Efficient Entity Resolution for Large Heterogeneous Information Spaces

George Papadakis
L3S Research Center
Hannover, Germany
papadakis@L3S.de

Ekaterini Ioannou
L3S Research Center
Hannover, Germany
ioannou@L3S.de

Claudia Niederée
L3S Research Center
Hannover, Germany
niederee@L3S.de

Peter Fankhauser[*]
Fraunhofer IPSI
Darmstadt, Germany
fankhaus@ipsi.fhg.de

ABSTRACT

We have recently witnessed an enormous growth in the volume of structured and semi-structured data sets available on the Web. An important prerequisite for using and combining such data sets is the detection and merge of information that describes the same real-world entities, a task known as Entity Resolution. To make this quadratic task efficient, blocking techniques are typically employed. However, the high dynamics, loose schema binding, and heterogeneity of (semi-)structured data, impose new challenges to entity resolution. Existing blocking approaches become inapplicable because they rely on the homogeneity of the considered data and a-priory known schemata. In this paper, we introduce a novel approach for entity resolution, scaling it up for large, noisy, and heterogeneous information spaces. It combines an attribute-agnostic mechanism for building blocks with intelligent block processing techniques that boost blocks with high expected utility, propagate knowledge about identified matches, and preempt the resolution process when it gets too expensive. Our extensive evaluation on real-world, large, heterogeneous data sets verifies that the suggested approach is both effective and efficient.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms

Algorithms, Experimentation, Performance

Keywords

Data Cleaning, Entity Resolution, Attribute-Agnostic Blocking

1. INTRODUCTION

Due to the success of Social and Semantic Web applications, as well as the establishment of standards and best practices for publishing and exchanging data on the Web, a large and quickly growing volume of structured and semi-structured data sets has become available on the Web. The resulting data sets, such as Wikipedia[1] and Freebase[2], are composed of data originating from a variety of sources, including information extractors, user-generated content, as well as preexisting and reused content that is combined in new ways. These data sets form large, loosely structured, and valuable information spaces, which exhibit unprecedented levels of heterogeneity, both in terms of the schemata describing the same entity types, and in terms of the separate profiles describing the same entity. Google Base[3], for instance, encompasses $100,000$ distinct schemata corresponding to $10,000$ entity types [16].

At the same time systems that aggregate, reuse, and combine such data collections are gaining popularity (e.g., Yahoo! Pipes[4] and other mashup solutions) due to their ability to leverage the effort invested in the creation of the individual data sets. This raises the need for effective integration solutions that can deal with such heterogeneous, noisy, and constantly expanding data collections through an effective, as well as efficient methodology. An important task in data integration is *Entity Resolution (ER)*, also called De-duplication [3, 8].

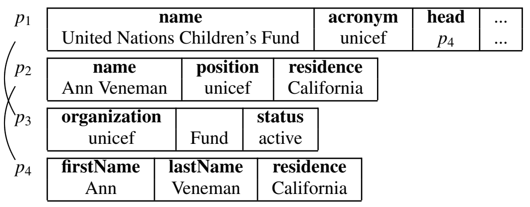
Characteristics & Challenges. ER deals with the identification and merging of data that describe the same real-world entity, e.g., a person, a product, or a location. A variety of effective and efficient ER approaches exist [7, 23]. However, their fundamental assumptions are broken by the (semi-)structured data sets of the Web of Data,due to the inherent characteristics of the latter:

- *Loose schema binding*, since the schemata describing entities may range from locally defined to pure tag-style annotations.
- *Noise, missing, and inconsistent values*, introduced by extraction errors, sources of low quality, and use of alternative descriptions. As a result, entity profiles may involve deficient, or even false information.
- *Large, heterogeneous information spaces*, composed of data stemming from a variety of sources and applications.

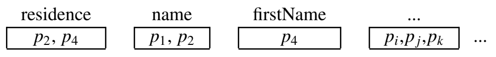
In these settings, existing ER techniques and especially blocking techniques are not applicable: in the core, ER is a quadratic process, as every entity needs to be compared with all other entities. To make ER techniques applicable to large data sets, *blocking* mechanisms [7, 4, 22] have been introduced. They use schema information to split the data into blocks, so that it suffices to conduct comparisons only between entities placed within the same block. More precisely, they identify the most appropriate attributes (usually the most distinctive ones), and then derive effective blocking criteria from them, possibly combining several blocking criteria into a blocking schema [18]. As such, these approaches heavily

[*]This work was done while the author was at L3S Research Center.
[1]http://www.wikipedia.org

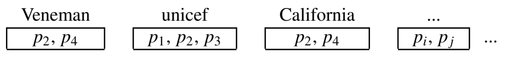
[2]http://www.freebase.com
[3]http://www.google.com/base
[4]http://pipes.yahoo.com



(a) Four entity profiles.



(b) Blocks with traditional mechanism.



(c) Attribute-agnostic blocks.

Figure 1: An illustration of (a) entity profiles, and (b) blocks with a traditional approach and (c) with our approach. All entities that are grouped together in one box are considered to be in the same block.

depend on the existence of an a priori known binding schema, thus being tailored to homogeneous data sets.

Consider, for example, the entities depicted in Figure 1(a). They are represented as a set of attributes comprised of a name (shown in bold) and a value. To process these entities, existing approaches would build blocks based on one or more selected attribute names. For each attribute name they would create blocks containing those entities that share the same corresponding value (possibly after applying a function to the respective values). Figure 1(b) shows the possible resulting blocks using different attribute names as blocking criteria. As we can see, some of the generated blocks, such as that of the attribute 'residence', indeed group together entity profiles that refer to the same real-world objects (p_2 and p_4 in particular). However, this process is not always successful, and some of the blocking criteria fail. This is the case, for instance, with the attributes 'name', 'firstName', and 'lastName', since entities p_1 and p_2, that describe the same object, are split in three separate blocks.

Applying a schema matching method to address this problem is not possible in the settings we consider, due to the overabundance of schemata and attributes, the high change rates, and the subtle differences in the meaning of attributes that the (user-generated) Web data encompasses [16].

Approach & Contributions. For handling such data sets there is a need for methodologies that —apart from being effective in correctly identifying duplicates— are also efficient enough to scale up to their large volume. In this paper, we introduce a novel approach to blocking that accommodates such information spaces through an attribute-agnostic mechanism. In essence, it is based on the idea that profiles of duplicate entities have at least one value in common, independently of the corresponding attribute names. Therefore, all entities that contain the same token in any attribute of their profile are placed in one block (i.e., we use an *attribute-agnostic* blocking condition). The approach is thus very robust against heterogeneity, noise, and loose schema binding, as is illustrated in Figure 1(c).

However, this results in a high level of redundancy, since each entity is placed in multiple blocks that are inevitably overlapping, inevitably increasing the number of required comparisons. In order to make our approach efficient and scalable, we combine the attribute-agnostic block building with intelligent ways of block processing: utility-based block scheduling, duplicate propagation, and early detection of the blocks that are unlikely to contain undetected duplicates.

The approach we present here focuses on identifying duplicates between different data sets, with the individual data collections that compose the input to our method containing no matching entities in themselves. Furthermore, we focus on enhancing the effectiveness and the efficiency of the blocking method in heterogeneous settings. Thus, we do not present a method for the detailed comparison of entity profiles, but introduce a method that serves as an underlying layer, capable of accommodating on top of it different methods for the detailed comparison of entity profiles. For our evaluation, we assume the existence of an oracle: when comparing two profiles, it always judges correctly whether they are duplicates or not. This is in line with the best practice of other blocking approaches proposed in the literature, such as [2, 18].

The main contributions of this paper are the following:

- We introduce the *attribute-agnostic blocking methodology* as an effective and robust technique for grouping into blocks entities of noisy, loosely structured, and highly heterogeneous data sets.
- We introduce the notion of *block utility* and provide an estimation for it through a probabilistic analysis. We also demonstrate that scheduling blocks according to their utility enables a set of strategies that enhance the efficiency and foster the scalability of the ER process.
- We evaluate our blocking solution through extensive experiments on two large data sets, stemming from real-world applications. The outcomes of our experiments demonstrate the effectiveness and scalability of our approach.

Organization. The rest of the paper is structured as follows. In Section 2 we discuss related work, and in Section 3 we present a formal definition of the problem. Section 4 introduces and explains our approach, while Section 5 reports the results of our experimental evaluation. Finally, Section 6 provides conclusions and directions for future work.

2. RELATED WORK

A plethora of algorithms have been proposed in the literature with the aim of effectively identifying data that describe the same real-world entities. Suggested methods span from string similarity metrics [3], to similarity methods using transformations [20, 21], and relationships between data [6, 12, 14]. A complete overview of the existing work in this domain can be found in [5, 7, 15].

As noted in [7], the prevalent method for enhancing the efficiency of the resolution process is data blocking. Relevant methods typically associate each record with a *Blocking Key Value* (BKV) summarizing the values of selected attributes and then operate exclusively on it. The Sorted Neighborhood approach [11], for instance, sorts records according to their BKV and then slides a window of fixed size over them, comparing the records it contains. The StringMap method [13] maps the BKV of each record to a multidimensional Euclidean space, and employs suitable data structures for efficiently identifying pairs of similar records. Alternatively, the q-grams based blocking presented in [9] builds overlapping clusters of records that share at least one q-gram (i.e., sub-string of length q) of their BKV. Canopy clustering [17] employs a cheap string similarity metric for building high-dimensional overlapping blocks, whereas the Suffix Arrays approach, coined in [1] and enhanced in [4], considers the suffixes of the BKV instead. As marked in [4], the performance of existing blocking methods depends on a wealth of application- and data-specific parameters. The error characteristics in the data to be linked, for example, affect the performance of the similarity metric used by canopy clustering, whereas the distribution of values affects the optimal window size of the Sorted Neighborhood approach.

To facilitate parameter setting, several proposed methods automate this procedure with the help of machine learning. More specifically, [18] models this problem as learning disjunctive sets of conjunctions that consist of an attribute (used for blocking) and a method (used for comparing the corresponding values). A typical machine learning algorithm is then applied, yielding a significant improvement over the parameters manually set by experts. Similarly, [2] considers disjunctions of *blocking predicates* (i.e., conjunctions of attributes and methods) along with predicates combined in disjunctive normal form (DNF) and compares them against canopy clustering. The experimental evaluation suggests that DNF predicates have the best performance, with the disjunctive ones outperforming solely canopy clustering. However, these approaches are only applicable to data sets having a schema with a restricted number of distinct attributes.

An alternative approach for improving blocking techniques is introduced in [22]. This work argues that more duplicates can be detected and more pair-wise comparisons can be saved through the iterative distribution of identified matches to subsequently (re-)processed blocks. In this way, both the effectiveness and the efficiency of the ER process can be improved. Similarly to the rest of the existing techniques, this method assumes an a priori known schema that contains a limited number of agreed upon attributes. Therefore, it cannot deal with the unstructured and semi-structured noisy data coming from the Web [7]. Our approach explores this direction through the attribute-agnostic processing of the data.

3. PROBLEM DEFINITION

Our aim is to develop an ER technique that can be efficiently applied in use cases where large volumes of noisy and heterogeneous data are prevalent. In this context, we cannot make assumptions on the used schemata, but we rather have to consider data sets that generally represent domain entities, along with their properties and relationships. To describe the problem more succinctly, we introduce a unifying data model with simple entity profiles that corresponds to real-world entities. These profiles are basically composed of a local ID (i.e., an identifier of this profile in the considered data set) coupled with a set of attributes and the corresponding values. Assuming an infinite set of attribute names $\mathcal{AN}$, along with an infinite set of possible values $\mathcal{V}$, and an infinite set of identifiers $\mathcal{ID}$, an entity profile p is formally defined as follows:

Definition 1. *An **entity profile** p is a tuple $\langle id, A_p \rangle$, where A_p is a set of attributes a_i, and $id \in \mathcal{ID}$ is a local identifier for the profile. Each **attribute** $a_i \in A_p$ is a tuple $\langle n_i, v_i \rangle$, consisting of an **attribute name** $n_i \in \mathcal{AN}$ and an **attribute value** $v_i \in \mathcal{V} \cup \mathcal{ID}$.*

Tag-style attributes values, i.e., attribute values without associated attribute names, are represented by using the empty string as attribute name. Attribute values might also be identifiers of other entities, thus representing relationships between profiles. Moreover, attribute names can be associated with more than one value, by assigning several tuples with the same attribute name to a profile.

Through its simplicity, our model is versatile enough to accommodate entities described in more complex formats, such as relational, RDF, or XML data. It is suitable, therefore, for representing data both in Web [24] and dataspace applications [10].

Definition 2. *A **data set** D is a tuple $\langle A_D, V_D, ID_D, P_D \rangle$, where $A_D \subseteq \mathcal{AN}$ is the set of attribute names appearing in it, $V_D \subseteq \mathcal{V}$ is the set of values used in it, $ID_D \subseteq \mathcal{ID}$ is the set of local identifiers contained in it, and $P_D \subseteq ID_D \times \mathcal{P}(A_D \times V_D)$ is the set of entity profiles that it comprises.*

Where there is no risk of ambiguity, we use D to represent the set of the profiles P_D within the data set as a shorthand.

Consider the case where two data sets have to be merged with each other. Since the profiles contained in them were created independently, some of the entity profiles might describe the same real-world entity, without necessarily containing the same information.

Definition 3. *Two profiles p_i and p_j are said to **match**, denoted with $p_i \equiv p_j$, when they describe the same real-world entity.*

In line with other works on blocking such as [2, 18], we do not consider any specific method for profile comparison to decide about a match. Instead, we assume an oracle that judges always correctly whether two entity profiles match or not. In addition, we consider two data sets, D_1 with $P_{D_1} = \{p_1, \ldots, p_k\}$ and D_2 with $P_{D_2} = \{q_1, \ldots, q_n\}$, that are separately duplicate-free, but probably overlapping with each other:

$$\forall o_i, o_j \in D_1 \cup D_2 : o_i \equiv o_j \Rightarrow$$
$$(o_i \in P_{D1} \wedge o_j \in P_{D_2}) \vee (o_i \in P_{D_2} \wedge o_j \in P_{D_1})$$

To successfully integrate these two data sets, we need to identify their matching profiles (and merge them). Comparing every $p_i \in P_{D_1}$ with every $q_j \in P_{D_2}$ would require $|P_{D_1}| \cdot |P_{D_2}|$ steps. This is, however, inefficient or even infeasible in the case of voluminous data sets. As demonstrated by other approaches [7, 23], it is better to split the data into blocks, such that profiles within a block are more likely to generate matches in comparison to profiles in separate blocks.

A *block b* is the subset of a data set D that a *blocking criterion bc* defines. For our purpose, bc is modeled as a combination of two functions:

1. A transformation function f_t that derives the appropriate representation for blocking from the complete entity profile (or parts of it). An example of such a function, is one that represents each entity with its value for the attribute "zip code".

2. A transitive, symmetric constraint function f^i_{cond} that encapsulates the blocking condition, and has to be satisfied by two entities, if they are to be placed in the same block b_i. For example, the equality on the zip code.

Definition 4. *A **blocking criterion** bc for a data set D is defined by a transformation function $f_t : P_D \mapsto T$ and a set of constraint functions[5] $f^i_{cond} : T \times T \mapsto \{true, false\}$.*

For defining effective blocking criteria, existing blocking approaches rely on schema information such as attribute names and knowledge about the domains of the respective attributes. As mentioned in Section 2, these approaches typically create their blocking criteria by combining a transformation function that derives the BKV from a selected (set of) attribute(s) and a constraint function demanding equality or similarity on it. In our case, though, where there is merely a loose schema binding and attribute names are optional in the entity profiles, we may not rely on schema information in the definition of blocking criteria.

[5]It is worth noting at this point, though, that a single constraint function $f_{cond} = f^1_{cond} = f^2_{cond} = ...$ is often sufficient for expressing the blocking criterion.

Definition 5. *If a blocking criterion bc is applied to a data set D, it creates a set B of blocks. A* ***block*** *$b_i \in B$ is a maximal subset of the profiles P_D of the considered data set D that is defined by the blocking criterion, i.e., by a transformation function and one of the constraint functions f^i_{cond}:*

$$b_i \subseteq P_D \wedge \forall p_1, p_2 \in P_D : f^i_{cond}(f_t(p_1), f_t(p_2)) = true \Rightarrow p_1, p_2 \in b_i$$

For the case we are considering (i.e., resolution of two duplicate-free data sets), the blocking criterion is applied to the union of the data sets D_1 and D_2: $D = D_1 \cup D_2$. Thus, each block b_i can be divided into two parts, $b_{i,1}$ and $b_{i,2}$, which we call *inner blocks*:

Definition 6. *For a block b_i that is defined on the union of two data sets $D = D_1 \cup D_2$, the* ***inner block*** *$b_{i,1}$ ($b_{i,2}$) is the subset of elements in b_i that originate from the data set D_1 (D_2): $b_{i,k} = \left\{e \in b_i : e \in P_{D_k}\right\}$ for k = 1,2.*

The gist of *blocking techniques* is that it suffices to consider pair-wise comparisons solely between profiles contained within the same block. They reduce, therefore, the number of required comparisons from $k \cdot n$ to $\sum comp_i$, where $comp_i$ is the number of comparisons in block b_i, equal to $|b_{i,1}| \cdot |b_{i,2}|$ for the inner blocks $b_{i,k}$ for $k = 1, 2$ of the considered block b_i.

According to [18], a blocking technique should balance the trade-off between the following two competing quality targets:

- Target 1: Matches should share at least one block, otherwise they cannot be detected (i.e., minimize false negatives). This requirement determines the *effectiveness* of the blocking approach.
- Target 2: Each block should contain as few irrelevant (non-matching) entities as possible (i.e., minimize false positives). In other words, the goal is to reduce the number of unnecessary pair-wise comparisons, thus enhancing the *efficiency* of the blocking approach.

As a result, the ideal blocking is achieved by those criteria that group only matches into a block, while each entity is contained in no more than one block. However, in the context of noisy data sets, several approaches, such as [4, 11, 22], allow profiles to be associated with multiple blocks, that are thus overlapping. In this way, redundancy is introduced so as to avoid missing matches. Practically, this can be achieved through the combination of several blocking criteria into a *blocking schema* [18].

Given that we deal with heterogeneous, noisy data sets and that we cannot rely on a strong schema binding, we need a robust blocking technique for ensuring the effectiveness of blocking. This will introduce a considerable amount of redundancy that increases the number of required comparisons at the expense of the efficiency of the approach. In short, the following problems have to be solved:

- *Problem 1:* Find a *blocking criterion bc* defined by functions f_t and f^i_{cond} that is robust enough to produce a set of blocks B of high quality in the context of heterogeneous and noisy data sets.
- *Problem 2:* Find an efficient and scalable mechanism of block processing that effectively reduces the number of required pair-wise comparisons.

4. ATTRIBUTE-AGNOSTIC BLOCKING

For dealing with the problems identified above, we introduce an attribute-agnostic blocking approach. Its core idea is that the profiles of two matching entities have at least one value in common — independently of the associated attribute names (*attribute-agnostic blocking criterion*). Hence, entities are grouped into blocks considering only the values of their profiles. Though effective and robust, this liberal blocking criterion associates each entity with multiple blocks, resulting in *overlapping blocks* and high redundancy. The increased number of comparisons that this redundancy brings about is reduced by a set of block processing techniques that complement our approach. In more detail, our approach consists of the following steps that deal with the problems identified in the previous section:

1. *Block Building* produces a set of blocks B with the necessary robustness for the data sets we are considering (Problem 1),
2. *Block Scheduling* defines a block sorting strategy based on probability theory (contributing to Problem 2), and
3. *Block Processing* processes efficiently the ordered set of blocks through propagation of the identified duplicates and pruning of blocks with low utility (contributing to Problem 2).

These steps are outlined in Algorithm 1 and further explicated in the following sections.

4.1 Block Building

To address *Problem 1*, we introduce a blocking criterion that places two entities in the same block on the *attribute-agnostic* condition that they share a token in their profiles. More formally, our blocking criterion employs a *transformation function* f_t that is based on a tokenization function, `tokenize`. This function is applied to all values of the entity profiles in the data sets D_1 and D_2, transforming them into sets of tokens:

$$f_t(p) = \left\{t_i : \exists n_i, v_i : \langle n_i, v_i\rangle \in A_p \wedge t_i \in tokenize(v_i)\right\}$$

Thus, it defines two sets $TOKENS(D_1)$ and $TOKENS(D_2)$, comprising all the distinct tokens in the values of the profiles in D_1 and D_2, respectively.

Constraint functions f_{cond} are then defined individually on these value tokens t_i. In fact, it suffices to consider only the intersection of the token sets of D_1 and D_2 (i.e., $t_i \in TOKENS(D_1) \cap TOKENS(D_2)$), since both data sets are duplicate-free. More formally, the constraint function is defined as:

$$f^{t_i}_{cond}(p, q) = ((t_i \in f_t(p)) \wedge (t_i \in f_t(q)))$$

The *blocking criterion* that results from the combination of the transformation function f_t with all constraint functions $f^{t_i}_{cond}$ creates a set of blocks that follow Definition 5 of the previous section. Intuitively, each block b_i corresponds to a token $t_i \in TOKENS(D_1) \cap TOKENS(D_2)$ and consists of all the entities containing this token in their profile values.

This blocking criterion has two major advantages: first, it can be efficiently implemented employing the well-established IR technique of posting lists, i.e., an inverted index. Second, it is very robust to noise and heterogeneity, because the likelihood of two duplicates with no common block at all is quite low. Indeed, this would mean that the corresponding profiles do not share a single token, which is very unlikely for two profiles describing the same real-world entity.

Revisiting the blocking targets defined in Section 3, our blocking criterion improves on Target 1, keeping the likelihood of missed matches low. However, it introduces redundancy at the expense of Target 2, since the resulting blocks are overlapping, and partially large in size. To improve on Target 2, the following two steps of our approach aim at cutting down the number of potential pair-wise comparisons, without any impact on effectiveness.

4.2 Block Scheduling

The block scheduling step aims at sorting all elements of the set of blocks, B, into a processing order that allows for a significant reduction in the overall number of comparisons during the next step (Section 4.3). It assigns a *utility value* to each $b_i \in B$ by comparing the cost for processing it (number of comparisons) against the corresponding gain (comparisons spared in the other blocks due to effective propagation). More formally, the utility u_i of a block $b_i \in B$ is defined as follows:

$$u_i = \frac{gain_i}{cost_i}, \text{ where} \tag{1}$$

- $gain_i$ expresses the expected benefit of processing b_i, and
- $cost_i$ denotes the cost of processing it.

Both are measured in terms of number of comparisons. We chose this definition over the alternative form of $u_i = gain_i - cost_i$, as it is more appropriate for expressing the gain per unit of cost. By ordering blocks in this way, we can maximize the gain that is achieved until the processing is stopped during the block pruning phase (presented in Section 4.3.3).

Blocks are then sorted according to a *ranking function* $r : B \mapsto \Re$ that defines a partial order on the blocks B based on the utility u_i of a block $b_i \in B$. This function sorts the elements of B in descending order according to the following implication: $u_i \le u_j \Rightarrow r(b_i) \ge r(b_j)$ (see lines 11 and 13 in Algorithm 1). In summary, we employ the following definitions of *block utility* for ranking the set of blocks B:

$$u_i \approx \frac{1}{max(|b_{i,1}|, |b_{i,2}|)} \tag{2}$$

$$u_i \approx \frac{avg_{j,k}(min(|f_t(p_j)|, |f_t(q_k)|) - 1))}{max(|b_{i,1}|, |b_{i,2}|)} \tag{3}$$

We call Formula 2 *plain probabilistic scheduling method*, and Formula 3 *enhanced probabilistic scheduling method*. The latter provides a more precise estimation of utility, at the expense of a higher computational cost. In our experiments, we investigate whether this cost is justified by a significantly higher performance in comparison with Formula 2.

4.2.1 Derivation of the formulas

In the following we present the probabilistic analysis that leads to the above formulas. More specifically, $cost_i$ in Formula 1 is equal to the number of comparisons required for processing a block b_i with inner blocks $b_{i,1}$ and $b_{i,2}$ from datasets D_1 and D_2, respectively:

$$cost_i = |b_{i,1}| \cdot |b_{i,2}|$$

In Formula 1, $gain_i$ denotes the number of comparisons potentially spared after block b_i has been examined. A comparison is spared if two entities in a block match, and their profiles are contained in subsequently processed blocks. Hence, it actually depends on two factors:

1. the expected number of matches in a block, and
2. the expected number of blocks affected by a match.

Note, that sparing comparisons requires that information on the results of comparisons is propagated during block processing.

Expected number of matches in a block. To estimate the number of duplicates in a block b_i, we need to estimate the probability $P(M|t_i = t_i)$[6] of a match given that the corresponding entities share the token t_i in their profiles. According to the Bayes theorem, this probability can be estimated as:

$$P(M|t_i = t_i) = \frac{P(M) \cdot P(t_i = t_i|M)}{P(t_i = t_i)}, \text{ where}$$

- $P(M)$ represents the prior probability that we have a match in the dataset. This is equal to $P(M) = m/(|D_1| \cdot |D_2|)$, where m is the number of all duplicate pairs in the data set (i.e., the ratio of all possible entity pairs that are actually duplicates).
- $P(t_i = t_i|M)$ is the probability that, given a match M, the matching profiles share the token t_i. It is, thus, equal to $P(t_i = t_i|M) = m_i/m$, where m_i is the number of matches that share the token t_i, and m, as above, is the total number of matches in the given data set. We assume here, that tokens that are frequent (rare) in **both** data sets are also frequent (rare) in matches. Therefore, the number of elements in the smaller inner block[7] of the block b_i that is associated with t_i is a good proportional measure for m_i: $m_i \approx min(|b_{i,1}|, |b_{i,2}|)$.
- $P(t_i = t_i) = (|b_{i,1}| \cdot |b_{i,2}|)/(|D_1| \cdot |D_2|)$ as it expresses the number of pairs that agree on a particular token t_i with frequencies $|b_{i,1}|$ and $|b_{i,2}|$ among all possible pairs.

Plugging all these together gives us:

$$\begin{aligned} P(M|t_i = t_i) &\approx \frac{m}{|D_1| \cdot |D_2|} \cdot \frac{min(|b_{i,1}|, |b_{i,2}|)}{m} \frac{|D_1| \cdot |D_2|}{|b_{i,1}| \cdot |b_{i,2}|} \\ &= \frac{1}{max(|b_{i,1}|, |b_{i,2}|)} \end{aligned}$$

The expected number of duplicates in block b_i is equal to $|b_{i,1}| \cdot |b_{i,2}| \cdot P(M|t_i = t_i)$. Assuming that every match in b_i spares on average $comp_i$ comparisons, the corresponding expected gain is:

$$\begin{aligned} gain_i &\approx comp_i \cdot |b_{i,1}| \cdot |b_{i,2}| \cdot P(M|t_i = t_i) \\ &= comp_i \cdot min(|b_{i,1}|, |b_{i,2}|) \end{aligned}$$

This results in the following formula for the expected utility of processing block b_i:

$$\begin{aligned} u_i &\approx \frac{m \cdot comp_i \cdot min(|b_{i,1}|, |b_{i,2}|)}{min(|D_1|, |D_2|)} \cdot \frac{1}{|b_{i,1}| \cdot |b_{i,2}|} \\ &= \frac{comp_i}{max(|b_{i,1}|, |b_{i,2}|)} \end{aligned} \tag{4}$$

For the number of comparisons spared by a match, $comp_i$, we investigate two options in our approach:

1. We assume that, for the utility-based ranking of the blocks, this number can be approximated by a block-independent number C, that can be omitted. In this case we arrive at Formula 2.
2. We estimate $comp_i$ in a block-dependent way, as described below.

Expected number of blocks affected by a match. If two matching profiles are found in a block, these profiles do not have to be compared again in the blocks they occur in, due to additional common tokens. Hence, to estimate the number of comparisons $comp_i$ spared by processing block b_i, we need to know the size of the bag cb_i of other blocks, where the matching profiles of block b_i co-occur. However, computing the cardinality of cb_i is infeasible without knowing all matching profile pairs in advance.

Therefore, we approximate $comp_i$ with the average number of blocks shared between all pairs of profiles, $p_j \in P_{D_1}$ and $q_k \in P_{D_2}$, in block b_i. Instead of measuring the precise number of common

[6] $t_i = t_i$ is a shorthand notation for the event two profiles share the token t_i.

[7] We use the size of the smaller inner block, since a token might be frequent in one of the data sets and rare in the other.

Algorithm 1: Outline of Attribute-agnostic Blocking

```
    Input: (a) data set $D_1$, (b) data set $D_2$, (c) $\rho$
    Output: A set of matches $\{(p_i \equiv p_j)\}$
1   $I_1 \leftarrow$ TOKENS($D_1$);
2   $I_2 \leftarrow$ TOKENS($D_2$);
3   m $\leftarrow$ min($I_1$.size(),$I_2$.size());
4   common_blocks $\leftarrow \{\}$;
5   foreach $token \in I_1$ do
6       if $I_2.contains(token)$ then
7           $c_1 \leftarrow I_1$.entitiesWithToken($token$);
8           $c_2 \leftarrow I_2$.entitiesWithToken($token$);
9           if $min\ (c_1,\ c_2) < \rho \times m$ then
10              $b_i \leftarrow$ new Block($token$);
11              $b_i$.setUtility(1/max($c_1$, $c_2$));
12              common_blocks $\leftarrow$ common_blocks $\cup \{b_i\}$;
13  common_blocks $\leftarrow$ sort(common_blocks);
14  $dh_{max} \leftarrow$ getMaxDuplicateOverhead(common_blocks);
15  matches $\leftarrow \{\}$;
16  $comp \leftarrow 0$;
17  foreach $b_i \in common\_blocks$ do
18      $matches_i \leftarrow b_i$.deduplicateBlock(matches);
19      $comp$ += $b_i$.noOfComparisons();
20      if $matches_i \neq \{\}$ then
21          matches $\leftarrow$ matches $\cup\ matches_i$ ;
22          if $dh_{max} < \frac{comp_k}{matches_i.size()}$ then
23              break;
24          else
25              $comp \leftarrow 0$;
26  return $matches$;
```

blocks for each such pairs, we estimate it as the minimum size of their tokenized profiles, minus the current block, b_i[8]:

$$\begin{aligned} comp_i &\approx avg_{j,k}(min(|f_t(p_j)|, |f_t(q_k)|) - 1) \\ &\forall \quad p_j \in P_{D_1}, q_k \in P_{D_2} \end{aligned}$$

Replacing $comp_i$ in Formula 4 with the above estimation and ignoring the constant measures (m and $min(|D_1|, |D_2|)$), we arrive at Formula 3.

4.3 Block Processing

During this step, individual blocks are thoroughly examined by comparing pair-wise the entities within a block that stem from different data sets. To minimize the number of required comparisons, this step encompasses three techniques that ensure the efficiency of the whole resolution procedure, based on the ordering of blocks produced by block scheduling:

1. *Block Purging*, which cuts off the oversized blocks that do not contribute significantly to the effectiveness of the method.

2. *Duplicate Propagation*, which saves comparisons by distributing the already identified matches to the subsequently processed blocks.

3. *Block Pruning*, which terminates the resolution process as soon as the estimated cost of detecting the remaining duplicates exceeds an upper bound.

In the following paragraphs, we elaborate on each one of these techniques.

[8]Note, however, that it is expensive to calculate this average value over all pairs of profiles.

4.3.1 Block Purging

In the resolution process, it makes sense to compare only entities that share a token that is not too common (i.e., not a stop word found in an excessive number of profiles). To this end, this step aims at cleaning the block processing list from *oversized blocks*. These are blocks that correspond to tokens of little discriminativeness, thus entailing a large number of comparisons while being unlikely to contribute *non-redundant matches* (i.e., duplicates whose profiles have no other token in common). Hence, they can be safely excluded from the block processing procedure, enhancing considerably the efficiency without any significant impact on the effectiveness. For instance, one of the data sets we employed in our experimental study entails in total 40, 825 distinct blocks and contain 22, 387 duplicates. The 395 largest blocks involve 50% of the total comparisons, while containing just 121 non-redundant matches.

We heuristically identify oversized blocks by estimating an upper limit on the number of matches a block is expected to contain. More specifically, we derive the purging threshold as follows: the maximum possible number of matches among two data sets, D_1 and D_2, is $m_{max} = min(|D_1|, |D_2|)$. We define the maximum number of matches m_{max_i} that we expect to find inside a block b_i as a percentage of m_{max}, namely $\rho \cdot m_{max}$. Given that each block b_i contains two inner blocks $b_{i,j}$ for j= 1, 2, it has to satisfy the following condition for being considered in the processing list:

$$m_{max_i} = min(|b_{i,1}|, |b_{i,2}|) < \rho \cdot m_{max}, \text{where } \rho \in [0, 1] \qquad (5)$$

The purging conditions, therefore, sets an upper bound on the size of the smallest inner block. The actual value of ρ is determined experimentally, by targeting an increase in efficiency with limited impact on effectiveness. As depicted in the lines 9 to 12 of Algorithm 1, this step is integrated with block scheduling for efficiency reasons. Before estimating the utility of each block, we check whether it satisfies the above condition, so as to avoid unnecessary utility computations and to reduce the number of blocks considered in the subsequent steps.

4.3.2 Duplicate Propagation

As mentioned above, the set of blocks, B, contains overlapping blocks, and the same entities - without further action - are compared more than once. This situation is alleviated by propagating the identified matches to the subsequently processed blocks, as it is done in the approach described in [22]. To incorporate this idea into our approach, we employ a central data structure (a hash table) containing at any time all the matches that have been detected so far (line 15 of Algorithm 1). Before comparing a pair of entities, we check whether either of them is already involved in one of the identified pairs of duplicates. If this is true for at least one of them, we skip the comparison (line 18 of Algorithm 1).

As in [22], we do not maintain a similar data structure with all profile pairs that have been examined but are not matching. There are two arguments against managing and propagating this information:

1. The number of possible pairs is so large that managing all compared pairs becomes quickly inefficient, and

2. The probability of actually using this information is much lower than using it for matching pairs; in contrast to matching pairs, non-matching ones are unlikely to share additional tokens and to co-occur in more than one block.

4.3.3 Block Pruning

As explained in Section 4.2, block scheduling promotes blocks that combine high expected gain (i.e., a high number of duplicates) with a low cost (i.e., a low number of comparisons) in high ranking positions (earlier processing). Moving towards lower ranking positions, we find blocks with fewer duplicates and a higher comparison cost. Duplicate propagation further reduces the number of duplicates to be detected in such blocks, since identified matches are not re-considered in subsequently processed blocks. We can conclude that the lower the position of a block, the lower its probability to contain new, yet unidentified duplicates. This suggests a break-even point where the possibility of finding additional matches is no longer worth the cost.

Block pruning aims at identifying the blocks lying after this point in order to remove them from the processing list. In this way, the efficiency of our approach is further enhanced, at the expense of a negligible number of missed matches. For detecting this break-even point, block pruning keeps track of the *duplicate overhead*, denoted as *dh*, which expresses the average number of comparisons performed for detecting the latest matches. The ER process terminates when *dh* reaches a *maximum duplicate overhead*, denoted as dh_{max}: this indicates that the expected remaining number of duplicates is considered small compared to the high processing cost of the required comparisons.

As shown in lines 19 to 25 of Algorithm 1, the value of the duplicate overhead dh_k after processing the *k*-th block containing duplicates (i.e., blocks with $nm_k \geq 1$) is computed as follows:

$$dh_k = \frac{comp_k}{nm_k}, \text{ where}$$

- $comp_k$ denotes the number of comparisons performed after processing the $k-1$-th block, and
- nm_k stands for the number of **new** matches identified within the current block that contains duplicates.

Dh_{max} is computed relative to the overall size of the considered data set. More precisely, it is computed as follows:

$$dh_{max} = 10^{\frac{log(n)}{2}}, \qquad (6)$$

where *n* is the number of all possible comparisons after the block purging step. The intuition behind this formula is that the number of comparisons required for detecting a match is considered too large, when it reaches half the order of magnitude of all possible comparisons in the considered blocks.

5. EXPERIMENTAL EVALUATION

In this section we present a series of experiments that investigate the performance and the scalability of our method on real-world, large, heterogeneous, and noisy data sets. Our approach was fully implemented in Java 1.6, employing the Lucene search engine library[9] for the underlying inverted index functionality. All experiments were performed on a server with Intel Xeon 3.0GHz, running Linux with kernel version 2.6.18.

Data Sets. We employed two data sets with the technical characteristics summarized in Table 1. D_{movies} is a collection of movies shared among IMDB and DBPedia. For deriving the ground-truth, we relied on the *"imdbid"* attribute in the profiles of the DBPedia movies.

The $D_{infoboxes}$ data set consists of two different versions of the DBPedia Infobox Data Set[10]. They have been collected by extracting all name-value pairs from the infoboxes of the articles in Wikipedia's english version. $DBPedia_1$ is a snapshot of Wikipedia Infoboxes in October 2007, whereas $DBPedia_2$ dates from October 2009, so $DBPedia_1$ is older than $DBPedia_2$. Although it may seem simple to resolve two versions of the same data set, this is not the case. During the two years that intervene between these two versions, Wikipedia Infoboxes were so heavily modified that there is only a small overlap between their profiles, even for duplicate entities. As shown in Table 2, just 25% of all name-value pairs and 50% of the attribute names are shared among the entities common in both versions. Regarding the ground-truth, we considered as matches those entities that had exactly the same URL. As a result, a small part of the actual matches has been ignored; these are actually the entities that had their URL changed (e.g., due to disambiguation reasons).

	D_{movies}		$D_{infoboxes}$	
	DBPedia	*IMDB*	$DBPedia_1$	$DBPedia_2$
Entities	27,615	23,182	1,190,734	2,164,058
Name-Value Pairs	186,013	816,012	17,453,516	36,653,387
Avg. Profile Size	6.74	35.20	14.66	16.94
Attribute Names	7	5	30,757	52,554
Common Attr.	1		27,253	
Duplicates	22,405		892,586	
Comparisons	$6.40 \cdot 10^8$		$2.58 \cdot 10^{12}$	

Table 1: Overview of the data sets used in the experiments.

	Attribute Names	Name-Value Pairs
$DBPedia_1$	30,757	17,453,516
$DBPedia_2$	52,554	36,653,387
Common	27,253	10,361,467
Distinct	56,058	43,745,435

Table 2: Overlap in the profiles of duplicates in $D_{infoboxes}$.

Evaluation Metrics. To measure the performance of our method, we follow [2, 4, 18] and consider the following, established metrics for blocking techniques:

- *Pair Completeness* (*PC*) expresses the ratio between the matches identified by our method and the matches existing in the data set. It is computed by the following formula: $PC = dm/gm$, where *dm* denotes the number of detected matches, and *gm* represents the number of matches in the ground-truth. *PC* is comparable to the Recall metric of Information Retrieval and takes values in the interval $[0, 1]$. Higher values indicate higher *effectiveness* of the blocking method.
- *Reduction Ratio* (*RR*) expresses the reduction in the number of pair-wise comparisons required by our method with respect to the baseline one. In our case, we evaluate the RR of each step with respect to the comparisons of the previous one. It is defined as follows: $RR = 1 - mc/bc$, where *mc* stands for the number of comparisons entailed by the considered method, and *bc* expresses the number of comparisons entailed by the baseline method. *RR* takes values in the interval $[0, 1]$ (for $mc \leq bc$), with higher values denoting higher *efficiency*.

Similar to [2, 4, 18], we do not report any numbers on Precision, since our method focuses at improving the efficiency of the resolution process, while maintaining its effectiveness at high levels. The high values of *PC* that our method exhibits (Tables 3 and 4) actually imply that the Precision of the ER process depends solely on the entity matching method that performs the detailed profile comparisons. Our framework is intended to incorporate any such method.

[9]http://lucene.apache.org

[10]http://wiki.dbpedia.org/Datasets

	Compar.	Duplicates	PC	RR
Block Building	$3.05{\cdot}10^8$	22,387	99.92%	-
Block Purging	$2.67{\cdot}10^7$	22,268	99.39%	91.23%
Sched.-Propag.	$9.75{\cdot}10^6$	22,268	99.39%	63.50%
Block Pruning	$1.03{\cdot}10^6$	22,268	99.39%	89.39%
Brute Force	$6.40{\cdot}10^8$	22,405	100.00%	-

Table 3: Performance overview of each step on D_{movies}.

	Compar.	Duplicates	PC	RR
Block Building	$5.72{\cdot}10^{20}$	892,560	99.99%	-
Block Purging	$3.98{\cdot}10^{10}$	891,599	99.89%	99.99%
Sched.-Propag.	$7.42{\cdot}10^9$	891,599	99.89%	81.36%
Block Pruning	$1.12{\cdot}10^8$	838,760	93.97%	98.49%
Brute Force	$2.58{\cdot}10^{12}$	892,586	100.00%	-
StateOTArt$_{NAME}$	$1.33{\cdot}10^6$	380,673	42.65%	99.99%
StateOTArt$_{WEBS}$	$1.62{\cdot}10^6$	400,538	44.87%	99.99%

Table 4: Performance overview of each step on $D_{infoboxes}$, and with two state-of-the-art blocking approaches.

5.1 Overall Performance Analysis and Comparison to Brute Force Approach

To investigate the performance of our method, we first used D_{movies} as a test bed for identifying the optimal parameter values for each step of our approach. We then applied the resulting configuration on $D_{infoboxes}$ to verify the generality of the method. This process resulted in the following configuration:

1. Block scheduling with the plain probabilistic method (Sec. 4.2),
2. Block purging with $\rho = 0.005$ (Formula 5 in Sec. 4.3.1), and
3. Block pruning with $dh_{max} = 10^{\frac{log(n)}{2}}$ (Formula 6 in Sec. 4.3.3).

The performance of each step on D_{movies} and $D_{infoboxes}$ is summarized in Tables 3 and 4, respectively. The table also reports the *Brute Force approach (BFA)*, which compares each entity of one subset with all entities of the other. The *RR* value is computed with respect to the number of comparisons of the previous step, i.e., previous row in the tables.

Regarding D_{movies}, our method reduced the required number of comparisons from $6.40{\cdot}10^8$ to $1.03{\cdot}10^6$ (overall RR = 99.84%), while maintaining a *PC* of 99.39%. Excluding the time required for comparing entity profiles, it took 15 seconds to create and store the indices on the hard disk, and another 62 seconds to apply all its steps. The size of the indices were $6MB$ and $22MB$ for the DBPedia and the IMDB movies respectively, whereas their actual sizes are $13.08MB$ and $31.72MB$.

As far as $D_{infoboxes}$ is concerned, the required number of comparisons was restricted to $1.12{\cdot}10^8$ from $2.58{\cdot}\ 10^{12}$ (overall RR = 99.99%), with a *PC* of 93.97%. It required 12 minutes for building and storing the indices of both subsets, while the processing of the resulting set of blocks required 145 minutes. The size of the $DBPedia_1$ index was $715MB$, while that of $DBPedia_2$ amounted to $1.4GB$ (their actual sizes are $1.16GB$ and $2.43GB$ respectively).

Comparing Tables 3 and and 4 we can see that the performance of each step is quite similar on both data sets. It is worth commenting, though, the difference between the two data sets in the performance of block building. For D_{movies} it entails less than half of the comparisons of BFA, whereas for $D_{infoboxes}$ the number of comparisons resulting after blocking exceeds the number of all possible pair combinations. This discrepancy is caused by the redundancy our method introduces in order to deal with noise and heterogeneity (Section 4.1); each entity is placed in multiple, overlapping blocks and the larger its profile, the higher the number of blocks associated with it. The effect of redundancy is restricted in the case of D_{movies}, since it encompasses fewer entities of smaller profiles (Table 1), leading to a low number of predominantly small blocks (Table 5). In contrast, $D_{infoboxes}$ entails a million of much larger blocks as a result of both the higher number of entities and their larger profiles. This inevitably results in a considerable increase in the number of possible comparisons, which is however substantially reduced by the following steps of our method.

5.2 Comparison to State-of-the-Art

We could not compare directly our methodology against existing schema-based blocking techniques, since they require an a priori known, homogeneous schema. However, this is not the case with heterogeneous data sets. The schema space of $D_{infoboxes}$, for instance, contains more than $57,000$ distinct attributes, out of which just $24,000$ (a mere 42%) appears in both versions of DBPedia (Table 1). Nevertheless, to get a feeling of the potential performance of state-of-the-art approaches in this area, we introduced the $StateOTArt_{NAME}$ and $StateOTArt_{WEBS}$ methods that operate on the most *discriminative* attributes, similar to schema-based blocking techniques [4, 19].

In more detail, we first evaluated the $StateOTArt_{NAME}$, which considers the attribute "name" that is the most discriminative one in both data sets (covering 40.56% of the $DBPedia_1$ and 44.87% of the $DBPedia_2$ entities). *RR* was computed with respect to the *Brute Force* approach. As shown in Table 4, blocks with entities having exactly the same value on this attribute result in very high *RR*, but cover less than half of the existing matches (i.e., very low *PC*).

To improve it, we tested the $StateOTArt_{WEBS}$ that, in addition to "name", considers the attribute "website", the second most discriminative attribute in both data sets. Together, they cover 42.36% and 47.07% of the entities in $DBPedia_1$ and $DBPedia_2$, respectively. The number of comparisons involved in the resulting blocks was increased by 20%, maintaining though a very high *RR*, but *PC* improved only by 2%.

In general, the less discriminative an attribute is, the larger the blocks that are created, and the more are the comparisons it involves, without necessarily increasing *PC*. Thus, the more attributes we consider in the context of a schema-based blocking scheme, the higher the impact on *RR*, and the lower the increase in *PC*. For this reason, such techniques are not applicable in the settings we are considering.

5.3 Performance Analysis for each step

In this section we analyze the procedure we followed for tuning the performance of each step in order to identify its optimal configuration.

Block Building. No configuration is required for our attribute-agnostic blocking technique. However, it is interesting to examine its behavior through some relevant statistics that are presented in Table 5. As expected, it achieves a quite high *PC* for both data sets: 99.92% for D_{movies} and 99.99% for $D_{infoboxes}$. We manually checked all 34 missed pairs of duplicates and found out that at least

	D_{movies}	$D_{infoboxes}$
Blocks	40,825	1,210,838
Average Block Size	42.67	81.49
Duplicates	22,387	892,560
Comparisons	$3.04{\cdot}10^8$	$5.72{\cdot}10^{20}$
PC	99.92%	99.99%
RR	49.67%	-

Table 5: Performance of the block building step.

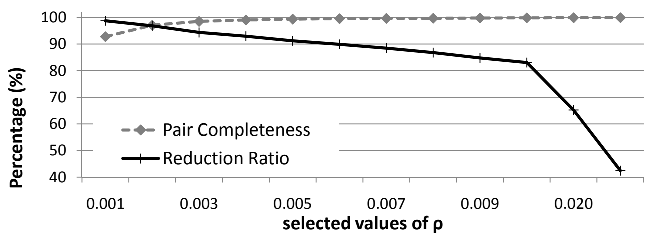


Figure 2: Performance of block purging for several values of ρ.

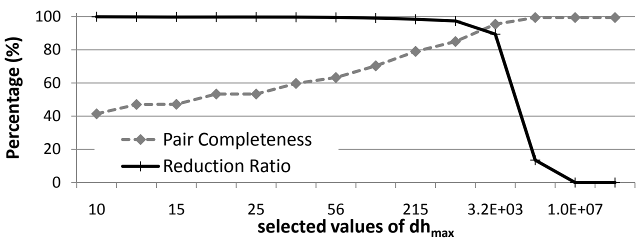


Figure 3: Performance of block pruning for several values of dh_{max}.

one of their profiles contained a rather rudimentary and noisy description of the corresponding entity. In other words, these problematic profiles consisted of insufficient or incorrect information that shared no tokens, and thus no blocks, with their matching profile. Regarding the *RR*, it has a fairly good value for D_{movies} but it cannot be computed for $D_{infoboxes}$ due to the effect of redundancy, as explained above.

Block Purging. To examine the effect of block purging we considered numerous values of the parameter ρ (Formula 5 in Section 4.3.1). Among them, we selected and present in Figure 2 the most meaningful ones: all values in the interval $[0.001, 0.010]$ with a step of 0.001. We also consider the values 0.020 and 0.050 that are indicative of the performance for higher values. The outcomes suggest that the lower the value of ρ, the higher the *RR* and the lower the *PC*. For choosing the value of ρ, we opt for a conservative purging: $\rho = 0.005$ that has $RR = 91.23\%$ and $PC = 99.39\%$. In this way, we enhance efficiency to the extent that effectiveness is affected the least. Indeed, lower thresholds convey higher *RR* combined with more missed matches, whereas higher thresholds involve a negligibly higher PC coupled with a significant decrease in *RR*.

Block Scheduling & Duplicate Propagation. To investigate the effect of block scheduling on the efficiency of our method, we employed four different scheduling approaches in conjunction with duplicate propagation:

1. Random scheduling,
2. Plain probabilistic scheduling, i.e., Formula 2 in Section 4.2,
3. Enhanced probabilistic scheduling, i.e., Formula 3 in Section 4.2, and
4. Supervised scheduling, i.e., a method that orders blocks knowing exactly how many matches they contain and how many comparisons can be spared after they are identified.

The last two methods provide the lower and the upper bound on *RR* respectively (*PC* apparently takes the same value for all sorting methods). The outcomes are presented in Table 6. We can see that the enhanced probabilistic method outperforms the plain probabilistic one only to a negligible extent. Most importantly, though, both have a performance very close to the supervised scheduling. Thus, the plain probabilistic method constitutes the best choice for block scheduling, since it provides a good approximation to the optimal block ordering and does not entail the high computational of the enhanced probabilistic method.

The contribution of duplicate propagation to the efficiency of our method is reflected in the performance of random scheduling: its *RR* is just 9, 78% lower than that of the other scheduling techniques. This indicates that duplicate propagation enhances efficiency significantly, even when blocks are not ordered in a meaningful way (as was proven in [22]).

	Random	Plain Probab.	Enhanced Probab.	Supervised
Duplicates	22,268	22,268	22,268	22,268
Comparisons	12,366,025	9,755,009	9,754,829	9,753,081
PC	99.39%	99.39%	99.39%	99.39%
RR	53.72%	63.49%	63.50%	63.50%

Table 6: Performance of several block scheduling methods.

Block Pruning. To identify the optimal threshold for duplicate overhead, we examined the performance of our method for several values of dh_{max}. More specifically, we considered the following formula: $dh_{max} = 10^{\frac{log(n)}{i}}$, where n is the number of required pairwise comparisons resulting from the block purging step, and i is a real number taking values in the interval $[1, n]$.

In our experiments, we had $n = 7$ and $i \in [1, 7]$ with a step of 0, 5, which results in dh_{max} values that span several orders of magnitude. As depicted in Figure 3, this quite divergent set of values clearly illustrates the trade-off between *RR* and *PC*: for small dh_{max} ($i \geq 3.0$), we have *RR* values well over 99.00%, whereas *PC* remains low (between 41% and 78%). For higher dh_{max} (lower i), *PC* gets higher values, while *RR* becomes substantially lower. The best balance for this trade-off is apparently achieved for $i = 2.0 \Rightarrow dh_{max} = 3.2{\cdot}10^3$, since it corresponds to $RR = 89.39\%$ and $PC = 95.34\%$. In this way, we arrive at Formula 6 in Section 4.3.3.

5.4 Scalability Analysis

To evaluate the scalability of our method, we employed random samples of $DBPedia_1$ and $DBPedia_2$. We considered all sample sizes that are multiples of 10%, and for each of them, we generated 10 different random samples of both data sets to ensure the generality of the outcomes. Note that the size of a sample is defined with respect to the number of entities and not the number of matches. That is to say each sample of $DBPedia_1$ of size x, contains $x\%$ of its entities, and, when resolving it with a sample of $DBPedia_2$ of the same size ($x\%$ of the entities in $DBPedia_2$), the number of matches is not fixed. Therefore, another pair of samples of $DBPedia_1$ and $DBPedia_2$ of equal size will probably contain a different number of matches.

We then applied the aforementioned optimal configuration of our method on the resulting data sets, and estimated the following metrics for each sample size:

- average *PC*, and
- average *comparisons per entity* (*CPE*), which is defined as follows:

$$CPE = \frac{comp_{av}}{|DBPedia_{1,i}| + |DBPedia_{2,i}|}, \text{ where}$$

  – $comp_{av}$ is the average number of required comparisons over all 10 random samples of size i, and
  – $|DBPedia_{1,i}|$ is the number of profiles contained in the

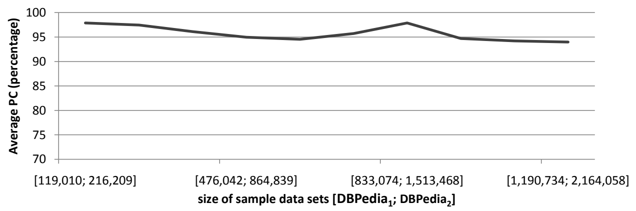


Figure 4: **Average PC for subsets of $D_{infoboxes}$ with increasing sizes.**

subset of $DBPedia_1$ of size $i\%$ ($|DBPedia_{2,i}|$ is similarly defined).

In Figure 4, we see that PC remains at high levels ($\geq$ 94%) throughout all sample sizes, thus verifying the effectiveness of our method under different settings. Most importantly, though, we see in Figure 5 that CPE remains almost constant and low (below 100 on average) for all size samples. This verifies the scalability of our method, since the number of required pair-wise comparisons increases linearly with the total number of entities involved in the resolution process.

5.5 Summary

The results of our thorough experimental study verify the viability of the attribute-agnostic blocking approach with respect to both effectiveness and efficiency. For the former, we can see that only a negligible amount of pairs of duplicates (<< 1%) do not have any block in common after block building. The latter is demonstrated by the overall reduction in the number of comparisons with respect to those required by the brute force approach (i.e., comparing each entity with all the others): it amounts to 2.8 and 4.4 orders of magnitude for D_{movies} and $D_{infoboxes}$ respectively. Most importantly, though, this efficiency is achieved with only a minor loss in effectiveness, as PC remains high ($\geq$ 94%) in all experiments. Scalability is also proven experimentally, since the number of comparisons required per entity is constant on average (< 100), even if the data sets under consideration grow by orders of magnitude. The experimental methodology we followed, employing a training and a testing data set, also ensures the general applicability of our approach.

6. CONCLUSIONS

In this paper we introduce the *attribute-agnostic blocking* methodology as an effective approach to entity resolution in the context of voluminous, highly heterogeneous, and loosely structured information spaces. In contrast to existing blocking approaches, our method makes no use of schema information in the blocking step of ER, thus being applicable in heterogeneous settings with loose schema binding. The robustness of our method introduces redundancy through overlapping blocks, increasing the required number of comparisons. This is, however, reduced by ordering blocks according to their utility (based on an analysis of matching probabilities), effectively propagating identified matches, and avoiding the processing of unpromising blocks. Our thorough experimental evaluation verified the effectiveness, as well as the efficiency of our method on real-world data sets.

In the future, we plan to extend our approach in several ways. First, we intend to modify the probabilistic foundation of block scheduling so that it applies to data sets with duplicates in them. Moreover, we will examine possible partial uses of attribute names for achieving higher efficiency. For example, this could involve special treatment of attributes with high discriminativeness. Finally, we will try to integrate parallelization techniques into our approach, in order to further enhance efficiency.

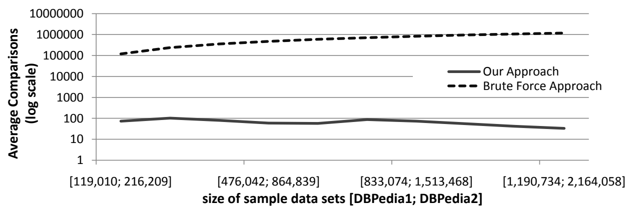


Figure 5: **Average CPE for subsets of $D_{infoboxes}$ with increasing sizes.**

Acknowledgments. This work is partly funded by the European Commission under GLOCAL (248984) and ARCOMEM (270239).

References

[1] A. N. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, 2005.
[2] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.
[3] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.
[4] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 305–314, 2009.
[5] A. Doan and A. Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 2005.
[6] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.
[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 2007.
[8] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations*, 2005.
[9] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.
[10] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.
[11] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.
[12] E. Ioannou, C. Niederée, and W. Nejdl. Probabilistic entity linkage for heterogeneous information spaces. In *CAiSE*, pages 556–570, 2008.
[13] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, 2003.
[14] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *TODS*, 2006.
[15] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, 2006.
[16] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.
[17] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.
[18] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, 2006.
[19] H. B. Newcombe and J. M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Commun. ACM*, 5(11):563–566, 1962.
[20] B.-W. On, N. Koudas, D. Lee, and D. Srivastava. Group linkage. In *ICDE*, 2007.
[21] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, 2002.
[22] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232, 2009.
[23] W. Winkler. Overview of record linkage and current research directions. Technical report, Statistical Research Division, U.S. Bureau of the Census, 2006.
[24] M. Zhong, M. Liu, and Q. Chen. Modeling heterogeneous data in dataspace. In *IRI*, pages 404–409, 2008.