Populating a Linked Data-based Entity Name System

Mayank Kejriwal
University of Texas at Austin
kejriwal@cs.utexas.edu

Abstract. Data Integration is the problem of combining data in various data sources and providing a user with a unified view over these sources. Building an automatic data integration system that can process large, semi-structured data sources has emerged as an important problem. An automated data integration system requires automatic population of an Entity Name System (ENS). An ENS is a thesaurus for entities and is used to serve instance matching needs across data sources. Resource Description Framework (RDF) is a graph-based data model used to publish data on the Web as linked data. To build and populate an ENS over linked data, the fundamental problem of data matching needs to be solved. Traditionally, data matching concerned identifying pairs of logically equivalent entities across one or more structurally homogeneous data sources, and required a human in the loop. Additionally, most systems run on serial architectures.

These assumptions cannot be expected to hold for linked data. Given the size and variety of data sources on Linked Open Data, a growing repository of linked data published under an open license, a data matching system must be scalable and account for multiple forms of heterogeneity in an automated fashion. The thesis is that building such a data matching system requires simultaneously resolving the three challenges of scalability, heterogeneity and automation.

This dissertation proposal spans three pieces of work leading to a data matching system that will automatically link multi-domain semi-structured data sources in a shared-nothing parallel architecture. The first work addresses automation, and enables a key process in the data matching pipeline called blocking to be done using unsupervised learning, that is, without human involvement. The contribution is extended to implement a high-quality, fully unsupervised data matching system on structurally homogeneous relational database inputs. The second piece of work accommodates graph database inputs, thereby addressing heterogeneity. The third piece of work addresses scalability. Proposed is a fully unsupervised heterogeneous pipeline implemented in a MapReduce-based framework, and with strong run-time bounds. The pipeline is also extended to accommodate multiple domains.

The final prototype will be implemented on cloud infrastructure and evaluated on real-world test suites already published as Linked Open Data. The prototype will then be used to populate an Entity Name System to serve instance matching needs in an actual data integration system designed for the Semantic Web.
# Table of Contents

1 Introduction ................................................................. 3
2 Background ................................................................. 6
  2.1 Data Models ........................................................... 6
  2.2 Two-Step Data Matching ............................................. 8
  2.3 Automation, Heterogeneity and Scalability ....................... 17
  2.4 Concluding Note .................................................... 21
3 Completed Work ........................................................... 22
  3.1 Contribution 1: An Unsupervised DNF-BSL ..................... 23
  3.2 Contribution 2: A Heterogeneous Unsupervised DNF-BSL ....... 26
  3.3 Contribution 3: Fully Unsupervised Tabular Data Matching .... 29
4 Proposed Work ............................................................. 31
  4.1 Proposed Contribution 1: A Heterogeneous Data Matching Workflow ............................................ 31
  4.2 Proposed Contribution 2: The Multi-Domain Problem ......... 32
  4.3 Proposed Contribution 3: Generic Link-Discovery (Optional) .. 32
  4.4 Deliverable ........................................................... 33
5 Work Plan ................................................................. 34

Appendices ................................................................. 45
A An Unsupervised Algorithm for Learning Blocking Schemes ....... 45
  A.1 Introduction ......................................................... 45
  A.2 Blocking Schemes ................................................... 46
  A.3 Generating Weakly Labeled Training Set ......................... 51
  A.4 Feature Selection .................................................. 54
  A.5 Experiments ........................................................ 57
  A.6 Results and Discussion ............................................ 59
  A.7 Related Work ....................................................... 61
  A.8 Future Work and Conclusion ..................................... 62
B A DNF Blocking Scheme Learner for Heterogeneous Datasets ....... 64
  B.1 Introduction ......................................................... 64
  B.2 Related Work ....................................................... 66
  B.3 Preliminaries ....................................................... 67
  B.4 The generic pipeline ............................................... 70
  B.5 Property Table Representation ................................... 71
  B.6 Extending the formalism .......................................... 72
  B.7 An Unsupervised Instantiation .................................... 76
  B.8 Experiments ........................................................ 78
  B.9 Results and Discussion .......................................... 82
  B.10 Conclusion and Future Work ..................................... 87
Fig. 1. A high-level schematic of data integration

1 Introduction

Data Integration is the problem of combining data residing in multiple sources in order to provide a unified interface to an application or end user [24]. The unification is virtual and typically accomplished by means of wrappers, as shown in Figure 1. Data integration has numerous applications and continues to be an active area of research [24].

Resource Description Framework (RDF)\(^1\) is the data model of the Semantic Web that is used to publish semi-structured data as directed labeled graphs. An example is Dataset 1 in Figure 2. Nodes in the graph represent entities and edges represent properties connecting these entities.

The scope of data integration has grown in concert with the Semantic Web, and with it, the publishing of new data on the Web as linked data. For example, consider Linked Open Data\(^2\) (LOD), which is the collection of RDF datasets published under an open license [10]. According to a recently published study, the LOD cloud currently contains many billions of triples in over 1000 individually published datasets [76].

On the LOD cloud, two nodes may refer to the same logical entity, despite being syntactically disparate. As a simple example, the entity Mickey Beats in Dataset 1 in Figure 2 is represented by two syntactically different nodes. In the Semantic Web model, data matching is the problem of linking logically equivalent entities using a special owl:sameAs property\(^3\) [30]. The two mentions of Mickey Beats in Dataset 1 should be linked using the dotted sameAs edge. The data matching problem is not restricted to the RDF data model but can be stated abstractly as identifying and resolving semantically equivalent entities in one or more datasets [27]. As an example of data matching on relational data sources, consider Datasets 2 and 3 in Figure 2. In these datasets, an entity is represented

---

\(^1\) [http://www.w3.org/RDF/](http://www.w3.org/RDF/)

\(^2\) [linkeddata.org](http://linkeddata.org)

\(^3\) Hereafter, written simply as sameAs
Fig. 2. An instance of the heterogeneous data matching problem. We will use this figure as a running example throughout this document.

as a tuple. The goal is to identify duplicate tuples, that is, tuples referring to the same logical entity.

Data matching is an important problem that has been acknowledged as occurring in structured, semi-structured and even unstructured data models. Different communities refer to the problem using different names. Examples include entity resolution [3], deduplication [27], record linkage [26], co-reference resolution [57], instance matching, link discovery [30], the merge-purge problem [35], discovering entity synonyms [14], and hardening soft databases [22].

Data matching is a vital component of data integration, as it is required to populate an Entity Name System or ENS [11]. An ENS is a thesaurus for entities and is used to serve instance matching needs across source databases. Figure 3 shows an abstract schematic of an ENS. Correctly populating an ENS requires finding the sameAs edges (the ‘synonyms’) between logically equivalent entities.

A previous study had suggested that LOD contains many such syntactically disparate but logically equivalent entities that have not yet been inter-linked [67]. A recent study confirmed this and showed that, despite its growth, LOD continues to be quite sparse in triples that link datasets [76]. The vocabulary used by entities on LOD is also quite varied, with datasets spanning many different domains. Many datasets do not have associated meta-data.
These findings, together with the ongoing growth of LOD, demonstrate that LOD exhibits volume, variety and velocity. Populating an ENS in support of matching instances in LOD has emerged as a Big Data problem [55]. Solving the problem requires one to devise scalable data matching solutions that can deal with the heterogeneous nature (or variety) of linked data. Given the expense of domain expertise, a data matching solution should minimize supervision and be automated. Finally, the emergence of cloud services and dataspaces [39] implies that it would be an added boon for the system to be cloud-deployable, since it could then be accessed as a service over the Internet. Such a service would benefit efforts besides data integration that also require data matching as a vital first step. Two examples of such efforts are semantic search [11] and knowledge graph identification [70].

In Section 2, we detail the data matching workflow and the status of the current state-of-the-art in meeting the motivated requirements of heterogeneity, scalability and automation. We will show that no current system meets all three requirements simultaneously.

This dissertation research will develop and implement algorithms for building a fully unsupervised data matching prototype. The key components will be implemented in the MapReduce framework [23], which is a shared-nothing parallel paradigm that has numerous documented advantages and is especially suited for cloud deployment. In the general case, the implemented prototype will accept a set of multi-domain heterogeneous databases as input, and will output a set of matched entities. The matched entities may then be used by subsequent applications. In the dissertation, the specific application will be the population of
an ENS, which will then be integrated into an existing data integration system designed specifically for the Semantic Web.

Additionally, experiments will show that the data matching system can also be used to support relational database inputs with slight modifications. Thus, the applicability of the proposed techniques is not limited to the RDF data model but may also be used to support ENS applications in the Deep Web, which is the collection of back-end relational databases powering Web queries and faceted search [34]. The Deep Web is an important application because it has shown super-linear growth and, according to a study, is at least 500 times greater than the Surface Web [34].

All proposed algorithms will be evaluated on established benchmarks, as well as new real-world datasets published already as Linked Open Data. The prototype will be implemented on 32 HDInsight\(^4\) nodes in the Microsoft Windows Azure cloud infrastructure.

2 Background

2.1 Data Models

In this dissertation, the primary data model of concern is the Resource Description Framework (RDF) model. The relational model is also important for historical reasons, given that much of the data matching literature has traditionally been confined to the relational database community. Both models are described below before providing background on data matching.

**Resource Description Framework (RDF)** An RDF data source is a graph-based model that is formally described by a set of triples, with a triple representing an edge and being a 3-tuple of the form \((\text{subject, property, object})\). Subjects and properties are necessarily Information Resource Identifiers (IRIs), but an object may either be an IRI or a literal. We note that, in practice, IRI elements in the Semantic Web are associated with a name or label property which can be dereferenced and used to mnemonically refer to that element. For example, in Dataset 1 in Figure 2, the subject and property elements are informally referred to by their names and not the IRIs themselves. Dataset 1 in Figure 2 shows that an RDF dataset can be equivalently represented as a directed labeled graph \(G = (V, E)\). The set \(V\) is simply the union of the set of subjects and objects that occur in the set of triples, and for each triple in the set, there is an edge.

\(^4\) [http://azure.microsoft.com/en-us/services/hdinsight/](http://azure.microsoft.com/en-us/services/hdinsight/). These nodes are designed to facilitate Apache Hadoop (an open-source MapReduce implementation) as a service

\(^5\) Note that predicate is sometimes used in place of property; we uniformly use property in this paper, to avoid confusion

\(^6\) IRIs are technically more general than URIs (Uniform Resource Identifiers), but in practice, URIs are almost always employed. In this document, both are interchangeable
RDF is the data model used for linked data, but in the general Semantic Web technology stack, it has also become the basis for representing RDF Schema (RDFS)\(^7\) and for the Web Ontology Language (OWL).\(^8\) The RDFS standard is described as a semantic extension of RDF. It provides a convenient data modeling vocabulary that can be used to publish a schema (comprising classes and properties) for an RDF dataset. OWL is a semantic markup language that is primarily used for publishing and sharing ontologies on the World Wide Web. OWL also provides reasoning capabilities over ontologies.

The full details of both RDFS and OWL are beyond the scope of this dissertation, as we are mainly concerned with the RDF data itself, and not its associated meta-data. One reason is that in the Semantic Web universe, meta-data is optional. In Linked Open Data, both missing and shallow schemas are known to be common.\(^76\) We do note, however, that OWL provides a convenient vocabulary for operationalizing the data matching task in the Semantic Web, via the owl:sameAs property. In other words, when we discover that two entities are semantically equivalent, we can simply publish a new triple connecting the two nodes with an owl:sameAs property. This property is pre-defined to have RST\(^9\) or equivalence class semantics, which can be further exploited by query answering and reasoning systems. From the perspective of a practitioner, easy operationalizing implies that the problem of entity resolution is no more difficult than the problem of entity identification.

Relational Model

The relational data model is formally based on first-order logic.\(^73\) The model is tabular, and is accompanied by its own relational algebra that can be used to define constraints and keys on an instance conforming to the model. The algebra also provides for operators that form the underlying basis for expressive languages (like SQL) for querying the instance.

Formally, define the schema \(S'\) of a relational database as a set of relation names, with each name associated with a list of attributes or fields. A relational instance \(S\) associates with each relation name \(R' \in S'\), a set \(R\) of records or tuples. For shorthand, we henceforth refer to \(R\) as a table.

Example 1. In Figure 2, Datasets 2 and 3 are both tables. Dataset 2 has schema Emergency Contact(Name, Contact, Relation). The first tuple of the Dataset 2 instance has field values Mickey Beats, Joan Beats and Spouse respectively. The keyword null is reserved.

Finally, in the vein of Elmagarmid et al. [27], define a pair of tables to be structurally homogeneous if their underlying fields are identical\(^10\), otherwise define them to be structurally heterogeneous. The next section will show that in the relational data matching community, the structural homogeneity assumption is

\(^7\) http://www.w3.org/TR/rdf-schema/
\(^8\) http://www.w3.org/2001/sw/wiki/OWL
\(^9\) Reflexive, Symmetric and Transitive
\(^10\) It is incorrect to say that the schemas are identical, since two tables will always have distinct names
pervasive. This implies that relational database techniques cannot be naturally adapted to data matching in the Semantic Web, since the semi-structured nature of RDF renders the structural homogeneity assumption problematic. We address this issue in the dissertation as fulfilling a heterogeneity requirement.

2.2 Two-Step Data Matching

In this dissertation, the commonly used two-step workflow for data matching is assumed [27]. A high-level overview is shown in Figure 4. In this workflow, a preprocessing step called blocking is used to cluster entities in the datasets into overlapping blocks [17]. Entities within each block are paired and compared in the second step using a sophisticated similarity function. The main goal of blocking is to address the quadratic cost of brute-force pairwise comparisons of all $n$ entities in the input datasets. We describe each of the steps in detail in this section; in a following section, we evaluate existing data matching systems conforming to this workflow from the criteria-specific perspective of scalability, heterogeneity and automation. We note that, in much of the literature, the two-step workflow remains dominant [16]; alternatives that interleave blocking and classification make restrictive assumptions that are unsuitable for generic domain-independent systems [45].

An additional point to note is that prevailing data matching techniques in the relational setting overwhelmingly assume that either one relational instance is provided as input or, in the case of more than one relational instance, the instances are structurally homogeneous [27]. Unless otherwise stated, the structural homogeneity assumption is expected to hold for the relational data matching background provided in this chapter.

**Blocking** Assume a set of $n$ entities, across one of more datasets. Even given a similarity function that determines whether an entity pair is duplicate or non-duplicate, a naive approach would pair each entity with every other entity and evaluate each of the $\Theta(n^2)$ pairs using the function. Such quadratic complexity is not tenable, even for in-memory databases. As an example, consider a simple relational dataset comprising just 2000 tuples. The naive approach would require
2000*1999/2 \approx 2 \text{ million comparisons. The number of comparisons grows quite rapidly due to the quadratic nature of the problem.}

Blocking was proposed precisely to overcome this limitation [17]. A blocking scheme, which we shall precisely define, is used by a blocking method in the first step to cluster entities into blocks. In the classification step, only entities within blocks are paired and subject to comparison. Since an entity can be placed in more than one block, blocks are overlapping. The goal is to place entities in blocks in such a way that all duplicate entities share at least one block. A good blocking scheme and blocking method should together have a high reduction ratio, so that the resulting data matching workflow has near-linear cost compared to the naive quadratic procedure. However, this should not come at the expense of coverage of true duplicates. In the literature, (and also in Section A.2) these metrics are well-defined quantitatively and are extensively used for measuring blocking quality [26].

In the next section, we introduce an important class of blocking schemes, called Disjunctive Normal Form (DNF) blocking schemes. These schemes were formally introduced relatively recently, and have some important advantages [5], [60]. DNF blocking schemes play a key role in the developments in this thesis.

**DNF Blocking Schemes** DNF blocking schemes are an important class of blocking schemes that have demonstrated excellent empirical performance in multiple research efforts over the past decade [5], [60], [12]. They can also be learned using machine learning techniques. We model the formalism below after the one proposed originally by Bilenko et al. [5].

The most basic elements of a blocking scheme are indexing functions $h_i(x_t)$ [5]. An indexing function accepts a field value from a tuple as input and returns a set $Y$ that contains 0 or more blocking key values (BKVs). A BKV identifies a block in which the tuple is placed. Intuitively, one may think of a block as a hash bucket, except that blocking is many-to-many while hashing is typically many-to-one [17]. For example, if $Y$ contains multiple BKVs, a tuple is placed in multiple blocks.

**Definition 1.** Given a field value $x_t$ from some tuple $t$ and an alphabet $\Sigma$, an indexing function $h_i : \text{Domain}(h_i) \rightarrow \Sigma^*$ takes $x_t$ as input and returns a set $Y$ that contains 0 or more Blocking Key Values (BKVs) from the set of all possible BKVs $\Sigma^*$.

The domain $\text{Domain}(h_i)$ is usually just the string datatype. The range is a set of BKVs assigned to the tuple. Each BKV is represented by a string identifier.

**Example 2.** An example of an indexing function is Tokens. When applied to the Last Name field value of the fourth tuple in Dataset 3, the output set $Y$ is \{W., Beats, Jr.\}.

This leads to the notion of a general blocking predicate (GBP).

**Definition 2.** Given field values $x_{t_1}$ and $x_{t_2}$ from two tuples, $t_1$ and $t_2$, and an indexing function $h_i$, a general blocking predicate (GBP) $p_i : \text{Domain}(h_i) \times$
Domain($h_i$) → {True, False} is a boolean function that takes $x_{t_1}$ and $x_{t_2}$ as input returns True iff $h_i(x_{t_1}) \cap h_i(x_{t_2}) \neq \Phi$, and returns False otherwise.

Intuitively, a GBP returns True iff $Y_1$ and $Y_2$ share elements, or equivalently, if $t_1$ and $t_2$ have a block in common. Each GBP $p_i$ is always associated with an indexing function $h_i$.

Example 3. Consider the GBP ContainsCommonToken, associated with the previously introduced Tokens. Suppose it was input the Last Name field values from the first and fourth tuples in Dataset 3. Since these field values have a token Beats in common, the GBP returns True.

A specific blocking predicate (SBP) explicitly pairs a GBP to a specific field.

**Definition 3.** Given two tuples $t_1$ and $t_2$, a field $f$ and a general blocking predicate $p_i$, a specific blocking predicate (SBP) is a function characterized by the pair ($p_i$, $f$). An SBP takes $t_1$ and $t_2$ as inputs and applies $p_i$ to the field values from both tuples corresponding to $f$. A tuple pair is said to be covered if the SBP returns True for that pair.

Previous DNF research assumed that all available GBPs can be applied to all fields of the relation [42], [5], [12], [60]. Hence, given a relational instance $R$ with $m$ fields in its schema$^{11}$, and $s$ GBPs, the number of SBPs is exactly $ms$.

Finally, a DNF blocking scheme is defined as:

**Definition 4.** Given a set $H$ of SBPs, a DNF blocking scheme $f_P$ is a positive propositional formula constructed in Disjunctive Normal Form$^{12}$ (DNF), using $H$ as the set of atoms. Additionally, if each term is constrained to comprise at most one atom, the blocking scheme is referred to as disjunctive.

SBPs cannot be negated, since the DNF scheme is a positive formula. A tuple pair is said to be covered if the blocking scheme returns True for that pair. Intuitively, this means that the two constituent tuples share a block. Since blocking is just a pre-processing step, both duplicate and non-duplicate tuple pairs can end up getting covered.

Example 4. Consider the disjunctive scheme (ContainsCommonToken, Last Name) ∨ (SameFirstDigit, Zip), applied on Dataset 3. While the two tuples referring to Mickey Beats would share a block (with the BKV Beats), the non-duplicate tuples referring to Susan and Samuel would also share a block (with the BKV 6). Note also that the first and fourth tuples share more than one block, since they also have BKV 7 in common.

---

$^{11}$ Structural homogeneity implies exactly one set of fields, even if there are multiple relational instances

$^{12}$ A disjunction of terms, where each term is a conjunction of literals
Learning DNF Blocking Schemes Before 2006, it was typically assumed that a blocking scheme would be provided by a domain expert [17]. Given the expense of domain expertise and the rising popularity of machine learning techniques in the data matching community at large, two DNF Blocking Scheme Learners (DNF-BSLs) were proposed in 2006 by Bilenko et al. [5] and Michelson and Knoblock respectively [60]. These initial DNF-BSLs were supervised and required training sets of duplicate and non-duplicate entity pairs in order to learn a DNF blocking scheme. In 2011, Cao et al. detailed a semi-supervised technique, which also required a training set, but could leverage unlabeled entity pairs in the dataset to enhance performance [12]. In 2013, we published the first unsupervised technique for learning DNF blocking schemes [42]. We describe this as a thesis contribution in Section 3.1.

We present the core principles here as background, since the proposed DNF-BSL shares many similarities with the three earlier systems that required supervision [5], [60],[12].

Table 1. DNF-BSL Systems for Structurally Homogeneous Tables

<table>
<thead>
<tr>
<th>ID</th>
<th>System</th>
<th>Parameters</th>
<th>Supervision</th>
<th>Approx. Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michelson and Knoblock [60]</td>
<td>$\epsilon, k$</td>
<td>Supervised</td>
<td>Sequential SC</td>
</tr>
<tr>
<td>2</td>
<td>Bilenko et al. [5]</td>
<td>$\epsilon, \eta, k$</td>
<td>Supervised</td>
<td>Red-Blue SC</td>
</tr>
<tr>
<td>3</td>
<td>Cao et al. [12]</td>
<td>$s, \epsilon, \tau, \alpha, k$</td>
<td>Semi-supervised</td>
<td>Sequential SC</td>
</tr>
<tr>
<td>4</td>
<td>Kejriwal and Miranker [42], Section 3.1</td>
<td>Generator: $\epsilon, ut, lt, d, nd$ Learner: $\epsilon, \eta, k$</td>
<td>Unsupervised</td>
<td>Greedy Algorithm + Fisher Discrimination</td>
</tr>
</tbody>
</table>

In effect, all four existing DNF-BSLs (including the contributed work) attempt to solve the optimization problem first formalized by Bilenko et al.:

$$f^* = \arg\min_f \sum_{\{(r,s)\} \in N} f(r, s)$$ (1)

s.t.

$$\sum_{\{(r,s)\} \in D} f^*(r, s) \geq |D|\epsilon$$ (2)

Here, $D$ and $N$ are sets of duplicate and non-duplicate pairs respectively and are assumed to be provided as training sets. Given a set of SBPs $H$, let $f$ be a blocking scheme from the space $F$ of all possible DNF blocking schemes. $F$ provably has cardinality $2^{2^{|H|}}$, since each DNF scheme is merely a positive DNF formula $f$ constructed by treating $H$ as the set of atoms (Definition 4). $\epsilon$ is a coverage parameter. Only schemes that cover at least a fraction $\epsilon$ of the set $D$ are included in the search space of the above optimization. The optimal blocking
scheme $f^* \in \mathcal{F}$ satisfies the objective by minimizing coverage of non-duplicates in $N$.

Unfortunately, optimally solving the problem above is NP-hard, as Bilenko et al. proved [5] by reducing from a known NP-hard problem called Red-Blue Set Covering [13]. For the proof, we refer the reader to the original paper [5].

All four DNF-BSLs attempt to solve this NP-hard problem by devising their own approximation algorithms. In order to tractably approximate a solution to the optimization above, assume a set of GBPs $G$ instead of SBPs $H$. The core of all approximation algorithms would first construct an abstract search space of SBPs $H$ by forming the cross product of $G$ and the set $A$ of fields. The goal is to choose a subset $H' \subseteq H$ such that the optimization condition above is approximately satisfied. The disjunction of SBPs in $H'$ is the disjunctive blocking scheme output by the DNF-BSL. Unsurprisingly, $\epsilon$ is a parameter common to all four systems in Table 1.

In order to learn a DNF scheme (as opposed to merely disjunctive), a parameter $k$ (also common to all four systems) is required. Michelson and Knoblock referred to this parameter as the beam search parameter, since their approximation algorithm involved sequential set covering and beam searching [60].

Other papers on the subject used $k$ in a similar manner, although their algorithms were not modeled after sequential set covering. In general, $k$ was used to supplement the original set $H$ with terms, to obtain a new set, $H_c$. This combinatorial process is demonstrated in Figure 5. To reflect the general application of $k$ in controlling search complexity and making the learning process tractable, $k$ is henceforth designated as the combinatorial parameter.

$H$ originally consists of the SBPs $a$, $b$, and $c$. These SBPs cover some tuple pairs (TPs). Suppose $k = 2$. A term of size 2 is formed by checking if any TP is covered by (at least) two SBPs. For example, TP-3 is covered by SBPs $a$ and $b$, and hence, also covered by the term $a \land b$. For $k > 2$, terms from size 2 to size $k$ are recursively added to $H$; the final supplemented set is denoted as $H_c$. Note that for $|H|$ predicates, building $H_c$ takes $O(|H|^k)$ time per TP. Given the exponential dependence on $k$ and diminishing returns, previous results capped $k$ at 2 [5]. If $k = 1$, $H_c = H$.

---

13 $\epsilon$ was designated as min thresh in the original System 1 paper [60], and $\sigma$ in the original System 3 paper [12].
The set $H' \subseteq H_c$ now chosen by the approximation scheme would potentially consist of terms and SBPs, with their disjunction yielding a k-DNF scheme, where a k-DNF formula is defined as having at most k literals in each term.

While System 1 only requires $\epsilon$ and $k$ as its parameters, Systems 2 and 4 prune their search spaces by removing all SBPs and terms from $H_c$ that cover more than $\eta |N|$ non-duplicates. Note that this step heuristically improves both quality and run-time. It comes at the risk of failure, since if the search space is pruned excessively (by high $\eta$), it may become impossible to cover at least $\epsilon |D|$ duplicates. Details on the remaining System 4 parameters appear in Section 3.1.

Blocking Methods Assume that we have a blocking scheme, either learned or provided. An interesting variety of methods can be used to apply the blocking scheme to the entities in the dataset. Christen provides a comprehensive survey of these methods [17]. These methods (and their variants) provide their own tradeoffs; we review two popular methods that continue to be widely used [17].

Traditional blocking is a technique akin to hashing. It has been in use since the 1960’s [29]. In its original version, traditional blocking explicitly assumed that a blocking scheme assigned exactly one BKV to each entity. This meant that, like hash buckets, blocks were non-overlapping. An extended version of traditional blocking is more commonly in use now, where blocking schemes may assign multiple BKVs to a given entity [16].

**Example 5.** To efficiently apply the blocking scheme $(\text{ContainsCommonToken, Last Name}) \lor (\text{SameFirstDigit, Zip})$ on each individual tuple in Dataset 3 (Figure 2), tokens from the field value corresponding to field Last Name are extracted, along with the first character from the field value of the Zip field, to obtain the tuple’s set of BKVs. For example, applied to the first tuple of Dataset 3, the BKV set $\{\text{Beats, 7}\}$ is extracted. An index is maintained, with the BKVs as keys and tuple pointers as values. With $n$ tuples, traditional blocking computes the blocks in time $\Theta(n)$ [17].

Let the set of generated blocks be $\Pi$. $\Pi$ contains sets of the form $B_v$, where $B_v$ is the block referred to by the BKV $v$. The candidate set of pairs $\Gamma$ is given below:

$$\Gamma = \bigcup_{B_v \in \Pi} \{(r, s) \mid \forall r, s \in B_v | r \neq s\}$$

(3)

$\Gamma$ is precisely the set input to the classification step of data matching, which classifies each pair as a duplicate, non-duplicate or probable duplicate as we will describe shortly [16]. Explicitly computing $\Gamma$ as a set eliminates redundancies that may be caused by overlapping blocks. Example 5 demonstrated how this could occur. If $\Gamma$ is too large, other techniques may have to be employed. Parallel and distributed data matching remain active areas of research [54].

On the other hand, the Sorted Neighborhood (SN) blocking method proposed by Hernández and Stolfo [35] does not use a hashing technique, but sorts the

---

14 Since $H_c$ now contains only a few, high-quality elements
Fig. 6. A timeline of research efforts in data matching, based on the survey by Elmagarmid et al. [27]

entire dataset using the BKVs as sorting keys. SN makes the same assumption as the original version of traditional blocking, that only one BKV is assigned to an entity. A window of constant size $w$ is slid over the sorted records, and all entity pairs within each window are generated and added to $\Gamma$, or passed along in a streaming fashion to the classification step. Hernández and Stolfo expanded the original work to incorporate a parallel implementation, and also showed that by using different blocking schemes in multiple passes, coverage can be further increased [36]. A small window size ($w < 10$) was typically found to be sufficient, even for large datasets. In the two decades following Sorted Neighborhood, many variants have been proposed and SN remains popular. Christen provides details on these variants in his survey [17]. Distributed and parallel implementations continue to be proposed; a recent work showed a multi-pass SN implementation in MapReduce [47].

**Classification Step** The second step accepts a set of entity pairs (either generated directly from the set of blocks $\Pi$ or as pairs in the candidate set $\Gamma$) and attempts to classify each pair as a duplicate, non-duplicate and in many cases, a probable duplicate [27]. Typically, if this three-category classification is adopted, duplicates and non-duplicates are classified with high confidence while probable duplicates are in need of clerical review [16].

Newcombe et al. were among the first to recognize data matching as a Bayesian inference problem [61]. Fellegi and Sunter formalized the notion, and even today, the Fellegi-Sunter model of data matching is widely adopted [29]. Elmagarmid et al. provide a comprehensive review of classification steps widely in use today [27], with more recent updates in the text by Christen [16]. A timeline of summarized data matching research is in Figure 6.

At a high-level, the classification step typically involves applying a *similarity function*, usually with associated parameters, on each entity pair. A similarity function can be quite complex and may employ a variety of *field matching* techniques, including phonetic techniques (e.g. Soundex, Double Metaphone), character-based techniques (Levenstein, Jaro-Winkler), token-based techniques (cosine similarity, Jaccard) and numeric functions. Several open-source libraries
exist that implement many of these functions, including FEBRL and Second-String [18], [20]. We list some important functions in Figure 7; surveys can be found in numerous texts [16].

For simplicity, assume a field-set $A$ and a set of field matching functions $V$. Similar to how we defined specific blocking predicates in the case of DNF blocking schemes (Definition 3), we can define a set of $|A||V|$ features. Assume that each feature, when applied to an entity pair, returns a real value. Each entity pair can then be represented as a feature vector with $|A||V|$ elements, as shown in Figure 8. The issue now becomes one of combining these features, using some set of operators, to yield a single real value given an entity pair. Coupled with two thresholds, the real value can be used to indicate if the pair is a duplicate, non-duplicate or probable duplicate [27], [29]. Equivalently, we can state the problem as classifying a given feature vector into one of these three categories.

Given the longevity of the data matching problem, many different classification techniques have been proposed, including rule-based techniques [83], distance-based techniques [21] and in recent years, supervised, semi-supervised and unsupervised machine learning techniques [27]. Again, we do not survey
these developments here; Elmagarmid et al. [27], Winkler [85] and Köpcke and Rahm [48] have all surveyed state-of-the-art techniques in data matching comprehensively. New techniques continue to be proposed, showing that data matching remains an active area of research; for example, in recent years, Locality Sensitive Hashing (LSH) techniques have gained prominence [45], [44].

**Semantic Web Extensions** The techniques described for the classification step can be applied in both the relational database and Semantic Web communities due to their generic nature. In the Semantic Web, data matching is often referred to as *instance matching* and also *link discovery*. In Figure 9, some specific instance matching systems are listed. The list is not exhaustive but is indicative of how Semantic Web instance matchers have evolved since the Linked Open Data movement began in 2007. These systems were also mentioned in two different surveys that describe Semantic Web instance matching efforts [87], [30].

Rule-based techniques have gained popularity in the Semantic Web, in the form of *link specifications* [37]. The Silk system, which was one of the first proposed in the wake of the Linked Open Data movement [82], and is periodically updated with new state-of-the-art modules, can learn such link specifications using *supervised* genetic algorithms [37]. Silk has also incorporated multi-dimensional blocking methods to speed up link specification processing [38], but it does not learn blocking schemes. Instead, it requires *reference* links or domain expertise in order to efficiently index and aggregate attributes for efficient link discovery. Other efforts that attempt to speed up link discovery, but are different from blocking, make other assumptions. For example, LIMES can significantly speed up link discovery but requires the link specification to be a *metric* distance function [62].

Song and Hellin proposed a system that comes closest to learning an actual blocking scheme [79]. The learned blocking scheme belongs to a class that is a subset of the DNF class. The technique was found to perform well empirically; it was the first indication that DNF blocking schemes (or a narrow subset) are suited to Semantic Web datasets. In Section 3.2, we present a general DNF blocking scheme learner for RDF datasets.
Several supervised or semi-supervised techniques attempt to learn link specifications themselves (similar to the classification step in Figure 4). We already noted Silk and LIMES as examples [82], [62]. A recent trend is to use active learning to minimize user involvement [63],[64]. Ferraram et al. provide a survey of current state-of-the-art link discovery systems [30]. Two important examples, besides the ones described above, are KnoFuss [65] and RDF-AI [75]. RDF-AI offers a comprehensive suite of tools to perform sophisticated data matching [75]. However, it requires the user to specify the workflow and the associated details and configurations. KnoFuss requires the RDF datasets to have associated ontologies; its functioning is based on the premise that the ontologies can be aligned with respect to a reference ontology [65]. If a reference ontology is not available, or if input RDF datasets do not have ontologies, KnoFuss cannot be applied.

Finally, in the last few years, ontology matching systems have been participating in the instance matching track of the Ontology Alignment Evaluation Initiative15 (OAEI), an annual initiative at the International Semantic Web Conference that pits systems against each other using common methodology and benchmarks. RiMOM [50] and LogMap [40] both performed well in 2013 and 2012, although they were originally proposed as ontology matching systems. The cross-fertilization between data and meta-data systems is possible because, in the Semantic Web, both are ultimately expressed in the RDF graph model. However, systems that require the data to be annotated with meta-data (e.g by providing an associated schema or ontology) have limited applicability since, as noted earlier, many datasets in LOD do not have such annotations [76].

A key weakness noted in a comprehensive Semantic Web-focused data linking report by Wolger et al. is that many instance matching efforts require complex specifications serialized in XML, or the presence of a user [87]. In other words, automation is a major issue in current state-of-the-art link discovery in the Semantic Web. Many systems are also inherently serial; in present form, it is not evident how to implement them in a parallel architecture, let alone a shared-nothing paradigm. Benchmark datasets on which many of these systems were evaluated are quite small and fit in main memory. This is also true for the majority of systems in the relational database community. Thus, scalability is also an issue.

2.3 Automation, Heterogeneity and Scalability

In the previous section, we provided generic background on data matching. In this section, we evaluate data matching efforts, as they currently stand in the research literature, from the standpoint of fulfilling automation, heterogeneity and scalability criteria.

**Automation:** Prior to our preliminary work, supervised and semi-supervised techniques exist for learning DNF blocking schemes [5], [60], [12], but there is no

15 http://oaei.ontologymatching.org/
Many authors who evaluate the classification step are forced to provide their own blocking scheme [27], [17]. Thus, the first bottleneck in implementing a fully unsupervised data matching system is that of learning a blocking scheme automatically, without labeled examples. The only unsupervised blocking scheme learner that we found in the literature was the one by Song and Heflin [79]. However, it does not learn general DNF blocking schemes, and has an inherently serial implementation, requiring passes over the full dataset. It also requires the data to be annotated with an ontology, limiting its applicability. In subsequent sections, Contribution 2 will address the latter issue by presenting an unsupervised DNF-BSL that does not require an ontology, but can operate on a non-annotated RDF graph. Contribution 3 will show that the ‘expensive’ component of DNF blocking scheme learning can be implemented in MapReduce, and that only a single pass over the full dataset is sufficient.

Let us assume that we have a blocking scheme. An important question is whether we can automate the remainder of the process (namely, the classification step). The first unsupervised procedure for data matching was the Expectation Maximization (EM) procedure adapted by Winkler [86]. Later authors noted that the method was prone to overfitting and imposed some qualitative restrictions on the data in order to function well empirically [71], [27]. Three important unsupervised classification techniques have since been proposed. Hierarchical graphical models (HGM) proposed by Ravikumar and Cohen [71] attempt to fix overfitting problems posed by Winkler’s EM. They use a graphical model and latent variables to generatively model the problem. The method requires the specification of a distance function and a graphical model. A manually determined blocking technique was used in conjunction with the proposed classification technique, which was evaluated on two small structurally homogeneous tables [71].

Like EM, the method requires cycling through the data and updating a single model’s parameters. A shared-nothing parallel implementation for partitioned data is not evident. These observations demonstrate that the utility of the current method is limited from the perspective of both heterogeneity and scalability.

Bhattacharya and Getoor use a similar generative procedure based on Latent Dirichlet Allocation (LDA). Their method is well-suited for only certain domains, such as matching author names in publications [4]. This is because their method relies on collective linking techniques. The goal is to match (semantically equivalent) author names collectively, based on frequency and pattern of co-authorship among the authors. Thus, each pair is not matched in isolation. Problems may arise both because of specificity of applicable domains, as well as the difficulty of a non-trivial shared-nothing parallel implementation. Furthermore, it is not obvious how the technique can be extended to RDF.

Finally, Christen proposed an SVM-based method [15] that also requires cycling through the data before convergence. The method relies on first locating

---

16 Ravikumar and Cohen [71] used SoftTFIDF, which was shown to be high-performing.
17 Records that shared a 4-gram character sequence were assigned to the same block.
an initial ‘seed’ set of feature vectors that are almost certainly duplicates (or non-duplicates) given that their feature values are almost all 1.0 (or 0.0). Using this seed set, an SVM is trained and more examples are classified. The seed set is supplemented by including duplicates and non-duplicates classified with high confidence in the first iteration. The process is repeated for a given number of iterations or until it converges. At a high-level, the method is similar in principle to EM but is more robust because it uses an SVM. Again, the method was evaluated only on structurally homogeneous tables, and the cyclical nature of the method disables the possibility of a non-iterative shared-nothing implementation. We believe that the ideas of iteratively improving performance and of locating a ‘seed’ set are qualitatively important; we incorporate some of these principles in our own system.

**Heterogeneity**  Heterogeneity, as a requirement, is not fulfilled by the majority of data matching systems. This is because, as the surveys show, researchers in the relational database community did not anticipate the challenges presented by the semi-structured and schema-free nature of Linked Open Data. In the Semantic Web, many existing systems assume associated ontologies for the RDF datasets to be linked, which can be an issue given that RDF datasets are not required to be accompanied by meta-data. In the relational database community, the assumption of structural homogeneity is equally problematic. Even at the blocking level, sophisticated techniques have not been developed for structurally heterogeneous tables [17]. Many authors assume that schema matching, itself a difficult, active area of research [1], [31], has been conducted and the schemas of the structurally heterogeneous tables, physically reconciled, prior to data matching [16], [27]. In Section 3.2, Contribution 2 develops data structures and techniques to address these problems.

**Scalability**  From the discussion on automation, it is evident that scalability and automation seem to be naturally at odds with each other in the data matching community. This is not surprising given that unsupervised techniques in data matching tend to be generative in nature [4], [71], may require multiple passes over the dataset before converging [86], [15], or use global features like TF-IDF\(^\text{18}\). The last is especially true for clustering-based algorithms like the Canopies method of McCallum et al. [56].

This is not to say significant scalability efforts have not been made in data matching. The biomedical community, for example, has devised its own solutions to parallel data matching but these are specific in nature and do not meet automation or heterogeneity requirements [54]. Several efforts have attempted parallelizing SN and other blocking methods in MapReduce [46], [47]. In the relational database community, non-MapReduce parallel and distributed efforts

---

\(^{18}\) A global feature for a pair of strings cannot be computed by considering the strings in isolation. For example, computing the IDF term in TF-IDF requires a pass through the entire dataset [8]
include P-Swoosh and D-Swoosh [41], [2]. In the Semantic Web, we are not aware of any complete MapReduce-based data matching systems. Existing systems are either serial or are implemented in custom architectures [65], [82], [62].

Given the importance of MapReduce (and by extension, cloud-deployment) in real-world Big Data systems [78], we explicitly focus on MapReduce-style shared-nothing parallelism in this dissertation. A good overview of MapReduce is found in the original paper [23]. MapReduce has numerous technical advantages, including implementation on commodity hardware, dynamic task scheduling and robustness to failures, but its shared-nothing nature limits its applicability.

In the context of this dissertation, we denote a data matching system as scalable if it has an evident shared-nothing implementation and if it scales near-linearly in the number of compute nodes with the input. However, in some sense, this denotation is subjective, since there are data matching systems that do have parallel and distributed implementations but not necessarily in the shared-nothing paradigm. Some examples were mentioned above.

Finally, we note that if the distance function is given and the databases are relational, the problem of data matching is akin to a fuzzy or self-similarity join on these databases [81]. Although it makes very specific assumptions, the case is important enough to have witnessed extensive independent research, especially with respect to MapReduce-based implementations. A recent work in this direction is V-smart-join by Metwally and Faloutsos [59].

Scalability has also been receiving attention in areas that are orthogonal to data matching but fall within data integration. A good example is ontology matching [28].

Multiple Domains Systems which meet at least one of the three criteria are summarized in Table 2, together with the associated citation. Note that there are several state-of-the-art systems that do not meet any of the criteria but have excellent empirical performance. A good example is the Marlin system, which uses labeled data to train an SVM in an advanced feature space [6]. We do not preclude the possibility that some of the listed systems may further be extended to account for more criteria. However, as the preceding sections detailed, extending any listed system to meet all three criteria involves overcoming some fundamental challenges.

The additional problem of multiple domains must be noted given that we seek to build a data matching system for Linked Open Data. Consider Figure 10 to understand the problem that arises when we are provided large semi-structured graphs as inputs. Because of the nature of LOD, these graphs can span across several domains. A concrete example is linking the encyclopedic collections of DBpedia and Freebase. If we assume that a dataset is a set of triples over a single domain, the input now comprises collections of datasets. The first step then is to determine the domains and map datasets in the same domain to each other. The single-domain architecture can then be invoked on each mapped dataset pair.

None of the systems in Table 2 distinguish between single-domain or multi-domain inputs. We believe that the problem involves some interesting challenges,
Table 2. Status of current systems in fulfilling Automation (A), Scalability (S) and Heterogeneity (H) requirements. Systems that only perform blocking or that do not meet any of the three criteria below are not listed. For scalability, we adopt the broad definition of a system being scalable if it has a documented parallel and/or distributed implementation, with * indicating the paradigm is shared-nothing. For heterogeneity, we adopt the definition that the system must have been evaluated on semi-structured (or structurally heterogeneous) inputs in the published work.

<table>
<thead>
<tr>
<th>System/Algorithm</th>
<th>Paper</th>
<th>A</th>
<th>S</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winkler’s Expectation Maximization</td>
<td>[86]</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchical Graphical Models</td>
<td>[71]</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collective Linking</td>
<td>4</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Christen’s Support Vector Machine</td>
<td>15</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMES</td>
<td>62</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Silk</td>
<td>82</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swoosh family</td>
<td>3</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEBRL</td>
<td>18</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dedoop</td>
<td>46</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Locality Sensitive Hashing techniques</td>
<td>44</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Papadakis et al.</td>
<td>68</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

especially (as would be the case in an automated system), the number and nature of domains is unknown. We tackle this issue as part of our proposed work in Section 4.

2.4 Concluding Note

There are several high-level points that we would like to emphasize to conclude this chapter. First, data matching is an old problem that spans different communities [16], [27], [30]. Predominantly, it has witnessed research in the relational database community but under the assumption of structural homogeneity. Heterogeneity, therefore, remains the biggest ‘gap’ in the present body of research, since it is not evident how structurally homogeneous techniques can be carried over to semi-structured, schema-free data on Linked Open Data. We will show, as part of our contributions, that this divide is not fundamental; it may be reconciled if the appropriate data structures are employed, even with minimal assumptions about the input databases.

Second, the existing body of work puts automation and scalability at odds with each other. We identified some causes of this earlier; another reason is that scalability researchers tend to assume that the learning process has concluded and that a distance function or a blocking scheme is already available. The focus then is on efficiently implementing the distance function and applying the blocking scheme in a scalable blocking method. Even in non-data matching domains, scalable machine learning is an active area of research [53]. MapReduce-based machine learning packages like Mahout are either iterative or, more recently, have been shunning MapReduce in favor of a custom parallel architecture [66].
We will explore this issue in our contributions, on how to make both learning and application of learned functions scalable in a data matching pipeline.

Finally, even Semantic Web data matching systems implicitly assume inputs to be from a single domain. It is not evident how to adapt these techniques to accommodate input data that spans across multiple (unknown) domains. The problem would not be as severe if the provision of ontologies is a given, but this is often not the case on Linked Open Data. As part of proposed work, we will develop techniques that can handle multi-domain inputs.

From the background, we conclude that building a fully unsupervised data matching system that can operate on semi-structured, schema-free RDF data across multiple domains, and whose workflow has an efficient cloud-deployable implementation, continues to present challenges. This dissertation aims to overcome these challenges in a single system.

3 Completed Work

In this section, we give high-level overview of work that has already been, or is almost, completed. Figure 11 summarizes the key purpose of each contribution and of the proposed work that will be described in Section 4. Section 3.1 (Contribution 1) describes a technique for learning a DNF blocking scheme on structurally homogeneous relational datasets in an unsupervised fashion, and therefore proposes to fulfill an automation requirement in the limited blocking context. Section 3.2 (Contribution 2) extends this work and introduces new machinery so that DNF blocking schemes can be learned in an unsupervised fashion on a pair of semi-structured datasets. Contributions 1 and 2 only concern blocking and not the full data matching pipeline. Section 3.3 extends the scope of
Contribution 1 to realize a fully unsupervised data matching system that has MapReduce-based components. This system only applies to pairs of structurally homogeneous tables, similar to Contribution 1. The extensions in Contribution 2 have not been integrated into Contribution 3. Such an integration involves some challenges and is presently an ongoing effort presented as proposed work (Section 4).

We reproduce the full papers concerning Contributions 1 and 2 in the Appendices. These papers may therefore be read in isolation, even without placing the rest of the thesis in context. We do not reproduce the full paper concerning Contribution 3 since it is still being edited for an upcoming submission. We list it as a contribution and not as proposed work since the system in that contribution has been implemented, and the full suite of experiments has been run. Selected experimental results from all three contributions are also provided.

### 3.1 Contribution 1: An Unsupervised DNF-BSL

The **blocking** phase of two-step data matching has thus far required a human in the loop. This is because blocking methods require a blocking **scheme**, a function that either had to be provided by an expert or learned using manually labeled training data [17]. Although multiple methods have investigated the use of a given scheme in a variety of ways [17], there has been comparatively less research on learning the scheme itself. Three papers sought to address this gap by learning DNF blocking schemes in either a supervised or semi-supervised fashion [60],[5],[12]. We provided some details earlier in Table 1.

---

The author is willing to provide the current draft of this work upon request

---
We extend the state-of-the-art by devising an unsupervised method for learning DNF blocking schemes. The algorithm runs in two separate phases, as shown in Figure 12. In this section, the contribution is described at a high level. The interested reader can find complete technical details in the original paper [42], reproduced in Appendix A.

The main issue of not having manually labeled training samples is addressed in the paper by generating the training set automatically. Training set generation in the data matching context comes with a unique set of challenges. First, the generation must be inexpensive since blocking methods are typically near-linear [17]. Secondly, the quality of the generator output cannot be too degraded given this is the very first step in the full data matching pipeline, and noise tends to have an amplifying effect in each subsequent stage of the pipeline. Given the difficulty of overcoming these challenges, no training set generator for generating both duplicates and non-duplicates had been proposed earlier in the literature.

Our training set generator uses field structure and cosine similarity to generate a small set of duplicates and non-duplicates with minor amounts of noise. Specifically, assuming $m$ fields $f_1, \ldots, f_m$, we apply the simple disjunctive blocking scheme $\text{ContainsCommonToken}(f_1) \lor \cdots \lor \text{ContainsCommonToken}(f_m)$ to the tuples in the table to obtain a set of blocks. We slide a window of constant size $c$, where $c$ is a parameter, over the tuples in each block and compute the cosine similarity between tuples that fall within a common window. If the score is higher than a pre-specified threshold, the pair is added to the duplicates set $D$ and if lower than a second pre-specified threshold, the pair is added to the non-duplicates set $N$. If the score is between the two thresholds, the pair is ignored. Finally, if limits are imposed on the sizes of $D$ or $N$, only the top or bottom scoring pairs are respectively retained in each set.

Since the procedure relies on heuristics, some noise is expected in both $D$ and $N$. Empirically, we found that as long as small numbers of duplicates and non-duplicates were retrieved, the generator performed well. Figure 13 shows the

---

20 Recall that the goal of blocking is to mitigate $\Theta(n^2)$ complexity, for $n$ entities
Table 3. Results of learning disjunctive blocking schemes. RR (Reduction Ratio) and PC (Pairs Completeness) are standard blocking metrics that measure efficiency and effectiveness respectively. FM (F-Measure) is their harmonic mean.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fisher</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR</td>
<td>PC</td>
</tr>
<tr>
<td>Restaurant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.9971</td>
<td>0.9306</td>
</tr>
<tr>
<td>Best</td>
<td>0.9800</td>
<td>0.9554</td>
</tr>
<tr>
<td>Cora</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.9108</td>
<td>0.9113</td>
</tr>
<tr>
<td>Best</td>
<td>0.9330</td>
<td>0.9443</td>
</tr>
<tr>
<td>Census</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.9910</td>
<td>0.9887</td>
</tr>
<tr>
<td>Best</td>
<td>0.9916</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

precision of obtained noisy duplicates at default parameter values. We guarantee that the training set generator runs in time $\Theta(n)$, assuming $n$ total entities. The proof relies on the sliding of the constant-sized window described above. We also show the learner to be empirically robust to varying parameter values within a reasonable range (Appendix A).

In the second phase, the problem of learning blocking schemes from the noisy training set of duplicates $D$ and non-duplicates $N$ is cast as a feature selection problem, using a greedy algorithm and the simple Fisher criterion [32]. First, each pair in $D$ and $N$ is converted into a feature vector with $|H_c|$ elements, where each element is 0 or 1 and $H_c$ is the (possibly supplemented) set of specific blocking predicates (Section 2.2). Specifically, suppose the $i^{th}$ element of feature vector $f$ corresponds to some term $h \in H_c$. Recall that $h$ takes as input a pair of tuples and returns True if it covers, or is satisfied, for that pair (Definition 3). The $i^{th}$ element is assigned a value of 1 if $h$ covers the pair, and is assigned 0 otherwise. In this way, each pair can be converted to an $|H_c|$-dimensional feature vector. Designate the sets of feature vectors corresponding to the pairs in $D$ and $N$ as $D_f$ and $N_f$ respectively. The Fisher score $\rho_i$ of the $i^{th}$ feature is given by the formula below:

$$\rho_i := \frac{|D_f|(|\mu_{D,i} - \mu_i|^2 + |N_f|(|\mu_{N,i} - \mu_i|^2)}{|D_f|\sigma_{D,i}^2 + |N_f|\sigma_{N,i}^2}$$  \(4\)

Here, $\mu_{D,i}$ and $\sigma_{D,i}^2$ is the mean and variance respectively of the $i^{th}$ feature with respect to feature vectors in the set $D_f$ (similarly for $N_f$), while $\mu_i$ is the mean of the $i^{th}$ feature with respect to all feature vectors. Intuitively, a higher scoring feature has greater discriminative ability than a lower scoring one.

Despite its simplicity, Fisher-based feature selection was found to be appropriate for the learning problem because it was robust to noise in the generated training set. Traditional methods, such as the set covering variants proposed in earlier supervised learners [5],[60], were not found to be as robust. We showed that our unsupervised DNF blocking scheme learner was competitive with the supervised state-of-the-art [5]. Table 3 shows the results of learning disjunctive blocking schemes. In the table, Fisher is the proposed feature selection-based
The property table representation. Each property has its own column, and each subject, its own row. null and the delimiter ; are both reserved, the latter used to impart set semantics to each field value.

(a) shows a generic pipeline for learning heterogeneous DNF blocking schemes. (b) shows an unsupervised instantiation

learner that was input noisy samples, with an equal number of perfectly labeled samples provided to the baseline [5]. On average, the system is at par with the baseline, while it outperforms the baseline on Cora.

The high accuracy of the noisy training set influenced us to train a classifier and implement a fully unsupervised data matching system, rather than just learn a blocking scheme, as described in Section 3.3.

3.2 Contribution 2: A Heterogeneous Unsupervised DNF-BSL

Although the previous contribution addresses automation, it is limited to structurally homogeneous tabular datasets as input. In order to expand the contribution to include RDF datasets as well as structurally heterogeneous tabular datasets as input, some challenges need to be overcome. This contribution develops novel techniques to deal with these challenges. Thus, this contribution addresses heterogeneity.

First, we showed that an RDF dataset can also be represented tabularly, and not necessarily as a set of triples or a directed graph, as previous work often assumed. Such a tabular representation already exists as a physical implementation and is known as a property table. It is used in triple stores to store large RDF datasets [84]. We chose to use it as a logical data structure instead. An example of an RDF dataset in property table form is shown in Figure 14.

![Property Table Representation](image)
The problem of learning DNF blocking schemes on an RDF-RDF dataset pair or RDF-tabular dataset pair now becomes one of learning a DNF blocking scheme on a pair of structurally heterogeneous tabular datasets. However, the DNF blocking scheme formalism was explicitly for structurally homogeneous tables. We extend the formalism to accommodate structural heterogeneity by introducing schema mappings as inputs. The full paper, currently being refined for a journal submission, is reproduced in Appendix B.

The generic pipeline is shown in Figure 15. Using a schema matching module (see [1] for a survey), an extended DNF blocking scheme can be learned if training samples are provided. In the paper, we show that of the existing Blocking Scheme Learners (BSLs) covered in Table 1 in Section 2, the supervised and semi-supervised systems (Systems 1-3) can be extended, but our own unsupervised work (Contribution 1) could not, primarily because of the training set generator, which assumes structural homogeneity.

We propose an unsupervised instantiation in Figure 15 (b) by using an existing system called Dumas as our schema matching module. Dumas outputs 1:1 field mappings by first using a duplicates generator to locate tuple pairs with high cosine similarity. In the second step, Dumas uses Soft-TFIDF to build a similarity matrix from each generated duplicate. If n duplicates are input to the second step, n similarity matrices are built and then averaged into a single similarity matrix. The assignment problem is then solved by invoking the Hungarian Algorithm on this matrix [51]. Given two field-sets A1 and A2 from the input datasets, this results in exactly \( \min(|A_1|, |A_2|) \) 1:1 field mappings (the set \( Q \) in the figure) being output.

In addition to using \( Q \), we recycle the noisy duplicates of Dumas and pipe them into the blocking scheme learner. Note that Dumas does not generate non-duplicates. We address this issue in a novel way, by permuting the generated duplicates set \( D \). Suppose that \( D \) contains \( n \) tuple pairs \( \{(r_1, s_1), ..., (r_n, s_n)\} \), with each \( r \) and \( s \) respectively from datasets \( R_1, R_2 \). By randomly permuting the pairs in \( D \), we heuristically obtain non-duplicate pairs of the form \( (r_i, s_j) \), \( i \neq j \). Note that (at most) \( n! \) distinct permutations are possible. For balanced supervision, we set \( |N| = |D| \), with \( N \) the permutation-generated set.

Empirically, the permutation is expected to yield a precise \( N \) because of observed duplicates sparsity in datasets [17, 42]. This sparsity is also a key tenet underlying the blocking procedure itself. If the datasets were dense in duplicates, blocking would not yield any savings.

The extended DNF blocking scheme is learned by using an algorithm that uses only two parameters, and has strong approximation guarantees. We also make some empirical contributions by using an expanded feature set and by making extensive use of phonetic features [16]. This helped compensate for the breadth and sparseness of property tables (even the simple case in Figure 14 demonstrates this), and also noise in Dumas outputs.

Table 4 compares the performance of the unsupervised heterogeneous DNF-BSL to adapted semi-supervised and supervised baselines. The supervised method

\[ \text{The formalism was covered in Section 2} \]
performed better on the RR (efficiency) metric but, in general, all methods were equally effective (evidenced by PC). An important consequence that this result conveys is that a high-performing training set generator is not as crucial for achieving effectiveness as efficiency. This is yet another manifestation of the automation-scalability tradeoff described in Chapter 2 and that frequently appears in other data matching literature. It is encouraging that the blocking scheme learner is able to somewhat compensate, but as part of proposed work (Section 4.1) we are designing a high-performing heterogeneous training set generator (instead of relying on Dumas’s generator) both to improve RR performance, as well as to ensure we successfully learn a high-quality heterogeneous classifier for the second data matching step. Finally, note that the test cases (DPs) used in the evaluation of this work were larger and noisier than the benchmarks in Contribution 1. This might be why Dumas’s generator did not perform as well as it did in the original paper [9]. For full details, refer to Appendix B.
3.3 Contribution 3: Fully Unsupervised Tabular Data Matching

In Section 3.1, we used the noisy training sets to learn only a DNF blocking scheme. It is conceptually possible to reuse such a training set for learning the classifier function, which is the second step of two-step data matching. The result would be a fully unsupervised data matching system that would satisfy the automation requirement of the dissertation. If such a system is also amenable to a MapReduce-based implementation, then it can be adapted to satisfy the scalability requirement while keeping automation intact. Recall that in Section 2.3, we stipulated that a correct MapReduce implementation was the main criterion for ensuring scalability.

We noted (in the concluding note in Chapter 2) that current work often pits scalability against automation. This consideration needs to be borne in mind as automation and scalability are addressed in a single system.

The full schematic is shown in Figure 16. The system is fully unsupervised and the key steps are implemented in MapReduce, but it still only accepts a pair of structurally homogeneous tables as input. The training set generator in Contribution 1 is modified so that it is amenable to a shared-nothing implementation. It is then implemented in a first MapReduce algorithm, the output of which is the noisy training set of duplicates and non-duplicates.

These sets are used in a serial module for learning two functions, namely a DNF blocking scheme (the learner being nearly identical to the feature selection-based learner in Contribution 1) as well as a classifier. Since the classifier operates at a finer level of granularity, we use a smaller (but more precise) subset of the noisy training samples to train it. Good empirical results are expected because of the high accuracy in the noisy training set, as noted in Contribution 1. We note that because of the global dependence of learning algorithms, these learners can’t be implemented in single-step MapReduce. However, we provide provably tight bounds on the run-time of the serial module by ensuring that constant numbers

---

22 These continue to be widely used, in part because qualitatively challenging data matching test cases are hard to come by, and also because practitioners assume structural homogeneity, as described in Chapter 2 [17]
Fig. 17. (a) shows results after a single run of the system on the Cora benchmark compared to a supervised SVM, while (b) shows performance improvement on a challenging Census benchmark after a single iteration following the run. 10% and 50% represent two different configurations described in detail in the paper of noisy training samples\textsuperscript{23} are input to both learners. Technically, we note that the serial module can be implemented ‘trivially’ in a MapReduce step, by piping all records to a single reducer and performing the work in the reducer. Whether the serial module is implemented in MapReduce or as a program running on a single node is an implementation issue\textsuperscript{24}.

The second MapReduce algorithm again takes the datasets as input and applies the learned blocking scheme in the mapper. Each block is collected in an individual reducer instance, where the classifier is applied to label pairs as duplicates or non-duplicates. In our implementation, we used a Support Vector Machine (SVM) classifier, and configured it to also return confidence estimates. Such estimates are useful because high-confidence pairs output after the first run may then be fed back into the serial module to iterate the process. In Figure 17, we find that on the Cora benchmark, the system is at par with the supervised method, while on other\textsuperscript{25} benchmarks like Census, this feedback process is required to achieve at-par results.

These results are promising but preliminary; empirical testing of a cloud-deployed version of this system on large datasets is part of the proposed work plan (Section 5). An important point to note is that, although the intermediate output is piped to a serial module, the final output of the system is written to the distributed file system (DFS), which is also where the input is read from, per MapReduce semantics. Technically, the intermediate output is first written to the DFS and then transferred to a local file system. Other than providing strong run-time bounds on the serial module, future versions will attempt to

\textsuperscript{23} Or the full noisy training set, if it is smaller than the constant
\textsuperscript{24} The MapReduce version becomes important, for instance, if the full system is required to be MapReduce-compatible, because of infrastructure constraints
\textsuperscript{25} This benchmark was a larger and more challenging version of the Census benchmark in Contribution 1, although Cora was the same
control the output of the first MapReduce task, so that intermediate bandwidth and I/O costs are minimized.

We end with a note on why a heterogeneous extension of the system is problematic. Recall the work in Contribution 2, where we used Dumas to achieve an unsupervised instantiation of the heterogeneous BSL pipeline. In our experiments (Appendix B), we observed that the training set output by Dumas had low accuracy compared to our homogeneous training set generator in Contribution 1; this was not surprising since we assumed homogeneity and used it to guide our generator. Low accuracy of the Dumas training set implies that it cannot be directly used to learn a good classifier, although we were able to use the samples to learn a well-performing DNF blocking scheme by exploiting the coarse granularity of blocking schemes. The first step, therefore, is to devise a high-accuracy heterogeneous training set generator that can additionally be implemented in MapReduce. We are currently addressing this challenge as proposed work, described in Section 4.

4 Proposed Work

The remaining contributions that we propose are described in this section.

4.1 Proposed Contribution 1: A Heterogeneous Data Matching Workflow

We loosen the constraint that the datasets input to the MapReduce-based architecture are structurally homogeneous tables. A direct integration of Section 3.2 techniques into the fully unsupervised data matching architecture in Section 3.3 is problematic. First, the training set generator in Figure 16 uses a composite distance function to automatically generate the training set. We were able to experimentally show high levels of accuracy for that composite distance function for homogeneous datasets; the high accuracy enabled us to use the generator for training an SVM. The automatic feedback allowed us to refine results that were not initially satisfactory. In Section 3.2, Dumas’s generator was shown not to perform as well as we might have hoped; more importantly, the cosine similarity function used by Dumas is not amenable to single-step MapReduce, a weakness that was also noted when we considered using the generator in Section 3.1 as a module in Figure 16.

As a first step, therefore, we propose to adapt the composite distance function to yield precise duplicates for heterogeneous datasets. We believe that achieving high precision will prove to be a challenge. It is quite likely that changes will be required to the classifier we train in the intermediate module, to ensure that it is able to compensate for higher noise levels than what it was initially accustomed to in the quality evaluations in Section 3.3. We will also integrate the new BSL proposed in Section 3.2 together with an appropriate schema matcher, in the intermediate module. Thus, the end result of this contribution is that the schematic in Figure 16 will still be fully unsupervised and MapReduce-based,
but will now be able to accept semi-structured, schema-free (in other words, heterogeneous) dataset pairs as input.

4.2 Proposed Contribution 2: The Multi-Domain Problem

A key assumption that has been made thus far is that (at most) two data sources are provided as input, with both conforming to a single domain. We loosen this restriction to allow an arbitrary number of domains (say $N$). The input then is not a pair of datasets but a collection of $M$ datasets, where $M \geq N$. The problem is to perform data matching on this collection. Furthermore, given that this collection is merely a set of triples, $M$ and $N$ will both be unknown.

A multi-domain data matching system is applicable for many real-world test suites. Consider, for example, DBpedia and Freebase, both of which contain many domains. DBpedia and Freebase dumps are available as sets of triples. To match data between the two databases, individual domains will first have to be identified in each database and mapped to each other. A single-domain data matching system can then be run on each mapped dataset pair. We designate the process of finding mappings between datasets of the same domain as dataset mapping.

The multi-domain problem presents some unique challenges. The primary challenge is that of heterogeneity, since the number of domains is unknown. If the collection is naively treated as a single-domain input, the results of the data matching system will likely be erroneous and contain both false positives and negatives. It will also lead to missed savings, from a scalability perspective. As a simple case, suppose a database contains $n$ entities equally distributed over $d$ domains. Suppose a single-domain data matching system has complexity $O(f(n))$ for $n$ entities. If dataset mapping took place, the system run-time would be $O(d.f(n/d))$, while without dataset mapping, it is $O(f(n))$. In practice, each of the $d$ runs would be parallelized, bringing the run-time $O(d.f(n/d))$ down to $O(f(n/d))$.

We have already developed some insight into the multi-domain problem by publishing a workshop paper detailing some preliminary work and a simple algorithm for dataset mapping [43]. Developing more advanced techniques that are evaluated on difficult cases is part of proposed work.

4.3 Proposed Contribution 3: Generic Link-Discovery (Optional)

Although the dissertation expressly addresses data matching, broader Semantic Web applications are possible with the proposed system. Recall that, in the Semantic Web, data matching is the problem of finding and establishing missing owl:sameAs properties between subject nodes. In the generic link-discovery problem statement, the link does not have to be owl:sameAs or obey equivalence class semantics. Instead, suppose we were given a few training examples implicitly demonstrating the link. A real-world example, taken from the Constitute Project\textsuperscript{26}, is shown in Figure 18.

\textsuperscript{26}https://www.constituteproject.org/
We have two instances, one describing sections from country constitutions, and another describing cases tried in that country. The goal is to link the case to the correct sections of the constitution used in the courts to describe that case. A domain expert would be able to specify a small number of examples, without detailed technical knowledge of how to actually specify the link in a link specification language, such as Silk [37]. To accomplish the task of locating all such links, we have to implement a process similar to the two-step data matching process, which is blocking followed by classification. The difference is that the training set generator would have to be taken out of the loop, since it is predated on the assumption that we require training samples explicitly for locating owl:sameAs links. One challenge is that the domain expert might provide only a sparse training set. While insufficient representation of data in the training set might seem fundamentally insurmountable, we showed (in preliminary workshop proceedings) that good empirical results are possible by including adapted bagging algorithms in our heterogeneous DNF blocking scheme learner [43]. We will test if such bagging techniques also improve generic link-discovery results for the full system; if not, we will not address this issue further in the present dissertation but leave it for future research.

4.4 Deliverable

The full system with or without the optional contribution will be evaluated on a cluster in the Windows Azure infrastructure on large datasets. We will run a detailed series of scalability tests, including testing dependence on number of nodes, input dataset size and the effect on scalability by the ‘bottleneck’ serial module. A work plan is provided in the final section of this document. The completed schematic should closely resemble the one shown in Figure 19. We do not include the (optional) generic link-discovery contribution in the diagram, but in essence, for non-data matching link-discovery, the training set generator would be removed and replaced with a small number of manually provided training examples. Additionally, appropriate bagging techniques would be applied in the serial module, as earlier described.

We will use this prototype to populate application-specific Entity Name Systems for the Emerald data integration system currently being built by our research group. The architecture of Emerald is shown in Figure 20. Much of Emerald...
ald has already been built by prior research efforts, but the ENS has limited capabilities and has to be manually populated. This dissertation aims to overcome the current limitations.

Fig. 19. The heterogeneous, automated, scalable multi-domain data matching system.

5 Work Plan

We propose the following plan, along with a Gantt chart (Figure 21) visualizing the expected progress of incomplete work:

1. Gather benchmarks and qualitatively challenging test cases for data matching:
   (a) Gather and curate structurally homogeneous tabular benchmarks for Contribution 1 (Complete).
   (b) Gather and curate structurally heterogeneous and RDF datasets for Contribution 2 (Complete).
   (c) Gather and curate datasets (both homogeneous and heterogeneous) for large-scale scalability evaluations28 in Contribution 3 and its proposed extension (Proposed, 1 month).

28 In the completed work described in Section 3.3, we conducted only limited scalability evaluations
Populating a Linked Data-based Entity Name System

Fig. 20. The *Emerald* data integration system. The Entity Name Service is the API exposing the services of the Entity Name System, which is what this dissertation seeks to automatically populate for large, heterogeneous inputs.

Fig. 21. A Gantt chart showing expected progress of incomplete work.
(d) Gather and curate multi-domain evaluation suite (Proposed, 1 month). We expect to use DBpedia and Freebase extensively as part of our evaluations.

2. Automation:
   (a) Implement and evaluate unsupervised DNF-BSL in Section 3.1 (Complete).
   (b) Implement and evaluate fully unsupervised serial data matching on structurally homogeneous tabular pairs in Section 3.3 (Complete).
   (c) Implement fully unsupervised MapReduce-based data matching in Section 3.3 (Complete).
   (d) Evaluate the above MapReduce-based data matching system on large homogeneous datasets on a cloud-computing cluster (In Progress, cluster allocation acquired on Windows Azure, dataset sources identified).

3. Implement and evaluate fully unsupervised serial heterogeneous DNF-BSL in Section 3.2 (Complete)

4. Proposed Work (all evaluations on large datasets):
   (a) Adapt and evaluate MapReduce-based training set generator for heterogeneous dataset pairs (Proposed, 1 month).
   (b) Adapt and evaluate MapReduce-based classifier for heterogeneous dataset pairs (Proposed, 1 month).
   (c) Run full scalability evaluations of heterogeneous MapReduce-based system (Proposed, 1 month).
   (d) The multi-domain problem in MapReduce, leading to the realization of Figure 19 (Proposed, 4 months).
   (e) Expanding current efforts to accommodate generic link-discovery (Optional, 1 month).

5. Theoretical analysis of selected efforts (Proposed, 1 month).

6. Wrap-up, dissertation and final exam (Proposed, 2-3 months).

References


Appendices
A An Unsupervised Algorithm for Learning Blocking Schemes

Abstract. A pairwise comparison of data objects is a requisite step in many data mining applications, but has quadratic complexity. In applications such as record linkage, blocking methods may be applied to reduce the cost. That is, the data is first partitioned into a set of blocks, and pairwise comparisons computed for pairs within each block. To date, blocking methods have required the blocking scheme be given, or the provision of training data enabling supervised learning algorithms to determine a blocking scheme. In either case, a domain expert is required. This paper develops an unsupervised method for learning a blocking scheme for tabular data sets. The method is divided into two phases. First, a weakly labeled training set is generated automatically in time linear in the number of records of the entire dataset. The second phase casts blocking key discovery as a Fisher feature selection problem. The approach is compared to a state-of-the-art supervised blocking key discovery algorithm on three real-world databases and achieves favorable results.

A.1 Introduction

Record Linkage, or the identification of entities within a database that are coreferent, is a long standing problem with no less than eight separate terms referring to the same problem [27]. Wikipedia lists at least fifteen different names, and despite much research the problem does not have an automated solution. Ad hoc and domain dependent solutions are still common, with human intervention required.

Record Linkage typically requires two primary steps [27]. The first step is referred to as blocking. Blocking methods mitigate full pairwise comparisons by selecting a small subset of pairs from the database that are considered to be good candidates for pairwise comparison, while discarding the vast majority of pairs that are clearly non-coreferent. Without blocking, each entity must be compared with every other entity to determine whether the two corefer. This naive approach grows quadratically with the input, and is impractical for large databases; hence, the need for blocking. The blocking step is comprehensively surveyed by Christen [17].

The pairs generated by blocking are then used as input for a second step, which typically involves machine-learning techniques, among others, to isolate duplicates according to some similarity measure. The second step is comprehensively surveyed by Elmagarmid et al. [27].

The blocking phase of this two-step procedure has thus far required a human in the loop. This is because blocking methods require a blocking scheme, a function assumed to be provided by a domain expert [17]. Although multiple methods have investigated the use of a given scheme in a variety of ways [17], there has

[29] https://en.wikipedia.org/wiki/Record_linkage
been negligible research on learning the scheme itself. Two papers sought to address this gap by learning schemes given training data [60, 5]. However, labeling duplicates in large databases is troublesome, particularly if duplicates are sparse or the data is confidential. As the size and diversity of datasets continues to grow in the current era of Big Data, the need for an automated procedure is pressing.

In this paper, an unsupervised method is presented for learning blocking schemes. The algorithm runs in two separate phases. In the first phase, the algorithm efficiently generates a weakly labeled training set. In the second phase, the problem of learning blocking schemes from this weakly labeled set is cast as a feature selection problem. The validity of both phases of the algorithm is demonstrated on three real-world datasets.

A.2 Blocking Schemes

The formalism for the rest of the paper is introduced, along with illustrative examples. For consistency, terminology proposed by Bilenko et al. is used [5], although many terms below were not formally defined in that work.

Definitions and Examples The most basic elements of a blocking scheme are indexing functions $h_i(x_t)$ [5]. An indexing function accepts a field value from a tuple as input and returns a set $Y$ that contains 0 or more blocking key values (BKVs). A BKV identifies a block in which the tuple is placed. Intuitively, one may think of a block as a hash bucket, but more often the function is used to sort the records [35]. If the set $Y$ contains multiple BKVs, then the tuple is assigned to multiple blocks.

Table 5. Two sample tuples from the Restaurant dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Cuisine</th>
</tr>
</thead>
<tbody>
<tr>
<td>arnie morton’s of chicago</td>
<td>435 s. la cienega blv.</td>
<td>los angeles</td>
<td>american</td>
</tr>
<tr>
<td>campanile</td>
<td>624 s. la brea ave.</td>
<td>los angeles</td>
<td>american</td>
</tr>
</tbody>
</table>

Definition 5. An indexing function $h_i : \text{Dom}(h_i) \rightarrow U^*$ takes as input a field value $x_t$ from some tuple $t$ and returns a set $Y$ that may contain 0 or more Blocking Key Values (BKVs) from the set of all possible BKVs $U^*$.

For clarity, the domain of an indexing function is always a string, although technically it can be any primitive data type. The range is a set of Blocking Key Values that the tuple is assigned to. Each BKV is represented by a string identifier. Note that all tuples are converted to lower-case as a first preprocessing step.

Example 6. An example of an indexing function is Tokens. Tokens takes a field value of a tuple and parses it into a set of tokens using a set of common delimiters.
(such as whitespace and comma). This set is then returned as the BKV set that identifies the blocks in which the tuple is placed. For example, consider the first sample tuple in Table 5 taken from the Restaurant dataset, which has four fields. If Tokens is applied to the name field, the set \{arnie, morton’s, of, chicago\} is obtained.

This leads to the notion of a general blocking predicate. Intuitively, a general blocking predicate \(p_i(x_{t_1}, x_{t_2})\) takes as input field values from two different tuples, \(t_1\) and \(t_2\), and uses the \(i\)th indexing function to obtain sets of keys \(Y_1\) and \(Y_2\) for the two arguments. The predicate is satisfied if and only if the two sets intersect i.e. if both tuples share at least one common block.

**Definition 6.** A general blocking predicate \(p_i : \text{Dom}(h_i) \times \text{Dom}(h_i) \rightarrow \{\text{True}, \text{False}\}\) takes as input field values \(x_{t_1}\) and \(x_{t_2}\) from two different tuples, \(t_1\) and \(t_2\), and returns True if \(h_i(x_{t_1}) \cap h_i(x_{t_2}) \neq \emptyset\), and returns False otherwise.

Note that each general blocking predicate is always associated with an indexing function. Also note the distinction between a field and a field value (an instantiation of a field). For example, in Table 5 cuisine would be a field, but american would be a field value.

**Example 7.** Consider the general blocking predicate Contains Common Token, associated with the indexing function Tokens in Example 1. Suppose this predicate is applied to the address field values of the two tuples shown in Table 5. The predicate first applies Tokens to the address field values of both these tuples and then intersects their resulting BKV sets. Since the intersection yields two common tokens \{s., la\}, the two tuples will share at least two common blocks. Hence, Contains Common Token will return True for this pair, when applied to address.

Such predicates are called general because their indexing functions are not associated, a priori, with specific fields. For example, Contains Common Token above could have been applied just as easily to some other field (like name) than to address.

On the other hand, a specific blocking predicate explicitly pairs a general blocking predicate to a specific field.

**Definition 7.** A specific blocking predicate is a pair \((p_i, f)\) where \(p_i\) is a general blocking predicate and \(f\) is a field. A specific blocking predicate takes two tuples \(t_1\) and \(t_2\) as arguments and applies \(p_i\) to the appropriate field values \(f_1\) and \(f_2\) from both tuples. Furthermore, a tuple pair for which the specific blocking predicate is True is said to be covered by that specific blocking predicate.

**Example 8.** In the Tokens example, \((\text{Contains Common Token}, \text{address})\) and \((\text{Contains Common Token}, \text{name})\) would both be specific blocking predicates.

In this paper, it is assumed that all available general blocking predicates can be applied to all fields of the relation. Hence, given a relation \(R\) with \(m\) fields, and \(s\) general blocking predicates, the number of specific blocking predicates will be exactly \(ms\).

A blocking scheme is defined as:
Definition 8. A DNF blocking scheme $f_P$ is a function constructed in Disjunctive Normal Form (disjunction of terms), using a given set $P$ of specific blocking predicates as literals, and with the constraint that negated literals may not be used in the construction. Furthermore, if the additional constraint is employed that each term comprise at most 1 literal, the blocking scheme is referred to as a Disjunctive Blocking Scheme.

Note that a Disjunctive Blocking Scheme is merely a disjunction of specific blocking predicates while a general DNF Blocking Scheme is a disjunction of terms, where each term is a conjunction of specific blocking predicates. A blocking scheme may be applied to a tuple pair, which is said to be covered if the scheme returns True for the pair, just like with specific blocking predicates.

The Blocking Step

Given a blocking scheme $f_P$ on a relation $R$, the blocking step of the record linkage process takes each tuple $t$ of $R$ and applies the indexing functions corresponding with the literals (that is, the specific blocking predicates) in $f_P$ to $t$. For a term $C$ of specific blocking predicates in $f_P$, a cross product of the BKV sets of the individual literals is computed. Each element of this cross product set now references a block to which the tuple is assigned.

Example 9. Consider the (rather simple) blocking scheme $(\text{Contains Common Token, address}) \land (\text{Exact Match, city})$. Given this scheme, the BKV sets returned (by applying the indexing functions) for the first tuple are \{435, s., la, cienaga, blv.\} and \{'los angeles'\} while for the second tuple the sets \{624, s., la, brea, ave.\} and \{'los angeles'\} are returned. The cross product set of the first tuple is thus \{435 ‘los angeles’, s. ‘los angeles’, la ‘los angeles’, cienaga ‘los angeles’, blv. ‘los angeles’\} and of the second tuple is \{624 ‘los angeles’, s. ‘los angeles’, la ‘los angeles’, brea ‘los angeles’, ave. ‘los angeles’\}. Whitespace is used as a delimiter within each cross product element. The first tuple is assigned to 5 blocks (the cardinality of its cross product set); similarly for the second tuple. Because the values s. ‘los angeles’ and la ‘los angeles’ are common to both sets, the tuples will share exactly two common blocks.

Because the blocking scheme is framed in DNF, each term can be thought of as a single pass over the relation $R$. A disjunction of terms in the scheme can be used to simulate the multi-pass approach of Hernandez and Stolfo [35]. Essentially, a DNF blocking scheme $C_1 \lor C_2 \ldots \lor C_k$ with $k$ terms is the same as $k$ blocking schemes applied to the relation $R$, each in its own independent pass. The union of the $k$ total sets of blocks produced in the $k$ passes results in a single set of blocks $\Pi$.

Within each block in $\Pi$, all tuples are paired with each other to produce a set of pairs for that block. The union of all such sets of pairs is then used to generate the final candidate set of pairs, as given by the formula:

$$\Gamma = \bigcup_{B \in \Pi} \{\{t_i, t_j\} : \forall \{t_i, t_j\} \subseteq B | i \neq j\}$$ (5)
where $\Gamma$ is the candidate set of pairs, $B$ refers to a block in $\Pi$ and $t_i, t_j$ are tuples within that block. The constraint $i \neq j$ is imposed to prevent a tuple pairing with itself. As the candidate set is formed by a union, a pair can only occur once, even though it could potentially have been generated more than once across multiple blocks. Moreover, since each pair is modeled in the above equation as a set of cardinality 2, the order of the pair elements do not matter. Note that a block with fewer than two tuples is discarded, since no pairs can be generated for that block.

This final candidate set of pairs becomes the input to the second step of the record linkage process, which essentially involves classifying each pair as a match, non-match or possible match [27]. If the blocking scheme was ‘good’, this set would satisfy some desirable properties. This notion is explored further in the next section.

**Metrics** Not all blocking schemes are equally good or useful. As an example, consider a simple blocking scheme (Exact Match, Gender) on a personnel database. Assume that the domain of Gender is the set \{'M', 'F'\} and that half of the personnel are men and half are women. This blocking scheme would produce two blocks with half the tuples in one block and half in the other. Obviously, such a scheme would not be of much help in reducing the cardinality of the final candidate set by more than a small constant factor. On the other hand, a blocking scheme on the last four digits of the Social Security Numbers of personnel would result in a huge reduction in the candidate set, but might accidentally exclude true duplicates from the set if typographical or other errors are present in those digits.

Thus, there is a tradeoff between ensuring that true positives are included in the candidate set and cutting down the size of the candidate set. Two metrics, *Reduction Ratio* and *Pairs Completeness* are employed to express these goals [17].

The *Reduction Ratio* (RR) quantifies the extent to which the blocking scheme minimizes the number of candidate pairs. It can be expressed by the formula:

$$RR = 1 - \frac{|\Gamma|}{|\Omega|}$$

where $\Gamma$ is the final candidate set as given by (1) and $\Omega$ is the set of candidate pairs that would be generated in the absence of blocking. As an example, for a single relation with $n$ tuples, the number of candidate pairs in the absence of blocking would be $n(n-1)/2$, since a tuple is not allowed to pair with itself. An RR close to 1.0 indicates that few candidate pairs have been generated, while an RR close to 0.0 indicates that the reduction achieved by the blocking scheme was small.

*Pairs Completeness* (PC) is the ratio of the true positives or duplicates in the candidate set to those in the full set of pairs $\Omega$. PC can be expressed using the formula:

$$PC = \frac{|\Gamma_m|}{|\Omega_m|}$$
where \( \Gamma_m \) is the set of true positives in the candidate set generated by the blocking step and \( \Omega_m \) is the set of true positives in the entire dataset (equivalently, the number of true positives in the full set of pairs \( \Omega \)). The PC metric, therefore, captures the notion of recall in the framework of blocking.

Ideally, PC should be exactly 1.0, indicating perfect recall, while RR should be close to 1.0 (but not exactly 1.0 unless the dataset contains no duplicates at all), indicating the blocking scheme did a good job of reducing the candidate set. There is a clear tradeoff in achieving both goals.

In order to succinctly express this trade-off, the harmonic mean of the two quantities, called the F-Measure or \( FM \), is computed. It can be expressed by the formula:

\[
FM = \frac{2 \cdot RR \cdot PC}{RR + PC}
\]  

Another metric Pairs Quality \( PQ \) is similar to \( PC \) except that it computes the percentage of true positives in the candidate set of pairs \( \frac{|\Gamma_m|}{|\Gamma|} \). Although this metric has been used to evaluate different blocking methods, given the blocking scheme, it has been unemployed by researchers seeking to learn blocking schemes [5], [60].

**Learning Blocking Schemes**

Learning a blocking schemes can be stated as an optimization problem, given a training set of positive and negative examples. Bilenko et al. [5] stated this objective as:

\[
f^*_P = \arg\min_{f_P} \sum_{(t_i, t_j) \in N} f_P(t_i, t_j)
\]  

s.t.

\[
|D| - \sum_{(t_i, t_j) \in D} f_P(t_i, t_j) < \epsilon
\]  

where \( N \) is the set of negative examples (non-duplicate pairs), \( D \) is the set of positive examples (duplicate pairs), \( t_i, t_j \) are tuples, \( f_P \) is a DNF Blocking Scheme, and \( \epsilon \) is a parameter that allows up to \( \epsilon \) pairs to be uncovered by the learned scheme. Intuitively, an ‘optimal’ blocking scheme \( f^*_P \) should maximize the duplicate training set covered, while minimizing the non-duplicates covered, thereby achieving the tradeoff mentioned earlier. The parameter \( \epsilon \) is a practical means of loosening the rigid constraint that all duplicate pairs be covered.

**Baseline**

Two independent works developed blocking schemes given a set of manually labeled examples [60, 5]. Both works employed a greedy approach. Michelson and Knoblock [60] employed a variant of a sequential set covering algorithm, \textit{SequentialCovering}, to learn a DNF Blocking Scheme. Bilenko et al. [5] framed the optimization problem in Equations 9 and 10 as a variant of the Red-Blue Set Cover problem [13].
Despite being shown to be NP-hard [13], several approximation algorithms have been proposed for this problem [69]. Bilenko et al. [5] adopted the algorithm by Chvatal [19] and titled it \textit{ApproxRBSetCover} [5]. Blocking Schemes are learned by \textit{ApproxRBSetCover} with inputs comprising a given set of $t$ specific blocking predicates, a training set of duplicates $D$ and non duplicates $N$, and parameters $\epsilon$ and $\eta$, with $\eta$ the maximum number of negative examples a predicate is allowed to cover. Full pseudocode and details can be found in the baseline paper [5].

Like \textit{ApproxRBSetCover}, the algorithms in this paper handle the DNF case separately from the more restrictive Disjunctive case, and accept similar parameters. To enable fair comparison, therefore, \textit{ApproxRBSetCover} was chosen as the baseline.

\section*{A.3 Generating Weakly Labeled Training Set}

\textbf{Algorithm} To the best of our knowledge, current proposals for learning blocking schemes mandate a manually labeled set. For several reasons, generating even a small, perfectly labeled set is non-trivial. First, many databases are sparse in duplicate pairs. For example, on the three benchmark datasets used in this paper, fewer than 1.5 percent of the full set of pairs were duplicates. Secondly, a domain expert is required to have access to the database and carefully locate duplicates.

To address this problem, a scheme for generating a \textit{weakly} labeled training set is proposed. The pseudocode of the algorithm is shown in Algorithm 1. The user may specify a maximum of $d$ duplicates and $nd$ non duplicates to be returned. By default, these are set to the maximum possible number of pairs. Three additional parameters, the upper threshold, $ut$, lower threshold, $lt$, and window size $c$ may be optionally specified with $0.0 < lt \leq ut \leq 1.0$ and $c > 1$. Intuitively, $ut$ and $lt$ control the strictness of the algorithm in generating the weakly labeled set. The role of $c$ is detailed subsequently.

The default values of $ut$, $lt$ and $c$ are 0.05, 0.01 and 20 respectively. These values were found after some preliminary experimentation on one of the three benchmarks, and did not require any consequent tuning (see the Experiments section). Moreover, as demonstrated in Results and Discussion, these values are stable. That is, variation of these parameters over a non-trivial range did not significantly degrade performance on any of the three datasets, all of which have differing characteristics.

The key step of the algorithm comprises a simple Disjunctive Blocking Scheme \newline \textit{(Contains Common Token, $f_1$)} $\lor$ \textit{(Contains Common Token, $f_2$)} $\ldots$ $\lor$ \textit{(Contains Common Token, $f_m$)} where the relation $R$ is assumed to have $m$ fields $f_1, f_2, \ldots, f_m$. Thus, the blocking scheme merely makes $m$ passes over $R$, once for each field, and tokenizes the corresponding field value of each tuple in each pass. The tuple is then placed in (possibly multiple) blocks, each indexed by a single token. Note that a token is local to its field; the same token generated from another field will reference a separate block.
Algorithm 1 WeakTrainingSet\((R, ut, lt, c, d, nd)\)

**Input:**

- Relation \(R\)
- Upper Threshold \(ut\)
- Lower Threshold \(lt\)
- Blocking Window Size \(c\)
- Maximum Duplicate Pairs Requested \(d\)
- Maximum Non-Duplicate Pairs requested \(nd\)

**Output:**

- A set of positive examples \(P\)
- A set of negative examples \(N\)

**Method:**

1. Initialize set \(P\) of duplicates to be empty
2. Initialize set \(N\) of non-duplicates to be empty
3. Initialize set of tuple pairs \(C\) to be empty
4. Generate TFIDF statistics of \(R\)
5. for all fields \(f\) in the schema of \(R\) do
   - for all tuples \(t \in R\) do
     - Tokenize \(t\)'s field value in \(f\)
     - Block \(t\) on generated tokens
   - end for
6. end for
7. for all blocks \(B\) generated in previous step do
   - Slide a window of size \(c\) over tuples in \(B\)
   - Generate all possible pairs within window and add to \(C\)
8. end for
9. for all pairs \((t_1, t_2) \in C\) do
   - Compute TFIDF similarity \(sim\) of \((t_1, t_2)\)
   - if \(sim \geq ut\) then
     - if \(|P| < d\) then
       - add \((t_1, t_2)\) to \(P\)
     - continue
     - end if
   - if \(sim > lowest\ sim\ in\ P\) then
     - Replace pair with lowest \(sim\) in \(P\) with \((t_1, t_2)\)
   - end if
   - if \(sim < lt\) then
     - if \(|N| < nd\) then
       - add \((t_1, t_2)\) to \(N\)
     - continue
     - end if
   - if \(sim > lowest\ sim\ in\ N\) then
     - Replace pair with lowest \(sim\) in \(N\) with \((t_1, t_2)\)
   - end if
10. end for
11. Return \(P\) and \(N\)
After all the blocks have been generated, for all tuples and fields, a sliding window passes over \( c \) tuples at a time and all pairs within this window are generated. The similarity measure (\( sim \) in the pseudocode) of each pair \((t_1, t_2)\) is then computed using the log TFIDF measure [8]:

\[
sim(t_1, t_2) = \sum_{q \in t_1 \cap t_2} w(t_1, q)w(t_2, q)
\]

(11)

where

\[
w'(t, q) = \log(tf_{t,q} + 1).\log(\frac{|R|}{df_q} + 1)
\]

(12)

and

\[
w(t, q) = \frac{w'(t, q)}{\sqrt{\sum_{q \in t} w'(t, q)^2}}
\]

(13)

where \( w(t, q) \) is the normalized TFIDF weight of a term \( q \) in a tuple \( t \), \( tf_{t,q} \) is the term frequency of \( q \) in \( t \), \( |R| \) is the total number of tuples in the relation \( R \) and \( df_q \) is the number of tuples in which the term \( q \) appears.

Hence, given a pair of tuples, \( sim \) is computed, and its value is checked against \( ut \) and \( lt \). If \( sim \) falls strictly between \( ut \) and \( lt \), it is considered ambiguous and ignored. Otherwise, if \( sim \geq ut \), the algorithm checks its current list, \( D \), of the best (i.e. with highest \( sim \) values greater than \( ut \)) duplicates found so far. If \( D \) contains fewer than \( d \) elements, the current pair gets added to the list. Otherwise, the current pair replaces the lowest ranking pair in \( D \) if the current pair’s \( sim \) value is higher. An analogous procedure ensues if \( sim < lt \). However, the algorithm picks the nd worst non-duplicates below \( lt \), where one non-duplicate pair is worse than another if it has higher \( sim \) value. The rationale for this choice is that trivially best non-duplicates (\( sim \approx 0.0 \)) usually do not have good discriminative power. For example, consider the phone number field of a personnel database of a company that has offices only in Illinois and Texas. The first three digits of the phone number for all employees will always be ‘512’ or ‘217’ and irrelevant for discriminating between duplicates and non-duplicates. This irrelevance should be reflected in the set of non-duplicates but would not be if only the best non-duplicates (\( sim \approx 0.0 \)) had been selected.

**Complexity Analysis** Algorithm 1 is shown to run in worst case linear time (in the number of records \( n \)) theoretically. In practice, however, multiple optimizations make it much faster.

Consider the pseudocode of Algorithm 1. First, TFIDF statistics must be collected for each tuple, which requires a single pass over \( R \) and hence, takes \( O(n) \) time (line 4). Next, for each field, \( R \) must be blocked, which takes total of \( O(mn) \) time where \( m \) is the number of fields in \( R \) (lines 5-6). This step presents an opportunity for parallelism: since each field is processed independent of the others, up to \( m \) processors can be used to speed up the process further, with ‘broad’ databases, (\( m \) at least in the tens) witnessing significant savings.
Algorithm 2 FisherScore($P_f, N_f, i$)

Input:
- Set of positively labeled feature vectors $P_f$
- Set of negatively labeled feature vectors $N_f$
- Conjunction of one or more feature indices $i$

Output:
- Fisher Score $\rho_i$ of feature $i$ with respect to $P_f$ and $N_f$

Method:
1. Compute mean $\mu_{p,i}$ of feature $i$ in $P_f$
2. Compute variance $\sigma_{p,i}^2$ of feature $i$ in $P_f$
3. Compute mean $\mu_{n,i}$ of feature $i$ in $N_f$
4. Compute variance $\sigma_{n,i}^2$ of feature $i$ in $N_f$
5. Compute mean $\mu_i$ of feature $i$ in $P_f \cup N_f$
6. Compute Fisher Score of $\rho_i$ of $i$
7. where $\rho_i := \frac{|P_f|((\mu_{p,i} - \mu_i)^2 + |N_f|((\mu_{n,i} - \mu_i)^2}{|P_f|\sigma_{p,i}^2 + |N_f|\sigma_{n,i}^2}$
8. Return $\rho_i$ computed above

Once the blocks are generated, a window of size $c$ is slid over the block and all pairs within this window generated (lines 7-8). Specifically, for a block with $b$ tuples, the total number of pairs generated is $b(b - 1)/2$ iff $b \leq c$ and $(b - c + 1)(c - 1) + (c - 2)(c - 1)/2 = O(b)$ otherwise. Note that the former achieves its maximum (and the latter its minimum) at $b = c$. Since $c$ is fixed, the pair generation step can take at most $\sum b = O(n)$ time/field, with total time $O(mn)$ over all fields. Given the TFIDF statistics collected, the complexity of computing $\text{sim}$ between generated pairs is $O(1)$ time/pair, with a mere lookup required when the pair is first generated, so that lines 7-8 and lines 9-10 are essentially parallelized.

Adding the above, a worst case complexity of $O(n + mn + mn) = O((2m+1)n)$ is obtained. Note that the sliding window $c$ scheme was crucial for obtaining this guarantee. If instead, all pairs within all blocks were generated, a worst case complexity of $O(n^2)$ would have been obtained, if a single large block happened to span the entire dataset ($b \approx n$ for some block). $c$, therefore, mitigates the adverse effects of large blocks at a price of potentially missing some pairs. In the Experiments section, $c$ is varied and we show that small values ($c = 20$) suffice in practice.

A.4 Feature Selection

Given a weakly labeled set, feature vectors are now extracted by applying all the given specific blocking predicates on each duplicate (or non-duplicate) and storing the boolean result of each predicate as an element in a feature vector.
Algorithm 3 FisherDisjunctive($P_f, N_f, \epsilon, \eta$)

Input :
- Set of positively labeled feature vectors $P_f$
- Set of negatively labeled feature vectors $N_f$
- Maximum positive vectors that may be left uncovered $\epsilon$
- Maximum negative vectors any feature is allowed to cover $\eta$

Output :
- Disjunctive Blocking Scheme $f_P^{Disj}$

Method :
1. Initialize set $f_P^{Disj}$ to be empty
2. Initialize list of valid features $K$ to be empty
3. for all features $i$ do
   - if feature $i$ covers fewer than $\eta$ examples in $N_f$ then
     - Add $i$ to $K$
   end if
4. end for
5. if Disjunction of all features in $K$ covers fewer than $\epsilon$ vectors in $P_f$ then
   - Terminate with message Cannot find blocking scheme under specified parameters
6. end if
7. for all $i$ in $K$ do
   - Call FisherScore($P_f, N_f, i$)
7. end for
8. Sort features in $K$ in descending order of their scores as computed above
9. while more than $\epsilon$ vectors in $P_f$ uncovered do
   - Let $i$ be feature in $K$ with highest Fisher Score
   - Remove $i$ from $K$
   - if $i$ covers at least 1 new vector in $P_f$ compared to $f_P^{Disj}$ then
     - Add $i$ to $f_P^{Disj}$
   end if
10. end while
11. Return disjunction of elements in $f_P^{Disj}$
Algorithm 4 Terms($P_f$, $N_f$, $k$)

Input:
- Set of positively labeled feature vectors $P_f$
- Set of negatively labeled feature vectors $N_f$
- Maximum predicates in a term $k$

Output:
- Set of positively labeled feature vectors $P'_f$
- Set of negatively labeled feature vectors $N'_f$

Method:
1. Initialize $count := 0$
2. Initialize forbidden feature set $forbidden = \emptyset$
3. Initialize $P'_f$ and $N'_f$ to be $P_f$ and $N_f$ respectively
4. while $count < k$ do
   for all features $i$ not in $forbidden$ do
      Compute Fisher Scores for $i$ by calling FisherScore($P'_f$, $N'_f$, $i$)
   end for
   Compute average score $avg$ by averaging all feature scores in the previous step
   for all features $i \notin forbidden$ in decreasing order of scores do
      for all original features $j$ such that $j \neq i$ do
         Form new feature $l$ that is conjunction of $j$ and $i$
         Evaluate score of $l$ by calling FisherScore($P'_f$, $N'_f$, $l$)
         if score of $l$ is not less than $avg$ then
            Add $l$ as new feature to each vector in $P'_f$ and $N'_f$
         end if
      end for
      Add $i$ to $forbidden$
   end for
   Increment $count$
5. end while
6. Return $P'_f$ and $N'_f$
Thus, a set of boolean feature vectors is obtained for the duplicate set, and another set for the non-duplicate set.

The Fisher discrimination criterion [33] is then used to determine the best features using Algorithm 2. The pseudocode of the algorithm, FisherDisjunctive, is given in Algorithm 3. Fisher scores are used in the algorithm to impose an ordering in which eligible feature elements are considered. A feature is ineligible if it covers more than $\eta$ negative examples. In order of descending scores of eligible features, each feature’s contribution is evaluated. If the feature covers positive examples that the current disjunction does not cover, it is added to the current disjunction. This step of adding features continues till at most $\epsilon$ positive examples remain uncovered. When the while loop in line 11 finally terminates, the algorithm is guaranteed to return a disjunctive scheme satisfying this constraint, since otherwise, line 5 would have returned an error.

For learning DNF blocking schemes, a similar approach to the baseline is adopted. Specifically, instead of designing a completely new algorithm, the original feature vectors are supplemented with new features, and the expanded feature vectors input to FisherDisjunctive. Each new feature corresponds to a term or conjunction of the original features. The disjunction returned by FisherDisjunctive then is not necessarily a single clause but a disjunction of terms, corresponding to a DNF blocking scheme as defined earlier.

Naively supplementing the feature set with all possible conjunctions of arbitrary length would lead to an exponential blowup. Bilenko et al. [5] addressed this by taking an extra input parameter $k$ and greedily composing terms with at most $k$ literals. Algorithm 4 also requires a parameter $k$ but adopts a slightly different (although still greedy) heuristic, to compose terms with (at most) $k$ literals. The output, a set of supplemented vectors, may then be used to learn a DNF blocking scheme.

A.5 Experiments

This section details the methodology of the experiments and the benchmarks used.

**Benchmarks** Three benchmark datasets were used to validate the algorithms proposed in this paper. All three sets are commonly employed in the record linkage and blocking key literature [17], and can be found in the SecondString\textsuperscript{30} toolkit. The first dataset, Restaurant, comprises tuples from the Fodor and Zagat restaurant guides. The second dataset, Cora contains bibliographic citations of machine learning publications extracted from the ‘Cora’ search engine. Finally, Census contains records generated by the US Census Bureau based on real census data. Details about the datasets are provided in Table 6. Note that Restaurant and Cora fall under the ‘Deduplication’ category because they only contain one relation each. Census falls under the ‘Linkage’ category because it contains

\textsuperscript{30} http://secondstring.sourceforge.net/
two relations, albeit with the same schema (with 449 and 392 tuples each; see Table 6). The linkage task involves finding duplicate pairs with one tuple coming from the first relation, and one from the second relation. The procedures and algorithms described earlier work identically but with the explicit constraint that no generated pair should contain both tuples from the same relation.

Table 6. Benchmark datasets used in experiments

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Task</th>
<th>Tuples</th>
<th>True duplicate pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant</td>
<td>Deduplication</td>
<td>864</td>
<td>112</td>
</tr>
<tr>
<td>Cora</td>
<td>Deduplication</td>
<td>1295</td>
<td>17184</td>
</tr>
<tr>
<td>Census</td>
<td>Linkage</td>
<td>449+392</td>
<td>327</td>
</tr>
</tbody>
</table>

**Methodology: WeakTrainingSet** To evaluate WeakTrainingSet, the algorithm was run on all three datasets. In a first set of experiments, the maximum number of required duplicates \( d \) and non-duplicates \( nd \) were varied and the resulting precisions against true duplicates and non-duplicates reported. The other parameters \( ut, lt, c \) were set at default values of 0.05, 0.01 and 20 respectively. These were determined by some initial tuning on the Restaurant dataset, and used, unchanged, for the other sets.

However, in practice, such parameter tuning may not be feasible. Therefore, a set of experiments was conducted to show the robustness of this algorithm to reasonable changes in these parameter values. \( ut, lt \) and \( c \) were varied individually across a non-trivial range and the resulting duplicate and non-duplicate precisions reported.

**Methodology: Feature Selection** Supervised ApproxRBSetCover was adopted as the baseline. The weakly labeled duplicates and non-duplicates from WeakTrainingSet are used as input to the feature selection algorithms. Since the baseline is supervised, an equal number of manually labeled duplicates and non-duplicates are provided to the baseline algorithms. However, while WeakTrainingSet can generate both false positives and negatives, the supervised baseline is given a training set that is perfectly labeled. For Restaurant and Census, 56 and 163 duplicates were input (50 percent of the true positives). For Cora, a different approach was required because under the given parameters, WeakTrainingSet would only return a maximum of 3063 duplicates for the default value of the upper threshold parameter, \( ut = 0.05 \). This is just 18 percent of the training set but is still fairly large in the absolute number of duplicates. The number of non-duplicates input was fixed at 1000 for all three datasets. A larger number of non-duplicates offered little extra discriminative power and, experimentally, showed no improvement for either algorithm.
This leads to the choice of $\eta$ and $\epsilon$ for both baseline and feature selection. For convenience, $\epsilon$ and $\eta$ are henceforth expressed as percentages of the duplicate and non-duplicate training set respectively, to ensure uniform comparisons across all three benchmarks.

Cast as percentage, $\epsilon$ is varied from 0.0 to 1.0 in increments of 0.1 while four values of $\eta = 0.0, 0.1, 0.5, 1.0$ are considered. Average and best achieved results across these 44 parameter settings are reported on all three datasets for $FM$, together with corresponding $RR$ and $PC$ numbers. Note that the average result indicates the expected value of the metrics over the parameter space. We chose to report this result because often, in practice, it will not be possible to determine the best values of $\eta$ and $\epsilon$. Moreover, only one run may be allowed if the dataset is too large. Such an expected value, therefore, has relevance in this problem context.

Separate results are shown for learning Disjunctive and DNF Blocking Schemes. For the baseline, training data was randomly sampled; all results are reported as averages of ten independent runs. For learning DNF Blocking Schemes, the parameter $k$ was set to 2 for both baseline and Terms. Increasing $k$ beyond 2 showed no experimental improvements for either method.

We use the same set of 25 general blocking predicates as the baseline to enable fair comparison. These predicates are fairly intuitive and include blocking on exact match of field values, token based match, and blocking on common integer values, among others. Details about these predicates are provided by Bilenko et al. [5]. Recall that for a given relation $R$ with $m$ fields and $s$ general blocking predicates, the number of specific blocking predicates was $ms$ with $s = 25$ here.

A final point is that none of the results were amenable to statistical significance tests. The only non-deterministic component in the procedures described was the random sampling (over ten runs) of the labeled set on which the baseline was trained. An analysis showed that the same blocking scheme was discovered over the ten runs for a given set of parameters. The coarse granularity of blocking schemes compensated for this random sampling, therefore, making the baseline experimentally deterministic.

A.6 Results and Discussion

**WeakTrainingSet** Figure 22 shows the precision of duplicates obtained on Restaurant and Census, against the number of duplicates retrieved by the algorithm. Because the number of duplicates in Cora is an order of magnitude larger than those in Restaurant and Census, it is shown separately in Figure 23. For no dataset does the precision dip below 90 percent, even as the number of duplicates requested/retrieved keeps increasing.

The precisions of non-duplicates retrieved for Cora are shown in Figure 24. For Census and Restaurant, perfect precision (100 percent) was achieved over a range of 1000 to 20000 non-duplicates required/retrieved. The figures for these sets are therefore omitted for lack of space.

In the next experiment, the upper threshold parameter $ut$ was varied from 0.0 to 0.1 in increments of 0.005. Setting $ut$ above 0.1 or below 0.04 is not typically
recommended, otherwise too few duplicates or too many false duplicates will be retrieved respectively. However, the algorithm performs quite well over this range, with accuracy almost always above 90 percent with a non-trivial number of duplicates requested; see Figure 25. Note that for Restaurant, the total number of true duplicate pairs is 112, although 200 are requested. Even at low levels of \( \alpha t \), the algorithm did not return more than 100 pairs. The algorithm, hence, is robust to requests that overestimate.

The same experiment is run again with varying \( \alpha t \) and the precision of non-duplicates is reported. The results are shown in Figure 26. Again, the algorithm performs well as \( \alpha t \) is varied.

A final set of experiments were run varying the window parameter \( c \) from 20 to 50 in increments of 10. The same experiments described above were run. There was no difference at all in any of the results. Hence, setting \( c \) to a low constant such as 20 is justified.

**Feature Selection**

The results for the best and average F-Measure, RR and PC for learning both Disjunctive and DNF Blocking Schemes are shown in Tables 7 and 8. Usually, Fisher Discrimination manages to at least equal the supervised baseline, with improvements most apparent on Cora, the largest (in both fields and tuples) and most complicated of our benchmarks. Also, the improvements usually manifest in the average, rather than the best, case. This indicates that tuning either \( \eta \) and \( \epsilon \) is often not necessary for Fisher discrimination. A closer analysis of the results confirms their stability across the parameter space. For the baseline, on the other hand, many values of the parameters resulted in a failure of the algorithm, particularly for high levels of recall. Parameter tuning appears to be a necessity as the dataset gets noisier.

More importantly, small percentage improvements are significant in this context. For example, Cora contains 1295 tuples and 837,865 pairs. A single percent RR improvement means not sending 8300 pairs through expensive (subsequent) record linkage. Industry scale and web databases can be orders of magnitude larger. Table IV shows a nine percent RR improvement on Cora, when learning DNF schemes.

Conversely, recall (or PC) can be a bigger concern in other applications, especially with sparse duplicates. In this case also, Fisher performs as well, and sometimes much better (> 25%; see Table 7 for Cora).

Additionally, Fisher was compared against an unsupervised baseline by providing the same weak samples to both (Table 9). The improvements remain; however, the best baseline results are not much worse compared to the supervised version. Thus, *WeakTrainingSet* may potentially be employed (standalone) for building or supplementing a training set in record linkage.

Moreover, once the noisy (or even perfectly labeled) training set is cast as a set of feature vectors, it becomes amenable to many different feature selection and clustering methods. We chose a relatively simple Fisher Discrimination criterion for this work, but more powerful criteria could potentially offer better performance. Casting blocking scheme learning as a generic feature selection
### Table 7. Results: Learning Disjunctive Blocking Schemes

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fisher</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR</td>
<td>PC</td>
</tr>
<tr>
<td>Restaurant</td>
<td>Average</td>
<td>0.9971</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9800</td>
</tr>
<tr>
<td>Cora</td>
<td>Average</td>
<td>0.9108</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9330</td>
</tr>
<tr>
<td>Census</td>
<td>Average</td>
<td>0.9910</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9916</td>
</tr>
</tbody>
</table>

### Table 8. Results: Learning DNF Blocking Schemes

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fisher</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR</td>
<td>PC</td>
</tr>
<tr>
<td>Restaurant</td>
<td>Average</td>
<td>0.9993</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9993</td>
</tr>
<tr>
<td>Cora</td>
<td>Average</td>
<td>0.8909</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9330</td>
</tr>
<tr>
<td>Census</td>
<td>Average</td>
<td>0.9910</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9916</td>
</tr>
</tbody>
</table>

### Table 9. Results: Learning DNF Blocking Schemes II

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fisher</th>
<th>Unsupervised Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR</td>
<td>PC</td>
</tr>
<tr>
<td>Restaurant</td>
<td>Average</td>
<td>0.9993</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9993</td>
</tr>
<tr>
<td>Cora</td>
<td>Average</td>
<td>0.8909</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9330</td>
</tr>
<tr>
<td>Census</td>
<td>Average</td>
<td>0.9910</td>
</tr>
<tr>
<td></td>
<td>Best</td>
<td>0.9916</td>
</tr>
</tbody>
</table>

problem, therefore, allows one to utilize an existing body of rich research for solving a hard problem [32].

### A.7 Related Work

To the best of our knowledge, there are only two works that explicitly address the learning of blocking schemes [60],[5]. The method developed by Bilenko et al. [5] was used as our baseline and has been described in detail; Michelson and Knoblock [60] develop a sequential set covering algorithm that adopts a greedy heuristic similar to the baseline. However, given a blocking scheme, there has been much research on refining the blocking step itself. Examples include the bi-gram indexing method, and the sorted neighborhood method [35], with a good survey by Christen [17].
The concept of generating weakly labeled training data has also been investigated in prior work, although to the best of our knowledge, only one work has attempted to incorporate it into record linkage [7]. Moreover, this work only developed an unsupervised method for generating weak non-duplicates. Otherwise, generating artificial training data has been explored in machine learning to generate diverse learners, for instance [58]. The algorithms presented in this paper also draw heavily on feature selection. To the best of our knowledge, this work is the only one to cast blocking scheme discovery as a feature selection problem. A good survey of feature selection is provided by Guyon and Elisseeff [32].

A.8 Future Work and Conclusion

In this paper, an unsupervised framework for learning good blocking schemes was presented and experimentally validated on real-world benchmarks. The al-
Fig. 24. The precision of non-duplicates retrieved by *WeakTrainingSet* on Cora. $ut = 0.05$, $lt = 0.01$ and $c = 20$. Census and Restaurant achieved perfect accuracy.

Fig. 25. Precision of duplicates retrieved as $ut$ is varied. 200 duplicates were requested from Restaurant and Census, and 5000 from Cora. The number actually retrieved depended inversely on the magnitude of $ut$.

gorithms presented were found to present favorable results compared to a supervised state-of-the-art algorithm. Future research includes adapting *WeakTrainingSet* for completely automated record linkage, since an unsupervised record linkage system may be transparently integrated with many cloud workflows. This suggests, in turn, implementations inside a MapReduce framework. The MapReduce framework has garnered much attention since the paper by Google researchers [23]. Adapting our algorithms to a single MapReduce job presents challenges left for future work. Finally, much research has been done in the machine learning community on feature selection. Incorporating more sophisticated feature selection in our algorithms is an interesting opportunity for future investigation.
B A DNF Blocking Scheme Learner for Heterogeneous Datasets

Abstract. Entity Resolution concerns identifying co-referent entity pairs across datasets. A typical workflow comprises two steps. In the first step, a blocking method uses a one-many function called a blocking scheme to map entities to blocks. In the second step, entities sharing a block are paired and compared. Current DNF blocking scheme learners (DNF-BSLs) apply only to structurally homogeneous tables. We present an unsupervised algorithmic pipeline for learning DNF blocking schemes on RDF graph datasets, as well as structurally heterogeneous tables. Previous DNF-BSLs are admitted as special cases. We evaluate the pipeline on six real-world dataset pairs. Unsupervised results are shown to be competitive with supervised and semi-supervised baselines. To the best of our knowledge, this is the first unsupervised DNF-BSL that admits RDF graphs and structurally heterogeneous tables as inputs.

B.1 Introduction

Entity Resolution (ER) is the identification of co-referent entities across datasets. Different communities refer to it as instance matching, record linkage and the merge-purge problem [27], [30]. Scalability indicates a two-step solution [27]. The first step, blocking, mitigates brute-force pairwise comparisons on all entities by clustering entities into blocks and then comparing pairs of entities only within blocks [17]. Blocking results in the selection of a small subset of pairs, called the candidate set, which is input to a second step to determine co-reference using a sophisticated similarity function. We exclusively address blocking in this work.

Blocking methods use a blocking scheme to assign entities to blocks. Recently, Disjunctive Normal Form (DNF) Blocking Scheme Learners (BSLs) were pro-
posed to learn DNF blocking schemes using supervised [5], [60], semi-supervised [12] or unsupervised machine learning techniques [42].

DNF-BSLs operate in an expressive hypothesis space and have shown excellent empirical performance [5]. Despite their advantages, current DNF-BSLs assume that input datasets are tabular and have the same schemas. The latter assumption is often denoted as structural homogeneity [27].

The assumption of tabular structural homogeneity restricts application of DNF-BSLs to other data models. Recent growth of graph datasets and Linked Open Data (LOD) [10] motivates the development of a DNF-BSL for the RDF data model. These graphs are often published by independent sources and are heterogeneous [82].

In this paper, we present a generic algorithmic pipeline for learning DNF blocking schemes on pairs of RDF datasets. The formalism of DNF blocking schemes relies on the existence of a schema. RDF datasets on LOD may not have accompanying schemas [10]. Instead of relying on possibly unavailable metadata, we build a dynamic schema using the properties in the RDF dataset. The RDF dataset can then be logically represented as a property table, which may be populated at run-time. Previously, property tables were defined as physical data structures used in the implementation of triplestores [84]. Using a logical property table representation admits application of a DNF-BSL to RDF datasets.

As a special case, the pipeline also admits structurally heterogeneous tabular inputs. That is, the pipeline can be applied to tabular datasets with different schemas. Thus, previous DNF-BSLs become special cases of the pipeline, since they learn schemes on tables with the same schemas. As a second special case, the pipeline accommodates RDF-tabular heterogeneity, with one input, RDF, and the other, tabular. RDF-tabular heterogeneity applies when linking datasets between LOD and the relational Deep Web [34], [10].

Much existing ER research bypasses the issue of structural heterogeneity by assuming that schema matching is physically undertaken prior to executing an ER algorithm [27]. Schema matching itself is an active, challenging research problem [1, 31]. Instead of relying on the unrealistic assumption of perfect schema reconciliation, the pipeline admits possibly noisy schema mappings from an existing schema matcher, and directly uses the mappings to learn a DNF blocking scheme. If a matcher is unavailable, the pipeline allows for a fall-back option.

We show an unsupervised instantiation of the generic pipeline by strategically employing an existing instance-based schema matcher called Dumas [9]. In addition to schema mappings, Dumas also generates noisy duplicates, which we recycle and pipe to a new DNF-BSL. The new DNF-BSL uses only two parameters, which may be robustly tuned for good empirical performance across multiple domains. The BSL is also shown to have a strong theoretical guarantee.

We evaluate the full unsupervised pipeline on six real-world dataset pairs against two extended baselines, one of which is supervised [5], [42]. Our system performance is found to be competitive, despite training samples not being manually provided. Given that unsupervised methods exist for the second ER

---

51 Resource Description Framework
step in both the Semantic Web and relational communities [30], [27], this first unsupervised heterogeneous DNF-BSL enables, in principle, fully unsupervised ER in both communities.

Section B.2 describes some related work in the area, followed by preliminaries in Section B.3. Section B.4 describes the generic pipeline, followed by an unsupervised instantiation in Section B.7. Section B.8 describes the experiments, with results discussed in Section B.9. The paper concludes in Section B.10.

B.2 Related Work

Entity Resolution was comprehensively surveyed by Elmagarmid et al. [27], with a generic approach represented by Swoosh [3]. Separately, blocking has witnessed much specific research, with Christen surveying blocking methods [17]. Bilenko et al. [5], and Michelson and Knoblock [60] independently proposed supervised DNF-BSLs in 2006. Since then, a semi-supervised adaptation of the BSL proposed by Bilenko et al. has been published [5], [12], as well as an unsupervised system [42]. The four systems assume structural homogeneity. We discuss their core principles in Section B.6.

Heterogeneous blocking may be performed without learning a DNF scheme. One example is Locality Sensitive Hashing (LSH), employed by the Harra system, for instance [44]. LSH is promising but applies only to specific distance measures, like Jaccard and cosine. The Typifier system by Ma et al. is another recent example that relies on type inferencing and was designed for Web data published without semantic types [52]. In contrast, DNF-BSLs can be applied generally, with multiple studies showing strong empirical performance [60], [5], [12], [42]. Finally, although related, clustering is technically treated separately from blocking in the literature [17].

In the Semantic Web, ER is simultaneously known as instance matching and link discovery, and has been surveyed by Ferraram et al. [30]. Existing works restrict inputs to RDF. Also, most techniques in the Semantic Web do not learn schemes, but instead present graph-based blocking methods, a good example being Silk [82].

Finally, the framework in this paper also relies on schema mapping. Schema mapping is an active research area, with a good survey provided by Bellahsene et al. [1]. Gal notes that it is a difficult problem [31].

Schema matchers may return 1:1 or n:m mappings (or even 1:n and n:1). An instance-based schema matcher relies on data instances to perform schema matching [1]. A good example is Dumas [9], which relies on an inexpensive duplicates generator to perform unsupervised schema matching [9]. We describe Dumas in Section B.7.

The property table representation used in this paper is a physically implemented data structure in the Jena triplestore API [84]. In this paper, it is used as a logical data structure. We note that the concept of logically representing one data model as another has precedent. In particular, the literature abounds with
proposed methods on how to integrate relational databases (RDB) with the Semantic Web. Sahoo et al. extensively surveyed this topic, called RDB2RDF [74]. A use-case is the Ultrawrap architecture, which utilizes RDB2RDF to enable real-time Ontology-based Data Access or OBDA [77]. We effectively tackle the inverse problem by translating an RDF graph to a logical property table. This is the first application to devise such an inverse translation for heterogeneous ER.

B.3 Preliminaries

We present definitions and examples to place the remainder of the work in context. Consider a pair of datasets $R_1$ and $R_2$. Each dataset individually conforms to either the RDF or tabular data model. An RDF dataset may be visualized as a directed graph or equivalently, as a set of triples. A triple is a 3-tuple of the form $(\text{subject}, \text{property}, \text{object})$. A tabular dataset conforms to a tabular schema, which is the table name followed by a set of fields. The dataset instance is a set of tuples, with each tuple comprising field values.

Example 10. Dataset 1 (in Figure 27) is an RDF dataset visualized as a directed graph $G = (V, E)$, and can be equivalently represented as a set of $|E|$ triples. For example, $(\text{Mickey Beats}, \text{hasWife}, \text{Joan Beats})$ would be one such triple in the triples representation. Datasets 2 and 3 are tabular dataset examples, with the former having schema $\text{Emergency Contact(Name, Contact, Relation)}$. The first tuple of Dataset 2 has field values Mickey Beats, Joan Beats and Spouse respectively. The keyword null is reserved.

According to the RDF specification, subjects and properties must necessarily be Uniform Resource Identifiers (URIs), while an object node may either be a URI or a literal. URI elements in RDF files typically have associated names (or labels), obtained through dereference. For ease of exposition, we henceforth refer to every URI element in an RDF file by its associated name. Note also, that in the most general case, RDF datasets do not have to conform to any schema. This is why they are commonly visualized as semi-structured datasets, and not as tables. In Section B.5, we show how to dynamically build a property schema and logically represent RDF as a tabular data structure.

An entity is defined as a semantically distinct subject node in an RDF dataset, or as a (semantically distinct) tuple in a tabular dataset. The entity referring to Mickey Beats is shown in red in all datasets in Figure 27. Entity Resolution is the process of resolving semantically equivalent (but possibly syntactically different) entities. As earlier described, ER is traditionally conducted on tables with the same schemas, or on one dataset. An example would be identifying that the two highlighted tuples in Dataset 3 are duplicates.

33 Alternatively called predicate; we uniformly use property
34 http://www.w3.org/RDF/
35 Commonly just called deduplication, if this is the case [27]
Fig. 27. Three example datasets exhibiting various kinds of heterogeneity.

In the Semantic Web, ER is operationalized by connecting two equivalent entities with an `owl:sameAs` property edge. For example, the two nodes referring to Mickey Beats in Dataset 1 should be connected using an `owl:sameAs` edge. Easy operationalizing of ER (and more generally, `link specification` [82]) explains in part the recent interest in ER in the Semantic Web [30]. In the relational setting, ER is traditionally operationalized through joins or mediated schemas. It is less evident how to operationalize ER across RDF-tabular inputs, such as linking Datasets 1 and 2. We return to this issue in Section B.5.

To introduce the current notion of DNF blocking schemes, tabular structural homogeneity is assumed for the remainder of this section. In later sections, we generalize the concepts.

The most basic elements of a blocking scheme are indexing functions $h_i(x_t)$ [5]. An indexing function accepts a field value from a tuple as input and returns a set $Y$ that contains 0 or more blocking key values (BKV). A BKV identifies a block in which the tuple is placed. Intuitively, one may think of a block as a hash bucket, except that blocking is one-many while hashing is typically many-one [17]. For example, if $Y$ contains multiple BKVs, a tuple is placed in multiple blocks.

**Definition 9.** An indexing function $h_i : \text{Dom}(h_i) \rightarrow U^*$ takes as input a field value $x_t$ from some tuple $t$ and returns a set $Y$ that contains 0 or more Blocking Key Values (BKV) from the set of all possible BKVs $U^*$.

The domain $\text{Dom}(h_i)$ is usually just the `string` datatype. The range is a set of BKVs that the tuple is assigned to. Each BKV is represented by a string identifier.

---

36 http://www.w3.org/TR/owl-ref/
Example 11. An example of an indexing function is Tokens. When applied to the Last Name field value of the fourth tuple in Dataset 3, the output set $Y$ is \{W., Beats, Jr.\}.

This leads to the notion of a general blocking predicate (GBP). Intuitively, a GBP $p_i(x_{t_1}, x_{t_2})$ takes as input field values from two tuples, $t_1$ and $t_2$, and uses the $i^{th}$ indexing function to obtain BKV sets $Y_1$ and $Y_2$ for the two arguments. The predicate is satisfied iff $Y_1$ and $Y_2$ share elements, or equivalently, if $t_1$ and $t_2$ have a block in common.

**Definition 10.** A general blocking predicate $p_i : \text{Dom}(h_i) \times \text{Dom}(h_i) \rightarrow \{\text{True, False}\}$ takes as input field values $x_{t_1}$ and $x_{t_2}$ from two tuples, $t_1$ and $t_2$, and returns True if $h_i(x_{t_1}) \cap h_i(x_{t_2}) \neq \Phi$, and returns False otherwise.

Each GBP is always associated with an indexing function.

Example 12. Consider the GBP ContainsCommonToken, associated with the previously introduced Tokens. Suppose it was input the Last Name field values from the first and fourth tuples in Dataset 3. Since these field values have a token (Beats) in common, the GBP returns True.

A specific blocking predicate (SBP) explicitly pairs a GBP to a specific field.

**Definition 11.** A specific blocking predicate is a pair $(p_i, f)$ where $p_i$ is a general blocking predicate and $f$ is a field. A specific blocking predicate takes two tuples $t_1$ and $t_2$ as arguments and applies $p_i$ to the appropriate field values $f_1$ and $f_2$ from both tuples. A tuple pair is said to be covered if the specific blocking predicate returns True for that pair.

Previous DNF research assumed that all available GBPs can be applied to all fields of the relation \cite{42},\cite{5}, \cite{12}, \cite{60} . Hence, given a relation $R$ with $m$ fields in its schema\footnote{Structural homogeneity implies exactly one input schema, even if there are multiple relational instances}, and $s$ GBPs, the number of SBPs is exactly $ms$. Finally, a DNF blocking scheme is defined as:

**Definition 12.** A DNF blocking scheme $f_P$ is a positive propositional formula constructed in Disjunctive Normal Form (DNF), using a given set $H$ of SBPs as the set of atoms. Additionally, if each term is constrained to comprise at most one atom, the blocking scheme is referred to as disjunctive.

SBPs cannot be negated, since the DNF scheme is a positive formula. A tuple pair is said to be covered if the blocking scheme returns True for that pair. Intuitively, this means that the two constituent tuples share a block. In practice, both duplicate and non-duplicate tuple pairs can end up getting covered, since blocking is just a pre-processing step.

\footnote{A disjunction of terms, where each term is a conjunction of literals}
Example 13. Consider the disjunctive scheme \((\text{ContainsCommonToken, Last Name}) \lor (\text{SameFirstDigit, Zip})\), applied on Dataset 3. While the two tuples referring to Mickey Beats would share a block (with the BKV \(\text{Beats}\)), the non-duplicate tuples referring to Susan and Samuel would also share a block (with the BKV \(6\)). Note also that the first and fourth tuples share more than one block, since they also have BKV \(7\) in common.

Given a blocking scheme, a blocking method would need to map tuples to blocks efficiently. Per Definition 12, a blocking scheme takes a tuple pair as input. In practice, linear-time hash-based techniques are usually applied.

Example 14. To efficiently apply the blocking scheme in the previous example on each individual tuple, tokens from the field value corresponding to field Last Name are extracted, along with the first character from the field value of the Zip field, to obtain the tuple’s set of BKVs. For example, applied to the first tuple of Dataset 3, the BKV set \(\{\text{Beats}, 7\}\) is extracted. An index is maintained, with the BKVs as keys and tuple pointers as values. With \(n\) tuples, traditional blocking computes the blocks in time \(O(n)\) \[17\].

Let the set of generated blocks be \(\Pi\). \(\Pi\) contains sets of the form \(B_v\), where \(B_v\) is the block referred to by the BKV \(v\). The candidate set of pairs \(\Gamma\) is given below:

\[
\Gamma = \bigcup_{B_v \in \Pi} \{(r, s)\}, \forall r, s \in B_v | r \in R_1, s \in R_2
\]  

\(\Gamma\) is precisely the set input to the second step of ER, which classifies each pair as a duplicate, non-duplicate or probable duplicate \[16\]. Blocking should produce a small \(\Gamma\) but with high coverage and density of duplicates. Metrics quantifying these properties are defined in Section B.8.

Finally, schema mapping is utilized in the paper. The formal definition of a mapping is quite technical; the survey by Bellahsene et al. provides a full treatment \[1\]. In this paper, an intuitive understanding of the mapping as a pair of field-sets suffices. For example, \((\{\text{Name}\}, \{\text{First Name, Last Name}\})\) is a 1:n mapping between Datasets 2 and 3. More generally, mappings may be of cardinality n:m. The simplest case is a 1:1 mapping, with singleton components.

B.4 The generic pipeline

Figure 28 (a) shows a schematic of the generic pipeline. Two heterogeneous datasets are initially provided as input, with either dataset being RDF or tabular. If the dataset is RDF, we logically represent it as a property table. We describe the details and advantages of this tabular data structure in Section B.5. The key point to note is that the schema matching module takes two tables as input, regardless of the data model, and outputs a set of schema mappings \(Q\). An extended DNF-BSL accepts \(Q\) and also a training set of duplicates and non-duplicates as input, and learns an extended DNF blocking scheme. The DNF-BSL and the blocking scheme need to be extended because tables are now
Fig. 28. (a) shows the generic pipeline for learning a DNF blocking scheme on heterogeneous datasets, with (b) showing an unsupervised instantiation

structurally heterogeneous, and the Section B.3 formalism does not natively apply. In Section B.6, the formalism is extended to admit structural heterogeneity. Figure 28 (b) shows an unsupervised instantiation of the generic pipeline. We describe the details in Section B.7.

B.5 Property Table Representation

Despite their formal description as sets of triples, RDF files are often physically stored in triplestores as sparse property tables [84]. In this paper, we adapt this table instead as a logical tabular representation of RDF. To enable the logical construction on RDF dataset R, define the property schema \{subject\} \cup \{p|\exists (s, p, o), (s, p, o) \in R\}. In essence, we flatten the graph by assigning each distinct property (or edge label) a corresponding field in this schema, along with an extra subject field. Every distinct subject in the triples-set has exactly one corresponding tuple in the table.

Fig. 29. Property Table representation of Dataset 1 in Figure 27

For example, Figure 29 is the property table representation of Dataset 1 in Figure 27. If a subject does not have a corresponding object value for a given property, the reserved keyword null is entered. If a subject has multiple object values for a property, the values are concatenated using a reserved delimiter (; in Figure 29). Technically, field values now have set semantics, with null representing the empty set. Also, the original set of triples can be losslessly
reconstructed from the property table (and vice versa), making it an information-preserving logical representation.

The physical property table was proposed to eliminate expensive property-property self-joins that occur frequently in SPARQL\textsuperscript{39} queries. For ER, the logical data structure is useful because it allows for a dynamic schema that is resolvable at run-time. Triplestores like Jena allow updating and querying of RDF graphs, despite the underneath tabular representation [84]. If the RDF dataset is already stored in such a triplestore, it would not have to be moved prior to ER. This gives the property table a systems-level advantage.

More importantly, having a schema for an RDF dataset means that we can invoke a suitable schema matcher in the generalized pipeline. As we subsequently show, the extended DNF-BSL in the pipeline requires the datasets to have (possibly different) schemas\textsuperscript{40}. Finally, representing RDF as a table allows us to address RDF-tabular heterogeneity, by reducing it to tabular structural heterogeneity. Traditional ER operations become well-defined for RDF-tabular inputs.

One key advantage of the property schema is that it does not rely on RDF Schema (RDFS) metadata. In practice, this allows us to represent any file on Linked Open Data tabularly, regardless of whether metadata is available.

Even in the simple example of Figure 29, we note that the property table is not without its challenges. It is usually sparse, and it may end up being broad, for RDF datasets with many properties. Furthermore, properties are named using different conventions (for example, the prefix \textit{Has} occurs in all the properties in Figure 29) and could be opaque, depending on the dataset publisher. We show empirically (Section B.8) that the instantiated pipeline (Section B.7) can handle these difficulties.

### B.6 Extending the formalism

The formalism in Section B.3 assumed structural homogeneity. We extend it to accommodate structural heterogeneity. As input, consider two datasets $R_1$ and $R_2$. If either dataset is in RDF, we assume that it is in property table form. Consider the original definition of SBPs in Definition 11. SBPs are associated with a single GBP and a single field, making them amenable only to a single schema. To account for heterogeneity, we extend SBPs by replacing the field input with a mapping. Denote as $A_1$ and $A_2$ the respective sets of fields of datasets $R_1$ and $R_2$. We define simple extended SBPs below:

**Definition 13.** A simple extended specific blocking predicate is a pair $(p_i, m)$ where $p_i$ is a general blocking predicate and $m = (\{f_1\}, \{f_2\})$ is a mapping from a single field $f_1 \in A_1$ to a single field $f_2 \in A_2$. The predicate takes two tuples $t_1$ and $t_2$ as arguments and applies $p_i$ to the field values corresponding to $f_1$ and $f_2$ in the two tuples respectively.

\textsuperscript{39} \url{http://www.w3.org/TR/rdf-sparql-query/}

\textsuperscript{40} Even non-DNF blocking typically assumes this [52]
The correspondence in Definition 13 is denoted as simple since it uses a mapping of the simplest cardinality (1:1). Definition 11 can be reformulated as a special case of Definition 13, with \( A_1 = A_2 \) and \( f_1 = f_2 \).

The SBP semantics are not evident if the mapping cardinality is \( n:m \), \( 1:n \) or \( n:1 \), that is, between two arbitrary field-subsets, \( F_1 \subseteq A_1 \) and \( F_2 \subseteq A_2 \). If we interpret the two sets as representing \( |F_1| \) \( |F_2| \) simple extended SBPs \{\( p_i, \{|f_1|, \{f_2|\} \} | f_1 \in F_1, f_2 \in F_2 \}\).

The interpretation above is motivated by the requirement that an SBP should always return a boolean value. We approach the problem by using the \( n:m \) mappings to construct the set of simple extended SBPs, as shown above. We then use disjunction as a combine operator on all elements of the constructed set to yield a single boolean result. We can then define complex extended SBPs.

**Definition 14.** A complex extended specific blocking predicate is a pair \((p_i, M)\) where \( p_i \) is a general blocking predicate and \( M \) is a mapping from a set \( F_1 \subseteq A_1 \) to a set \( F_2 \subseteq A_2 \). The predicate takes two tuples \( t_1 \) and \( t_2 \) as arguments and applies on them \(|F_1|\) \( |F_2| \) simple extended SBPs \((p_i, m)\), where \( m \) ranges over all 1:1 mappings in the set \({\{(f_1, \{f_2\}) | f_1 \in F_1, f_2 \in F_2 \}}\). The predicate returns the disjunction of these \(|F_1|\) \( |F_2| \) values as the final output.

**Example 15.** Consider the mapping between sets \{Name\} in Dataset 2 and \{First Name, Last Name\} in Dataset 3, in Figure 27. Let the input GBP be ContainsCommonToken. The complex extended SBP corresponding to these inputs would be ContainsCommonToken({Name}, {First Name}) \( \lor \) ContainsCommonToken({Name}, {Last Name}). This complex extended SBP would yield the same result as the simple extended SBP ContainsCommonToken({Name}, {Name}) if a new field called Name is derived from the merging of the First Name and Last Name fields in Dataset 3.

An operator like conjunction is theoretically possible but may prove restrictive when learning practical schemes. The disjunction operator makes complex extended SBPs more expressive than simple extended SBPs, but requires more computation (Section B.6). Evaluating alternate combine operators is left for future work.

Finally, (simple or complex) extended DNF schemes can be defined in a similar vein as Definition 12, using (simple or complex) extended SBPs as atoms. One key advantage of using disjunction as the combine operator in Definition 14 is that, assuming simple extended SBPs as atoms for both simple and complex extended DNF schemes, the scheme remains a positive boolean formula, by virtue of distributivity of conjunction and disjunction.

**Extending Existing DNF-BSLs** Existing DNF-BSLs [5], [60], [42], [12] rely on similar high-level principles, which is to devise an approximation algorithm for solving the NP-hard optimization problem first formalized by Bilenko et al. [5]. The approximation algorithms are different in that they require different parameters and levels of supervision. These are detailed below and summarized in
Table 10. These BSLs were originally designed only for structurally homogeneous tables, with a single field-set $A$. We describe their underlying core principles before describing extensions in keeping with the formalism in Section B.6.

Assume a set of GBPs $G$. The core of all approximation algorithms would first construct a search space of SBPs $H$ by forming the cross product of $G$ and $A$. The goal of the algorithm is to choose a subset $H' \subseteq H$ such that the optimization condition laid out by Bilenko et al. is satisfied, at least approximately [5]. The condition assumes that training sets of duplicates $D$ and non-duplicates $N$ are provided. Intuitively, the condition states that the disjunctive blocking scheme formed by taking the disjunction of SBPs in $H'$ covers (see Definition 12) at least $\epsilon|D|$ duplicates, while covering the minimum possible non-duplicates [5]. Note that $\epsilon$ is a parameter common to all four systems\(^{41}\) in Table 10.

![Fig. 30. An example showing how $H_c$ is formed](image)

In order to learn a DNF scheme (as opposed to just disjunctive), a beam search parameter $k$ (also common to all four systems) is required. This parameter is used to supplement the original set $H$ with terms, to obtain a new set $H_c$. This combinatorial process is demonstrated in Figure 30.

$H$ originally consists of the SBPs $a$, $b$ and $c$. These SBPs cover some tuple pairs (TPs). Suppose $k = 2$. A term of size 2 is formed by checking if any TP is covered by (at least) two SBPs. For example, TP-3 is covered by SBPs $a$ and $b$, and hence, also covered by the term $a \land b$. For $k > 2$, terms from size 2 to size $k$ are recursively added to $H$; the final supplemented set is denoted as $H_c$. Note that for $|H|$ predicates, building $H_c$ takes $O(|H|^k)$ time per TP. Given the exponential dependence on $k$ and diminishing returns, previous results capped $k$ at 2 [42], [5]. If $k = 1$, $H_c = H$.

The set $H' \subseteq H_c$ that is now chosen by the approximation scheme would potentially consist of terms and SBPs, with their disjunction yielding a k-DNF scheme\(^{42}\).

While System 1 only requires $\epsilon$ and $k$ as its parameters, Systems 2 and 4 prune their search spaces by removing all SBPs and terms from $H_c$ that cover more than $\eta|N|$ non-duplicates. Note that this step heuristically improves both\(^{43}\).

---

\(^{41}\) $\epsilon$ was designated as $\text{min\_thresh}$ in the original System 1 paper [60], and $\sigma$ in the System 3 paper [12]

\(^{42}\) A k-DNF formula has at most $k$ literals in each term

\(^{43}\) Since $H_c$ now contains only a few, high-quality elements
quality and run-time. It comes at the risk of failure, since if the search space is pruned excessively (by high $\eta$), it may become impossible to cover at least $\epsilon |D|$ duplicates. Systems 3 and 4 require less supervision but significantly more parameter tuning, given they rely on many more parameters.

Systems 1-3 rely on different Set Covering (SC) variants [19]. All three systems can be extended by constructing a search space of complex extended SBPs using $G$ and the mappings set $Q$, instead of $G$ and a field set $A$. The underlying SC approximations operate in this abstract search space to choose the final set $H'$. An extended DNF scheme is formed by a disjunction of (extended) elements in $H'$.

Table 10. DNF-BSL Systems

<table>
<thead>
<tr>
<th>ID</th>
<th>System</th>
<th>Parameters</th>
<th>Supervision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michelson and Knoblock</td>
<td>$\epsilon, k$</td>
<td>Supervised</td>
</tr>
<tr>
<td>2</td>
<td>Bilenko et al. [5]</td>
<td>$\epsilon, \eta, k$</td>
<td>Supervised</td>
</tr>
<tr>
<td>3</td>
<td>Cao et al. [12]</td>
<td>$s, \epsilon, \tau, \alpha, k$</td>
<td>Semi-sup.</td>
</tr>
<tr>
<td>4</td>
<td>Kejriwal and Miranker [42]</td>
<td>Generator: $c, ut, lt, d, nd$; Learner: $\epsilon, \eta, k$</td>
<td>Unsup.</td>
</tr>
<tr>
<td>5</td>
<td>Algorithm 5 herein</td>
<td>$\kappa, k$</td>
<td>Unsup.</td>
</tr>
</tbody>
</table>

Modifying System 4 is problematic because the system is unsupervised and runs in two phases. The first phase (denoted as generator) generates a noisy training set, and the second phase (denoted learner) performs feature-selection on the noisy set to output a DNF blocking scheme. The feature-selection based learner is similar to Systems 1-3 and can be extended. Unfortunately, the generator explicitly assumes homogeneity, and cannot be directly extended to generate training examples for heterogeneous datasets. This implies that the DNF-BSL component of the proposed generic pipeline in Figure 28(a) cannot be instantiated with an existing unsupervised system.

In the event that a schema matcher (and thus, $Q$) is unavailable in Figure 28(a), we present a fall-back option. Specifically, we build a set $Q$ of all 1:1 mappings, $|Q| = |A_1| \cdot |A_2|$. Recall $A_i$ is the field set of dataset $i$. We denote the constructed set $H$ of SBPs as simple exhaustive. Note that for the set of all mappings ($\approx 2^{|A_1|2^{|A_2|}}$), the constructed set $H$ (denoted complex exhaustive) is not computationally feasible for non-trivial cases. Even the simple exhaustive case is only a fall-back option, since a true set of 1:1 mappings $Q$ would be much smaller\(^{44}\) than this set.

\(^{44}\) At most $\min(|A_1|, |A_2|)$
B.7 An Unsupervised Instantiation

A key question to ask is whether the generic pipeline can be instantiated in an unsupervised fashion. As we showed earlier, existing DNF-BSLs that can be extended require some form of supervision. An unsupervised heterogeneous DNF-BSL is important because, in principle, it enables a fully unsupervised ER workflow in both the relational and Semantic Web communities. As the surveys by Elmagarmid et al. and Ferraram et al. note, unsupervised techniques for the second ER step do exist already [27],[30]. A second motivation is the observation that existing unsupervised and semi-supervised homogeneous DNF-BSLs (Systems 3-4) require considerable parameter tuning. Parameter tuning is increasingly being cited as an important algorithmic issue, in applications ranging from schema matching [49] to generic machine learning [25]. Variety in Big Data implies that algorithm design cannot discount parameter tuning.

We propose an unsupervised instantiation with a new DNF-BSL that requires only two parameters. In Table 10, only the supervised System 1 requires two parameters. The schematic of the unsupervised instantiation (of the generic pipeline in Figure 28(a)) is shown in Figure 28(b). We use the existing schema matcher, Dumas, in the instantiated pipeline [9]. Dumas outputs 1:1 field mappings by first using a duplicates generator to locate tuple pairs with high cosine similarity. In the second step, Dumas uses SoftTFIDF to build a similarity matrix from each generated duplicate. If \( n \) duplicates are input to the second step, \( n \) similarity matrices are built and then averaged into a single similarity matrix. The assignment problem is then solved by invoking the Hungarian Algorithm on this matrix [51]. This results in exactly \( \min(|A_1|,|A_2|) \) 1:1 field mappings (the set \( Q \)) being output.

In addition to using \( Q \), we recycle the noisy duplicates of Dumas and pipe them into Algorithm 5. Note that Dumas does not generate non-duplicates. We address this issue in a novel way, by permuting the generated duplicates set \( D \). Suppose that \( D \) contains \( n \) tuple pairs \( \{(r_1,s_1),...,(r_n,s_n)\} \), with each \( r,s \) respectively from datasets \( R_1,R_2 \). By randomly permuting the pairs in \( D \), we heuristically obtain non-duplicate pairs of the form \( (r_i,s_j), i \neq j \). Note that (at most) \( n! \) distinct permutations are possible. For balanced supervision, we set \( |N| = |D| \), with \( N \) the permutation-generated set.

Empirically, the permutation is expected to yield a precise \( N \) because of observed duplicates sparsity in ER datasets [17],[42]. This sparsity is also a key tenet underlying the blocking procedure itself. If the datasets were dense in duplicates, blocking would not yield any savings.

Algorithm 5 shows the pseudocode of the extended DNF BSL. Inputs to the algorithm are the piped Dumas outputs, \( D \) and \( Q \). To learn a blocking scheme from these inputs, two parameters \( k \) and \( \kappa \) need to be specified. Similar to (extended) Systems 1-3 in Table 10, \( G,Q \) and \( k \) are used to construct the search space, \( H_c \). Note that \( G \) is considered the algorithm’s feature space, and is not a dataset-dependent input (or parameter). Designing an expressive \( G \) has computational-dependent and qualitative effects, as we empirically demonstrate. We describe the GBP’s included in \( G \) in Section B.8.
Algorithm 5 Learn Extended k-DNF Blocking Scheme

**Input**: Set $D$ of duplicate tuple pairs, Set $Q$ of mappings  
**Parameters**: Beam search parameter $k$, SC-threshold $\kappa$

**Output**: Extended DNF Blocking Scheme $B$

**Method**:  
//Step 0: Construct sets $N$ and $H$  
Permute pairs in $D$ to obtain $N$, $|N| = |D|$

Construct set $H$ of simple extended SBPs using set $G$ of GBPs and $Q$

Supplement set $H$ to get set $H_c$ using $k$

//Step 1: Build Multimaps $M'_D$ and $M'_N$

Construct $M_D = \langle X, H_X \rangle$, $X$ is a tuple pair in $D$, $H_X \subseteq H_c$ contains the elements in $H_c$ covering $X$

Repeat previous step to build $M_N$ for tuple pairs in $N$

Reverse $M_D$ and $M_N$ to respectively get $M'_D$ and $M'_N$

//Step 2: Run approximation algorithm

for all $X \in \text{keyset}(M'_D)$ do

Score $X$ by using formula $|M'_D(X)|/|D| - |M'_N(X)|/|N|$

Remove $X$ if $\text{score}(X) < \kappa$
end for

Perform W-SC on keys in $M'_D$ using Chvatal’s heuristic, weights are negative scores

//Step 3: Construct and output DNF blocking scheme

$B := \text{Disjunction of chosen keys}$

Output $B$

Step 0 in Algorithm 5 is the permutation step just described, to generate the non-duplicates set $N$. $G$ and $Q$ are then used to construct the set $H$ of simple extended\(^{45}\) SBPs (Definition 13), with $|H| = |G||Q|$. $H$ is supplemented (using parameter $k$) to yield $H_c$, as earlier described in Section B.6.

Fig. 31. Step 1 of Algorithm 5, assuming the information in Figure 30

Step 1 constructs multimaps\(^{46}\) on which Set Covering (SC) is eventually run. As a first logical step, multimaps $M_D$ and $M_N$ are constructed. Each tuple pair (TP) in $D$ is a key in $M_D$, with the SBPs and terms in $H_c$ covering that

\(^{45}\)Since Dumas only outputs 1:1 mappings

\(^{46}\)Multimap keys reference multiple values (or a value set)
TP comprising the value set, $M_D$ is then reversed to yield $M_D'$, $M_N'$ is built analogously. Figure 31 demonstrates the procedure, assuming $D$ contains TPs 1-5, covered as shown in Figure 30. The time complexity of building (both) $M_D'$ and $M_N'$ is $O(|H|^k(|D| + |N|))$.

In Step 2, each key is first scored by calculating the difference between the fractions of covered duplicates and non-duplicates. A threshold parameter, $\kappa$, is used to remove the SBPs and terms that have low scores. Intuitively, $\kappa$ tries to balance the conflicting needs of previously described parameters, $\epsilon$ and $\eta$, and reduce tuning effort. The range of $\kappa$ is $[-1,1]$. An advantage of the parameter is that it has an intuitive interpretation. A value close to 1.0 would indicate that the user is confident about low noise-levels in inputs $D$ and $Q$, since high $\kappa$ implies the existence of elements in $H_c$ that cover many positives and few negatives. Since many keys in $M_D'$ are removed by high $\kappa$, this also leads to computational savings. However, setting $\kappa$ too high (perhaps because of misguided user confidence) could potentially lead to excessive purging of $M_D'$, and subsequent algorithm failure. Experimentally, we show that $\kappa$ is easily tunable and even high values of $\kappa$ are robust to noisy inputs.

*Weighted Set Covering* (W-SC) is then performed using Chvatal's algorithm\(^{47}\) [19], with each key in $M_D'$ acting as a set and the tuple pairs covered by all keys as elements of the universe set $U$. For example, assuming all SBPs and terms in the keyset of $M_D'$ in Figure 31 have scores above $\kappa$, $U = \{1,2,3,4,5\}$. Note that only $M_D'$ is pruned (using $\kappa$) and also, W-SC is performed only on $M_D'$. $M_N'$ only aids in the score calculation (and subsequent pruning process) and may be safely purged from memory before W-SC commences.

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 SBPs or terms, their disjunction is the k-DNF blocking scheme.

Under plausible complexity assumptions, Chvatal’s algorithm is essentially the best-known polynomial-time approximation for W-SC [72]. For example, Bilenko et al. used Peleg’s approximation to Red-Blue SC [69], [13], which is known to have worse bounds [13]. The proposed DNF-BSL has the strongest theoretical approximation guarantee of all systems in Table 10.

**B.8 Experiments**

**Metrics** The quality and efficiency of blocking is evaluated by the special metrics, Reduction Ratio (RR), Pairs Completeness (PC) and Pairs Quality (PQ) [17]. Traditional metrics like precision and recall do not apply to blocking since it is a pre-processing step, and its output is not the final ER output. To define RR, which intuitively measures efficiency, consider the full set of pairs $\Omega$ that would be generated in the absence of blocking. Specifically, $\Omega$ is the set $\{(r,s)|r \in R_1, s \in R_2\}$. RR is then given by the formula, $1 - |\Gamma|/|\Omega|$. Given the

\(^{47}\) We include Chvatal’s algorithm in the SC Appendix survey

\(^{48}\) $P \subset NP$
Table 11. Details of dataset pairs. The notation, where applicable, is (first dataset) /× (second dataset)

<table>
<thead>
<tr>
<th>ID</th>
<th>Dataset Pairs</th>
<th>Fields</th>
<th>Total Entity Pairs</th>
<th>Duplicate Pairs</th>
<th>Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Restaurant 1 /Restaurant 2</td>
<td>8/8</td>
<td>339 × 2256 = 764,784</td>
<td>100</td>
<td>RDF/RDF</td>
</tr>
<tr>
<td>2</td>
<td>Persons 1 /Persons 2</td>
<td>14/14</td>
<td>2000 × 1000 = 2 million</td>
<td>500</td>
<td>RDF/RDF</td>
</tr>
<tr>
<td>3</td>
<td>IBM/vgchartz</td>
<td>12/11</td>
<td>1904 × 20,000 ≈ 38 million</td>
<td>3933</td>
<td>Tabular/Tabular</td>
</tr>
<tr>
<td>4</td>
<td>Libraries 1 /Libraries 2</td>
<td>5/10</td>
<td>17,636 × 26,583 ≈ 469 million</td>
<td>16,789</td>
<td>Tabular/Tabular</td>
</tr>
<tr>
<td>5</td>
<td>IBM/DBpedia</td>
<td>12/4</td>
<td>1904 × 16,755 ≈ 32 million</td>
<td>748</td>
<td>Tabular/RDF</td>
</tr>
<tr>
<td>6</td>
<td>vgchartz/DBpedia</td>
<td>11/4</td>
<td>20,000 × 16,755 ≈ 335 million</td>
<td>10,000</td>
<td>Tabular/RDF</td>
</tr>
</tbody>
</table>

sparsity of duplicates in real-world datasets [17], RR should ideally be close to, but not precisely, 1.0 (unless $\Gamma = \phi$).

Denote the set of all duplicate pairs, or the ground-truth, as $\Omega_m$. Similarly, consider the set of duplicates included in the candidate set $\Gamma_m = \Gamma \cap \Omega_m$. The metric PC quantifies the effectiveness of the blocking scheme by measuring coverage of $\Gamma_m$ with respect to $\Omega_m$. Specifically, it is given by the formula, $|\Gamma_m|/|\Omega_m|$. Low PC implies that recall on overall ER will be low, since many duplicates did not share a block to begin with.

PC and RR together express an efficiency-effectiveness tradeoff. The metric PQ is sometimes used to measure how dense the blocks are in duplicates, and is given by $|\Gamma_m|/|\Gamma|$. PQ has not been reported in recent BSL literature [5],[42]. One reason is that PQ can be equivalently expressed as $c.PC/(1 - RR)$, where $c$ is $|\Omega_m|/|\Omega|$. When comparing two BSLs, PQ can therefore be expressed wholly in terms of PC and RR. We do not consider PQ further in this paper.

Datasets The heterogeneous test suite of six dataset pairs (and nine individual datasets) is summarized in Table 11. The suite spans over four domains and the three kinds of heterogeneity discussed in the paper. All datasets are from real-world sources. We did not curate these files in any way, except for serializing RDF datasets as property tables (instead of triples-sets). The serializing was found to be near-instantaneous (¡ 1 second) in all cases, with negligible run-time compared to the rest of the pipeline.

Dataset Pairs (DPs) 1 and 2 are the RDF benchmarks in the 2010 instance-matching track[49] of OAEI[50], an annual Semantic Web initiative. Note that an earlier tabular version of DP 1 is also popular in homogeneous ER literature [17].

[50] Ontology Alignment Evaluation Initiative
DPs 3, 5 and 6 describe video game information. DP 6 has already been used as a test case in a previous schema matching work [80]. vgchartz is a tabular dataset taken from a reputable charting website\textsuperscript{51}. DBpedia contains 48,132 triples extracted from DBpedia\textsuperscript{52}, and has four (three\textsuperscript{53} properties and subject) fields and 16,755 tuples in property table form. Finally, IBM contains user-contributed data extracted from the Many Eyes page\textsuperscript{54}, maintained by IBM Research.

DP 4 describes US libraries. Libraries 1 was from a Point of Interest website\textsuperscript{55}, and Libraries 2 was taken from a US government listing of libraries.

DPs 2 and 4 contain n:m ground-truth schema mappings, while the others only contain 1:1 ground-truth mappings.

**Baselines**: Table 10 listed existing DNF-BSLs, with Section B.6 describing how the extended versions fit into the pipeline. We extend two state-of-the-art systems as baselines:

**Supervised Baseline**: System 2 was chosen as the supervised baseline [5], and favored over System 1 [60] because of better reported empirical performance on a common benchmark employed by both efforts. Tuning the parameter $\eta$ leads to better blocking schemes, making System 2 a state-of-the-art supervised baseline.

**Semi-supervised Baseline**: We adapt System 4 as a semi-supervised baseline by feeding it the same noisy duplicates generated by Dumas as fed to the proposed learner, as well as a manually labeled negative training set. The learner of System 4 uses feature selection and was shown to be empirically competitive with supervised baselines [42]. In contrast, System 3 did not evaluate its results against System 2. Also, the learner of System 4 requires three parameters, versus the five of System 3. Finally, the three System 4 parameters are comparable to the corresponding System 2 parameters, and evaluations in the original work showed that the learner is robust to minor parameter variations [42]. For these reasons, the feature-selection learner of System 4 was extended to serve as a semi-supervised baseline.

**Methodology: Dumas** In this section, we describe experimental methodology. Experimental results will be presented in Section B.9. Given that the test suite is larger and more heterogeneous in this paper than in the original Dumas work [9], we perform two preliminary experiments to evaluate Dumas. Dumas was briefly described in Section B.7.

**Preliminary Experiment 1**: We evaluate the performance of Dumas’s duplicates generator. The generator uses TFIDF to retrieve the highest scoring duplicates in order. Suppose we retrieve $t$ duplicates (as an ordered list). Denote,

\textsuperscript{51} vgchartz.com
\textsuperscript{52} dbpedia.org
\textsuperscript{53} genre, platform and manufacturer
\textsuperscript{54} www-958.ibm.com/software/data/cognos/manyeyes/datasets
\textsuperscript{56} http://www.poi-factory.com/poifiles
for any $k \leq t$, the number of true positives in sub-list $[1 \ldots k]$ as $d(k)$. Define $\text{Precision}_@k$ as $d(k)/k$ and $\text{Recall}_@k$ as $d(k)/|\Omega_m|$, where $\Omega_m$ is the ground-truth set of all duplicates. We plot $\text{Precision}_@k$ against $\text{Recall}_@k$ for all DPs to demonstrate the precision-recall tradeoff. To obtain a full set of data points, we set $t$ at 50,000 for all experiments, and calculate $\text{Precision}_@k$ and $\text{Recall}_@k$ for $k \in \{1 \ldots t\}$.

**Preliminary Experiment 2:** Although $t$ was set to a high value in the previous experiment, Dumas requires only a few top pairs (typically 10-50) for the schema matching step [9]. To compute the similarity matrix from $t$ pairs, Dumas uses $\text{SoftTFIDF}$, which requires an optional threshold $\theta$. For a given DP, denote $Q'$ as the ground-truth set of schema mappings, $Q$ as the set of Dumas mappings, and $Q_m \subseteq Q$ as the set of correct Dumas mappings. Define $\text{precision}$ (on this task) as $|Q_m|/|Q|$ and $\text{recall}$ as $|Q_m|/|Q'|$. In this experiment, we set $t$ and $\theta$ to default values of 50 and 0.5 respectively and report the results. We also vary these parameters and describe performance differences.

**Methodology: DNF BSL Parameter Tuning** We describe parameter tuning methodology. Note that the beam search parameter $k$ (see Table 10) is not technically tuned, but set, incumbent on the experiment (Section B.8).

**Baseline Parameters:** Both baseline parameters are tuned in similar ways. We do an exhaustive parameter sweep of $\epsilon$ and $\eta$, with values in the range of $0.90 - 0.95$ and $0.0 - 0.05$ (respectively) typically maximizing baseline performance. Low $\eta$ maximizes RR, while high $\epsilon$ maximizes PC. Although extreme values, with $\eta = 0$ and $\epsilon = 1$, are expected to maximize performance, they led to failure on all test cases. This demonstrates the necessity of proper parameter tuning. Fewer parameters imply less tuning, and faster system deployment.

**SC-threshold $\kappa$:** The single parameter $\kappa$ of the proposed method was tuned on the smallest test case (DP 1) and found to work best at 0.9. To test the robustness of $\kappa$ to different domains, we fixed it at 0.9 for all experiments.

$|D|$ and $|N|$: We emulate the methodology of Bilenko et al. in setting the numbers of positive and negative samples input to the system. Bilenko et al. input 50% of true positives and an equal number of negatives to train their system [5]. Let this number be $n = |D|, |N|$ for a given test suite. For example, $n = 50$ for DP 1. For fairness, we use these numbers for the semi-supervised baseline and proposed system also. We retrieve the top $n$ pairs from the Dumas generator and input the pairs to both systems as $D$. Additionally, we provide $n$ labeled non-duplicates (as $N$) to the semi-supervised baseline. In subsequent experiments, the dependence of the proposed system on $n$ is tested.

**Methodology: Extended DNF BSL** We describe the evaluation of the extended DNF BSLs.

**Set $G$ of GBPs:** Bilenko et al. first proposed a set $G$ that has since been adopted in future works [5], [12], [42]. This original set contained generic token-based functions, numerical functions and character-based functions [5]. No recent $e|D|$ duplicates could not be covered (see Section B.6)
work attempted to supplement \( G \) with more expressive GBPs. In particular, phonetic GBPs such as Soundex, Metaphone and NYSIIS were not added to \( G \), despite proven performance benefits [16]. We make an empirical contribution by supplementing \( G \) with all nine phonetic features implemented in an open-source package\textsuperscript{57}. For fairness, the same \( G \) is always input to all learners in the experiments below. Results on Experiment 2 will indirectly demonstrate the benefits of supplementing \( G \).

**Experiment 1:** To evaluate the proposed learner against the baselines, we input the same \( Q \) (found in Preliminary Experiment 2) to all three systems. The systems learn the DNF scheme by choosing a subset \( H' \subseteq H_c \) of SBPs and supplemented terms, with \( H_c \) constructed from \( G \), \( Q \) and \( k \) (Section B.6). We learn only disjunctive schemes by setting \( k \) to 1 in this experiment.

**Experiment 2:** We repeat Experiment 1 but with \( k = 2 \). We mentioned in Section B.6 that complexity is exponential in \( k \) for all systems in Table 10. Because of this exponential dependence, it was not possible to run the experiment for \( k = 2 \) on all DPs. We note which DPs presented problems, and why. Note that, if \( G \), \( Q \) and the training sets are fixed, increasing \( k \) seems to be the only feasible way of improving blocking quality. However, \( G \) is more expressive in this paper. Intuitively, we expect the difference across Experiments 1 and 2 to be narrower than in previous work.

**Experiment 3:** In a third set of experiments, we evaluate how blocking performance varies with \( Q \). To the baseline methods, we input the set of (possibly n:m) ground-truth mappings \( Q' \) while the Dumas output \( Q \) is retained for the proposed learner. The goal is to evaluate if extended DNF-BSLs are sensitive, or if noisy 1:1 matchers like Dumas suffice for the end goal.

**Experiment 4:** We report on run-times and show performance variations of the proposed system with the number of provided duplicates, \( n \). In industrial settings with an unknown ground-truth, \( n \) would have to be estimated. An important question is if we can rely on getting good results with constant \( n \), despite DP heterogeneity.

**Statistical Significance:** We conduct experiments in ten runs, and (where relevant) report statistical significance levels using the paired sample Student’s \( t \)-distribution. On blocking metrics, we report if results are not significant (NS), weakly significant (WS), significant (SS) or highly significant (HS), based on whether the p-value falls within brackets \([1.0, 0.1]\), \((0.05,0.1]\), \((0.01, 0.05]\] and \([0.0, 0.01]\] respectively. As for the choice of samples, we always individually paired PC and RR of the proposed system against the baseline that achieved a better average on the metric.

**Implementation:** All programs were implemented in Java on a 32-bit Ubuntu virtual machine with 3385 MB of RAM and a 2.40 GHz Intel 4700MQ i7 processor.

\textbf{B.9 Results and Discussion}

\textsuperscript{57} org.apache.commons.codec.language
Dumas Preliminary Experiment 1: Figure 32 shows the results of Dumas’s duplicates-generator. Except for DP 3, precision on all cases seems inadequate even at low recall levels. Although recall of 100% is eventually attained on most DPs, the price is (near) 0% precision. Closer inspection of the results showed that many false positives got ranked at the top. We discuss the implications of these noisy results shortly.

Table 12. Best Results of Dumas Schema-Matcher

<table>
<thead>
<tr>
<th>Dataset Pair</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100%</td>
<td>87.5%</td>
</tr>
<tr>
<td>2</td>
<td>92.86%</td>
<td>86.67%</td>
</tr>
<tr>
<td>3</td>
<td>91%</td>
<td>100%</td>
</tr>
<tr>
<td>4</td>
<td>75%</td>
<td>33.33%</td>
</tr>
<tr>
<td>5</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Preliminary Experiment 2: Table 12 shows the schema mapping results retrieved from Dumas with parameters set at $t = 50$ and $\theta = 0.5$. These default values were found to maximize performance in all cases, in agreement with similar values in the original work [9]. We also varied $t$ from 10 to 10,000 and $\theta$ from 0 to 0.9. Performance slightly declined (by at most 5%) for some DPs when $t < 50$, but remained otherwise constant across parameter sweeps. This confirms that Dumas is quite robust to $t$ and $\theta$. One disadvantage of Dumas is that it is a 1:1 matcher. This explains the lower recall numbers on DPs 2 and 4, which contain n:m mappings. In Experiment 3, we test if this is problematic by providing the ground-truth set $Q'$ to baselines, and comparing results.

Discussion: An important point to note from the (mostly) good results in Table 12 is that generator accuracy is not always predictive of schema mapping accuracy. This robustness to noise is always an important criteria in pipelines. If noise accumulates at each step, the final results will be qualitatively weak, pos-
ibly meaningless. Since Dumas’s matching component is able to compensate for generator noise, it is a good empirical candidate for unsupervised 1:1 matching on typical ER cases. Preliminary Experiment 1 also shows that on real-world ER problems, a simple measure like TFIDF is not appropriate (by itself) for solving ER. The generator noise provides an interesting test for the extended DNF-BSLs. Since both the mappings-set $Q$ and top $n$ generated duplicates (the set $D$) output by Dumas are piped to the learner, there are two potential sources of noise.

Finally, given that the proposed system (in subsequent experiments) permutes the Dumas-output $D$ (Algorithm 5) to generate $N$, we tested the accuracy of the permutation procedure in a follow-up experiment. With $|D|$ ranging from 50 to 10,000 in increments of 50, we generated $N$ of equal size and calculated non-duplicates accuracy\(^58\) of $N$ over ten random trials per value. In all cases, accuracy was 100%, showing that the permutation heuristic is actually quite strong.

**Extended DNF BSL Experiment 1:** Table 13 shows BSL results on all six DPs. The high overall performance explains the recent popularity of DNF-BSLs. Using the extended DNF hypothesis space for blocking schemes allows the learner to compensate for the two sources of noise earlier discussed. Overall, considering statistically significant results, the supervised method typically achieves better RR, but PC is (mostly) equally high for all methods, with the proposed method performing the best on DP 4 (the largest DP) and the supervised baseline on DP 2, with high significance. We believe the former result was obtained because the proposed method has the strongest approximation bounds out of all three systems, and that this effect would be most apparent on large DPs. Importantly, low standard deviation (often 0) is frequently observed for all methods. The DNF-BSLs prove to be quite deterministic, which can be important when replicating results in both research and industrial settings.

**Experiment 2:** Next, we evaluated if $k = 2$ enhances BSL performance and justifies the exponentially increased cost. With $k = 2$, only DPs 1 and 5 were found computationally feasible. On the other DPs, the program either ran out of RAM (DPs 4,6), or did not terminate after a long\(^59\) time (DPs 2,3). The former was observed because of high $n$ and the latter because of the large number of fields (see Table 11). Setting $k$ beyond 2 was computationally infeasible even for DPs 1 and 5. Furthermore, results on DPs 1 and 5 showed no statistical difference compared to Experiment 1, even though run-times went up by an approximate factor of 16 (for both DPs).

**Experiment 3:** We provided the ground-truth set $Q'$ to baseline methods (and with $k$ again set to 1), while retaining $Q$ for the proposed method. Again, we did not observe any statistically significant difference in PC or RR for either baseline method. We believe this is because the cases for which $Q'$ would most

\(^{58}\) $1 - |N \cap B_m|/|N|$  
\(^{59}\) Within a factor of 20 of the average time taken by the system for the $k = 1$ experiment (for that DP)
Table 13. Comparative Results of Extended DNF BSLs. Bold values are (at least) weakly significant, with significance levels (WS, SS or HS) in paranthesis.

<table>
<thead>
<tr>
<th>Dataset Pair (DP)</th>
<th>Proposed Method</th>
<th>Semi-Supervised Baseline</th>
<th>Supervised Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC</td>
<td>RR</td>
<td>PC</td>
</tr>
<tr>
<td>DP 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>100%</td>
<td>99.68%</td>
<td>100%</td>
</tr>
<tr>
<td>Standard</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>95%</td>
<td>96.11%</td>
<td>95%</td>
</tr>
<tr>
<td>Standard</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>100%</td>
<td>95.47%</td>
<td>100%</td>
</tr>
<tr>
<td>Standard</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>98.95% (HS)</td>
<td>99.68%</td>
<td>98.43%</td>
</tr>
<tr>
<td>Standard</td>
<td>0.1%</td>
<td>0.002%</td>
<td>0%</td>
</tr>
<tr>
<td>Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>100%</td>
<td>92.28%</td>
<td>100%</td>
</tr>
<tr>
<td>Standard</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>99.91%</td>
<td>99.69%</td>
<td>99.97%</td>
</tr>
<tr>
<td>Standard</td>
<td>0.08%</td>
<td>0.019%</td>
<td>0.07%</td>
</tr>
</tbody>
</table>

likely have proved useful (DPs 2 and 4, which contain n:m mappings that Dumas cannot output) already perform well with the Dumas-output Q.

Experiment 4: Theoretically, run-time of Algorithm 5 was shown to be $O(|H|^k(|D| + |N|))$; the run-times of other systems in Table 10 are similar. Empirically, this has never been demonstrated. For $k = 1$, we plot the run-time data collected from Experiment 1 runs on DPs 1-6 (Figure 33(a)). The trend is fairly linear, with the supervised system slower for smaller inputs, but not larger inputs. The dependence on $|Q|$ shows why a schema matcher is necessary, since in its absence, the simple exhaustive set is input (Section B.6).

As further validation of the theoretical run-time, Figure 33(b) shows the linear dependence of the proposed system on $|D|$. Again, the trend is linear, but the slope depends on the schema heterogeneities of the individual DPs. For example, DPs 2 and 3, the largest datasets in terms of fields (Table 11), do not scale as well as the others.

---

60 The slope is the hidden constant in the asymptotic notation.
Fig. 33. Experiment 4 results. (a) plots run-time trends of all three systems against the theoretical formula, while (b) and (c) respectively plot run-times and f-scores of the proposed system against sample sizes.

Figure 33(c) shows an important robustness result. We plot the PC-RR f-score\(^{61}\) of the proposed system against \(|D|\). The first observation is that, even for small \(|D|\), performance is already high except on DP 2, which shows a steep increase when \(|D| \approx 100\). On all cases, maximum performance is achieved at about \(|D| \approx 700\) and the f-score curves flatten subsequently. This is qualitatively similar to the robustness of Dumas to small numbers of positive samples.

Figure 33(c) also shows that the proposed method is robust to overestimates of \(|D|\). DP 1, for example, only has 100 true positives, but it continues to perform well at much higher \(|D|\) (albeit with a slight dip at \(|D| \approx 700\)).

Discussion: We earlier stated, when discussing the preliminary experimental results, that an extended DNF-BSL can only integrate well into the pipeline if it is robust to noise from previous steps. Previous research has noted the overall robustness of DNF-BSLs. This led to the recent emergence of a homogeneous unsupervised system [42], adapted here as a semi-supervised baseline. Experiment 1 results showed that this robustness also carries over to extended DNF-BSLs. High overall performance shows that the pipeline can accommodate heterogeneity, a key goal of this paper.

Experiment 2 results demonstrate the advantage of having an expressive \(G\), which is evidently more viable than increasing \(k\). On DPs 1 and 5 (that the systems succeeded on), no statistically significant differences were observed, despite run-time increasing by a factor of 16. We note that the largest (homogeneous) test cases on which \(k = 2\) schemes were previously evaluated were only about the order of DP 1 (in size). Even with less expressive \(G\), only a few percentage point performance differences were observed (in PC and RR), with statistical significance not reported [42], [5].

To confirm the role of \(G\), we performed a follow-up experiment where we used the originally proposed \(G\) [5] on DPs 1 and 5, with both \(k = 1\) and \(k = 2\). We observed lower performance with \(k = 1\) compared to Table 13 results, while \(k = 2\) results were only at par with those results. Run-times with less expressive

\(^{61}\) \(\text{RR PC} = \frac{\text{RR}}{\text{PC}}\)
$G$ were obviously lower (for corresponding $k$); however, $k = 2$ run-times were higher (with less expressive $G$) than $k = 1$ run-times with more expressive $G$. All differences just described were statistically significant (at the 95% level). This validates previous research findings, while also confirming our stated hypothesis about $G$.

Experiment 3 results showed that a sophisticated schema matcher is not always necessary for the purpose of learning a blocking scheme. However, the importance of good schema matching goes beyond blocking and even ER. Schema matching is an important step in overall data integration [1]. On noisier datasets, a good n:m schema matcher could make all the difference in pipeline performance, but we leave for future work to evaluate such a case.

The similar run-time trends shown by the various systems in Figure 33(a) also explain why, in Experiment 2, all systems simultaneously succeeded or failed on a given DP. Even if we replace our DNF-BSL with an extended version from the literature, the exponential dependence on $k$ remains. Figures 33(a) and (b) also validate theoretical run-time calculations empirically. Previous research on DNF-BSLs did not theoretically analyze (or empirically report) algorithmic runtimes and scalability explicitly [42], [5], [12], [60].

Figure 33(c) demonstrates the encouraging qualitative result that only a few (noisy) samples are typically enough for adequate performance. Given enterprise quality requirements, as well as expense of domain expertise, high performance for low $n$ and minimum parameter tuning is a practical necessity, for industrial deployment. Recall that we retained $\kappa$ at 0.9 for all experiments (after tuning on DP 1), while for the baselines, we had to conduct parameter sweeps for each separate experiment. Combined with results in both Table 13 and Figure 33(c), this shows that the system can be a potential use-case in industry. Combined with previous unsupervised results for the second ER step [30], [27], such a use-case would apply both to relational and Semantic Web data as a fully unsupervised ER workflow, which has thus far remained elusive.

B.10 Conclusion and Future Work

In this paper, we presented a generic pipeline for learning DNF blocking schemes on heterogeneous dataset pairs. We proposed an unsupervised instantiation of the pipeline that relies on an existing instance-based schema matcher and learns blocking schemes using only two parameters. We also showed a novel way of reconciling RDF-tabular heterogeneity by using the logical property table representation for building and populating a dynamic property schema for RDF datasets. Finally, we evaluated all techniques on six test cases exhibiting three separate kinds of heterogeneity.

Future research will address further exploration of the property table representation for tabularly mining RDF data. Additionally, refining $G$ (the set of GBP's) further is a promising, proven method of scalably improving BSL performance. We will also implement a fully unsupervised ER workflow that the proposed unsupervised DNF-BSL enables, and evaluate it in a similar fashion.