In *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM-06)*. pp. 87-96, Hong Kong, December 2006.

Adaptive Blocking: Learning to Scale Up Record Linkage

Mikhail Bilenko[*]
Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA
mbilenko@microsoft.com

Beena Kamath[*]
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043 USA
beena@google.com

Raymond J. Mooney
Dept. of Computer Sciences
University of Texas at Austin
1 University Station C0500
Austin, TX 78712 USA
mooney@cs.utexas.edu

Abstract

Many data mining tasks require computing similarity between pairs of objects. Pairwise similarity computations are particularly important in record linkage systems, as well as in clustering and schema mapping algorithms. Because the number of object pairs grows quadratically with the size of the dataset, computing similarity between all pairs is impractical and becomes prohibitive for large datasets and complex similarity functions. Blocking methods alleviate this problem by efficiently selecting approximately similar object pairs for subsequent distance computations, leaving out the remaining pairs as dissimilar. Previously proposed blocking methods require manually constructing an index-based similarity function or selecting a set of predicates, followed by hand-tuning of parameters. In this paper, we introduce an adaptive framework for automatically learning blocking functions that are efficient and accurate. We describe two predicate-based formulations of learnable blocking functions and provide learning algorithms for training them. The effectiveness of the proposed techniques is demonstrated on real and simulated datasets, on which they prove to be more accurate than non-adaptive blocking methods.

1 Introduction

A number of machine learning and data mining tasks involve computing similarity between pairs of instances. For example, in record linkage (also known as object identification [39], de-duplication [35], entity matching [11, 37, 2] and identity uncertainty [33, 25]), similarity must be computed between record pairs to identify groups of records that refer to the same underlying entity. Many clustering algorithms, e.g., greedy agglomerative or spectral clustering methods, require similarity to be computed between all pairs of instances to form the pairwise similarity matrix, which is then used by the clustering algorithm [19, 31]. A number of schema mapping methods also rely on pairwise similarity computations between descriptions of concepts or records belonging to them [12].

Because the total number of pairwise similarity computations grows quadratically with the size of the input dataset, scaling the above tasks to large datasets is problematic. Additionally, even for small datasets, estimation of the full similarity matrix can be difficult if computationally costly similarity functions are used. At the same time, in many tasks, the majority of similarity computations are unnecessary because most instance pairs are highly dissimilar and have no influence on the task output. Avoiding the unnecessary computations results in a sparse similarity matrix, which can be exploited by many algorithms, allowing them to scale up to large datasets.

Blocking methods efficiently select a subset of record pairs for subsequent similarity computation, ignoring the remaining pairs as highly dissimilar and therefore irrelevant. A number of blocking algorithms have been proposed by researchers in the past years [15, 20, 18, 26, 1, 6, 21, 41]. These techniques typically form groups (blocks) of observations using indexing or sorting. This allows efficient selection of instance pairs from each block for subsequent similarity computations. Some blocking methods are based on the assumption that there exists a pre-defined similarity metric that correctly captures similar pairs for the domain and task at hand, e.g., edit distance [6, 21] or Jaccard similarity [26], while others assume that forming blocks based on lexicographic sorting of records on some field(s) yields an optimal strategy [18, 41]. Manual selection of fields and parameter tuning are required by all existing blocking strategies to reduce the number of returned dissimilar pairs while retaining the similar pairs.

Since an appropriate blocking strategy can be highly domain-dependent, the ad-hoc construction and manual

[*]Work done at the University of Texas at Austin

tuning of blocking methods makes this task non-trivial. A suboptimal blocking method may lead to over-selection of many dissimilar pairs that impedes efficiency, or, worse, under-selection of important similar pairs that decreases accuracy. Because there can be many potentially useful blocking criteria over multiple record fields, there is a need for automating the process of constructing blocking strategies so that all same-entity or same-cluster pairs are retained while the maximum number of dissimilar pairs is discarded.

In this paper, we formalize the problem of learning an optimal blocking strategy using training data. In many record linkage domains, some fraction of instances contain true entity identifiers, e.g., UPC (bar code) numbers for retail products, SSN numbers for individuals, ISBN numbers for books, or DOI identifiers for citations. In domains where such supervision is unavailable, it can be obtained by performing clustering or record linkage on a subset of data where all object pairs can be used. Labeled data then allows evaluating possible blocking functions and selecting from them one that is optimal: it selects all or nearly all coreferent pairs (that describe the same entity or belong to the same cluster), and a minimal number of non-coreferent pairs (that describe different entities or belong to different clusters).

We propose to construct blocking functions based on sets of general *blocking predicates* which efficiently select all instance pairs that satisfy some binary similarity criterion. Table 1 below shows examples of predicates for specific record fields in different domains. We formulate the problem of learning an optimal blocking function as the task of finding a combination of blocking predicates that captures all or nearly all coreferent object pairs and a minimal number of non-coreferent pairs. Our approach is general in the sense that we do not place restrictions on the similarity predicates computed on instance pairs selected by blocking, such as requiring them to be an inner product or to correspond to a distance metric.

We consider two types of blocking functions: disjunctions of blocking predicates and predicates combined in disjunctive normal form (DNF). We show that the problem of constructing an optimal blocking function can be formulated as an instance of the Red-Blue Set Cover problem [5]. While finding a globally optimal solution for this problem is NP-hard, our formulation allows employing an efficient approximation algorithm for learning blocking functions. Empirical evaluation on synthetic and real-world record linkage datasets demonstrates the benefits of our approach and its advantages over non-adaptive blocking functions.

2 Problem Formulation

2.1 Record Linkage

While there are many tasks in data mining and artificial intelligence applications that require computing pairwise similarity over a dataset, in this paper we use record linkage as an example of an application that requires efficiently selecting a subset of similar pairs where objects are often described by heterogeneous fields. Our techniques can be directly extended to clustering and schema mapping, while extending them for use with kernel methods remains an interesting challenge for future work.

Domain	**Blocking Predicate**
Census Data	Same 1st Three Chars in *Last Name*
Product Normalization	Common token in *Manufacturer*
Citations	*Publication Year* same or off-by-one

Table 1. Examples of blocking predicates from different record linkage domains

Record linkage is the problem of determining which database records refer to the same underlying entity. It is an integral part of many systems that combine information from multiple sources on the web, such as digital libraries [16], customer review and recommendation sites [39, 28], and comparison shopping systems [3]. Other domains where record linkage problem is commonly studied include census and mailing data [40, 35, 6] and healthcare records [32].

The overall problem of identifying coreferent records has been studied in several research communities under different names that include record linkage [15, 40], the merge/purge problem [40], duplicate detection [29, 35, 4], reference matching [26, 23], object identification [39, 38], entity name matching and clustering [11, 37], hardening soft databases [10], fuzzy de-duplication [6], identity uncertainty [33, 25], robust reading [24], reference reconciliation [13], and entity resolution [2].

Most systems solve the record linkage problem in three stages. First, a subset of *candidate* pairs is selected among all record pairs that must contain all (or nearly all) pairs of coreferent records. Second, similarity is estimated for the candidate pairs using one or more similarity functions [11, 39, 35, 4]. Finally, linkage decisions are made for all candidate pairs using the computed pairwise similarities, either in isolation for each pair or collectively over the entire set of records. In this work, we focus on the first stage where candidate pairs are selected by a blocking algorithm for subsequent similarity calculations. Unlike previous approaches that rely on pre-selected or hand-constructed blocking methods, we will employ machine learning techniques to automatically construct efficient and accurate blocking functions.

2.2 Problem Setting

Let us formally define the problem of learning an optimal blocking function. We assume that a training dataset

$\mathcal{D}_{train} = \{\mathcal{X}, \mathcal{Y}\}$ is available that includes a set $\mathcal{X} = \{x_i\}_{i=1}^n$ of n records known to refer to m true objects: $\mathcal{Y} = \{y_i\}_{i=1}^n$, each y_i is the true object identifier for the i-th record: $y_i \in \{1, \dots, m\}$. Each record x_i may have one or more fields.

We assume that a set of s general *blocking predicates* $\{p_i\}_{i=1}^s$ is available, where each predicate p_i corresponds to two functions:

- *Indexing function* $h_i(\cdot)$ is a unary function that is applied to a field value from some domain $\mathrm{Dom}(h_i)$ (e.g., strings, integers, or categories) and generates one or more *keys* for the field value: $h_i : \mathrm{Dom}(h_i) \to \mathcal{U}^*$, where $\mathcal{U}$ is the set of all possible keys;
- *Equality function* $p_i(\cdot,\cdot)$ returns 1 if the intersection of the key sets produced by the indexing function on its arguments is non-empty, and returns zero otherwise: $p_i(x_j, x_k) = 1$ iff $h_i(x_j) \cap h_i(x_k) \neq \emptyset$. Any record pair (x_j, x_k) for which $p_i(x_j, x_k) = 1$ is *covered* by the predicate p_i.

Each general blocking predicate can be instantiated for a particular field (or a combination of fields) in a given domain, resulting in a set of *specific* blocking predicates for the domain. Given a database with d fields and a set of s general blocking predicates, we obtain $t \leq s \times d$ specific predicates $\mathcal{P} = \{p_i\}_{i=1}^t$ by applying the general predicates to all fields of the appropriate type. For example, suppose we have four general predicates defined for all textual fields: *"Contain Common Token"*, *"Exact Match"*, and *"Same 1st Three Chars"*, *"Contains Same or Off-By-One Integer"*. When these general predicates are instantiated for the bibliographic citation domain with five textual fields, *author*, *title*, *venue*, *year*, and *other*, we obtain $5 \times 4 = 20$ specific blocking predicates for this domain. Figure 1 below demonstrates the values produced by the indexing functions of these specific blocking predicates on a sample citation record (we assume that all strings are converted to lowercase and punctuation is removed before the application of the indexing functions).

Multiple blocking predicates are combined by an overall *blocking function* $f_{\mathcal{P}}$ constructed using the set $\mathcal{P}$ of predicates. Like the individual predicates, $f_{\mathcal{P}}$ corresponds to both an indexing function that can be applied to any record, and an equality function for any pair of records. Pairs for which the equality function returns 1 are covered: they comprise the set of candidate pairs returned for subsequent similarity computation, while pairs for which the blocking function returns 0 are ignored (uncovered). Efficient generation of the set of candidate pairs requires computing the indexing function for all records, followed by retrieval of all candidate pairs using inverted indices.

Given the set $\mathcal{P} = \{p_i\}_{i=1}^t$ containing t specific blocking predicates, the objective of the adaptive blocking framework is to identify an optimal blocking function $f^*_{\mathcal{P}}$ that combines all or a subset of the predicates in $\mathcal{P}$ so that the set of candidate pairs it returns contains all or nearly all coreferent (positive) record pairs and a minimal number of non-coreferent (negative) record pairs.

Formally, this objective can be expressed as follows:

$$\begin{aligned} f^*_{\mathcal{P}} &= \operatorname*{argmin}_{f_{\mathcal{P}}} \sum_{(x_i,x_j)\in\mathcal{R}} f_{\mathcal{P}}(x_i,x_j) \\ \text{s.t.} &\quad |\mathcal{B}| - \sum_{(x_i,x_j)\in\mathcal{B}} f_{\mathcal{P}}(x_i,x_j) \; < \; \varepsilon \end{aligned} \tag{1}$$

where $\mathcal{R} = \{(x_i,x_j) : y_i \neq y_j\}$ is the set of non-coreferent pairs, $\mathcal{B} = \{(x_i,x_j) : y_i = y_j\}$ is the set of coreferent pairs, and ε is a small value indicating that up to ε coreferent pairs may remain uncovered, thus accommodating noise and particularly difficult coreferent pairs. The optimal blocking function $f^*_{\mathcal{P}}$ must be found in a hypothesis space that corresponds to some method of combining the individual blocking predicates. In this paper, we consider two classes of blocking functions:

- **Disjunctive blocking** selects record pairs that are covered by at least one blocking predicate from the subset of predicates that comprise the blocking function. This strategy can be viewed as covering pairs for which a the equality function for at least one of the selected predicates returns 1. The blocking function is trained by selecting a subset of blocking predicates from $\mathcal{P}$.
- **Disjunctive Normal Form (DNF) blocking** selects object pairs that are covered by at least one conjunction of blocking predicates from a constructed set of conjunctions. This strategy can be viewed as covering record pairs for which at least one equality function of a conjunction of predicates returns 1. The blocking function is trained by constructing a DNF formula from the blocking predicates.

Each type of blocking functions leads to a distinct formulation of the objective (1), and we consider them individually in the following subsections.

2.2.1 Disjunctive blocking

Given a set of potential blocking predicates $\mathcal{P} = \{p_i\}_{i=1}^t$, a disjunctive blocking function corresponds to selecting some subset of predicates $\mathcal{P}' \subseteq \mathcal{P}$, performing blocking using each $p_i \in \mathcal{P}'$, and then selecting record pairs that share at least one common key in the key sets computed by the indexing functions of the selected predicates. Then, the equality function for the disjunctive blocking function based on predicate subset $\mathcal{P}' = \{p_{i_1}, \dots, p_{i_k}\}$ returns 1 if the equality function for at least one predicate returns 1: $f_{\mathcal{P}'}(x_i,x_j) = p_{i_1}(x_i,x_j) \vee \dots \vee p_{i_k}(x_i,x_j)$.

Learning the optimal blocking function $f^*_{\mathcal{P}}$ requires selecting a subset $\mathcal{P}^*$ of predicates that results in all or nearly all coreferent pairs being covered by at least one predicate

Sample record:

author	*year*	*title*	*venue*	*other*
Freund, Y.	(1995).	Boosting a weak learning algorithm by majority.	Information and Computation,	121(2), 256-285

Blocking predicates and key sets produced by their indexing functions for the record:

	author	*title*	*venue*	*year*	*other*
Contain Common Token	{*freund, y*}	{*boosting, a, weak, learning, algorithm, by, majority*}	{*information, computation*}	{*1995*}	{*121, 2, 256, 285*}
Exact Match	{*'freund y'*}	{*'boosting a weak learning algorithm by majority'*}	{*'information and computation'*}	{*'1995'*}	{*'121 2 256 285'*}
Same 1st Three Chars	{*fre*}	{*boo*}	{*inf*}	{*199*}	{*121*}
Contains Same or Off-By-One Integer	$\emptyset$	$\emptyset$	$\emptyset$	{*1994_1995, 1995_1996*}	{*120_121, 121_122, 1_2, 2_3, 255_256, 256_257, 284_285, 285_286*}

Figure 1. Blocking key values for a sample record

in $\mathcal{P}^*$, and a minimal number of non-coreferent pairs being covered. Then the general adaptive blocking problem in Eq.(1) can be written as follows:

$$\begin{aligned} \boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} & \sum_{(x_i,x_j)\in\mathcal{R}} [\![\boldsymbol{w}^T \boldsymbol{p}(x_i,x_j) > 0]\!] \\ \text{s.t.} \quad & |\mathcal{B}| - \sum_{(x_i,x_j)\in\mathcal{B}} [\![\boldsymbol{w}^T \boldsymbol{p}(x_i,x_j) > 0]\!] < \varepsilon \qquad (2) \\ & \boldsymbol{w} \text{ is binary} \end{aligned}$$

where $\boldsymbol{w}$ is a binary vector of length t encoding which of the potential blocking criteria are selected, and $\boldsymbol{p}(x_i,x_j)$ is a vector of binary values returned by the t predicates for pair (x_i,x_j).

This formulation of the learnable blocking problem is equivalent to the *Red-Blue Set Cover* problem [5] if $\varepsilon = 0$. Figure 2 illustrates the equivalence. The task of selecting a subset of predicates is represented by a graph with three sets of vertices. The bottom row of β vertices corresponds to positive (coreferent) record pairs designated as the set of *blue* elements $\mathcal{B} = \{b_1,\ldots,b_\beta\}$. The top row of ρ vertices corresponds to negative (non-coreferent) record pairs designated as the set of *red* elements $\mathcal{R} = \{r_1,\ldots,r_\rho\}$. The middle row of t vertices represents the set of blocking predicates $\mathcal{P}$, where each $p_i \in \mathcal{P}$ corresponds to a set covering some red and blue elements. Every edge between an element vertex and a predicate vertex indicates that the record pair represented by the element vertex is covered by the predicate. Learning the optimal disjunctive blocking function is then equivalent to selecting a subset of predicate vertices with their incident edges so that at least $\beta - \varepsilon$ blue (positive) vertices have at least one incident edge, while the *cover cost*, equal to the number of red (negative) vertices with at least one incident edge, is minimized.

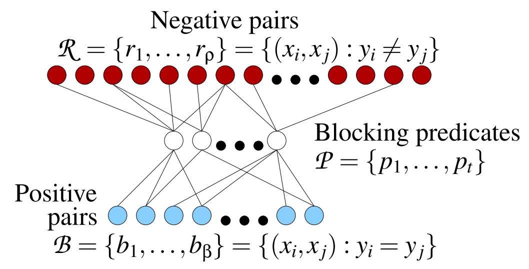
Negative pairs
$\mathcal{R} = \{r_1,\ldots,r_\rho\} = \{(x_i,x_j) : y_i \neq y_j\}$
Blocking predicates
$\mathcal{P} = \{p_1,\ldots,p_t\}$
Positive pairs
$\mathcal{B} = \{b_1,\ldots,b_\beta\} = \{(x_i,x_j) : y_i = y_j\}$

Figure 2. Red-blue set cover view of blocking

2.2.2 DNF Blocking

In some domains, a disjunctive combination of blocking predicates may be an insufficient representation of the optimal blocking strategy. For example, in US Census data, conjunctions of predicates such as *"Same Zip AND Same 1st Char in Surname"* yield useful blocking criteria [41]. To incorporate such predicate conjunctions, we must extend the disjunctive formulation described above to combine predicates in disjunctive normal form (DNF). Then, that hypothesis space for the blocking function must include disjunctions of not just individual blocking predicates, but also of their conjunctions.

A search for the optimal DNF blocking function can be viewed as solving an extended variant of the red-blue set cover problem. In that variant, the cover is constructed using not only the sets representing the original predicates, but also using additionally constructed sets representing predicate conjunctions. Because the number of all possible conjunctions is exponential, only conjunctions up to predetermined length k are considered. In Figure 2, considering a conjunction of blocking predicates corresponds to adding a vertex to the middle row, along with edges connecting it to the red and blue vertices from the intersection of covered vertex sets for the individual predicates in the conjunction.

The learnable blocking problem is then analogous to one for disjunctive blocking: it is the task of selecting a set of predicate vertices, including those for conjunctions, so that at least $\beta - \varepsilon$ blue vertices have one or more incident edges, while the number of red vertices with at least one incident edge is minimized.

3 Algorithms

3.1 Pairwise Training Data

For record linkage, supervision is available in many domains in the form of records for which the true entities to which they refer are known, as discussed in Section 1. Such labeled records comprise the training dataset $\mathcal{D}_{train} = \{\mathcal{X}, \mathcal{Y}\}$ that can be used to generate pairwise supervision for learning the blocking function in the form of coreferent (blue) and non-coreferent (red) record pairs. For large databases, it is impractical to explicitly generate and store in memory all positive and negative pairs. However, the set of covered pairs for each predicate can be computed using the indexing function and stored in an inverted index. Then, predicate covers can be stored and accessed efficiently using bit arrays.

If training data is unavailable, it can be obtained automatically by performing linkage or clustering without blocking on a subset of the database. Then, linkage or clustering output provides training data for learning the blocking function for future iterations.

3.2 Learning Blocking Functions

3.2.1 Disjunctive Blocking

The equivalence of learning optimal disjunctive blocking and the red-blue set cover problem described in Section 2.2.1 has discouraging implications at first glance. The red-blue set cover problem is NP-hard, and Carr et al. [5] have shown that unless P=NP, it cannot be efficiently approximated within a factor $O(2^{\log^{1-\delta} t})$, $\delta = 1/\log\log^c t$ for any constant c, where t is the number of predicates under consideration. However, several approximate algorithms have been proposed for the red-blue set cover problem [5, 34]. We propose to learn disjunctive blocking functions by modifying Peleg's greedy algorithm to obtain a "soft cover" where up to ε blue elements may remain uncovered. The algorithm has an approximation ratio of $2\sqrt{t \log \beta}$ [34], and is particularly suitable for the adaptive blocking setting as it involves early discarding of particularly costly sets (blocking predicates that cover too many non-coreferent pairs), leading to more space-efficient learning of the blocking function.

The outline of the algorithm ApproxRBSetCover is shown in Figure 3. The algorithm is provided with training data in the form of β coreferent record pairs $\mathcal{B} = \{b_1, \ldots, b_\beta\}$ and ρ non-coreferent records pairs $\mathcal{R} = \{r_1, \ldots, r_\beta\}$, where each r_i and b_i represents a record pair (x_{i_1}, x_{i_2}). For each predicate $p_i \in \mathcal{P}$, let *covered negatives* $\mathcal{R}(p_i)$ be the set of negative pairs it covers, *predicate cost* $r(p_i)$ be the number of negative pairs it covers $r(p_i) = |\mathcal{R}(p_i)|$, *covered positives* $\mathcal{B}(p_i)$ be the set of positive pairs it covers, and *coverage* $b(p_i)$ be the number of covered positives, $b(p_i) = |\mathcal{B}(p_i)|$. For each negative pair $r_i = (x_{i_1}, x_{i_2})$, let the *degree* $\deg(r_i, \mathcal{P})$ be the number of predicates in $\mathcal{P}$ that cover it; degree for a positive pair, $\deg(b_i, \mathcal{P})$, is defined analogously. In step 1 of the algorithm, blocking predicates that cover too many negative pairs are discarded, where the parameter η can be set to a fraction of the total number of pairs in the dataset. Then, negative pairs covered by too many predicates are discarded in step 4, which intuitively corresponds to disregarding non-coreferent pairs that are highly similar and are placed in the same block by many predicates.

Next, a standard weighted set cover problem is set up for the remaining predicates and pairs by setting the cost of each predicate to be the number of negatives it covers and removing the negatives. The resulting weighted set cover problem is solved in steps 6-11 using Chvatal's greedy approximation algorithm [8]. The algorithm iteratively constructs the cover, at each step adding the blocking predicate p_i that maximizes a greedy heuristic: the ratio of the number of previously uncovered positives over the predicate cost. To soften the constraint requiring all positive pairs to be covered, we add an early stopping condition permitting up to ε positives to remain uncovered. In practice, ε should be set to 0 at first, and then gradually increased if the cover identified by the algorithm is too costly for the application at hand (that is, when covering all positives incurs covering too many negatives). If the minimal acceptable proportion of covered positives (minimum desired recall) is known to be r for the domain at hand, ε can be set to $r\beta$.

3.2.2 DNF Blocking

Learning DNF blocking can be viewed as an extension of learning disjunctive blocking where not only individual blocking predicates may be selected, but also their conjunctions. We assume that conjunctions that include up to k predicates are considered. Because enumerating over all possible conjunctions of predicates results in an exponential number of predicate sets under consideration, we propose a

```
Algorithm: ApproxRBSetCover
Input: Training set $\mathcal{B} = \{b_1, \ldots, b_\beta\}$ and $\mathcal{R} = \{r_1, \ldots, r_\rho\}$ where
        each $b_i \in \mathcal{B}$ is a pair of coreferent records $(x_{i_1}, x_{i_2})$ s.t. $y_{i_1} = y_{i_2}$
        each $r_i \in \mathcal{R}$ is a pair of non-coreferent records $(x_{i_1}, x_{i_2})$ s.t. $y_{i_1} \neq y_{i_2}$
    Set of blocking predicates $\mathcal{P} = \{p_1, \ldots, p_t\}$
    Maximum number of coreferent pairs allowed to be uncovered $\varepsilon$
    Maximum number of pairs that any predicate may cover $\eta$
Output: A disjunctive blocking function based on subset $\mathcal{P}^* \subset \mathcal{P}$
Method:
1. Discard from $\mathcal{P}$ all predicates $p_i$ for which $r(p_i) \geq \eta$:
   $\mathcal{P} \leftarrow \{p_i \in \mathcal{P} | r(p_i) \leq \eta\}$.
2. If $|\mathcal{B}| - |\mathcal{B}(\mathcal{P})| > \varepsilon$ return $\mathcal{P}$ (cover is not feasible, $\eta$ is too low)
3. Set $\gamma = \sqrt{t / \log \beta}$.
4. Discard all $r_i$ covered by more than $\gamma$ predicates:
   $\mathcal{R} \leftarrow \{r_i \in \mathcal{R} | \deg(r_i, \mathcal{P}) \leq \gamma\}$
5. Construct an instance of weighted set cover $\mathcal{T}$ by discarding
   elements of $\mathcal{R}$, creating a set $\tau_i$ for each $p_i \in \mathcal{P}$, and setting
   its weight $\omega(\tau_i) = r(p_i)$.
6. $\mathcal{T}^* \leftarrow \emptyset$
7. while $|B| \geq \varepsilon$
8.     select $\tau_i \in T$ that maximizes $b(\tau_i)/\omega(\tau_i)$
9.     $\mathcal{B} \leftarrow \mathcal{B} - \mathcal{B}(\tau_i)$
10.    $\mathcal{T}^* \leftarrow \mathcal{T}^* \cup \{\tau_i\}$
11. Return the set of predicates $\mathcal{P}^*$ corresponding to $\mathcal{T}^*$.
```

Figure 3. The algorithm for learning disjunctive blocking

```
Algorithm: ApproxDNF
Input: Training set $\mathcal{B} = \{b_1, \ldots, b_\beta\}$ and $\mathcal{R} = \{r_1, \ldots, r_\rho\}$ where
        each $b_i$ is a pair of coreferent records $(x_{i_1}, x_{i_2})$ s.t. $y_{i_1} = y_{i_2}$
        each $r_i$ is a pair of non-coreferent records $(x_{i_1}, x_{i_2})$ s.t. $y_{i_1} \neq y_{i_2}$
    Set of blocking predicates $\mathcal{P} = \{p_1, \ldots, p_t\}$
    Maximum number of coreferent pairs allowed to be uncovered $\varepsilon$
    Maximum number of pairs that any predicate may cover $\eta$
    Maximum conjunction length, $k$
Output: A DNF blocking function based on $\mathcal{P}$:
        $(p_{i_1} \wedge \cdots \wedge p_{i'_1}) \vee \cdots \vee (p_{i_n} \wedge \cdots \wedge p_{i'_n})$, each $i'_j \leq k$
Method:
1. Discard from $\mathcal{P}$ all predicates $p_i$ for which $r(p_i) \geq \eta$:
   $\mathcal{P} \leftarrow \{p_i \in \mathcal{P} | r(p_i) \leq \eta\}$.
2. $\mathcal{P}^{(c)} = \emptyset$
3. For each $p_i \in \mathcal{P}$
4.     Construct $k-1$ candidate conjunctions $p_i^{(c)} = p_i \wedge \cdots \wedge p_{i_k}$
       by iteratively selecting $p_{i_j}$ that maximizes cover $b(p_i^{(c)})/r(p_i^{(c)})$,
       adding each $p_i^{(c)}$ to $\mathcal{P}^{(c)}$.
5. Return ApproxRBSetCover$(\mathcal{R}, \mathcal{B}, \mathcal{P} \cup \mathcal{P}^{(c)}, \varepsilon, \eta)$.
```

Figure 4. The algorithm for learning DNF blocking

two-stage procedure, shown in Figure 4.

First, a set of $t(k-1)$ predicate conjunctions of lengths between 2 and k is created in a greedy fashion in steps 3-4. Candidate conjunctions are constructed iteratively starting with each predicate $p_i \in \mathcal{P}$. At each step, another predicate is added to the current conjunction so that the ratio between the number of positives and the number of negatives covered by the conjunction is maximally improved. This leads to $k-1$ conjunctions being added to the candidate set $\mathcal{P}^{(c)}$ for each predicate. If a certain conjunction has already been added in a previous iteration, the conjunction with next-best coverage ratio is added instead.

After the candidate set of conjunctions $\mathcal{P}^{(c)}$ is constructed, the conjunctions are added to $\mathcal{P}$, the set of individual predicates. Then, the ApproxRBSetCover algorithm described in the previous section is used to learn a blocking function that corresponds to a DNF formula over the blocking predicates.

3.3 Blocking with the Learned Functions

Efficiency considerations, which are the primary motivation for this work, require the learned blocking functions to perform the actual blocking on new, unlabeled data in an effective manner. After the blocking function is learned using training data, it should be applied to the test data (for the actual linkage or clustering task) without explicitly constructing all pairs of records and evaluating the predicates on them. This is achieved by applying the indexing function for every blocking predicate or conjunction in the learned blocking function to every record in the test dataset. Thus, an inverted index is constructed for each predicate or conjunction in the blocking function. In each inverted index, every key is associated with a list of instances for which the indexing function of the corresponding predicate returns the key value. Disjunctive and DNF blocking can then be performed by iterating over every key in all inverted indices and returning all pairs of records that occur in the same list for any key.

The computational complexity as well as the reduction in the number of pairs due to disjunctive and DNF blocking are analogous to those for previously proposed blocking functions such as canopies [26] and key-based blocking [20]. If the indexing function for a selected predicate generates an average of f keys per record, and $|\mathcal{U}|$ total keys are produced, the computational complexity for applying each blocking predicate as well as the number of candidate pairs it produces is $O(f^2n^2/|\mathcal{U}|)$. Given that f is typically 1 or a small integer, while $|\mathcal{U}|$ is a large number, this represents a $O(f^2/|\mathcal{U}|)$ reduction in the number of candidate pairs from all pairs in a dataset. Computational complexity of blocking scales linearly with the number of selected predicates, which is typically small.

4 Experiments

4.1 Methodology and Datasets

We evaluate the effectiveness of the proposed methods for learning blocking functions using two metrics: reduction ratio and recall. These measures are defined with respect to the number of coreferent and non-coreferent record pairs that are covered by a blocking function $f_{\mathcal{P}}$ in a database of n records:

$$ReductionRatio = 1.0 - \frac{\sum_{(x_i,x_j)\in\mathcal{R}} f_{\mathcal{P}}(x_i,x_j) + \sum_{(x_i,x_j)\in\mathcal{B}} f_{\mathcal{P}}(x_i,x_j)}{n(n-1)/2}$$

$$Recall = \frac{\sum_{(x_i,x_j)\in\mathcal{B}} f_{\mathcal{P}}(x_i,x_j)}{|\mathcal{B}|}$$

Intuitively, recall captures blocking accuracy by measuring the proportion of truly coreferent record pairs that have been covered by the blocking function. an ideal blocking function would have recall of 1.0, indicating that all coreferent pairs are covered. Reduction ratio measures the efficiency gain due to blocking by measuring what proportion of all pairs in the dataset is filtered out by the blocking function. Without blocking, the reduction ratio is 0 since all record pairs are returned, while a higher number indicates what proportion of pairs is not covered, and therefore will not require similarity computations in the subsequent record linkage stages or in the clustering algorithm. Note that efficiency savings due to blocking are more substantial if collective (graph-based) inference methods are used for linkage or clustering [33, 25, 38, 2], as the time complexity of these methods increases superlinearly with the number of record pairs under consideration.

Results are obtained using 10 runs of two-fold cross-validation. Using a higher number of folds would result in fewer coreferent records in the test fold, which would artificially make the blocking task easier. During each run, the dataset is split into two folds by randomly assigning all records for every underlying entity to one of the folds. The blocking function is then trained using record pairs generated from the training fold. The learned blocking function is used to perform blocking on the test fold, based on which recall and reduction ratio are measured.

We present results on two datasets: *Cora* and *Addresses*. The *Cora* dataset contains 2191 5-field citations to 305 computer science papers. It was obtained by combining the multiple datasets used in [26], and removing records that are exact duplicates. While it is a relatively small-scale dataset, accurate linkage on this dataset requires computationally intensive string similarity functions and benefits from collective linkage methods, justifying the need for blocking [4, 26]. *Addresses* is a dataset containing names and addresses of 50,000 9-field records for 10,000 individuals that was generated using the DBGEN program provided by Hernandez and Stolfo [18]. We use the following general predicates are for constructing learnable blocking functions:

- *Exact Match*: covers instances that have the same value for the field;
- *Contain Common Token*: covers instances that contain a common token in the field value;
- *Contain Common Integer*: covers instances that contain a common token consisting of digits in the field value;
- *Contain Same or Off-by-One Integer*: covers instances that contain integer tokens that are equal or differ by at most 1;
- *Same n First Chars,* $n = 3,5,7$: covers instances that have a common character prefix in the field value;
- *Contain Common Token n-gram,* $n = 2,4,6$*:* covers instances that contain a common length-n contiguous subsequence of tokens;
- *Token-based TF-IDF* $> \delta$, $\delta = 0.2,0.4,0.6,0.8,1.0$*:* covers instances where token-based TF-IDF cosine similarity between field values is greater than the threshold δ;
- *n-gram-based TF-IDF* $> \delta$, $\delta = 0.2,0.4,0.6,0.8,1.0$, $n = 3,5$*:* covers instances where TF-IDF cosine similarity between n-gram representations of field values is greater than the threshold δ.

As described in Section 2.2, these general predicates are instantiated for all fields in the given database. Algorithms presented in Section 3 are used to construct blocking functions by selecting subsets of the resulting field-specific predicates. For DNF blocking, conjunctions of length 2 were employed, as experiments with longer conjunctions did not lead to improvements over blocking based on a 2-DNF.

We vary the value of the parameter ε (which specifies the number of coreferent pairs allowed to remain uncovered) by setting to $r\beta$ for different values of desired recall r between 0.0 and 1.0, where β is the number of coreferent record pairs in the training fold. This parameter captures the dependence between the reduction ratio and recall: if ε is high, fewer predicates are selected resulting in lower recall since not all coreferent pairs are retrieved. At the same time, the reduction ratio is higher for higher ε since fewer pairs are covered by the learned blocking function, leading to higher efficiency. By varying ε, we obtain a series of results that demonstrate the trade-off between obtaining higher recall and improving the reduction ratio.

We compare the proposed methods with CANOPIES [26], a popular blocking method relying on token-based or n-gram-based TF-IDF similarity computed using an inverted index. In a previous study, Baxter *et al.* [1] have compared several manually-tuned blocking strategies and found CANOPIES to produce the best overall results. CANOPIES also allows trading off recall and the reduction ratio by varying the threshold parameter that controls the coverage of the blocking.[1] We tried both token-based CANOPIES and tri-gram based CANOPIES and chose the best-performing variants as baselines: token-based indexing for *Cora*, and tri-gram indexing for *Addresses*. This difference is due to

[1]The original CANOPIES algorithm allows varying two separate threshold parameters, however, empirical results have shown that using the same value for both thresholds yields the best performance [26].

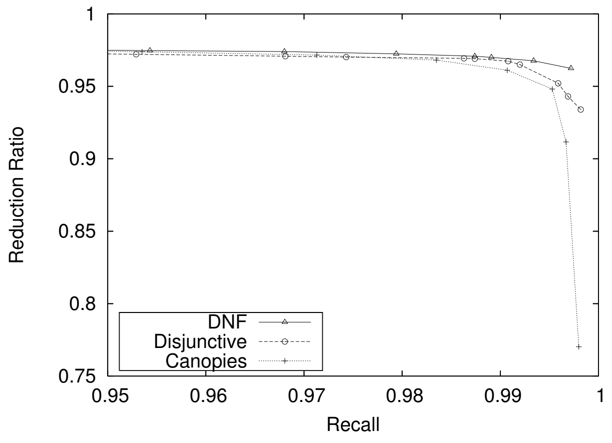

Figure 5. Blocking results for the *Cora* dataset

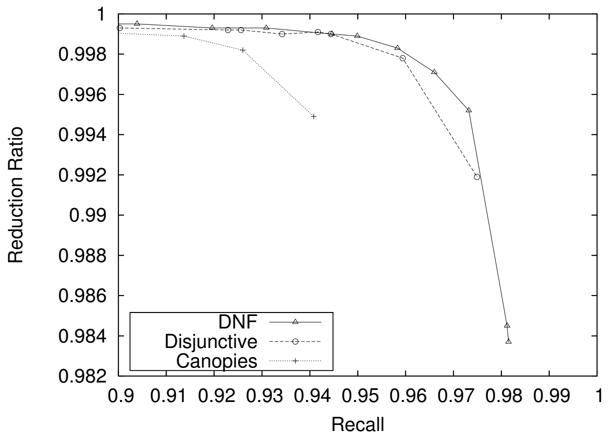

Figure 6. Blocking results for the *Addresses* dataset

the fact that most variation between coreferent citations in *Cora* is due to insertions and deletions of whole words, making token-based similarity more appropriate. Coreferent records in *Addresses*, on other hand, mostly differ due to misspellings and character-level transformations that n-gram similarity is able to capture.

4.2 Results and Discussion

Figures 5 and 6 show the reduction ratio versus recall curves for the two types of learned blocking functions described above and for CANOPIES. From these results, we observe that both variants of adaptive blocking outperform the unlearned baseline: combining multiple predicates allows achieving higher recall levels as well as achieving higher reduction ratios. DNF blocking is more accurate than disjunctive blocking, and on *Addresses* it also achieves higher recall, while for *Cora* the maximum recall is comparable. Because DNF blocking is based on predicate conjunctions, non-coreferent pairs are more easily avoided by the blocking function: conjunctions effectively form high-precision, low-recall rules that cover smaller subsets of coreferent pairs but fewer non-coreferent pairs compared to single predicates. While none of the methods achieve 100% recall (as it would effectively require returning all record pairs), for both datasets adaptive blocking is able to achieve higher recall than CANOPIES. Thus, using learnable blocking functions leads to *both* accuracy and efficiency improvements.

Table 2 shows the actual number of record pairs returned by the different blocking methods at highest achieved recall. These results demonstrate the significance of differences in the reduction ratio between the different blocking functions: because the total number of pairs is very large, differences in the reduction ratio translate into significant savings in the number of pairs for which similarity must be computed. Note that the smaller number of pairs returned by CANOPIES and disjunctive blocking on *Addresses* corresponds to a significantly lower recall, while for a fixed recall level DNF blocking either does as well or better.

	Cora	*Addresses*
DNF Blocking,	23,499	4,890,410
Disjunctive Blocking	41,439	4,090,283
Canopies	125,986	1,745,995
Total number of pairs	606,182	312,487,500

Table 2. Average number of pairs covered by the learned blocking functions at the highest achieved recall

	Cora	*Addresses*
DNF Blocking	26.9	735.81
Disjunctive Blocking	32.4	409.4
Canopies	16.0	572.7

Table 3. Average blocking time, CPU seconds

Table 3 shows the blocking times for the different methods measured at maximum achieved recall, where blocking time is defined as the total CPU time required by the blocking function to construct the blocks and generate the candidate pairs. These results show that learnable blocking functions incur a relatively modest increase in computational time despite the fact that they utilize many predicates. This is due to the fact that the learned predicates that cover few negatives typically require smaller inverted indices than the one built by canopies using tokens or n-grams, where each token or n-gram occurs in many strings. Many predicates employed by the adaptive blocking functions, on other hand, map each string to a single key, resulting in more efficient retrieval of covered pairs. Inverted indices corresponding to conjunctions are even more efficient as they contain many keys (the cross product of the key sets for the predicates in the conjunction) and thus produce smaller blocks. This results in better performance of DNF blocking compared to disjunctive blocking on *Cora*, where the number of predicates in the constructed blocking function is similar for the two methods. On *Addresses*, DNF blocking constructs functions containing more predicates, which on one hand incurs a computational penalty, but on the other hand leads to higher recall.

Overall, the results demonstrate that adaptive blocking functions significantly improve the efficiency of record linkage, and provide an attractive methodology for scaling up data mining tasks that rely on similarity computations between pairs of instances.

5 Related Work

A number of blocking methods have been proposed by researchers for speeding up record linkage and clustering [15, 22, 30, 20, 18, 26, 1, 6, 21, 17, 41].

Our predicate-based blocking approach is most closely related to key-based methods developed by researchers working on record linkage for Census data [22, 30, 20, 41]. These methods form blocks by applying an indexing function for a chosen predicate to each record and assigning all records that return the same value (key) to the same block [22, 20, 41]. While the predicates are typically chosen manually by trial and error, in recent work Winkler [41] proposed a capture-recapture methodology for evaluating the accuracy of individual blocking predicates. Integrating this approach within an adaptive blocking algorithm is an interesting avenue for future work.

Another popular blocking technique is the sorted neighborhood method proposed by Hernandez and Stolfo [18]. This method forms blocks by sorting the records in a database using lexicographic criteria and selecting all records that lie within a window of fixed size. Multiple sorting passes are performed to increase coverage.

The CANOPIES blocking algorithm of McCallum *et al.* [26] relies on a similarity function that allows efficient retrieval of all records within a certain distance threshold from a randomly chosen record. Blocks are formed by randomly selecting a "canopy center" record and retrieving all records that are similar to the chosen record within a certain ("loose") threshold. Records that are closer than a "tight" threshold are removed from the set of possible canopy centers, which is initialized with all records in the dataset. This process is repeated iteratively, resulting in formation of blocks selected around the canopy centers. Inverted index-based similarity functions such as Jaccard or TF-IDF cosine similarity are typically used with the canopies method as they allow fast selection of nearest neighbors based on co-occurring tokens. Inverted indices based on character *n*-grams are employed in the blocking methods described by Elfeky *et al.* [14], Chaudhuri *et al.* [6], and Christen and Churches [7].

Recently, Jin *et al.* [21] proposed a blocking method based on mapping database records to a low-dimensional metric space, where fast nearest-neighbor searching methods can be used. While this approach can be used with arbitrary similarity functions, it is computationally expensive compared to the sorting and index-based algorithms.

A key distinction between prior work and our approach is that previously described methods focus on improving blocking efficiency while assuming that an accurate blocking function is known and its parameters have been tuned manually. In contrast, our approach attempts to construct an optimal blocking function automatically. Because blocking functions can be learned using any combination of similarity predicates on different record fields, and no assumptions are made about the number of record fields or their type, our approach can be used for adapting the blocking function in any domain, while allowing human experts to add domain-specific predicates.

In parallel independent work, Michelson and Knoblock [27] have proposed an alternative method for automatically learning blocking functions in the context of record linkage. Their approach is similar to ours in that it learns predicate-based blocking functions in DNF form using an iterative covering algorithm. Experimental studies of the different variants of the overall iterative algorithm present an interesting issue for future research.

6 Future Work and Conclusions

In this paper, we have formalized the problem of adapting the blocking function to a given domain, described two types of blocking functions, and provided learning algorithms for training the blocking functions. In future work, we are interested in extending our adaptive blocking approach to other data mining tasks, such as clustering and schema mapping. An interesting open question is whether blocking can be used for kernel methods [36], where the need to construct a kernel matrix over all pairs of input instances along with the requirement that the matrix be positive semidefinite presents a significant obstacle to scaling up to large datasets. Another avenue for future research is comparing our approach to the use of rule learning algorithms such as RIPPER [9] for constructing blocking functions.

References

[1] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*.

[2] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *Proceedings of SDM-2006*.

[3] M. Bilenko, S. Basu, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proceedings of ICDM-2005*.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of KDD-2003*.

[5] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the Red-Blue Set Cover problem. In *Proceedings of SODA-2000*.

[6] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of SIGMOD-2003*.

[7] P. Christen and T. Churches. Febrl – freely extensible biomedical record linkage. http://datamining.anu.edu.au/linkage.html.

[8] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[9] W. W. Cohen. Fast Effective Rule Induction. In *Proceedings of ICML-1995*.

[10] W. W. Cohen, H. Kautz, and D. McAllester. Hardening soft information sources. In *Proceedings of KDD-2000*.

[11] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of KDD-2002*.

[12] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of WWW-2002*.

[13] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of SIGMOD-2005*.

[14] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. TAILOR: A record linkage tool box. In *Proceedings of ICDE-2002*.

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

[16] C. L. Giles, K. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In *Proceedings of ACM DL-1998*.

[17] L. Gu and R. Baxter. Adaptive filtering for efficient record linkage. In *Proceedings of SDM-2004*.

[18] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of SIGMOD-95*.

[19] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[20] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[21] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proceedings of DASFAA-2003*.

[22] R. P. Kelley. Advances in record linkage methodology: a method for determining the best blocking strategy. In *Record Linkage Techniques*, pages 199–203, Arlington, VA, 1985.

[23] S. Lawrence, K. Bollacker, and C. L. Giles. Autonomous citation matching. In *Proceedings of AGENTS-1999*.

[24] X. Li, P. Morie, and D. Roth. Robust reading: Identification and tracing of ambiguous names. In *Proceedings of NAACL-2004*.

[25] A. McCallum and B. Wellner. Conditional models of identity uncertainty with application to noun coreference. In *NIPS 17*, 2005.

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

[27] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of AAAI-2006*.

[28] S. N. Minton, C. Nanjo, C. A. Knoblock, M. Michalowski, and M. Michelson. A heterogeneous field matching method for record linkage. In *Proceedings of ICDM-2005*.

[29] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of SIGMOD DMKD*, 1997.

[30] H. B. Newcombe. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford, 1988.

[31] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS 13*, 2001.

[32] G. N. Norén, R. Orre, and A. Bate. A hit-miss model for duplicate detection in the WHO Drug Safety Database. In *Proceedings of KDD-2005*.

[33] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *NIPS 15*, 2003.

[34] D. Peleg. Approximation algorithms for the Label-CoverMAX and Red-Blue Set Cover problems. In *Proceedings of SWAT-2000*, LNCS 1851.

[35] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of KDD-2002*.

[36] B. Schölkopf and A. J. Smola. *Learning with kernels - support vector machines, regularization, optimization and beyond.* MIT Press, 2002.

[37] W. Shen, X. Li, and A. Doan. Constraint-based entity matching. In *Proceedings of AAAI-2005*.

[38] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *Proceedings of PKDD-2005*.

[39] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of KDD-2002*.

[40] W. E. Winkler. The state of record linkage and current research problems. Tech. report, Statistical Research Division, U.S. Census Bureau, Washington, DC, 1999.

[41] W. E. Winkler. Approximate string comparator search strategies for very large administrative lists. Tech. report, Statistical Research Division, U.S. Census Bureau, Washington, DC, 2005.