test cases. A second important practice that we adopted to ensure domain-independence was to only use a single development dataset for refining the various components of the approach presented in Section 5.2. Although this chapter deals primarily with the design of a robust RDF training set generation algorithm, it also introduces the multi-domain test suite that will be used again in Chapters 6 and 7, where the generated training set is used for learning multiple functions.

5.2 Approach

Input: Property tables P_1 and P_2

Parameters n and $thresh$

Tokenizer T

Output: Set D of positive training samples

Set N of negative training samples

Steps:

1. Initialize empty *list* D_l
2. Initialize empty sets D and N
3. Convert each record in P_1 and P_2 to a bag-of-words document using T to tokenize each record

heuristics. The reason why an *unsupervised* version of EM is inappropriate was described in Chapter 3, and EM will also be used as a baseline in later chapters.

4. Collect *term frequencies* and *inverse document frequencies* over all documents
5. Collect all record pairs (r, s) with *Log TFIDF* score above $thresh$ in D' where $r \in P_1$ and $s \in P_2$
6. Compute *Jaccard* scores of all pairs in D'
7. Sort D' in descending order based on *Jaccard* score
8. Place in D the top $\min(|D'|, n)$ pairs in D' such that a record occurs at most once in any pair in D'
9. Permute pairs in D to get N distinct pairs such that $|N| = |D|, N \cap D$ is non-empty
10. Output D and N

Algorithm 5.1: *An algorithm for training set generation.*

An effective training set generator (TSG) must overcome at least two challenges. First, the TSG must yield reasonable results without being too expensive, otherwise it risks becoming the computational bottleneck in the full system. In practice, the run-time of an appropriate TSG should be near-linear, similar to other preprocessing steps such as blocking. The second challenge is the quality of the generated training set. Since the TSG relies on heuristics, at least some fraction of the training set is expectedly noisy, and training set precision falls rapidly as a function of coverage with respect to the ground-truth. Prior results on TSGs (verified by the results in Section 5.3) have demonstrated this fall in precision to start occurring at relatively low levels of coverage (Bilke & Naumann, 2005; Kejriwal & Miranker, 2013). This means that a high-quality training set risks not being representative enough, potentially leading to problems such as overfitting when training classifiers further down the pipeline. Conversely, the more representative the set, the noisier it is likely to be.

Algorithm 5.1 presents the pseudocode of a *serial TSG* that was designed with these two challenges in mind. The TSG tokenizes each record in a property table using a tokenizer T, and converts it into a bag-of-words document[81]. Drawing on standard information retrieval techniques that were

[81] Technically different from the type document described in Chapter 4, but the construction is similar.

also utilized in Chapter 4 (Cohen, 2000), term frequencies (TF) and inverse document frequencies (IDF) of tokens are computed.

In preliminary experiments, off-the-shelf tokenizers were found to be inadequate for the challenges (such as URI prefixes) posed by RDF elements in Linked Data. Based on these observations, a process of trial-and-error on an experimental dataset was used to design the tokenizer T used both in Algorithm 5.1 and in other algorithms developed in this dissertation. The tokenizer is designed to specifically handle the delimiters often encountered in URIs and other RDF elements[82].

The $Log\ TFIDF\ score$, defined herein as the *Cosine* similarity of two vectors[83] r and s, with $Log\ TFIDF$ weights, is given by the formula below:

$$Log\ TFIDF(r,s) = \sum_{q \in r \cap s} w(r,q)w(s,q) \qquad (5.1)$$

For any bag-of-words representation t, and word q:

$$w(t,q) = \frac{w'(t,q)}{\sqrt{\sum_{q \in t} w'(t,q)^2}} \qquad (5.2)$$

The function $w'(t,q)$ is defined as follows:

$$w'(t,q) = \log(tf_{t,q} + 1)\log(\frac{|P|}{df_q} + 1) \qquad (5.3)$$

The equations assume that r and s are records from RDF logical property tables P_1 and P_2 respectively, $w(t,q)$ is the *L2-normalized* TFIDF weight of a term q in a record t (from either property table), $tf_{t,q}$ is the term frequency of q in t, $|P| = |P_1| + |P_2|$ is the total number of records in both property tables and df_q is the number of records in which the term q appears. Note that IDF statistics are collected over both property tables.

Using the parameter $thresh$ as a filter, only the pairs with $Log\ TFIDF$ score above $thresh$ are retained in the list D_1. If $thresh$ is too high, there may be fewer than n pairs with score above $thresh$. In practice, setting $thresh$ to a default low value (such as 0.001) is found to suffice (Section 5.3). The rationale behind setting a low (but non-zero) $thresh$ is to eliminate the vast majority of pairs that share only unimportant tokens (such

[82] A specific but simple example is including the delimiter :// in the tokenizer. This delimiter is often encountered in Web URIs, and serves only a syntactic purpose.

[83] In a slight abuse of notation, r and s are also used as symbols for the bag-of-words representation.

as *http*). A default value of $thresh$ can be set in a self-tuning manner in an actual implementation; if fewer than n samples are returned, $thresh$ decreases by a small value till n samples are returned by Algorithm 5.1.

Efficient implementations of the $Log\ TFIDF$ function have been extensively researched in the information retrieval community and drawing on the prior work of Cohen (2000), lines 1-5 of Algorithm 5.1 can be implemented with guaranteed run-time $O(\alpha(|P_1||\mathrm{A}_1| + |P_2||\mathrm{A}_2|))$ where α is the slow-growth inverse Ackermann function, and A_i is the *attribute set*[84] of property table P_i (for $i = 1,2$).

The record pairs collected in D' are scored in line 6 using the previously described $Jaccard$ set similarity measure, first formally introduced in Chapter 4. Briefly, given two token-bags S_1 and S_2 as input, their $Jaccard$ similarity score is given by:

$$Jaccard(S_1, S_2) = \frac{S_1 \cap S_2}{S_1 \cup S_2} \tag{5.4}$$

As described in Chapter 4, a key property of $Jaccard$ is that it is a *local* similarity function in that it does not rely on external information sets (such as IDF) that may require a pass over the entire dataset. Instead, the dependence is only on the two arguments. Since $Log\ TFIDF$ and $thresh$ have already served as a filter for eliminating obvious non-duplicates, $Jaccard$ can be used to further refine and sort D'. In line 8, the top n (or $|D'|$, whichever is smaller) pairs in the sorted list are added to the output set D.

A natural question is if the $Jaccard$ refinement step is even necessary, given that $Log\ TFIDF$ is roughly accomplishing the same goals. In fact, the training set generator used by the *Dumas* schema matcher does not include this step, but sorts the list based on the $Log\ TFIDF$ scores and outputs the top n results (Bilke & Naumann, 2005). The rationale for including this step is that $Jaccard$ places higher emphasis on token overlap with respect to the union of the tokens-sets, and is agnostic to how common the tokens are in the other records. This aggressive strategy would expectedly lead to many false positives getting included in D if applied in an unfiltered setting, but $Log\ TFIDF$ has already filtered out non-duplicates with high token overlap.

[84] The attribute set (equivalently, *property set*) of a property table is the set of column labels in the table. For ease of presentation, the *subject* column is assumed to be included in this set, even though it is technically not an RDF property.

The empirical benefits of using two heuristics, instead of one, are demonstrated in Section 5.3.

A constraint in line 8 of Algorithm 5.1 is that a record (from either property table) occurs at most once in D'. Intuitively, this constraint attempts to make the training sets as representative as possible by preventing a single record from getting undue coverage in the training set. This relates directly to the quality-representation tradeoff earlier mentioned as a challenge in prior TSG work (Bilke & Naumann, 2005; Kejriwal & Miranker, 2013).

Example 5.1: Suppose $n = 3$ and the sorted list at the end of line 7 in Algorithm 5.1 is $D' = [(r_1, s_3), (r_2, s_5), (r_1, s_7), (r_6, s_1)]$ where r_i and s_j denote the i^{th} and j^{th} records in property tables P_1 and P_2 respectively. The chosen positive training set would then be $D = \{(r_1, s_3), (r_2, s_5), (r_6, s_1)\}$ since record r_1 has already appeared in a higher-scoring pair.

Non-duplicates can be automatically generated by relying on the observation that real-world datasets are often sparse in duplicates. This assumption is also predicated by the blocking step, which can only be applied if the vast majority of pairs are assumed to be non-duplicates (Christen, 2012). Line 9 in Algorithm 5.1 permutes the pairs in D to obtain new pairs ($\notin D$) that are assumed to be non-duplicates. For balanced training, $|N| = n$.

Example 5.2: Continuing from the previous example, where $n = 3$ and the generated duplicated-set was $D = \{(r_1, s_3), (r_2, s_5), (r_6, s_1)\}$, a possible non-duplicates set N generated by permuting D is $\{(r_6, s_5), (r_1, s_1), (r_2, s_3)\}$.

In practice, such a permutation is found to lead to near-perfect accuracy on the generated non-duplicates set (Section 5.3). Also, note that while lines 1-8 of Algorithm 5.1 (and by virtue, D) are deterministic, there are usually many possibilities for N. An alternative simple option for generating N is to randomly pair records in P_1 with records in P_2. One advantage of the adopted approach is that it is expected to exhibit less randomness, since both sets D and N are constructed using common records[85] and n is expected to be small compared to dataset sizes. Empirically, both methods were found to

[85] The probability of picking a non-duplicate record pair by randomly picking a pair from P_1 and P_2 is (assuming duplicates-sparsity) approximately $\frac{1}{|P_1||P_2|}$, whereas for the adopted approach, it is only of the order $\frac{1}{n^2-n}$.

yield near-perfect (98%+) accuracy, and either may be deployed in a practical application.

In Chapter 8, the scaling of Algorithm 5.1 is explored. Therein, a MapReduce-based TSG, with functionality similar to Algorithm 5.1, is elaborated upon.

5.3 Evaluations

Although the goal of this section is limited to evaluating the heuristic TSG, the test suite introduced herein is *also* used for evaluations in subsequent chapters, and is argued as being crucial for establishing domain-independence. Considerable space is allocated in Section 5.3.1 to describing the suite. All serialized datasets, code, experimental results and ground-truth files are available on a project website[86], and on the author's GitHub page[87].

The evaluations in this chapter (and in Chapters 6 and 7) were serially conducted on a 32-bit Ubuntu virtual machine with 3385 MB of RAM and a quad-core 2.40 GHz Intel 4700MQ i7 processor. The Student's t-test for paired sample means was used for statistical significance purposes, with the parameter[88] α set at 0.01.

5.3.1 Test Suite

The test suite introduced in this section is used for evaluations in this chapter, as well as Chapters 6 and 7. In total, there are ten test cases in the suite, each comprising a pair of individually serialized files[89] (Table 5.1). In total, the test cases cover almost twenty types, with six of the ten cases being multi-type. Many of these test cases are real-world benchmarks that have been made available through competitions and Semantic Web initiatives such as the Ontology Alignment Evaluation Initiative[90] (OAEI). We introduced three test cases, *Libraries*, *Parks* and *Video Game*, as benchmark contributions to the

86 https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/unsupervised-im

87 https://github.com/mayankkejriwal

88 The maximum Type I error rate, *given* that the null hypothesis of no difference (between means) is true.

89 Where relevant, / is used in Table 5.1 for distinguishing between the statistics of the two files.

90 http://islab.di.unimi.it/im_oaei_2014/index.html

instance matching community in an article that forms the primary published reference for *serial* implementations described in this dissertation (Kejriwal & Miranker, 2015c).

Note that type alignment on the files is not performed in either this chapter or the next two. The reason is that, in the majority91 of cases, the instances share the same type information. In other words, the type alignment problem has already been solved in these test cases, permitting a controlled evaluation of the other modules.

In keeping with the observation about Linked Open Data being *roughly schema-free*, the following descriptions do not distinguish between *object* and *datatype* properties. In the few test cases where separate OWL ontologies were provided, they were ignored.

ID	Name	Number of types	Number of properties	Property Alignments	Number of instances	True positives	Number of triples
1	Persons 1	2/2	15/14	15	2000/1000	500	9000/7000
2	Persons 2	2/2	15/14	15	2400/800	400	10,800/5600
3	Restaurants	2/2	8/8	7	339/2256	89	1130/7520
4	Eprints-Rexa	3/3	24/115	24	1130/18,492	171	4121/99,260
5	IM-Similarity	1/1	9/9	9	181/180	496	2204/2184
6	IIMB-059	5/5	31/25	23	1549/519	412	9995/8979
7	IIMB-062	5/5	31/34	30	1549/265	264	9995/22,058
8	Libraries	1/1	4/10	9	17,636/26,583	16,789	70,544/265,830
9	Parks	1/1	3/10	8	567/359	322	1701/3590
10	Video Game	1/1	11/4	4	20,000/16,755	10,000	220,000/48,132

Table 5.1: *Test cases used in domain-independent evaluations.*

91 In the few cases where the type labels were not identical, a cursory structural comparison (e.g. counting the number of properties) of the singly-typed property tables in each file yielded a trivial type mapping.

Test cases 1, 2 and 3

The first three test cases, *Persons 1*, *Persons 2* and *Restaurants* were first released by OAEI in 2010 and are described on the website92 as 'real data cases'. In the literature, there is some source of confusion about this, with at least one paper describing them as *synthetic* (Ngomo et al., 2013). *Restaurants* was originally a tabular dataset and is still widely used to evaluate *record linkage* systems[93] (Christen, 2008a). Along with describing restaurants and people (for the *Persons* test cases), these datasets also contain instances of type *Address*.

Test case 4

Eprints-Rexa is another publicly available benchmark in the Semantic Web community (Stoilos, Simou, Stamou & Kollias, 2006). *Eprints*[94] is a small dataset containing information about papers produced within the AKT research project, while *Rexa* was extracted by the Rexa search server[95] constructed at the University of Massachusetts. Both datasets are real-world and known to contain noise, although *Rexa* is believed to contain less noise than *Eprints* (Stoilos et al., 2006). This dataset is also the most heterogeneous dataset in terms of properties, since *Eprints* contains far fewer properties (and also instances) than *Rexa*.

Test case 5

Test case 5, *IM-Similarity*, describes books and was generated from real-world data using crowdsourcing[96]. It was released relatively recently (OAEI 2014), and the actual ground-truth had not been made available at the time of experimentation. To counter this, a reference alignment was manually created by using ad-hoc rules. Although the ad-hoc rules were framed to infer *:sameAs* links as closely as possible, there is always a possibility that the

92 http://oaei.ontologymatching.org/2010/im/index.html

93 In the record linkage literature, *Restaurants* is unambiguously considered real-world and not artificial (Christen, 2012b).

94 eprints.aktors.org

95 www.rexa.info

96 http://islab.di.unimi.it/im_oaei_2014/index.html

reference alignment contains noise. It is thus more appropriate to interpret this task as a *link discovery* task rather than the more specific instance matching task. Chapter 4 included a brief discussion on this task. Note also that this test case contains multilingual property values.

Test cases 6 and 7

Test cases 6 and 7 are over the *movies* domain and were artificially generated from real movie data using SWING, which injects controlled degrees of heterogeneity into an underlying corpus of real-world IIMB movie instances (Ferrara, Montanelli, Noessner & Stuckenschmidt, 2011). The types of heterogeneity (value, structural and semantic) were earlier described in a companion paper (Ferrara, Lorusso, Montanelli &Varese, 2008), and the datasets were introduced as instance matching OAEI benchmarks in 2010 (along with *Persons* and *Restaurants*). Eighty target datasets were generated by SWING from a common source. These were partitioned into four equal-sized folders, based on whether they contained only one of the three heterogeneities above, or all three (denoted as *comprehensive* heterogeneity in the OAEI report).

Two pre-generated SWING configurations (folder numbers 59 and 62 in the publicly available files) were randomly picked for the evaluations, with one containing only semantic heterogeneity (IIMB-059) and the other containing comprehensive heterogeneity (IIMB-062). Given the schema-free assumption, IIMB-059 is an interesting test of system performance when faced purely with semantic heterogeneity.

Test cases 8, 9 and 10

Test case 8 describes US libraries. The first file was from a Point of Interest (POI) website97 that allows users to upload GPS (Global Positioning System) data, and the second file was taken from a US government listing of libraries. Both files were extracted in the CSV (Comma Separated Values) format and were serialized as RDF property tables by treating each column name in the CSV file as an attribute.

---

97 http://www.poi-factory.com/poifiles

Test case 9 is similar to test case 8 except it describes national parks in the United States. Although *Libraries* is much larger than *Parks*, both datasets exhibit similar challenges of schema heterogeneity, since the first file in both cases contains fewer attributes than the second file. Another challenge is that, since both cases have files from POI websites, they contain longitude and latitude information. For many of the matching entity pairs, the values are not identical, which makes the task challenging for domain-independent instance matchers (such as the proposed system) that are not specifically configured for matching geo-locational data instances.

Finally, test case 10 describes video game information. The first file contains a sampling of video games extracted from DBpedia, while the second file was extracted as structured data (and converted to RDF triples in a manner similar to *Libraries* and *Parks*) from a reputable charting website[98]. Similar to *Libraries* and *Parks*, it only contains singly typed instances (Tian, Kejriwal & Miranker, 2014).

5.3.2 Metrics

Precision, *recall* and their *F-Measure* (harmonic mean) were chosen as the metrics in these evaluations. These metrics were formally defined in Chapter 2. To recap, precision is the ratio of true positives to the sum of true positives and false positives, while recall is the ratio of true positives to all positives in the ground-truth. In terms of a TSG described by either Algorithm 5.1 or a baseline TSG (Section 5.3.3), a curve of precision vs. recall can be plotted by varying the parameter n or the number of requested duplicates or non-duplicates. The ground-truth in these experiments is the set of true positives, the size of which is given in Table 5.1.

5.3.3 Setup

To the best of our knowledge, the Dumas TSG is the only other current system that automatically detects heuristic duplicates in *structurally heterogeneous* datasets (Bilke & Naumann, 2005), and is thus used as the baseline in this experiment. Dumas uses $Log\ TFIDF$ to locate the desired set of duplicates, essentially comprising lines 1-5 of Algorithm 5.1. First, the

[98] http://www.vgchartz.com

precision-recall tradeoff offered by the Dumas TSG is plotted against that of the re-sorted list output by lines 6-7 of Algorithm 5.1 by using the $Jaccard$ score. Statistical significance is measured by comparing the F-Measure series generated by the two systems using the paired t-test for sample means.

Although the curves are plotted by considering a range of values for the parameter n in Algorithm 5.1, later algorithms depend on a specific value of n, since n is used to tune the quality-representativeness tradeoff of Algorithm 5.1. Ideally, n should be large enough to adequately represent the characteristics of the underlying dataset, but not be so large that too many incorrectly labeled pairs get included in the generated training set. Towards this end, n was chosen to equal 500 for the second part of the experiment. That is, the top 500 elements from the re-sorted list are picked as duplicates, such that no instance is repeated more than once[99] (line 8 of Algorithm 5.1). The chosen duplicates are permuted (line 9 of Algorithm 5.1) to yield 500 non-duplicates. For fairness, the same procedure is conducted on the Dumas list and the resulting precision, recall and F-Measure of the 500 duplicates are reported. For the Dumas list, the results are reported both with and without the uniqueness constraint. Because the data is both deterministic and single-valued (i.e. not obtained as a series unlike the previous experiment), statistical significance testing does not apply to this part of the experiment.

The parameter $thresh$ in Algorithm 5.1 is set at 0.01 (and in self-tuning mode; see Section 5.2) for all experiments. The self-tuning functionality was never invoked, indicating that the default value of $thresh$ is typically adequate, even across the wide variety of test cases.

Finally, the precision of the 500 non-duplicates generated through permutation is also reported. Given that the vast majority of entity pairs are non-duplicates, computing the recall of the generated non-duplicates training set serves no purpose, since it is expected to nearly equal 0. The permutations are conducted across ten independent trials for each test case, and averages and standard deviations (of the resulting non-duplicates precision) are both recorded.

---

99 This constraint is denoted as the *uniqueness constraint*.

5.3.4 Results and Discussion

Figures 5.2 and 5.3 illustrate the results of the proposed TSG against the Dumas TSG based on whether the highest F-Measure achieved by either method was above 60%. Except on *Eprints-Rexa* and *Parks*, the proposed TSG outperforms the Dumas TSG. Except on *Parks*, the performance difference between the systems is statistically significant, with the test conducted over the two F-Measure series.

Closer investigation of the anomalous *Eprints-Rexa* result showed that the problem arose because of severe property heterogeneity. *Rexa* has 115 distinct property labels, while *Eprints* only has 24 distinct property labels. While the $Log\ TFIDF$ distance measure was able to somewhat compensate for this mismatch, the subsequent $Jaccard$ measure that was used to re-sort the list in lines 6-7 of Algorithm 5.1 led to a decline in the overall results. To test the hypothesis that property heterogeneity caused $Jaccard$ to perform so poorly, an additional experiment was conducted where the top 500 duplicates output by the initial run of the TSG on *Eprints-Rexa* was input to the hybrid property aligner (described in Algorithm 6.1 in the following chapter). The properties that were absent in the alignment set output by the aligner were discarded and the TSG was re-run.

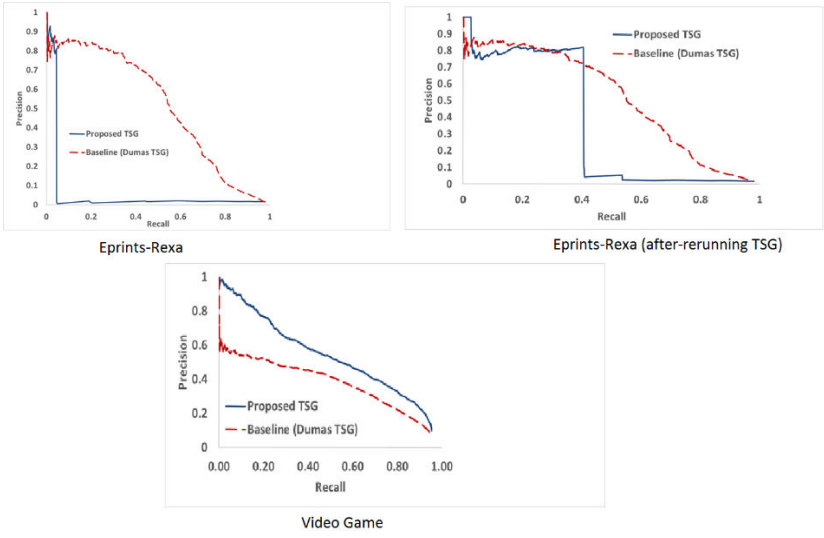


Figure 5.2: *Results for the test cases where maximum achieved F-Measure did not exceed 60%.*

The second figure in Figure 5.2 shows that this simple unsupervised step can be used to boost results in cases where the property heterogeneity is severe, even though this is not the primary purpose of the property aligner.

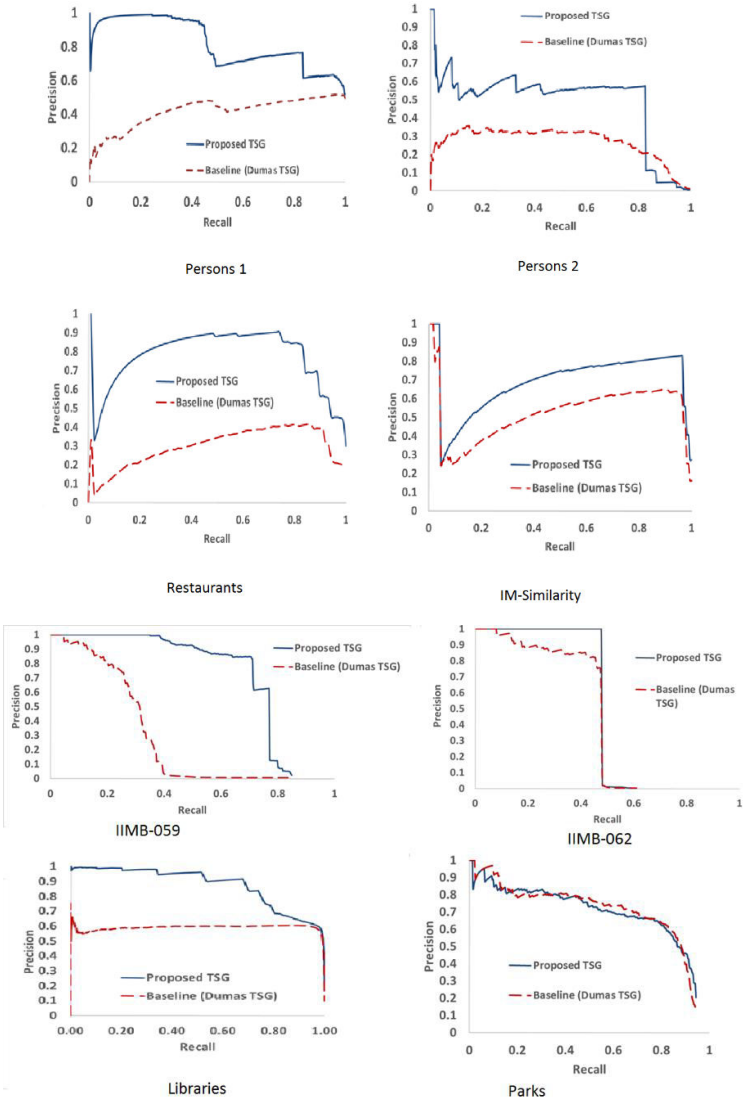


Figure 5.3: *Results for the test cases where maximum achieved F-Measure exceeded 60%.*

	Algorithm 5.1			Dumas-constrained			Dumas-original		
Name	**Recall**	**Prec.**	**FM**	**Recall**	**Prec.**	**FM**	**Recall**	**Prec.**	**FM**
Persons 1	75.20	75.20	**75.20**	47.80	47.80	47.80	47.80	47.80	47.80
Persons 2	23.75	19.00	21.11	23.75	19.00	21.11	39.25	31.40	**34.89**
Restaurants	100	36.18	**53.13**	100	36.03	52.98	100	17.80	30.22
Eprints-Rexa	43.40	43.40	43.40	39.20	39.20	39.20	68.00	68.00	**68.00**
IM-Similarity	96.49	92.18	**94.29**	96.49	92.18	**94.29**	97.66	33.40	49.78
IIMB-059	84.95	80.83	**82.84**	84.47	80.37	82.37	33.74	27.80	30.48
IIMB-062	67.80	67.80	67.80	73.23	73.23	**73.23**	47.73	25.20	32.98
Libraries	100	100	**100**	41.40	41.40	41.40	60.40	60.40	60.40
Parks	73.60	66.95	70.12	75.16	68.56	**71.70**	85.71	55.20	67.15
Video Game	88.60	88.60	**88.60**	61.00	61.00	61.00	56.00	56.00	56.00
Average	*75.38*	*67.01*	*69.65*	*64.25*	*55.88*	*58.51*	*63.63*	*42.30*	*47.77*

Table 5.2: *Comparative results for three training set generation systems with fixed parameters.* Prec. and FM stand for Precision and F-Measure respectively. All values are percentages.

Table 5.2 shows the precision, recall and F-Measure of the top 500 duplicates retrieved from the lists returned by the Dumas TSG and Algorithm 5.1, such that no instance is ever repeated more than once in the set of duplicates. Since the original Dumas TSG does not apply the uniqueness constraint, the results of retrieving the actual top 500 duplicates (regardless of whether instances are repeated) from the Dumas list are also reported alongside. Note that the recall metric is computed differently in Table 5.2. Specifically, the number of true positives in the retrieved 500 duplicates is divided by the quantity $\min(500, |\Omega_m|)$, where $|\Omega_m|$ is the actual number of matching entities (Table 5.1), instead of $|\Omega_m|$ (as with traditional recall computation[100]). The table shows that the described TSG equals or outperforms the baseline systems on six datasets. On two of the remaining datasets, its F-Measure is within 6% of the winning F-Measure and on average, the system outperforms the baselines on all metrics.

[100] The reason for bounding the denominator in this particular experiment is to prevent the recall from exceeding 100%.

Finally, the set of 500 non-duplicates generated by permuting the set of duplicates obtained from the three systems in Table 5.2 had high overall quality, with average precision on all test cases (and for all three systems) at least 98%, and with less than 1% standard deviation across ten independent trials per test case and system. At a p-value of less than 0.01, the difference between the three setups was not found to be statistically significant for any of the test cases at the 99% level. Since the results on all ten test cases were near-identical, they are not tabulated.

Chapter 6: Property Alignment

This chapter[101] describes property alignment, which is one of the three functions that must be learned using the generated training set. Property alignment is, in principle, similar to type alignment but is finer-grained, and as argued in both Chapters 1 and 2, has implications for both quality and complexity of overall instance matching.

Figure 6.1 illustrates the goal of property alignment in the overall context of the running example introduced earlier in the dissertation. A property alignment, which is a set of *semantically related* property (equivalently, attribute) pairs, must be output for each pair of aligned types. Caveats noted about type alignment in Chapter 4 (e.g. about the problem being ill-defined and data-driven) also apply to property alignment.

In a series of exploratory experiments, we found that the performance of a property alignment algorithm on the *recall* metric is tightly coupled with the success of later steps[102]. High-recall (i.e., with recall greater than 80%) property alignment is thus a bottleneck that is required for successful execution of the overall pipeline. While existing solutions from the literature achieve high recall on some test cases (Section 6.2), they do not achieve the required levels (on average) on the ten test cases introduced in the previous chapter.

If high recall was all that was required, an algorithm could simply output all possible property alignments. Such an output would have 100% recall, but would not be useful because its precision would be trivially low. A viable algorithm must achieve high recall while still maintaining moderately high precision. The sensitivity of the precision-recall tradeoff to tunable parameters further suggests that the algorithm should effectively be *parameter-free* to achieve the desired balance.

This chapter presents a property alignment algorithm that is designed to accommodate the requirements above in a domain-independent fashion. To

---

[101] The technical material in this chapter was previous published as part of an article in the Journal of Web Semantics (Kejriwal & Miranker, 2015c).

[102] More precisely, those preliminary experiments (not reproduced herein) showed that a key metric (*Reduction Ratio*) of the blocking algorithm was highly correlated with the recall of property alignment.

the best of our knowledge, this is the first such algorithm that achieves consistently high recall without requiring any parameter tuning. For this reason, we consider it as the second core contribution of this dissertation, similar to the training set generator in Chapter 5.

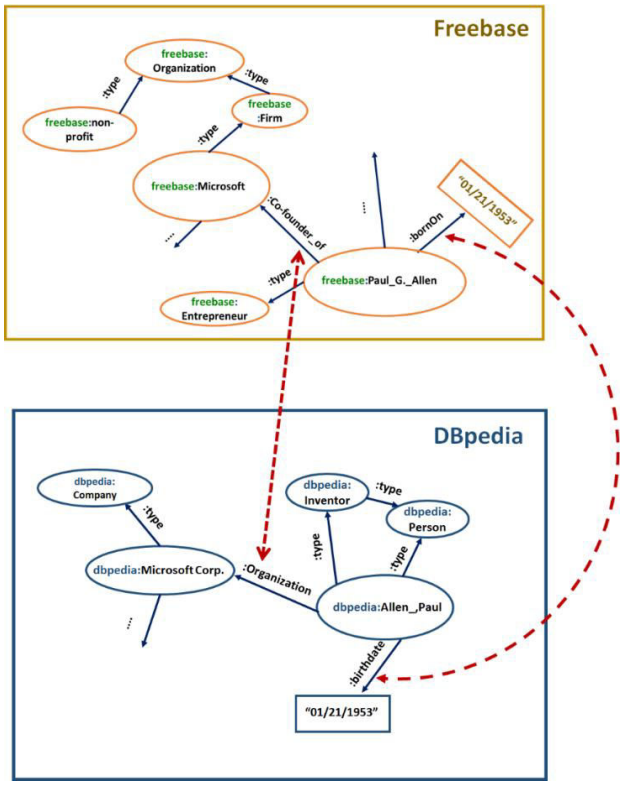


Figure 6.1: *An illustration of property alignment.* Note that, despite the visual similarity to *:sameAs* properties, these lines do not represent actual property declarations; hence, are unlabeled. Also, unlike the dashed lines in previous figures, the alignments in this figure are between edges, not nodes.

6.1 Approach

This section presents an algorithm for performing alignment between the attribute sets A_1 and A_2 of the two property tables P_1 and P_2 respectively. The alignment algorithm only uses the training samples (generated by Algorithm 5.1), and not the full datasets. This observation impacts the design of a scalable implementation in Chapter 8.

For notational succinctness, refer to an attribute of A_1 as a_i^1 and an attribute of A_2 as a_j^2 where i and j range from 1 to the number of attributes in A_1 and A_2 respectively. Using this notation, an *aligned attribute pair* is defined as follows.

Definition 6.1 **(Aligned Attribute pair)** Given attribute sets A_1 and A_2 from two RDF datasets D_1 and D_2 respectively, an attribute pair $(a_i^1, a_j^2) \in A_1 \times A_2$ is said to be *aligned* if a_i^1 and a_j^2 are *semantically related*.

Let Q denote a set of aligned attribute pairs. In keeping with the terminology first introduced in Chapter 4, Q is henceforth referred to as a *property alignment.* If the attribute sets of the input property tables are interpreted in a manner similar to Relational Database (RDB) schemas, Q is like a set output by a schema matcher with local $1:1$ cardinality but global $m:n$ cardinality. The following example illustrates the concept.

Example 6.1: Consider the two property tables in Figure 6.2. The property alignment Q should ideally contain the alignments *(Subject, Subject), (d1:hasWife, d2:spouse), (d1:hasBrother, d2:sibling), (d1:hasBrotherInLaw, d2:inlaw), (d1:year, d2:birthdate), (d1:month, d2:birthdate), (d1:day, d2:birthdate)*, since alignments can be partial[103]. The *global cardinality* is $m:n$ since an attribute participates in more than one alignment in Q. The local cardinality is $1:1$ since each alignment is between two attributes and not two sets of attributes.

Herein, an aligned pair (a_i^1, a_j^2) is meant to indicate semantic relatedness between the i^{th} and j^{th} columns of P_1 and P_2 respectively. The alignment set Q is not important by itself, but like the training set, will prove to be an important input to the feature generator component described in Chapter 7. Intuitively, the alignment set will enable the feature generator to constrain the size of the feature space (Chapter 7).

[103] The alignment is referred to as partial because the notion of semantic relatedness between two attributes is closest to the notion of a *partial overlap* between their set representations.

Subject	d1:hasWife	d1:hasBrother	d1:hasBrotherIn Law	d1:date_ of_birth	d1:year	d1:month	d1:day
d1:Mike_B eats	d1:Joan_Cr ax_Beats	*null*	d1:Sam_Crax	d1:Date	*null*	*null*	*null*
d1:Joan_Cr ax_Beats	*null*	d1:Sam_Crax; d1:Roger_Crax	*null*	*null*	*null*	*null*	*null*
d1:Date	*null*	*null*	*null*	*null*	1950	"December"	12
d1:Sam_Cr ax	*null*	*null*	d1:Mike_Beats	*null*	*null*	*null*	*null*

Property table 1

Subject	d2:spouse	d2:inlaw	d2:birthdate	d2:sibling
d2:Michael_Wa tt_Beats_Jr.	d2:Joan_Beats	d2:Samuel_Crax	12/12/1950	null
d2:Joan_Beats	d2:Michael_Watt _Beats_Jr.	null	null	d2:Samuel_Crax
d2:Samuel_Crax	null	null	null	d2:Joan_Beats

Property table 2

Figure 6.2: *Two single-type RDF graphs, serialized as logical property tables, used as running examples in this chapter for illustrating property alignment*.

One solution to generating an alignment set is to use an instance-based matcher such as Dumas (Bilke & Naumann, 2005). As earlier stated, Dumas generates noisy duplicates using its own TSG and performs (both global and local) 1:1 schema matching. As the simple example of Figure 6.2 shows, global 1:1 pairings are not adequate for this task and could lead to loss of information.

Dumas also does not consider the *names* of attributes, which can be quite indicative, especially in Linked Data property namespaces (Papadakis, Demartini, Fankhauser & Kärger, 2010). As a simple application of this finding, consider that the first column of every property table is always named *subject*; the pair *(subject, subject)* should always be included in Q. At the same time, the *birthdate* pair in Example 6.1 shows that *only* considering the names can be problematic, since the attribute *d1:date_of_birth* is lexically more similar to *d2:birthdate* than *d1:year*, *d1:month* and *d1:day*.

Similar issues arise if *column-based matchers* are adapted instead of instance-level matchers (e.g. Dumas). Column-based matchers match columns based on the degree of overlap between their value-sets. Several

property aligners used both within and without the context of instance matching employ a similar technique based on *extensional* (or *object-value overlap*) of RDF properties[104]. Some of these were described in Chapter 3. Both Dumas and a generic column-based matcher are used as baselines when evaluating the property alignment step.

To address the described challenges, a *hybrid parameter-free* property aligner is presented. The aligner considers both the names of the properties, as well as columnar aggregations of training data. The thesis is that, by using a judicious combination of informative signals, robust performance can be achieved, especially with respect to the recall metric.

Algorithm 6.1 shows the pseudocode of the property aligner. First, the algorithm strips the URI prefixes of property columns[105] and uses a basic exact-match indexing procedure on the resulting URI stems (after converting them to lower case strings) to heuristically determine the trivial $1:1$ alignments (lines 1-5). An obvious consequence is that Q is guaranteed to include the pair *(subject, subject)*.

Input: Sets $D$and N of positive and negative training samples respectively

Attribute sets A_1 and A_2

Output: Property Alignment Q

Steps:

1. Initialize empty set Q
2. Initialize numeric variable *avg* := 0
3. Initialize empty $|A_1|\times|A_2|$ dimensional matrix M
4. **for all** attribute pairs (a_i^1, a_j^2) in $A_1\times A_2$ **do**

   if URI stems[106] of a_i^1 and a_j^2 exactly match **then**

   Add (a_i^1, a_j^2) to Q

---

[104] In the Raven system, for example, stable matching of properties primarily relied on object-value overlap (Ngomo, Lehmann, Auer & Höffner, 2011).

[105] The subject column is an exception since it is not a URI; it is assumed to be its own URI stem.

[106] A URI stem is the part of the string that follows the URI prefix. For example, in Figure 6.1, the URI stem of *dbpedia:Allen_,Paul* is *Allen_,Paul*.

end if

5. end for
6. **for all** attribute pairs (a_i^1, a_j^2) in $A_1 \times A_2$ **do**

$$M[i,j] \coloneqq ColumnSim\left(D, a_i^1, a_j^2\right) - ColumnSim(N, a_i^1, a_j^2)$$

7. end for
8. **for all** pairs $(a_i^1, a_j^2) \in Q$ **do**

$$avg += \frac{ColumnSim\left(D, a_i^1, a_j^2\right) - ColumnSim(N, a_i^1, a_j^2)}{|Q|}$$

9. end for
10. **for all** entries in M **do**

if entry $M[i,j] \geq avg$ **then**

Add (a_i^1, a_j^2) to Q

end if

11. end for
12. Output Q

Algorithm 6.1: *An algorithm for property alignment.*

Before describing the rest of the algorithm, the $ColumnSim$ score over a set (e.g. D) of n record pairs $D = \{(r_1, s_1), \dots, (r_n, s_n)\}$ is computed as follows. The $ColumnSim$ function takes as input D and two attributes, $a_i^1 \in A_1$ and $a_j^2 \in A_2$. Denote as R and S the tables containing the records $r_1, \dots, r_n$, and $s_1, \dots, s_n$ respectively. $ColumnSim$ tokenizes the i^{th} and j^{th} *columns* (using the same tokenizer T in Algorithm 5.1) of R and S respectively to obtain two sets of tokens[107], R' and S'. The $Jaccard$ score, with a formula provided in Chapter 5, of R' and S' yields the final $ColumnSim$ score.

[107] Reserved keywords like *null* are automatically excluded from token sets being processed at any stage of the pipeline.

In lines 6-7, a matrix M is populated, with the $[i, j]^{th}$ cell of the matrix containing the value obtained by subtracting the $ColumnSim$ scores of the corresponding attributes a_i and b_j over D and N. The subtraction serves as a conservative filter to prevent accidental matches from happening.

Using the current alignments in Q (obtained earlier through exact matching of URI stems), the average score of matrix cells corresponding to elements in Q is computed as avg, and used as an automatic threshold to pick property alignments (line 10). The resulting alignment set Q is then output (line 12).

Using a hash-based method, the loop in line 4 runs in time $O(|A_1| + |A_2|)$. Assuming (based on characteristics of commonly encountered real-world data) that within an attribute set A, no two attributes have the same URI *stems*[108], Q (at the end of line 5) has maximum cardinality $min(|A_1|, |A_2|)$. Populating the matrix in lines 6 and 7 can be done in time $O((|D| + |N|)|A_1||A_2|)$, making this the most expensive step of the algorithm. The run-time of this step subsumes the computations in the remainder of the algorithm, since lines 8-11 require two passes over the matrix M.

In practice, Algorithm 6.1 was found to run near-instantaneously (less than a minute) even in the case of a benchmark with over a hundred properties (Section 6.2). The parameter-free nature of Algorithm 6.1 lends it an advantage in that it can be run like an off-the-shelf black box by a practitioner, precluding the need for cumbersome parameter tuning. To the best of our knowledge, a hybrid parameter-free property aligner does not exist in the current research literature.

Scalability of the algorithm (using MapReduce) is based on the same premise as the scalability of the type alignment algorithm in Chapter 4. The intuition therein was that, *once* the type matrix was constructed, all its elements could be routed to a single Reducer, and the Reduce program would essentially be a serial type alignment strategy. In the current scenario, a similar intuition applies: once the training set is generated, it can all be routed to a single Reducer. Algorithm 6.1 is used as the Reduce program. Note that an

[108] The actual implementation does not fail if this assumption is occasionally violated. It is invoked here mainly for the sake of analysis.

assumption here, justified in Chapter 5, is that the training set is not too large[109].

6.2 Evaluations

The test suite used for the evaluations in this chapter was described in Chapter 5, and summarized in Table 5.1. The column 'Property Alignments' in Table 5.1 contains the number of pairs in a putative (i.e. manually constructed[110]) property alignment ground-truth.

6.2.1 Setup

The 500 duplicates and non-duplicates output by the proposed TSG are input to the hybrid property aligner described by Algorithm 6.1. Two baselines are used in this experiment to illustrate the benefits of a hybrid aligner. The first baseline is the Dumas schema matcher (Bilke & Naumann, 2005), which uses the noisy duplicates generated by the Dumas TSG (*without* the uniqueness constraint; see Table 5.2). The matcher computes a *similarity matrix* for each of the n duplicates, and then aggregates them into a single matrix on which the *max. Hungarian algorithm* is executed (Munkres, 1957). Recall, from Chapter 4, that the Hungarian algorithm is a generic procedure which assigns a different row to each column[111], such that the total sum of values in the chosen cells is maximized over all valid assignments. Because the procedure cannot assign the same row to two different columns, Dumas can only output an alignment set with global[112] $1{:}1$ cardinality. When evaluating Dumas, a full parameter sweep was conducted to ensure optimal performance. Thus, the number of generated duplicates was not fixed at 500 for Dumas, but tuned for each test case. Only the best results are reported for Dumas.

---

109 If it is, the situation can still be resolved by building the matrix M in Algorithm 6.1 using parallel algorithms, and then treating M like the type matrix. This precaution is usually unnecessary as instance matching applications do not involve 'large' schemas (Christen, 2012a).

110 Unlike with type alignment, the use of a putative ground-truth for evaluating property alignment is inescapable. Blocking metrics cannot be used, since instances are *composed of* properties and property values (unlike types and blocks, which are each composed of instances).

111 Without loss of generality, assume that the number of columns is no greater than the number of rows.

112 That is, each property can participate in at most one match.

The second baseline is denoted as the *Column Matcher*, and uses similar principles as property aligners proposed in recent Semantic Web instance matchers such as Raven (Ngomo et al., 2011c). The Column Matcher directly constructs a single similarity matrix by computing the $Jaccard$ score of all values in two columns corresponding to a cell of the similarity matrix. Unlike Dumas and Algorithm 6.1, the *Column Matcher* does not use a training set. Once the similarity matrix is constructed, all values above a threshold are output as a match. Similar to the evaluations over Dumas, a full sweep is conducted over the threshold range $[0,1]$ to ensure optimality. Note that the parameter sweeps confer an empirical advantage on both baselines, since the presented aligner takes as arguments the top 500 samples (with the uniqueness constraint) output by the training set generator described in Algorithm 5.1, and is parameter-free. Note that, on *Eprints-Rexa*, the top 500 samples output by the original TSG run (not the re-run; see Figure 5.2) are passed as arguments to Algorithm 6.1 to avoid biasing the results.

The metrics for these experiments are precision, recall and their F-Measure, defined as earlier. Since all parameters have already been fixed (and for the baselines, at optimal values), single-point estimates are obtained and tabulated. Graph plots are not applicable to this scenario.

6.2.2 Results and Discussion

The results of property alignment are tabulated in Table 6.1. The superior performance of both Dumas and Algorithm 6.1 against the *Column Matcher* presents a strong case for the use of instance information (even with noise present) when aligning the properties.

Note that, although Algorithm 6.1 is outperformed by the baselines on six of the test cases, it is more balanced in its precision-recall tradeoff and outperforms, on average, both baselines by over 10% in terms of F-Measure. Algorithm 6.1 scores below 75% on precision on only two of the datasets (*Libraries* and *Parks*), and never below 75% on recall. Dumas scores below 75% on recall on half the datasets. The *Column Matcher* is even more skewed, with less than 75% recall on eight of the datasets.

Given that the alignment set is not intended to be used as an output in itself but for building a tractable feature space (Chapter 7), this distinction is important, since property alignment recall is more important in the context of

the overall instance matching task than precision. Intuitively, recall matters more than precision[113] because not every feature in the feature space (utilized by subsequent learning algorithms) has to be 'high-performing', but the absence of good features (due to low recall) potentially leads to degraded blocking and classification recall. By this argument, the machine learning algorithms simply ignore the features that are 'not useful', so a few wrong pairs in the alignment are not expected to have significant impact on blocking or classification.

	Algorithm 6.1				**Dumas**			**Column Matcher**		
Name	**Recall**	**Prec.**	**FM**		**Recall**	**Prec.**	**FM**	**Recall**	**Prec.**	**FM**
Persons 1	80.00	100	88.89		93.33	100	**96.55**	73.33	100	84.61
Persons 2	85.71	80.00	82.76		92.86	86.67	**89.66**	83.33	66.67	74.07
Restaurants	85.71	100	**92.31**		71.43	62.50	66.67	71.43	71.43	71.43
Eprints-Rexa	100	92.31	**96.00**		33.33	33.33	33.33	4.17	100	8.00
IM-Similarity	100	81.82	90.00		100	100	**100**	88.89	61.54	72.73
IIMB-059	100	82.14	**90.19**		78.26	72.00	75.00	60.87	60.87	60.87
IIMB-062	100	100	**100**		16.67	16.13	16.40	10.00	100	18.18
Libraries	100	22.50	36.73		33.33	75.00	46.15	55.55	62.50	**58.82**
Parks	100	26.67	42.11		37.50	100	**54.55**	37.50	100	**54.55**
Video Game	75.00	75.00	75.00		100	100	**100**	50.00	100	66.67
Average	***92.60***	*76.04*	***79.40***		*65.67*	*74.56*	*67.83*	*53.51*	***82.30***	*56.99*

Table 6.1: *Comparative results for three property alignment systems.* Prec. and FM stand for Precision and F-Measure respectively. All values are percentages.

A last point to note is the proposed aligner's robustness to noise, as exhibited by the performance on *Eprints-Rexa*, where the generated set of 500 duplicates had an F-Measure below 50% (Table 5.2).

[113] Even on the precision metric, both baselines score below 75% roughly half of the time, exhibiting the unpredictable nature of their performance.

Chapter 7: Blocking and Classification

This chapter covers the final steps of the schematic in Figure 1.5, namely blocking and similarity[114]. In Chapter 2, a computational motivation for this two-step formulation was described: quadratic complexity of exhaustive pairwise instance matching is untenable, even for datasets containing only thousands of entities. In the blocking step, a blocking scheme is used to cluster entities[115] into (possibly overlapping) blocks. Earlier discussions on the advantages[116] of DNF blocking schemes (Chapter 2) suggest their application to RDF data. Prior work assumed a strictly homogeneous Relational Database framework for the formalism and learning of DNF blocking schemes, making their application to heterogeneous RDF graphs an uncertain prospect (Michelson & Knoblock, 2006; Bilenko, Kamath & Mooney, 2006). Another issue arises in the extant learning algorithm, which assumes a supervised setting with perfectly labeled training data, as opposed to the noisy data generated by the training set generator described in Chapter 5.

As a third core contribution in support of this dissertation, we present both formalism and a learning algorithm for DNF blocking schemes that execute on RDF entities. Empirically, the learned schemes are shown to be competitive with, and in many cases, outcompete, a leading blocking algorithm designed for RDF datasets (Section 7.2.1).

Once blocks are formed, a blocking method is used to form a candidate set. In the similarity step, each entity pair in the candidate set is transformed to a feature vector, which is then classified by a previously trained machine learning model. Although rule-based, distance-based and other similarity techniques have also been explored in the research community (over a period of 50 years), machine learning has emerged as a dominant similarity paradigm (Christen, 2012a; Bilenko & Mooney, 2003). For this reason, the similarity step is equivalently referred to as the *classification* step in the rest of this chapter.

---

114 The technical material in this chapter was previous published as part of an article in the Journal of Web Semantics (Kejriwal & Miranker, 2015c).

115 Recall that an RDF entity is operationally equivalent to a record in a property table serialization.

116 Two advantages were that they can be learned from training data, and have exhibited superior empirical performance compared to alternatives.

Practical implementations of blocking, feature generation and similarity are not quite as straightforward. Several details have to be worked out, the most important of which is the specific feature space in which to represent each pair of entities. Also, the algorithms in this chapter take as input both the automatically generated training set (Chapter 5) and the property alignment (Chapter 6), which adds a layer of complexity to their development and evaluation. Since the feature generation and learning procedures are conceptually related, their treatment is covered in this single chapter.

7.1 Approach

7.1.1 Feature Generator

The generated training set and property alignments, output by Algorithms 5.1 and 6.1 respectively, are now input to a feature generator that converts each record pair in the training set to a feature vector. The features are subsequently described. The output of the feature generator comprises two sets containing n feature vectors each, where n is the number of duplicates[117] in the training set.

The property tables, P_1 and P_2 introduced in Figure 6.2 are used as running examples in this chapter as well. Recall that the attribute sets of P_1 and P_2 were denoted by the symbols A_1 and A_2 respectively. An attribute in A_1, representing the i^{th} column in P_1, is denoted by the symbol a_i^1; similarly, for an attribute in A_2. Finally, the symbols r and s are again used to denote generic records from P_1 and P_2 respectively.

Using the symbol $*$ for the Kleene star, a *property-specific indexing function* (P-SIF) is defined as follows:

Definition 7.1 (**Property-specific indexing function**) Given an alphabet Σ, a property table P, and an attribute a_i from the attribute set of P, a *property-specific indexing function* (P-SIF) is defined as a function $h_i: P \rightarrow 2^{\Sigma^*}$ that takes as input a record from the table P and is applied on the attribute value

117 And also the number of non-duplicates, since balanced training was assumed in the TSG (Section 5.2).

of the record corresponding to a_i. The resulting output is a set Y of strings over the alphabet Σ.

While technically possible to construct a special P-SIF h_i for the i^{th} column of the table, it is more appropriate for an unsupervised procedure to consider a set G of *general indexing functions* or GIFs. A GIF is a generic function that accepts a primitive data type (taken to be *String*, without loss of generality) as input and returns a set of primitive data types (i.e. Strings) as output. Given such a property-agnostic set G and an attribute set A of some property table, the set H of all possible P-SIFs can be constructed by forming the Cartesian product of G and A. If some function in G is inapplicable to an attribute in A, the P-SIF returns the empty set.

Example 7.1 Consider the first tuple of P_1 in Figure 6.2 and the simple GIF $Tokens$, which accepts a string as input, tokenizes it and returns the set of tokens as output. Applied to each of the eight attributes in A_1, eight P-SIFs $h_1, \ldots, h_8$ can be constructed. For the *null* attribute values, an empty set is returned. On the other hand, consider a GIF $AddOneToIntegers$. This GIF would also parse the tokens in the string but it would discard all non-integer tokens. The tokens that can be parsed as integers would be incremented, re-cast as strings and collectively output as a set. $AddOneToIntegers$ would only be applicable to certain numeric attributes (such as *d1:day* in A_1), and would return the empty set for all others.

In the rest of the chapter, the sets H_1 and H_2 of P-SIFs are assumed to be formed over respective attribute sets A_1 and A_2, by taking the cross-product of the attribute set with a *given* set G of GIFs. Thus, it is always the case that $|H_i|$ equals $|G||A_i|$, for $i = 1,2$. The set G forms the atomic feature set, from which feature spaces for each property table are individually constructed by using the attribute set. The twenty-eight GIFs used in the system are described below.

(1) Identity: Returns a singleton set containing the string.

(2) Tokens: Tokenizes the string based on a set of delimiters specifically designed for RDF elements, and outputs the set of tokens.

(3) Integers: Similar to (2) but discards all strings in the output that cannot be parsed as integers.

(4) ManipulateIntegersByOne: Same as (3), except that for every integer a, integers $a - 1$ and $a + 1$ are converted to strings and added to the output set

along with a. The GIF $AddOneToIntegers$, described in Example 7.1, is a simplified version of this GIF.

(5-7) ExtractNCharPrefixes: Same as (2) except that each token is further truncated to its first N characters. If the token has fewer than N characters, it is left intact. Three GIFs were implemented, with N set to 3, 5 and 7 respectively.

(8-10) ExtractTokenNGrams: Tokenizes the string as an ordered list and extracts length-N contiguous subsequences of tokens. If the list of tokens contains fewer than N tokens, the list becomes its own only subsequence. Each subsequence is added to the output set. Three GIFS were implemented, with N set to 2,4 and 6 respectively.

(11-17) ExtractNonSoundexPhoneticFeatures: Tokenizes the string and adds the *phonetic* encoding of each token to the output set. The phonetic functions used for implementing seven GIFs in total are *Caverphone1, Caverphone2, ColognePhonetic, DoubleMetaphone, MatchRatingApproachEncoder, Metaphone* and *NYSIIS*. The popular Soundex encoding is treated specially (see below). A library implementing all these encoding functions efficiently exists in an Apache open-source package[118] and was adapted for the system.

(18-27) ExtractSoundexPhoneticFeatures: Tokenizes the string and adds the Soundex encoding of each token to the output set. Along with the original Soundex encoding algorithm (implemented in the Apache open-source package), a refined version (also implemented in the package) as well as eight variations implemented in the open-source FEBRL package are also considered (Christen, 2008a). An example of a variation is to truncate each Soundex encoding to only the first four characters.

(28) ExtractAlphaNumeric: Extracts all tokens from the string such that a token contains at least one alphabet as well as a numerical digit (in addition to other optional characters). A rationale for this feature is subsequently provided.

The first ten GIFs are standard and have already been found to work well in previous work, including the original Relational Database setting in which they were first proposed (Bilenko, Kamath & Mooney, 2006). A brief

[118] org.apache.commons.codec.language

rationale for the features is provided below. Further details and accompanying examples are provided on the project website[119].

GIFs 1-2 are appropriate for strings that have high token overlap or for alphanumeric codes (e.g. in product databases) that tend to match exactly and have high correlation with duplicate classification. GIFs 3-4 are more appropriate for phone numbers, zip codes, street numbers, social security numbers, dates of birth and other numeric quantities that commonly occur in databases. GIFs 5-7 are empirically robust to many *data representation* issues; for example, GIF 5 would not distinguish between strings that spell 'Avenue' as *Avenue* or *Ave*. GIFs 8-10 generate token N-grams and are useful for detecting discriminative phrases in long descriptions.

Christen (2012a) evaluates the phonetic encodings, including Soundex and its variations, used by GIFs 11-27. The advantage of phonetic functions is that they are robust to spelling variations (especially in names) that the other GIFs cannot easily accommodate (e.g. *Kathryn* vs. *Catherine*). Using a range of phonetic encodings compensates for the quirks of a single encoding. Variations of phonetic encodings can further help to compensate for other sources of noise, such as missing prefixes and extreme misspellings. Since phonetic encodings are not trivial to compute, it makes computational sense to only consider variations of one particular phonetic encoding. The Soundex encoding was chosen for this purpose because it is well-studied and has an efficient, transparent implementation in packages such as FEBRL (Stephenson, 1980; Christen, 2008a).

Finally, the utility of GIF 28 is best realized in the cases where ID strings are often present and can be used to identify duplicate entities. Such strings tend to have both alphabets and digits and are relatively rare. Compared to more general token-based features (such as GIF 2), GIF 28 tends to be more discriminative, which helps the subsequent feature selection process.

Let h_i^1 denote a P-SIF in H_1 (and a similar analysis applies to a P-SIF h_j^2 in H_2), where h_i^1 is the P-SIF obtained by combining the GIF $g \in G$ and the attribute $a_i^1 \in A_1$. Applied to a record $r \in P_1$, let the output of the P-SIF h_i^1 be denoted as $h_{i(r)}^1$.

---

[119] https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/unsupervised-im

Given this notation, let a *property-specific feature* be defined as follows:

Definition 7.2 (**Property-specific feature**) Given two P-SIFs $h_i^1 \in H_1$ and $h_j^2 \in H_2$, a *property-specific feature* f_{ij} is defined as a binary function that takes as input a record pair (r, s) and returns 1 *iff* $h_{i(r)}^1 \cap h_{j(s)}^2$ is non-empty, and returns 0 otherwise.

Input: Sets $D$and N of duplicates and non-duplicates respectively

Property Alignment Q

Set G of General Indexing Functions (GIFs)

Output: Sets D_f and N_f of duplicates and non-duplicates feature-vectors respectively

---

Steps:

5. Initialize empty feature-vectors sets D_f and N_f
6. **if** Q is empty **then**

$Q \coloneqq A_1 \times A_2$

7. **end if**
8. **for all** record pairs $(r, s) \in D$ **do**

Initialize f to a $|Q||G|$ 0-vector

for all alignments $(a_i^1, a_j^2) \in Q$ **do**

for all GIFs $g \in G$ **do**

Let h_i^1 and h_j^2 be the P-SIFs obtained by

combining g with a_i^1 and a_j^2 respectively

Let f_{ij} denote the property-specific feature

formed from h_i^1 and h_j^2 respectively

if $f_{ij}\big((r, s)\big) = 1$ **then**

$f[|G|i + j] \coloneqq 1$

end if

---

end for

end for

9. **end for**
10. Repeat steps 4-5 by iterating over N, and populate N_f
11. Output D_f and N_f

Algorithm 7.1: *An algorithm for generating feature-vectors sets from training sets.*

Given an alignment set Q, each tuple pair in the training set can be converted to a feature vector with $|G||Q|$ binary elements, with each element corresponding to a single invocation of a property-specific feature f_{ij} on the tuple pair, where (a_i^1, a_j^2) is an element in Q. The pseudocode is provided in Algorithm 7.1.

Note that the dimensionality of each feature vector is directly proportional to Q. In the event that Q is unavailable, the only recourse (a 'fallback' option) for the system is to consider the exhaustive set $A_1 \times A_2$ (line 2). This demonstrates why having a compact, high-recall property alignment set Q is important, since both the quality and size of the resulting feature space depends on the quality of, and number of alignments in, Q.

Given a training set with n duplicates and non-duplicates, the feature generator outputs two feature sets with n binary vectors each. Assuming that the run-time of each GIF $g \in G$ can be bounded above by $O(c)$, the time taken by Algorithm 7.1 is $O(cn|G||Q|)$.

7.1.2 Learning Procedures

The two sets of feature vectors are now input to two independent training procedures, which respectively learn a blocking scheme for the blocking step, and an SVM classifier, which serves as a probabilistic link specification function for the similarity step.

Blocking Scheme Learner

In the following discussion, let ϱ denote the set of precisely those property-specific features that have the value 1 in at least one feature-vector in the set $D_f \cup N_f$. In other words, ϱ contains property-specific features that cover at least one feature vector in the training set. An obvious upper bound on $|\varrho|$ is $|G||Q|$, since each feature vector has at most $|G||Q|$ elements. In practice, the diversity of the twenty-eight GIFs in Section 7.1.1 results in $|\varrho|$ being less than $|G||Q|$. Interpreting each of the features in ϱ as a Boolean[120] variable, a *property-specific blocking scheme* in *Disjunctive Normal Form* (DNF) can be defined as follows:

Definition 7.3 (**Property-specific Disjunctive Normal Form blocking scheme**) Given a set ϱ of property-specific features, let a *property-specific Disjunctive Normal Form blocking scheme B* be defined as a disjunction of terms, where each term is a conjunction of features from ϱ.

As described in Chapter 2, this *class* of blocking schemes (henceforth, simply referred to as *DNF blocking schemes*) was first proposed for structurally homogeneous Relational Databases (RDBs) and found to deliver excellent empirical performance (Bilenko et al., 2006; Michelson & Knoblock, 2006). To the best of our knowledge, our work was the first to adapt this class of blocking schemes to heterogeneous RDF data (Kejriwal & Miranker, 2015a; 2015c). The class of DNF blocking schemes devised for RDBs is a special case of the class of property-specific DNF blocking schemes defined in Definition 7.3. Note that a DNF blocking scheme is a positive formula, since a term cannot contain negated features from ϱ.

Let the DNF blocking scheme be denoted as a k-DNF blocking scheme if each term is constrained to contain at most k features. Let a 1-DNF blocking scheme be denoted as a *disjunctive blocking scheme*, since it is a single clause. In order to learn a DNF blocking scheme, k must be specified as a parameter. The DNF blocking scheme is said to cover a record pair (or its equivalent feature vector representation) if it evaluates to $True$ for that pair. Ideally, the learned scheme should cover as many of the duplicates as possible, while minimizing coverage of the non-duplicates.

---

[120] With the 1 value interpreted as *True* and 0 as False.

The problem formulation described above is similar to that of the classic *Set Covering* (SC) problem (Chvatal, 1979). This connection (between DNF blocking scheme learning and SC) was first showed by Bilenko et al. (2006), when the DNF blocking scheme learning problem for structurally homogeneous RDBs was reduced to Red-Blue SC (Peleg, 2007). Although the problem discussed herein is more general (since two property tables may be structurally heterogeneous), a similar reduction applies.

Input: Sets D_f and N_f of duplicates and non-duplicates feature-vectors respectively

Set ϱ of property-specific features

Term parameter k

Set cover threshold parameter κ

Output: Property-specific k-DNF blocking scheme B

Steps:

1. Supplement set ϱ to get set ϱ_k (Equation 7.1)
2. Construct $M_D =< X, \varrho_X >$, where X is a feature vector in D_f, and $\varrho_X \subseteq \varrho_k$ contains the elements in ϱ_k covering X
3. Repeat Step 2 to build M_N for feature vectors in N_f
4. Reverse M_D and M_N to get M'_D and M'_N respectively
5. **for all** $X \in keyset(M'_D)$ **do**

   Score X by using formula $\frac{|M'_D(X)|}{|D_f|} - \frac{|M'_N(X)|}{|N_f|}$

   Remove X if $score(X) < \kappa$
6. **end for**
7. Perform *Weighted Set Covering* on keys in M'_D using Chvatal's heuristic (Chvatal, 1979), with weights set to the *negation* of the scores calculated above
8. $B :=$ *disjunction* of chosen keys
9. Output B

Algorithm 7.2: *An algorithm for learning a property-specific k-DNF blocking scheme.*

Unfortunately, SC is NP-complete[121] (Carr, Doddi, Konjevod & Marathe, 2000). Using an additional threshold parameter κ, an SC approximation algorithm from the literature can be leveraged to learn a k-DNF blocking scheme. The pseudocode is given in Algorithm 7.2.

In line 1, Algorithm 7.2 uses k to supplement the set ϱ and obtain the set ϱ_k. If an i-term is defined as a term that is constructed by forming a conjunction of exactly i property-specific features from ϱ, and S_i as the set of all possible i-terms, ϱ_k is given by the expression:

$$\varrho_k = \bigcup_{i=1}^{k} S_i \qquad (7.1)$$

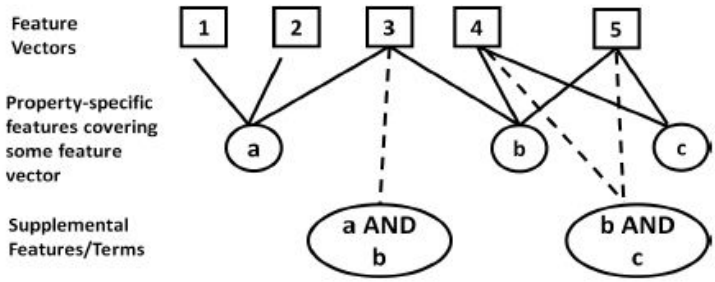


Figure 7.1: *Step 1 of Algorithm 7.2, using a pruning strategy.*

Note that S_1 is simply the alignment set ϱ. In practice, not all terms will be used by Algorithm 7.2, making an exhaustive construction of ϱ_k unnecessary. Instead, a *pruning strategy* includes only those terms in ϱ_k that cover some feature-vector in $D_f \cup N_f$, since only those terms will actually be used (lines 2-3). Figure 7.1 illustrates the strategy. Assuming that $k = 2$, there are three possible 2-terms $a\ AND\ b$, $a\ AND\ c$ and $b\ AND\ c$, but only the two shown in Figure 7.1 would get added to the supplemented set ϱ_2. A consequence is that if $k = 3$, then $\varrho_3 = \varrho_2$. This is because there is no feature-vector that is simultaneously covered by a term with three features from ϱ.

---

[121] Many variants are also known to be NP-Complete (Carr et al., 2000).

Using the supplemented set ϱ_k, lines 2-3 construct *multimaps*[122] by assigning each feature-vector in D_f a key in M_D, and with the elements in ϱ_k covering that feature-vector comprising its value set. M_D is then reversed to yield M'_D. M'_N is similarly constructed. Figure 7.2 demonstrates the key-value reversal procedure, assuming D_f contains feature-vectors 1-5, covered as shown in Figure 7.1. The time complexity of building M'_D and M'_N is $O(|\varrho_k|(|D_f|+|N_f|))$.

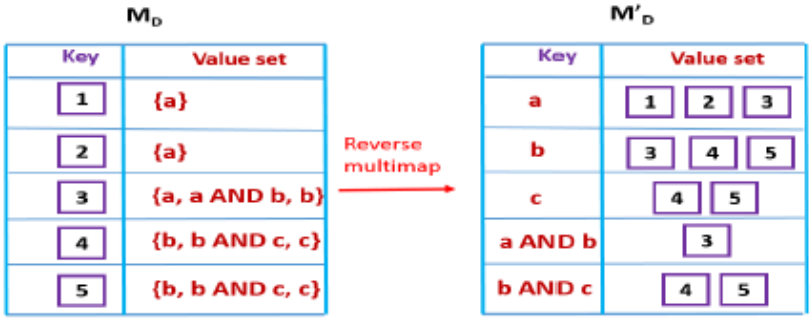


Figure 7.2: *Construction of multimaps and reversed multimaps.*

In lines 5-6, each key is first scored by calculating the difference between the fractions of covered duplicates and non-duplicates. A threshold parameter, κ, is used to remove the keys that have low scores. κ is designed to improve quality by removing those features from ϱ_k that either cover too few duplicates, or cover too many non-duplicates (or both). The range of κ is [-1,1]. A value close to 1.0 would indicate that the user is confident about low noise-levels in inputs D, N and the property alignment set Q, since high κ implies the existence of elements in ϱ_k that cover many positives and few negatives. Since many keys in M'_D are removed by high κ, this also leads to computational savings. However, setting κ too high (perhaps because of misguided user confidence) could lead to excessive purging of M'_D, and subsequent failure of Algorithm 7.2. A low κ is safer, but may result in slower run-times.

---

[122] A multimap is a generalized version of a map, whereby a key can reference multiple values (i.e. a *value set*).

Similar to the parameter $thresh$ in Algorithm 5.1, κ can also be set in self-tuning mode, with a low (but not too low) default value of 0.2. If Algorithm 7.2 fails with a given value of κ, it is indicative of κ being too high. κ is then decreased by a small number (e.g. 0.05) till Algorithm 7.2 successfully returns a blocking scheme. In one of the conducted experiments (Section 7.2), the self-tuning methodology is found to lead to seamless execution of Algorithm 7.2.

In line 7, Weighted Set Covering (W-SC) is performed using Chvatal's approximation algorithm (Chvatal, 1979), with each key in M'_D acting as a set, and the record pairs covered by all keys as elements of the universe set U.

For example, assuming that all features in ϱ_k in the keyset of M'_D in Figure 7.2 have scores above κ, $U = \{1,2,3,4,5\}$. Note that *only* M'_D is pruned (using κ) and also, W-SC is performed only on M'_D. M'_N only aids in the score calculation (and subsequent pruning process) in line 5 and may be safely purged from memory before line 7.

W-SC needs to find a subset of the M'_D keyset that covers all of U and with *minimum* total weight. For this reason, the weight of each set is the negative of its calculated score. Given that sets chosen by W-SC actually represent features in ϱ_k, their disjunction is the desired k-DNF blocking scheme (line 8).

As stated before, Set Covering (and also Weighted Set Covering) is known to be an NP-Complete problem (Carr et al., 2000). Under plausible[123] complexity assumptions, Chvatal's algorithm is currently the best-known polynomial-time approximation for W-SC (Raz & Safra, 1997). Since Algorithm 7.2 directly invokes Chvatal's algorithm as a subroutine, it is conferred with similar theoretical guarantees.

In practice, setting k to 1 has been shown to be a viable option even on noisy test cases (Kejriwal & Miranker, 2015a). This is an important computational benefit since $|\varrho_k|$ is exponential in k in the worst-case.

[123] $P \subseteq NP$.

Training the Classifier

The feature-vectors sets D_f and N_f are also used for training a supervised classifier that serves as a probabilistic link specification function in the classification step. Note that, although the sets D_f and N_f are re-used for training the classifier, it is theoretically possible to devise a new feature space for this step. For example, a new floating-point valued feature $Levenstein$ could be added, yielding $|Q|$ new features[124] for each record pair in D and N. Indeed, a similar supplemental step was performed in Algorithm 7.2 for learning k-DNF blocking schemes, when $k > 1$. In this chapter, the binary feature-vectors output by Algorithm 7.1 are re-used for training the classifier.

The primary reason for re-using the original feature-vectors is computational. Each additional feature computation incurs cost $|\varrho|$ for each tuple pair, and would additionally increase the run-time for training a classifier. Re-using the feature vectors further implies that the feature generator only needs to be run once and that in a shared-memory architecture, both the DNF blocking scheme learner and the classifier trainer can access the same feature-vectors, resulting in savings in both time and space.

As for the specific classifier trained on the feature-vectors, the noise in the training sets, the sparsity of non-zero elements in individual feature-vectors and the potential curse-of-dimensionality issue that would arise if ϱ is large compared to either D or N, suggest the use of a kernel-based maximum margin classifier such as a Support Vector Machine or SVM (Joachims, 1999). Previous studies in the instance matching community have validated this empirically by showing that *supervised* SVM-based classifiers such as FEBRL and MARLIN achieve state-of-the-art performance on standard benchmarks (Bilenko & Mooney, 2003; Christen, 2008a; Köpcke, Thor & Rahm, 2010).

In the learner, the training sets D_f and N_f are used to train an SVM with a Radial Basis Function (RBF) kernel (Chang & Lin, 2011). A polynomial kernel is not adapted because it requires the tuning of more hyper-parameters, which is problematic given that the system only has a limited, noisy number of training samples available to it. It is also known that a linear

---

[124] One $Levenstein$ calculation for each aligned pair in Q.

kernel (and for certain parameters, a sigmoid kernel) is a special case of the RBF kernel, making it a reasonable choice (Hsu, Chang & Lin, 2003).

Finally, while more sophisticated machine learning classifiers can always be used in this module instead of a kernel-based SVM, a user should be aware of their typically higher training times. For example, multilayer perceptron classifiers, which were recently shown to deliver slightly better performance on average than SVMs, were simultaneously found to be almost an order of magnitude slower on several test cases (Soru & Ngomo, 2014).

Blocking Method and Similarity Step

Given a blocking key (Definition 2.2), there has been extensive research on how best to use the key in a *blocking method* (Christen, 2012b), including a variety of methods specifically designed for heterogeneous information spaces such as the Web of Linked Data (Papadakis, Ioannou, Palpanas, Niederée & Nejdl, 2013). A promising blocking method is *block purging*. The method works by using a given blocking key on each entity to generate blocking key values (BKV). Entities are clustered into (possibly overlapping) blocks, with each block uniquely identified by a BKV. To control data skew, block purging eliminates all blocks that generate more pairs than a threshold, designated in this dissertation as $maxPairs$. An algorithm was proposed to calculate $maxPairs$ automatically, but required a two-pass approach over the generated blocks (Papadakis et al., 2013). In preliminary experiments, manually determining the $maxPairs$ threshold was found to lead to significantly superior results over automatically determining $maxPairs$. Tuning $maxPairs$ was also not found to be cumbersome; hence, this approach is adopted in the current implementation. Practitioners can customize this step per their needs without disrupting the remainder of the system in Figure 1.5.

Finally, the classifier trained in the previous step is used on the candidate set of instance pairs to output links *probabilistically* in the similarity step. The score output by the SVM classifier is interpreted subjectively as the classifier's belief in the instance pair being a duplicate. Note that the highest-scoring pairs output by the classifier can be used to repeat parts of the learning process in the hope of achieving better performance (typically through higher

recall). This self-training option (denoted as an *iterative run*) is described and evaluated further in Section 7.2.3

7.2 Evaluations

Unlike evaluations in Chapters 5 and 6, this chapter contains three sets of evaluations. The first evaluation concerns blocking, while the second and third concern classification. Setup, results and discussion of each of these evaluations will be described in individual sub-sections. The test suite used in all evaluations is the domain-independent suite described in Section 5.3, and summarized in Table 5.1.

7.2.1 Blocking

The goal of this experiment is to evaluate the heterogeneous DNF blocking scheme learner (DNF-BSL).

Setup

First, a preliminary experiment is used to determine if advanced blocking techniques are even warranted in real-world cases, or if a simple token-based clustering approach suffices, by running the classic *Canopies* algorithm on each of the ten test cases (McCallum, Nigam & Ungar, 2000). *Canopies* was earlier described in Chapter 2, and also used in Chapter 4. *Canopies* uses a threshold[125], and an inexpensive distance metric[126].

In the main experiment, Algorithm 7.2 is evaluated against the *trigrams-based Attribute Clustering* (AC) baseline (Papadakis et al., 2013). AC is considered to be a state-of-the-art unsupervised blocking approach for schema-free data represented only as a set of attribute-value pairs. The method extracts trigrams from each attribute value in the dataset, and then clusters attributes by computing the overlap between trigram value-sets.

---

[125] Technically, it uses *two* thresholds, but assigning them a common value was found to yield the best empirical results in record linkage applications (Baxter, Christen & Churches, 2003).

[126] Typically cosine similarity on TFIDF vectors (Cohen, 2000; Baxter et al., 2003).

The DNF-BSL in Algorithm 7.2 required setting two parameters k and κ. Since the algorithm is exponential in k and previous experimental results have not found large differences between the $k = 1$ and $k = 2$ settings (Kejriwal & Miranker, 2015a), k is set to 1. Similar to the parameter $thresh$ in Chapter 5 evaluations (Section 5.3), κ was set to a default value of 0.2 and in self-tuning mode with a decrement of 0.05. In the majority of the cases, self-tuning was not invoked. In two cases (*Eprints-Rexa* and *Libraries*), the self-tuning was invoked and the value of κ at which the algorithm succeeded was 0.01. Either way, the algorithm was able to successfully output a DNF blocking scheme. The blocking results are evaluated using the PC, RR and F-Measure[127] metrics described in Chapter 2.

Results and Discussion

Table 7.1 shows the results of the preliminary experiment. While *Canopies* achieves over 90% F-Measure on four test cases, its general performance exhibits much deviation. On *Parks*, the method fails completely, achieving 100% RR and 0% PC[128]. Additionally, the algorithm was found to run quite slowly on large datasets, and the threshold parameter had to be tuned separately for each run. These results show that the performance of *Canopies* is unpredictable for schema-free RDF data, and for domain-independent applications[129]. Its average achieved F-Measure (67.87%) is also quite low compared to state-of-the-art techniques, as the following experiment will demonstrate.

Table 7.2 shows the results for the main experiment, where the proposed DNF-BSL is evaluated against the *Attribute Clustering* baseline. Both methods perform quite well generally, although the proposed system outperforms the baseline on six of the test cases, and by 1.5% F-Measure on average. An important difference between the systems' performances is that the proposed method tends to favor the RR metric over PC. While the F-Measure treats PC and RR equally, blocking practitioners have argued that

---

[127] PC stands for *Pairs Completeness*, RR for *Reduction Ratio* and F-Measure is the harmonic mean of PC and RR. These metrics were also used for type alignment evaluations in Chapter 4.

[128] 100% PC and 0% RR can also be achieved if the threshold parameter is set high enough. Even with exhaustive parameter sweeps, no other value sets were obtained except for these two extremes.

[129] As described in Chapter 5, *Parks* and *Libraries* contained POI (Point of Location) data. The smaller size of *Parks*, compared to *Libraries*, might account for performance degradation due to domain effects.

even small differences in RR can be consequential (Christen, 2012b). This is because, as argued at length in Chapter 2, RR is measured over the full set of entity pairs, which is a quadratic function and can number in the millions even for moderately sized datasets. In contrast, PC is a linear function of the matching entity pairs in the files, which are typically quite small in number. By this argument, low values of RR can lead to the overall instance matching task becoming intractable in a practical implementation. Table 7.2 shows that the proposed DNF-BSL only achieves RR below 95% on two datasets (and never below 90%), while the baseline can be more unpredictable (less than 95% RR on four datasets).

Test case	PC	RR	F-Measure
Persons 1	100	97.96	98.97
Persons 2	99.75	98.58	99.16
Restaurants	75.28	99.39	85.67
Eprints-Rexa	6.32	99.99	11.89
IM-Similarity	26.90	89.99	41.42
IIMB-059	100	96.98	98.47
IIMB-062	51.89	98.09	67.87
Libraries	87.74	99.99	93.47
Parks	0	100	0
Video Game	69.24	99.92	81.80
Average	*61.71*	*98.09*	*67.87*

Table 7.1: *Results of the preliminary blocking experiment. PC stands for Pairs Completeness, and RR for Reduction Ratio.* All values are percentages.

An advantage of *Attribute Clustering* is that, unlike the heterogeneous DNF-BSL, it does *not* require training samples or a space of indexing functions (Papadakis et al., 2013). Its expressiveness over other non-adaptive token-based techniques such as *Canopies* arises because it extracts trigram representations from each of the tokens. The results in Table 7.2 may thus *also* be interpreted as favoring *Attribute Clustering* in a MapReduce-based implementation (Chapter 8). In serial implementations,

the adaptiveness of the DNF-BSL gives the method a small performance advantage.

	Algorithm 7.2			Attribute Clustering (AC)		
Name	**PC**	**RR**	**FM**	**PC**	**RR**	**FM**
Persons 1	100	99.75	**99.88**	100	98.86	99.43
Persons 2	99.00	99.79	**99.39**	99.75	99.02	99.38
Restaurants	100	99.73	**99.87**	100	95.57	99.79
Eprints-Rexa	98.16	99.28	98.72	99.60	99.37	**99.48**
IM-Similarity	100	98.14	**99.06**	100	62.79	77.14
IIMB-059	99.76	93.35	**96.45**	97.33	73.09	83.49
IIMB-062	47.73	98.11	64.22	77.27	90.80	**83.49**
Libraries	97.96	99.99	98.96	99.99	99.87	**99.93**
Parks	95.96	94.41	**95.18**	99.07	88.27	93.36
Video Game	98.73	99.96	99.34	99.72	99.85	**99.79**
Average	*93.73*	***98.25***	***95.11***	***97.27***	*91.15*	*93.53*

Table 7.2: *Results of the main blocking experiment.* PC stands for Pairs Completeness, RR for Reduction Ratio, and FM for F-Measure. All values are percentages.

7.2.2 Similarity (*non-iterative run*)

This experiment evaluates three Support Vector Machines (SVMs)[130] against each other, with the goal of determining how the degree of supervision (the number of samples an SVM is trained on) and noise (incorrect labeling of training samples) affect overall classification performance. The SVMs are also evaluated against an unsupervised baseline method that combines Locality Sensitive Hashing (LSH) and Expectation Maximization (EM) (Winkler, 2002; Datar, Immorlica, Indyk & Mirrokni, 2004), which have been successfully applied to both instance and ontology matching in the recent past (Kim & Lee, 2010; Duan et al., 2012).

130 All SVMs in this paper use RBF (Radial Basis Function) kernels, for which an efficient implementation may be found in the *LibSVM* library (Chang & Lin, 2011).

Setup

As a first step, the candidate set of pairs for the classification step is generated using the blocking approach that had the higher F-Measure in the previous experiment (for a particular test case). An SVM is trained using the 500 duplicates and non-duplicates generated by the proposed TSG. Let this SVM be denoted as *Unsupervised*[131], since the training sets are automatically generated. Two supervised SVMs are trained on 10% and 50% of the ground-truth. These perfectly labeled samples are used for both training and cross-validation, with the rest of the ground-truth not seen by the classifier till actual testing time to avoid bias. The SVM performances are compared using precision-recall graphs. The supervised SVMs are expected to illustrate the limits of the unsupervised system by showing how the numbers (and levels of noise) in the samples affect the training of the SVMs in the described feature space.

The *alternate unsupervised baseline* is set up as an Expectation Maximization (EM) clustering procedure as follows. First, to improve robustness and reduce processing times, the original feature space is reduced by computing, for each feature vector in the candidate set, seven hashes[132] using an open-source Locality Sensitive Hashing (LSH) package[133]. Thus, a new feature-vectors file was produced, where each vector is represented by seven real numbers. Appropriately formatted, this file was then input to the EM algorithm implemented in the Weka[134] machine learning package. To maximize performance, all vectors were probabilistically clustered into two clusters. The final step was to map the two clusters to classes (that is, duplicates or non-duplicates). This was achieved by using a simple but effective heuristic: the larger cluster was considered to map to the non-duplicates class. This mapping was found to maximize this baseline's metrics. As the alternate baseline is independent of the noisy training set, it tests how well simple heuristics or traditional techniques (such as EM) fare on roughly schema-free instance matching relevant to the Linked Data ecosystem.

---

131 The label is a misnomer and must be interpreted with care, since even an 'unsupervised' SVM has to be trained. Thus, the label refers to the way the training set was acquired, not the classifier itself.

132 *CosineHash*, three variants of *EuclideanHash*, and three variants of *CityBlockHash*.

133 https://github.com/JorenSix/TarsosLSH

134 http://www.cs.waikato.ac.nz/ml/weka/

Figure 7.3 shows the results for the cases where the highest achieved F-Measure (for any of the systems) was greater than 60%. On four test cases (*Libraries*, *Restaurants* and both *Persons* cases), there is no statistically significant difference between either of the supervised systems. On all these datasets, the SVM is able to adapt to the unseen data without much supervision. The *IIMB-059* illustrates that this is not necessarily the case for every test case. The overall results also show that the SVM is able to adapt even when instances from multiple types are present. In Figure 7.3, only the *Libraries* and *IM-Similarity* test cases have instances from a single type.

Results and Discussion

Although the unsupervised SVM does not generally perform as well as the supervised SVMs, it is still competitive on three of the test cases (*Persons 1*, *Restaurants* and *Libraries*) over a particular *range* of precision and recall. On *IIMB-059*, *Unsupervised* outperforms *Supervised 10%* in terms of the highest F-Measure. An iterative approach is explored in the next experiment to improve the unsupervised SVM performance on some of these test cases.

Figure 7.4 illustrates the four test cases for which no SVM manages to achieve a high F-Measure. In all cases, the unsupervised SVM performs at least as well as (and on *Video Game* and *Eprints-Rexa*, outperforms) one of the supervised SVMs. A closer look at the results showed that all the SVMs were returning many false positives for these cases. An obvious hypothesis is that the SVMs were overfitting the data on these four test cases.

To study this hypothesis, it is instructive to compare the graphs in Figures 7.3 and 7.4 with the results of training set generation (TSG) in Section 5.3. On some cases, particularly *Parks*, *Video Game* and even *Eprints-Rexa*, the TSG outperforms even the supervised SVMs by a considerable margin. This is most apparent in the *Video Game* case, where the *Supervised 50%* SVM performs the worst. Automatically determining when to choose the TSG over the adaptive classifier is an important issue for future work, and directly related to the recent research interest in self-configuring systems (Han, Lee & Crespi, 2014). At least one other instance matching study has also made a

similar finding; namely, that an adaptive classifier is not a silver bullet[135] for every instance matching test case (Soru & Ngomo, 2014).

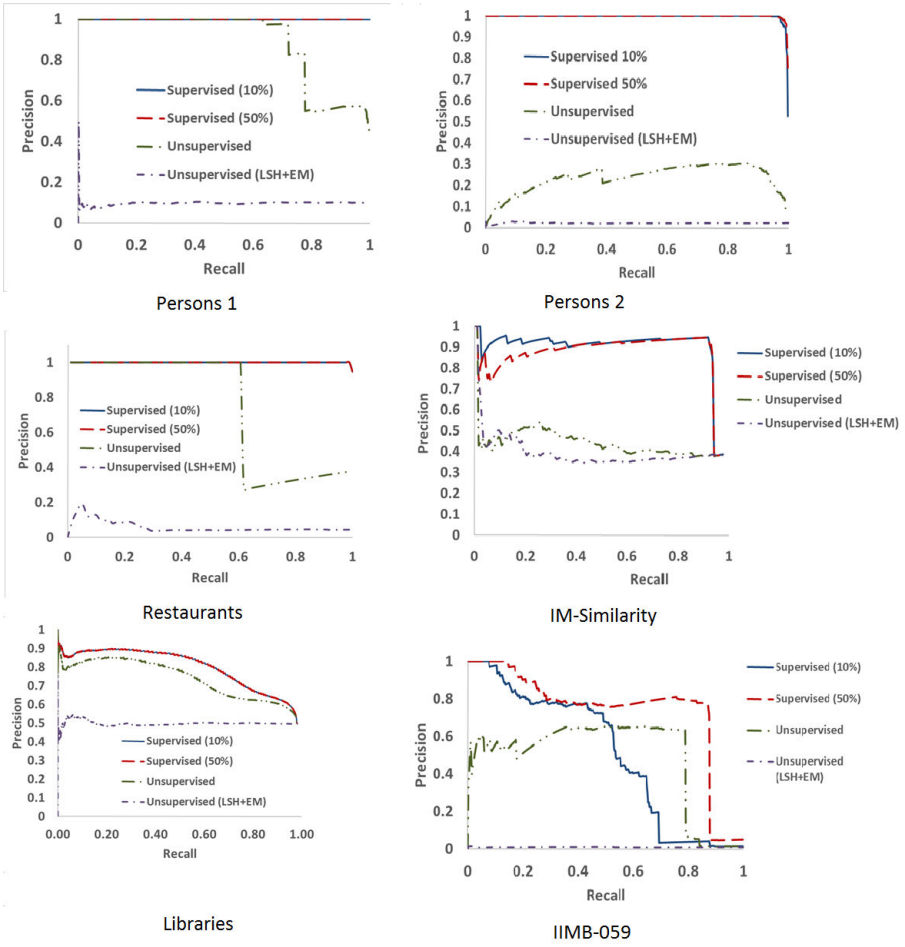


Figure 7.3: *SVM results for the test cases where highest-achieved F-Measure was over 60%.*

135 The study showed that, on a third of the employed test cases, all tested supervised classifiers (including multilayer perceptron, SVM and logistic regression classifiers) achieved less than 50% F-Measure (see page 3 of Soru & Ngomo (2014)).

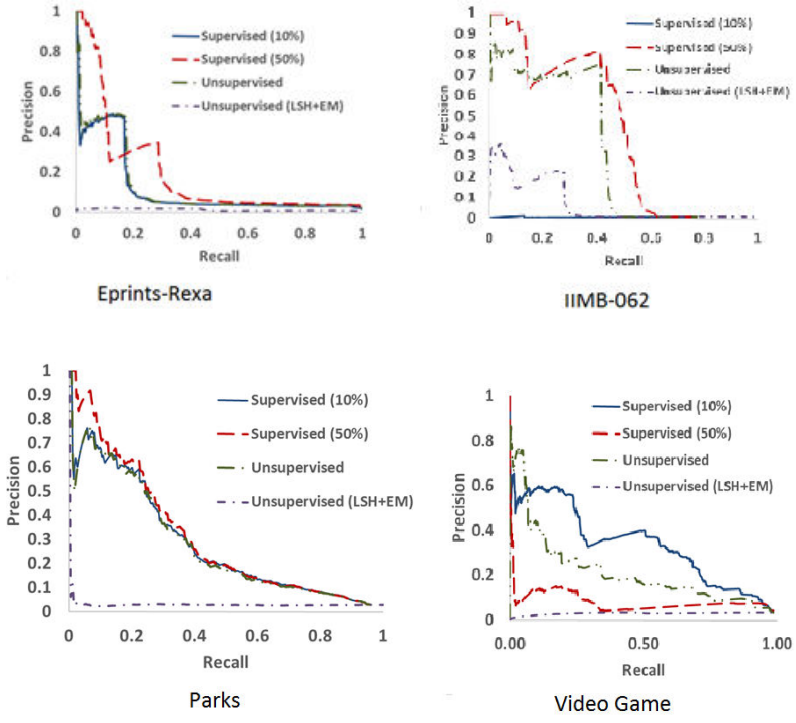


Figure 7.4: *SVM results for the test cases where highest-achieved F-Measure was below 60%.*

Finally, the alternate unsupervised baseline is outperformed by all methods on the majority of the cases. There are only two exceptions: low recall levels in the *IM-Similarity* test case (where it is briefly competitive with all methods), and the *IIMB-062* test case, where it outperforms the *Supervised 10%* SVM. The results show, quite unambiguously, that simple heuristics-based, distance-based and clustering-based techniques are not adequate by themselves, or even in simple combinations, for noisy schema-free RDF data. This conclusion is similar to the one in the previous experiment on blocking, where the performance of *Canopies* was found to be inadequate. This may explain why the instance matching literature covering these methods makes strong assumptions about the underlying datasets, including existence of structure and meta-data (e.g. ontologies) (Winkler, 1993; Kim & Lee, 2010; Duan et al., 2012). Adapting these traditional methods so that they perform well on roughly schema-free Linked Data is left for future work.

7.2.3 Similarity (*iterative run*)

This experiment explores an iterative approach for improving the performance of the unsupervised SVM and the alternate approach in the previous experiment. At the end of Section 7.1, the possibility was mentioned that the highest-confidence samples output by the classifier can be used to re-learn certain functions. In principle, the samples can be used to re-learn both the blocking scheme and the SVM classifier. A disadvantage of this naïve re-learning is that it does not take into account (1) the expense of the blocking method, and (2) the high performance achieved by the DNF-BSL using just the generated training set. A better cost-gain tradeoff is achieved by *only* re-training the SVM classifier. This section evaluated this tradeoff.

Setup

The previous experiment showed that, in at least five test cases (see the graphs for *Persons 1*, *Persons 2*, *Restaurants*, *IM-Similarity* and *Libraries* in Figure 7.3), the SVM trained on 10% of the ground-truth was able to achieve better performance than the unsupervised SVM, despite being trained on fewer samples (but without noise). It would thus seem that in these cases, the noise (more than the size of the training set) dictates SVM performance. In this experiment, this hypothesis is explored and it is shown that the effects of this noise can be accounted for, while still keeping the system unsupervised, if an iterative approach is adopted. Namely, the SVM re-trains itself on a small set of top-scoring duplicates initially output by it, after which the classification step is re-run[136]. Specifically, the 50 most confident samples output by the unsupervised SVM are first permuted to obtain 50 non-duplicates (line 9 of Algorithm 5.1), which are together used to re-train the SVM. A much smaller (re-)training set than that[137] used in the first pass is used, to skew the quality-representation tradeoff in favor of quality. The expectation is that, in the cases where *Supervised 10%* outperformed *Unsupervised* in the previous experiment, the gap between the two systems will significantly narrow, if it is not eliminated altogether. On the datasets where representation mattered more

136 That is, the same candidate set from the previous experiment (for each of the test cases) is re-classified.

137 This number was 500, as described in the evaluations in Chapter 6.

than quality, the performance is expected to decline, an example being IIMB-062 (Figure 7.4).

In order to test the post-iteration performance of the alternate baseline, the $ClassificationViaClustering$ facility in the latest version of Weka is used. This facility allows a practitioner to use labeled instances to inform the clustering of unseen data. Since only 100 labeled samples (50 duplicates and 50 non-duplicates[138]) are being used, the clustering is not expected to be radically different. Other details on how the clustering was conducted and instances were classified can be found in the *Setup* sub-section of Section 7.2.2.

Results and Discussion

Figures 7.5 and 7.6 compare the two unsupervised runs (before and after iteration). In six of the ten test cases (Figure 7.5), the highest achieved F-Measure improved after iteration. After iteration, the performance difference between the unsupervised system and both supervised systems (from the previous experiment) on *Persons 1* narrows so that there is no statistically significant difference between the three systems (the post-iteration SVM and the supervised SVMs from the previous experiment). Near-perfect results are observed, showing that (on *Persons 1*) training set *noise* had a more significant impact on SVM performance than training set *size*.

The opposite is true for both the IIMB datasets, and more surprisingly, *Persons 2*. The results on *Persons 2* were surprising because, as stated earlier, *Supervised 10%* achieved excellent performance on it. In further experiments, the SVM was re-trained on larger sample numbers (ranging from 10-400), to test if the number of samples *predominantly* affects performance. Even with this tuning, the post-iteration SVM never outperformed the pre-iteration SVM in the conducted experiments. It is conceivable the primary cause behind the post-iteration SVM's performance decline is the low precision of the pre-iteration SVM on *Persons 2*. The SVM is not able to compensate for the high levels of noise even in the most confident samples retrieved by the pre-iteration SVM on *Persons 2*, regardless of training set size. This shows that

---

138 There is a small caveat: for the alternate baseline, the 50 duplicates were not permuted to yield 50 non-duplicates. Instead, the 50 *lowest-ranked* samples were taken from the probabilistically scored candidate set in the experiment in Section 7.2.2.

the iterative procedure is not always successful; in particular, it can lead to a decline in overall performance if the original first-pass SVM outputs low-quality results to begin with.

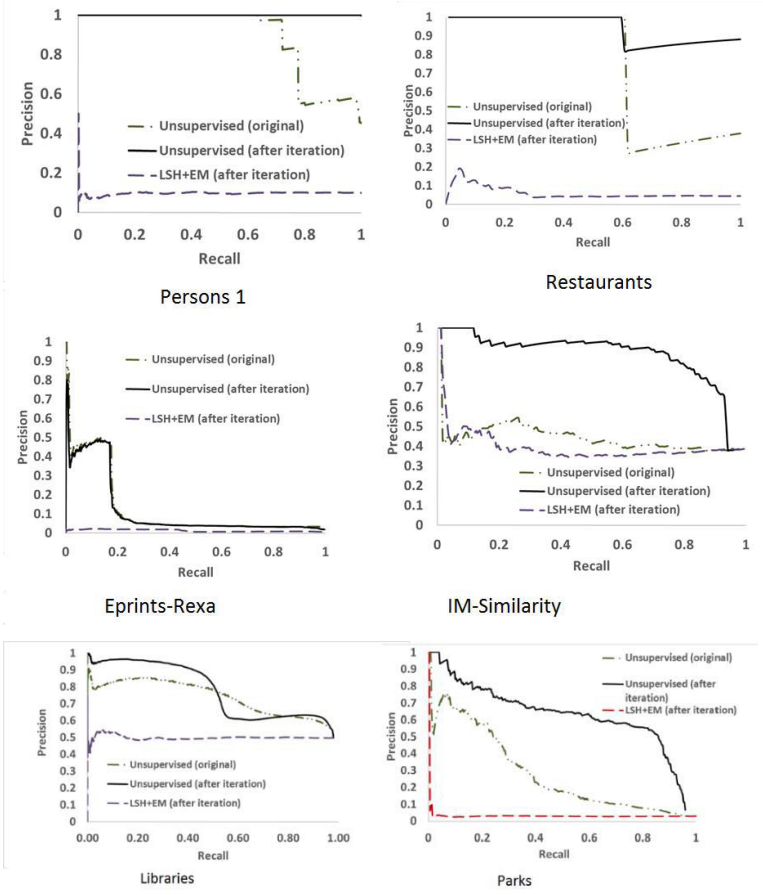


Figure 7.5: *Pre-iteration SVM, post-iteration SVM and alternate baseline results for the six cases where an improvement in highest-measured F-Measure performance for post-iteration SVM was observed.*

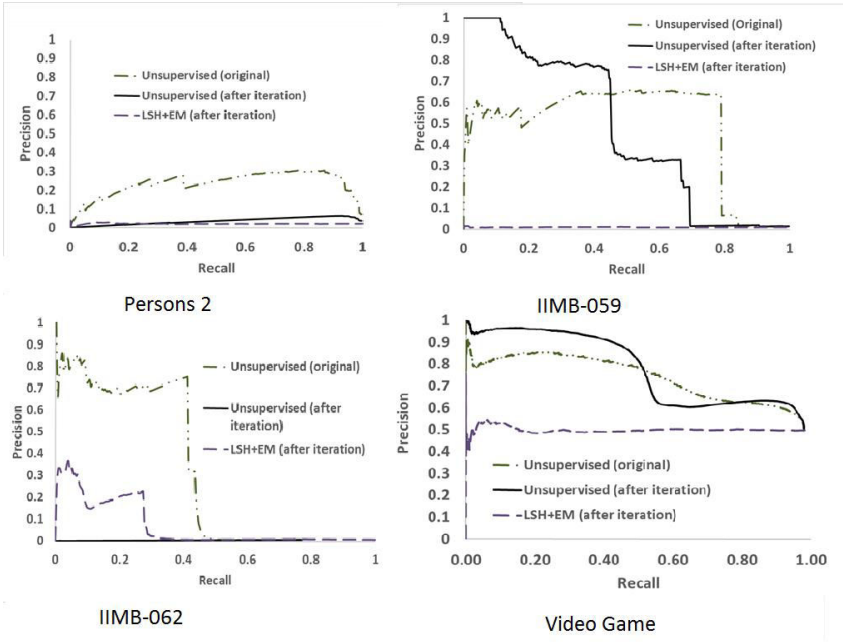


Figure 7.6: *Pre-iteration SVM, post-iteration SVM and alternate baseline results for the four cases where an improvement in highest-measured F-Measure performance for post-iteration SVM was not observed.*

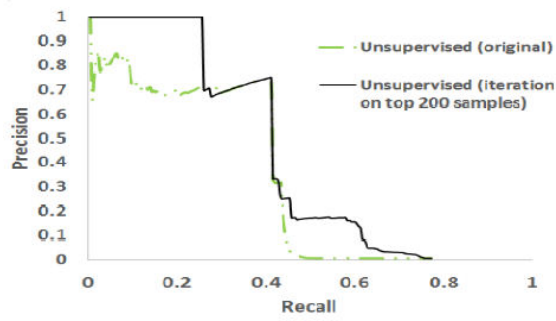


Figure 7.7: *Pre-iteration and post-iteration SVM results for IIMB-062 when re-training on the top 200 (rather than the top 50) samples.* Improvement of the latter over the former is statistically significant.

On the IIMB datasets, the post-iteration curves resemble those of *Supervised 10%* from the previous experiment. On IIMB-062 in particular, the curve 'collapses', showing that the number of samples is the determining factor on performance. To test this claim, the SVM was re-trained on the top 200 samples instead of the top 50 samples. Figure 7.7 shows the results of this supplementary experiment, where the post-iteration SVM now outperforms the original SVM and also *Supervised 50%* at low recall levels (not shown in the figure).

On *IM-Similarity* and *Parks*, the improvements are quite drastic, with the post-iteration unsupervised system effectively outperforming both supervised systems from the previous run. On these test cases, the iteration achieves its maximum utility. On the other two cases, *Eprints-Rexa* and *Video Game*, iteration also improves performance but by a near-indistinguishable margin.

The iteration did not improve (or otherwise modify) the clustering (i.e. the alternate baseline) procedure at all; hence, the post-iteration and pre-iteration performance of the unsupervised LSH are coincidental. This can be attributed to the relative stability of EM, especially given the very small labeled set that was provided to the system. Furthermore, the distinguishing characteristics (that is, different feature values) of individual instances in the training set are neutralized by the LSH feature-reducing computations and provide less information to the EM procedure than is provided to the SVMs. The net result is that there is no improvement in alternate baseline performance. This finding also implies that the baseline may not be amenable to techniques such as *active learning* where a user is continuously trying to improve the system through incremental labeling (Settles, 2010).

It is also worthwhile considering the benefits of further iterations. In alternate work, not covered in this chapter, it was shown that the maximal benefits of such iterations tend to be realized in the first three iterations, with major gains in the first iteration itself (Kejriwal & Miranker, 2015b). At least for some test cases, single-run iterative procedures suggest an attractive, and efficient, methodology to bettering unsupervised instance matching performance.

Chapter 8: Scalability

This chapter covers the scalability of the described system using MapReduce (Dean & Ghemawat, 2008). First, the algorithms in Chapters 4-7 are summarized in terms of their inputs and outputs in Section 8.1 This input-output perspective provides clarity on why the scaling of the system is not expected to lead to major revisions in the presentation thus far. This is in contrast to the current state-of-the-art in the literature, where scalability and automation were largely considered at odds with each other (Chapter 3).

8.1 Summary of Algorithms

Component	**Input**	**Output**
Type Alignment (Algorithm 5.1)	Two (multi-type) RDF graphs G_1 and G_2	Type Alignment Θ
Training Set Generator (Algorithm 6.1)	Two (single-type) property tables P_1 and P_2	Sets D and N of positive and negative training sets resp.
Property Alignment (Algorithm 7.1)	Sets D and N of positive and negative training sets resp.	Property Alignment Q
Other Learning Procedures (Algorithms 8.1, 8.2)	Sets D and N of positive and negative training sets resp., Property Alignment Q	Property-specific DNF blocking scheme B, machine learning classifier C
Blocking and Similarity	Property-specific DNF blocking scheme B, machine learning classifier C, Property Tables P_1 and P_2	*:sameAs* declarations (i.e. duplicates)

Table 8.1: *An input-output summary of selected algorithms described heretofore.*

Table 8.1 summarizes the main algorithms described in Chapters 4-7 from the perspective of primary inputs[139] and outputs. From this input-output perspective, algorithms fall within two categories. Algorithms in the first category take as input the *full* data, and are most amenable to parallelism. The type alignment algorithm, for example, needs to access all data in order to build the type documents, construct a type matrix and derive type alignment pairs. Similarly, the training set generator and the blocker and classifier need to access all the data to perform their computations[140].

Algorithms in the second category, mainly comprising the learning procedures and property alignment algorithm, do *not* need to access the full dataset. Instead, they accept as input the outputs of earlier algorithms (e.g. the training set generator). Because their inputs are of modest size, the algorithms can be exactly re-implemented in a parallel setting using a convenient technique outlined in the next section.

8.2 Motivation and Use-Cases

The challenges of scalable instance matching are different from those of applications that are *inherently large-scale* (e.g. high-fidelity physics simulations). Such applications tend to involve numerical and scientific data, and arguably fall within the auspices of high-performance computing, rather than Artificial Intelligence research (Schroeder & Gibson, 2010). In more traditional data management, *set similarity* joins operate on similar principles. Although such joins accept large-scale inputs, a set similarity function (e.g. $Jaccard$) and a threshold is assumed, and the output is necessarily expected to be both complete and consistent (Vernica, Carey & Li, 2010; Metwally & Faloutsos, 2012; Das Sharma, He & Chaudhuri, 2014).

The preceding chapters showed that a full instance matching pipeline is more complex, and includes a wide variety of algorithms. For example, Table 8.1 lists the algorithms developed specifically within this dissertation. A survey of instance matching literature that explicitly involves parallel or

---

139 Primary inputs are the principal data units on which these algorithms operate. For the sake of discussion, parameters (e.g. $thresh$ in Algorithm 5.1) are considered secondary inputs.

140 Although these algorithms only accept single-typed (or multiple *different-domain* typed) property tables as input within a given execution, all the data must still be processed, as the algorithm is executed for each aligned type pair. Like blocking, type alignment only serves as an efficiency constraint in this setting.

distributed processing by virtue of fulfilling the scalability requirement shows that a *large-scale instance matching application* involves *medium-scale inputs*[141]. Some of these efforts were briefly reviewed in Chapter 3.

To recap the discussion in Section 2.4.1, the unavoidable pairwise nature of instance matching, even with blocking, leads to large *intermediate* outputs (i.e. the generated candidate set). For example, given two files with 20,000 and 30,000 entities respectively, the cardinality of the *exhaustive set* (i.e. the set of all possible instance pairs) is 600 million. Even with a reduction ratio of 99%, currently only achievable by a state-of-the-art blocking algorithm (Chapter 7), over 6 million entity pair candidates have to be distributed across nodes and evaluated by a link specification function. Because of the *class imbalance* problem in real-world data, the number of duplicates is typically quite small (sub-linear in the number of entities). It is precisely because of duplicates sparsity that blocking methods are able to reduce the exhaustive space without causing performance degradation (Christen, 2012b). Regardless, even the best currently known blocking methods go only so far, especially when the data is noisy or the distribution of duplicates is non-uniform (Christen, 2012b).

Another motivation stems from an analysis of current domain-independent datasets on Linked Open Data, for which the system was primarily designed. Some of the largest datasets on Linked Open Data fall within the medium-scale category, as putatively defined in this dissertation. To cite some statistics from a well-known work, the English versions of two influential knowledge bases, DBpedia and Yago, both contain about 3 million English entities (Suchanek, Abiteboul & Senellart, 2011). Wikidata[142], another influential knowledge base, currently contains about 5 million English entities. Other knowledge bases (e.g. the New York Times ontology[143]) on Linked Open Data are far smaller. These statistics show that achieving scale on medium-scale datasets (using small clusters, for reasons outlined below) is a well-motivated problem at the present moment.

In a shared-nothing paradigm like MapReduce (Dean & Ghemawat, 2008), the pairs in the candidate set are partitioned and split among the various

---

141 This phenomenon was detailed in Section 2.4.1, which the interested reader may want to review at this juncture. Therein, a small-scale dataset was putatively defined to contain 100,000 entities or fewer, and a medium-scale dataset was defined to contain between 100,000-5 million entities.

142 https://www.wikidata.org/wiki/Wikidata:Statistics/Wikipedia

143 http://data.nytimes.com/

nodes. Thus, a third motivation is that the number of required nodes must grow *quadratically* in the input size for a constant reduction ratio and constant performance. Because practical and cost-effective scaling must exhibit near-linear time performance, it is important to achieve both a high reduction ratio and successfully deploy algorithms on a relatively small cluster[144] (between 16-60 cores) on current large-scale instance matching applications. The algorithms and evaluations in this section are designed with these motivations in mind.

8.3 MapReduce Implementations

The MapReduce paradigm[145] was described as the framework of choice for scaling the schematic in Figure 1.5 (Dean & Ghemawat, 2008). MapReduce was designed to be implemented on commodity hardware, as it relies on dynamic master-slave principles to achieve a high degree of data locality and fault tolerance. It has been widely adopted since its introduction (Dean & Ghemawat, 2008). To the best of our knowledge, all the major cloud vendors include an implementation of MapReduce as a service offering, with customers given the ability to 'spin up' and 'tear down' MapReduce clusters on demand. At the time of writing, new higher-level services continue to be implemented on top of the basic (sometimes, modified) MapReduce framework, Spark being a good example (Zaharia, Chowdhury, Franklin, Shenker & Stoica, 2010).

To be widely applicable, all implementations herein only rely on basic MapReduce functionality. To this end, all MapReduce experimental runs in this chapter were executed in the commercial *Microsoft Azure* cloud platform, using MapReduce-based HDInsight clusters[146], with post-execution analyses done locally on a serial machine. A3 nodes were homogeneously used to spawn clusters. A3 nodes contain 4 cores, 7 GB of RAM, and 285 GB disk space. In the configuration adopted for the experiments described in this chapter, two nodes were always used for the master node, while the number

144 In an era of cheap computational power, this requirement might seem odd. The reason is that Linked Open Data continues to grow. For a current system to be sustainable even in the immediate future, quadratic growth in cluster size (e.g. scaling from 16-60 cores to 100-1000 cores) must continue to be feasible and affordable for a community that aligns itself with Open Data initiatives.

145 The paradigm is briefly described in Appendix A.

146 https://azure.microsoft.com/en-us/pricing/details/hdinsight/

of data (i.e. slave) nodes was varied from 4-10. In the evaluations described subsequently in Sections 8.3.1-8.3.4, the term 'node' is used to refer to a data node. Note that this cluster range meets a key requirement laid out in Section 8.2 (the importance of using small clusters containing 10-60 cores). In the following experiments, the benefits of scaling are shown to be realized well within this range for all the datasets.

To assess the scalability of the various modules, wall-clock times are recorded and used for run-times. Where relevant, both map and reduce times (and also the overall run-time) are reported. The overall run-time does not necessarily equal the sum of map and reduce times, since reducers are known to commence before the map stage of a MapReduce algorithm has completely terminated.

For clarity of exposition, the MapReduce algorithms, where possible, are described by way of illustrations, rather than technical pseudocode. One reason for taking this liberty is that, after adjusting for the specifics of the MapReduce paradigm, much of the pseudocode repeats what has already been presented in prior chapters. The illustrations are expected to provide a more intuitive framework in which to assess system scalability.

Finally, the knowledge graphs input to the system are assumed to be serialized in the NoSQL format that was described and illustrated in Figure 2.1d.

8.3.1 Type Alignment

In Chapter 4, two applications of type alignment were described. The first application, alignment of *non-interlinked* types147, primarily concerns data integration, with real-world use-cases limited to small datasets (and serial settings). The second application concerns the alignment of interlinked types and is most useful when considering cross-domain knowledge graphs (e.g. DBpedia) as input. Such graphs have many hundreds of types, and millions of entities, and a scalable solution is warranted. The algorithmic principles behind the approach are not dissimilar to those introduced in Chapter 4 (Algorithm 4.1). Briefly, a type document is constructed for each type, after

147 An analogy used to explain the first application in Chapter 5 was that of matching semantically related files between two directories.

which a type matrix is constructed by computing a similarity score between each pair of type documents. Computations (e.g. the *max. Hungarian* algorithm) are executed on the type matrix to yield a type alignment, defined as a set of semantically aligned type pairs.

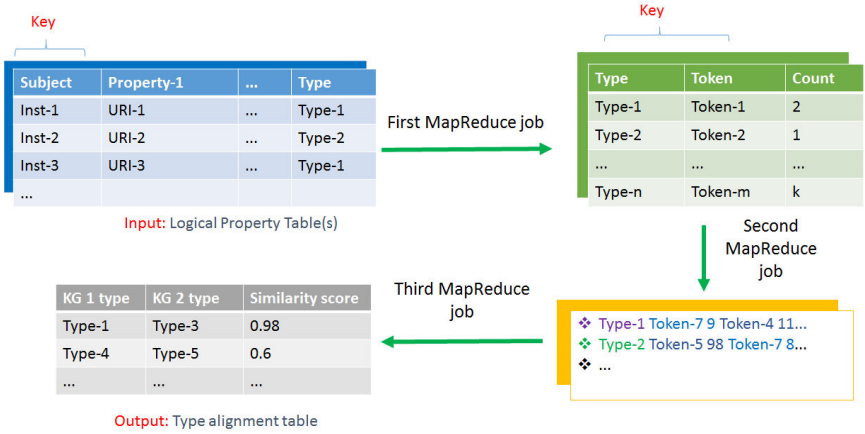


Figure 8.1: *Illustration of the MapReduce-based algorithm for scalable type alignment.*

The stages of a MapReduce algorithm for the steps above are illustrated in Figure 8.1. The algorithm consists of three *chained*[148] MapReduce jobs. An instance in each input knowledge graph (KG) is serialized in a NoSQL data structure (Figure 2.1d) containing the same information set as a row in a *logical property table* encoding (Kejriwal & Miranker, 2015a). First, a MapReduce job is run separately for each KG (say A and B), where a mapper takes an instance I_A ($\in A$) as input, converts it into a set of tokens using some standard delimiters (e.g. punctuations) and emits a key-value pair of the form ($Type(I_A)$-$token_i$, 1), where $Type(I_A)$ is the type of instance I_A and $token_i$ is a token from the information set of the instance. If a token occurs multiple times within an instance, it is still counted only once, since only type-token statistics need to be collected. The reducer simply counts all type-tokens and emits an intermediate output visualized in Figure 8.1.

[148] That is, the outputs of a job coincide with the inputs of the next job in the chain. In practice, chaining jobs can lead to significant reduction in individual job start-up times.

In the second MapReduce job, all the type-tokens, with their counts, are consolidated in a single line[149]. Data skew was found to be a significant problem in the original run of the second MapReduce job. To avoid skew, the mapper in the second MapReduce job did not emit tokens that had count less than 5; in the reducer, only the first 30,000 unique tokens emitted by a mapper (per type) were output. Formally, only these tokens constitute the contents of the *type document* representing that type.

An advantage of this construction is that the type documents are quite compact. The cross-domain knowledge graph (KG), Freebase, which is just slightly under 400 GB in uncompressed form, yielded a final type-token output of less than 150 MB upon execution of these first two MapReduce jobs. Due to the compactness, types between two KGs could be matched (by constructing a type matrix and performing the requisite computations), using the type documents, in a *single* reducer. Namely, in a third MapReduce job, each type document is assigned a single key, guaranteeing that all documents arrive at (exactly) one reducer. The construction of the type matrix (through pairwise similarity computations) and subsequent processing on the matrix are all performed at that one reducer[150]. The final type alignment is output to the distributed file system.

Following earlier results in Chapter 4, two standard set-based similarity measures (normalized in the [0,1] range) were used: $Jaccard$ and $Log\ TF$[151]. Formulae for these measures, and a rationale for why they are sufficient, were provided in Chapter 4.

Intuitively, the higher the similarity score according to a given measure, the more likely it is that the corresponding types are aligned. Using a threshold, which can be varied to trade-off various metrics against each other, a curve can be plotted to visualize empirical type alignment performance against a ground-truth.

---

149 Just like with the first MapReduce job, the second job is executed separately for each knowledge graph.

150 Though rarely warranted, some load balancing is possible in more extreme situations since the type matrix construction is equivalent to performing all-pairs set similarity joins (Vernica et al., 2010).

151 *Log-Term Frequency.* In supplementary work, more measures were also tried (e.g. a *generalized* version of $Jaccard$); the results are qualitatively similar.

Evaluations

The MapReduce-based type alignment module is assessed by its performance on the second application of type alignment, which concerns alignment of interlinked types. In recent work, this application has been tested on two knowledge graphs, DBpedia and Freebase (Duan et al., 2012). A similar methodology is followed herein.

The evaluation was set up as follows. First, the publicly available N-Triples files containing DBpedia and Freebase facts were downloaded[152] and stored in Microsoft Azure cloud storage. For DBpedia, two separate triples files had to be downloaded and merged into a single file. The first of these described instance type information[153], while the second described instance properties (facts). There were 3.279 million unique subjects, 67.1 million unique triples, and 417 unique types in the merged file. For Freebase, only one triples file was available and contained 121.629 million unique subjects, 3.023 billion unique triples and 4811 unique types.

A third-party file describing approximately 3.3 million *:sameAs* links between the instances was also downloaded and used as the ground-truth, following similar principles and arguments as those outlined in Chapter 4. Namely, type alignment was treated akin to blocking, and evaluated on the usual metrics of Pairs Completeness or PC (measuring *coverage* of instance pairs by the type alignment) and Reduction Ratio or RR (measuring *efficiency* savings).

Unlike the first application of type alignment, assumed in Chapter 4, one-one type alignments can no longer be assumed in the second type alignment application. Instead, a threshold-based approach is adopted. Given a threshold that varies from 0.0 to 1.0, a tradeoff between PC and RR can be plotted by aligning any type pair $< type_{freebase}, type_{dbpedia} >$ if the similarity score between the two types exceeds the threshold. This approach is both simple, and places no restriction on alignment cardinality.

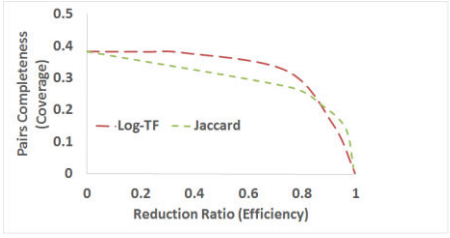
A MapReduce-based serialization algorithm was executed to convert the N-Triples files to the NoSQL format that is required by all MapReduce algorithms described in this chapter, including Algorithm 8.1. For all

152 The versions available in early August, 2015, were downloaded. To the best of our knowledge, Freebase has not been updated since then, but DBpedia continues to be updated annually.

153 Only *dbpedia.org/ontology* types were considered.

MapReduce algorithms, between 6 and 15 quad-core A3 data nodes[154] were used. The total serialization times for DBpedia and Freebase were 14 minutes and 4.5 hours respectively.

Once the data was serialized, the type alignment algorithms were executed for both DBpedia and Freebase, the respective run-times being 13 minutes and 6 hours for the first *two* jobs in the chained sequence shown in Figure 8.1. The data was then downloaded to a serial machine for further analysis. First, we verified that the data output by the second MapReduce job in the chain was compact (i.e. <1 GB) in order to justify the design decision in making the *third* MapReduce job in the chain *inherently serial* (by using a single key to route all elements to a single reducer, then constructing, and performing computations on, a type matrix within that reducer). The analysis showed that the Freebase output was less than 150 MB and the DBpedia output, less than 10 MB. Considering that Freebase is currently the world's largest-known encyclopedic knowledge graph, and is over 400 GB in uncompressed N-Triples form, this is a significant reduction, and provides support for the design decision.



(a)

Scoring method	Highest F-measure	Corr. Pairs Completeness	Corr. Reduction Ratio
Log-TF	45.06%	32.36%	74.17%
Jaccard	39.87%	26. 95%	76.53%

(b)

Figure 8.2: *Results of the MapReduce-based type alignment algorithm on DBpedia and Freebase, using blocking metrics.* (a) measures Pairs Completeness vs. Reduction Ratio by varying a threshold, while in (b), the highest-obtained F-measures are recorded, along with the corresponding values of Pairs Completeness and Reduction Ratio at which the F-measure was obtained.

Figure 8.2 illustrates the quality of type alignment itself in terms of the blocking metrics. Figure 8.2a shows that the $Jaccard$ measure is not as

154 These nodes were described at the beginning of Section 8.3.

effective as $Log\ TF$, which achieves the highest F-measure at 45.06%, with corresponding PC at 32.36%, and corresponding RR at 74.17%.

It is worthwhile comparing these numbers to those obtained in Chapter 4. Even at low RR, the maximum coverage obtained by the current system is only in the range of 40%, a worryingly low number. While a complete post-mortem analysis of this issue is beyond the scope of this dissertation, we note that an important cause for such poor coverage arises from *noisy* type declarations in Freebase and DBpedia. At least one other paper has noted a similar finding (Duan et al., 2012). Another minor reason is that a fraction of instances in DBpedia and Freebase *lack* type information, and would be ignored by the type alignment system.

Indirectly, these results show that data integration practitioners are rightfully concerned about not using these cross-domain graphs in mainstream applications. A promising avenue of future research is to automatically deduce and flag noisy type declarations, along with inferring type information for instances lacking type information (Section 9.2.1). On this latter issue, some progress has already been made in the literature by utilizing *hierarchical clustering* approaches (Ma, Tran & Bicer, 2013).

8.3.2 Training Set Generator (TSG)

A scalable training set generator (TSG) is implemented using two separate (i.e. unchained) MapReduce job sequences. Recall that the pseudocode of the original TSG algorithm was provided in Algorithm 5.1. In summary, the algorithm first located all instance pairs, using an efficient retrieval algorithm (Cohen, 2000), having a $LogTFIDF$ score above a provided threshold parameter $thresh$. The top-n pairs (with n also specified as a parameter) were output in ranked order according to their $Jaccard$ score, under the constraint that an instance occurs at most once in the entire output set (the so-called *uniqueness constraint*). Non-duplicates were generated by permuting this set. Note that the TSG (and all algorithms following it) has to be executed separately for each aligned type pair, unless the types belong to different domains[155].

---

155 Earlier evaluations showed that, if this were the case, type alignment was usually unnecessary and the TSG (and other algorithms) were able to *implicitly* differentiate between instances of incompatible types.

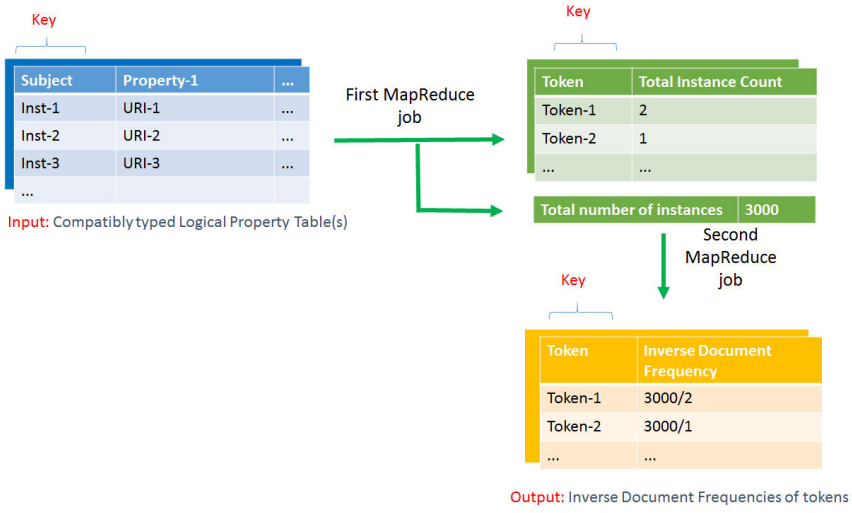


Figure 8.3: *Illustration of the chained MapReduce-based algorithm for generating token Inverse Document Frequency (IDF) statistics.*

To replicate the serial procedure in MapReduce, a preprocessing step is required to first generate an inverse document frequency (IDF) file. This file records the IDF[156] for each token as the quantity $\frac{Total\ number\ of\ instances}{Number\ of\ instances\ containing\ the\ token}$. While the division cannot be carried out in a single MapReduce job, the numerator (which is constant for a given type) and the denominator can be computed in parallel. The division is then performed in a MapReduce job chained to the first job (Figure 8.3).

[156] Per this interpretation, a 'document' is the set of tokens comprising each instance (line 3 in Algorithm 5.1). The property (or column in the property table) from which this token was derived is irrelevant.

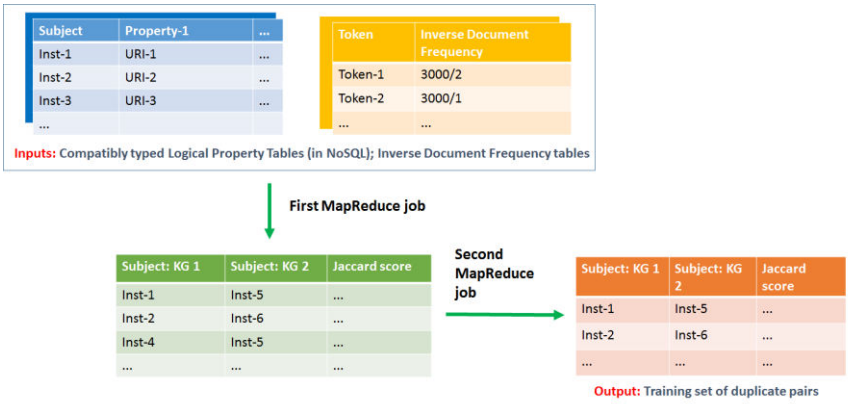


Figure 8.4: *Illustration of the MapReduce-based Training Set Generator.*

Figure 8.4 illustrates the primary TSG algorithm that accepts the generated IDF tables and the two single-type property tables (serialized using NoSQL data structures), and outputs a heuristically generated training set. The first MapReduce job is described as follows. In the mapper, the algorithm first checks whether the input is a row from one of the property tables or from the IDF tables. If the input is from an IDF table (of the form $< token, idf - count >$ the mapper emits $token$ as a key (with a *special symbol* s as the value) if $token$ has $idf - count$ above a pre-specified IDF threshold. The goal of this step is to reduce potential data skew by not transmitting tokens that are unlikely to be of much value. If the input is instead from a property table (i.e. a NoSQL object representing an instance), the instance r is tokenized into a bag of tokens $bag(r)$. Each (unique) token serves as an emitted key, with the corresponding value being the pair $< subject(r), bag(r) >$[157]. The mapper is identical to the mapper in the first MapReduce job in Figure 8.3, except that instead of a number, the entire bag of tokens (along with the subject of the instance) is emitted as a value. For a given value v, the symbols $v_{subject}$ and v_{bag} are used to denote its corresponding $subject$ and bag elements.

In the reducer, several measures are adopted to reduce potential data skew and increase training set quality. For completeness, these measures are

[157] Designating the token as $token$, the emitted unit would be of the form $< token, < subject(r), bag(r) >>$.

reproduced in the reducer pseudocode in Algorithm 8.1. The knowledge graphs that are input to the TSG, as NoSQL property tables, are designated using the symbols KG_A and KG_B. First, the reducer checks if the value is the special symbol s. It is important for the reducer to encounter this symbol, for the assumption otherwise is that the token that serves as the key of the reducer is too common in the dataset. Further processing will not take place (line 6). Otherwise, using the $subject$ of the value, the reducer identifies the value as representing an instance from either KG_A or KG_B and places it in A or B respectively.

Input: Key $token$ with associated set of $values := < v_1, \dots, v_n >$

Threshold parameters $purgeThresh, LogTFIDFThresh, JaccardThresh$

Special symbol s

Output: Key-value pairs of the form $\ll subject_1, subject_2 >, JaccardScore >$

Steps:

1. Initialize empty sets A, B and D'
2. Initialize Boolean flag $idfPresent := False$
3. **while** $values$ are streaming **and** $|A| \leq purgeThresh$ **and** $|B| \leq purgeThresh$:
4. **if** $value == s$ **then**

$idfPresent := True$

continue

else if $value$ represents an instance from KG_A

Add $value$ to A

else if $value$ represents an instance from KG_B

Add $value$ to B

else

Emit error signal and terminate

5. **end if**
6. **if** either $|A| == 0$ or $|B| == 0$ or $!\, s$ **then**

Terminate

7. **end if**
8. Place in D' all elements (a, b) ∈ $A \times B$ such that $LogTFIDF(a_{bag}, b_{bag}) \geq LogTFIDFThresh$
9. **for all** pairs $(a, b) \in D'$ **do**

$JaccardScore$:=Jaccard similarity between a_{bag} and b_{bag}

if $JaccardScore \geq JaccardThresh$ **then**

Emit $\ll a_{subject}, b_{subject} >, JaccardScore >$

end if

10. **end for**
11. Terminate

Algorithm 8.1: *The Reducer algorithm in the first MapReduce job in Figure 8.4.*

In the interest of avoiding data skew, a parameter $purgeThresh$ is used in the same spirit as *block purging* (Papadakis, Ioannou, Palpanas, Niederée & Nejdl, 2013). Specifically, the reducer stops streaming in new values emitted by mappers once either A or B reaches this threshold. The special symbol must have been encountered by the reducer by the time this happens. While conservative, this step ensures that no one reducer instance ends up blocking the MapReduce chain from terminating[158]. It also places strong theoretical guarantees on reducer performance.

The rest of the algorithm employs strategies similar (but not identical) to the serial TSG described earlier in Algorithm 5.1. $LogTFIDF$ is computed between the bags of tokens in A and B. The reducer key $token$ is explicitly disregarded in the computations since it is known to be common to all bags in that reducer.

The $Jaccard$ threshold parameter $JaccardThresh$ replaces the parameter n in Algorithm 5.1, the number of pairs desired, in Algorithm 8.1. Namely, all pairs with $Jaccard$ score above $JaccardThresh$ are output (as pairs of subjects), along with the $Jaccard$ score. Because MapReduce is a

[158] In informal parlance, referred to also as 'the curse of the last reducer'.

shared-nothing paradigm, this was a necessary change. If a practitioner insists on using n as a parameter, but does not wish to re-run the sequence in Figure 8.4 more than once, the only safe course of action is to set the $Jaccard$ threshold parameter to 0, and run an additional sorting algorithm on the output of the MapReduce sequence in Figure 8.4. In practice, adopting a high $Jaccard$ threshold is more sensible: a practitioner is unlikely to know (or even correctly estimate) a good value for n for large datasets. On the other hand, $Jaccard$ is a *local* similarity function, independent of the actual size or statistics of the dataset. A threshold can be specified with some confidence, independent of the dataset[159]. This threshold should be reasonably high (e.g. 0.8) to prevent the training data from becoming too noisy.

In the second MapReduce job, the training set is deduplicated. This step is necessary because duplicate pairs typically share several tokens in common, meaning that more than one reducer can emit the same pair. The program is fairly trivial: each mapper 'passes through' its input to the reducer by emitting it as the key, along with some dummy value. The reducer emits the key as output only once, ignoring multiple occurrences.

A last point is the implementation of the uniqueness constraint, and the generation of non-duplicates (via permutation). For minimal overhead, these steps are best implemented in subsequent steps (the property alignment and learning procedures).

Evaluations

Test Cases

To rigorously test the TSG in an environment where the size, distribution of duplicates and type of noise are controlled for, medium-scale *census* datasets were generated using a synthetic benchmark generator, FEBRL, that is well established as a testbed in the instance matching community (Christen, 2008a; Köpcke, Thor & Rahm, 2010). FEBRL uses real-world underlying census data to generate synthetic datasets with some user-specified parameters. The specifiable parameters include (1) the total

[159] For the same reason, a practitioner must be conservative (and pessimistic) about the global thresholds (e.g. $LogTFIDFThresh$) since these *do* depend on dataset size and statistics. In Chapter 5, it was argued that setting a low value for such thresholds was typically sufficient.

number of original records (2) the total number of duplicate records, (3) the maximum number of duplicates for an original record, (4) the distribution of duplicates (one of $Uniform, Zipf$ or $Poisson$), (5) the maximum number of modifications per attribute when generating a duplicate record, and (6) the maximum number of modifications per record[160] when generating a duplicate record. Another parameter that can be specified but that is left fixed for all experiments in this section is the *type* of noise (phonetic, typographical, optical character recognition or all three) that can be used to distort attribute values in an original record when generating a duplicate. For maximal real-world representativeness, all types of noise were permitted in the generative process.

Dataset Name	**Total number of records = File 1 + File 2**	**Max. duplicates per original record**	**Max. modifications (Per field/Per Record)**
Census(X; 50,000)	50,000 = 30,000 + 20,000	5	3/5
Census(X; 100,000)	100,000 = 60,000 + 40,000	5	3/5
Census(X; 500,000)	500,000 = 300,000 + 200,000	5	3/5
Census(X; 1,500,000)	1,500,000 = 900,000 + 600,000	3, 4, 5	4/5, 2/4, 3/5

Table 8.2: *Parameter settings for the four generated dataset classes.*

The variable X in the name column may take a value from the duplicates distribution set $\{Uniform, Poisson, Zipf\}$, leading to a total of twelve datasets generated. See text for an explanation of ternary-valued column values (Columns 3 and 4) for the last dataset. Only matches between the two files are valid.

To assess instance matching scalability, twelve datasets were generated, four for each of the three duplicates distributions in (4). The parameters of these four datasets are given in Table 8.2. Note that the FEBRL

[160] That is, across all attributes. This number must necessarily be larger than the one specified in (5).

generator has a random component; for exact reproducibility of the results in this section, the generated datasets have also been archived.

The dataset sizes in Table 8.2 are motivated by the discussion in Section 8.2. Specifically, a medium-scale test suite was argued to fall within the domain of a large-scale instance matching application. Within an order of magnitude, the datasets in Table 8.2 reflect the sizes of the most popular datasets on Linked Open Data. In Chapter 3, it was noted that many parallel and distributed instance matchers were often evaluated on datasets containing far fewer than a million entities (Section 3.1.3). This has also been the case in more recent work[161].

Even during generation, the FEBRL generator ran out of memory on a node with over 7 GB of RAM when generating the Census(X; 1,500,000) for all three duplicates distributions ($X \in \{Uniform, Poisson, Zipf\}$). In order to build a dataset with 1.5 million records, three datasets with 500,000 records were generated (hence, the ternary-valued parameter values in Columns 3 and 4 in Table 8.2), and combined[162]. Note that this *piecewise* methodology is expected to have a negative impact on the precision and recall measures of any instance matcher. This is because there is a non-trivial probability that records in one of the three datasets matches with records in the other two datasets, and will be retrieved by the system. In the ground-truth file, such matches will not have been recorded (because the three datasets were independently generated), meaning that they will be counted as incorrect in the evaluations. Although this caveat is expected to affect accuracy metrics, scalability metrics are less likely to be impacted, especially if *duplicates distribution indifference* can be conclusively established.

Note that the FEBRL generator was originally intended for Relational Database instance matchers and the generated files adhere to the Comma Separated Values (CSV) format. Among the serializations introduced in Chapter 2, and illustrated in Figure 2.1, CSV most closely conforms to the format of the logical property table (Figure 2.1c). The MapReduce algorithms presented in this chapter rely on a NoSQL format (Figure 2.1d). A natural first question to investigate is if the serialization can be achieved in a near-linear

---

161 For example, Sadosky, Shrivastava, Price & Steorts (2015) show that even datasets containing about 300,000 entities require extremely high reduction ratios to successfully terminate.

162 For convenience, because the parameter values of the *first* of the three datasets and Census(X; 500,000) are identical, the corresponding Census(X; 500,000) dataset was re-used as the first dataset.

time fashion with only a few nodes. To that end, a serialization algorithm converting the CSV files to NoSQL files was executed on a 4-node HDInsight cluster that can be spun up in Microsoft Azure in only a few minutes. The algorithm operates by performing all its computations in the mapper. First, the property schema is placed in the distributed cache. Using the schema, each record is converted into a self-contained NoSQL JSON-like object. The reducer is a trivial identity function.

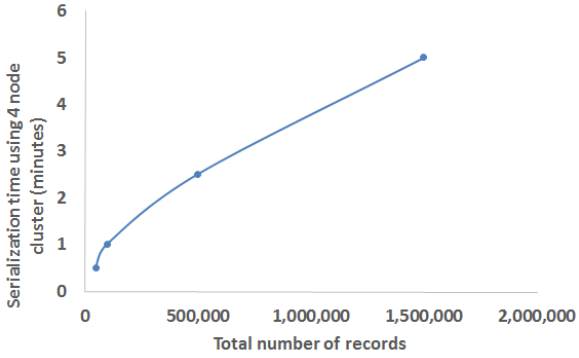


Figure 8.5: *Serialization results on a 4-node HDInsight cluster.*

Figure 8.5 illustrates the results of this process, and confirms both the low wall-clock run-time and the linear-time dependence. For the largest dataset, for example, the run-time was in the vicinity of only five minutes. The results show that assuming the NoSQL serialization for inputs is not expected to cause problems of scale. Also, the serialization run-time was found to be independent of the duplicates distribution.

Methodology

All MapReduce experiments were conducted on the Microsoft Azure platform, described at the beginning of Section 8.3. The TSG parameters were set as follows. The IDF threshold required in the TSG mapper was set using the formula $\frac{Total\ Number\ of\ records}{50}$. This number ensures that a token is only acceptable as a key if it occurs in at least 2% of the total number of records (e.g. 1000 for the dataset containing 50,000 records). This rather conservative

estimate ensures that the majority of the records do not get filtered out in the mapper itself, in which case, studying the scaling properties of the TSG becomes problematic. Three other threshold parameters that need to be set in the reducer are the purge threshold $purgeThresh$, $LogTFIDF$ threshold $LogTFIDFThresh$, and $Jaccard$ threshold $JaccardThresh$. Following earlier arguments, $purgeThresh$ is set to a moderately low value of 50 (to avoid data skew), $LogTFIDFThresh$ is set to an extremely low (but non-zero) value of 0.001 and $JaccardThresh$ is set to a high value of 0.8. In several early experiments, these values were found to lead to good TSG performance.

Results

Duplicates distribution indifference: The primary rationale behind generating datasets with different duplicate distributions was to test the sensitivity of the algorithms to the distribution. To that end, the TSG run-times across the three distributions were recorded and compared.

We found that, for constant cluster size (across a range of cluster sizes) and dataset size, the duplicates distribution had no impact on map, reduce or overall run-times. The maximum difference in the recorded run-time data across two different distributions was only half a minute (usually less than 5% of the total recorded run-time), which could be attributed to cluster variance. In summary, the TSG exhibits duplicates distribution indifference.

Scaling: Using a small cluster with four data nodes and two master nodes (24 cores in total), the TSG was executed on the four different datasets in Table 8.2 for the Zipf duplicates distribution. To evaluate whether scaling had been achieved, the TSG was also executed on clusters that were approximately twice as large (with 40-48 cores in total).

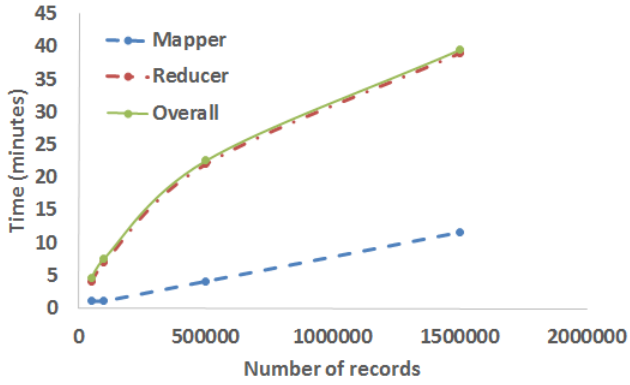


Figure 8.6: *Training set generator run-time results.*

Figure 8.6 illustrates the results of the 24 core run for the datasets exhibiting the Zipf duplicates distribution. The reducer (outlined in Algorithm 8.1) is more compute-intensive than the mapper, and is nearly coincidental with the overall run-time curve. Concerning runs on bigger clusters, the results were found to be near-identical to the 24-core run, implying that full scaling had already been achieved on the small cluster, even for the dataset with 1.5 million records. This provides good evidence in support of a key motivation outlined in Section 8.2, namely, that existing systems must scale near-linearly using relatively *small* clusters[163].

Accuracy of Training Set: In keeping with previously introduced notions that the TSG is only reliable for small training sets, the top hundred pairs (sorted by $Jaccard$ score, using $JaccardThresh = 0.8$) output by the TSG were analyzed using the *precision* metric. In all twelve cases, the precision was found to be 100%, justifying the use of a high $Jaccard$ threshold and a small retrieval size[164].

---

163 The rationale was that, due to the large intermediate output sizes, clusters would have to grow quadratically with proportional growth in the data (number of entities). Linked Open Data has consistently shown super-linear growth since it first emerged in 2007 (Schmachtenberg, Bizer & Paulheim, 2014).

164 It is possible to achieve 100% precision with higher retrieval sizes for the larger datasets, but this implies, unrealistically, that the number of duplicates is known *a priori* and that the retrieval size can be tuned accordingly. A similar argument was used to motivate the results in Table 5.2 in Chapter 5.

It is also constructive to study the total number of instance pairs output by the algorithm for each of the cases. Given real-world observations, this number should only grow linearly with the dataset.

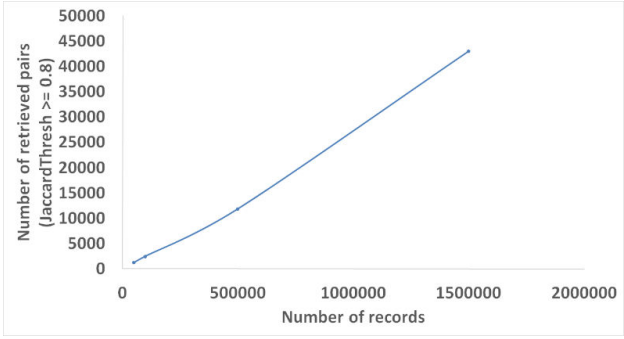


Figure 8.7: *The mean number of instance pairs output by the TSG as a function of the total number of records, with the mean taken across the three duplicates distributions.*

Figure 8.7 shows that this is roughly the case: the growth is only slightly super-linear for small datasets, and linear for larger datasets. This provides evidence that the parameter settings are judicious (and may even be further improved). Indifference of the algorithm to duplicates distributions was again established: regardless of the number of records, the standard deviation of the number of pairs retrieved by the TSG, taken across the three duplicates distributions, was less than 3% of the mean.

8.3.3 Property Alignment and Learning Procedures

Although finer-grained, the property alignment procedure (Algorithm 6.1) is not dissimilar, in principle, to the type alignment procedure (Algorithm 4.1). Both algorithms rely on building a similarity matrix, and then performing a variety of calculations on the matrix. For type alignment, simple distance calculations between type documents suffices for populating the matrix (Chapter 4), while for property alignment, hybrid techniques are required for good performance (Chapter 6).

From a scalability perspective, an important difference arises from the observation that Algorithm 6.1 only needed the training sets, assumed to be of

modest size (per the discussion at the end of Section 5.2), to derive the similarity matrix. The algorithm itself is *inherently serial* in that the matrix needs to be on a single node[165] to facilitate the intended computations. In no real-world case (even those beyond the scope of data integration) did we encounter a scenario where this assumption was problematic.

These observations indicate that designing a parallel algorithm from scratch is not necessitated, and Algorithm 6.1 can be exactly re-implemented in MapReduce by first assigning all pairs output by the TSG, a *single key* in the map program, and then executing Algorithm 6.1 in an off-the-shelf fashion in the one reducer where all inputs are guaranteed to arrive[166]. One additional step that needs to take place in the reducer before Algorithm 6.1 is executed is the implementation of the uniqueness constraint and the generation of non-duplicates (via permutation). Since all the duplicate training samples arrive at the reducer, this would mean executing lines 8 and 9 of Algorithm 5.1 before executing Algorithm 6.1 Once computed, the property alignment is written out to the distributed file system as output. To avoid repetitive computations, the 'new' training set (with the uniqueness constraint implemented, and the non-duplicates generated) is also output. The evaluation of property alignment is trivial from a scalability standpoint, and is not reproduced here.

Deriving the blocking scheme B and classifier C (the learning procedures in Chapter 7) relies on similar observations. Again, because the (new) training set and property alignment are both of modest size, a single reducer can be relied upon for the inherently serial processing. Note that, in the case of the learning procedures, some computation (by way of feature generation) can be offloaded to the mapper. This is accomplished by placing the property alignment in the *distributed cache* and using it to convert each instance pair (in the training set) into a feature vector in the mapper itself. The feature vector, and not the raw strings constituting the instance pair, is shuffled to the reducer. Because the features proposed in Chapter 7 are binary, efficient representations can be used to significantly reduce shuffling costs.

In the reducer, either the blocking scheme or classifier (or both) can be learned using the vectors, with the model parameters output to the

---

165 A stronger assumption is that the matrix needs to be in main memory, which was also not found to be problematic in real-world cases. In the rare case where this would be violated, Algorithm 6.1 can still be implemented without loss of functionality.

166 An erroneous assumption would be that, the inputs being of modest size, they can be expected to reside in a single map node. The distributed file system does not guarantee this for any input size.

distributed file system. The next section reports on some evaluation results for the blocking and similarity steps.

8.3.4 Blocking and Similarity

Similar to the training set generator, the blocking and similarity steps involve processing the complete property tables P_1 and P_2, although they also take as inputs the blocking scheme B and the machine learning classifier C. Given that B and C are modest-sized files, representing functions, they are well-suited for the distributed cache functionality accompanying all known MapReduce implementations. Before mappers and reducers are executed, the files describing B and C are read, transformed to the appropriate data structures, and placed in the cache. Because of the modest size of these files, the shuffling cost of this pre-execution step is low.

In each mapper, an entity in the property table is read in as a NoSQL object. Blocking key values for the entity are generated by accessing B and applying it on the entity. The values serve as the reducer keys for that entity. In each reducer, the entities are collected into two sets[167] as they are shuffled. Using block purging, data skew is controlled in a straightforward fashion (Papadakis et al., 2013). Once a set exceeds a pre-specific size ($purgeThresh$), the reducer is terminated. This part of the algorithm is nearly identical to the first part of Algorithm 8.1.

Once the streaming of blocked entities (into a given reducer) is complete, and assuming that (1) the reducer has not terminated, and (2) both sets are non-empty, each pair is converted into a feature vector, and the distributed cache is accessed again. The classifier C is used to classify the vector with a given probability. If the probability exceeds a pre-specified threshold, the pair is output.

Some important points are of note. Although the experiments in Chapter 7 assumed an SVM classifier, any model can be used, as long as the program interprets the model correctly. It is also possible for C to not be a machine learning classifier at all, but a rule-based or distance-based program. Similar observations apply to the blocking scheme B (which may not be a DNF scheme) and also to the feature vector generation process. At a high-

[167] One set for each property table.

level, any well-defined blocker can be executed in the mapper, and any well-defined similarity function, in the reducer.

This genericity has obvious practical ramifications. In the instance matching literature, much recent progress has been due to the use of better classifiers, better blocking techniques (Table 7.2) and better features (Christen, 2012b). As argued in Chapter 2, the two-step instance matching framework is unlikely to be superseded in the foreseeable future. The MapReduce algorithms presented in this section explicitly accommodate genericity of classifiers, blocking schemes and feature generators.

Evaluations

The goal of these evaluations is to demonstrate a proof-of-concept execution of the described MapReduce-based blocking and similarity algorithms using a classifier, blocking scheme and feature generator that are different from those adopted in the serial evaluations in Chapter 7. The twelve census datasets used in the TSG evaluations are used again as test cases.

Methodology

As a first step, a series of pilot experiments were conducted on the three datasets containing 50,000 records each (for all three duplicates distributions) using three classifiers available in the Weka[168] package: decision table, Gaussian Processes (GP) and linear regression (Hall, Frank, Holmes, Pfahringer, Reutemann & Witten, 2009). A small set of 100 duplicate pairs and 1000 non-duplicate pairs were used for training and validating each of the three classifiers, with the rest used for testing. The goal is to settle on a classifier for the actual proof-of-concept experiment. Sixteen token-based real-valued features were used to represent each instance pair, and were generated as follows.

First, given an instance pair $< i, j >$, each of the two instances was parsed into four bags. The first bag contains all tokens that occur in property URIs. The second bag contains all object tokens, where the object is a literal. The third bag contains all tokens in the instance that do *not* fall within the first

168 Accessed at http://www.cs.waikato.ac.nz/ml/weka/.

two bags (e.g. subject tokens, and object tokens where the object is not a literal). The fourth bag is the union of the first three bags.

Once both i and j have been parsed into four bags each, sixteen features can be constructed by computing a similarity score between all pairwise combinations of the four bags (between the two instances). Given its robust performance in prior evaluations, the $Jaccard$ similarity score was used[169].

A particular advantage of the feature generation process described above is that it does not rely on a property alignment. In that sense, the features are *schema-free*, not dissimilar to computing a bag-of-words representation for each instance. Note that, despite its simplicity, not using a property alignment has an implicit performance cost: prior structural information that *could* be used to inform the instance matching process is not being exploited (Chapter 6).

In the pilot evaluations, for almost all classifiers and duplicates distributions, a high enough classification threshold (0.7 and above) was found to lead to F-Measures (of precision and recall) well above 95%, and in many cases, 99%. The linear regression classifier fared slightly worse than the other two. The experiment also confirmed earlier findings of duplicates distribution indifference.

Following these results, the main proof-of-concept experiment was set up as follows. Attribute Clustering (AC), rather than DNF blocking, was used as the blocking method (Papadakis et al., 2013). The Gaussian Processes (GP) model was chosen as the classifier and loaded into the distributed cache. The sixteen features described above were used for the feature generation process. In terms of overall code-writing effort, the changes were not extensive. The mapper from the TSG component of the system was re-used, but with a more aggressive strategy for tuning the IDF threshold[170]. Similarly, the $purgeThreshold$ parameter was fixed at a lower value of 10 to ensure a sufficiently high reduction ratio. The goal of this tuning is to filter out more

169 Before computing the score, an additional preprocessing step was undertaken. Specifically, given that many strings in RDF graphs are *opaque* (i.e. semantically meaningless, except for syntactic or record-keeping processes), all bags were purged of strings that did not only contain alphabets. This step was found to have significant benefits.

170 The original IDF threshold value (given earlier by $\frac{Total\ number\ of\ records}{50}$) is multiplied by 5.

records in the mapper, so that the reducer is less compute-intensive than before.

Results

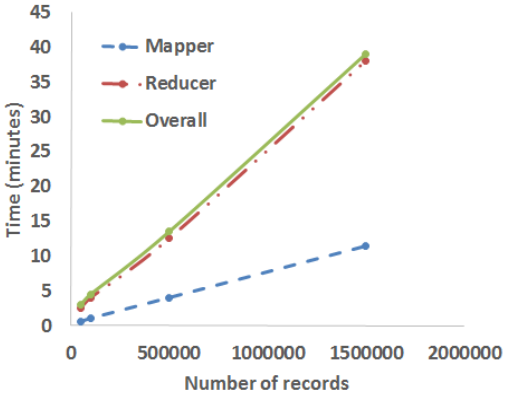


Figure 8.8: *Blocking-similarity run-time results using Attribute Clustering (AC) blocking and a Gaussian Processes (GP) classifier.*

Figure 8.8 illustrates the results of the proof-of-concept experiment. Similar to the TSG, the process takes most of its time in the reducers, but achieves performance that is considerably more linear, owing to more aggressive parameter tuning. Figure 8.8 also reinforces the evidence of the system achieving scale using small clusters only. Most importantly, the experiment demonstrates, through a live run, that the system can be deployed in a flexible manner. That is, innovations in blocking and feature generation are not expected to make the architecture unusable.

Chapter 9: Conclusion

9.1 Summary

Datasets in Linked Open Data (LOD) are published on the Web using a guiding set of four Linked Data principles (Bizer, Heath & Berners-Lee, 2009). The fourth principle stipulates that datasets should not be published in silos, but be interlinked. To fulfill this principle, *:sameAs* declarations are often used to declare synonymy relations between entities that refer to the same underlying entity. These synonyms are equivalent to the population of an *Entity Name System* (ENS), defined as a thesaurus for entities. An ENS is a crucial component in a *data integration* architecture (Lenzerini, 2002; Doan, Halevy & Ives, 2012).

Devising an algorithmic solution to the problem of finding synonymous pairs of entities is known as *instance matching* (Ferrara, Nikolov & Scharffe, 2013). Solving the instance matching problem is central to populating a Linked Data ENS.

The thesis statement in Chapter 1 stated that *given the current state of Linked Open Data, a feasible instance matcher must simultaneously fulfill the four requirements of domain-independence, automation, scalability and heterogeneity, referred to henceforth as the DASH requirements*.

A set of arguments, both data-driven and intuitive, is used to support the statement. Domain-independence is necessary because organizations in many domains contribute to Linked Open Data (LOD). At the time of writing, numerous datasets in the domains of social media, publications, government and education are already available (Schmachtenberg, Bizer & Paulheim, 2014). *Cross-domain* knowledge bases, which describe encyclopedic content and are like structured versions of Wikipedia, are also extremely popular on LOD.

Automation is necessitated both by current trends, as well as the growth of LOD. Since LOD is an open community, and involves many organizations that are not always equipped for dealing with complex technical challenges, a sustainable solution must use techniques that are both robust and involve low amounts of labeling effort.

Scalability is motivated by the *pairwise* nature of instance matching (Section 2.4.1). Even though many of the largest and most important LOD datasets are *medium-scale* (between 100,000 to 5 million entities), processing them serially is challenging because of the *super-linear* (though sub-quadratic) run-time complexity of state-of-the-art instance matching workflows. Examples were used to illustrate that, in the context of instance matching, medium-scale datasets lead to large-scale applications. In Section 2.4.2, MapReduce was introduced as a distributed paradigm of choice for implementing a scalable instance matching workflow in the cloud (Dean & Ghemawat, 2008).

Finally, heterogeneity, more precisely referred to as *structural heterogeneity*, arises on Linked Open Data both because of type and property heterogeneity. The former problem arises because entities of different types are interlinked with each other in a non-trivial way, while the latter arises because entities of compatible types are represented using different sets of properties. In the vast majority of the instance matching literature, two structurally heterogeneous datasets are typically assumed to be homogenized (perhaps in a preprocessing step) before being input to an instance matcher (Elmagarmid, Ipeirotis & Verykios, 2007). On Linked Open Data, such an assumption is unrealistic.

The development of an instance matcher meeting the DASH requirements was motivated through a review of related work from the lens of the DASH requirements. In Chapter 3, this review was used to argue that, at present, no one system can purport to simultaneously meet the full set of requirements.

The output of this dissertation is an instance matcher that fulfills the four DASH requirements. The schematic of this system was presented and briefly described in Section 1.5. The system accepts two structurally heterogeneous RDF graphs as input, and outputs a set of *:sameAs* links that can be used to populate a Linked Data ENS.

The primary core contribution of the dissertation is an unsupervised training set generator (TSG) designed specifically for heterogeneous RDF graphs. To the best of our knowledge, this is the first such TSG that has been shown to be viable in enabling an unsupervised execution of a full instance matching pipeline. The TSG uses two fast, token-based heuristics to yield a small set of 'easy' examples that are used to bootstrap later learning processes.

Because the TSG is completely unsupervised, there is often noise in the 'seed' training set generated by the algorithm, which makes the design of robust, generalizable learning processes challenging. The second core contribution of this dissertation is a property alignment algorithm that accommodates some of these challenges. In particular, the algorithm is *parameter-free* and uses a combination of informative signals to achieve consistently high recall (without trivially degrading precision) across a range of domains and datasets. By virtue of high recall, the property alignments are viable for extracting useful structural features that are used to learn discriminative blocking and similarity functions.

A particular class of blocking keys called Disjunctive Normal Form (DNF) blocking keys is known to be particularly useful for *homogeneous Relational Databases*. As a third core contribution, we present a DNF blocking key learner for *heterogeneous RDF graphs*. Empirically, the learned DNF blocking keys are shown to outperform a state-of-the-art RDF blocking method.

Additionally, note that the *domain independence* of various components detailed in Chapters 5-7 was established by employing a test suite comprising ten multi-domain RDF datasets. Chapter 8 discussed a MapReduce-based implementation of the schematic. A set of controlled experiments on a small cluster showed that the implementation exhibited near-linear scaling, and successfully accommodated data skew (i.e. was indifferent to the distribution of duplicates). Auxiliary evaluations were also used to illustrate system performance on the large-scale type alignment task.

9.2 Future Work

While we hope that this work represents progress in realizing the overall goal of viable information integration in the Linked Open Data ecosystem, we recognize that it does not solve the problem. To that end, several areas of future research that we believe to be promising, are briefly (and non-exhaustively) covered below.

9.2.1 Linked Data Quality

The large-scale type alignment evaluation in Chapter 8 showed that, due to their encyclopedic nature, *cross-domain* knowledge graphs on Linked Open Data exhibit a high degree of type heterogeneity. Consider Freebase, for example, which contains 4811 types, and DBpedia, which contains just over 400 ontological types. This would not be problematic if there was a well-defined way of defining a *ground-truth* for evaluating type alignment between such graphs. In some of our most recent experiments, we found that there are at least *three* different ways of constructing such a ground-truth, and that each has been used in prior research at least once. These ground-truths were not found to be overly consistent, implying that the *perceived* success of a type alignment solution is based on the adopted ground-truth[171].

This result is important because type alignment is a basic preprocessing step in many knowledge discovery processes on structured graphs, with extensions beyond instance matching. Two other noted applications are semantic search and ontology matching (Bouquet & Molinari, 2013; Euzenat & Shvaiko, 2007). The question arises as to *why* these ground-truths are inconsistent to begin with. We attempted a preliminary answer to this question by hypothesizing that a sizeable number of type declarations in Linked Open Data knowledge graphs are noisy. This noise is likely to have a direct impact on any application that makes use of these declarations. Since the number of types is far fewer than the number of instances and properties, dealing with noise at the *type* level, rather than the *instance* level, is a promising avenue for improving the overall quality of medium-scale datasets on Linked Open Data. This, in turn, could facilitate broader, more mainstream, data integration applications.

9.2.2 Schema-free Approaches

The growing diversity of Linked Data suggests that the time may be ripe for further investigation of *schema-free* knowledge discovery approaches. Traditional approaches, inspired mainly by the Relational Database literature, almost always perform some form of schema matching (e.g. *property*

---

171 In the evaluations in Chapter 8, one definition of ground-truth was adopted throughout (which was also the definition adopted in Chapter 4). This is because type alignment was evaluated in the *context* of instance matching. Without this context, evaluating type alignment becomes ill-defined.

alignment was a core component in the dissertation system) in an attempt to homogenize the data before doing further processing. The conventional wisdom was that such homogenization is necessary both qualitatively and computationally[172].

A growing body of research is disputing this wisdom, with the result being an increased use of schema-free techniques. In a recent paper, for example, we developed a schema-free version of the classic Sorted Neighborhood algorithm for RDF data, and showed that it compares favorably to an established baseline[173] (Kejriwal & Miranker, 2015d).

Schema-free algorithms constitute a relatively novel area of research, and many questions remain. For instance, which schema-free features are 'good', and how do we discover them? Can deep learning[174] be used to automate this feature discovery process? Can schema-free algorithms be used to bypass type and property alignment completely, or embedded in a hybrid framework to realize the benefits of both worlds?

9.2.3 Transfer Learning

The Linked Data ecosystem is a good candidate for applying *transfer learning* techniques (Pan & Yang, 2010), the reason being the high connectivity of many datasets to the encyclopedic graphs (e.g. DBpedia). When linking a new dataset to existing datasets on Linked Open Data, the exhaustive and time-consuming process of gathering training data, and determining best-fit parameters, among other things, could well be automated by utilizing past models and training data from approximately similar sources.

While transfer learning is hardly a new area (Baxter, 1998), its potential continues to be actively researched in several mainstream machine learning applications (Mesnil et al., 2012). To the best of our knowledge, its potential on Linked Open Data has not been fully explored, especially at

---

172 In Chapter 2, the rationale behind this wisdom was presented and illustrated in Section 2.3.2.

173 The baseline in that paper was Attribute Clustering (AC), which was also employed as a baseline in Chapter 7.

174 It is by no means certain that deep learning can be successfully used for instance matching, owing to data sparsity.

scale[175]. Given the recent trends in automation, such exploration may prove to be rewarding.

[175] We are aware of only one work that applies transfer learning techniques in a Linked Data instance matching application (Rong et al., 2012).

Appendix A: MapReduce

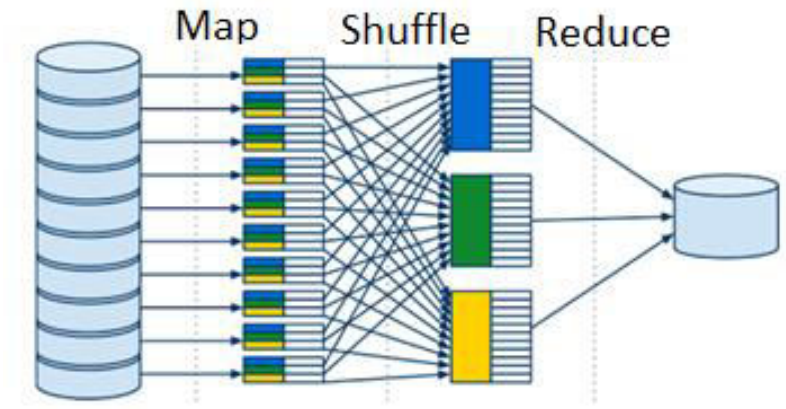


Figure A.1: *Abstract overview of the MapReduce paradigm*. For each unit of data (e.g. a record) input to a *mapper*, a set of $< key, value >$ pairs are generated. Each pair is *shuffled* (i.e. logically transported) to a reducer, which is said to be indexed by the key. All pairs sharing a key are guaranteed to arrive at the same reducer. Inputs to the mappers are read from, and outputs from the reducers written to, the distributed file system in the form of $< key, value >$ pairs.

MapReduce is a shared-nothing master-slave paradigm that relies on a distributed file system or DFS (Dean & Ghemawat, 2008). An illustration is provided in Figure A.1. A MapReduce program comprises a *map* program and a *reduce* program[176]. A mapper reads in a unit of data as a $< key, value >$ pair, and processes it to output (equivalently, *emit*) a set of $< key, value >$ pairs. Mapper outputs are *shuffled* across the network so that each pair with the same key is guaranteed to arrive at the same reducer. Thus, the reducer receives as input a single key and a set of values. The reducer processes its inputs and emits another set of $< key, value >$ pairs. While local storage may be used during intermediate MapReduce computations, the final outputs are always written out to the DFS. Note that mappers and reducers are *logical*

[176] Optionally, a *combine* program can also be specified to save network shuffling costs (Dean & Ghemawat, 2008).

processes; several mappers and reducers may be spawned on a single node in a *physical* implementation (White, 2012).

The master-slave principles of MapReduce have some powerful advantages. The first of these is *data locality* (Dean & Ghemawat, 2008). The master process dynamically spawns mappers and reducers in an attempt to keep shuffling and I/O costs to a minimum. It also controls redundancy, and the re-spawning of processes in the event of a failure. For this reason, MapReduce proves to be surprisingly robust. A typical cluster can be implemented on commodity hardware, without the need for specialized servers.

References

Alani, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H., & Shadbolt, N. R. (2003). Automatic ontology-based knowledge extraction from web documents. *Intelligent Systems, IEEE*, *18*(1), 14-21.

Allemang, D., & Hendler, J. (2011). *Semantic web for the working ontologist: effective model*

Angles, R., & Gutierrez, C. (2005). Querying RDF data from a graph database perspective. In *The Semantic Web: Research and Applications*(pp. 346-360). Springer Berlin Heidelberg.

Araujo, S., Hidders, J., Schwabe, D., & De Vries, A. P. (2011). Serimi-resource description similarity, rdf instance matching and interlinking. *arXiv preprint arXiv:1107.1104*.

Arenas, M., Díaz, G., Fokoue, A., Kementsietsidis, A., & Srinivas, K. (2014). A principled approach to bridging the gap between graph data and their schemas. *Proceedings of the VLDB Endowment*, *7*(8), 601-612.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., ... & Harris, M. A. (2000). Gene Ontology: tool for the unification of biology. *Nature genetics*, *25*(1), 25-29.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). *Dbpedia: A nucleus for a web of open data* (pp. 722-735). Springer Berlin Heidelberg.

Baxter, J. (1998). Theoretical models of learning to learn. In Learning to learn (pp. 71-94). Springer US.

Baxter, R., Christen, P., & Churches, T. (2003, August). A comparison of fast blocking methods for record linkage. In ACM SIGKDD (Vol. 3, pp. 25-27).

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval* (Vol. 463). New York: ACM press.

Bellahsene, Z., Bonifati, A., & Rahm, E. (2011). *Schema matching and mapping* (Vol. 20). Heidelberg (DE): Springer.

Benjelloun, O., Garcia-Molina, H., Gong, H., Kawai, H., Larson, T. E., Menestrina, D., & Thavisomboon, S. (2007, June). D-swoosh: A family of algorithms for generic, distributed entity resolution. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on* (pp. 37-37). IEEE.

Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S. E., & Widom, J. (2009). Swoosh: a generic approach to entity resolution. The VLDB Journal—The International Journal on Very Large Data Bases, 18(1), 255-276.

Bhattacharya, I., & Getoor, L. (2006, April). A Latent Dirichlet Model for Unsupervised Entity Resolution. In *SDM* (Vol. 5, No. 7, p. 59).

Bilenko, M., & Mooney, R. J. (2003, August). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 39-48). ACM.

Bilenko, M., Kamath, B., & Mooney, R. J. (2006, December). Adaptive blocking: Learning to scale up record linkage. In *Data Mining, 2006. ICDM'06. Sixth International Conference on* (pp. 87-96). IEEE.

Bilke, A., & Naumann, F. (2005, April). Schema matching using duplicates. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (pp. 69-80). IEEE.

Bizer, C. (2009). The emerging web of linked data. *Intelligent Systems, IEEE*, *24*(5), 87-92.

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts, 205-227.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, *3*, 993-1022.

Borthakur, D. (2007). The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, *11*(2007), 21.

Bouquet, P., & Molinari, A. (2013). A global entity name system (ens) for data ecosystems. *Proceedings of the VLDB Endowment*, *6*(11), 1182-1183.

Cao, Y., Chen, Z., Zhu, J., Yue, P., Lin, C. Y., & Yu, Y. (2011, July). Leveraging unlabeled data to scale blocking for record linkage. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence* (Vol. 22, No. 3, p. 2211).

Carr, R. D., Doddi, S., Konjevod, G., & Marathe, M. V. (2000, January). On the red-blue set cover problem. In *SODA* (pp. 345-353).

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004, May). Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (pp. 74-83). ACM.

Chakrabarti, K., Chaudhuri, S., Cheng, T., & Xin, D. (2012, August). A framework for robust discovery of entity synonyms. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1384-1392). ACM.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 27.

Chaudhuri, S., Ganti, V., & Motwani, R. (2005, April). Robust identification of fuzzy duplicates. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (pp. 865-876). IEEE.

Chen, P. P. S. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, *1*(1), 9-36.

Christen, P. (2008, August). Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1065-1068). ACM.

Christen, P. (2008, August). Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 151-159). ACM.

Christen, P. (2012). *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media.

Christen, P. (2012). A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*,*24*(9), 1537-1555.

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. Mathematics of operations research, 4(3), 233-235.

Codd, E. F. (1970). A relational model of data for large shared data banks.*Communications of the ACM*, *13*(6), 377-387.

Cohen, W. W. (2000). Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, *18*(3), 288-321.

Cohen, W. W., Kautz, H., & McAllester, D. (2000, August). Hardening soft information sources. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 255-259). ACM.

Cucerzan, S. (2007, June). Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *EMNLP-CoNLL* (Vol. 7, pp. 708-716).

Das Sarma, A., He, Y., & Chaudhuri, S. (2014). Clusterjoin: a similarity joins framework using map-reduce. *Proceedings of the VLDB Endowment*, *7*(12), 1059-1070.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004, June). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry* (pp. 253-262). ACM.

Date, C. J., & Darwen, H. (1993). *A guide to the SQL Standard: a user's guide to the standard relational language SQL* (Vol. 55822). Addison-Wesley Longman.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1-38.

Dietze, S., Sanchez-Alonso, S., Ebner, H., Qing Yu, H., Giordano, D., Marenzi, I., & Pereira Nunes, B. (2013). Interlinking educational resources and the web of data: A survey of challenges and approaches. *Program*, *47*(1), 60-91.

Doan, A., Halevy, A., & Ives, Z. (2012). *Principles of data integration*. Elsevier.

Dong, X. L., & Srivastava, D. (2013, April). Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (pp. 1245-1248). IEEE.

Dredze, M., McNamee, P., Rao, D., Gerber, A., & Finin, T. (2010, August). Entity disambiguation for knowledge base population. In *Proceedings of the 23rd International Conference on Computational Linguistics* (pp. 277-285). Association for Computational Linguistics.

Duan, S., Fokoue, A., Hassanzadeh, O., Kementsietsidis, A., Srinivas, K., & Ward, M. J. (2012). Instance-based matching of large ontologies using locality-sensitive hashing. In *The Semantic Web–ISWC 2012* (pp. 49-64). Springer Berlin Heidelberg.

Elfeky, M. G., Verykios, V. S., & Elmagarmid, A. K. (2002). TAILOR: A record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (pp. 17-28). IEEE.

Elmagarmid, A. K., Ipeirotis, P. G., & Verykios, V. S. (2007). Duplicate record detection: A survey. Knowledge and Data Engineering, IEEE Transactions on, 19(1), 1-16.

Euzenat, J., & Shvaiko, P. (2007). *Ontology matching* (Vol. 333). Heidelberg: Springer.

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, *64*(328), 1183-1210.

Ferrara, A., Lorusso, D., Montanelli, S., & Varese, G. (2008, October). Towards a benchmark for instance matching. In The 7th International Semantic Web Conference (p. 37).

Ferrara, A., Montanelli, S., Noessner, J., & Stuckenschmidt, H. (2011). Benchmarking matching applications on the semantic web. In The Semantic Web: Research and Applications (pp. 108-122). Springer Berlin Heidelberg.

Ferrara, A., Nikolov, A., Noessner, J., & Scharffe, F. (2013). Evaluation of instance matching tools: The experience of OAEI. *Web semantics: Science, services and agents on the World Wide Web*, *21*, 49-60.

Ferrara, A., Nikolov, A., & Scharffe, F. (2013). Data linking for the semantic web. Semantic Web: Ontology and Knowledge Base Enabled Tools, Services, and Applications, 169.

Getoor, L., & Machanavajjhala, A. (2013, August). Entity resolution for big data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1527-1527). ACM.

Gropp, W., Lusk, E., Doss, N., & Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard.*Parallel computing*, *22*(6), 789-828.

Gusfield, D., & Irving, R. W. (1989). *The stable marriage problem: structure and algorithms*. MIT press.

Halevy, A., Rajaraman, A., & Ordille, J. (2006, September). Data integration: the teenage years. In Proceedings of the 32nd international conference on Very large data bases (pp. 9-16). VLDB Endowment.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, *11*(1), 10-18.

Han, S. N., Lee, G. M., & Crespi, N. (2014). Semantic context-aware service composition for building automation system. Industrial Informatics, IEEE Transactions on, 10(1), 752-761.

Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, *143*(1), 29-36.

Hernández, M. A., & Stolfo, S. J. (1995, June). The merge/purge problem for large databases. In ACM SIGMOD Record (Vol. 24, No. 2, pp. 127-138). ACM.

Hernández, M. A., & Stolfo, S. J. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. Data mining and knowledge discovery, 2(1), 9-37.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). A practical guide to support vector classification.

Hu, W., Qu, Y. Z., & Sun, X. Z. (2011). Bootstrapping object coreferencing on the semantic web. *Journal of Computer Science and Technology*, *26*(4), 663-675.

Isele, R., Jentzsch, A., & Bizer, C. (2011, June). Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*.

Jaffri, A., Glaser, H., & Millard, I. (2008). Uri disambiguation in the context of linked data.

Jean-Mary, Y. R., Shironoshita, E. P., & Kabuka, M. R. (2010). Asmov: Results for oaei 2010. *Ontology Matching*, *126*.

Jeffery, S. R., Franklin, M. J., & Halevy, A. Y. (2008, June). Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 847-860). ACM.

Jiménez-Ruiz, E., & Grau, B. C. (2011). Logmap: Logic-based and scalable ontology matching. In *The Semantic Web–ISWC 2011* (pp. 273-288). Springer Berlin Heidelberg.

Joachims, T. (1999). Making large scale SVM learning practical. Universität Dortmund.

Kejriwal, M., & Miranker, D. P. (2013, December). An unsupervised algorithm for learning blocking schemes. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on* (pp. 340-349). IEEE.

Kejriwal, M., & Miranker, D. P. (2014). A two-step blocking scheme learner for scalable link discovery. *Ontology Matching*, *49*.

Kejriwal, M., & Miranker, D. P. (2015). A DNF Blocking Scheme Learner for Heterogeneous Datasets. *arXiv preprint arXiv:1501.01694*.

Kejriwal, M., & Miranker, D. P. (2015). Semi-supervised Instance Matching Using Boosted Classifiers. In *The Semantic Web. Latest Advances and New Domains* (pp. 388-402). Springer International Publishing.

Kejriwal, M., & Miranker, D. P. (2015). An Unsupervised Instance Matcher for Schema-free RDF Data. *Web Semantics: Science, Services and Agents on the World Wide Web*.

Kejriwal, M., & Miranker, D. P (2015). Sorted Neighborhood for Schema-free RDF Data.

Kim, H. S., & Lee, D. (2010, March). HARRA: fast iterative hashed record linkage for large-scale data collections. In *Proceedings of the 13th International Conference on Extending Database Technology* (pp. 525-536). ACM.

Kirsten, T., Kolb, L., Hartung, M., Groß, A., Köpcke, H., & Rahm, E. (2010). Data partitioning for parallel entity matching. *arXiv preprint arXiv:1006.5309*.

Klyne, G., & Carroll, J. J. (2006). Resource description framework (RDF): Concepts and abstract syntax.

Kolb, L., Thor, A., & Rahm, E. (2012). Multi-pass sorted neighborhood blocking with MapReduce. Computer Science-Research and Development, 27(1), 45-63.

Kolb, L., Thor, A., & Rahm, E. (2012). Dedoop: efficient deduplication with Hadoop. *Proceedings of the VLDB Endowment*, *5*(12), 1878-1881.

Köpcke, H., Thor, A., & Rahm, E. (2010). Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, *3*(1-2), 484-493.

Köpcke, H., & Rahm, E. (2010). Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, *69*(2), 197-210.

Lenzerini, M. (2002, June). Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (pp. 233-246). ACM.

Leonardi, E., Abel, F., Heckmann, D., Herder, E., Hidders, J., & Houben, G. J. (2010). *A flexible rule-based method for interlinking, integrating, and enriching user data* (pp. 322-336). Springer Berlin Heidelberg.

Li, J., Tang, J., Li, Y., & Luo, Q. (2009). Rimom: A dynamic multistrategy ontology alignment framework. *Knowledge and Data Engineering, IEEE Transactions on*, *21*(8), 1218-1232.

Ma, Y. (2014, June). Effective Instance Matching for Heterogeneous Structured Data.

Ma, Y., Tran, T., & Bicer, V. (2013, April). Typifier: Inferring the type semantics of structured data. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (pp. 206-217). IEEE.

Ma, K., & Yang, B. (2015, September). Parallel NoSQL Entity Resolution Approach with Mapreduce. In *Intelligent Networking and Collaborative Systems (INCOS), 2015 International Conference on* (pp. 384-389). IEEE.

McCallum, A., Nigam, K., & Ungar, L. H. (2000, August). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM*

SIGKDD international conference on Knowledge discovery and data mining (pp. 169-178). ACM.

McCarthy, J. F., & Lehnert, W. G. (1995). Using decision trees for coreference resolution. arXiv preprint cmp-lg/9505043.

McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C recommendation*, *10*(10), 2004.

Menestrina, D., Whang, S. E., & Garcia-Molina, H. (2010). Evaluating entity resolution results. *Proceedings of the VLDB Endowment*, *3*(1-2), 208-219.

Moro, A., Raganato, A., & Navigli, R. (2014). Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, *2*, 231-244.

Mesnil, G., Dauphin, Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I. J., ... & Vincent, P. (2012). Unsupervised and Transfer Learning Challenge: a Deep Learning Approach. ICML Unsupervised and Transfer Learning, 27, 97-110.

Metwally, A., & Faloutsos, C. (2012). V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *Proceedings of the VLDB Endowment*, *5*(8), 704-715.

Michelson, M., & Knoblock, C. A. (2006, July). Learning blocking schemes for record linkage. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 21, No. 1, p. 440). Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*,*5*(1), 32-38.

Newcombe, H. B., Kennedy, J. M., Axford, S. J., & James, A. P. (1959). Automatic Linkage of Vital Records Computers can be used to extract" follow-up" statistics of families from files of routine records. *Science*, *130*(3381), 954-959.

Ngomo, A. C. N. (2011). A time-efficient hybrid approach to link discovery. *Ontology Matching*, 1.

Ngomo, A. C. N., & Auer, S. (2011). Limes-a time-efficient approach for large-scale link discovery on the web of data. *integration*, *15*, 3.

Ngomo, A. C. N., Lehmann, J., Auer, S., & Höffner, K. (2011, October). Raven–active learning of link specifications. In *Proceedings of the Sixth International Workshop on Ontology Matching* (pp. 25-37).

Ngomo, A. C. N., & Lyko, K. (2012). Eagle: Efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications* (pp. 149-163). Springer Berlin Heidelberg.

Ngomo, A. C. N., & Lyko, K. (2013, October). Unsupervised learning of link specifications: deterministic vs. non-deterministic. In *OM* (pp. 25-36).

Ngomo, A. C. N., Lyko, K., & Christen, V. (2013). Coala–correlation-aware active learning of link specifications. In *The Semantic Web: Semantics and Big Data* (pp. 442-456). Springer Berlin Heidelberg.

Nikolov, A., Uren, V., Motta, E., & De Roeck, A. (2008). Integration of semantically annotated data by the KnoFuss architecture. In *Knowledge Engineering: Practice and Patterns* (pp. 265-274). Springer Berlin Heidelberg.

Nikolov, A., Uren, V., Motta, E., & De Roeck, A. (2009). Towards Data Fusion in a multi-ontology Environment.

Niu, X., Rong, S., Wang, H., & Yu, Y. (2012, October). An effective rule miner for instance matching in a web of data. In *Proceedings of the 21st ACM international conference on Information and knowledge management* (pp. 1085-1094). ACM.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, *22*(10), 1345-1359.

Papadakis, G., Demartini, G., Fankhauser, P., & Kärger, P. (2010, November). The missing links: Discovering hidden same-as links among a billion of triples. In Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (pp. 453-460). ACM.

Papadakis, G., Ioannou, E., Palpanas, T., Niederée, C., & Nejdl, W. (2013). A blocking framework for entity resolution in highly heterogeneous information spaces. *Knowledge and Data Engineering, IEEE Transactions on*, *25*(12), 2665-2682.

Peleg, D. (2007). Approximation algorithms for the label-cover max and red-blue set cover problems. Journal of Discrete Algorithms, 5(1), 55-64.

Puhlmann, S., Weis, M., & Naumann, F. (2006). XML duplicate detection using sorted neighborhoods. In Advances in Database Technology-EDBT 2006 (pp. 773-791). Springer Berlin Heidelberg.

Pujara, J., Miao, H., Getoor, L., & Cohen, W. (2013). Knowledge graph identification. In The Semantic Web–ISWC 2013 (pp. 542-557). Springer Berlin Heidelberg.

Quilitz, B., & Leser, U. (2008). *Querying distributed RDF data sources with SPARQL* (pp. 524-538). Springer Berlin Heidelberg.

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *the VLDB Journal*, *10*(4), 334-350.

Raimond, Y., Sutton, C., & Sandler, M. B. (2008). Automatic Interlinking of Music Datasets on the Semantic Web. *LDOW*, *369*.

Ravikumar, P., & Cohen, W. W. (2004, July). A hierarchical graphical model for record linkage. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence* (pp. 454-461). AUAI Press.

Raz, R., & Safra, S. (1997, May). A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (pp. 475-484). ACM.

Rong, S., Niu, X., Xiang, E. W., Wang, H., Yang, Q., & Yu, Y. (2012). A machine learning approach for instance matching based on similarity metrics. In *The Semantic Web–ISWC 2012* (pp. 460-475). Springer Berlin Heidelberg.

Sadosky, P., Shrivastava, A., Price, M., & Steorts, R. C. (2015). Blocking Methods Applied to Casualty Records from the Syrian Conflict. *arXiv preprint arXiv:1510.07714*.

Salton, G., & McGill, M. J. (1986). Introduction to modern information retrieval.

Sahoo, S. S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., ... & Ezzat, A. (2009). A survey of current approaches for mapping of relational databases to RDF. *W3C RDB2RDF Incubator Group Report*, 113-130.

Scharffe, F., Liu, Y., & Zhou, C. (2009). Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*.

Schmachtenberg, M., Bizer, C., & Paulheim, H. (2014). Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014* (pp. 245-260). Springer International Publishing.

Schroeder, B., & Gibson, G. (2010). A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, *7*(4), 337-350.

Sequeda, J. F., & Miranker, D. P. (2013). Ultrawrap: SPARQL execution on relational data. *Web Semantics: Science, Services and Agents on the World Wide Web*, *22*, 19-39.

Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, *52*(55-66), 11.

Shadbolt, N., O'Hara, K., Berners-Lee, T., Gibbins, N., Glaser, H., & Hall, W. (2012). Linked open government data: Lessons from data. gov. uk. *IEEE Intelligent Systems*, *27*(3), 16-24.

Song, D., & Heflin, J. (2011). Automatically generating data linkages using a domain-independent candidate selection approach. In *The Semantic Web–ISWC 2011* (pp. 649-664). Springer Berlin Heidelberg.

Soru, T., & Ngomo, A. C. N. (2014, September). A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th International Conference on Semantic Systems* (pp. 41-44). ACM.

Stephenson, C. (1980). The methodology of historical census record linkage: a users guide to the Soundex. Journal of Family History, 5(1), 112-115.

Stoilos, G., Simou, N., Stamou, G., & Kollias, S. (2006). Uncertainty and the semantic web. Intelligent Systems, IEEE, 21(5), 84-87.

Suchanek, F. M., Abiteboul, S., & Senellart, P. (2011). Paris: Probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment*, *5*(3), 157-168.

Tian, A., Kejriwal, M., & Miranker, D. P. (2014, June). Schema matching over relations, attributes, and data values. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*(p. 28). ACM.

Vernica, R., Carey, M. J., & Li, C. (2010, June). Efficient parallel set-similarity joins using MapReduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 495-506). ACM.

Volz, J., Bizer, C., Gaedke, M., & Kobilarov, G. (2009). Silk-A Link Discovery Framework for the Web of Data. *LDOW*, *538*.

Whang, S. E., Menestrina, D., Koutrika, G., Theobald, M., & Garcia-Molina, H. (2009, June). Entity resolution with iterative blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*(pp. 219-232). ACM.

White, T. (2012). *Hadoop: The definitive guide*. " O'Reilly Media, Inc.".

Winkler, W. E. (1993). Improved decision rules in the Fellegi-Sunter model of record linkage.

Winkler, W. E. (1999). The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*.

Winkler, W. E. (2002). *Methods for record linkage and bayesian networks*. Technical report, Statistical Research Division, US Census Bureau, Washington, DC.

Yan, S., Lee, D., Kan, M. Y., & Giles, L. C. (2007, June). Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries* (pp. 185-194). ACM.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010, June). Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Vol. 10, p. 10).

Zhai, C., & Lafferty, J. (2001, September). A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 334-342). ACM.

Zhu, X., & Goldberg, A. B. (2009). Introduction to semi-supervised learning.*Synthesis lectures on artificial intelligence and machine learning*, *3*(1), 1-130.