Information Systems 75 (2018) 75–89

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

BLOSS: Effective meta-blocking with almost no effort

Guilherme dal Bianco[a,*], Marcos André Gonçalves[b], Denio Duarte[a]

[a] Universidade Federal da Fronteira Sul - Campus Chapecó, Brasil
[b] Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brasil

ARTICLE INFO

Article history:
Received 20 September 2017
Revised 8 February 2018
Accepted 14 February 2018
Available online 16 February 2018

Keywords:
Data integration
Deduplication
Blocking
Meta-blocking

ABSTRACT

Record deduplication aims at identifying which records represent the same real-world object in a dataset. As it is a task naturally quadratic (i.e. each record is a potential duplicate), a blocking step is usually used to reduce the computational cost. With blocking, only records inside the same block (cluster) are compared with each other, considerably reducing the search space for finding duplicate records. Traditionally, blocking strategies produce a high degree of redundancy to avoid that some record mistakes (such as typographic errors, attribute inversions, and missing fields) impact the quality of the output. On the other hand, blocking redundancy results in wasted computational cost, especially in large datasets. To alleviate this cost, meta-blocking has been proposed to reduce the number of unnecessary pairs produced by blocking. Meta-blocking approaches rely on a representative set of labeled pairs for training (supervised) or thresholds values (unsupervised). In this work, we propose a new sampling strategy (called BLOSS) that can select a reduced and informative sample of pairs to configure the meta-blocking. BLOSS is divided into three main stages. First, we fragment the set of candidate pairs into levels to alleviate the problem of selecting samples. Second, within these levels, we apply a rule-based active learning to select the most informative non-redundant pairs. However, we observed that the selected non-matching pairs with a high degree of similarity impact negatively on the deduplication process when they are added to the training set. Thus, in the BLOSS's third stage, we propose a strategy to identify and remove such pairs to maximize the number of the matching pairs produced by blocking. This latter stage helps to significantly improve the number of true matching pairs recovered by BLOSS. Our results demonstrate that our approach can reduce the training set size until 39 times compared with the baselines, while improving the precision, in several datasets.

1. Introduction

Record deduplication is the process of identifying which objects are the same in a data repository [1,2], despite potentially different representations. Though an old problem (the first definition was provided by [3] with the first rigorous mathematical treatment on deduplication published in 1969 [4]), the increase in dataset size, dirtiness diversity, and representation complexity have brought new challenges to the research community. For instance, data-centric applications such as digital libraries, data streaming, online stores, among others, all require high-quality data to provide trustful services.

A typical deduplication method, illustrated in Fig. 1 (A), is divided into three main steps [5]: *Blocking*, *Comparison* and *Classification*. The Blocking step aims at reducing the number of potential comparisons by grouping together pairs of records that share common features [6]. Only pairs within the same block shall be compared to each other to find duplicates. This avoids a quadratic generation of pairs, i.e., a situation in which the records are matched all-against-all, infeasible for large datasets. A simplistic blocking approach, for example, puts together all records with the same first letter of the name and surname attributes in the same block. The Comparison step quantifies the degree of similarity between pairs belonging to the same block, by applying some similarity function (e.g., [7]). Finally, the Classification step identifies which pairs are matchings and which are not (aka, non-matchings).

An ideal blocking strategy should include in a block only matching pairs, thus avoiding unnecessary comparisons. Such scenario is, however, unrealistic in most applications due to the differences in patterns of matching pairs found in each dataset. For instance, if the dataset is created using a trusted global `id` (e.g. the Social Security Number), blocking is straightforward. However, if a dataset does not have such a global `id`, that allows to uniquely identify each record, and contains typographical or other types of errors,

* Corresponding author.
E-mail addresses: guilherme.dalbianco@uffs.edu.br (G. dal Bianco), mgoncalv@dcc.ufmg.br (M.A. Gonçalves), duarte@uffs.edu.br (D. Duarte).

https://doi.org/10.1016/j.is.2018.02.005
0306-4379/

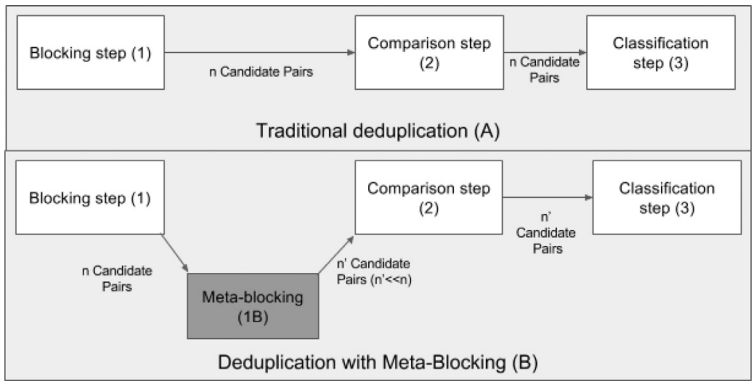

Fig. 1. Overview of traditional deduplication (A) and deduplication with meta-blocking (B).

noise or very heterogeneous representations, blocking becomes a challenging task. Thus the main objective of blocking is to reduce the number of the non-matching pairs within a block to a reasonable number while avoiding as much as possible to miss matching pairs. Ideally, in the classification step, the real non-matching pairs selected for the same block are discarded (i.e., they are indeed classified as non-matching) using global thresholds that can be manually defined [8–11] or learned by using a supervised classification model based on a labeled training set [12].

It is worth noticing that optimizing blocking contrasts significantly with the goals related to improving the classification step. An ideal classification process should maximize the correct assignment of true matching pairs (related to precision) while reducing to the minimum the number of false negatives, i.e., matching pairs wrongly classified (related to recall). On the other hand, the blocking step should maximize recall (avoiding missing true matchings) while, as the second goal, keeping precision as high as possible to avoid unnecessary comparisons within a block. A good tradeoff between both goals is usually hard to achieve, and the focus of the blocking step on achieving high recall may lead to high redundancy, implying in low precision.

In order to address such unique challenges, recent works have proposed an intermediary filtering step, called *meta-blocking* [13–15]. Meta-blocking aims at keeping high recall while improving precision by filtering out unnecessary and redundant comparisons. Fig. 1B shows how meta-blocking (1B) is incorporated into the overall deduplication process: a set of pairs of cardinality n is the input of the meta-blocking phase and the output is a set of (only potentially matching) pairs of cardinality n', such that $n' << n$. In comparison, in the traditional deduplication all n pairs, produced by the blocking, are processed in the classification step, increasing the overhead and the risk of inaccurate classifications.

As in the classification step, meta-blocking strategies can be divided into supervised (i.e., based on classifiers such as Naive Bayes, Decision trees, SVM) or unsupervised (i.e., based on heuristics) to prune the non-matching pairs. For instance, in [13], a supervised meta-blocking strategy is proposed that use a graph structure to connect the pairs produced by the blocking step. The graph is processed to extract features, which are employed by a binary classifier (e.g., SVM, Naive Bayes, Decision Tree) to identify the most promising pairs. This strategy has the drawback of demanding a substantial manual labeling of pairs.

The unsupervised approach proposed in [15] is based on the relation among attributes to produce pairs (similar to schema matching [16]) along with a weight measure between the nodes (using entropy) to prune useless connections. Supervised approaches usually outperform unsupervised ones at the expense of having to label a substantially large set of pairs used for training the filter. Manual labeling is a very costly and cumbersome process that should be exploited with parsimony.

In this article, we propose a new supervised meta-blocking approach, called BLOSS (meta-BLOcking Sampling Selection), whose main goal is to select a reduced and very informative set of pairs to configure the meta-blocking step. To the best of our knowledge, our proposed approach is the first that aims at reducing the user labeling effort in the meta-blocking step. The main challenge is to select informative pairs in the highly unbalanced scenario of deduplication in which most pairs are non-matching in order to remove them from the blocks at large extents.

BLOSS is comprised of three main stages: (1) pre-selection of candidate pairs using a metric that assesses the potential of a pair being a matching; (2) actual selection of candidate pairs for labeling using a state-of-the-art active learning method [17] applied to the pre-selected pairs – it is unfeasible to apply any active learning approach to all unlabeled pairs due to the involved computational costs; and (3) filtering out noisy non-matching pairs labeled in the previous stage. That is, the first stage of BLOSS makes it feasible to apply the active learning solution used in the second stage, besides accelerating its convergence. The focus of the 2*nd* stage is on maximizing the gain of information for the meta-blocking learning process while reducing the user labeling effort. Finally, stage 3 corrects some potential bias of the learning process due to skewness issues (i.e., a dominance of non-matchings in the dataset) and presence of outliers that could harm the effectiveness of the learned model.

Our experiments, using real and synthetic datasets, show that BLOSS is able to reduce the training set size until 39 times when compared to state-of-the-art baselines. It means that the labeling cost is significantly decreased compared to the baselines. Moreover, we observe that final selection of pairs is much more precise (up to 5 times), i.e., we select less non-matching pairs for the blocks, while keeping similar recall levels (around 95%) when compared to unsupervised and supervised state-of-the-art meta-blocking solutions.

This article is structured as follows. Section 2 briefly introduces some background concepts. Our proposed approach is described in Section 3. Section 4 discusses our experimental evaluation and results. Section 5 reviews related work, while Section 6 concludes the paper with final considerations and possible future work.

2. Background

In this section, we present some background necessary to properly understand our proposed approach, as some of the used concepts are recent and non-trivial. As our focus is on reducing the labeling efforts to configure the meta-blocking step, we

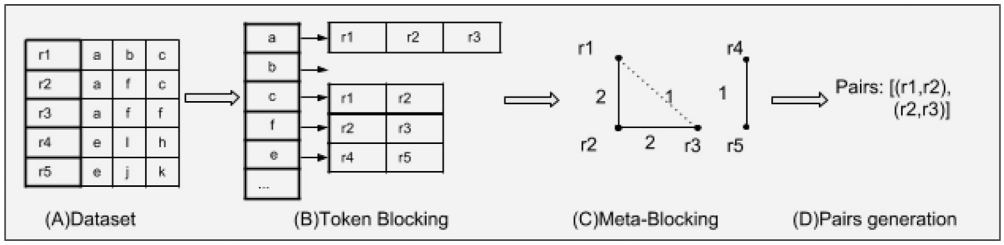

Fig. 2. A toy example of pair generation using a graph structure.

take advantage of a robust strategy (called SSAR) to produce the candidate pairs as we explain next.

2.1. Pair generation and meta-blocking

Traditionally, deduplication requires a pairwise comparison of records. Initially, all records have a potential to be "paired" with all others. Blocking strategies have been proposed to reduce the computational cost of the matching problem, by maximizing the number of matching pairs, while keeping the number of non-matching ones as low as possible. The main challenge is how to identify the correct matching pairs without a quadratic generation of pairs.

A variety of works explores different data structures to produce blocks from the original dataset. For instance, [9,18,19] exploit an inverted index to group together records with a common token[1]. The tokens are used to define blocking keys. All records that share the same key are grouped together. Clearly, such blocking can produce high recall levels at the expense of a very low precision. Large blocks indicate that a very common token key has been used (e.g., the name "John" in a customer dataset). Such large blocks contain a substantial number of pairs with few pieces of evidence supporting a matching.

Meta-blocking has been recently proposed to reconstruct the previously created blocks by eliminating the pairs with a low probability of representing a matching [19]. More formally, meta-blocking receives as input a set of unlabeled pairs of size n, and it outputs a set of unlabeled pairs with size n', where n' is substantially smaller than n. The meta-blocking goal is to reduce the number of candidate pairs as much as possible, but with a focus on high recall for true positive pairs. Note that meta-blocking is orthogonal to any redundancy-positive blocking approach (i.e., more tokens the pairs share, more likely they match [20]).

Two pre-processing steps are usually applied in meta-blocking: block filtering [14] and block purging [21]. Block purging is responsible for dropping off large blocks, produced by highly frequent tokens key (aka stop words[2]). Block filtering is employed to reconstruct the large and redundant blocks. Such blocks are composed of a substantial number of non-matching pairs based on the notion of pair redundancy. The main works on meta-blocking also exploit a graph structure in which each pair is mapped to a graph node, and the edges represent connections between the pairs [15,19].

Fig. 2 presents pictorially how a graph is built. The records (r1, r2, r3, r4, r5) are split into tokens that represent their contents, *i.e.*, *a, b, c, f*, and so on. An inverted index is build based on token key to produce the blocks (2 B). From the blocks, a graph $G = (V, E)$ is built as follows: a node V represents a record and a labeled edge $E = (V', V'', n_t)$ represents the relationship between two nodes (records) V' and V'', where n_t represents the number of tokens shared by the records. Remark that an edge is created only if two records share one or more blocks (*e.g.*, there is no edge between nodes *r4* and *r3* in *G* - Fig. 2C). For instance, in Fig. 2B the tokens *a* and *c* are shared by records *r1* and *r2*, and so there is an edge labeled 2 between both nodes *r1* and *r2* (2 C). The graph structure thus aims to help the meta-blocking step to identify redundancies in the dataset. Edges labeled with small values (*i.e.*, not so many shared tokens) are considered poor and may indicate unpromising pairs. A threshold value (or more complex features) may be used to eliminate the least promising connections, as proposed in some unsupervised meta-blocking approaches [14,15,19]. For example, a simplistic threshold $n_t > 1$ in the example would make the pairs *(r1, r3)* and *(r4, r5)* to be pruned, and the pairs *(r1, r2)* and *(r2, r3)* would remain as shown in Fig. 2D.

The graph structure and blocks can produce valuable information about pair connections. Such information can be useful when mapped to a set of features to identify matchings and non-matchings based on some identified patterns. For instance, the following features (not limited to) can be extracted/inferred from the graph [13]:

1. *Common blocks*: represents the number of common blocks that a pair belongs to – it can be used to infer a pair's relevance. In other words, pairs with a significant number of common blocks have high potential for being a matching. For instance, in Fig. 2 the pairs *(r1, r2)* and *(r4, r5)* share two and one block(s), respectively;
2. *Node degree*: corresponds to the number of non-redundant nodes (records) connected to a specific node. In other words, the number of pairs produced by a record. In Fig. 2, for example, the node *r1* has two non-redundant connections;
3. *Jaccard similarity*: based on the co-occurrence of edges (i.e., tokens in common belonging to a pair) in the graph. In other words, it computes the number of shared blocks pairs over the number of tokens of both records;
4. *Reciprocal aggregate cardinality of common blocks*: computes the sum of the inverse blocks' size shared by a pair – it is based on the idea that 'true' matching pairs belong to smaller blocks;
5. *Co-occurrence Frequency-Inverse Block Frequency (CF_IBF)*: combines the *Common blocks* feature (described above) with the inverse number of blocks containing each record (i.e. similar to IDF commonly used in Information Retrieval [22]). It can be formalized as follows [13]:

$$CF\_IBF(e_{i,j}) = |\, B_{i,j} \,| \cdot log\frac{|\, B \,|}{|\, B_i \,|} \cdot log\frac{|\, B \,|}{|\, B_j \,|}$$

where B is the block collection, $B_i \subseteq B$ represents the blocks containing r_i, $B_j \subseteq B$ represents the blocks containing r_j, $B_{i,j} \subseteq B$ represents the blocks containing both r_i and r_j, $e_{i,j}$ is the edge between r_i and r_j, and $|.|$ denotes the size of the parameter. The idea behind this approach is to identify the most promising pairs. For example, to achieve a high CF_IBF value, a pair must have a large number of edges or tokens in common and the number of blocks that each record belongs to should be small

[1] An example of a *token* in this context is a sub-string of a record attribute value. For example, the record attribute value "John Smith" is split into two tokens "John" and "Smith".

[2] Words such articles and adverbs that do not carry semantic information important for the deduplication task.

compared to the whole number of blocks (i.e., the record feature values must be low frequent in the dataset). CF_IBF is one of the most discriminative feature among the ones described above [13].

These features (and others) can be used to feed a binary classifier (e.g., SVM, Naive Bayes, or Decision trees) to identify which pairs have the potential to be a matching (only defined in the classification step) or can be directly discarded. The binary classifier depends on an informative and representative labeled training set to induce an effective model. Such a process is highly costly since the unlabeled pairs should be selected and manually labeled by the user. We describe in the Section 3, how our proposed approach is used to greatly reduce such effort, while improving effectiveness.

2.2. Active learning

Active learning approaches have been proposed to considerably reduce the number of pairs to be manually labeled when compared to random selection in order to produce competitive effectiveness [23].

Classifier committees have been used in active learning to allow deduplication approaches to identify informative pairs based on divergences among the committee members [24,25]. However, in the initial stages, these approaches still require a minimum training set (which is usually not small) and the definition of some thresholds to allow the classifiers to accurately learn a model. Consequently, they still rely on a considerable effort from the experts. On the other hand, Christen et al. [26] propose an active learning approach based on heuristics to select a limited number of pairs to be labeled by the user. It produces a hyperplane using features values and selects the pair based on distance and distribution. However, its focus is on the classification step (step 2) of the deduplication process, not in meta-blocking. In a meta-blocking scenario, their solution would be highly expensive.

In [17], a rule based active learning, called SSAR, is proposed based on association rules induction to identify non-redundant pairs (*i.e.*, informative pairs) to be added to a training set. By redundant, we mean pairs carrying very similar information. The purpose of SSAR is to select the most informative pairs to be labeled, thus maximizing the training set diversity, while reducing labeling effort.

When compared to the aforementioned active learning approaches (e.g., [24,26,27]), SSAR has several advantages such as: (1) it does not require an initial labeled set, as needed by approaches based on committees; (2) it has a clear stopping criterion, a property that many approaches do not possess; (3) it can select very few but highly informative instances based on an informativeness criteria grounded on lazy association rules; and (4) it does not require threshold values as input to configure the approach. In Section 3.2, we detail SSAR.

3. Proposed approach

In this section, we present our proposed Meta-Blocking Sampling Selection approach (BLOSS). Blocking and meta-blocking focus firstly on the maximization of the number of matching pairs present in a block and, secondly, keeping the number of non-matching pairs as low as possible.

Fig. 3 illustrates the context in which BLOSS is introduced into the whole deduplication process. Meta-blocking (2) receives pairs produced by the Blocking step (1), some which will be filtered out to remove useless comparisons. The surviving pairs are sent to the Comparison step (3). In the Meta-Blocking, the candidate pairs generated by Blocking are mapped to a graph structure to extract information based on neighborhood, common blocks, among other (2A). BLOSS (2B) receives the unlabeled pairs to select the most informative ones to be labeled by the user. A supervised approach (2C) is then trained with the pairs selected by BLOSS to build a filtering model (2D). Such model is applied to the unlabeled set to identify the most promising pairs to be sent to the Comparison (3) and Classification (4) steps.

BLOSS is composed of three main stages:

1. A strategy is employed to group the n candidate pairs into levels. These levels are used as clusters to randomly choose candidate pairs to build a smaller and balanced set N_i. By balanced, we mean a set with a similar number of non-matching and matching pairs without labeling information. At this point, BLOSS does not have the label information. As BLOSS selects a controlled number of pairs in different levels, it is expected that various patterns of dirtiness will be present in the sample, including matching and non-matching pairs.
2. N_i feeds SSAR to eliminate redundant pairs and label the remaining ones. A training set N_j is produced with the labeled pairs.
3. BLOSS analyses N_j to identify "highly similar" non-matching pairs. Then it tries to find *outliers*, i.e., non-matching pairs that deviate from the average similarity of the non-matchings already in training. In other words, it is a non-matching pair that resembles more a matching (i.e., a near false-positive) than the average non-matching pair of the training. These outliers can produce some distortions in the learned model (e.g., by reducing recall). We prune the outliers out, producing N_w that will be used to learn the final filtering model.

In the following, we detail the three stages.

3.1. First stage: definition of similarity levels and random sampling

Since blocking produces a very redundant set of candidate pairs dominated by non-matching pairs, this stage aims at constructing a balanced set of pairs to feed the active learning algorithm. A graph data structure is constructed based on the pairs produced by the blocking step, as explained before. The graph structure allows us to extract the CF_IBF metric (described in Section 2.1) used to group together the candidate pairs into *similarity levels*. A small value of CF_IBF represents a high probability of a pair to be a non-matching (and vice-versa).

Accordingly, the candidate pairs are split into levels with a fixed size organized by thresholds (using the CF_IBF metric value).The idea behind the levels with a fixed size is to avoid that very frequent pairs (non-matching) hide the uncommon pairs (matching). Most discretization strategies aim at producing equilibrate level sets, which is opposed to our goal. The first levels have small threshold values, that is, they probably have more non-matching pairs. The last levels, on the other hand, group more matching pairs (see Fig. 4)[3]. As the metric is not normalized, the number of levels depends on the dataset patterns and size.

Algorithm 1 formalizes the level creation step based on the CF_IBF metric. First, the algorithm selects each graph edge (i.e. pairs) to calculate the CF_IBF value (main loop, lines 3–7). For each edge, the metric value (i.e., $CFIBF\_value/T$) is used to identify the level it belongs to based on the threshold value in which level the pair should be inserted (Line 5). Finally, the pair is added to the correspondent level at Line 6. The algorithm's output is a set of unlabeled pairs grouped into levels.

It would be very impractical to directly apply any active learning technique on these levels due to the high skewness and large

[3] The Figure corresponds to the ground truth in one dataset but it is only illustrative for the sake of our argument. We do not use any ground truth information in our solution.

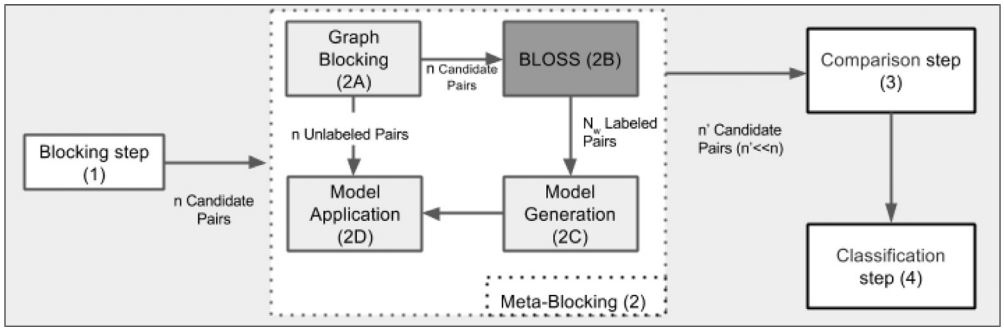

Fig. 3. Overview of BLOSS in a general deduplication process.

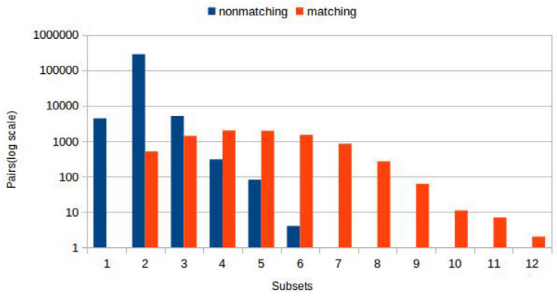

Fig. 4. Interpolation for Dataset 1.

Algorithm 1 Producing levels based on *CF_IBF* metric.

Require: Graph (G) where each edge represents a pair, threshold (T) defines the levels boundary
Ensure: Set of levels $L = L_1, L_2, L_3..L_N$
1: $L \leftarrow \{\}$
2: $level \leftarrow 0$;
3: **for all** $e \in G$ **do**
4: $CFIBF\_value \leftarrow getCFIBF(e)$
5: $level \leftarrow (CFIBF\_value/T)$
6: $L_{level} \leftarrow L_{level} \cup e$
7: **end for**
8: **return** L

size of some of the levels. Fig. 4 illustrates such imbalance in a dataset of 10,000 records[4]. Clearly, identifying the matching pairs inside the lowest levels is very important, as these pairs offer valuable information about matching patterns. Note in the same figure, that most non-matching pairs are grouped, as expected, into the three first levels, allowing the random selection to select different matching patterns in the other levels. In other words, subsets one and two are composed almost completely of non-matching pairs in a dataset, while levels above six contain mostly matching pairs. Levels between two and six are composed of mixed pairs.

To be able to select a sampling with enough diversity, a random selection of candidate pairs is applied inside each level L_i of L (with a defined size, as described in the experimentation section). The idea is that the random selection would be able to select a variety of patterns inside each level. Such set of pairs (N_i) is sent to the active learning stage as described next.

3.2. Second stage: selection of pairs for labeling

Though very important to our approach, to induce diversity, the level-based random sampling still produces a very large set of candidate pairs to be labeled by the user to configure the meta-blocking. To reduce even further the manual labeling effort, SSAR is applied to select only the rarest and most informative pairs present in N_i, as explained in Section 2.2.

Given a training set N_j (which can be initially empty), SSAR selects only non-redundant pairs to be labeled by the user. By non-redundant, we mean an unlabeled pair that, when labeled and included in N_j, brings most information to the process. This pair usually shares few or no feature values at all with instances already in N_j. This means that the rarest and most challenging pairs, which carry valuable information, are included in the training set N_j. In more details, SSAR works as follows: an unlabeled pair u_i is selected to be labeled by the user by using inferences about the number of association rules produced within a projected training set specific for u_i. The projected training set is produced by removing from the current training set instances and features that do not share features values with u_i. When compared to the current training set, the unlabeled pair with fewer classification rules over the projected training set represents the most informative pair, i.e., the one most dissimilar to the current information in the training set. SSAR stops its selection when it has already captured most of the information contained in the unlabeled set.

A simplified example of SSAR is shown in Fig. 5. The pairs are composed of three attributes with values represented by strings. Suppose that in a previous step, pairs $l1$, $l2$, $l3$, and $l4$ have been already labeled. These labeled pairs are then used as filters to build a projected set of unlabeled pairs (in the figure, pairs $u1$, $u2$, and $u3$)

Note that the uncommon features between the labeled pairs and unlabeled ones are removed (see the *Projected Set*). The unlabeled pair $u1$, for instance, produces two projected training pairs since it shares features with the pairs $l3$ and $l4$. The unlabeled pair $u3$ produces three projected training pairs since it shares features with the pairs $l1$, $l3$, and $l4$. On the other hand, the unlabeled pair $u2$ produces a projected training set with only one pair since only $l3$ shares common features with $u2$ (producing a very small number of association rules), meaning that $u2$ is the most informative in the current set of unlabeled pairs, and it should be the next pair to be labeled.

The training set selected by SSAR is then labeled by the user, but not all its elements are used to train the BLOSS model. Before the training phase, we still have to remove some non-matching outliers that can affect the overall process as discussed next.

3.3. Third tage: pruning non-matching outliers

One issue we have to deal with is that SSAR's selection, by design, optimizes criteria that are important for a general classifier, i.e., it maximizes, as much as possible, both criteria at the same time: recall and precision. A balanced focus on precision means that, as a side effect, a meta-blocking filter trained with SSAR's selection may miss a considerable number of matching

[4] The dataset details are presented in Section 4.1.

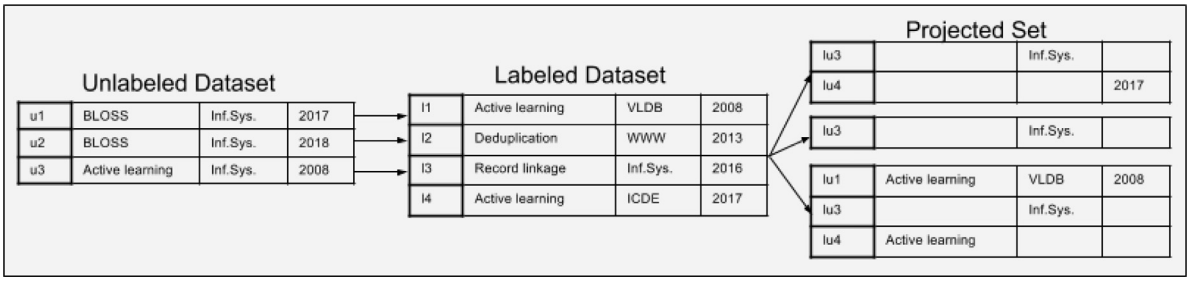

Fig. 5. Example of SSAR in a subset of unlabeled pairs.

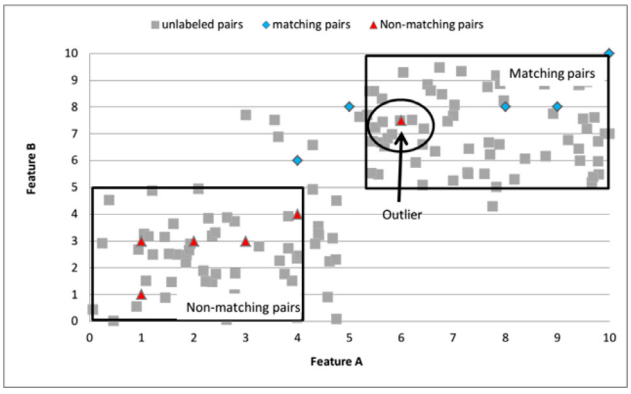

Fig. 6. An example of matching (blue color), non-matching (red color) and unlabeled (gray color) pairs in a two-dimensional space. The non-matching pairs flagged, which represent outliers, when used to training can produce a bias classification model in meta-blocking context. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

pairs while excluding a large portion of non-matching ones. This happens because, when a rare non-matching pair u_j is included in the training set by SSAR, other matching pairs still present in the unlabeled set which are very similar to u_j have their chance to be sent to the classification step decreased.

Fig. 6 illustrates this situation with matching (blue color), non-matching (red color) and unlabeled (gray color) pairs selected by SSAR. The upper rectangle represents a region with matching pairs, and the lower rectangle represents a region with non-matchings. Notice that in the matching region exists a non-matching pair – an outlier according to our approach. If BLOSS is trained with such pairs, the matching pairs that are next to the outlier are probably identified as non-matching ones, as Fig. 6 shows. Depending on the non-matching outliers in the training set, the number of matching pairs recovered may be substantially reduced (reducing recall). However, in the blocking context, it is better to retrieve some non-matching pairs as false-positives than missing a significant group of matching pairs.

Since our goal is to retrieve almost all matching pairs present in the dataset, we have to remove from the training set the non-matching pairs that potentially may impact the blocking quality. Indeed, there are several situations in which it is desirable or even necessary to remove some labeled instances in order to achieve a higher goal, such as improving overall classification effectiveness, by removing noise (e.g., wrongly labeled instances) [28,29] or making adjustments in the underlying data distributions of the samples (under-sampling) to deal with highly skewed distributions [30].

In fact, since BLOSS has the goal of maximizing the retrieval of matching pairs (i.e., recall), the removal of noise (in this case, 'near-positive' non-matching pairs or outliers) is paramount to achieve such goal, mainly in the face of the small training sets produced by SSAR. However, we would like to emphasize that this is done only to improve the meta-blocking step because of its aforementioned particularities, and that all labeled pairs can indeed be exploited in the next deduplication step (i.e., classification). Remark, it is important to notice that we do not want to remove non-matching pairs highly similar among themselves from the meta-blocking, as we will use these representative samples to train "the meta-blocking" in order to better identify other non-matchings in the whole dataset. By removing outliers during meta-blocking, we just want to remove noise in this training, while maximising recall of actual matchings.

Accordingly, after applying SSAR, we use a filter over the training labeled set to remove the non-matching outliers. The filter threshold value is defined by Eq. (1).

$$\Upsilon = \frac{1}{|N|}\sum_{i=0}^{|N|}\theta_i \tag{1}$$

where θ_i represents the similarity value of the pair i and N represents the set of non-matching pairs in the training set.

Therefore, the filter is configured with the average value of similarity between the non-matching pairs. The intuition behind Υ is the following:

1. SSAR selects more non-matching pairs from the lower levels, due to the low CF_IBF value and the natural skewness of the datasets. For example, in Fig. 4, levels 1 and 2 are composed basically of non-matching pairs. As a consequence, more non-matching pairs are selected to be processed by SSAR.
2. When applying Eq. (1), the similarity of those pairs is close to values of the initial levels. For example, consider a training set formed by five non-matching pairs (with the similarity value of 0.2, 0.2, 0.1, 0.2, 0.2, and 0.7). When we apply the Eq. (1), these pairs result in a threshold value of 0.27 and only the pair with similarity value 0.7 will be pruned.

As the training set selected by SSAR can be very small, the filter (Eq. (1)) may produce some distortions. To minimize those distortions, we truncate the value of equation to set it closer to the next highest level. For instance, in the above example, 0.27 is truncated to 0.3. We use the **ceil** function to truncate since if any threshold calculated by Υ is smaller than 0.1, we still have a valid threshold to filter out the outliers, otherwise, if we use a floor function, the threshold would be 0.0, and all non-matching pairs would be removed.

Algorithm 2 presents the implementation of Stage 3. It prunes the outliers pairs from the labeled training set. Initially, each non-matching pair is identified (Line 4) and its similarity value is computed using Jaccard Similarity measuring[5] to be accumulated

[5] The Jaccard similarity measuring is applied here because it produces a normalized value between 0 and 1. We also used CF_IBF in this step and results were qualitatively the same.

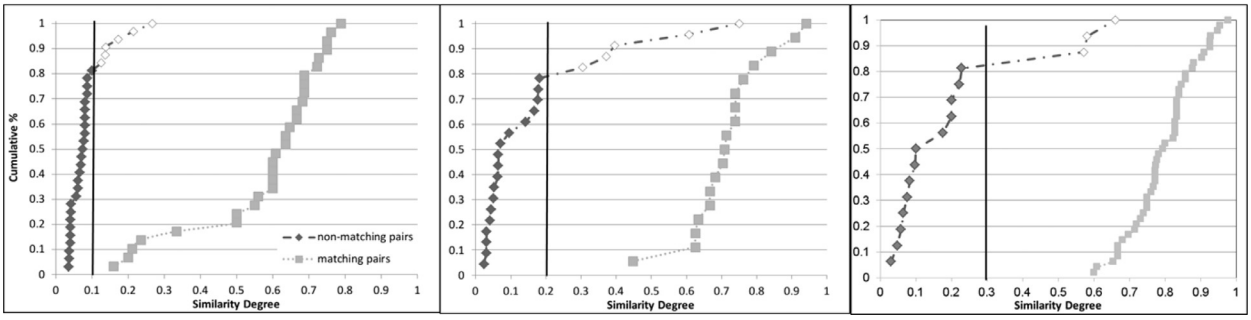

Fig. 7. Cumulative distribution of similarity values in the Synthetic, Scholar, and ACM datasets (as detailed in the Section 4.1).

Algorithm 2 Pruning outliers from the training set.

Require: Labeled training set N_j
Ensure: Set of labeled pairs N_w
1: $sim\_value \leftarrow 0$
2: $pair\_number \leftarrow 0$
3: **for all** $l \in N_j$ **do**
4: **if** $l \in \{non\text{-}matching\}$ **then**
5: $sim\_value \leftarrow sim\_value + getSimJaccard(l)$
6: $pair\_number \leftarrow pair\_number + 1$
7: **end if**
8: **end for**
9: $\Upsilon \leftarrow \lceil (sim\_value / pair\_number) \rceil$
10: $N_w \leftarrow N_j$
11: **for all** $l \in N_w$ **do**
12: **if** $l \in \{non\text{-}matching\} \wedge simJaccard(l) > \Upsilon$ **then**
13: $N_w \leftarrow N_w - l$
14: **end if**
15: **end for**
16: **return** N_w

(Lines 5 and 6). After all labeled pairs are processed, the average similarity between the non-matching pairs (Line 9) is computed. In the end, the non-matching pairs with similarity above the rounded up average are pruned from the labeled set (Lines 12 to 14). The output is a set of labeled pair minus the outliers.

Eq. (1) captures an interesting pattern in the similarity of outliers: they are usually above the average similarity of all non-matching pairs. We ran a simple experiment based on the first and second stages of BLOSS to characterize the similarity among pairs in Synthetic, Scholar, and DBLP datasets (details about the datasets are presented in the next Section). We calculate the cumulative distributions of matching and non-matching pairs and we plot them in Fig. 7. The vertical black line in each graph represents the threshold value defined by the Eq. (1). The matching and non-matching curves have a clear distribution, as can be seen. The non-matching curve is much more skewed than the matching one in all datasets, except in the final region in which non-matching points have a smooth inclination. Such region represents the non-matching pairs, with similarity degree at the right of the vertical line that, in our context, represent the outliers. Thus, non-matching pairs with a curve inclination similar to the matching ones, are the outliers successfully identified by our strategy.

We apply a skewness test in this same set of similarities of non-matching pairs to confirm our hypothesis of how much skewed they are, that is, the presence of outliers in the similarities. Skewness is a measure of the asymmetry or, more precisely, the lack of symmetry, of the probability distribution of a real-valued random variable about its mean. Given χ random variables,

Table 1
Skewness test results.

Dataset	Skewness	Skew type	High
Synthetic	1.86	Positive	True
Scholar	1.75	Positive	True
ACM	1.28	Positive	True

Eq. (2) shows the mathematical definition of skewness [31].

$$\frac{\sum_{i=1}^{|\chi|}(x_i - \tilde{x})^3 / |\chi|}{s^3} \tag{2}$$

where $\tilde{x}$ is the average of χ, $x_i \in \chi$, and s is the standard deviation of χ. Table 1 shows the results of the test. The second column (`skewness`) presents the skewness of the datasets that represent the magnitude of the asymmetry of the datasets. When the skewness is close to 0.0, the dataset tends to have a normal distribution. Column *Skew type* shows the possibilities: symmetrical ($\texttt{skewness} \cong 0$ - no skewness), positive ($\texttt{skewness} \gg 0$ - positive asymmetry), and negative ($\texttt{skewness} \ll 0$ - negative asymmetry). All datasets have a positive skewness – the curve distribution tilt to the right. Based on skewness value, the dataset can be considered to be highly ($\texttt{skewness} > 1$), moderate ($0.5 < \texttt{skewness} \leq 1$), or fairly ($0 \leq \texttt{skewness} \leq 0.5$) skewed. The last column (*High*) shows the level of skewness of the dataset distribution. Note that all datasets have a high skewness score, showing that there are indeed non-matching pairs with "outlier-like" values of similarity. These non-matching pairs should be pruned to optimize the specific goals of the meta-blocking step.

To summarize, our proposed approach aims at improving the overall meta-blocking phase by reducing the user effort in labeling pairs and the number of pairs to be classified by a supervised approach. It is accomplished by ranking the pairs into levels, randomly selecting some pairs from each level, applying an active learning approach over the selected pairs, and removing outliers from the resulting labeled set N_j. The filtered set N_w is then used to configure the BLOSS classifier to select the most promising pairs for the next deduplication step.

4. Experimental evaluation

This section outlines our experimental evaluation that demonstrates the effectiveness, run-time of our approach, and the reduction in the manual labeling effort of BLOSS. We should stress that our meta-blocking proposal can be used along with any redundancy-positive blocking strategy. We only evaluate the meta-blocking step since the classification step is out of the scope of this work.

Table 2
Datasets characteristics.

Dataset	Records	Matching pair	Cartesian product
SYNT10K	10,000	8,615	100×10^6
SYNT50K	50,000	42,668	25×10^8
DBLP–Scholar	2, 616 64,263	2,308	168.1×10^6
DBLP–ACM	2,616 2,294	2,224	6×10^6

4.1. Datasets

We use both synthetic and real datasets to evaluate our approach, as detailed in Table 2. The synthetic datasets were created by [19,32][6] using Febrl [33]. This generator works by first creating the original records based on real-world vocabularies (e.g. names, surnames, street name, among others). Afterwards, the original records are randomly changed (e.g. merging words, inserting character mistakes, and deleting attributes) to create the matching pairs. Errors are based on noise patterns found in real data. The datasets' size is defined with 10,000 (SINT10K) and 50,000 (SINT50K) records, respectively.

The real datasets are created by merging two different real datasets in the same domain to produce a deduplication scenario (as Table 2 details). Such deduplication datasets were previously used in [19,32,34] and were compiled by [19,32]. The first dataset was created by merging the bibliographic dataset DBLP[7] and queries submitted to Scholar[8], (SCHOLAR for short). The SCHOLAR dataset is composed of 2,616 × 64,263 records, resulting in 168 million candidate pairs and only 2,308 of them represent matching pairs. The second dataset was produced by merging DBLP and the ACM[9] digital library (ACM for short). ACM is composed of 2616 × 2,294 records. The product Cartesian of this dataset results in 6 million candidate pairs, with only 2,224 matching ones. In both datasets the matching pairs were manually identified.

4.2. Evaluation metrics

The blocking quality is usually assessed by the following standard measurements: *Pair Quality (PQ), Pair Completeness (PC)* and F_β*-measure* [5,15,19,35]. The metrics described next are based on the number of true positives (TP), true negatives (TN), and false positives (FP) edges in the graph (or candidate pairs). PQ (Eq. (3)) evaluates the rate of pairs that have been correctly identified (TP) compared with the whole number of pairs recovered (TP+FP). This is equivalent to *precision*. PC (Eq. (4)) measures the rate of correct pairs (TP) compared with the matching pairs present in the dataset (TP+FN), a metric similar to *recall*.

Finally, F_β (Eq. (5)) combines PQ and PC values into a single value. The β parameter controls the relative importance of PC and PQ. With a β value of two, PC is two times more important than PQ. We define exactly this value ($\beta = 2$, i.e. F2) as we consider that, for the meta-blocking scenario, PC is more important than PQ[10]. Classifying true positives (i.e., real matchings) is usually the most important goal of linkage methods. Thus, an approach that avoids missing them in meta-blocking at the expense of increasing a bit the number of false positives is worth it in our view.

Note also that we exploit the F2 measure mostly to evaluate the outliers detection capability. In the rest of experiments, we concentrate on commonly used measures such as PQ and PC, using them separately to compare BLOSS with the baselines[11].

$$PQ = \frac{TP}{TP + FP} \tag{3}$$

$$PC = \frac{TP}{TP + FN} \tag{4}$$

$$F_\beta = (1 + \beta^2)\frac{PC * PQ}{\beta^2 * PQ + PC} \tag{5}$$

We ran each experiment 10 times, reporting the average rates and standard deviation. The results in all datasets were compared employing statistics significance tests (paired t-test) with a 95% confidence interval.

4.3. Experimental setup

We consider two state-of-the-art baselines: (1) a supervised meta-blocking approach, referred as SMB [13][12]; and (2) an unsupervised meta-blocking approach proposed in [15][13], called BLAST.

BLOSS is implemented on top of an open source framework presented in [13] using *Java* 8. The baseline supervised meta blocking (SMB)[13] was implemented in this framework. The other baseline BLAST [15] also takes advantage of such framework to implement its approach. The experiments have been performed using Ubuntu 14.04, with 128 GBytes of RAM, Intel Xeon E7-4850 (2.00 GHz) with four processors, and Storage PowerVault MD with 12 TB.

BLOSS uses the same features proposed by SMB (see Section 2). We configure SMB with the binary classifier Naive Bayes, as it achieves competitive results when compared to the other baselines, even when using the default parameters, meaning that no expensive parameterization is required.

As previously explained, the use of CF_IBF metric naturally separates the pairs into similarity levels. As this metric produces non-normalized values, we use a pre-defined threshold value to define the boundaries of the ranking. We empirically identify that a threshold value of 30 could be used in all datasets. For instance, the first level is composed of pairs with a feature value between 0–29, the second with values between 30–59, and so on. Clearly, the number of levels $L = L_1, L_2, L_3..L_N$ produced by such threshold changes based on the dataset size. This is important to produce a dynamic ranking size[14].

All analyzed approaches use token blocking to produce the initial pairs to feed the meta-blocking, as suggested in [14,21]. In fact, any redundancy-positive blocking approach could be used, since meta-blocking demands only a set of candidate pairs as input. The only actual requirement is to keep high levels of recall. Moreover, we apply block purging and block filtering [14,21,32] to reduce the number of candidate pairs. We define the block purge filtering with value 1.005 and blocking filtering with value 0.8, as defined in the original works [19,32]. The threshold 0.8 for purging is used to eliminate 20% oversized blocks. Block filtering is used to eliminate less important pairs belong to the oversized blocks, such number is experimentally identified. As we used the same datasets exploited by our baselines, we exploited the same parameters that optimized their performance on those datasets. This also allows an

[6] All datasets are available at https://sourceforge.net/projects/erframework/files/.
[7] http://www.dblp.org.
[8] http://scholar.google.br.
[9] http://db.acm.org.
[10] We have tested with other values for β and the results remained qualitative the same. For space reasons, we chose to not show them in the paper and to stay with $\beta = 2$, since it is easier to interpret.
[11] Notice that in here we are not comparing different linkage methods, we are only evaluating the effectiveness of the meta-blocking step.
[12] SMB source code is available at github.com/vefthym/ParallelBlocking.
[13] BLAST source code is available at github.com/Stravanni/blast.
[14] We should stress that we have run some experiments with some discretization strategies (e.g., TUBE [36], equal distance, etc) and none of them produced better results than the identified threshold.

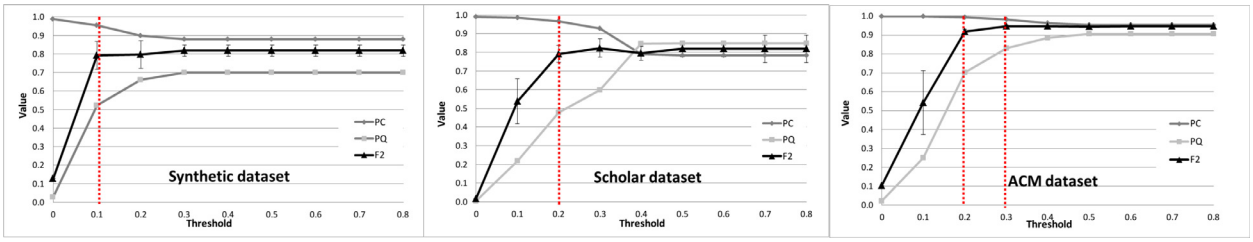

Fig. 8. PC and PQ values when the thresholds is ranged in SINT10K, SCHOLAR and ACM datasets using the F2 metrics.

easy comparison and replication of experiments, making it a valid benchmarking for meta-blocking.

4.4. Identifying the filtering threshold value

In these first experiments, we analyze the strategy used to define the threshold filtering value (Eq. (1)). This threshold (Υ) is used to eliminate outliers from the training set. We aim at demonstrating that our strategy to identify the threshold value is effective. We defined a configuration of 50 pairs for every L_i of L (i.e. the number of pairs randomly select inside each level in Stage 1) – other configurations led to similar results as we detail in Section 4.6.

To conduct these experiments, we considered the SYNT10K, ACM, and SCHOLAR datasets. We also use the F2 measure to analyses the trade-off between PC and PQ. Remind that our focus here is on analyzing the behavior of the meta-blocking step.

Fig. 8 presents the behavior of our method with regard to PC, PQ, and F2 when the threshold value varies from 0.0 to 0.8. We first note that when the outlier threshold value is set to 0.0, all non-matching pairs are removed from the training set and the meta-blocking recovers almost all matching pairs present in the dataset (PC value close to maximum) while the PQ value is near to zero. As we use Naïve Bayes, in this particular scenario (similarity 0.0), we only have the positive training instances as all negatives ones are already removed by the threshold. Note that this is not the ideal scenario, as we also want to increase PC by having negative instances for training. This scenario is only illustrative of the behavior of the method. Note also that even when the threshold value is 0.0, we do not have an optimal output of matching pairs due to the previous step (i.e. token blocking) that removes some matching pairs.

Analyzing now the proposed heuristics to define the threshold, in the SYNT10K dataset, BLOSS identifies (in most cases) a threshold value of 0.1. When the threshold assumes this value, PC reduces to 96% while PQ achieves a value of 53%. Interestingly, when the threshold value is set to 0.2, PC reduces to 90% but the PQ is statically equivalent to using the threshold of 0.1. Both thresholds, 0.1 and 0.2, are suitable values to reduce the non-matching pairs in the training set, but the first value achieves a higher PC with a similar PQ. This is better seen with the F2 metric, with a stable value achieved after the threshold= 0.1. Higher thresholds values (i.e. 0.3, 0.4, ...,0.8) produce stable PC and PQ values around 88% and 70%. This means that few (or no) outlier pairs are eliminated from the training set with such (high) threshold values.

In the SCHOLAR dataset, BLOSS identifies the threshold value 0.2 as default (in most cases). Such threshold achieves PC and PQ values of 97% and 49%, respectively. It represents an improvement in PQ of 55% in comparison with the threshold value of 0.1 along with an equivalent PC value. The F2 metrics shows that an almost optimum value is achieved with the default threshold. The threshold value of 0.3 reduces the PC value by almost four percentual points while PQ is statistically equivalent to the default value. The stabilization point occurs around the threshold value of 0.5. Clearly, our approach identifies successfully a threshold value close to the ideal to maximize the PC value, while identifying an appropriate PQ value.

In the ACM dataset, BLOSS is configured with thresholds values that vary between 0.2 and 0.3 (as the threshold value is defined by the average of the similarity values). Interestingly, in this dataset, until threshold= 0.2 the PC value remains fixed at 100%, suffering a smooth reduction of 4% until threshold= 0.4. On the other hand, PQ increases until threshold=0.4, up to the value of 87%.

To sum up, our experiments show the ability of BLOSS to identify appropriates threshold values that are able to eliminate outliers from the training set.

4.5. BLOSS-TH vs BLOSS-WTH

In these experiments, we further analyze the strategy used to eliminate the outliers from the training set. BLOSS with a threshold (called BLOSS-TH) is compared with BLOSS without a threshold (called BLOSS-WTH) in real (ACM and SCHOLAR) and synthetic (SINT10K) datasets. We experimentally show that BLOSS-TH is an effective way to produce a training set that maximizes the number of matching pairs (*recall*) in comparison with BLOSS-WTH. As in the previous experiments, we configure the level size with 50 pairs as default. This value shows to be the best balance between PC and PQ in all analyzed datasets (detailed experiments in Section 4.6) from every L_i of L. However, other level sizes could have been used as well, as a valid threshold can be obtained independently of the particular choice.

Fig. 9 shows the behavior of the PQ and PC metrics with BLOSS-TH and BLOSS-WTH in the datasets. Furthermore, Fig. 10 presents the number of matching, non-matching, and outliers in each dataset.

As we can see in SINT10K, BLOSS-TH has a PC and PQ values of 96% and 53%, respectively, while BLOSS-WTH achieves a PC and PQ values of 88% and 70%. In other words, BLOSS-TH retrieves 8% more matching pairs with an increase of 32% in the number of non-matching pairs. The higher the PC value, the better, since the classification step is able to remove non-matching pairs produced by blocking, but it is unable to increase the number of matching pairs not included in the blocks. BLOSS-TH achieves this PC value by eliminating from the training set only six non-matching pairs (see Fig. 10), of a whole of 59 labeled pairs in SINT10K.

BLOSS-TH achieves a similar PC value of 97% and 99%, while BLOSS-WTH has a PC value of 75% and 69% in SCHOLAR and ACM, respectively. Since the training set is only composed of 45 (SCHOLAR dataset) and 58 (ACM dataset) pairs (see Fig. 10), BLOSS-TH removes only seven and four non-matching pairs from each training set, while improving the number of selected matching pairs by 29% and 38% when compared to BLOSS-WTH. These relevant PC values of BLOSS-TH are followed by a reduction in PQ of 42% and 5% in PQ, respectively. Interestingly, BLOSS achieves a higher PQ value in ACM. We believe that this happens due to less complex dirtiness patterns present in this dataset.

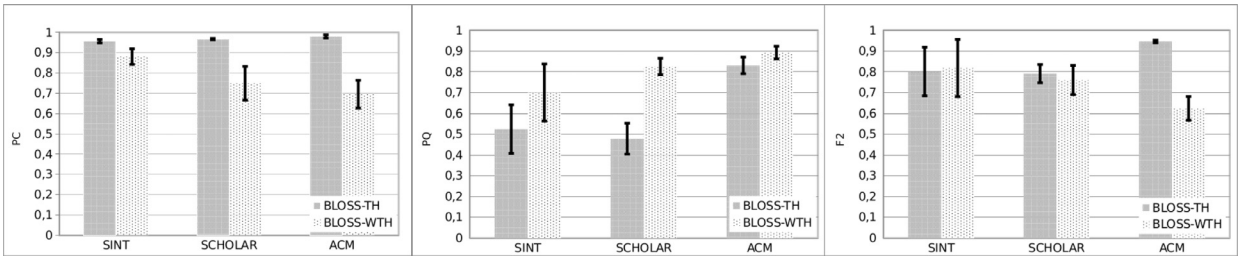

Fig. 9. Comparing PC, PQ and F2 values of BLOSS-TH (using threshold) with BLOSS-WTH(without threshold) in real and synthetic datasets.

Table 3
Comparison SMB and BLOSS in synthetic datasets.

	SINT10K dataset						SINT50K dataset				
	SS	PC	PQ	PL	LR		SS	PC	PQ	PL	LR
SMB		0.95+− (0.00)	0.46+− (0.05)	871		SMB		0.96+− (0.00)	0.20+− (0.01)	2179	
BLOSS	10	0.95+− (0.03)	0.63+− (0.21)	28	31.1	BLOSS	10	0.95+− (0.02)	0.63+− (0.21)	23	94.7
	50	**0.96+−(0.01)**	**0.53+− (0.11)**	**59**	**14.7**		**50**	**0.95+− (0.01)**	**0.53+−(0.11)**	**55**	**39.5**
	100	0.96+− (0.01)	0.53+− (0.16)	73	11.9		100	0.95+− (0.01)	0.53+− (0.16)	67	32.7
	500	0.95+− (0.02)	0.61+− (0.20)	83	10.5		500	0.93+− (0.03)	0.61+− (0.20)	88	24.7
	1.000	0.95+− (0.02)	0.53+− (0.23)	105	8.3		1.000	0.95+− (0.02)	0.53+− (0.23)	102	21.4

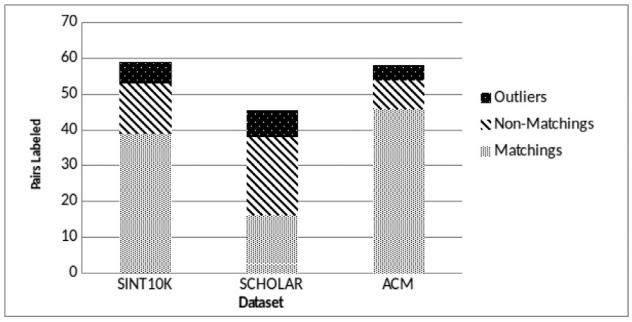

Fig. 10. Non-matching, matching, and outliers pairs selected to configure BLOSS.

Overall, we claim that our strategy properly eliminates some non-matching pairs to achieve higher PC values. This happens with the drawback of selecting some non-matching pairs, which can be eliminated in the classification step.

4.6. Comparison with the baselines: BLOSS versus SMB

In this section, we analyze BLOSS by comparing the reduction in labeling effort and the effectiveness with a stronger baseline: SMB [13]. As we explain in Section 2, SMB prunes unnecessary edges using supervised algorithms such as SVM or Naïve Bayes. SMB creates the training set by randomly selecting 5% of matching pairs and the same number of non-matching pairs. Note that the user should know beforehand the complete number of matchings to define the training set proportion, an unrealistic scenario. Moreover, in a real setting, a very large number of pairs should be labeled to find the necessary number of matchings due to the highly skewed scenario. This means that the actual labeling effort would be much higher.

Tables 3 (for the synthetic datasets) and 4 (for the real datasets) provide the detailed results. In these tables, the second column, named *SS (Sample Selection)*, specifies the level sizes: 10, 50, 100, 500, and 1,000 pairs. As previously described, the level size defines the number of pairs collected at each level. The PC and PQ metrics with their standard deviation values are shown in the respective columns. The average number of labeled pairs is shown in "PL" column. Finally, the "LR" column reports the reduction rate of BLOSS compared with the baseline (i.e., training set size of BLOSS over the training set size of SMB).

As we can see in Table 3, SMB requires labeling 871 pairs to achieve a PC of 95% in SYNT10K. With a level size of 10 pairs, BLOSS achieves a statistically tie PC value and enhances the PQ by around 37%, i.e., it retrieves less non-matching pairs than SMB, demanding only 28 pairs to be labeled by the user. This corresponds to a reduction of 31 times in the training set size. When we configure BLOSS with a level size of 50 pairs, the training set increases to 59 pairs, PC has a marginal statistically improvement of 1%, and a PQ value statistically tie. Compared to the baseline, with a level size of 50 pairs, BLOSS decreases the manual effort in around 14 times, achieving statistically equivalent PQ and a marginal improvement in PC value. When the level size increases to 1,000 pairs, it achieves statistically equivalent PC and PQ to the others levels size, i.e., BLOSS have no improvements with an increase in the training set. Note that even using 1,000 pairs as level size, the reduction in the training set size remains substantial (around nine times).

Interesting results are also obtained in SYNT50k. BLOSS achieves a PQ improvement of almost three times in the lower level size (10 pairs) and PC value statistically tie than SMB. When the level size increases to 50 pairs, PC and PQ remain statistically equivalent to the level size of 10 pairs but PQ achieves a more stable value. With the level size of 500 pairs, PC value drops a bit (around 2%), but the PQ value increases by 15% than the level size of 50 pairs. It means that such training set allows removing more non-matching pairs than the preview level size with the drawback of missing a few matching pairs.

Results in the real datasets are shown in Table 4. In the SCHOLAR, with a level size of 10 pairs PC values is 3% lower but PQ value is around 13 times higher than SMB's, requiring around 13 times less labeled pairs. When the level size increases to 50 pairs, PQ continues substantially higher, and PC has an improvement of 2% with 45 labeled pairs (reduction of almost five times). With a level size of 1,000 pairs, PC has a reduction in 2% and PQ improvements in 40% but with 34 more labeled pairs compared with the level size of 50 pairs. It means that the highest level size (1,000 pairs) recovery less non-matching pairs but it misses some matching pairs. The manual effort increases, but continues to be substantially lower than SMB's.

In ACM, PC value is close to the ideal with both, BLOSS (all level sizes) and SMB. This probably happened since this dataset is cleaner, with clear matching patterns. The PQ value in first level size (10 pairs) is substantially higher than SMB's (almost two

Table 4
Comparison of SMB with BLOSS in real datasets.

	SCHOLAR dataset					ACM dataset					
	SS	PC	PQ	PL	LR		SS	PC	PQ	PL	LR
SMB	–	0.95+– (0.01)	0.09+– (0.02)	222	–	SMB	–	0.96+– (0.02)	0.39+– (0.09)	229	
BLOSS	10	0.92+– (0.13)	0.51+– (0.23)	17	13	BLOSS	10	0.98+– (0.01)	0.79+– (0.13)	27	8.3
	50	**0.97+–(0.00)**	**0.49+– (0.05)**	**45**	**4.9**		**50**	**0.99+–(0.01)**	**0.75+– (0.09)**	**58**	**3.9**
	100	0.97+– (0.01)	0.58+– (0.10)	51	4.4		100	0.99+– (0.02)	0.75+– (0.11)	72	3.2
	500	0.97+– (0.01)	0.58+– (0.17)	73	3.0		500	0.98+– (0.01)	0.81+– (0.17)	90	2.5
	1.000	0.95+– (0.01)	0.70+– (0.03)	79	2.8		1.000	1.00+– (0.00)	0.65+– (0.05)	108	2.1

Table 5
Comparison of BLOSS and BLAST in the synthetic datasets.

	SYNT10K			SYNT50K		
	PC	PQ	LR	PC	PQ	LR
BLAST	0.83+– (0.00)	0.21+– (0.00)	–	0.83+– (0.00)	0.07+– (0.00)	–
BLOSS	0.96+– (0.01)	0.53+– (0.11)	59	0.95+– (0.01)	0.53+– (0.11)	45

Table 6
Comparison of BLOSS and BLAST in the real datasets.

	SCHOLAR			ACM		
	PC	PQ	LR	PC	PQ	LR
BLAST	0.95+– (0.00)	0.05+– (0.00)	–	0.99+– (0.00)	0.60+–(0.00)	–
BLOSS	0.97+– (0.00)	0.49+– (0.05)	45	0.99+– (0.01)	0.75+– (0.09)	58

times), being PC also slightly higher: 2%. It occurs with a reduction of eight times in the training set size. When the level size increases to 50 pairs, PQ becomes more stable with equivalent PC values. Finally, with a level size of 1,000, PC achieves an ideal value (4% higher than SMB) with a PQ 60% higher than SMB's. It comes with a reduction in 47% in the labeling effort when compared with SMB.

In all datasets, BLOSS achieves the best tradeoff between effectiveness and labeling effort with a level size of 50. This value is our default recommendation to use the method. In summary, the proposed sampling selection strategy used by BLOSS is able to produce a reduced training set with the most informative pairs when compared with the random selection used by SMB.

4.7. Comparison with the baselines: BLOSS vs BLAST

In this section, we compare BLOSS with the unsupervised BLAST [15]. BLAST builds upon [19] by improving precision based on a clever blocking key selection (similar to schema matching), along with a edges' weight measure (using entropy) to prune useless connections.

We report in Tables 5 and 6 the results with the real and synthetic datasets. We only report the BLOSS results with a level size of 50 pairs, as it achieves the best effectiveness (see Section 4.6). In the synthetic datasets, BLOSS achieves a substantial higher PC and PQ values. These results are partially justified because BLAST employs a strategy based on attribute matching, which is not useful when only one dataset is considered, as is the case of the synthetic datasets.

Regarding the real datasets, in SCHOLAR, BLOSS achieves an equivalent PC value (slight improvement of 2%), while improving PQ by almost ten times. In ACM, BLOSS achieves competitive PC values, improving PQ by almost 25%. In other words, BLOSS is able to retrieve more matching pairs in these datasets with better precision than BLAST. It can achieve those results with less than 60 labeled pairs.

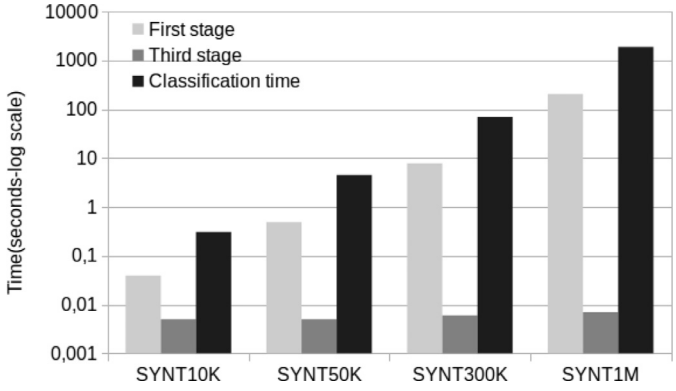

Fig. 11. Processor time-consuming of BLOSS for the first stage, third stage, and classification (in seconds and log scale).

4.8. Run-time of BLOSS stages

To show the potential of our approach regarding its time consumption, as well as its capacity on dealing with large datasets where imbalance becomes even a worse problem, we present next analyses of the time consumption of individual BLOSS steps. To better illustrate this issue, we added two large additional datasets: SYNT300K (300,000 records) and SYNT1M (1,000,000 records). Table 8 summarizes all dataset characteristics used in this section.

The graph in Fig. 11 shows the execution time of BLOSS in the four synthetic datasets, splitting into three bars. The first bar measures the time to define the similarity levels and random sampling (i.e., BLOSS first stage). The second bar considers the time to prune the non-matching outliers (i.e., BLOSS third stage). The last bar measures the classification time (as BLOSS is a supervised approach). We decided not to include the second stage of BLOSS since the active learning is an iterative process that demands user feedback. Thus, the time will be biased by the user interaction.

Table 7
The execution time of SSAR to select each pair.

Dataset	Time(s)
SYNT10K	0.10
SYNT50K	0.12
SYNT300K	0.16
SYNT1M	0.33

Table 8
Dataset characteristics, number of comparisons, and resolution time when is applied a token based blocking and block purging (without meta-blocking).

Dataset	Records	Duplicates	PC	Comparisons	RT(s)
SYNT10K	10,000	8,615	0.99	3.35×10^5	3.6
SYNT50K	50,000	42,663	0.99	7.42×10^6	71
SYNT300K	300,000	254,686	0.99	2.70×10^8	2,550
SYNT1M	1,000,000	849,276	0.99	2.94×10^9	28,800

However, we present in Table 7 the active learning cost based on the time to select each pair. The time presented in that table does not consider the time spent during the user labeling process. The intuition behind this experiment is to show how long takes to SSAR to select a pair to be sent to the user labels it. Remind that the time grows slowly based on the size of the datasets, that is, while the processing time increases three times, the size of the datasets increases one hundred times.

As we can see in Fig. 11, the relation among the first, and third bars of the total time-consuming is approximately 10%, and 90% in all datasets, respectively. The second bar represents almost constant time, since that the number of pairs selected by SSAR is substantially low even in large datasets. BLOSS first stage represents comparable high time since it demands to perform a large number of candidate pairs to produce the levels (used to feed the active learning process). Clearly, BLOSS spends a significant fraction of its time in the classification phase as the pairs need to be processed. As some classification approaches are highly parallelizable, we believe that such overall time can be largely reduced using more sophisticated classification strategies [37]. However, this is not the focus of our manuscript, and we leave this for future work.

4.9. Run-time and effectiveness of BLOSS vs baselines

Next, we compare BLOSS run-time with those of the baselines SMB and BLAST and the respective tradeoffs between performance and effectiveness. Here, we evaluate three additional metrics: Relative Resolution Time (RRT), ΔPC, ΔPQ. RRT computes the token blocking time over the meta-blocking time; i.e., it evaluates the time reduction when meta-blocking is applied [13]. The closer RRT is to zero, the better. ΔPC computes the variation in PC comparing the PC of meta-blocking approach minus the PC of token Blocking (almost the ideal value) [13]. That is, ΔPC identifies the quality loss when applying meta-blocking. The closer ΔPC is to zero, the better. ΔPQ measures the number of pairs produced by token blocking over the number of pairs produced by meta-blocking. It indicates the relative reduction in the number of pairs when meta-blocking is applied. The bigger value of ΔPQ, the better. Here, we also add two large datasets SYNT300K and SYNT1M (detailed in the Table 8) to illustrate the scalability.

To compute the metrics RRT, ΔPC, and ΔPQ the number of comparisons and processing time of a "pure" blocking approach (without meta-blocking) is required. To accomplish that we apply a block strategy based on Token Blocking, Block Purging, and Jaccard similarity measuring, as used in [13]. Table 8 shows dataset characteristics, number of comparison, and resolution time. Notice that, without meta-blocking, a huge number of comparisons has to be performed. For example, in the SYNT1M dataset, it can reach up to 2.94×10^9 comparisons. Although the PC is very high (around 99%), the processing time is also very high (around 8 hours).

Fig. 12 illustrates the RRT, ΔPC, ΔPQ metrics of BLOSS, SMB, and BLAST. As we can observe, all approaches have a substantial reduction in RRT ranging from 1/5 until 1/16 compared with the token blocking time. It illustrates the viability of meta-blocking in the context, especially in large datasets. More specifically, when BLOSS is contrasted with BLAST, we can see that BLAST is faster (around two times) and it reduces by almost two times the ΔPQ value, particularly in large datasets. However, this happens with a substantial comparative drawback in the quality of the pairs, i.e., BLOSS has ΔPC value around three times better than BLAST in SINTY1M. It means that BLAST is faster but missed a substantial number of matching pairs. As the meta-blocking focus on reducing the number of candidate pairs without major quality impacts mainly for matchings pairs, we can say that it provides a solution with a better tradeoff.

Comparing BLOSS with SMB, the RRT results are very similar (increasing the run-time in 10% because of BLOSS first step, showed in Section 4.8) in both approaches, as they constitute supervised strategies. Note that, the number of pairs used by BLOSS in the largest dataset (SYNT1M) is around 65; SMB takes around 84,000 pairs be labeled. Despite this substantial difference, BLOSS has a very competitive ΔPC and RRT, but with a substantial reduction in ΔPQ, i.e., BLOSS is able to reduce the number of non-matching pairs around three times in the dataset SYNT1M. Such reduction is reflected in the other datasets, as the other figures show.

To summarize, the overall results of the experiments have shown that BLOSS is a competitive approach for meta-blocking in terms of effectiveness and scalability. BLOSS is able to select a reduced number of pairs to be labeled by the user with very competitive results regarding ΔPC and substantial improvements in ΔPQ values (reduced number of candidate pairs produced in all datasets). The processing time of BLOSS is comparable to SMB, but with a better pruning quality, i.e., BLOSS produce less non-matching pairs with a massively reduced training set. BLAST showed to be a bit more efficient but with substantial losses in effectiveness when compared to BLOSS. Overall, we believe that, when compared to these two state-of-the-art baselines, BLOSS achieves the best tradeoffs.

5. Related work

A naive deduplication process has a quadratic cost. Blocking has been applied to scale this process to large datasets to avoid the cartesian product [5,6,38]. Even so, blocking can represent 70% of the computational costs of whole deduplication process [2]. In this context, complex blocking approaches have been proposed to address such computational cost. For example, [10,39] explore the record token position to filter out unnecessary pairs reducing the computational costs. On the other hand, [40,41] propose an approach to speed up the process exploring the parallelism of many-cores GPUs. Meta-blocking [13,15,19], which is orthogonal to any blocking approach, aims at contributing to improve the blocking effectiveness by reducing the number of false positive pairs, and consequently the blocking step time complexity.

Meta-blocking. In [14,19] an unsupervised meta-blocking approach is proposed that takes advantage of a graph structure, in which the edges between two records specify the number of common blocks. A weight threshold or a top-k strategy is employed to preserve the most promising edges and filter out the remaining ones. Recently, BLAST [15] built upon this previous work [14] by using attribute statistics to group together only tokens of related attributes (similar to schema matching) to enhance the pair's selection. BLAST uses entropy to identify the most informative

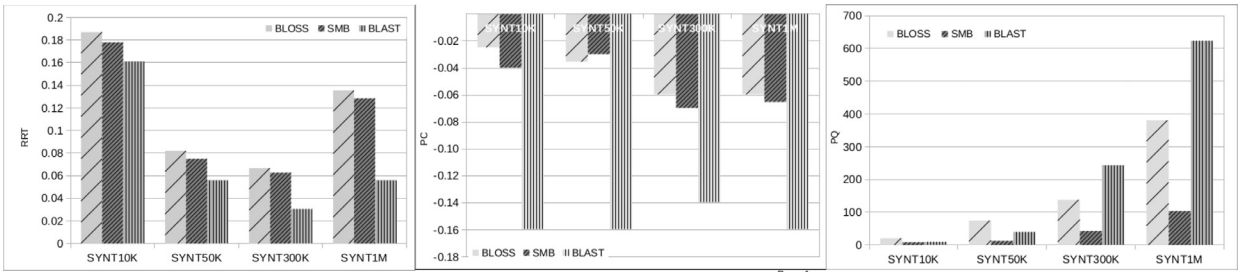

Fig. 12. RRT, ΔPC, and ΔPQ of BLOSS, SMB and BLAST. .

attributes and improve their weight. These unsupervised methods depend on a global or local threshold value, which is very sensitive to the dirtiness level of the dataset.

The supervised meta-blocking approach SMB eliminates the less promising edges by transforming the graph structure into features to be applied to a binary classifier (e.g. SVM, Naive Bayes, decision tree). SMB produces an effective classification model but at the cost of a large training set. Such set is produced by selecting an equal proportion of matching and non-matching pairs (defined beforehand). Such strategy is unrealistic in real-word scenarios since the matching pairs are hidden behind a large number of non-matching pairs. This means that a very large set of pairs will have to be labeled. BLOSS addresses such drawback by selecting a reduced set composed of the most informative pairs to be labeled by the user.

Active learning. Active learning aims at selecting a reduced and non-redundant set of the pairs to produce a training set with information that faithfully represents an (much larger) unlabeled dataset. Active learning has been extensively used in deduplication especially in the classification step [2,23,24]. The basic idea is to produce a committee of models (each one with some variations) to identify the pairs that induce a disagreement in the committee. However, such approaches require an initial training set, manually produced, and usually do not have a clear stopping criterion. Christen et al. [26] proposes an active learning approach with a limited budget, defined by the user, for manual labeling at the presence of noisy labeled pairs in the training set. Their solution is focused on the classification step of the deduplication task. Such work recursively processes the blocks and selects the pairs based on heuristics, e.g., the pairs farthest from the hyperplane decision constructed for the current training set (and thus have more diversity) are selected. However, such approach depends heavily on a set of thresholds appropriated defined. Moreover, such approach is heavily based on cluster analysis (e.g., computing the blocking matching proportion), which is computationally prohibitive in the context meta-blocking since large and oversized blocks are present.

Christen et al. [26] also experimentally show that monotonicity[15] does not hold universally since some datasets are composed of pairs with very low similarity degree that still represent a matching. BLOSS prune the outliers using the similarity pairs degree to produce a “pure” training set. After that, the labeled set is used to train the classifier with a set of different features, i.e., the features focus on the selection diverse aspects of the dataset, as the token frequency, number of common tokens, number of distinct tokens, blocking size. In this way, since BLOSS combines several features values to select instances to label in a non-linear way by means of classification rules defined on-demand on an instance-basis, we believe that monotonicity does not represent a problem for BLOSS.

[15] The assumption of monotonicity of similarities indicates that a pair with higher similarity is more likely to be a matching than a pair with lower similarity.

In [17], the authors overcame previous active learning approaches limitations by proposed a method called SSAR, to identify the most informative pairs based on the number of association rules project over the actual training set.

Recently, [2] proposed a sampling selection approach for the classification step, called T3S. It combines a heuristic to select informative subsets of pairs with the SSAR rule-based active learning [17]. T3S produces a ranking to discretize the unlabeled pairs into levels and incrementally exploits each level (using SSAR) to identify and label the most informative pairs. As a large number of pairs is present, the labeled pairs are used to recognize boundaries where the most challenging pairs lie. Only the most challenging pairs are processed by supervised or unsupervised classification approaches to identify the matching pairs. Note that at a higher level, T3S is conceptually similar to BLOSS – they aim at selecting a small representative training set, but with very different goals. This requires different solutions like the one proposed by BLOSS. (Meta-)Blocking focuses on the improvement firstly in the number of the true matching pairs to be retrieved (PC) and secondly in the precision. To the best of knowledge, we are the first to propose an approach to reduce the user effort in the meta-blocking step by using an active learning approach.

6. Conclusions

In this article, we presented a new sampling selection strategy, called BLOSS, that is able to improve the supervised Meta-blocking step. By improvement, we mean the decreasing of user labeling effort and the increasing of pairs quality. BLOSS is divided into three main stages. In the first stage, it selects small and representative sets of pairs. In the second stage, such sets are filtered out, using a rule based active learning, to remove redundant information. In the third stage, BLOSS identifies and removes from the training set, the non-matching pairs that operate as outliers when used to configure the supervised algorithms. We show that removing outliers improves PC (i.e., recall). Our experiments demonstrated that BLOSS can substantially reduce the user labeling effort in all analyzed datasets. Moreover, BLOSS improves the precision and, generally, the recall.

We believe that BLOSS outperformed the state-of-the-art approaches because (*i*) the clustering (levels) of unlabeled pairs allowed the creation of smaller and balanced sets to be seeded to an active learning approach, (*ii*) the active learning approach eliminates the redundant pairs producing a very small set of pairs to be labeled by the user, (*iii*) the labeled pairs produced are, then, pruned to drop the outliers out, (*iv*) the resulting labeled pairs are quite informative to be used as input in the classification process, and (*v*) although BLOSS requests more run-time than the baselines, it results in better effectiveness and reduced manual user efforts.

As future work, we intend to propose a new approach which combines BLOSS with BLAST. In more details, the unsupervised meta-blocking can be applied to prune the less promising edges from the graph, reducing the universe of pairs (i.e., high recall but

low precision, as experimentation shows). BLOSS would be applied to the survivor candidate pairs to improve the precision. With such an idea, it would be possible to take advantage of both approaches to improve the selection process. Other potential future works come in the direction of combine BLOSS with more effective blocking approaches [5] and learning the blocking schemes [42]. With these, we believe that is possible to improve the blocking quality reducing the meta-blocking efforts. Finally, parallelization of the Classification Step of BLOSS is in our plans since, as we have seen, it is the most demanding time portion of the whole meta-blocking process.

Acknowledgments

This work was partially funded by projects InWeb (grant MCT/CNPq 573871/2008- 6) and MASWeb (grant FAPEMIG/PRONEX APQ-01400-14), and by the authors individual grants from CNPq, FAPEMIG, and CAPES under process number 88881.119081/2016-01.

References

[1] M.G. de Carvalho, A.H.F. Laender, M.A. Goncalves, A.S.d. Silva, A genetic programming approach to record deduplication, IEEE Trans. Knowl. Data Eng. (2012).
[2] G. Dal Bianco, R. Galante, M.A. Gonçalves, S. Canuto, C.A. Heuser, A practical and effective sampling selection strategy for large scale deduplication, IEEE Trans. Knowl. Data Eng. 27 (9) (2015) 2305–2319.
[3] H.L. Dunn, Record linkage, Am. J. Public Health Nations Health 36 (12) (1946) 1412–1416.
[4] I. Fellegi, A. Sunter, A theory for record linkage, J. Amer. Stat. Assoc. 64 (328) (1969) 1183–1210. http://www.jstor.org/stable/2286061.
[5] P. Christen, A survey of indexing techniques for scalable record linkage and deduplication, IEEE Trans. Knowl. Data Eng. 24 (9) (2012) 1537–1555.
[6] G. Papadakis, J. Svirsky, A. Gal, T. Palpanas, Comparative analysis of approximate blocking techniques for entity resolution, Proc. VLDB Endowment 9 (9) (2016) 684–695.
[7] A.K. Elmagarmid, P.G. Ipeirotis, V.S. Verykios, Duplicate record detection: A survey, IEEE Trans. Knowl. Data Eng. 19 (1) (2007) 1–16.
[8] S. Chaudhuri, V. Ganti, R. Kaushik, A primitive operator for similarity joins in data cleaning, in: Proc. ICDE, 2006, p. 5.
[9] R.J. Bayardo, Y. Ma, R. Srikant, Scaling up all pairs similarity search, in: Proc. WWW, ACM, New York, NY, USA, 2007, pp. 131–140.
[10] C. Xiao, W. Wang, X. Lin, J.X. Yu, G. Wang, Efficient similarity joins for near-duplicate detection, ACM TODS 36 (3) (2011) 15:1–15:41.
[11] J. Wang, G. Li, J. Fe, Fast-join: An efficient method for fuzzy token matching based string similarity join, in: Proc. ICDE, 2011, pp. 458–469.
[12] H. Köpcke, E. Rahm, Frameworks for entity matching: A comparison, Data Knowl. Eng. 69 (2010) 197–210.
[13] G. Papadakis, G. Papastefanatos, G. Koutrika, Supervised meta-blocking, Proc. VLDB Endowment 7 (14) (2014) 1929–1940.
[14] G. Papadakis, G. Papastefanatos, T. Palpanas, M. Koubarakis, Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking, EDBT, 2016.
[15] G. Simonini, S. Bergamaschi, H.V. Jagadish, Blast: a loosely schema-aware meta-blocking approach for entity resolution, Proc. VLDB Endowment 9 (12) (2016) 1173–1184.
[16] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, VLDB J. 10 (4) (2001) 334–350.
[17] R.M. Silva, M.A. Gonçalves, A. Veloso, A two-stage active learning method for learning to rank, JASIST 65 (1) (2014) 109–128.
[18] C. Xiao, W. Wang, X. Lin, J.X. Yu, Efficient similarity joins for near duplicate detection, in: WWW '08, ACM, New York, NY, USA, 2008, pp. 131–140.
[19] G. Papadakis, G. Koutrika, T. Palpanas, W. Nejdl, Meta-blocking: Taking entity resolutionto the next level, IEEE Trans. Knowl. Data Eng. 26 (8) (2014) 1946–1960.
[20] A. McCallum, K. Nigam, L.H. Ungar, Efficient clustering of high-dimensional data sets with application to reference matching, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2000, pp. 169–178.
[21] G. Papadakis, E. Ioannou, C. Niederée, P. Fankhauser, Efficient entity resolution for large heterogeneous information spaces, in: Proceedings of the fourth ACM international conference on Web search and data mining, ACM, 2011, pp. 535–544.
[22] R.A. Baeza-Yates, B.A. Ribeiro-Neto, Modern Information Retrieval, ACM Press / Addison-Wesley, 1999.
[23] S. Sarawagi, A. Bhamidipaty, Interactive deduplication using active learning, in: Proc. ACM SIGKDD, ACM, 2002, pp. 269–278.
[24] K. Bellare, S. Iyengar, A.G. Parameswaran, V. Rastogi, Active sampling for entity matching, in: Proc. ACM SIGKDD, 2012.
[25] S. Sarawagi, A. Kirpal, Efficient set joins on similarity predicates, in: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, in: SIGMOD '04, ACM, New York, NY, USA, 2004, pp. 743–754.
[26] P. Christen, D. Vatsalan, Q. Wang, Efficient entity resolution with adaptive and interactive training data selection, in: Data Mining (ICDM), 2015 IEEE International Conference on, IEEE, 2015, pp. 727–732.
[27] A. Arasu, M. Gotz, R. Kaushik, On active learning of record matching packages, in: Proc. SIGMOD, ACM, 2010, pp. 783–794.
[28] C.E. Brodley, M.A. Friedl, et al., Identifying and eliminating mislabeled training instances, in: Proceedings of the National Conference on Artificial Intelligence, 1996, pp. 799–805.
[29] B. Frénay, M. Verleysen, Classification in the presence of label noise: a survey, IEEE Trans. Neural Netw. Learn. Syst. 25 (5) (2014) 845–869.
[30] N. Segata, E. Blanzieri, S.J. Delany, P. Cunningham, Noise reduction for instance-based learning with a local maximal margin approach, J. Intell. Inf. Syst. 35 (2) (2010) 301–331.
[31] M.G. Bulmer, Principles of Statistics, Courier Corporation, 1979.
[32] G. Papadakis, G. Papastefanatos, T. Palpanas, Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking, Institute for the Management of Information Systems, 2015.
[33] P. Christen, T. Churches, Febrl - Freely extensible biomedical record linkage, Technical Report, DSpace at The Australian National University, 2002. http://hdl.handle.net/1885/40723.
[34] H. Köpcke, A. Thor, E. Rahm, Evaluation of entity resolution approaches on real-world match problems, PVLDB 3 (1) (2010) 484–493.
[35] M. Michelson, C.A. Knoblock, Learning blocking schemes for record linkage, in: Proceedings of the National Conference on Artificial Intelligence, vol. 21, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 440.
[36] G. Schmidberger, E. Frank, Unsupervised discretization using tree-based density estimation, in: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2005, pp. 240–251.
[37] M.V. Joshi, G. Karypis, V. Kumar, Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets, in: Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998, IEEE, 1998, pp. 573–579.
[38] M. Bilenko, B. Kamath, R.J. Mooney, Adaptive blocking: Learning to scale up record linkage, in: Data Mining, 2006. ICDM '06. Sixth International Conference on, 2006, pp. 87–96.
[39] J. Wang, G. Li, J.X. Yu, J. Feng, Entity matching: how similar is similar, Proc. VLDB Endow. 4 (10) (2011) 622–633.
[40] L.A.-R. Sidney Ribeiro-Júnior, R.D. Quirino, W.S. Martins, gSSJoin: a GPU-based set similarity join algorithm, in: Proceedings of 31st Brazilian Symposium on Databases, 2016.
[41] X. Feng, H. Jin, R. Zheng, L. Zhu, Near-duplicate detection using GPU-based simhash scheme, in: Smart Computing (SMARTCOMP), 2014 International Conference on, IEEE, 2014, pp. 223–228.
[42] M. Kejriwal, D.P. Miranker, An unsupervised algorithm for learning blocking schemes, in: Data Mining (ICDM), 2013 IEEE 13th International Conference on, IEEE, 2013, pp. 340–349.

Guilherme Dal Bianco is an adjunct professor of computer science at the Federal University of Fronteira Sul (UFFS), Brazil. His research interests include data matching, data integration and Big Data.

Marcos André Gonçalves is an associate professor of computer science at the Universidade Federal de Minas Gerais. His research interests include information retrieval and text mining.

Denio Duarte is an associate professor at Federal University of Fronteira Sul (UFFS), Brazil. He holds a PhD in Computer Science (Université François-Rabelais de Tours - France). His research interests are database, semistructured database, and machine learning.