Schema-agnostic vs Schema-based Configurations for Blocking Methods on Homogeneous Data

George Papadakis[$], George Alexiou[#], George Papastefanatos[#], Georgia Koutrika[◇]

[◇] HP Labs, USA koutrika@hp.com
[#]IMIS, Research Center "Athena", Greece {galexiou,gpapas}@imis.athena-innovation.gr
[$]Dep. of Informatics & Telecommunications, University of Athens, Greece gpapadis@di.uoa.gr

ABSTRACT

Entity Resolution constitutes a core task for data integration that, due to its quadratic complexity, typically scales to large datasets through blocking methods. These can be configured in two ways. The schema-based configuration relies on schema information in order to select signatures of high distinctiveness and low noise, while the schema-agnostic one treats every token from all attribute values as a signature. The latter approach has significant potential, as it requires no fine-tuning by human experts and it applies to heterogeneous data. Yet, there is no systematic study on its relative performance with respect to the schema-based configuration. This work covers this gap by comparing analytically the two configurations in terms of effectiveness, time efficiency and scalability. We apply them to 9 established blocking methods and to 11 benchmarks of structured data. We provide valuable insights into the internal functionality of the blocking methods with the help of a novel taxonomy. Our studies reveal that the schema-agnostic configuration offers unsupervised and robust definition of blocking keys under versatile settings, trading a higher computational cost for a consistently higher recall than the schema-based one. It also enables the use of state-of-the-art blocking methods without schema knowledge.

1. INTRODUCTION

Entity Resolution (ER) is an essential task for data cleaning and integration, which aims to identify all entity profiles that correspond to the same real-world entity. In principle, it compares every profile with all others, yielding a quadratic complexity that does not scale well to voluminous data collections. A bulk of relevant research has focused on improving its efficiency [6, 7, 11]. The most popular approach in this direction is *blocking* [3, 7, 26]. Unlike the exhaustive ER techniques, it offers an approximate solution, sacrificing some recall in order to enhance precision. In essence, it clusters similar entity profiles into blocks so that it suffices to perform pair-wise comparisons inside each block. The clustering is based on signatures for higher efficiency: from each entity profile, blocking methods extract one or more *blocking keys* that summarize a subset of its attribute values. Subsequently, profiles with identical or similar blocking keys are placed into the same block.

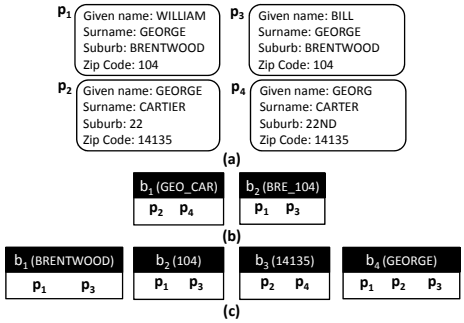

Figure 1: (a) A set of homogeneous entity profiles, (b) the blocks created by a schema-based configuration, (c) the blocks created by a schema-agnostic configuration.

There are two main approaches for defining blocking keys. The *schema-based configuration* exploits a-priori schema information to come up with blocking keys that summarize the content of the attributes with the most distinctive values and/or the lowest levels of noise [6, 7]. It creates blocks that involve few comparisons, while placing most of the duplicates in at least one common block.

To illustrate this functionality, consider the entities in Figure 1(a), which are drawn from census data. Observe that p_1 and p_2 are matching with p_3 and p_4, respectively. A possible schema-based configuration is to represent every entity by a key that concatenates the first 3 characters from the values of "Given name" and "Surname". In this way, p_1 is represented by *WIL_GEO*, p_2 by *GEO_CAR*, p_3 by *BIL_GEO* and p_4 by *GEO_CAR*. The only key that appears in at least 2 profiles is *GEO_CAR*, which forms the block $b_{GEO\_CAR}=\{p_2, p_4\}$. With a single comparison, b_{GEO_CAR} identifies one of the 2 pairs of duplicates (recall=0.5). To detect the second pair and raise recall to 1.0, we consider an additional key for each entity that appends the entire value of "Zip Code" to the first 3 characters from the value of "Suburb". Indeed, we get the block $b_{BRE\_104}=\{p_1, p_3\}$, which contains the missing duplicates. Figure 1(b) presents the blocks resulting from the 2 schema-based keys, which cover both pairs of duplicates with just 2 comparisons.

The second approach for defining blocking keys does not take any schema information into account, using as blocking keys all tokens from all attribute values [26]. Thus, we call it *schema-agnostic configuration*. As an example, consider the blocks in Figure 1(c); similar to the blocks in Figure 1(b), they place both pairs of duplicate entities, $p_2 \equiv p_4$ and $p_1 \equiv p_3$, in at least one common block, but require 6 comparisons, in total.

The schema-based configuration is typically applied to blocking methods like Sorted Neighborhood and Suffix Arrays that are

Proceedings of the VLDB Endowment, Vol. 9, No. 4

crafted for *homogeneous data collections* [6, 7, 10]. These collections pertain to structured data, which abide by a specific schema with known semantics and qualitative characteristics for each attribute. In contrast, the schema-agnostic configuration is typically combined with blocking methods like Token Blocking crafted for *heterogeneous data collections* [26, 27]. These collections contain semi-structured or unstructured data with noise in their attribute values and their attribute names. In this category fall web data, which comprise user-generated data posted on-line without any curation. The result is a large diversity of schemata that lack semantics; Google Base (`http://www.google.com/base`) alone has 100,000 distinct schemata corresponding 10,000 entity types [21].

Comparing the two configurations, we make several observations. First, the schema-based one cannot handle heterogeneous data collections, due to the lack of reliable qualitative meta-data about the schema(ta) they contain. This issue could be ameliorated through schema matching, but the relevant techniques do not scale well to the unprecedented schema heterogeneity of web data [29]. In contrast, the schema-agnostic configuration could be used for homogeneous data, but its performance has not been tested yet.

Second, the schema-agnostic configuration is quite straightforward, hence its outcome is predetermined. On the other hand, the performance of schema-based configuration is very sensitive to the definition of the blocking keys, where minor modifications can result in major differences in the quality of the produced blocks [6, 7]. Our schema-based example in Figure 1(b) already demonstrated that an additional key definition raises recall from 0.5 to 1.0. Further, if the two schema-based keys were extracted from the first 5 characters of each value, only the block $b_{BRENT\_104} = \{p_1, p_3\}$ would be generated, thus missing one out of the two pairs of duplicates; on the other hand if they used the first 2 characters from every value, there would be a much larger number of blocks and comparisons. Overall, the schema-based configuration needs fine-tuning, while the schema-agnostic one essentially has one possible output.

Third, the fine-tuning of schema-based blocking keys can be performed in two ways: either manually, by exploiting human expertise, or automatically, through labeled data that are fed to a classification algorithm. The latter approach comprises methods that learn to extract optimal blocking keys from combinations of attribute names [4, 23]. These supervised methods, though, are of limited utility, because they require different training sets for every domain, which are rarely available. This means that the schema-based blocking keys have to be manually configured in most of the cases, a requirement that significantly restricts their utility [7].

The above observations naturally lead to the following question: *What would happen if we replaced the schema-based configuration of the blocking methods for homogeneous data with the schema-agnostic one typically used for heterogeneous data*? This can be accomplished by modifying their functionality such that they treat every single token in the attribute values as a separate blocking key. Could the inherent simplicity of the schema-agnostic configuration affect positively the performance of the blocking methods? In which cases is this impact good enough? Or is it the case that the schema-based configuration can always give significant gains that overcompensate for its complexity? Are there combinations of blocking methods and configurations that work best? How does the size and characteristics of the dataset to which we apply entity resolution weigh in the decision on which setting should be chosen?

In this work, we aim to answer these questions by analytically examining the relative performance of the schema-agnostic and the schema-based configurations in combination with 9 state-of-the-art blocking methods for Entity Resolution in structured data. For each method, we compare its schema-agnostic configuration with 3 different schema-based ones that have been used in [7]. We consider various configurations for the other parameters of each method and investigate how these affect its effectiveness and time efficiency. We employ 4 real-world homogeneous datasets of varying sizes and characteristics. We also perform a thorough scalability analysis using 7 synthetic homogeneous datasets of increasing size, ranging from 10,000 entities with $\sim 10^7$ comparisons to 2 million entities with $\sim 10^{12}$ comparisons. To the best of our knowledge, no prior study has elaborated on the impact of the schema-agnostic configuration on the performance of blocking methods for structured data.

To facilitate the interpretation of the experimental results, we define a novel two-dimensional taxonomy for the blocking methods that helps identify the most important parameters that affect the effectiveness and the time efficiency for each configuration of blocking keys. Our studies demonstrate that the schema-agnostic configuration can address the main drawbacks of the schema-based one, offering unsupervised and robust definition of blocking keys under versatile settings. Its keys reduce precision and increase the overhead time to a significant extent, but achieve consistently higher recall, which is of paramount importance in the context of blocking-based Entity Resolution. These results have two important practical implications. First, they show that we can facilitate the use of state-of-the-art blocking methods for structured data without the need to know the data schema; while overhead and resolution times may increase, the methods remain scalable. Second, these blocking methods are now applicable to heterogeneous web data.

In more detail, we make the following contributions:

- We perform an extensive experimental study to compare the schema-agnostic with the schema-based configuration of 9 state-of-the-art blocking methods for homogeneous data with respect to effectiveness, in terms of their blocks quality, and time efficiency.
- We analytically examine the scalability of both configurations in combination with the 9 blocking methods over synthetic datasets of increasing size that range from 10,000 to 2,000,000 entities.
- We coin a two-dimensional taxonomy of the blocking methods, highlighting the main parameters that determine their performance in combination with either of the blocking key configurations.
- We have organized the implementation of all blocking methods and their configurations as an extensible framework, whose code has been publicly released [1]. We have also published the datasets used in our study. In this way, we encourage and facilitate other practitioners and researchers in the field to extend our work.

The rest of the paper is structured as follows: Section 2 presents the main notions of blocking-based ER and the measures that evaluate its performance. Section 3 introduces the blocking methods of our study and categorizes them in our taxonomy. Section 4 describes our experimental setup and Section 5 analyzes the outcomes of our study. Related work is discussed in Section 6 and a summary of our findings and directions for future work is given in Section 7.

2. BLOCKING-BASED ER

At the core of ER lies the notion of the *entity profile* p, alternatively called entity or profile in the following. As such, we consider a set of name-value pairs that is associated with a unique id and describes a real-world object. A set of entity profiles is called *entity collection* E. Given such a collection, the goal of ER is to partition E into a set of equivalence clusters, $D(E)$, such that every cluster contains all entity profiles that pertain to the same real-world object. Two such entities, p_i and p_j, are said to be *matching* or *duplicates* and are symbolized by $p_i \equiv p_j$. This task is also called *Deduplication* [6, 7] in the context of homogeneous data collections and involves a quadratic time complexity, as every entity has to be compared with all the possibly matching profiles.

To scale ER, blocking restricts the executed comparisons to similar entities. It groups them into clusters, called *blocks*, and performs pair-wise comparisons only between the entities of each block $b_i \subseteq E$. The size of b_i, i.e., the number of entities it contains, is denoted by $|b_i|$. The cardinality of b_i is denoted by $||b_i||$ and represents the total number of comparisons it contains: $||b_i|| = |b_i| \cdot (|b_i| - 1)/2$. A set of blocks is called *block collection* (B). Its size ($|B|$) denotes the number of blocks it contains, while its cardinality denotes the total number of comparisons it involves: $||B|| = \sum_{b_i \in B} ||b_i||$.

Following the best practice in the literature, we consider entity matching as an orthogonal task to blocking [6, 7, 26, 27]. That is, we assume that two duplicates are detected in B as long as they *co-occur* in at least one of its blocks. Provided that the vast majority of duplicate entities are co-occurring, the performance of ER depends on the accuracy of the method used for profile comparison. The set of co-occurring duplicate entities is denoted by $D(B)$, while $|D(B)|$ symbolizes its size, i.e., their number.

2.1 Evaluation Measures

The *effectiveness* of a block collection B is assessed using three well-known measures [6, 7, 26, 27]:

(M1) *Pairs Completeness* (*PC*) estimates the *recall* of B, i.e., the portion of duplicates from the input entity collection(s) that co-occur in at least one block. More formally, $PC = |D(B)|/|D(E)|$. *PC* is defined in $[0, 1]$, with higher values showing higher recall.

(M2) *Pairs Quality* (*PQ*) estimates the *precision* of B, i.e., the portion of its comparisons that correspond to duplicate entity profiles. More formally, $PQ = |D(B)|/||B||$. *PQ* takes values in the interval $[0, 1]$, with higher ones indicating higher precision.

(M3) *Reduction Ratio* (*RR*) estimates the portion of comparisons that are saved by a blocking method in relation to the naive, brute-force approach. Formally, it is defined as: $RR(B, E) = 1 - ||B||/||E||$, where $||E||$ denotes the computational cost of the naïve approach, i.e., $||E|| = |E| \cdot (|E| - 1)/2$. *RR* takes values in the interval $[0, 1]$, with higher values indicating higher efficiency.

In theory, the goal of blocking is to maximize all three measures. That is, to maximize the number of detected duplicates ($|D(B)|$), while minimizing the cardinality of B ($||B||$). In practice, though, there is a clear trade-off between *PC* and *PQ-RR*: the more comparisons are executed (higher $||B||$), the more duplicates are detected (higher $|D(B)|$), thus increasing *PC*; given, though, that $||B||$ increases quadratically for a linear increase in $|D(B)|$, *PQ* and *RR* are reduced [13, 14]. Therefore, blocking methods should aim for a balance between precision (*PQ*) and recall (*PC*) that minimizes the executed comparisons and ensures that most matching entities co-occur (i.e., high *PC*). Achieving high *PC* (> 0.80) is critical; otherwise, blocking will miss a significant portion of duplicates even if iterative methods, such as iterative blocking [30], are used.

The *time efficiency* of blocking is assessed using two measures:

(M4) *Overhead Time* (*OTime*) is the total time required by a blocking method to cluster the given entity profiles into blocks, i.e., the time between receiving the original entity collection(s) and returning a set of blocks as output.

(M5) *Resolution Time* (*RTime*) adds to *OTime* the time required for performing all pair-wise profile comparisons with a specific entity matching technique. As such, we consider the Jaccard similarity of the tokens in all attribute values of the compared entities.

3. BLOCKING FOR STRUCTURED DATA

The main blocking methods for structured data have been summarized in the recent survey by Christen [7]. Figure 2 presents an overview of them, where every edge $A \rightarrow B$ indicates that method B improves on method A either by modifying the definition of the blocking keys or by changing the way they are used for the creation of blocks. We now describe the functionality of every method with respect to its schema-based and its schema-agnostic configuration.

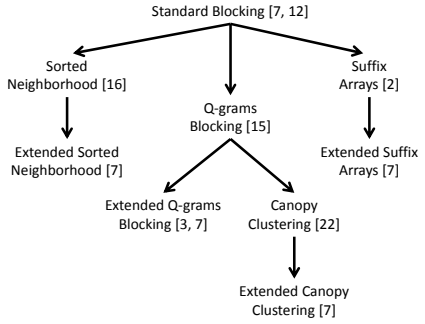

Figure 2: The main blocking methods for structured data.

Standard Blocking (`StBl`) [7, 12] is the cornerstone technique, as all other methods rely on it and aim to improve its simple functionality. It represents every entity by one or more keys and creates blocks on their equality; that is, every block corresponds to a specific key and contains all entities that have it in their representation. Its *schema-based* configuration is illustrated in the example of Figure 1(b), while Figure 1(c) demonstrates its *schema-agnostic* functionality, which is equivalent to Token Blocking [26].

Sorted Neighborhood (`SoNe`) [16] uses the same blocking keys as Standard Blocking, but builds blocks on their similarity instead of their equality: it sorts the blocking keys in alphabetical order and slides a window of fixed size w over the resulting list of ordered entities. In every step, it compares the entities that co-occur within the same window. In the example of Figure 1(b), the sorted list of keys would be {*BRE*_104, *GEO_CAR*} and of entities $\{p_1, p_3, p_2, p_4\}$.[1] Thus, for a window of size $w = 2$, we get the blocks $b_1 = \{p_1, p_3\}$, $b_2 = \{p_3, p_2\}$, $b_3 = \{p_2, p_4\}$. Its *schema-based* configuration usually employs multiple key definitions and forms a separate sorted list for each of them. The *schema-agnostic* configuration uses a single list of sorted keys that comprises all tokens in the values of all entities. For both configurations, entities with the same key are placed in the sorted list in a random order.

Extended Sorted Neighborhood (`ESoNe`) [7] aims for addressing the main drawback of Sorted Neighborhood, namely its sensitivity on the window size, w: small windows lead to low recall, but large ones yield many comparisons and low precision. `ESoNe` offers more robust performance by sliding the window over the sorted keys, instead of the sorted entities. In the simplified example of `SoNe`, a window of size $w = 2$ for `ESoNe` would create a single block for the two blocking keys, placing all associated entities in it: $b_1 = \{p_1, p_3, p_2, p_4\}$. Again, the *schema-based* configuration employs a different list for each blocking key definition, whereas the *schema-agnostic* one uses a single list of sorted keys/tokens.

Q-grams Blocking (`QGBl`) [15] relies on the blocking keys of Standard Blocking, but transforms them in a format that ensures higher resilience to noise. Instead of using the entire keys, it considers their q-grams (i.e., sub-sequences of q characters) and builds blocks on their equality. Assuming that $q = 3$, the blocking key *BRE*_104 in the example of Figure 1(b) would be transformed into the following keys: *BRE*, *RE*_, *E*_1, _10 and 104. The *schema-based* definitions apply this transformation to the keys that are extracted from specific (combinations of) attribute names, whereas the *schema-agnostic* configuration applies it to all tokens in the val-

[1] For simplicity, we assume that entities with the same blocking key are sorted by their id and that both blocking keys are extracted from the same combination of attribute names.

		Size Limit	
		Size-free	Size-constrained
Block Definition	Equality-based	• Standard Blocking (StBl) • Q-Grams Blocking (QGBl) • Extended Q-Grams Blocking (EQGBl)	• Suffix Arrays (SuAr) • Extended Suffix Arrays (ESuAr)
	Similarity-based	• Extended Sorted Neighborhood (ESoNe)	• Sorted Neighborhood (SoNe) • Canopy Clustering (CaCl) • Extended Canopy Clustering (ECaCl)

Figure 3: Two-dimensional taxonomy of blocking methods.

ues of all entities. In both cases, a new block is created for every q-gram that appears in at least two entities.

Extended Q-grams Blocking (EQGBl) [3, 7] improves on QGBl by producing blocking keys of higher discriminativeness. In this way, it tries to decrease the number of pairwise comparisons in the resulting blocks, without missing any of the detected duplicates. More specifically, instead of individual q-grams, EQGBl considers combinations that stem from the concatenation of at least L q-grams. L is derived from a user-defined parameter, called threshold (T), as follows: L=$max(1, \lfloor k \cdot T \rfloor)$, where k is the number of q-grams in the original blocking key. T is defined in the interval $[0, 1)$, with larger values reducing the number of combinations. In our example, the following combinations are used as keys for T=0.9 and L=4: *BRERE_E_1_10104*, *BRERE_E_1104*, *BRERE_10104*, *RE_E_1_10104*, *BRERE_E_1_10*, *BREE_1_10104*. Again, the *schema-based* configuration extracts such keys from the (combined) values of selected attribute names, whereas the *schema-agnostic* one extracts them from the tokens of all attribute values. Blocks are then created for every key that appears in at least two entities.

Canopy Clustering (CaCl) [22] uses the same blocking keys as Q-grams Blocking, but creates blocks on their similarity, instead of their equality. Initially, it places all entities in a pool of candidate matches, P. In every iteration, it removes a random *seed* p_i from P and compares its blocking keys (q-grams) with that of all other entities in P. Those entities with a similarity higher than w_1 are placed in a new block together with p_i, while those exceeding another threshold w_2($>w_1$) are removed from the pool.

Extended Canopy Clustering (ECaCl) [7] addresses the main drawback of Canopy Clustering, namely its sensitivity to the values of the weight thresholds: if w_1 is too high, many entities will be placed in no block. To avoid missing any entity, ECaCl replaces w_1 and w_2 with two cardinality thresholds, n_1 and n_2, such that $1 \leq n_2 \leq n_1$. Similar to CaCl, it initially places all entities in a pool P and iteratively removes a random seed p_i from it. In every iteration, it compares p_i with all entities still in P using the blocking keys defined by Q-grams Blocking. Then, instead of using weight thresholds, it inserts the n_1 nearest neighbors in a new block – together with p_i – and removes the n_2 nearest neighbors from P.

Suffix Arrays (SuAr) [2] improves the noise-tolerance of Standard Blocking by applying another transformation to its keys: it converts them into the suffixes that are longer than a minimum length l_m. In our example, the blocking key *BRE_104* would be converted into the suffixes *BRE_104*, *RE_104* and *E_104* for l_m = 5. The *schema-based* configuration applies this transformation to the combined values of selected attribute names, while the *schema-agnostic* one applies it to the tokens of all attribute values. Blocks are created on the equality of blocking keys. Additionally, SuAr discards the highly frequent suffixes in order to avoid the creation of very

Figure 4: The workflow for size-free blocking methods with schema-agnostic configuration.

large blocks: keys that appear in more than b_M entities are ignored.

Extended Suffix Arrays (ESuAr) [7] improves the robustness of SuAr by supporting noise at the end of blocking keys. It converts every key of Standard Blocking into all substrings that are larger than the minimum length l_m. Continuing our example, *BRE_104* would be transformed into *BRE_104*, *BRE_10*, *RE_104*, *BRE_1*, *RE_10* and *E_104* for l_m = 5. The *schema-based* configuration considers only the (combined) values of selected attributes, while the *schema-agnostic* one considers all tokens from all attribute values. Again, b_M sets an upper limit on the size of the resulting blocks.

3.1 Taxonomy

We now introduce a new taxonomy of the blocking methods that is formed by two orthogonal criteria, one pertaining to time efficiency and the other to effectiveness.

We call the first criterion *block definition*, as it categorizes the blocking methods based on how they use their keys. The *equality-based* methods define blocks on the equality of their keys. Thus, they can be efficiently implemented with an inverted index that points from blocking keys to entities – a block is simply created for every posting list that contains at least two entries [7]. To this category belong StBl, QGBl, EQGBl, SuAr and ESuAr. The remaining methods (SoNe, ESoNe, CaCl, ECaCl) are *similarity-based*, because they derive blocks from the similarity of their keys. Their implementation relies on inverted indices too [7], but they apply additional processing to the posting lists, thus involving a higher overhead than the equality-based methods. Hence, we expect this criterion to be critical for the time efficiency of blocking methods.

The second criterion, called *size limit*, distinguishes the blocking methods into *size-constrained* and *size-free* ones. The former inherently impose a limit on the maximum eligible size of individual blocks, i.e., on the maximum number of entities per block. This limitation is expressed through the window size w in SoNe, the parameter n_1 in ECaCl and the parameter b_M in SuAr and ESuAr. Alternatively, size-constrained methods impose a limit on the size of entire block collections, specifying the maximum number of blocks that can be extracted from the input entity collection E. This limitation applies to CaCl, which creates at most one block per entity. In contrast, the *size-free* methods do not impose any limit on the size or the number of their blocks. StBl, ESoNe, QGBl and EQGBl fall in this category. We expect this criterion to be decisive for the effectiveness of blocking methods, since a size limit results in fewer comparisons, decreasing *PC* at the benefit of higher *PQ* and *RR*.

In combination, the two criteria form the two-dimensional taxonomy that is presented in Figure 3 along with the methods belonging to every category. Note that every category is represented by at least one blocking method. For presentation clarity, the methods of each category are indicated with a different color. In Section 5, we examine analytically how these categories affect the effectiveness, the time efficiency and the scalability of the blocking methods.

4. EXPERIMENTAL SETUP

We implemented all blocking methods and experiments in Java, version 8. The experiments were performed on a desktop computer with Intel i7 (3.4GHz) and 16GB of RAM, running Ubuntu 14.04 (kernel version 3.13.0-45). All time measurements were repeated 10 times and the average value is reported.

	D_{census}	D_{rest}	D_{cora}	D_{cddb}
$\|E\|$	841	864	1,295	9,763
$\|D(E)\|$	344	112	17,184	299
$\|N\|$	5	5	12	106
$\|V\|$	3,913	4,319	7,166	183,072
$\|\bar{p}\|$	4.65	5.00	5.53	18.75
$\|\|E\|\|$	$3.53{\cdot}10^5$	$3.73{\cdot}10^5$	$8.38{\cdot}10^5$	$4.77{\cdot}10^7$
$RT(E)$	2 s	3 s	11 s	25 min

(a)

	D_{10K}	D_{50K}	D_{100K}	D_{200K}	D_{300K}	D_{1M}	D_{2M}
$\|E\|$	10,000	50,000	100,000	200,000	300,000	1,000,000	2,000,000
$\|D(E)\|$	8,705	43,071	85,497	172,403	257,034	857,538	1,716,102
$\|N\|$	12	12	12	12	12	12	12
$\|V\|$	106,108	530,854	1,061,421	2,123,728	3,184,885	10,617,729	21,238,252
$\|\bar{p}\|$	10.61	10.62	10.61	10.62	10.62	10.62	10.62
$\|\|E\|\|$	$5.00{\times}10^7$	$1.25{\times}10^9$	$5.00{\times}10^9$	$2.00{\times}10^{10}$	$4.50{\times}10^{10}$	$5.00{\times}10^{11}$	$2.00{\times}10^{12}$
$RT(E)$	10 min	3.5 hrs	14 hrs	77 hrs	134 hrs	~1,380 hrs	~5,500 hrs

(b)

Table 1: (a) The real and (b) the synthetic datasets, ordered in increasing size from left to right. $|E|$ stands for the number of entity profiles, $|D(E)|$ for the number of duplicate pairs, $|N|$ for the number of distinct attribute names, $|V|$ for the total number of name-value pairs, $|\bar{p}|$ for the mean number of name-value pairs per profile, $||E||$ for the number of comparisons executed by the brute-force approach and $RT(E)$ for the respective resolution time.

4.1 Datasets

Our experimental analysis involves several homogeneous data collections, both real and synthetic ones, that have been widely used in the literature [6, 7, 8, 9, 10].

The real datasets were employed in the survey by Christen [7]. D_{census} involves records generated by the US Census Bureau based on real census data, D_{rest} contains records from the Fodor and Zagat restaurant guides, D_{cora} has bibliographic records of machine learning publications, and D_{cddb} contains records of audio CDs, randomly selected from freeDB (`http://www.freedb.org`). Their technical characteristics are presented in Table 1(a). Compared to their characteristics in [7], there are some differences for D_{census} and D_{cddb}. The former was used for Record Linkage in [7] on the assumption that it can be partitioned in two duplicate-free entity collections; given, though, that both collections contain duplicates in themselves, the dataset is only suitable for Deduplication. For D_{cddb}, we use an updated, corrected ground-truth that involves half the original pairs of duplicates [8, 9].

These datasets are small enough for the brute-force approach to scale. Yet, we used them in our experimental analysis for two reasons: *(i)* They differ largely in their characteristics, allowing us to test the blocking methods in versatile conditions. For example, D_{cddb} contains multilingual records, while the number of duplicate pairs is an order of magnitude smaller than the number of entities; in contrast, D_{cora} exclusively contains records in English, while the equivalence clusters are so large that the number of duplicate pairs is an order of magnitude higher than the number of entities. *(ii)* Every dataset is associated with three different schema-based configurations of blocking keys that have been evaluated in [7]. In this way, we are able to perform a thorough comparison with the schema-agnostic configuration for every blocking method and dataset.

The synthetic, homogeneous datasets were created by the Febrl data generator [5] according to the parameters specified in [7]. First, duplicate-free census records were produced based on frequency tables of real-world names and addresses. Then, duplicates of these records were randomly generated based on real-world error characteristics and modifications. The resulting datasets contain 40% duplicate records with up to 9 duplicates per record, no more than 3 modifications per attribute, and up to 10 modifications per record. Their technical characteristics are presented in Table 1(b). Again, every dataset is associated with three schema-based definitions of blocking keys that were used in the experimental analysis of [7].

4.2 Evaluated Workflows

In our experimental study, we do not consider the above blocking methods in isolation. Instead, we combine them with *block processing* techniques in order to improve their performance. These methods receive as input an existing block collection and transform it to a new block collection that contains fewer unnecessary comparisons. Such comparisons do not contribute to recall, but merely decrease precision. They come in two forms [26, 27]: the *redundant* ones repeatedly compare the same entities in different blocks, while the *superfluous* ones compare two non-matching entities. In the example of Figure 1(c), the comparison in b_2 is redundant, repeated in b_1, while the comparison p_1-p_2 in b_4 is superfluous.

We now explain the functionality of the two block processing techniques we employ in our experiments.

Comparison Propagation cleans a set of overlapping blocks from all redundant comparisons, without any impact on recall. For small datasets, this can be accomplished through a central data structure that stores all executed comparisons in memory. This naïve approach does not scale to large datasets with (tens of) billions of comparisons. In this case, the executed comparisons have to be stored indirectly [26]: every block is associated with an id indicating its position in the processing order and an inverted index points from entity ids to block ids; a pairwise comparison is identified as redundant if the id of the current block is larger than the least common block id of the involved entities.

Block Filtering [28] applies to the size-free blocking methods that are coupled with the schema-agnostic configuration. The reason is that their block collections have a power-law distribution in their block cardinalities. That is, the vast majority of blocks are of minimum size, involving just one comparison, while the frequency of blocks decreases with the increase of block cardinality. In this distribution, the larger a block is, the less likely it is to contain duplicate entities [25, 26]. Based on this pattern, Block Filtering operates on individual entities to remove them from the blocks that are the least important for them. First, it sorts all blocks in ascending order of cardinality, from the smallest to the largest one. Then, it uses a ratio $r \in [0, 1]$ to specify the percentage of the most important (smallest) blocks that every entity is retained in. Block Filtering targets both redundant and superfluous comparisons at a small cost in recall, which depends on the configuration of r.

We apply Comparison Propagation to the block collections of all the blocking methods for both configurations, because they all yield overlapping blocks. For the size-free methods with schema-agnostic configuration, we use both Comparison Propagation and Block Filtering as in the workflow of Figure 4.

4.3 Parameter Tuning

The functionality of every blocking method is determined by one or more parameters. We can distinguish them in two main types. The first one specifies the definition of blocking keys and pertains to Standard Blocking. The second type adjusts the use of Standard Blocking keys and pertains to the other blocking methods, which improve on it. We perform a holistic evaluation of both types, by testing several configurations for each of their parameters.

For the blocking keys of Standard Blocking, we employ three different schema-based configurations across all datasets. These configurations were used in the experimental analysis of [7]. Each

	StBl	QGBl	EQGBl	ESoNe	SoNe	CaCl	ECaCl	SuAr	ESuAr
BK$_1$	0.169	0.018±0.017	0.051±0.006	0.024±0.016	0.066±0.044	0.481±0.066	0.077±0.029	0.196±0.079	0.134±0.090
BK$_2$	0.159	0.040±0.048	0.135±0.003	0.027±0.017	0.057±0.040	0.704±0.108	0.073±0.030	0.186±0.081	0.132±0.079
BK$_3$	0.113	0.031±0.032	0.091±0.008	0.025±0.015	0.069±0.044	0.492±0.041	0.079±0.031	0.160±0.070	0.137±0.077
BK$_{all}$	0.054	0.022±0.013	0.023±0.001	0.015±0.011	0.030±0.018	0.227±0.116	0.068±0.026	0.093±0.026	0.080±0.039
				D_{census}					
BK$_1$	0.011	0.004±0.004	0.010±0.000	0.004±0.002	0.018±0.012	0.252±0.212	0.022±0.007	0.060±0.031	0.050±0.029
BK$_2$	0.001	$1.1_{\pm 0.3}\cdot 10^{-3}$	0.001±0.000	$9.7_{\pm 1.7}\cdot 10^{-4}$	0.008±0.004	0.009±0.002	0.027±0.010	0.044±0.020	0.039±0.019
BK$_3$	0.006	0.002±0.002	0.005±0.000	0.003±0.001	0.011±0.005	0.038±0.006	0.015±0.004	0.029±0.016	0.027±0.015
BK$_{all}$	0.003	0.002±0.001	0.003±0.001	0.001±0.001	0.005±0.003	0.960±0.013	0.025±0.013	0.024±0.016	0.020±0.015
				D_{rest}					
BK$_1$	0.458	0.055±0.029	0.277±0.126	0.151±0.073	0.407±0.084	0.951±0.065	0.484±0.080	0.583±0.020	0.479±0.036
BK$_2$	0.026	0.022±0.002	0.026±0.000	0.025±0.001	0.218±0.038	0.767±0.030	0.501±0.063	0.781±0.023	0.611±0.052
BK$_3$	0.583	0.082±0.065	0.392±0.080	0.198±0.095	0.430±0.080	0.976±0.039	0.498±0.083	0.678±0.028	0.569±0.048
BK$_{all}$	0.049	0.032±0.010	0.077±0.003	0.025±0.005	0.129±0.042	0.975±0.021	0.668±0.076	0.464±0.065	0.435±0.075
				D_{cora}					
BK$_1$	0.042	$9.1_{\pm 1.2}\cdot 10^{-5}$	0.016±0.004	0.007±0.005	0.009±0.006	0.042±0.012	0.005±0.002	0.007±0.004	0.006±0.004
BK$_2$	0.001	$2.0_{\pm 0.9}\cdot 10^{-5}$	$4.6_{\pm 2.0}\cdot 10^{-4}$	$1.0_{\pm 0.1}\cdot 10^{-3}$	0.004±0.002	$1.1_{\pm 0.4}\cdot 10^{-3}$	0.002±0.001	0.004±0.002	0.003±0.002
BK$_3$	0.010	$2.0_{\pm 0.9}\cdot 10^{-5}$	$1.0_{\pm 0.7}\cdot 10^{-3}$	0.003±0.001	0.005±0.002	0.004±0.004	0.002±0.001	0.004±0.002	0.003±0.002
BK$_{all}$	$6.1\cdot 10^{-5}$	$3.5_{\pm 3.3}\cdot 10^{-5}$	$1.4_{\pm 0.5}\cdot 10^{-4}$	$4.5_{\pm 1.3}\cdot 10^{-5}$	$2.6_{\pm 1.9}\cdot 10^{-4}$	0.898±0.024	0.008±0.004	0.002±0.001	$1.1_{\pm 0.7}\cdot 10^{-3}$
					D_{cddb}				

Table 2: Average value and standard deviation of *PQ* (precision) for the four blocking key configurations in combination with the main blocking methods across the real homogeneous datasets. The category of each method is indicated by its colour.

configuration represents every entity profile by two different blocking keys. The keys depend on the dataset's schema quality and are formed by concatenating a transformation of the values of 2 to 3 attribute names. We denote these schema-based configurations by BK_1, BK_2, BK_3. For every dataset, there is also a unique schema-agnostic configuration that is symbolized by BK_{all}.

For the internal parameters of the other methods, we consider several values, based again on the experimental analysis of [7]. For `SoNe` and `ESoNe`, we used the following sizes for their window: $w \in \{2, 3, 5, 7, 10\}$. For `QGBl`, we considered four lengths for its q-grams: $q \in \{2, 3, 4, 5\}$. For `EQGBl`, we employed the following combinations of length q and threshold T: $(q, T) \in \{(2, 0.9), (2, 0.8), (3, 0.9), (3, 0.8)\}$. For `CaCl` and `ECaCl`, we used the Jaccard coefficient to estimate the similarity between the blocking keys of two entities. The weight thresholds of the former method were set to (w_1, w_2)=(0.8, 0.9) and (w_1, w_2)=(0.7, 0.8), while the cardinality thresholds of the latter method were set to (n_1, n_2)=(10, 5) and (n_1, n_2)=(20, 10). Finally, for `SuAr` and `ESuAr`, we used exactly the same configurations: $l_m \in \{3, 5\}$ and $b_M \in \{5, 10, 20\}$. We applied all relevant settings to every blocking method and estimated the average value of the five measures in Section 2 in order to assess its performance with respect to effectiveness and time efficiency.

Regarding the configuration of the block processing techniques, Comparison Propagation is parameter-free. Its effect depends on the redundancy of blocks. In most cases, it reduces the number of executed comparisons by 10% to 30% [26]. Blocking Filtering involves a single parameter – the portion r of the retained blocks per entity. We set to $r = 0.64$, which retains every entity in around 2/3 of its most important (i.e., smallest) blocks. This value has been verified to reduce the total cardinality of blocks $\|B\|$ by at least an order of magnitude for a minor decrease (<5%) in recall [28].

5. EXPERIMENTAL RESULTS

We now compare the schema-agnostic configuration with the schema-based ones over all blocking methods and datasets in terms of effectiveness (Section 5.1), time efficiency (Section 5.2) and scalability (Section 5.3). In our analysis, we consider the two-dimensional taxonomy in Figure 3 as well as the relation between the basic and the extended version of the blocking methods.

5.1 Effectiveness

The outcomes of our experiments with respect to *PC* (**M1** in Section 2.1) and *RR* (**M3** in Section 2.1) are presented in Figures 5(A) to (I), on the left and the right column, respectively. All diagrams have the same scale and are aligned in a way that facilitates the comparison between the individual methods. *PQ* (**M2** in Section 2.1) is presented in Table 2. Remember that for all metrics, higher values correspond to better performance.

Starting with the size-free methods, we observe in Figures 5(A) to (D) that BK_{all} maintains a robust *PC* that exceeds 0.95 in almost all cases. The only exception is when used with `EQGBl` over D_{census}, where recall drops to 0.80: the small entity profiles (low $|\bar{p}|$ in Table 1(a)) and the high distinctiveness of the blocking keys lead to a small number of blocks per entity that are further reduced by Block Filtering. In contrast, the schema-based configurations of these methods exhibit rather unstable recall, with significant variation even for the same dataset. In many cases, their recall falls below 0.80, producing blocks that miss a significant portion of the existing duplicates. Compared to BK_{all}, their *PC* is consistently lower. Furthermore, there is no schema-based configuration that works well for all the datasets or for all the blocking methods.

With respect to *RR*, we observe that BK_1 and BK_3 exhibit equivalent performance across all datasets and methods. BK_2 follows them in close distance over D_{census} and D_{cddb}, but executes significantly more comparisons (lower *RR*) over D_{rest} and D_{cora}, due to the high frequency of its keys. For these two datasets, BK_{all} typically exhibits higher *RR* than BK_2. In all other cases, though, it scores the lowest value across all configurations. This should be attributed to the higher number of keys it involves, which produce a larger number of blocks. Given that the frequency of its keys is usually higher than the schema-based configurations, the total cardinality of the resulting blocks is significantly larger, in general. Their difference is so large, that despite its higher recall, BK_{all} scores the lowest *PQ* (precision) in the vast majority of cases (see Table 2). There are only few exceptions to this rule, mainly for D_{rest} and D_{cora}, where BK_{all} outperforms BK_2 with respect to *RR*.

In the case of size-constrained methods, Figures 5(E) to (I) reveal totally different patterns. First of all, there is a uniform behavior with respect to *RR*, as it exceeds 0.95 in almost all cases – there are just two exceptions for BK_{all} in combination with `SoNe`. These

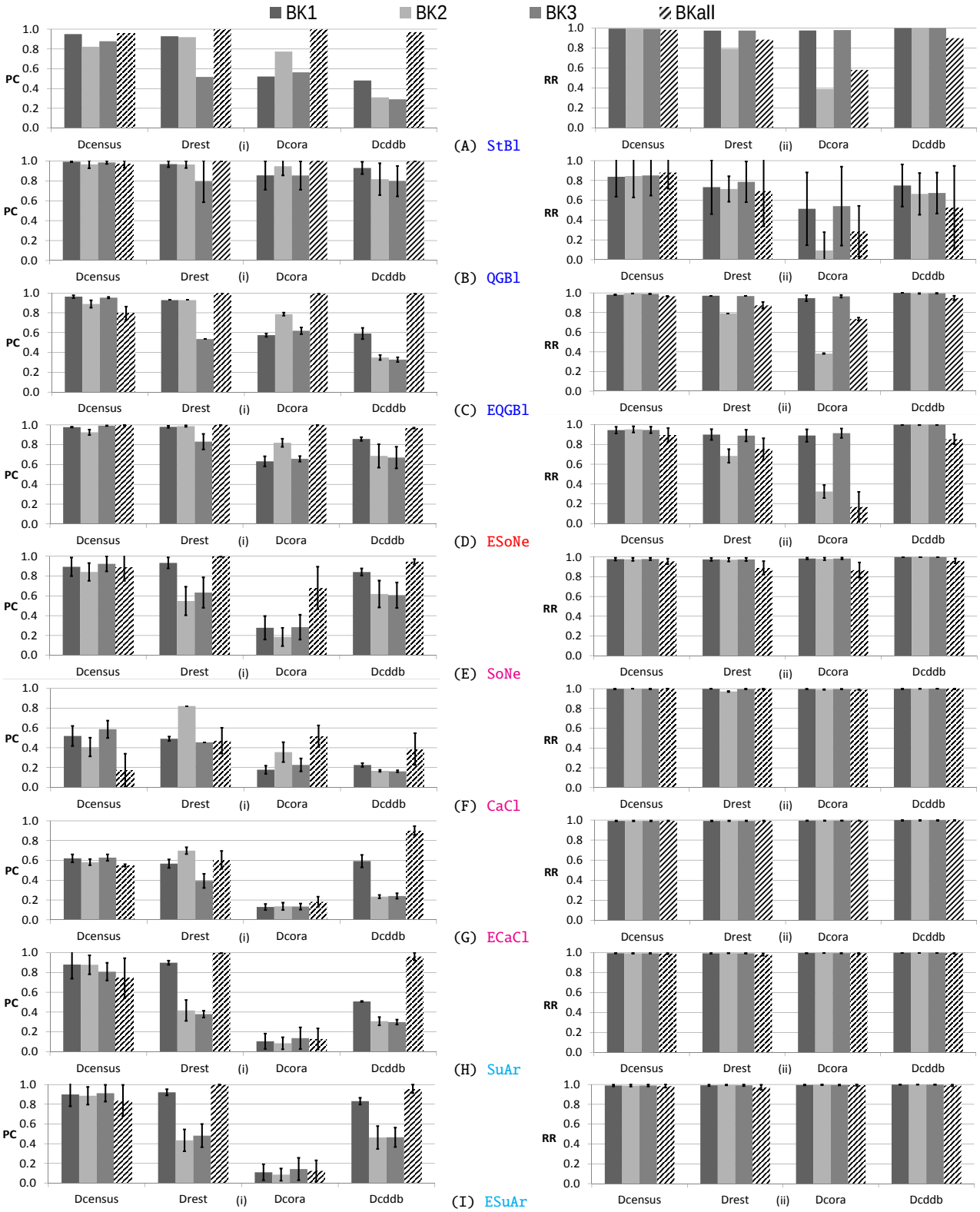

Figure 5: Effectiveness of the four blocking key configurations in combination with all blocking methods across the real datasets with respect to the average value and the standard deviation of (i) Pairs Completeness (left column) and (ii) Reduction Ratio (right column). For both measures, higher bars indicate better performance. The category of each method is indicated by its colour.

high values indicate that the blocks cardinality, $||B||$, is lower than the number of comparisons executed by the brute-force approach, $||E||$, by 2 to 3 orders of magnitude. In other words, the four configurations require practically equal number of comparisons across all datasets and methods. This should be attributed to the drastic effect of the size limit, which discards the overly frequent blocking keys.

Second, all configurations exhibit a rather unstable behavior with respect to *PC*. Among the schema-based ones, there is significant deviation even for the same dataset. BK_{all} outperforms them only in half the cases. Most importantly, few configurations per blocking method exceed 0.80 for any of the datasets. Particularly in D_{cora}, the maximum recall across all size-constrained methods and configurations is just 0.68. Given that the size-free methods achieve significantly higher recall over this dataset, we deduce that it has a limited vocabulary and, thus, most of its duplicate entities share only very frequent keys (to this attests the low *RR* over D_{cora} for many configurations of size-free methods). This applies to the other datasets as well, though to a lesser extent. Hence, the size-constrained methods should be combined with a size limit that is analogous to the number of input entities $|E|$ in order to produce blocks with sufficient recall. Instead, we used a fixed, low value across all datasets in order to repeat the experiments of [7].

On the whole, we conclude that for the size-free methods, the schema-agnostic configuration excels in *PC*, while the schema-based one emphasizes *RR* and *PQ* at the cost of insufficient recall. For the size-constrained methods, both key configurations minimize the number of executed comparisons; the recall is usually higher for the schema-agnostic keys, but it rarely exceeds 0.80. Despite the insufficient recall, the low number of comparisons leads to significantly higher *PQ* than the size-free methods in most cases.

Another interesting point is to compare every blocking method with its extended version. Starting with `SoNe`, we observe that `ESoNe` trades higher recall for lower *RR* and *PQ*. This should be expected, as the former is size-constrained and the latter size-free. All other pairs belong to the same category in our taxonomy. In the case of `QGBl` and `EQGBl`, the latter achieves significantly higher *RR* and *PQ* at the cost of lower *PC*. The reason is that `EQGBl` uses more distinctive keys, which appear in less entities, thus producing smaller blocks. For `CaCl`, its extended version increases *PC* in most cases, because it places every entity in at least one block. Despite the same *RR* with `CaCl`, `ECaCl` executes more comparisons in absolute numbers and decreases *PQ*. Exactly the same pattern applies to `SuAr` and `ESuAr`, because the latter uses more keys.

5.2 Time Efficiency

We now examine the time requirements of the four key configurations over the real datasets. Figure 6 shows *OTime* (**M4** in Section 2.1) in the left column, and *RTime* (**M5** in Section 2.1) in the right column. For both metrics, lower values indicate better time efficiency. All diagrams use exactly the same scale so as to facilitate the comparison between the two measures and between the various blocking methods and configurations. Due to the significant difference in the size of the four datasets, the vertical axes are logarithmic, with their maximum value corresponding to 67 minutes (4 million milliseconds). The time requirements of the brute-force approach, $RT(E)$, is reported in Table 1(a) – note that $RT(E)$=1.5 million milliseconds for D_{cddb}. We expect that a successful blocking method should exhibit a significantly lower resolution time than the brute-force approach, in addition to a recall over 0.80.

Starting with *OTime*, we first examine the relative efficiency of schema-based and schema-agnostic configurations. With a few exceptions, BK_{all} requires significantly more time than the schema-based configurations for all datasets and blocking methods. The differences increase when moving from left to right, i.e., from smallest dataset (D_{census}) to the largest one (D_{cddb}) due to two factors. First, the larger datasets happen to convey more information per entity than the smaller ones, as implied by $|\bar{p}|$ in Table 1(a). Hence, the larger a dataset is, the larger is the difference in the number of keys used by BK_{all} and the schema-based configurations. Second, larger datasets produce more comparisons that have to be processed by Comparison Propagation, increasing its overhead. Consequently, the distance of BK_{all} from the other configurations rises to 2 orders of magnitude for many blocking methods over D_{cora} and D_{cddb}.

Depending on the blocking method, though, there are additional factors that increase the distance of BK_{all} from the schema-based configurations with respect to *OTime*. For the size-free methods, Block Filtering brings an additional computational cost. For `SoNe` and `ESoNe`, BK_{all} pays a higher cost for sorting and traversing a larger list of blocking keys. For `CaCl` and `ECaCl`, the workload of Comparisons Propagation plays a minor role, as all configurations yield a limited number of comparisons. Instead, *OTime* depends on the size of the signatures that represent every entity and are compared with Jaccard similarity. The larger they are, the more time-consuming is their processing (in Section 6, we discuss possible ways for accelerating this procedure).

Regarding the relative overhead of the schema-based configurations, there are minor differences between them in most cases, since none of the above factors apply: they all use the same number of signatures per entity and yield a similar number of blocking keys for every dataset. They only differ in the number of comparisons they forward to Comparison Propagation for processing. Thus, this factor accounts for all their differences with respect to *OTime*.

It is also interesting to examine the relative overhead of equality- and similarity-based methods. We compare `StBl` in Figure 6(A) with its similarity-based counterparts, `SoNe` and `ESoNe` in Figures 6(E) and (D), respectively. `StBl` is faster than `ESoNe` across all datasets and key configurations, requiring 74% less time on average. The main reason is that `ESoNe` is a size-free method that yields blocks with significantly larger cardinalities than `StBl`. Thus, it increases the workload of Comparison Propagation. Compared to `SoNe`, `StBl` requires half the time, on average, when combined with the schema-based configurations; for BK_{all}, its advantage is bigger as it requires up to a whole order of magnitude less time than `SoNe`. There are just 2 exceptions out of the 20 cases, where the size-constrained functionality of `SoNe` yields significantly fewer comparisons than `StBl`, thus involving a lower overhead.

The impact of size constraints becomes clearer when comparing `QGBl` in Figure 6(B) with its similarity-based counterparts, `CaCl` and `ECaCl`. The latter method (Figure 6(G)) is faster than `QGBl` across all datasets and configurations by 55%, on average. For BK_{all}, in particular, `ECaCl` requires at least 85% less time than `QGBl`. This counter-intuitive pattern should be attributed to the high value of parameter n_2, which determines the number of entities that `ECaCl` removes from the pool of candidate matches in every iteration; the higher n_2 is, the less iterations `ECaCl` performs and the faster it gets. Furthermore, `QGBl` yields blocks with significantly higher cardinality, thus increasing the overhead of Comparison Propagation.

In the case of `CaCl` (Figure 6(F)), the number of iterations is implicitly specified through w_2. The higher w_2 is, the fewer entities are removed from the pool at each round, the more iterations are performed and the more time it requires. Due to the high weight thresholds we apply to `CaCl`, `QGBl` is faster by 32%, on average, in 10 out of the 12 combinations with schema-based configurations. For BK_{all}, though, the situation is reversed, with `CaCl` being consistently faster by 75%. Again, this is caused by Comparison Propagation, which slows down `QGBl`, due to the very large blocks produced in combination with BK_{all} (low *RR* in Figure 5(B-ii)).

On the whole, we can conclude that the similarity-based methods involve a more time-consuming functionality than their equality-based counterparts in the general case. This relation can only be reversed by the overhead of Comparison Propagation in cases where

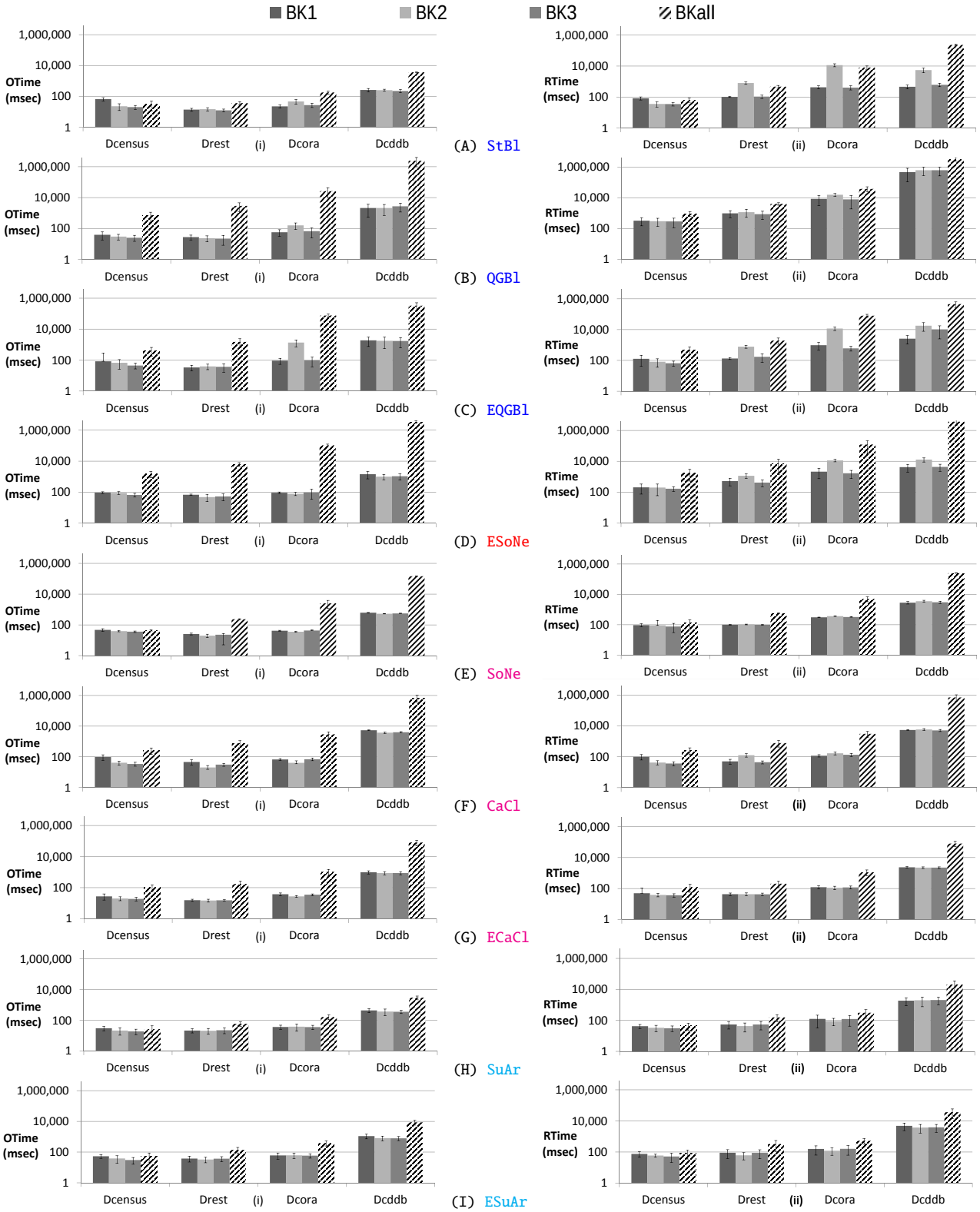

Figure 6: Time efficiency of the four blocking key configurations in combination with all blocking methods across the real datasets with respect to the average value and the standard deviation of (i) Overhead Time (left column) and (ii) Resolution Time (right column). In all diagrams, the vertical axis is logarithmic. For both measures, lower bars indicate better performance. The category of each method is indicated by its colour.

the former methods execute significantly fewer comparisons than they latter, due to their combination with a strict size limit.

Regarding *RTime*, remember that it equals *OTime* plus the time required for performing the pairwise comparisons in the resulting

blocks. This means that it is highly correlated with *OTime*, especially for blocks with low total cardinality. As a result, *RTime* maintains the relative efficiency of the 4 key configurations across most datasets and methods; the only differences arise between the schema-based configurations, in cases where they differ significantly in the number of executed comparisons. Most importantly, though, *RTime* saves at least 50% of the time required by the brute-force approach, $RT(E)$, in the vast majority of cases. For BK_{all}, the only exceptions are its combination with ESoNe and QGBl over the 3 largest datasets as well as QGBl over D_{cora}. For the schema-based configurations, BK_2 is slower than $RT(E)$ over D_{cora} in combination with StBl, ESoNE, QGBl and EQGBl. All these exceptions correspond to low *RR* and, thus, involve high time requirements both for Comparison Propagation and the execution of comparisons.

5.3 Scalability

To assess the scalability of the four key configurations, we applied them to the seven synthetic datasets of Table 1(b). Figure 7 presents the blocks cardinality, $||B||$, on the left column, and recall, *PC*, on the right column. Again, the same scale is used in all figures of each column to facilitate the comparisons. The horizontal axes are logarithmic and express the number of input entities, $|E|$. Also logarithmic are the vertical axes on the left column.

Starting with the *size-free methods* in Figures 7(A) to (D), we observe that BK_{all} executes at least an order of magnitude more comparisons than the schema-based configurations, due to its larger number of keys. For the same reason, its *PC* is consistently higher by at least 20% across most methods; the only exception is QGBl, where all configurations execute practically the same number of comparisons and the recall of BK_{all} exceeds that of BK_1 by 7%, of BK_2 by 12%, and of BK_3 by just 3%. Most importantly, though, the recall of the schema-based configurations is below 0.80 in most cases, whereas BK_{all} achieves almost perfect recall. This pattern demonstrates that the schema-agnostic configuration of the size-free methods emphasizes recall, while the schema-based configurations focus on precision, at the cost of insufficient *PC*.

More complex patterns appear in the case of *size-constrained methods* in Figures 7(E) to (I). All configurations yield similar $||B||$ in combination with all methods except SoNe, where BK_{all} executes 2 to 5 times more comparisons (due to the higher number of blocking keys). With respect to *PC*, there are three different patterns: *(i)* For SoNe, SuAr and ESuAr, the recall of BK_{all} starts from a value higher than 0.90 and decreases steadily with the increase of the input entities; for D_{2M}, its *PC* consistently drops to 0.75. The schema-based configurations exhibit a similar behavior, but they begin from and end at much lower levels of recall. *(ii)* For ECaCl, BK_{all} retains the maximum *PC*, but its absolute value fluctuates between 0.45 and 0.40; for the schema-based configurations, *PC* drops below 0.40 and decreases for larger input entity collections. *(iii)* For CaCl, the relative performance of the configurations is reversed. The lowest *PC*, just 0.10, corresponds to BK_{all}, while the schema-based configurations fluctuate between 0.15 and 0.22.

Note that the behavior of SoNe, SuAr and ESuAr contradicts the recall patterns observed in Figures 5 (D-i), (H-i) and (I-i), respectively. The reason is that the synthetic datasets have more diverse vocabularies than the real ones. Thus, the duplicate entities share keys that are not frequent enough to be pruned by the size limit. This does not apply to CaCl and ECaCl, which rely on the similarity rather than the frequency of keys. Their recall continues to be poor, as their restrictive configurations allow every entity to be compared with just 2 other profiles, on average. Thus, they need further fine-tuning in order to produce useful blocks.

Another interesting aspect is to examine which configurations and blocking methods scale linearly to the increased size of the input entity collections. Given that D_{2M} is 200 times larger than D_{10K}, a linear increase in $||B||$ requires that the blocks for D_{2M} involve two orders of magnitude more comparisons than those for D_{10K}. This condition is satisfied only by the size-constrained blocking methods – regardless of the key configuration. The size-free blocking methods are less effective in this respect, relying on Block Filtering for this purpose. Given that we use a fixed configuration for Block Filtering, its impact depends heavily on the configuration of blocking keys. Hence, the size-free methods scale linearly only for BK_1 in combination with StBl and ESoNe.

It is also interesting to estimate the portion of saved comparisons in relation to the brute-force approach (i.e., *RR*). We can do so by comparing $||B||$ on the left column of Figure 7 with $||E||$ in Table 1(b). The worst *RR* corresponds to QGBl, which saves just 68% of the possible comparisons across all datasets, regardless of the key configuration. All other blocking methods achieve a very high *RR* that practically equals 1.00.[2] For the size-constrained methods, the larger the dataset, the more comparisons they discard: they start saving 1 to 2 orders of magnitude for D_{10K} and end up saving up to 6 orders of magnitude for D_{2M}. On the other hand, the size-free methods reduce the executed comparisons by 1 to 3 orders of magnitude in combination with all blocking key configurations.

On the whole, we can conclude that BK_{all} scales much better than the brute-force approach with respect to blocks cardinality and much better than the schema-based configurations with respect to recall. For size-free methods, BK_{all} consistently maintains *PC* well over 0.90 and maximizes *PQ* in combination with EQGBl, which executes more than two orders of magnitude less comparisons than the brute-force approach. For the size-constrained blocking methods, except CaCl and ECaCl, the recall of BK_{all} lies higher than or close to 0.80, while its blocks cardinality scales linearly to the number of input entities. Its maximum *PQ* corresponds to SuAr, which executes 6 orders of magnitude less comparisons than the brute-force approach.

6. RELATED WORK

General discussions about blocking methods can be found in several surveys, books and tutorials about Entity Resolution [6, 11, 13, 14, 20]. In this work, we exclusively consider *approximate* blocking techniques, which sacrifice recall to a minor extent for significantly higher precision. We do not cover *exact* blocking methods, which guarantee that all pairs of duplicates are identified (i.e., they achieve perfect recall). Methods of this type are employed by SILK [17] and Limes [24], but are less popular in the literature, because the approximate techniques are more flexible in reducing the number of executed comparisons. We also excluded from our analysis the blocking methods that are inherently crafted for heterogeneous data collections, such as Attribute Clustering [26] and Total Description [25]; they already involve a schema-agnostic configuration, but are incompatible with the schema-based one.

More relevant to our work is the recent experimental analysis of approximate blocking methods for structured data by Christen [7]. Our study builds on this work, using the same real homogeneous datasets, almost the same blocking methods and the same configurations for each blocking method. Our conclusions verify those of [7] for the common part of the experimental study. Yet, we go beyond this study in several ways: *(i)* we examine the performance of the schema-agnostic configuration for the selected blocking methods, *(ii)* we elucidate the experimental results based on

[2]Given that so high values would prevent us from drawing detailed conclusions, we present $||B||$ instead of *RR* in Figures 7(A) to (I).

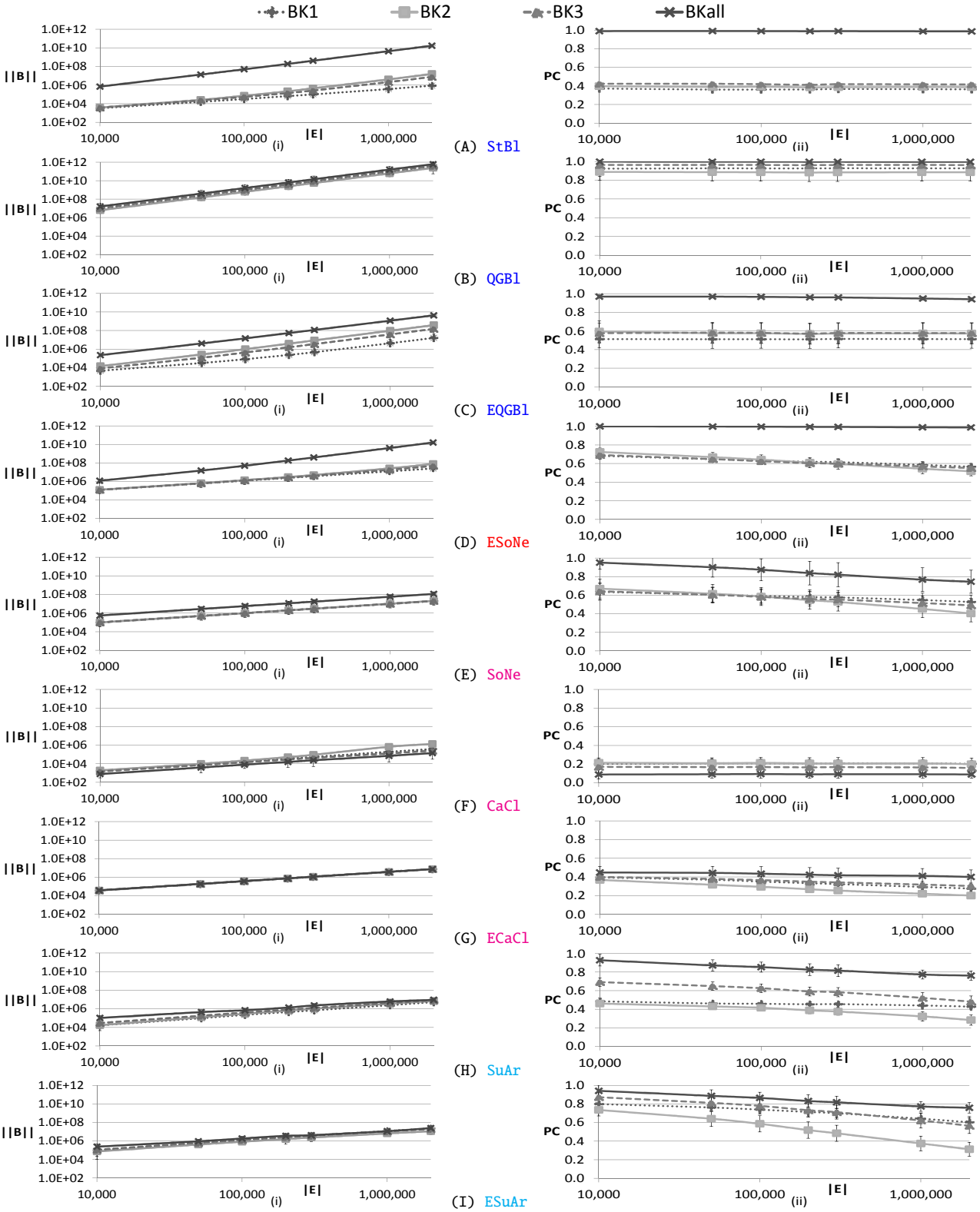

Figure 7: Scalability of the four blocking key configurations in combination with all blocking methods across all synthetic datasets with respect to the average value and the standard deviation of (i) Blocks Cardinality ($\|B\|$, left column) and (ii) Pairs Completeness (PC, right column). For $\|B\|$, lower values indicate better performance and vice versa for PC. The horizontal axes are logarithmic.

our taxonomy of blocking methods, *(iii)* we consider larger, synthetic datasets for the scalability analysis, *(iv)* we examine the performance of the three schema-based configurations per dataset independently of one another, while [7] presents only their average performance per dataset, *(v)* we investigate the time requirements of every configuration and blocking method in detail with respect

to both Overhead and Resolution Time, and *(vi)* we apply Comparison Propagation to every method and configuration, improving precision at no cost in recall. We also replaced StringMap [19] with Q-Grams Blocking so as to compare EQGBl with its basic version.

Also relevant to our work is a recent experimental evaluation of the major techniques for efficient string similarity joins [18]. Typically, these methods are applied to an existing block collection in order to perform entity matching, comparing the profiles that co-occur in the same block. It is possible, though, to integrate them into blocking methods so as to accelerate their processing. More specifically, both versions of Canopy Clustering can benefit from such techniques in order to accelerate the comparison of the blocking keys of two entities. Indeed, the outcomes of [18] indicate the most efficient method for similarity join depending on the size of the blocking keys and the corresponding weight threshold. This is particularly useful for the schema-agnostic configurations of CaCl and ECaCl, which involve very large signatures.

7. CONCLUSIONS

Our thorough experimental study leads to several novel conclusions about the performance of blocking methods.

First, different schema-based configurations yield a similar number of comparisons when applied to the same blocking method and dataset, but exhibit a rather unstable performance with respect to *PC*. Their recall is usually lower than 0.80, especially when combined with size-constrained methods, thus missing a considerable part of the existing duplicates.

Second, the schema-agnostic configuration of size-free methods consistently achieves a very high recall that is robust to the effect of the other parameters. It also saves a large number of the comparisons executed by the brute-force approach, especially in combination with StBl and EQGBl. Another advantage of these two methods is that they involve a low overhead, partly because they are equality-based. Note also that StBl is parameter-free for the schema-agnostic configuration.

Third, the size-constrained methods reduce the executed comparisons by whole orders of magnitude compared to the brute-force approach. Yet, their recall is very sensitive to the size limit, which should be analogous to the number of input entities, taking low values only in the case of datasets with diverse vocabulary. Their overhead is also very low, especially when using one of the equality-based techniques: SuAr and ESuAr. The schema-agnostic is again the preferable configuration.

On the whole, we conclude that the schema-agnostic configuration offers a reliable, robust alternative to the schema-based blocking keys. Further, it turns the blocking methods for structured data applicable to heterogeneous data and facilitates their use, as it requires no human intervention. On the flip side, it typically executes a significantly higher number of comparisons, downgrading precision, and results in a higher resolution time.

In the future, we plan to apply the blocking methods for structured data to heterogeneous data with the help of their schema-agnostic configuration. Then, we will compare them with blocking methods that are inherently crafted for heterogeneous data.

Acknowledgements. This work was partially supported by EU H2020 BigDataEurope (#644564), OpenAIRE2020 (#643410) and SlideWiki (#688095) projects.

References

[1] Blocking framework: http://sourceforge.net/projects/erframework.

[2] A. N. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, pages 30–39, 2005.

[3] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *Workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 25–27, 2003.

[4] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.

[5] P. Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD*, pages 1065–1068, 2008.

[6] P. Christen. *Data Matching*. Data-centric systems and applications. Springer, 2012.

[7] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.

[8] U. Draisbach and F. Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, pages 51–56, 2009.

[9] U. Draisbach and F. Naumann. Dude: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010.

[10] U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg. Adaptive windows for duplicate detection. In *ICDE*, pages 1073–1083, 2012.

[11] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[12] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[13] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[14] L. Getoor and A. Machanavajjhala. Entity resolution for big data. In *KDD*, page 1527, 2013.

[15] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[16] M. Hernández and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[17] R. Isele, A. Jentzsch, and C. Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *WebDB*, 2011.

[18] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[19] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, pages 137–146, 2003.

[20] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.

[21] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[22] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[23] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[24] A. N. Ngomo and S. Auer. LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, pages 2312–2317, 2011.

[25] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Beyond 100 million entities: Large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.

[26] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.

[27] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolutionto the next level. *IEEE Trans. Knowl. Data Eng.*, 26(8):1946–1960, 2014.

[28] G. Papadakis, G. Papastefanatos, and T. Palpanas. Boosting the efficiency of large-scale entity resolution with enhanced meta-blocking. *Technical Report*, 2014 (available at: http://www.imis.athena-innovation.gr/en/publications/publication/244).

[29] E. Rahm. Towards large-scale schema and ontology matching. In *Schema Matching and Mapping*, pages 3–27. Springer, 2011.

[30] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.