A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces

George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl

Abstract—In the context of entity resolution (ER) in highly heterogeneous, noisy, user-generated entity collections, practically all block building methods employ redundancy to achieve high effectiveness. This practice, however, results in a high number of pairwise comparisons, with a negative impact on efficiency. Existing block processing strategies aim at discarding unnecessary comparisons at no cost in effectiveness. In this paper, we systemize blocking methods for clean-clean ER (an inherently quadratic task) over highly heterogeneous information spaces (HHIS) through a novel framework that consists of two orthogonal layers: the effectiveness layer encompasses methods for building overlapping blocks with small likelihood of missed matches; the efficiency layer comprises a rich variety of techniques that significantly restrict the required number of pairwise comparisons, having a controllable impact on the number of detected duplicates. We map to our framework all relevant existing methods for creating and processing blocks in the context of HHIS, and additionally propose two novel techniques: attribute clustering blocking and comparison scheduling. We evaluate the performance of each layer and method on two large-scale, real-world data sets and validate the excellent balance between efficiency and effectiveness that they achieve.

Index Terms—Information integration, entity resolution, blocking methods

1 Introduction

The amount of global, digital information has exhibited an annual increase of 30 percent in the last few years [16], due to the distributed production of information in businesses and organizations, the increased ability and interest for automatic extraction of information from raw data, and the contributions of valuable information from individual users worldwide through Web 2.0 tools. The combined effect of these factors gives rise to *highly heterogeneous information spaces* (HHIS), manifested in Dataspaces [17] and the Web of data [4].

The main characteristics of HHIS are the following. 1) *nonstructured data*: HHIS principally comprise semistructured data, loosely bound to a rich diversity of schemata, even when describing the same entity types. 2) *high levels of noise*: they suffer from incomplete information, as their user-generated part involves missing or inconsistent data, with extraction errors. 3) *large scale*: users contributing to HHIS are rather prolific, conveying an exponential growth in the content of Web 2.0 platforms, for example, Wikipedia [1].

To leverage the investment in creating and collecting the massive volume of HHIS, the *linked data vision* was recently proposed [4], advocating the combination of related resources in a unified way. A core part of this large-scale integration process is *entity resolution* (ER), which is the process of automatically identifying sets of profiles that pertain to the same real-world entities.

In the context of HHIS, ER comes in two different forms: as *dirty ER*, where the input comprises a single entity collection, and as *clean-clean ER*, which is the process of detecting pairs of matching entities among two large, heterogeneous, individually clean (i.e., duplicate-free), but overlapping collections of entities [5], [11], [21]. As an example for the former, consider the task of identifying duplicate Web pages in the index of a search engine; in the latter case falls the task of merging individual collections of consumer products, which stem from different online stores and, thus, have slightly varying descriptions and proprietary identifiers. Among these two versions of ER, clean-clean ER constitutes a more specific problem that is principally solved through specialized techniques relying on the cleanness of the input data collections. On the other hand, dirty ER is a more generic task that shares many challenges with clean-clean ER. For this reason, we exclusively focus on clean-clean ER in the following and highlight, where necessary, the techniques that are generic enough to handle dirty ER, as well.

Clean-clean ER constitutes an inherently quadratic task (every entity of a collection has to be compared to all entities of another collection). To scale to large volumes of data, *approximate techniques* are employed. These significantly enhance efficiency (i.e., reduce the required number of pairwise comparisons), by trading—to a limited extent—effectiveness (i.e., the percentage of detected duplicates). The most prominent among these techniques is *data blocking*, which clusters similar entities into blocks and performs comparisons only among entities in the same block. There is a plethora of techniques in this field, but the vast majority of them assumes that the schema of

- *G. Papadakis, C. Niederée, and W. Nejdl are with the L3S Research Center, Leibniz University of Hanover, Hannover 30167, Germany. E-mail: {papadakis, niederee}@L3S.de, nejdl@kbs.uni-hannover.de.*
- *E. Ioannou is with the Department of Electronic and Computer Engineering, Technical University of Crete, Technical University Campus, Chania 73100, Crete, Greece. E-mail: ioannou@softnet.tuc.gr.*
- *T. Palpanas is with the Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 14, Povo, TN 38123, Italy. E-mail: themis@disi.unitn.eu.*

Manuscript received 27 Oct. 2011; revised 9 May 2012; accepted 23 July 2012; published online 30 July 2012.
Recommended for acceptance by R. Miller.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-10-0660.
Digital Object Identifier no. 10.1109/TKDE.2012.150.

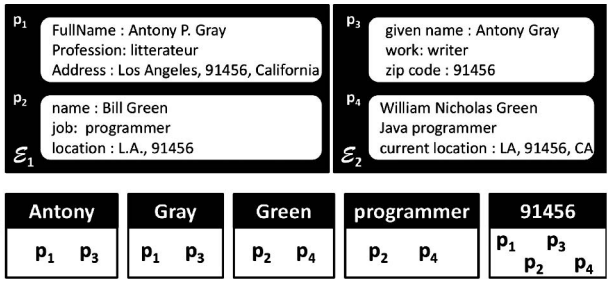


Fig. 1. (a) Two entity collections, and (b) the generated blocks.

the input data, as well as its qualitative features, are known in advance [5]. This requirement is essential to select the most reliable and distinctive attributes for assigning entities to blocks according to their values [5].

However, we note that traditional blocking techniques are incompatible with the inherent characteristics of HHIS mentioned above, rendering most of these methods inapplicable to our problem. To illustrate the peculiarities of HHIS, consider the entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$ that are presented in Fig. 1a. Judging from the similar values they share, we deduce that the entities p_1 and p_2 of $\mathcal{E}_1$ are matching with p_3 and p_4 of $\mathcal{E}_2$, respectively. However, every canonical attribute name has a different form in each profile; the name of a person, for instance, appears as "FullName" in p_1, as "name" in p_2, and as "given name" in p_3. This situation is further aggravated by tag-style values, such as the name of p_4, which is not associated with any attribute name at all. Traditional blocking methods cannot form any block in the context of so high levels of heterogeneity and are only applicable on top of a schema matching method. Although this task seems straightforward in our example, it is impractical in real-world HHIS; Google Base[1] alone encompasses 100,000 distinct schemata that correspond to 10,000 entity types [23].

In these settings, *block building methods* typically rely on *redundancy* to achieve high effectiveness: each entity is placed in multiple blocks, which significantly restricts the likelihood of missed matches. As an example, consider the *token blocking* approach [29], as shown in Fig. 1b; each created block corresponds to a single token and contains all entities with that token, regardless of the associated attribute name. Redundancy, however, comes at the cost of lower efficiency, since it produces overlapping blocks with a high number of unnecessary comparisons. In our example, we observe that the blocks "Gray," "programmer," and "91456" contain four repeated comparisons between the pairs $p_1 - p_3$ and $p_2 - p_4$. Block "91,456" also involves two unnecessary comparisons between the nonmatching pairs $p_1 - p_4$ and $p_2 - p_3$. Such comparisons can be discarded without missing any duplicates, thus enhancing efficiency at no cost in effectiveness. This is actually the purpose of numerous *block processing techniques*.

Several blocking methods have been proposed for clean-clean ER over HHIS. Some of them are competitive (i.e., serve identical needs), while others are complementary, as their combination leads to better performance. However, there is no systematic study on how these methods relate to each other.

In this paper, we propose a novel framework that organizes existing blocking methods, and covers the gap mentioned above. The framework comprises two orthogonal layers, each targeting a different performance requirement. The effectiveness layer encompasses methods that create robust blocks in the context of HHIS, aiming at placing duplicate entities in at least one common block (this directly translates to effectiveness, since entities in the same block will be compared to each other; therefore, the duplicate entities will be discovered). The main technique of this layer is *token blocking*, which requires no background knowledge of the input data, disregarding completely any schema information. In this study, we also propose *attribute clustering blocking*, which creates blocks of higher performance by partitioning attribute names with similar values into nonoverlapping clusters.

The efficiency layer aims at processing blocks efficiently, discarding the repeated and unnecessary comparisons they contain. To describe their functionality in an unambiguous way, we introduce a novel, 2D taxonomy that categorizes efficiency techniques according to the type of comparisons they target, and the granularity of their functionality (i.e., whether they operate on the coarse level of blocks or on the finer level of individual comparisons). We also propose a novel technique, called *comparison scheduling*, which specifies the processing order of individual comparisons so as to increase the number of superfluous comparisons that are discarded at no cost in effectiveness.

The goal of our framework is to facilitate practitioners in their effort to combine complementary blocking methods into highly performing ER solutions that can be easily tailored to the particular settings and requirements of each application (see Section 4.1). It also facilitates and guides the development of new methods that specialize in specific types of comparisons to yield higher efficiency enhancements. Of particular utility in this effort is the metric space we present in Section 3.2, which estimates the performance of blocking methods a-priori. Our framework is general in that it accommodates the existing methods for creating and processing blocks for clean-clean ER over HHIS, and can incorporate new methods as well. To this effect, we have publicly released its implementation, as well as the data of our experimental study.[2] Even though our framework focuses on a particular subtask of ER, most of the ideas it conveys could also be applied to other versions of the ER problem.

Our main contributions are as follows:

- We define a framework for blocking-based clean-clean ER over HHIS that consists of two orthogonal layers. It is generic and flexible to accommodate a variety of methods that in combination form comprehensive, highly performing ER approaches.
- We explain the incorporation of new blocking methods in our framework. For this, we introduce attribute clustering blocking, a novel approach to block building that achieves equally high effectiveness with token blocking, but at a significantly lower redundancy and higher efficiency. We also introduce

1. http://www.google.com/base.

2. http://sourceforge.net/projects/erframework.

comparison scheduling, a novel approach to block processing that enhances efficiency at no cost in effectiveness by specifying the processing order of all individual comparisons so that duplicates are detected first.
- We report evaluation results on two large-scale, real-world data sets that in total comprise over three million entities.

2 Related Work

ER constitutes a traditional problem in computer science [9], [11], [12], [22], [27], and numerous methods have been proposed over the years for tackling it, ranging from string similarity metrics [6] to methods relying on entity relationships [10]. Blocking is one of the established techniques for scaling ER to large data collections, and existing blocking methods can be distinguished in three broad categories: block building, block processing, and hybrid ones.

Block building methods aim at producing a set of blocks that offers a good balance between the number of detected duplicates and the number of required comparisons. In the context of homogeneous information spaces, these methods typically consider the frequency distribution of the values of attribute names, as well as their quality (i.e., presence of noise or missing values), to derive the most suitable *blocking key(s)* [5]. Given a collection of entities, a blocking key is extracted from every profile and blocks are formed on the similarity, or equality of the resulting keys. For example, the suffix array approach [7] considers suffixes of certain lengths of the blocking keys, placing in each block entities that share the corresponding suffix. The StringMap method [20] maps the key of each record to a multidimensional euclidean space, and employs suitable data structures for efficiently identifying pairs of similar records. Bigrams blocking [2] and its generalization, q-grams[3] blocking [15], create clusters of records sharing at least one bi- or q-gram of their keys. Canopy clustering [25] employs a computationally cheap string similarity metric for building high-dimensional, overlapping blocks.

Block processing methods focus on techniques that examine a set of blocks in such a way that effectiveness, or efficiency (or both of them) is enhanced. A typical example in this category is the *iterative blocking* approach, which relies on the repetitive examination of individual blocks. It is based on the principle that more duplicates can be detected and more pairwise comparisons can be saved through the iterative distribution of identified matches to the subsequently (re-)processed blocks. It was originally introduced in [33] and was extended in [21] so that it can accommodate LSH techniques. Another line of research in this area is presented in [30], which proposes a series of techniques for processing overlapping blocks such that no comparison is executed twice.

Hybrid blocking methods deal with the creation and processing of blocks in an integrated way. For example, the sorted neighborhood approach [18] creates blocking keys that are suitable for ordering them in such a way that similar entities are placed in neighboring positions. In another line of research, HARRA [21] introduces a hybrid, LSH-based technique for building blocks and processing them iteratively.

A common drawback of all these methods is that their performance depends on the fine-tuning of many application- and data-specific parameters [5], [7]. To avoid this, tuning methods based on machine learning algorithms have been proposed in the literature [3], [26]. Another common characteristic of most blocking methods is that they are crafted for homogeneous information spaces. As a result, they are able to extract blocking keys of high quality on the condition that schema information about the input data and the properties of its individual attributes are available. However, this assumption is impractical in the context of large-scale HHIS, for which attribute-agnostic blocking methods are needed.

3 Data Model

Our framework operates over collections of entities that describe real-world objects. We follow a recently introduced model [17], [19] that is schema-independent and flexible enough to support a wide spectrum of entity representation formats. It is also capable of representing multivalued attributes as well as entity relationships, thus accommodating any Web and data space application [23]. We assume infinite sets of attribute names $\mathcal{N}$, values $\mathcal{V}$, and identifiers $\mathcal{I}$.

Definition 1. *An* entity collection $\mathcal{E}_i$ *is a tuple* $\langle N_i, V_i, I_i, P_i \rangle$, *where* $N_i \subseteq \mathcal{N}$ *is the set of attribute names appearing in it,* $V_i \subseteq (\mathcal{V} \cup \mathcal{I})$ *is the set of values used in it,* $I_i \subseteq \mathcal{I}$ *is the set of global identifiers contained in it, and* $P_i \subseteq I_i \times \wp(N_i \times V_i)$ *is the set of entity profiles that it comprises. An* entity profile p_i *is a tuple* $\langle i, A_{p_i} \rangle$, *where* A_{p_i} *is the corresponding set of name-value pairs* $\langle n, v \rangle$, *with* $n \in \mathcal{N}$ *and* $v \in (\mathcal{V} \cup \mathcal{I})$.

Among two individually clean entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, two entity profiles, $p \in \mathcal{E}_1$ and $q \in \mathcal{E}_2$, are said to be *matching* if they refer to the same real-world entity. They are collectively called *duplicates* and their relationship is denoted with $p \equiv q$.

Given two duplicate-free, but overlapping entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, clean-clean ER needs to detect the matching entity profiles they contain as effectively (i.e., with high recall) and efficiently (i.e., with few entity comparisons) as possible. This is a problem of quadratic time complexity, as the naive solution compares each entity from the one collection with all entities from the other. To ensure scalability, approximate techniques skip some comparisons, sacrificing effectiveness to a limited and controllable extent. In the following, we consider the most prominent of these techniques, namely, data blocking.

3.1 Blocking-Based Entity Resolution

The goal of blocking is to make ER scalable by grouping similar entities into *blocks* (i.e., clusters) such that it suffices to execute comparisons only between entities of the same block. Blocks are created according to a blocking scheme that consists of two parts: first, a *transformation function* f_t that derives the appropriate representation for blocking

3. A q-gram of a textual value v is a substring of length q.

from every entity profile, and second, a set of *constraint functions* $\mathcal{F}_c$ that encapsulate the conditions for placing entities into blocks. For each block b_i, there is a function $f_c^i \in \mathcal{F}_c$ that decides for every entity profile whether it is going to be placed in b_i or not.

Definition 2. *Given two entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, a* blocking scheme *s comprises a transformation function f_t:$\mathcal{E}_1 \cup \mathcal{E}_2 \mapsto T$ and a set of constraint functions $\mathcal{F}_c$:$T \mapsto$ {true, false}, where T represents the space of all possible blocking representations for the given entity profiles.*

Applying the blocking scheme s on the entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$ yields a set of bilateral blocks $\mathcal{B}$, which is called *block collection*. Each *bilateral block* $b_i \in \mathcal{B}$ is the maximal subset of $\mathcal{E}_1 \times \mathcal{E}_2$ that is defined by the transformation function f_t and the constraint function f_c^i of s. Depending on the origin of its entities, it is internally separated into two nonempty *inner blocks*, $b_{i,1}$ and $b_{i,2}$, where $b_{i,j} = \{p \mid p \in \mathcal{E}_j, f_c^i(f_t(p)) = true\}$.

Given the absence of duplicates in the individual entity collections, it suffices to compare only entities between different inner blocks. In the remaining text, every comparison in a bilateral block b_k between the entities p_i and p_j is denoted by $c_{i,j}$ and requires that $p_i \in b_{k,1}$ and $p_j \in b_{k,2}$. The total number of comparisons entailed in b_i is called *individual cardinality* and is equal to $\|b_i\| = |b_{i,1}| \cdot |b_{i,2}|$, where $|b_{i,j}|$ denotes the number of entities contained in the inner block $b_{i,j}$. The total number of comparisons contained in $\mathcal{B}$ is called *aggregate cardinality* and is symbolized by $\|\mathcal{B}\|$, i.e., $\sum_{b_i \in \mathcal{B}} \|b_i\|$.

Example 1. Consider the entity collections in Fig. 1a. The used transformation function f_t represents each entity as the set of tokens contained in its attribute values. The constraint function $f_c^{91,456}$ places an entity in block $b_{91,456}$ only if token "91,456" is contained in the result given by f_t. Similarly, the participation to the rest of the blocks is defined by the constraint functions f_c^{Antony}, f_c^{Gray}, f_c^{Green}, $f_c^{programmer}$. Consider now block $b_{91,456}$ of Fig. 1b. It can be separated into two inner blocks, i.e., $b_{91,456} = \{b_{91,456,1}, b_{91,456,2}\}$, where $b_{91,456,1} = \{p_1, p_2\}$ and $b_{91,456,2} = \{p_3, p_4\}$.

The performance of a blocking scheme depends on two competing aspects of the blocks it produces: their efficiency and their effectiveness. The former expresses the number of pairwise comparisons a block collection entails and is directly related to the aggregate cardinality of the resulting $\mathcal{B}$. Effectiveness depends on the cardinality of the set $\mathcal{D}_\mathcal{B}$ of the detected pairs of matching entities (i.e., the pairs of matching entities that are compared in at least one block of $\mathcal{B}$). There is a clear tradeoff between these two measures: the more comparisons are executed within $\mathcal{B}$ (i.e., higher $\|\mathcal{B}\|$), the higher its effectiveness gets (i.e., higher $|\mathcal{D}_\mathcal{B}|$), but the lower its efficiency is, and vice versa. Thus, a blocking scheme is considered successful if it achieves a good balance between efficiency and effectiveness. This balance is commonly measured through the following metrics [3], [7], [26], [29]:

Pair completeness (PC) expresses how many of the matching pairs of entities have at least one block in common (otherwise they cannot be detected). It is defined as $PC = |\mathcal{D}_\mathcal{B}|/|\mathcal{D}_{\mathcal{E}_1 \cap \mathcal{E}_2}| \cdot 100\%$, where $|\mathcal{D}_{\mathcal{E}_1 \cap \mathcal{E}_2}|$ denotes the number of entities shared by $\mathcal{E}_1$ and $\mathcal{E}_2$ according to the golden standard. PC takes values in the interval $[0\%, 100\%]$, with higher values indicating higher *effectiveness* of the blocking scheme.

Reduction ratio (RR) measures the reduction in the number of pairwise comparisons contained in a block collection $\mathcal{B}$ with respect to a baseline block collection $\mathcal{B}'$. It is defined as $RR(\mathcal{B}, \mathcal{B}') = (1 - \|\mathcal{B}\|/\|\mathcal{B}'\|) \cdot 100\%$, thus taking values in the interval $[0\%, 100\%]$ (for $\|\mathcal{B}\| \leq \|\mathcal{B}'\|$). Higher values denote higher *efficiency* of the blocking scheme.

This work focuses on blocking methods for overlapping, but individually clean entity collections, defined as follows.

Problem Statement (Blocking-Based Clean-Clean ER). *Given two duplicate-free, but overlapping entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, along with a baseline block collection $\mathcal{B}'$ of high PC value, cluster the entities of $\mathcal{E}_1$ and $\mathcal{E}_2$ into blocks and process them such that both $RR(\mathcal{B}, \mathcal{B}')$ and PC are maximized.*

High RR values mean that the ER process can be efficiently applied to large data sets, while high PC values satisfy the application requirements (i.e., the acceptable level of effectiveness over HHIS). Note that the requirement for maximizing PC and RR simultaneously necessitates that the efficiency enhancements stem from the careful removal of unnecessary comparisons between irrelevant entities, rather than from a blind process. In the following, we address this optimization problem through a set of best effort strategies.

3.2 Metric Space for Clean-Clean ER Blocking Methods

The PC and RR of a given block collection $\mathcal{B}$ can only be measured through a-posteriori examination of its blocks; that is, through the execution of all pairwise comparisons in $\mathcal{B}$. However, estimating their actual values a-priori is crucial for certain tasks, such as the functionality of block processing methods. To cover this need, we now introduce a metric space that provides a close approximation of PC and RR without examining analytically the given block collection $\mathcal{B}$; instead, it simply inspects the external characteristics of its elements (i.e., size and individual cardinality per block).

The *blocking cardinality-comparison cardinality* (BC-CC) *metric space* constitutes a 2D coordinate system that is illustrated in Fig. 2. Its horizontal axis corresponds to *blocking cardinality* (BC) and its vertical one to *comparison cardinality* (CC). BC quantifies the redundancy of a block collection as the average number of *block assignments*[4] per entity. CC is orthogonal to it, deriving the efficiency of a block collection through the distribution of comparisons per block (i.e., the average number of block assignments per comparison). As was experimentally verified in [32], BC is positively correlated with PC (i.e., higher BC values lead to higher effectiveness), while CC is directly related to RR (i.e., higher CC values convey higher efficiency).

The value of BC depends not only on the blocking scheme at hand, but also on the data collection(s) it applies to; the same blocking scheme can yield different levels of

4. A *block assignment* is the association between a block and an entity.

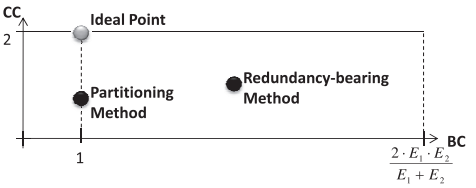


Fig. 2. The BC-CC metric space, illustrating the mapping of the two main categories of blocking methods (black dots) in comparison with the ideal one (gray dot).

redundancy, when applied to different entity collections. Thus, given a block collection $\mathcal{B}$ derived from $\mathcal{E}_1$ and $\mathcal{E}_2$, we distinguish two different versions of BC: the blocking cardinality of the individual entity collections (BC_{ind}) and the blocking cardinality of their conjunction (BC_{ov}).[5] Their formal definitions are, respectively, the following:

Definition 3. *Given a block collection $\mathcal{B}$, the* individual blocking cardinality of $\mathcal{E}_j$ *is defined as the average number of inner blocks $b_{i,j} \in \mathcal{B}$ an entity $p \in \mathcal{E}_j$ is placed in*

$$BC_{\text{ind}}(\mathcal{E}_j) = \frac{\sum_{p \in \mathcal{E}_j} |\{b_i \in \mathcal{B} : p \in b_{i,j}\}|}{|\mathcal{E}_j|} = \frac{\sum_{b_i \in \mathcal{B}} |b_{i,j}|}{|\mathcal{E}_j|},$$

where $j \in \{1, 2\}$ and $|\mathcal{E}_j|$ denotes the size of the entity collection $\mathcal{E}_j$ (i.e., number of entities it contains).

Definition 4. *Given a block collection $\mathcal{B}$, its* overall blocking cardinality *is defined as the average number of blocks $b_i \in \mathcal{B}$ an entity $p \in (\mathcal{E}_1 \cup \mathcal{E}_2)$ is placed in*

$$BC_{\text{ov}} = \frac{\sum_{p \in (\mathcal{E}_1 \cup \mathcal{E}_2)} |\{b_i \in \mathcal{B} : p \in b_i\}|}{|\mathcal{E}_1| + |\mathcal{E}_2|} = \frac{\sum_{b_i \in \mathcal{B}} |b_i|}{|\mathcal{E}_1| + |\mathcal{E}_2|},$$

where $|\mathcal{E}_1| + |\mathcal{E}_2|$ denotes the total size of the given entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, and $|b_i|$ is the size of block b_i.

BC_{ind} and BC_{ov} are defined in the intervals $[0, 2 \cdot |\mathcal{E}_1|]$ and $[0, \frac{2 \cdot |\mathcal{E}_1| \cdot |\mathcal{E}_2|}{|\mathcal{E}_1| + |\mathcal{E}_2|}]$, respectively, since their maximum, reasonable values correspond to the naive method of associating every entity of $\mathcal{E}_1$ with all entities of $\mathcal{E}_2$ in blocks of minimum size (i.e., $\forall b_i \in \mathcal{B} : |b_{i,1}| = 1 \wedge |b_{i,2}| = 1$). Values lower than 1 indicate blocking methods that fail to place each entity in at least one block; this is possible, for example, with blocking techniques that rely on a single attribute name and ignore entity profiles that do not possess it. A value equal to 1 denotes a technique that is close to a *partitioning blocking method* (i.e., one that associates each entity with a single block, thus producing a set of nonoverlapping blocks). Values over 1 indicate *redundancy-bearing blocking methods*, with higher values corresponding to higher redundancy.

CC estimates the efficiency of a block collection through the number of block assignments that account for each comparison; the higher this number is, the more efficient is the given block collection. The rationale behind this approach is that a large set of individually small blocks is substantially more efficient than a set of few, but extremely large blocks that has the same number of block assignments. CC relies, therefore, on the distribution of comparisons per block and depends on both the blocking scheme at hand and the input entity collection(s); that is, the same blocking scheme results in different comparison distributions, when applied to different entity collections. Formally, CC is defined as follows:

Definition 5. *Given a block collection $\mathcal{B}$, its* comparison cardinality *is defined as the ratio between the sum of block sizes and the aggregate cardinality of $\mathcal{B}$ and is given by $CC = \sum_{b_i \in \mathcal{B}} |b_i| \; / \; \|\mathcal{B}\|$.*

CC takes values in the interval $[0, 2]$, with higher values corresponding to fewer comparisons per block assignment, and, thus, higher efficiency (i.e., smaller blocks, on average). Its maximum value $CC_{max} = 2$ corresponds to the ideal case of placing each pair of matching entities in a single block that contains no other entity: $CC = \frac{2 \cdot \mathcal{D}_{\mathcal{B}}}{\mathcal{D}_{\mathcal{B}}} = 2$.[6] On the other hand, a blocking method that places all given entity profiles in a single block corresponds to $CC = \frac{|\mathcal{E}_1| + |\mathcal{E}_2|}{|\mathcal{E}_1| \cdot |\mathcal{E}_2|} \ll CC_{max}$. Thus, the closer CC is to CC_{max}, the more efficient the corresponding blocking method is.

Note that the combination of BC and CC effectively captures the tradeoff between the orthogonal measures of PC and RR: the more redundancy a blocking method entails, the higher its BC gets and, thus, its effectiveness (i.e., PC); the resulting blocks, however, involve a proportionally higher number of pairwise comparisons, downgrading its CC and, thus, its efficiency (i.e., RR). This means that the BC-CC metric space is suitable for a-priori estimation of the balance between PC and RR. The block purging method (see Section 5.2.1) offers an illustrative example of how to exploit this functionality.

The BC-CC metric space can be used for comparing a-priori the performance of blocking schemes, as well. As a reference, we employ the point (1,2) in Fig. 2, which is called *ideal point*. It corresponds to the mapping of the *optimal blocking method*, which builds a block of minimum size for each pair of matching entities (i.e., it involves no unnecessary comparison). The closer a blocking method is mapped to (1,2), the better its performance is [32].

To illustrate this functionality, consider the blocks in Fig. 1b; their PC is equal to 100 percent, while their RR is 0 percent with respect to the Cartesian product of $\mathcal{E}_1$ and $\mathcal{E}_2$. Their poor efficiency is reflected on their BC-CC mapping—the point $(3, 1.5)$—which lies 2.06 away from the ideal point. Imagine now a block processing method that discards all comparisons in the blocks "Gray," "programmer," and "91,456." It has no impact on effectiveness (i.e., PC remains 100 percent), but it reduces the executed comparisons to 2 (i.e., $RR = 50\%$). This efficiency enhancement is clearly depicted at the new BC-CC mapping, which now coincides with the ideal point.

On the whole, two are the main advantages of employing the BC-CC metric space: first, it a-priori approximates the actual performance of a block collection with high accuracy, thus providing insights on how to improve its processing. Second, it allows for a-priori selecting among a collection of

5. Note that the horizontal axis of the BC-CC metric space corresponds to the overall blocking cardinality BC_{ov} of $\mathcal{B}$.

6. CC_{max} also corresponds to any other blocking method that exclusively considers blocks of minimum size: $CC = \frac{2 \cdot |\mathcal{B}|}{|\mathcal{B}|}$, where $\forall b_i \in \mathcal{B} : |b_{i,1}| = 1 \wedge |b_{i,2}| = 1$. In this case, though, BC_{ov} takes its maximum value, as well, thus placing the corresponding blocking method to the farthest point from the ideal one (i.e., (1,2)).

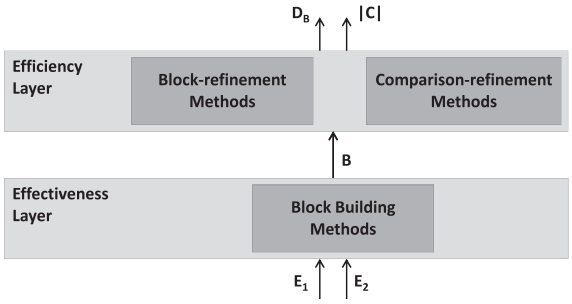


Fig. 3. Outline of our two-layer framework for clean-clean ER.

blocking methods the most appropriate one for the application at hand. Both functionalities involve a negligible computational cost, as the corresponding metrics are computed in linear time—$O(|\mathcal{B}|)$—through a single pass over the given blocks.

4 Blocking Framework for Entity Resolution

Our framework for blocking-based clean-clean ER over HHIS is depicted in Fig. 3. It consists of two orthogonal, but complementary layers: the effectiveness layer that groups entity profiles into blocks to achieve high PC, and the efficiency layer that aims at achieving high RR.

The effectiveness layer encompasses a set of blocking schemes that build blocks of high robustness in the context of HHIS. Their *input* consists of the two duplicate-free entity collections that are to be resolved, $\mathcal{E}_1$ and $\mathcal{E}_2$, while their *output* comprises the block collection $\mathcal{B}$ that results after applying one of the available blocking schemes on $\mathcal{E}_1$ and $\mathcal{E}_2$. To achieve high PC over HHIS, the block building methods of this layer typically have the following two characteristics: First, *attribute-agnostic functionality*, disregarding any a-priori knowledge about the schemata of the input entity profiles so as to ensure their applicability to HHIS. Second, *redundancy-bearing functionality*, placing each entity in multiple blocks; this guarantees the high BC_{ov} that is required for reducing the likelihood of missed matches (i.e., high PC), but produces a set of overlapping blocks that involves unnecessary comparisons.

The efficiency layer takes as *input* the set of blocks $\mathcal{B}$ that is derived from the effectiveness layer. Its *output* comprises the detected pairs of duplicates $\mathcal{D}_{\mathcal{B}}$, along with their cost in terms of the number of executed comparisons; in the following, we denote this measure by $|\mathcal{C}|$, where $\mathcal{C}$ is the set of all executed comparisons $c_{i,j}$. The goal of this layer is to enhance efficiency (i.e., RR) by reducing the cardinality of $\mathcal{C}$ at a controllable impact on PC. This can be accomplished by removing entire blocks or individual comparisons, a practice that moves the BC-CC mapping of a blocking method closer to the ideal point; as depicted in Fig. 4, its BC_{ov} value decreases toward the $x = 1$ axis, since its numerator decreases, while its denominator remains stable. On the other hand, its CC value increases, since its denominator decreases faster than its numerator.

To ensure high performance, the efficiency layer encompasses a set of techniques that target specific types of pairwise comparisons. Given a bilateral block $b_k \in \mathcal{B}$, every

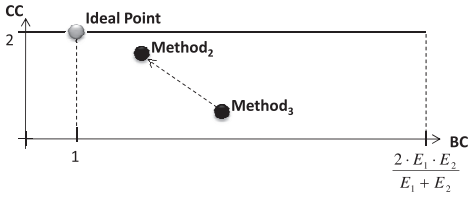


Fig. 4. Illustration of the effect of the efficiency techniques on the BC-CC mapping of a blocking method.

pairwise comparison $c_{i,j}$ it entails belongs to one of the following types:

1. *Matching comparison*, if $p_i \equiv p_j$.
2. *Repeated comparison*, if p_i and p_j have already been compared in a previously examined block.
3. *Superfluous comparison*, if p_i or p_j or both of them have been matched to some other entity profile and cannot be duplicates (i.e., clean-clean ER).
4. *Nonmatching comparison*, if $c_{i,j}$ is neither repeated nor superfluous and $p_i \not\equiv p_j$.

Based on this taxonomy, the goal of the efficiency layer is threefold: 1) to eliminate the repeated comparisons, 2) to discard all the superfluous comparisons, and 3) to restrict the execution of nonmatching comparisons.

The first two targets can be achieved without any effect on the matching comparisons and, thus, PC. This does not apply, though, to the third target: there is no safe way to determine whether two entities are duplicates or not, without actually comparing their profiles. Therefore, methods that target nonmatching comparisons are inherently approximate and partially discard matching comparisons, as well, incurring lower PC.

In this context, the block processing methods of the efficiency layer can be categorized according to the comparison type they target as follows:

1. *Repeat methods*, which aim at discarding repeated comparisons without affecting PC,
2. *Superfluity methods*, which try to skip superfluous comparisons without any impact on PC,
3. *Nonmatch methods*, which target nonmatching comparisons at a limited and controllable cost in PC, and
4. *Scheduling methods*, which enhance efficiency in an indirect way, specifying the processing order that boosts the effect of superfluity and nonmatch methods.

A complete clean-clean ER approach should encompass techniques of all these types to ensure high-efficiency enhancements. Combined with a block building method, such a collection of complementary efficiency methods is called *ER workflow*. Its composition typically depends on two factors: 1) The resources that are available for handling the time and space requirements of the selected efficiency methods, and 2) the performance requirements of the underlying application with respect to both PC and RR.

To facilitate the compilation of blocking methods into highly performing workflows, we introduce an additional categorization of block processing methods according to the granularity of their functionality:

		Repeat method	Superfluity method	Non-match method	Scheduling method
Block-refinement	Block Scheduling				X
	Block Purging			X	
	Block Pruning			X	
Comparison-refinement	Comparison Scheduling				X
	Comparison Pruning			X	
	Duplicate Propagation		X		
	Comparison Propagation	X			

Fig. 5. Taxonomy of efficiency methods according to the type of comparisons they target and the granularity of their functionality.

1. *block-refinement methods*, which operate at the coarse level of individual blocks, and
2. *comparison-refinement methods*, which operate at the finer level of individual comparisons.

The granularity of functionality constitutes a decisive parameter for both factors affecting the composition of ER workflows. Block-refinement methods exhibit limited accuracy when discarding comparisons, but they consume minimal resources, as they typically involve low time and space complexity. Thus, they offer the best choice for applications with limited resources, where entity comparisons can be executed in short time (e.g., due to entity profiles of small size). On the other hand, comparison-refinement techniques are more precise in the identification of unnecessary comparisons, but their higher accuracy comes at the cost of higher time and space complexity. They are suitable, therefore, for applications with time-consuming entity comparisons (e.g., due to large profiles), which can afford high complexity block processing.

On the whole, the comparisons' type and the granularity of functionality define a 2D taxonomy of efficiency methods that facilitates the combination of blocking methods into comprehensive ER workflows. Its outline is illustrated in Fig. 5, along with a complete list of the techniques that are analyzed in Section 5.2.

We stress that all efficiency methods of our framework share the same interface: they receive as input a block collection and return as output an improved one that involves fewer blocks, or fewer comparisons, or has its elements appropriately ordered.[7] In this way, an ER workflow can be simply created by specifying the methods that are included in it; regardless of its composition, its methods are applied consecutively, in the order they are added, so that the output of the one constitutes the input of the other. We elaborate on the creation of such workflows in the following section.

4.1 Using Blocking Framework to Build ER Workflows

As mentioned above, a core characteristic of our framework is its *flexibility* in combining blocking methods into highly performing ER workflows. The choice of the methods comprising them is only limited by the available resources and the performance requirements of the underlying application. In this section, we introduce a general procedure for composing ER workflows that can cover a variety of performance and resource requirements. It consists of five steps, outlined in Fig. 6, which are all optional—with the exception of the first one (i.e., the creation of blocks).[8] We elaborate on each step in the following.

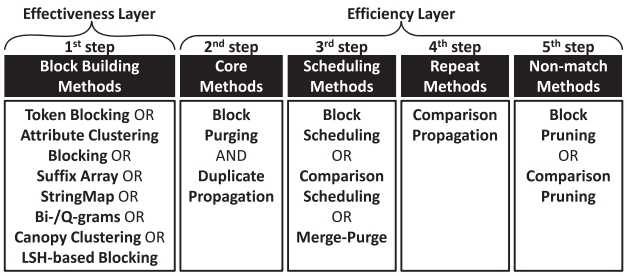


Fig. 6. Steps for creating a complete blocking-based ER approach.

The first step selects the most suitable block building method for the application at hand. Given that all methods of the effectiveness layer are competitive to each other, serving exactly the same need, it suffices to include only one of those depicted in the left-most column of Fig. 6.

The second step is to include the two core efficiency methods: block purging (see Section 5.2.1) and duplicate propagation (see Section 5.2.2). They are indispensable for an ER workflow, since they consume minimal resources, while yielding significant improvements in efficiency at a negligible cost in PC.

The third step opts for a scheduling method, which determines the processing order of blocks or comparisons that boosts the performance of duplicate propagation and block pruning (where applicable). Three are the valid options: block scheduling (see Section 5.2.1), comparison scheduling (see Section 5.2.2), and the merge-purge algorithm, on the condition that it is applicable to HHIS. Block Scheduling is better integrated with block-refinement efficiency techniques, whereas comparison scheduling exclusively operates in conjunction with comparison-refinement ones. Thus, the scheduling method constitutes a critical part of an ER workflow, determining its overall granularity of functionality and, consequently, its complexity and performance.

The fourth step incorporates the technique that eliminates all repeated comparisons, i.e., comparison propagation (see Section 5.2.2). Due to its high space complexity, it should be skipped in the case of ER workflows that can only afford minimal space requirements (i.e., workflows that exclusively involve block-refinement methods).

The last step determines the technique that—in addition to block purging—deals with nonmatching comparisons. The options can be restricted, though, by the method selected in the third step; workflows involving block scheduling can choose between block pruning and comparison pruning, whereas those involving comparison scheduling can only opt for comparison pruning. Note that in the latter case, it is good practice to add comparison propagation, as well, since it shares exactly the same space and time complexity with comparison pruning.

7. The only method that does not comply with this interface is duplicate propagation, which in practice operates as a data structure (see Section 5.2.2).

8. Note that the same procedure can be applied to Dirty ER, as well, by excluding the third step (i.e., scheduling methods) and duplicate propagation from the second step. All other blocking techniques merely need to adapt their internal functionality to *unilateral blocks* (i.e., blocks where all entities are comparable to each other).

As stressed in the previous section, the actual execution order of the methods comprising an ER workflow coincides with the order they are added to it. This rule applies to the procedure of Fig. 6 with one exception: Comparison Scheduling is added at the third step, but is the last to be executed in the workflows that involve it. Duplicate propagation constitutes a special case, since it is integrated into the entity comparison process, thus being executed together with the last method of each workflow.

4.2 Existing Methods in Blocking Framework

Any block building technique can be incorporated into the effectiveness layer, regardless of its internal functionality (e.g., whether it is signature-based or not), on the sole condition that it shares the same interface; that is, it should receive as input two clean, but overlapping entity collections and should return as output a set of bilateral blocks. Note, however, that the intricacies of HHIS are usually tackled through an attribute-agnostic functionality that employs redundancy. Given that the suffix array [7], the StringMap [20], and the q-grams [15] blocking methods already involve a redundancy-bearing functionality, they only need to be adapted such that they operate in an attribute-agnostic manner; that is, instead of deriving the blocking key(s) from the values of selected attributes, they should apply on all values of entity profiles. The same applies to canopy clustering [25]; a string similarity metric that considers the entity profiles in their entirety turns it suitable for our framework.

On the other hand, block processing methods can be readily integrated into the efficiency layer. The core method for eliminating redundant comparisons (i.e., comparison propagation [30]) has already been added, and so does part of the iterative processing method of [33] (i.e., duplicate propagation).

Hybrid blocking methods can be added, as well, after dividing their functionality in two separate processes that can be mapped to the respective layers: the creation of blocks and their processing. For instance, the LSH-based, block building technique of the HARRA framework [21] could be integrated into the effectiveness layer, whereas its iterative processing fits in the efficiency one. Similarly, decoupling the schema-specific functionality from the block building technique of the merge-purge algorithm [18] turns it suitable for the effectiveness layer, while its ordering technique can be mapped to the efficiency layer (see Fig. 6).

Equally important is the *extensibility* of our framework. Novel methods can be seamlessly plugged into it on the sole condition that they implement the same interface and serve the same goal as the corresponding layer. Methods fulfilling this requirement can be integrated into an ER workflow without any further modifications. To facilitate the development of such new methods, Sections 5.1 and 5.2 analyze the functionality of existing techniques and explain how the BC-CC metric space can be used to guide this process.

5 Approach

5.1 Effectiveness Layer

This layer currently encompasses two block building techniques: token blocking (see Section 5.1.1), the only existing blocking method that is applicable in the settings we are considering, and attribute clustering blocking (see Section 5.1.2), which is a novel blocking technique that improves on token blocking. They both entail an attribute-agnostic and redundancy-bearing functionality, being mapped to the right of the $x = 1$ axis on the BC-CC metric space.

5.1.1 Token Blocking

Token blocking [29] is based on the following idea: every distinct token t_i creates a separate block b_i that contains all entities having t_i in the values of their profile—regardless of the associated attribute names. The only condition is that t_i is shared by both input sets of attribute values, so that the resulting inner blocks are nonempty: $t_i \in (tokens(V_1) \cap tokens(V_2))$, where $tokens(V_j)$ represents the set of all tokens contained in the values V_j of the entity profiles in collection $\mathcal{E}_j$. In this way, blocks are built independently of the attribute names associated with a token (attribute-agnostic functionality), and each entity is associated with multiple blocks (redundancy-bearing functionality).

More formally, the *transformation function* f_t of this scheme converts an entity profile into the set of tokens comprising its attribute values: $f_t(p) = \{t_i : \exists n_i, v_i : \langle n_i, v_i\rangle \in A_p \wedge t_i \in tokens(v_i)\}$, where $tokens(v_i)$ is a function that returns the set of tokens comprising the value v_i. Its set of *constraint functions* $\mathcal{F}_c$ contains a function f_c^i for every token t_i that is shared by both input entity collections (i.e., $t_i \in (tokens(V_1) \cap tokens(V_2))$); f_c^i defines a block $b_i \in \mathcal{B}$ that contains all entities of $\mathcal{E}_1$ and $\mathcal{E}_2$ having t_i in at least one of their values. Thus, every f_c^i encapsulates the following condition for placing an entity p in block b_i: $f_c^i(f_t(p)) = (t_i \cap f_t(p)) \neq \emptyset$, where $p \in (\mathcal{E}_1 \cup \mathcal{E}_2)$. On the average case, the time complexity of this method is $O(BC_{\text{ov}} \cdot (|\mathcal{E}_1| + |\mathcal{E}_2|))$, while its space complexity is $O(|\bar{b}_i| \cdot (tokens(V_1) \cap tokens(V_2)))$, where $|\bar{b}_i|$ is the mean block size.

This approach has two major performance advantages: first, it can be efficiently implemented with the help of inverted indices, even in the case of large entity collections. Second, it is robust to noise and heterogeneity, because the likelihood of two matching entities sharing no block at all is very low. Indeed, this can only be the case when two matching entities have no token in common, a very unlikely situation for profiles describing the same real-world entity.

5.1.2 Attribute Clustering Blocking

We now describe attribute clustering, a novel blocking scheme that we introduce in this study, which exploits patterns in the values of attribute names to produce blocks that offer a better balance between PC and RR. At its core lies the idea of partitioning attribute names into nonoverlapping clusters, according to the similarity of their values. The resulting groups, denoted by K, are called *attribute clusters* and are treated independently of each other: given a cluster $k \in K$, every token t_i of its values creates a block containing all entities having t_i assigned to an attribute name belonging to k. As a result, the partitioning of attribute names into clusters leads to the partitioning of tokens into clusters, as well. Compared to token blocking, the resulting block collection $\mathcal{B}$ is larger in size (i.e., contains more blocks), but of lower aggregate cardinality (i.e., contains

smaller blocks on average)—assuming that they are both applied to the same input entity collections. Therefore, attribute clustering is expected to involve higher CC values than token blocking, while maintaining similar values of BC. This means that its BC-CC mapping lies closer to the Ideal Point, offering a PC-RR balance of higher efficiency.

To understand the difference of this approach from the previous one, consider a token t_i that is associated with n attribute names, which belong to k attribute clusters. Token blocking creates a single block for t_i, with all entities that have it in their values; that is, regardless of the associated attribute names. On the other hand, attribute clustering blocking creates k distinct blocks—one for each attribute cluster; each block contains all entities having at least one attribute name that is associated with t_i *and* belongs to the corresponding cluster. Given that the number of associated entities remains the same in both cases, the blocks of attribute clustering are expected to be more and individually smaller, thus having a higher CC value than Token Blocking. In fact, the higher k is, the higher is the resulting value of CC.

The functionality of attribute clustering is outlined in Algorithm 1. In essence, it works as follows: each attribute name from N_1 is associated with the *most similar* attribute name of N_2 (Lines 2-5), and vice versa (Line 6). The link between two attribute names is stored in a data structure (Line 5) on the sole condition that the similarity of their values exceeds zero (Line 4), a value that actually implies dissimilarity. The transitive closure of the stored links is then computed (Line 7) to form the basis for partitioning attribute names into clusters: each connected component of the transitive closure corresponds to an attribute cluster (Line 8). The resulting attribute clusters are examined for *singleton clusters*, which contain a single attribute name that was associated with no other. All these clusters are merged into a new one, called the *glue cluster* and symbolized as k_{glue} (Line 10). In this way, we ensure that no attribute names, and, thus, no tokens are excluded from the block building procedure.

Algorithm 1. Attribute Clustering Blocking

Input: Attribute name sets: N_1, N_2, Attribute values: V_1, V_2
Output: Set of attribute names clusters: K

1 $links \leftarrow \{\}$; $k_{glue} \leftarrow \{\}$;
2 **foreach** $n_{i,1} \in N_1$ **do**
3 $n_{j,2} \leftarrow getMostSimilarAttribute(n_{i,1}, N_2, V_2)$;
4 **if** $0 < sim(n_{i,1}.getValues(), n_{j,2}.getValues())$ **then**
5 $links.add(newLink(n_{i,1}, n_{j,2}))$;
6 **foreach** $n_{i,2} \in N_2$ **do** ... ; // same as with N_1
7 $links' \leftarrow computeTransitiveClosure(links)$;
8 $K \leftarrow getConnectedComponents(links')$;
9 **foreach** $k_i \in K$ **do**
10 **if** $|k_i| = 1$ **then** $K.remove(k_i)$; $k_{glue}.add(k_i)$;
11 $K.add(k_{glue})$;
12 **return** K;

The time complexity of the overall procedure is $O(|N_1| \cdot |N_2|)$, while its space complexity is $O(|N_1| + |N_2|)$, where $|N_1|$ and $|N_2|$ stand for the number of distinct attribute names in $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively. Note that at the core of attribute clustering lies an attribute-agnostic functionality, which partitions attribute names into clusters without considering schema information at all; instead, it merely relies on the similarity of their values. Similar to token blocking, it is based on redundancy, as well, associating each entity with multiple blocks.

We note that attribute clustering is different from schema matching techniques in three aspects. First, the latter are inapplicable to HHIS [29]. Second, our goal differs from that of schema matching; instead of trying to partition the input set of attribute names into clusters of semantically equivalent attributes, we rather aim at deriving attribute clusters that produce blocks with a comparison distribution that has a short tail (i.e., high CC values). Third, our algorithm associates singleton attributes with each other, a practice that is incompatible with the goal of schema matching.

Attribute name representation models. The functionality of attribute clustering relies on two components: 1) the model that uniformly represents the values of an attribute name, and 2) the similarity measure that captures the common patterns between the values of two attribute names. We consider the following established techniques for text classification (their performance is reported in Section 6):

1. The *term vector* representation model in conjunction with the *cosine similarity* metric. According to this model, the input sets of values, V_1 and V_2, form a Cartesian space, where each dimension corresponds to a distinct token contained in both of them. Thus, each attribute name is represented by a (sparse) vector whose ith coordinate denotes the $TF(t_i) \times IDF(t_i)$ weight of the corresponding token t_i [24]. $TF(t_i)$ stands for the *term frequency* of t_i (i.e., how many times t_i appears in the values of the attribute name), while $IDF(t_i)$ is equal to $\log(|N|/|N(t_i)|)$, where $N(t_i) \subseteq N$ stands for the set of attribute names containing t_i. The similarity of two attribute names is defined as the cosine similarity of the corresponding vectors.
2. The *character n-grams* representation model in conjunction with the *Jaccard similarity* metric. This model represents each attribute name as the set of n-grams (i.e., substrings of n consecutive characters) that appear in its values. The value of n is typically set equal to 3 (i.e., *trigrams*); in this way, the value $v =$ ''home phone'' is represented as $\{hom, ome, me\_, e\_p, \_ph, pho, hon, one\}$. The similarity between two attribute names n_i and n_j is defined as their Jaccard similarity:

$$J(n_i, n_j) = \frac{|trigrams(n_i) \cap trigrams(n_j)|}{|trigrams(n_i) \cup trigrams(n_j)|},$$

where function $trigrams(n_k)$ produces the trigrams representation of the attribute name n_k.
3. The *n-gram graphs* representation model [13] in conjunction with their *value similarity* metric. This model is richer than the character n-grams model, since it additionally incorporates contextual information by using edges to connect *neighboring n-grams*: these are n-grams that lie within a sliding window of n characters. Similar to the above

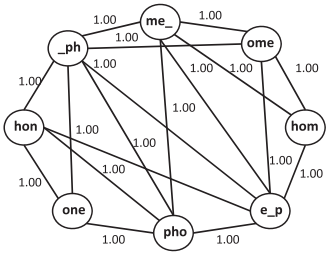


Fig. 7. The trigram graph for value "home_phone."

method, n is usually set equal to 3. To illustrate their functionality, the graph for the value $v =$ "home phone" is shown in Fig. 7. Individual n-gram graphs are combined in a single graph comprising the union of the nodes and edges of the original graphs, with the edges weighted with the mean value of the original weights [14]. To estimate the relevance of two n-gram graphs, we employ their *value similarity*, a graph metric that essentially expresses the portion of common edges sharing the same weight.

5.2 Efficiency Layer

Similar to the effectiveness layer, the efficiency layer internally consists of two parts: 1) the algorithms that define the processing of the given block collection, and 2) the data structures that facilitate their functionality. A typical example of the latter is the entity index, which associates each entity with the blocks containing it (see Fig. 8). On the other hand, the algorithms' part encompasses a wide diversity of efficiency techniques, with each one targeting a particular category of comparisons. We now review the best performing methods in the related literature, and introduce a novel approach, comparison scheduling.

5.2.1 Block-Refinement Methods

Block purging. The notion of block purging was introduced in [29] as a means of discarding nonmatching comparisons by removing *oversized blocks*. These are blocks that contain an excessively high number of comparisons, although they are highly unlikely to contain *nonredundant duplicates*, i.e., matching entities that have no other—smaller—block in common. Thus, they decrease RR, but have a negligible contribution to PC. The gist of block purging is, therefore, to specify a conservative upper limit on the individual cardinality of the processed blocks so that oversized ones are discarded without any significant impact on PC. This limit is called *purging threshold*.

For our framework, we adapted the method that was employed in [32] for determining the purging threshold in the case of dirty ER. It relies on the CC metric and the following observation, in particular: assuming that blocks are sorted in descending order of individual cardinality, the value of CC increases when moving from the top block to the ones in the lower ranking positions. The reason is that its denominator (i.e., aggregate cardinality) decreases faster than its numerator (i.e., number of block assignments). The purging threshold is specified as the first individual cardinality that has the same CC value with the next (smaller) one; discarding blocks with fewer comparisons can only reduce PC, while having a negligible effect—if any—on RR.

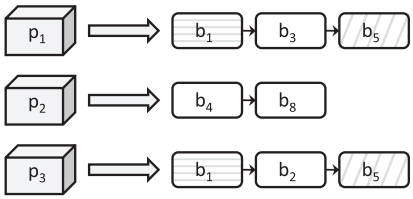


Fig. 8. The entity index employed by comparison propagation.

The outline of this approach is presented in Algorithm 2. Line 1 orders the given block collection $\mathcal{B}$ in ascending order of individual cardinality, thus making it possible to calculate the CC for each distinct cardinality with a single pass (Lines 4-10). Lines 11-12 ensure that the last block is also considered in the computation of the statistics. Starting from the largest individual cardinality, the CC values of consecutive ones are then compared (Lines 14-17). The procedure stops as soon as the value of CC remains stable (Lines 15-17).

Algorithm 2. Computing the Purging Threshold

```
   Input: Set of blocks: B
   Output: Purging threshold: maxICardinality
1  B' ← orderByICardinality(B);
2  blockAssignments ← 0;   index ← 0;
3  totalComparisons ← 0;   lastICardinality ← 1;   stats[] ← {};
4  foreach b_i ∈ B' do
5      if lastICardinality < ||b_i|| then
6          stats[index].iCardinality = lastICardinality;
7          stats[index].cc = blockAssignments / totalComparisons;
8          index++;
9          lastICardinality = ||b_i||;
10     blockAssignments += |b_i|;   totalComparisons += ||b_i||;
11 stats[index].iCardinality = lastICardinality;
12 stats[index].cc = blockAssignments / totalComparisons;
13 maxICardinality = lastICardinality;
14 for i ← stats.size()-1 to 1 do
15     if stats[i].cc=stats[i − 1].cc then
16         maxICardinality=stats[i].iCardinality;
17         break;
18 return maxICardinality;
```

Apparently, the time complexity of this algorithm is dominated by the initial sorting and is equivalent to $O(|\mathcal{B}| \cdot log|\mathcal{B}|)$. Its space complexity is dominated by the array that stores the statistics for every individual cardinality and is equal to $O(|\mathcal{B}|)$.

Block scheduling. This technique was introduced in [29] as a means of sorting the input block collection $\mathcal{B}$ so that its processing makes the most of duplicate propagation (see Section 5.2.2). To this end, it associates each block b_i with a *block utility* value, $u(b_i)$, which expresses the tradeoff between the cost of processing it, $cost(b_i)$, and the corresponding gain, $gain(b_i)$. The former corresponds to the number of comparisons entailed in b_i (i.e., $cost(b_i) = \|b_i\|$), while the latter pertains to the number of superfluous comparisons that are spared in the subsequently examined blocks—due to the propagation of detected duplicates. The actual value of the block utility $u(b_i)$ for a bilateral block

$b_i \in \mathcal{B}$ has been estimated through a probabilistic analysis to be equal to:

$$u(b_i) = \frac{gain(b_i)}{cost(b_i)} \approx \frac{1}{max(|b_{i,1}|, |b_{i,2}|)}.$$

To incorporate this measure in the processing of blocks, we employ a *ranking function* $r : \mathcal{B} \mapsto \Re$ that defines a partial order on $\mathcal{B}$, sorting its elements in descending order according to the following implication: $u(b_i) \leq u(b_j) \Rightarrow r(b_i) \geq r(b_j)$. Therefore, its complexity is equal to $O(|\mathcal{B}| \cdot log|\mathcal{B}|)$, while its space complexity is $O(|\mathcal{B}|)$.

Block pruning. This method, coined in [29], constitutes a coarse-grained approach to saving nonmatching comparisons. Instead of examining the entire block collection, it terminates the ER process prematurely, at a point that ensures a good tradeoff between PC and RR.

The functionality of this method relies on the block processing order defined by block scheduling; this ordering ensures that blocks placed at the highest ranking positions offer high expected gain at a low cost. In other words, blocks that are processed earlier involve a low number of comparisons, while entailing a high number of duplicates. In contrast, the lower the ranking position of a block is, the fewer the duplicates it contains and the more nonmatching comparisons it involves. Therefore, blocks placed at the low ranking positions are unlikely to contain new, yet unidentified duplicates. This means that there is a break-even point where the possibility of finding additional matches is no longer worth the cost; blocks lying after this point can be excluded from the ER process to enhance its efficiency (i.e., RR) at a negligible cost in the missed matches (i.e., small decrease in PC).

Block pruning aims at approximating this point to discard blocks dominated by nonmatching comparisons. It keeps track of the evolution of *duplicate overhead*, h, which assesses the (average) number of comparisons that were performed to detect the latest match(es). Its value after processing the kth block *containing duplicates* is defined as: $h_k = |\mathcal{C}_{k-1}|/|\mathcal{D}_k|$, where $|\mathcal{C}_{k-1}|$ represents the number of comparisons performed after processing the $k-1$th block *with duplicates*, and $|\mathcal{D}_k|$ stands for the number of *new* matches identified within the latest block (i.e., $|\mathcal{D}_k| \geq 1$).

As explained in [29], h takes low values (close to 1) for the blocks placed at the top ranking positions; that is, every new pair of duplicates they contain merely requires a small number of comparisons. Its value increases for duplicates discovered in blocks of lower ranking positions. As soon as it exceeds the *maximum duplicate overhead*—a predefined threshold denoted by h_{max}—the entire ER process is terminated; this indicates that the cost of detecting new duplicates is excessively high and the few remaining matches are not worth it. Although this threshold can be adapted to the requirements of the application at hand, a value that provides a good estimation of the break-even point was experimentally derived from $h_{max} = 10^{log\|\mathcal{B}\|/2}$, where $\|\mathcal{B}\|$ is the aggregate cardinality of the input block collection $\mathcal{B}$. The intuition behind this formula is that the comparisons required for detecting a match are considered too many, when they reach half the order of magnitude of all possible comparisons in the considered blocks.

Given that block pruning can be integrated in block processing, its time complexity is equal to $O(|\mathcal{B}|)$, where $|\mathcal{B}|$ is the number of blocks remaining after block purging.

5.2.2 Comparison-Refinement Methods

Comparison propagation. This method, introduced in [30], constitutes a general technique for discarding all repeated comparisons from any set of blocks, without any impact on PC. In essence, it propagates all executed comparisons indirectly, avoiding the need to explicitly store them. Its functionality relies on two pillars: the process of *block enumeration* and the data structure of *entity index* (EI). The former is a preparatory step that assigns to each block a unique index, indicating its processing order. As a result, b_i symbolizes the block placed in the ith position of the processing list. On the other hand, EI constitutes a structure that points from entities to the blocks containing them (see Fig. 8). It is actually a *hash table*, whose *keys* correspond to entity ids, while each *value* lists the indices of the blocks that contain the corresponding entity.

A comparison $c_{i,j}$ is recognized as repeated if the *least common block index* condition ($LeCoBI$) does not hold. This condition ensures that the current block is the first to contain both entities p_i and p_j. It returns `true` only if their lowest common block index is equal to the current block's index. Otherwise, if the least common index is lower than the current one, the entities have already been compared in another block, and the comparison should be discarded as redundant.

As an example, consider the entities p_1 and p_3 in Fig. 8. Two blocks are in common, namely, b_1 and b_5 and, thus, their LeCoBI is 1. This means that the $LeCoBI$ condition is satisfied in b_1, but not in b_5, saving in this way the repeated comparison of p_1 and p_3 in the latter case.

The examination of the $LeCoBI$ condition is linear with respect to the total number of blocks associated with a pair of entities. This is achieved by iterating once and in parallel over the two lists of block indices, after sorting them individually in ascending order. For higher efficiency, this sorting is executed only once, during the construction of the EI.

The time complexity for building this data structure is linear with respect to the number of given blocks and the entities contained in them; in the average case, it is equal to $O(BC_{\text{ov}} \cdot |\mathcal{B}|)$. Its space complexity, on the other hand, is linear with respect to the size of the input entity collections, depending, of course, on the overall level of redundancy; on average, it is equal to $O(BC_{\text{ov}} \cdot (|\mathcal{E}_1| + |\mathcal{E}_2|))$.

Duplicate propagation. This method is inspired from the technique introduced in [33] as a means of increasing PC in the context of dirty ER. It was adapted to clean-clean ER in [29], which converted it into a method that reduces the superfluous comparisons at no cost in PC. In this form, it relies on a central data structure, called *duplicates index* (DI), that contains at any time the profile ids of all the entities that have already been matched to another one. Before performing a comparison $c_{i,j}$, we check whether either of the entities p_i and p_j is contained in DI. If this applies to at least one of them, duplicate propagation discards the comparison as superfluous. Otherwise, if none of them is contained in DI, the comparison is executed. Note, though,

that the performance of this technique (i.e., the portion of superfluous comparisons that are discarded) depends on the block processing order. To boost its effect, it is typically employed in conjunction with a scheduling method.

Its time complexity is constant, i.e., $O(c)$, as it merely involves a couple of look-ups in a hash-table. Its space complexity depends on the size of the hash table of DI. It is, therefore, equal to the cardinality of the set of duplicates contained in the given block collection: $O(|D_{\mathcal{E}_1 \cap \mathcal{E}_2}|)$.

Comparison pruning. This technique was initially introduced in [31], offering another method to discard non-matching comparisons at a controllable cost in effectiveness (i.e., PC). It can be conceived as an improved version of block pruning, which, instead of considering entire blocks, operates on the level of individual comparisons: it prunes a comparison if the involved entities are deemed highly unlikely to be a match. Its decision relies exclusively on the blocks associated with the given entities and their overlap, in particular.

In more detail, the overlap of two entities p_i and p_j is called *entities similarity*—symbolized by $ES(p_i, p_j)$—and is defined as the Jaccard similarity of the list of block indices that are associated with them. Thus, it is derived from the following formula:

$$ES(p_i, p_j) = \frac{|indices(p_i) \cap indices(p_j)|}{|indices(p_i) \cup indices(p_j)|} = \frac{|indices(p_i) \cap indices(p_j)|}{|indices(p_i)| + |indices(p_j)| - |indices(p_i) \cap indices(p_j)|},$$

where $indices(p_k)$ denotes the set of block indices associated with the entity profile p_k. This formula indicates that we only need to estimate the number of indices that are shared by p_i and p_j to compute $ES(p_i, p_j)$. As explained above, this process is facilitated by EI and is linear with respect to the total number of indices: it suffices to iterate over the two lists of indices just once and in parallel, due to their sorting in ascending order.

A pair of entities, p_i and p_j, is considered similar enough to justify the comparison of their profiles if $ES(p_i, p_j)$ exceeds the predefined threshold that represents the minimum allowed similarity value, denoted by ES_{min}. The actual value of its threshold depends on the redundancy of the individual entity collection(s) and is derived from the following formula:

$$ES_{min} = \frac{a \cdot min(BC_{\text{ind}}(\mathcal{E}_1), BC_{\text{ind}}(\mathcal{E}_2))}{BC_{\text{ind}}(\mathcal{E}_1) + BC_{\text{ind}}(\mathcal{E}_2) - a \cdot min(BC_{\text{ind}}(\mathcal{E}_1), BC_{\text{ind}}(\mathcal{E}_2))}, \quad (1)$$

where a takes values in the interval $(0, 1]$. Intuitively, this threshold demands that two entities are analytically compared if their common blocks amount to $a \cdot 100\%$ of the minimum individual blocking cardinality (i.e., the average number of blocks an entity of the collection with the lowest level of redundancy is placed in). As demonstrated in [31], the performance of comparison pruning is robust to the fluctuation of a, with higher values corresponding to stricter similarity conditions, and vice versa.

Given that comparison pruning relies on the same data structures and operations as comparison propagation, it shares the same space and time complexity with it.

Comparison scheduling. We now introduce a novel technique that aims at reducing the superfluous comparisons to increase RR at no cost in PC. Similar to block scheduling, it achieves its goal indirectly, by boosting the effect of duplicate propagation. However, it is more effective than block scheduling, due to the finer granularity of its functionality: instead of handling entire blocks, it considers individual comparisons, ordering them in such a way that those involving real matches are executed first. Thus, more superfluous comparisons are saved in the subsequently processed blocks.

To this end, it first gathers the set of *valid comparisons*, which is denoted by $\mathcal{C}_v$ and encompasses all pairwise comparisons of $\mathcal{B}$ that remain after filtering the initial set of blocks with a combination of the aforementioned efficiency methods (typically, comparison propagation, and comparison pruning). Then, it associates each pairwise comparison $c_{i,j}$ with a *comparison utility* value, $u(c_{i,j})$, which—similar to the block utility value—is defined as $u(c_{i,j}) = gain(c_{i,j})/cost(c_{i,j})$; the denominator corresponds to the cost of executing $c_{i,j}$ which is unary for all comparisons (i.e., $cost(c_{i,j}) = 1$). Thus, $u(c_{i,j}) = gain(c_{i,j})$, where $gain(c_{i,j})$ represents the likelihood that the entities to be compared, p_i and p_j, are matching. Several approaches are possible for estimating $gain(c_{i,j})$; in this work, we consider a best effort scoring mechanism that is derived from the following measures:

1. The *entities similarity* $ES(p_i, p_j)$, which is the same measure employed by Comparison Pruning, i.e., the portion of common blocks between entities p_i and p_j. The higher its value is, the more likely are p_i and p_j to be matching. Hence, $u(c_{i,j})$ is proportional to $ES(p_i, p_j)$.
2. The *Inverse Comparison Frequency* (ICF) of each entity. Following the same rationale as the Inverse Document Frequency of Information Retrieval, this metric is based on the idea that the more *valid* comparisons are associated with a specific entity, the less likely it is to be matching with one of the associated entities. In other words, the lower the number of valid comparisons entailing an entity is, the higher is the likelihood that it is matching with one of the associated entities. The $ICF(p_i)$ for an entity p_i is computed by dividing the size of $\mathcal{C}_v$ by that of its subset $\mathcal{C}_v(p_i)$, which contains only comparisons involving entity p_i (i.e., $\mathcal{C}_v(p_i) = \{c_{i,k} \in \mathcal{C}_v\}$). More formally: $ICF(p_i) = \log |\mathcal{C}_v|/|\mathcal{C}_v(p_i)|$. The more comparisons entail p_i, the higher is the value of the denominator and the lower is the value of $ICF(p_i)$. Thus, the utility of comparison $c_{i,j}$ is proportional to both $ICF(p_i)$ and $ICF(p_j)$.

On the whole, the utility of a comparison $c_{i,j}$ is equal to[9]: $u(c_{i,j}) = ES(p_i, p_j) \cdot ICF(p_i) \cdot ICF(p_j)$.

To incorporate comparison scheduling in the ER process, we employ a *ranking function* $r : \mathcal{C}_v \mapsto \Re$ that defines a

9. Note that, in our experimental evaluation, we considered linear combinations of the three measures comprising comparisons utility, but they did not result in higher performance.

TABLE 1
Technical Characteristics of the Data Sets Used in the Experiments

	$\mathbf{D_{movies}}$		$\mathbf{D_{infoboxes}}$	
	DBPedia	*IMDB*	$DBPedia_1$	$DBPedia_2$
Entities	27,615	23,182	$1.19{\cdot}10^6$	$2.16{\cdot}10^6$
Name-Value Pairs	$1.86{\cdot}10^5$	$8.16{\cdot}10^5$	$1.75{\cdot}10^7$	$3.67{\cdot}10^7$
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 2
Execution Time for the Attribute Clustering Algorithms

		$\mathbf{D_{movies}}$ (minutes)	$\mathbf{D_{infoboxes}}$ (hours)
EM	Term Vector	1.49	116
	Trigrams	1.70	>200
AC	Term Vector	0.06	17
	Trigrams	0.09	66

partial order on $\mathcal{C}_v$, sorting its elements in descending order according to the following implication: $u(c_{i,j}) \leq u(c_{k,l}) \Rightarrow r(c_{i,j}) \geq r(c_{k,l})$. Therefore, its time complexity is equal to $O(|\mathcal{C}_v| \cdot log|\mathcal{C}_v|)$, while its space complexity is $O(|\mathcal{C}_v|)$.

6 Experimental Evaluation

The goal of our evaluation is threefold. First, to identify the best performing method of the effectiveness layer, by comparing token blocking with attribute clustering blocking (*AC Blocking* in the remainder of the paper) and a baseline clustering method (see Section 6.1). Second, to examine the behavior of the block purging algorithm in the context of clean-clean ER, by applying it on top of all block building approaches (see Section 6.2). Third, to compare the performance of three different efficiency workflows: two that were tested in the literature on top of token blocking and a new one that relies on comparison scheduling and operates exclusively on the level of comparisons. Our goal is to investigate the benefits of operating at the finest level of granularity, that of individual comparisons (see Section 6.3).

Measures. To evaluate the behavior of our approaches, we employ two kinds of measures: the *performance* and the *technical* ones. The former comprise the PC and RR of a blocking method, which capture its effectiveness and efficiency, respectively (see Section 3.1). The technical metrics encompass more practical measures that highlight internal aspects of a blocking method and affect its space and time complexity; these are: the total number of blocks it produces, the average number of comparisons per block, its BC_{ov} and CC values, as well as the *disk space* it occupies.

Note that we do not consider the performance of entity matching in terms of precision and recall. Entity matching is crucial for ER per se, but is orthogonal to the task of blocking for ER, which is the focus of our work. We follow the best practice in the related literature [3], [26], [29], [31], examining blocking methods independently of the profile matching techniques, by assuming the existence of an oracle that correctly decides whether two entity profiles are duplicates or not. Note that a highly performing blocking method with respect to PC and RR guarantees that the quality of a complete ER solution will be as good as the employed matching algorithm.

Data sets. In the course of our experimental study, we used two real-world, large-scale, heterogeneous data sets, which are presented in Table 1. They were also used in previous works [28], [29], [31], thus allowing for a direct comparison with prior results. The D_{movies} data set comprises a collection of movies from IMDB[10] and DBPedia,[11] which have been interlinked through the *"imdbid"* attribute in the profiles of DBPedia movies. $D_{infoboxes}$ is the largest data set, comprising more than 3 million entities that stem from two different versions of the DBPedia Infobox Data Set.[12] They have been collected by extracting all name-value pairs from the infoboxes of the articles in Wikipedia's english version. Theoretically, it may seem straightforward to resolve two versions of the same data set, but in practice it constitutes a quite challenging task; the older version ($DBPedia_1$) dates from October 2007, whereas the latest one ($DBPedia_2$) is a snapshot of October 2009. In the intervening two years, Wikipedia Infoboxes have evolved to such an extent that a mere 23.67 percent of all name-value pairs and 48.62 percent of the attribute names is common among both versions. To build the ground-truth, we considered as matching those entities that had exactly the same URL.

Baseline methods. As explained in Section 1, schema matching methods are not applicable to HHIS. Moreover, previous studies have demonstrated that schema-based methods exhibit high efficiency (i.e., very few entities per block), but suffer from remarkably poor PC in the context of HHIS (more than half of the matching entities do not share any common block) [29]. In this paper, we do not repeat the comparison experiments with such blocking methods. Instead, we use as baseline for our experiments the token blocking approach, which was verified to outperform schema-based techniques when applied to HHIS [29].

To evaluate the performance of our attribute clustering algorithm, we compare it with an established clustering technique that can offer the same functionality. In principle, any clustering algorithm is applicable, on the sole condition that it involves an *unconstrained* functionality (i.e., it does not require as input the number of returned clusters). For this reason, we have selected as our baseline method a variation of the expectation maximization (*EM*) algorithm [8], which specifies the number of clusters through an unsupervised procedure that relies on cross-validation.[13] EM can be combined with the term vector and the character n-grams model, and these combinations are denoted by *term vector EM* and *trigrams EM*, respectively, in the following. However, it is incompatible with the n-gram graphs, since this representation model is only suitable for pairwise comparisons (i.e., it does not produce features in a vector format).

Experimental setup. All approaches and experiments were fully implemented in Java, version 1.6. For the implementation of the blocking functionality (i.e., inverted

10. http://www.imdb.com.
11. http://dbpedia.org.
12. http://wiki.dbpedia.org/Datasets.
13. For more details, see http://weka.sourceforge.net/doc/weka/clusterers/EM.html.

TABLE 3
Statistics and Performance of the Blocking Building Methods on D_{movies}

			Technical Metrics				Performance Metrics			
	Attr. Clusters	Blocks	Av. Comp.	BC_{ov}	CC	Disk Space	Comparisons	RR	Duplicates	PC
Token Blocking	1	40,825	$7.46 \cdot 10^3$	34.30	$0.57 \cdot 10^{-2}$	28MB	$3.05 \cdot 10^8$	-	22,387	99.92%
Term Vector EM	4	33,777	$8.32 \cdot 10^3$	32.85	$0.59 \cdot 10^{-2}$	52MB	$2.81 \cdot 10^8$	7.82%	21,944	97.94%
Trigrams EM	2	18,707	$2.49 \cdot 10^3$	10.86	$1.18 \cdot 10^{-2}$	52MB	$0.47 \cdot 10^8$	84.73%	17,150	76.55%
Term Vector AC	3	43,270	$6.69 \cdot 10^3$	33.16	$0.58 \cdot 10^{-2}$	52MB	$2.90 \cdot 10^8$	4.94%	22,360	99.80%
Trigrams AC	3	43,271	$6.71 \cdot 10^3$	34.08	$0.59 \cdot 10^{-2}$	52MB	$2.91 \cdot 10^8$	4.67%	22,365	99.82%
Trigram Graphs AC	4	44,158	$4.83 \cdot 10^3$	32.96	$0.78 \cdot 10^{-3}$	52MB	$2.13 \cdot 10^8$	30.06%	22,304	99.55%

RR values were computed based on token blocking.

TABLE 4
Statistics and Performance of the Blocking Building Methods on $D_{infoboxes}$

			Technical Metrics				Performance Metrics			
	Attr. Clusters	Blocks	Av. Comp.	BC_{ov}	CC	Disk Space	Comparisons	RR	Duplicates	PC
Token Blocking	1	$1.21 \cdot 10^6$	$5.10 \cdot 10^6$	29.51	$0.16 \cdot 10^{-4}$	2.1GB	$6.18 \cdot 10^{12}$	-	892,560	99.997%
Term Vector EM	2	$1.35 \cdot 10^6$	$4.74 \cdot 10^6$	31.86	$0.17 \cdot 10^{-4}$	4.9GB	$6.38 \cdot 10^{12}$	-	892,546	99.995%
Term Vector AC	3,717	$1.22 \cdot 10^6$	$5.05 \cdot 10^6$	29.42	$0.16 \cdot 10^{-4}$	4.4GB	$6.18 \cdot 10^{12}$	0.01%	892,560	99.997%
Trigrams AC	24,927	$4.48 \cdot 10^6$	$0.23 \cdot 10^6$	41.76	$1.34 \cdot 10^{-4}$	5.0GB	$1.05 \cdot 10^{12}$	83.04%	892,425	99.982%
Trigram Graphs AC	26,762	$4.80 \cdot 10^6$	$0.21 \cdot 10^6$	43.22	$1.41 \cdot 10^{-4}$	5.0GB	$1.03 \cdot 10^{12}$	83.39%	892,516	99.992%

RR values were computed based on token blocking.

indices), we used the open source library of Lucene,[14] version 2.9. The functionality of the n-gram graphs was provided by the open source library of JInsect.[15] For the implementation of the unconstrained EM clustering algorithm, we employed the open source library of Weka,[16] version 3.6. All experiments were performed on a desktop machine with Intel i7, 16 GB of RAM memory, running Linux (kernel version 2.6.38).

6.1 Block Building

We start the evaluation by comparing token blocking with the three variations of AC Blocking: 1) the combination of the term vector model with the cosine similarity, symbolized by *term vector AC*, 2) the combination of character trigrams with Jaccard similarity, denoted by *trigrams AC*, and 3) the combination of trigram graphs with the value similarity metric, which is represented as *trigram graphs AC*. We also compare it with term vector EM and trigrams EM.

Before analyzing the performance of the blocks they create, it is worth probing into the applicability of all clustering algorithms with respect to their time complexity. We actually recorded the execution time of EM and AC blocking across both data sets and in combination with the term vector and the trigrams representation models. The outcomes are presented in Table 2. We can notice that AC is substantially faster than EM, requiring around 1/20 and 1/6 of its running time in the case of D_{movies} and $D_{infoboxes}$, respectively. In fact, EM in conjunction with trigrams was not able to process $D_{infoboxes}$ within a time frame of 200 hours. Thus, we consider this particular combination to be inapplicable to large-scale HHIS and do not report its performance in Tables 4 and 6. In the following, we examine the performance of the blocks created by the other EM-based methods to find out whether their quality is worth the high computational cost.

Table 3 presents the performance of all methods on the D_{movies} data set. We can see that all variations of the clustering algorithms produce a limited number of attribute clusters, since D_{movies} contains only 11 distinct attributes (see Table 2). As a result, there are minor differences in the behavior of the blocking methods (e.g., they all occupy the same disk space). Nevertheless, we can identify the following pattern: the higher the number of clusters is, the more blocks are produced and the less comparisons they entail, on average. This results in higher efficiency and moves the BC-CC mapping of the blocking methods closer to the ideal point: their BC_{ov} decreases, while their CC increases. This effect has a direct impact on their actual performance, reducing PC by less than 2 percent and decreasing comparisons to a considerable extent. The only exception to this pattern is trigrams EM, which involves the least number of comparisons, but fails to place in a common block almost one out of four pairs of duplicates. Thus, it constitutes the only clustering approach with inferior performance to token blocking. All others offer a better balance between PC and RR, with trigram graphs AC exhibiting the best tradeoff.

Table 4 offers stronger evidence for the differences in the performance of the individual blocking methods. The reason is that the high number of attribute names of $D_{infoboxes}$ allows for higher variation in the attribute clusters. It is noteworthy, though, that the performance pattern of D_{movies} applies in this data set, as well: the higher the number of attribute clusters is, the higher is the resulting number of blocks and the less comparisons they entail, on average. This effect leads to substantially higher CC values (even by an order of magnitude) and, thus, higher RR values, while PC remains practically stable. Unlike D_{movies}, however, the increase in the number of attribute clusters results in substantial increase in the values of BC_{ov} and the space occupied on the disk, due to the significantly higher number of blocks. It is also worth noting that all variations of AC blocking provide a better tradeoff between PC and RR than token blocking, while term vector EM exhibits the worst performance: it involves more comparisons than all other methods for practically identical PC with them.

On the whole, we can argue that AC Blocking substantially improves on token blocking, offering higher efficiency for the same levels of effectiveness. It also

14. http://lucene.apache.org/.
15. http://sourceforge.net/projects/jinsect.
16. http://www.cs.waikato.ac.nz/ml/weka/.

TABLE 5
Block Purging on D_{movies}

		Technical Metrics			Performance Metrics			
	Purged Blocks	Av. Comparisons	BC_{ov}	CC	Comparisons	RR	Duplicates	PC
Token Blocking	42	$2.73 \cdot 10^3$	30.23	$1.38 \cdot 10^{-2}$	$1.11 \cdot 10^8$	63.50%	22,384	99.91%
Term Vector EM	38	$3.25 \cdot 10^3$	29.09	$1.34 \cdot 10^{-2}$	$1.10 \cdot 10^8$	60.93%	21,879	97.65%
Trigrams EM	35	$0.48 \cdot 10^3$	8.11	$4.61 \cdot 10^{-2}$	$0.09 \cdot 10^8$	80.93%	16,939	75.60%
Term Vector AC	76	$1.86 \cdot 10^3$	27.72	$1.75 \cdot 10^{-2}$	$0.81 \cdot 10^8$	72.22%	22,356	99.78%
Trigrams AC	74	$1.88 \cdot 10^3$	28.65	$1.79 \cdot 10^{-2}$	$0.81 \cdot 10^8$	72.01%	22,361	99.80%
Trigram Graphs AC	52	$1.66 \cdot 10^3$	29.12	$2.02 \cdot 10^{-2}$	$0.73 \cdot 10^8$	65.73%	22,301	99.54%

RR values were computed based on the original performance of each method in Table 3.

TABLE 6
Block Purging on $D_{infoboxes}$

		Technical Metrics			Performance Metrics			
	Purged Blocks	Av. Comparisons	BC_{ov}	CC	Comparisons	RR	Duplicates	PC
Token Blocking	396	$4.70 \cdot 10^4$	16.24	$0.96 \cdot 10^{-3}$	$5.68 \cdot 10^{10}$	99.08%	891,767	99.91%
Term Vector EM	564	$3.23 \cdot 10^4$	17.24	$1.32 \cdot 10^{-3}$	$4.34 \cdot 10^{10}$	99.32%	891,709	99.90%
Term Vector AC	396	$4.65 \cdot 10^4$	16.24	$0.96 \cdot 10^{-3}$	$5.68 \cdot 10^{10}$	99.08%	891,767	99.91%
Trigrams AC	1,064	$0.68 \cdot 10^4$	27.50	$3.02 \cdot 10^{-3}$	$3.06 \cdot 10^{10}$	97.08%	892,402	99.98%
Trigram Graphs AC	1,358	$0.50 \cdot 10^4$	28.12	$3.90 \cdot 10^{-3}$	$2.42 \cdot 10^{10}$	97.64%	892,463	99.99%

RR values were computed based on the original performance of each method in Table 4.

outperforms EM-based blocking methods in many aspects: it is applicable to large entity collections, it can be combined with the n-gram graphs representation model, and it produces blocks of higher quality. The last aspect is probably caused by the "blind" functionality of EM: unlike our attribute clustering algorithm, EM does not guarantee that every cluster contains attribute names from both input entity collections. Instead, it is possible that clusters exclusively contain attributes stemming from the same source, thus rendering their values useless for blocking. Regarding the relative performance of the three established representation models, the n-gram graphs clearly exhibit the best performance across both data sets. This is because their noise-tolerant and language-agnostic functionality turns them more suitable than the other models for tackling the intricacies of HHIS.

6.2 Block Purging

This section examines the effect of our block purging algorithm on all blocking methods across both data sets. Its performance for D_{movies} and for $D_{infoboxes}$ is presented in Tables 5 and 6, respectively.

We notice that BC_{ov} decreases for all methods across both data sets, thus getting closer to the $x = 1$ axis. On the other hand, CC increases to a great extent, getting closer to its maximum value (i.e., $CC_{max} = 2$). All approaches move, therefore, closer to the ideal point, improving their balance between effectiveness and efficiency across both data sets: although PC decreases by less than 1 percent in all cases, the overall number of comparisons is reduced by 68 percent in D_{movies} and by 98 percent (i.e., two orders of magnitude) in $D_{infoboxes}$, on average. This behavior means that block purging accurately detects the oversized blocks, performing a conservative, but valuable cleansing.

Note that, in each data set, block purging removes almost the same portion of blocks from all approaches: in D_{movies} it discards between 0.10 and 0.17 percent of all blocks and in $D_{infoboxes}$ around 0.03 percent of them. Given that it triggers similar quantitative effects on the technical and the performance metrics of all methods, we can conclude that they all involve similar power-law distributions of comparisons: few blocks are oversized, containing the largest part of the comparisons, while their vast majority entails a handful of entities.

The outcomes of Tables 5 and 6 clearly indicate that trigrams AC maintains the best balance between PC and RR even after block purging. It has the smaller—on average—blocks, thus requiring by far the lowest total number of pairwise comparisons. In addition, its PC remains well above 99 percent in all cases, exhibiting the highest value across all approaches for $D_{infoboxes}$. For this reason, we employ trigrams AC as the block building method that lies at the core of all efficiency workflows we analyze in the following.

6.3 Efficiency Workflows

We now analyze the performance of three different efficiency workflows, which share the same *core workflow*: they are all based on trigram graphs AC for the creation of blocks and on block purging and duplicate propagation for their processing. They differ, though, in the other methods they involve: the first one, WF_1, adds exclusively block-refinement methods to the core workflow, namely, block scheduling, and block pruning [29]. The second one, WF_2, combines block-refinement methods with comparison-refinement ones, namely, block scheduling with comparison propagation and comparison pruning [31]. The third workflow, WF_3, is the only one that employs comparison scheduling, operating exclusively on the level of individual comparisons; it additionally involves comparison propagation and comparison pruning.[17]

We selected these workflows for a number of reasons. WF_1 and WF_2 have already been examined in [29] and [31], respectively, over token blocking; given that we employ the same data sets, our results are directly comparable with

17. In all cases, the ES_{min} threshold for comparison pruning was specified by setting $a = 0.20$ in (1), which is a conservative value lying very close to $a = 0.25$ that induces a minor reduction in PC, while boosting RR [31]. The slightly lower value of a is justified by the substantially higher number of blocks produced by attribute clustering techniques.

TABLE 7
Perfomance of Three Different Workflows on the D_{movies} and $D_{infoboxes}$ on Top of Block Purging and Block Building with Trigram Graphs AC

	Method	Compar.	RR	Duplicates	PC	Time
	Block Purging	$7.30{\cdot}10^7$	-	22,301	99.54%	0.14
WF_1	Block Scheduling	$3.83{\cdot}10^5$	99.48%	22,301	99.54%	0.05
WF_1	Block Pruning	$2.67{\cdot}10^5$	99.64%	22,295	99.51%	0.05
	Comp. Propagation	$6.10{\cdot}10^7$	16.65%	22,301	99.54%	0.67
WF_2	Block Scheduling	$3.15{\cdot}10^5$	99.57%	22,301	99.54%	0.05
WF_2	Comp. Pruning	$9.73{\cdot}10^4$	99.88%	21,449	95.73%	0.51
	Comp. Propagation	$6.10{\cdot}10^7$	16.65%	22,301	99.54%	0.67
WF_3	Comp. Pruning	$2.42{\cdot}10^6$	96.69%	21,449	95.73%	0.51
WF_3	Comp. Scheduling	$8.71{\cdot}10^4$	99.88%	21,449	95.73%	0.06

(a)

	Method	Compar.	RR	Duplicates	PC	Time
	Block Purging	$2.42{\cdot}10^{10}$	-	892,463	99.99%	0.05
WF_1	Block Scheduling	$1.55{\cdot}10^9$	93.58%	892,463	99.99%	0.16
WF_1	Block Pruning	$7.24{\cdot}10^7$	99.70%	879,446	98.53%	0.01
	Comp. Propagation	$1.24{\cdot}10^{10}$	48.86%	892,463	99.99%	5.75
WF_2	Block Scheduling	$9.20{\cdot}10^8$	96.20%	892,463	99.99%	0.16
WF_2	Comp. Pruning	$4.98{\cdot}10^7$	99.79%	837,286	93.80%	4.14
	Comp. Propagation	$1.24{\cdot}10^{10}$	48.86%	892,463	99.99%	5.75
WF_3	Comp. Pruning	$4.32{\cdot}10^8$	98.21%	837,286	93.80%	4.14
WF_3	Comp. Scheduling	$4.46{\cdot}10^7$	99.82%	837,286	93.80%	0.51

(b)

Baseline method for computing RR is block purging. The required time is measured in minutes for D_{movies} and in hours for $D_{infoboxes}$.

prior work. WF_3 is a novel workflow, but it is based on WF_2, modifying it so that it is compatible with comparison scheduling. Collectively, these workflows cover all efficiency methods presented in Section 5.2. They also differ significantly in the complexity of their functionality: WF_1 conveys minimum space and time requirements, whereas WF_3 involves the most complex methods with respect to both aspects. WF_2, on the other hand, lies in the middle of these two extremes. Last but not least, all workflows were formed according to the guidelines of Section 4.1.

The performance of all workflows over D_{movies} and $D_{infoboxes}$ is presented in Tables 7a and 7b, respectively, with the individual methods of each workflow appearing in the order they are executed. We can notice that methods targeting the repeated and superfluous comparisons (i.e., block scheduling, comparison propagation, and comparison scheduling) have no effect on PC, although they significantly enhance RR. It is worth clarifying at this point that the performance of the two scheduling methods is actually derived from their combination with duplicate propagation; it denotes, therefore, how many comparisons are saved just by ordering the block's or comparisons' execution and propagating the detected duplicates. This explains why block scheduling appears below comparison propagation in WF_2.

It is interesting to compare the performance of the only methods (apart from block purging) that affect PC: block and comparison pruning. This is done by contrasting the performance of WF_1 and WF_2. We can identify the following pattern across both data sets: block pruning has a negligible effect on PC, reducing it by less than 1.5 percent, whereas comparison pruning has a considerable impact on it, conveying a decrease of 5 percent. Both have RR values over 99 percent, but comparison pruning actually involves around 50 percent less comparisons than block pruning. Thus, the former discards more comparisons than the latter, sacrificing PC to a larger extent in favor of higher efficiency (i.e., RR). The main advantage of comparison pruning, though, is that it can be seamlessly combined with comparison scheduling (WF_3), which further reduces comparisons by around 10 percent, at no cost in PC.

Regarding the execution time of the workflows, we can notice the following patterns: WF_2 and WF_3 share almost the same time requirements across both data sets, with the latter taking slightly longer to complete its processing. On the other hand, WF_1 is around 100 times faster, due to its coarse granularity of functionality. Even in the worst case for $D_{infoboxes}$, though, WF_3 requires less than 12 hours for processing more than 3 millions of entities. Among the individual blocking methods, comparison propagation and comparison pruning involve the most time-consuming processing. Compared to them, all other techniques require at least 10 times less time.

On the whole, both data sets advocate that WF_3 requires the lowest number of comparisons per entity, followed by WF_2 and WF_1. Its substantially higher efficiency, though, comes at the cost of slightly lower effectiveness, as it detects around 4 percent less duplicates than WF_1. It also consumes more resources, due to comparison scheduling, and involves the highest execution time. For small data sets—with millions of comparisons—its computational cost is affordable, and typically WF_3 constitutes the best option. However, for large-scale applications—with billions of comparisons—the choice depends on the performance requirements and the available resources of the application at hand. In contrast, WF_1 is suitable for applications that have limited access to resources or are very strict with respect to effectiveness. Given that it involves the fastest processing, it is also suitable for applications with small entity profiles that can be efficiently compared; in this case, it can compensate for the higher number of comparisons it involves in comparison to WF_2 and WF_3. Finally, WF_2 lies in the middle of these two extremes, offering the same effectiveness as WF_3 at slightly lower blocking efficiency and time complexity.

7 Conclusions

We presented a generic and extensible framework for blocking-based clean-clean ER over HHIS, composed of the effectiveness and efficiency layers. We elaborated on the characteristics of each layer, showed how existing methods map to them, proposed novel techniques in this context, and discussed how to combine these methods into comprehensive ER workflows. We conducted a thorough experimental evaluation with 3.3 million entities, demonstrating the efficiency of the proposed approach, which requires just 13 pairwise comparisons per entity for a pair completeness of 93.80 percent. In the future, we plan to extend our framework with techniques that deal with dirty ER as well as incremental ER, and to explore ways of

parallelizing our approach on the basis of the MapReduce paradigm. We also intend to investigate ways of incorporating mediated schemas in the process of attribute clustering with the aim of yielding blocks of higher quality.

REFERENCES

[1] R. Almeida, B. Mozafari, and J. Cho, "On the Evolution of Wikipedia," *Proc. Int'l AAAI Conf. Weblogs and Social Media (ICWSM)*, 2007.

[2] R. Baxter, P. Christen, and T. Churches, "A Comparison of Fast Blocking Methods for Record Linkage," *Proc. Workshop Data Cleaning, Record Linkage and Object Consolidation at SIGKDD*, pp. 25-27, 2003.

[3] M. Bilenko, B. Kamath, and R.J. Mooney, "Adaptive Blocking: Learning to Scale Up Record Linkage," *Proc. Sixth Int'l Conf. Data Mining (ICDM)*, pp. 87-96, 2006.

[4] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data-The Story So Far," *Int'l J. Semantic Web Information Systems*, vol. 5, no. 3, pp. 1-22, 2009.

[5] P. Christen, "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication," *IEEE Trans. Knowledge and Data Eng.*, vol. 24, no. 9, pp. 1537-1555, Sept. 2012.

[6] W.W. Cohen, P.D. Ravikumar, and S.E. Fienberg, "A Comparison of String Distance Metrics for Name-Matching Tasks," *Proc. Workshop Information Integration Web (IIWeb)*, pp. 73-78, 2003.

[7] T. de Vries, H. Ke, S. Chawla, and P. Christen, "Robust Record Linkage Blocking Using Suffix Arrays," *Proc. 18th ACM Conf. Information and Knowledge Management (CIKM)*, pp. 305-314, 2009.

[8] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc.*, vol. 39, pp. 1-38, 1977.

[9] A. Doan and A. Halevy, "Semantic Integration Research in the Database Community: A Brief Survey," *AI Magazine*, vol. 26, no. 1, pp. 83-94, 2005.

[10] X. Dong, A. Halevy, and J. Madhavan, "Reference Reconciliation in Complex Information Spaces," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 85-96, 2005.

[11] A. Elmagarmid, P. Ipeirotis, and V. Verykios, "Duplicate Record Detection: A Survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 1, pp. 1-16, Jan. 2007.

[12] L. Getoor and C. Diehl, "Link Mining: A Survey," *ACM SIGKDD Explorations Newsletters*, vol. 7, pp. 3-12, 2005.

[13] G. Giannakopoulos, V. Karkaletsis, G. Vouros, and P. Stamatopoulos, "Summarization System Evaluation Revisited: N-Gram Graphs," *ACM Trans. Speech and Language Processing*, vol. 5, pp. 1-39, 2008.

[14] G. Giannakopoulos and T. Palpanas, "Content and Type as Orthogonal Modeling Features: A Study on User Interest Awareness in Entity Subscription Services," *Int'l J. Advances on Networks and Services*, vol. 3, no. 2, pp. 296-309, 2010.

[15] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate String Joins in a Database (Almost) for Free," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB)*, pp. 491-500, 2001.

[16] S. Guo, X. Dong, D. Srivastava, and R. Zajac, "Record Linkage with Uniqueness Constraints and Erroneous Values," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 417-428, 2010.

[17] A.Y. Halevy, M.J. Franklin, and D. Maier, "Principles of Dataspace Systems," *Proc. 25th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS)*, pp. 1-9, 2006.

[18] M. Hernández and S. Stolfo, "The Merge/Purge Problem for Large Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 127-138, 1995.

[19] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis, "On-the-Fly Entity-Aware Query Processing in the Presence of Linkage," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 429-438, 2010.

[20] L. Jin, C. Li, and S. Mehrotra, "Efficient Record Linkage in Large Data Sets," *Proc. Eighth Int'l Conf. Database Systems for Advanced Applications (DASFAA)*, 2003.

[21] H. Kim and D. Lee, "HARRA: Fast Iterative Hashed Record Linkage for Large-Scale Data Collections," *Proc. 13th Int'l Conf. Extending Database Technology (EDBT)*, pp. 525-536, 2010.

[22] N. Koudas, S. Sarawagi, and D. Srivastava, "Record Linkage: Similarity Measures and Algorithms," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 802-803, 2006.

[23] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu, "Web-Scale Data Integration: You Can Afford to Pay as You Go," *Proc. Conf. Innovative Data Systems Research (CIDR)*, pp. 342-350, 2007.

[24] C. Manning, P. Raghavan, and H. Schuetze, *Introduction to Information Retrieval*. Cambridge Univ. Press, 2008.

[25] A. McCallum, K. Nigam, and L. Ungar, "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 169-178, 2000.

[26] M. Michelson and C.A. Knoblock, "Learning Blocking Schemes for Record Linkage," *Proc. 21st Nat'l Conf. Artificial Intelligence (AAAI)*, pp. 440-445, 2006.

[27] A. Ouksel and A. Sheth, "Semantic Interoperability in Global Information Systems: A Brief Introduction to the Research Area and the Special Section," *SIGMOD Record*, vol. 28, no. 1, pp. 5-12, 1999.

[28] G. Papadakis, G. Giannakopoulos, C. Niederée, T. Palpanas, and W. Nejdl, "Detecting and Exploiting Stability in Evolving Heterogeneous Information Spaces," *Proc. ACM/IEEE 11th Ann. Int'l Joint Conf. Digital Libraries (JCDL)*, pp. 95-104, 2011.

[29] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser, "Efficient Entity Resolution for Large Heterogeneous Information Spaces," *Proc. Fourth ACM Int'l Conf. Web Search and Data Mining (WSDM)*, pp. 535-544, 2011.

[30] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl, "Eliminating the Redundancy in Blocking-Based Entity Resolution Methods," *Proc. 11th Ann. ACM/IEEE Int'l Joint Conf. Digital Libraries (JCDL)*, pp. 85-94, 2011.

[31] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl, "To Compare or Not to Compare: Making Entity Resolution More Efficient," *Proc. Int'l Workshop Semantic Web Information Management (SWIM)*, 2011.

[32] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl, "Beyond 100 Million Entities: Large-Scale Blocking-Based Resolution for Heterogeneous Data," *Proc. Fifth ACM Int'l Conf. Web Search and Data Mining (WSDM)*, pp. 53-62, 2012.

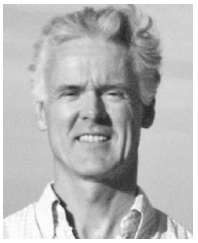
[33] S. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina, "Entity Resolution with Iterative Blocking," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 219-232, 2009.

George Papadakis received the diploma degree in computer engineering from the National Technical University of Athens (NTUA) and has worked at the NCSR "Demokritos," NTUA and the L3S Research Center. He is working toward the PhD degree at the Leibniz University of Hanover, Germany. His research interests include entity resolution and Web data mining. He has received the Douglas Engelbart Best Paper Award from ACM Hypertext 2011.

Ekaterini Ioannou received the BSc and MSc degrees from the University of Cyprus, the MSc degree from Saarland University, and the PhD degree in computer science from the University of Hannover, Germany. She is currently working as a research collaborator at the SoftNet Lab, at the Technical University of Crete. Her research interests include areas of information integration, management of uncertain data, and resolution methodologies for heterogeneous and large size collections.

Themis Palpanas is a professor of computer science at the University of Trento, Italy. Before that, he worked at the IBM T.J. Watson Research Center, and has also worked for the University of California at Riverside, Microsoft Research, and IBM Almaden Research Center. He is the author of five US patents, three of which are part of commercial products. He has received three Best Paper awards (PERCOM 2012, ICDE 2010, and ADAPTIVE 2009), and is general chair for VLDB 2013.

Claudia Niederée received the MSc and PhD degrees in computer science from the Technical University Hamburg-Harburg, Germany. She works as a senior researcher at the L3S Research Center in Hannover. Before she joined L3S, she worked at the Fraunhofer IPSI in Darmstadt, Germany. Her main research interests include technologies related to digital libraries, personalization support, digital libraries, e-science, and semantic Web.



Wolfgang Nejdl received the MSc and PhD degrees from the Technical University of Vienna. Currently, he is a professor of computer science at the University of Hannover, Germany, where he leads the L3S Research Center and the Distributed Systems Institute/Knowledge-Based Systems. His research interest include the areas of search and information retrieval, peer-to-peer, databases, technology-enhanced learning, and artificial intelligence.