Learning CNF Blocking for Large-scale Author Name Disambiguation

Kunho Kim[†*], **Athar Sefid**[‡], **C. Lee Giles**[‡]
[†]Microsoft Corporation, Redmond, WA, USA
[‡] The Pennsylvania State University, University Park, PA, USA
kuki@microsoft.com, azs5955@psu.edu, clg20@psu.edu

Abstract

Author name disambiguation (AND) algorithms identify a unique author entity record from all similar or same publication records in scholarly or similar databases. Typically, a clustering method is used that requires calculation of similarities between each possible record pair. However, the total number of pairs grows quadratically with the size of the author database making such clustering difficult for millions of records. One remedy is a blocking function that reduces the number of pairwise similarity calculations. Here, we introduce a new way of learning blocking schemes by using a conjunctive normal form (CNF) in contrast to the disjunctive normal form (DNF). We demonstrate on PubMed author records that CNF blocking reduces more pairs while preserving high pairs completeness compared to the previous methods that use a DNF and that the computation time is significantly reduced. In addition, we also show how to ensure that the method produces disjoint blocks so that much of the AND algorithm can be efficiently paralleled. Our CNF blocking method is tested on the entire PubMed database of 80 million author mentions and efficiently removes 82.17% of all author record pairs in 10 minutes.

1 Introduction

Author name disambiguation (AND) refers to the problem of identifying each unique author entity record from all publication records in scholarly databases (Ferreira et al., 2012). It is also an important preprocessing step for a variety of problems. One example is processing author-related queries properly (e.g., identify all of a particular author's publications) in a digital library search engine. Another is to calculate author-related statistics such as an h-index, and collaboration relationships between authors.

Typically, a clustering method is used to calculate AND. Such clustering calculates pairwise similarities between each possible pairs of records that then determines whether each pair should be in the same cluster. Since the number of possible pairs in a database with the number of records n is $n(n-1)/2$, it grows as O(n^2). Since n can be millions of authors in some databases such as PubMed, AND algorithms need methods that scale, such as a blocking function (Christen, 2012). The blocking function produces a reduced list of candidate pairs, and only the pairs on the list are considered for clustering.

Blocking usually consists of blocking predicates. Each predicate is a logical binary function with a combination of an attribute and a similarity criterion. One example can be exact match of the last name. A simple but effective way of blocking involves manually selecting the predicates, with respect to the data characteristics. Much recent work on large-scale AND uses a heuristic that is the *initial match of first name* and *exact match of last name* (Torvik and Smalheiser, 2009; Liu et al., 2014; Levin et al., 2012; Kim et al., 2016). Although this gives reasonable completeness, it can be problematic when the database is extremely large, such as the author mentions in CiteSeerX (10M publications, 32M authors), PubMed (24M publications, 88M authors), and Web of Science (45M publications, 163M authors)[1].

The blocking results on PubMed using this heuristic are shown in Table 1. Note that most of the block sizes are less than 100 names, but a few blocks are extremely large. Since the number

[*]Work done while the author was at Pennsylvania State University

A shorter preprint version of this paper was published at arXiv (Kim et al., 2017)

[1]Numbers were as of 2016.

Proceedings of the First Workshop on Scholarly Document Processing, pages 72–80

https://doi.org/10.18653/v1/P17

Table 1: Block Size Distribution of PubMed author mentions using the simple blocking heuristic.

Block Size	Frequency	Percentage
$2 \leq n < 10$	1,586,677	59.91%
$10 \leq n < 100$	910,272	34.37%
$100 \leq n < 1000$	144,361	5.45%
$1000 \leq n < 10000$	6,998	0.26%
$10000 \leq n < 50000$	184	0.01%
$n \geq 50000$	9	$< 0.01\%$
Total	2,648,501	100.0 %

of pairs grows quadratically, those few blocks can dominate the computation time. This imbalance of the block size is due to the popularity of certain surnames, especially Asian names (Kim et al., 2016). To make matters worse, this problem increases in time, since the growth rates of publication records are rapidly increasing.

To improve the blocking, there has been work on learning the blocking (Bilenko et al., 2006; Michelson and Knoblock, 2006; Cao et al., 2011; Kejriwal and Miranker, 2013; Das Sarma et al., 2012; Fisher et al., 2015). These can be categorized into two different methods. One is a disjoint blocking, where each block is separated so each record belongs to a single block. Another is non-disjoint blocking, where some blocks have shared records. Each has advantages. Disjoint blocking can make the clustering step easily parallelized, while non-disjoint blocking often produces smaller blocks. and also has more degrees of freedom from which to select the similarity criterion.

Here, we propose to learn a non-disjoint blocking with a conjunctive normal form (CNF). Our main contributions are:

- Propose a CNF blocking, which reduces more pairs compared to DNF blocking, in order to achieve a large number of pairs completeness. This also reduces the processing time, which benefits various applications such as online disambiguation, author search, etc.
- Extend the method to produce disjoint blocks, so that the AND clustering step can be easily parallelized.
- Compare different gain functions, which are used to find the best blocking predicates for each step of learning.

Previous work is discussed in the next session. This is followed by problem definition. Next, we describe learning of CNF blocking and how to use it to ensure the production of disjoint blocks. Next, we evaluate our methods on the PubMed dataset. Finally, the last section consists of a summary work with possible future directions.

2 Related Work

Blocking has been widely studied for record linkage and entity disambiguation. Standard blocking is the simplest but most widely used method (Fellegi and Sunter, 1969). It is done by considering only pairs that meet all blocking predicates. Another is the sorted neighborhood approach (Hernández and Stolfo, 1995) which sorts the data by a certain blocking predicate, and forms blocks with pairs of those records within a certain window. Yan et al. (2007) further improved this method to adaptively select the size of the window. Aizawa and Oyama (2005) introduced a suffix array-based indexing method, which uses an inverted index of suffixes to generate candidate pairs. Canopy clustering (McCallum et al., 2000) generates blocks by clustering with a simple similarity measure and use loose & tight thresholds to generate overlapping clusters. Recent surveys (Christen, 2012; Papadakis et al., 2016, 2020) imply that there are no clear winners and proper parameter tuning is required for a specific task.

Much work optimized the blocking function for standard blocking. The blocking function is typically presented with a logical formula with blocking predicates. Two studies focused on learning a disjunctive normal form (DNF) blocking (Bilenko et al., 2006; Michelson and Knoblock, 2006) were published in the same year. Making use of manually labeled record pairs, they used a sequential covering algorithm to find the optimal blocking predicates in a greedy manner. Additional unlabeled data was used to estimate the reduction ratio of their cost function (Cao et al., 2011) while an unsupervised algorithm was used to automatically generate labeled pairs with rule-based heuristics used to learn DNF blocking (Kejriwal and Miranker, 2013).

All the work above proposed to learn non-disjoint blocking because of the logical *OR* terms in the DNF. However, other work learns the blocking function with a pure conjunction, to ensure the generation of disjoint blocks. Das et al. (2012) learns a conjunctive blocking tree, which has different blocking predicates for each branch of the

tree. Fisher et al. (2015) produces blocks with respect to a size restriction, by generating candidate blocks with a list of predefined blocking predicates and then performs a merge and split to generate the block with the desired size.

Our work proposes a method for learning a non-disjoint blocking function in a conjunctive normal form (CNF). Our method is based on a previous CNF learner(Mooney, 1995), which uses the fact that a CNF can be a logical dual of a DNF.

3 Problem Definition

Our work tackles the same problem with baseline DNF blocking (Bilenko et al., 2006; Michelson and Knoblock, 2006), but in a different way to get the optimized blocking function. Let $R = \{r_1, r_2, \cdots, r_n\}$ be the set of records in the database, where n is the number of records. Each record r has k attributes, and A be the attribute set $A = \{a_1.a_2, \cdots, a_k\}$. A blocking predicate p is a combination of an attribute a and a similarity function s defined to a. An example of s is exact string match of a. A blocking predicate can be seen as a logical binary function applied to each pair of records, so $p(r_x, r_y) = \{0, 1\}$, where $r_x, r_y \in R$. A blocking function f is a boolean logic formula consisting with blocking predicates $p_1, p_2, \cdots, p_n$, and each predicate is connected with either conjunction $\wedge$ or disjunction $\vee$. An example is $f_{example} = (p_1 \wedge p_2) \vee p_3$. Since it is made up of blocking predicates, $f(r_x, r_y) = \{0, 1\}$ for all $r_x, r_y \in R$.

The goal is to find an optimal blocking function f^* that covers a minimum number of record pairs while missing up to a fraction ε of total number of matching record pairs. To formalize it,

$$f^* = \underset{f}{\operatorname{argmin}} \sum_{(r_x, r_y) \in R} f(r_x, r_y) \quad \text{such that} \geq (1 - \varepsilon) \times |R^+| \tag{1}$$

where R^+ is set of matching record pairs.

4 Learning the Blocking Function

Here, we first briefly review DNF blocking and then introduce our CNF blocking function. This section describes the gain functions that select an optimal predicate term for each step in the CNF learner. Finally, we discuss an extension that ensures the production of disjunctive blocks.

Algorithm 1 DNF Blocking

function LEARNCONJTERMS(L, P, p, k)
  Let Pos be set of positive samples in L
  Let Neg be set of negative samples in L
  $Terms \leftarrow \{p\}$
  $CurTerm \leftarrow p$
  $i \leftarrow 1$
  while $i < k$ **do**
    Find $p_i \in P$ that maximizes gain function CALCGAIN($Pos, Neg, CurTerm \wedge p_i$)
    $CurTerm \leftarrow CurTerm \wedge p_i$
    Add $CurTerm$ to $Terms$
    $i \leftarrow i + 1$
  end while
  return $Terms$
end function

function LEARNDNF(L, P, k)
  $CandTerms \leftarrow \phi$
  for $p \in P$ **do**
    $Terms \leftarrow$LEARNCONJ(L, P, p, k)
    $CandTerms \leftarrow CandTerms \cup Terms$
  end for
  Let Pos be set of positive samples in L
  Let Neg be set of negative samples in L
  $DNF \leftarrow \phi$
  while $|Pos| > \varepsilon \times |Pos|$ **do**
    Find $T \in CandTerms$ that maximizes gain function CALCGAIN(Pos, Neg, T)
    if CALCGAIN(Pos, Neg, t) > 0 **then**
      $DNF \leftarrow DNF \vee T$
      Let $PosCov$ be all $l \in Pos$ that satisfies T
      Let $NegCov$ be all $l \in Neg$ that satisfies T
      $Pos \leftarrow Pos - PosCov$
      $Neg \leftarrow Neg - NegCov$
    else
      break loop
    end if
  end while
  return DNF
end function

4.1 DNF Blocking

DNF blocking was originally proposed by (Bilenko et al., 2006; Michelson and Knoblock, 2006). Given labeled pairs, these methods attempt to learn the blocking function in the form of a DNF, the disjunction (logical *OR*) of conjunction (logical AND) terms. Learning DNFs is known to be a NP-hard problem (Bilenko et al., 2006). Thus, an approximation algorithm was used to learn k-DNF blocking by using a sequential covering algorithm. k-DNF means each conjunction term has, at most, k predicates. Algorithm 1 shows the process of DNF blocking. Function LEARNDNF in lines 16-38 is the main part of the algorithm. It has 3 inputs which are the L labeled sample pairs, P blocking predicates, and k parameters of maximum predicates considered for each conjunction term.

First, the algorithm selects a set of candidate conjunction terms with at most k predicates. For each predicate p, it generates k candidate conjunction terms with the highest gain function. Using the candidate terms, the algorithm learns the blocking function by using a sequential covering algorithm. It sequentially selects a conjunction term, from the set of candidates, that has the maximum gain value on the remaining samples, and attaches it with logical *OR* to the DNF term. In each step, all samples covered by the selected conjunction term are removed. This process repeats until it covers the desired minimum amount of positive samples, or there is no candidate term that can further be improved.

4.2 CNF Blocking

CNF blocking can be learned with a small modification to DNF blocking. CNF can be presented as the entire negation of a corresponding DNF and vice versa based on De Morgan's laws. Using this, Mooney proposed CNF learning (Mooney, 1995), which is a logical dual of DNF learning. This motivated our CNF blocking method.

Algorithm 2 illustrates the proposed CNF blocking and has a similar structure to algorithm 1. Instead of running a sequential covering algorithm to cover all positive samples, CNF blocking tries to cover all negative samples using negated blocking predicates. In other words, a DNF formula is learned that is consistent with a negated predicate, which we designate negated DNF ($NegDNF$). $NegP$ is the negation of each predicate p in P. LEARNCNF gets 3 inputs, where L are labeled

Algorithm 2 CNF Blocking

1: **function** LEARNNEGCONJTERMS($L, NegP, p, k$)
2: Let Pos be set of positive samples in L
3: Let Neg be set of negative samples in L
4: $Terms \leftarrow \{p\}$
5: $CurTerm \leftarrow p$
6: $i \leftarrow 1$
7: **while** $i < k$ **do**
8: Find $p_i \in NegP$ that maximizes gain function CALCNEGGAIN($Pos, Neg, CurTerm \wedge p_i$)
9: $CurTerm \leftarrow CurTerm \wedge p_i$
10: Add $CurTerm$ to $Terms$
11: $i \leftarrow i + 1$
12: **end while**
13: **return** $Terms$
14: **end function**
15:
16: **function** LEARNCNF(L, P, k)
17: $CandTerms \leftarrow \phi$
18: Let $NegP$ is negation of each $p \in P$
19: **for** $p \in NegP$ **do**
20: $Terms \leftarrow$LEARNNEGCONJ($L, NegP, p, k$)
21: $CandTerms \leftarrow CandTerms \cup Terms$
22: **end for**
23: Let Pos be set of positive samples in L
24: Let Neg be set of negative samples in L
25: $NegDNF \leftarrow \phi$
26: **while** $|Pos| > (1 - \varepsilon) \times |Pos|$ **do**
27: Find $T \in CandNegTerms$ that maximizes gain function CALCNEGGAIN($Pos, Neg, Term$)
28: **if** CALCNEGGAIN(Pos, Neg, T) > 0 **then**
29: $NegDNF \leftarrow NegDNF \vee T$
30: Let $PosCov$ be all l in Pos that satisfies T
31: Let $NegCov$ be all l in Neg that satisfies T
32: $Pos \leftarrow Pos - PosCov$
33: $Neg \leftarrow Neg - NegCov$
34: **else**
35: break loop
36: **end if**
37: **end while**
38: $CNF \leftarrow \neg(NegDNF)$
39: **return** CNF
40: **end function**

sample pairs, P are blocking predicates, and k is maximum number of predicates in each term.

The algorithm first generates a set of negated candidate conjunction term $Terms$ from all p in $NegP$ (line 19-22). A dual of the original gain function CALCNEGGAIN selects a predicate for generating a negated candidate conjunction. Then, as in DNF blocking, the sequential covering algorithm is used to learn the negated DNF formula (line 26-37), which iteratively adds a negated conjunction term until it covers the desired number of samples. We select a negated conjunction term with a gain function, CALCNEGGAIN. Also, note that the termination condition of the loop (line 26) is when ε of total positive samples are covered with the learned $NegDNF$. This ensures that we miss less than ε of the total number of positive samples in the final CNF formula. After getting the final $NegDNF$, it is negated to get the desired CNF.

4.3 Gain Function

The gain function estimates the benefit of adding a specific term to the learned formula. It is used in two different places in the algorithm - when choosing the conjunction candidates (line 8) and when choosing a term from the candidates for each iteration (line 27-28). Previous methods have proposed different gain functions. Here we describe each and compare the results in the experiments. P, N is the total number of positive and negative samples, and p, n is the number of remaining positive and negative samples covered by the term.

4.3.1 Information Gain

Originally from Mooney's CNF learner (1995), it is the dual of the information gain of a DNF learner

$$gain_{CNF} = n \times \left[\log\left(\frac{n}{n+p}\right) - \log\left(\frac{N}{N+P}\right) \right]. \quad (2)$$

4.3.2 Ratio Between Positive and Negative Samples Covered

Bilenko et al. (2006) used this for DNF blocking. It calculates the ratio between the number of positives and the number of negatives covered. For CNF learning, we use its dual

$$gain_{CNF} = \frac{n}{p}. \quad (3)$$

4.3.3 Reduction Ratio

Michaelson and Knoblock (2006) used terms with the maximum reduction ratio (RR). In addition, they filter out all terms with pairwise completeness (PC) below threshold t. We use the dual of the original function used as a CNF, which is now

$$gain_{CNF} = \begin{cases} \frac{p+n}{P+N} & \text{if } \frac{n}{N} > t \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Algorithm 3 Disjoint CNF Blocking

1: **function** DISJOINTCNF($L, P_{disjoint}, P_{full}, k$)
2: $Conj \leftarrow$ LEARNCNF($L, P_{disjoint}, 1$)
3: Let L' be set of $l \in L$ satisfies $Conj$
4: $CNF \leftarrow$ LEARNCNF(L_{remain}, P_{full}, k)
5: $Blocks \leftarrow$ Apply $Conj$ to whole data
6: **for** $Block \in Blocks$ **do**
7: Let L'' be $l \in Block$ that satisfies CNF
8: Consider pairs in L'' only for clustering
9: **end for**
10: **end function**

4.4 Learning Disjoint Blocks

Disjoint blocking functions generate blocks for each record that resides in a single block; thus such blocks are mutually exclusive. It has the advantage that parallelization can be performed efficiently after applying the blocking by running processes for each blocks separately. A blocking function is disjoint if and only if it satisfies the following conditions: 1) it only consists of pure conjunction (logical *AND*), 2) all predicates use non-relative similarity measures. That is, measures that compare the absolute value of blocking key, e.g. exact match of first n characters.

DNF and CNF blocking are both non-disjoint blocking due to the condition 1 above. We introduce a simple extension to ensure our CNF blocking can produce disjoint blocks. This is done by first producing two blocking functions. The first function learns a blocking function with only conjunctions based on our CNF blocking method using $k = 1$ and a limited set of predicates with non-relative similarity measures. Then, CNF blocking is learned with our k-CNF method with the whole set of predicates for pairs remaining after applying 1-CNF (conjunction of single attributes).

We first apply the 1-CNF to the whole database to produce disjoint blocks. Then for each block, we apply the second k-CNF blocking function to filter out pairs not satisfies the k-CNF function. This is similar to applying a filter as in Gu and Baxter

Table 2: Summary of PubMed Benchmark Dataset

# Authors	# Mentions	# Total Pairs	# Matched Pairs
214	3,964	7,854,666	51,052

(2004) and Khabsa et al. (2015). While they use a heuristic, our method automatically learns the optimal one. Note that this method still produces a CNF since it combines conjunction terms and k-CNF with logical *AND*.

5 Experiments

5.1 Benchmark Dataset

We use the PubMed to evaluate these methods. PubMed is a public large-scale scholarly database maintained by the National Center for Biotechnology Information (NCBI) at the National Library of Medicine (NLM). We use NIH principal investigator (PI) data for evaluation, which include PI IDs and corresponding publications. We randomly picked 10 names from the most frequent ones in the dataset and manually verified that all publications belong to each PI. The set of names include C* Lee, J* Chen, J* Smith, M* Johnson, M* Miller, R* Jones, S* Kim, X* Yang, Y* Li, Y* Wang, where C* means any name starts with C.

Table 2 shows the statistics of the dataset. Experiments are done with 5-fold cross validation.

5.2 Methodology

5.2.1 Evaluation Metrics

We evaluate our CNF blocking with reduction ratio (RR), pairs completeness (PC), and F-measure. These metrics are often used to evaluate blocking methods. Those metrics can be calculated as follows:

$$RR = 1 - \frac{p+n}{P+N}, \tag{5}$$

$$PC = \frac{p}{P}, \tag{6}$$

$$F = \frac{2 \times RR \times PC}{RR + PC}. \tag{7}$$

where P, N are the numbers of positive and negative samples, and p, n are the numbers of positive and negative samples covered with the blocking function. RR measures the efficiency of the blocking function, PC measures the quality of the blocking function. F is the harmonic mean of RR and PC.

Table 3: Blocking Predicates Used for Learning Non-Disjoint Blocking Function

Blocking Key	Similarity Criterion
First Name	$exact, first(n), last(n), compatible$
Last Name	$exact, first(n), last(n), compatible$
Middle Name	$exact, first(n), last(n), compatible$
Title	cos
Affiliation	$exact, cos, compatible$
Coauthor	cos
Order	$order$
Year	$exact, digit, diff$
Venue	$exact, cos$

5.2.2 Blocking Predicates Used

We first define the similarity criterion used for the experiments. We observed an important characteristic of the data: some attributes are empty (e.g. year: 7.8%, affiliation: 81.1%) or have only partial information (54.5% has only initials for the first name). To deal with this, we add $compatible$ to those blocking keys. Below is brief explanation of each similarity criterion.

- $exact$: Exact match.
- $first(n), last(n)$: First/Last n character match, where n is an integer. We check $\{1, 3, 5, 7\}$ for name attributes.
- $order$: Assigns $True$ if both records are first authors, last authors, or non-first and non-last authors.
- $digit(n)$: First n digit match. We check $\{1, 2, 3\}$ for year.
- $compatible$: $True$ if at least one of the records are empty (Eq. 8). If the key is name, it also checks if the initial matches if one of the records has only initial.

$$compatible(A, B) = \begin{cases} True & \text{if at least one is empty} \\ exact(A, B) & \text{otherwise} \end{cases} \tag{8}$$

- cos: Cosine distance of TF-IDF bag-of-words vector. We check with threshold $\{0.2, 0.4, 0.6, 0.8\}$.
- $diff$: Year difference. We use the threshold $\{2, 5, 10\}$.

Using those similarity measures, We define two different sets of blocking predicates. Table 3 shows

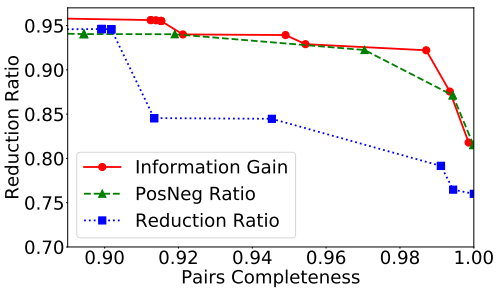

Figure 1: Gain Functions for PC–RR where PosNeg Ratio is the ratio between positive and negative samples covered.

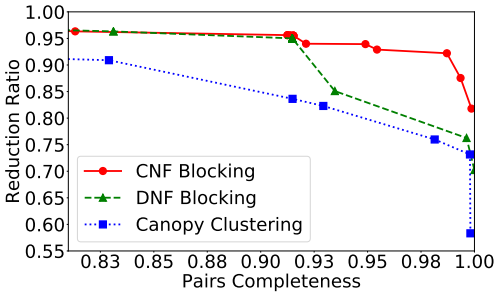

Figure 2: PC–RR for non-disjoint blocking methods

blocking predicates used for non-disjoint blocking. Disjoint blocking requires the use of predicates with non-relative similarity measures to ensure blocks are mutually exclusive. For disjoint blocking, we use the set of blocking predicates excluding the ones with the relative similarity measures ($exact$, $compatible$, $diff$) in Table 3.

5.2.3 Parameter Setting

The parameter ε is used to vary the PC. We tested values in $[0, 1]$ to get the PC–RR curve. k is selected experimentally to calculate the maximum reachable F-measure. We use $k = 3$ for further experiments.

5.3 Experiments

5.3.1 Gain Function

Figure 1 shows the PC–RR curve tested on three different gain functions. Blocking usually requires a high PC, so that we do not lose matched pairs after it is applied. As such, we focused on experiments with high PC values. As we can see from the results, information gain has highest RR overall. Thus, we use it as the gain function for the rest of the experiments.

5.3.2 Non-disjoint CNF Blocking

We compare non-disjoint CNF blocking with the DNF blocking (Bilenko et al., 2006; Michelson and Knoblock, 2006) and canopy clustering (McCallum et al., 2000). We used the set of Jaro–Winkler distance attributes for canopy clustering. Figure 2 shows the PC–RR curve for each method. Both CNF and DNF were better than canopy clustering, as was shown in Bilenko et al. (2006). CNF and DNF results are comparable for lower PC values. However, for high PC (>0.9) values, CNF has a better RR. We also tested another dataset used in (Khabsa et al., 2015). For PC=0.99, RR for CNF blocking was 0.882 while DNF blocking was 0.745.

We believe this is due to certain characteristics of scholarly databases. As discussed on the previous section, some attributes are empty for some records. DNF learns a blocking function by adding conjunction terms to gradually cover positive pairs. Although the proposed similarity criterion $compatible$ could catch positive pairs with empty attributes, it allows many negative pairs to pass the criterion, which makes the RR low. On the other hand, CNF learns a blocking function to cover (and filter out) negative pairs gradually. Negative pairs are much more obvious to define (pairs with different values), which makes the CNF more effective.

Another advantage of using CNF is the processing time. Fast processing time to apply blocking is important for some applications, one example is when we do a online disambiguation (Khabsa et al., 2015), another is to do an author search which requires to find the relevant cluster quickly (Kim et al., 2018). We measured the average processing time of applying each blocking method at high PC (PC=0.99), CNF blocking, DNF blocking, canopy clustering took 1.39s, 2.09s, 0.44s respectively. Canopy clustering was the fastest but generally we saw from the Figure 2 that its RR is much lower in high PC. CNF blocking has a faster processing time compared to DNF blocking. This is because CNF is composed with conjunctions, so it can quickly reject pairs that are not consistent with any terms. On the other hand, DNF consists of disjunction terms, so each pair should check all terms to make the decision. Learned CNF is also simpler than DNF. Learned CNF at this level is as below (fn, mn, ln is first, middle, last name respectively):

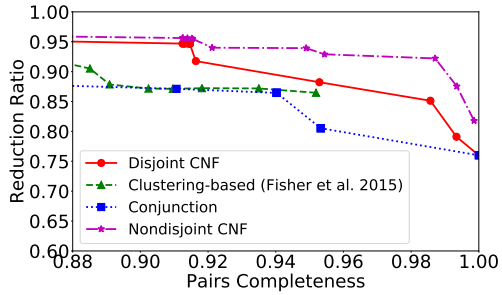

Figure 3: PC–RR for disjoint blocking methods

{(*fn*,*first*(5))∨(*fn*,*compatible*)∨(*coauth*,*cos*(0.8))}
∧{(*ln*,*exact*)}
∧{(*mn*,*compatible*)}
∧{((*fn*,*first*(3))∨(*fn*,*compatible*))}

And learned DNF is:
{((*coauth*,*cos*(0.8))∧(*ln*,*exact*)∧(*mn*,*compatible*))}
∨{((*venue*,*cos*(0.4))∧(*mn*,*first*(1)∧(*fn*,*compatible*)}
∨{(*fn*,*compatible*)∧(*mn*,*first*(1)∧(*ln*,*exact*)}
∨{((*venue*,*cos*(0.8))∧(*fn*,*exact*)}

In addition, we observed that proposed *compatible* predicate was frequently used in our result. This shows the effectiveness of *compatible* in dealing with the empty value.

5.3.3 Extension to Disjoint CNF Blocking

We evaluate our extension to disjoint blocks with CNF blocking. We compare the blocking learned with a pure conjunction, our proposed method, and the method of Fisher et al. (2015).

Figure 3 shows the reduction ratio pair completion (RR–PC) curve for each method. We also plot the original non-disjoint CNF blocking for comparison. We see that our proposed disjoint CNF blocking is the best amongst all disjoint methods. Fisher's method produced nearly uniform-sized blocks, but had limitations in reaching a high PC and had a generally lower RR compared to our method. Disjoint CNF didn't perform as well when compared to non-disjoint CNF because it is forced to use a pure conjunction on its first step. However, this simple extension easily helps parallelize the clustering process, so that the algorithm scales better. Testing our method to all of PubMed, 82.17% of the pairs are created in 10.5 min with 24 threads. Parallelization is important for disambiguation algorithms to scale to PubMed size scholarly databases (Khabsa et al., 2014).

Processing time for disjoint CNF blocking comparable to the original non-disjoint CNF blocking. The learned disjoint CNF is:
{(*fn*,*first*(1))} ∧ {(*ln*,*exact*)} ∧
{(*fn*,*compatible*)∨(*coauth*,*cos*(0.8))} ∧
{(*mn*,*compatible*)}

First two terms are from 1-CNF, and others from 3-CNF learner. We also tested this function to the whole PubMed.

6 Conclusion

We show how to learn an efficient blocking function with a conjunctive normal form (CNF) of blocking predicates. Using CNF as a negation of the corresponding disjunctive normal form (DNF) of predicates (Mooney, 1995), our method is a logical dual of existing DNF blocking methods (Bilenko et al., 2006; Michelson and Knoblock, 2006). We find that our method reduces more pairs for a large number of target pairs completeness and has a faster run time. We devise an extension that ensures that our CNF blocking produces disjoint blocks. Thus, the clustering process can be efficiently parallelized.

Future work could use multiple levels of blocking functions for processing each block (Das Sarma et al., 2012) and using linear programming to find an optimal CNF (Su et al., 2016).

7 Acknowledgement

We gratefully acknowledge partial support from the National Science Foundation and the National Bureau of Economic Research and useful discussions with Bruce Weinberg.

References

Akiko Aizawa and Keizo Oyama. 2005. A fast linkage detection scheme for multi-source information integration. In *International Workshop on Challenges in Web Information Retrieval and Integration*, pages 30–39.

Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. 2006. Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the 6th IEEE International Conference on Data Mining(ICDM'06)*, pages 87–96.

Yunbo Cao, Zhiyuan Chen, Jiamin Zhu, Pei Yue, Chin-Yew Lin, and Yong Yu. 2011. Leveraging unlabeled data to scale blocking for record linkage. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 22, page 2211.

Peter Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(9):1537–1555.

Anish Das Sarma, Ankur Jain, Ashwin Machanavajjhala, and Philip Bohannon. 2012. An automatic blocking mechanism for large-scale de-duplication tasks. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM)*, pages 1055–1064.

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

Anderson A. Ferreira, Marcos André Gonçalves, and Alberto H.F. Laender. 2012. A brief survey of automatic methods for author name disambiguation. *Acm Sigmod Record*, 41(2):15–26.

Jeffrey Fisher, Peter Christen, Qing Wang, and Erhard Rahm. 2015. A clustering-based framework to control block sizes for entity resolution. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 279–288.

Lifang Gu and Rohan Baxter. 2004. Adaptive filtering for efficient record linkage. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 477–481.

Mauricio A Hernández and Salvatore J Stolfo. 1995. The merge/purge problem for large databases. In *ACM Sigmod Record*, volume 24, pages 127–138.

Mayank Kejriwal and Daniel P Miranker. 2013. An unsupervised algorithm for learning blocking schemes. In *Proceedings of the IEEE 13th International Conference on Data Mining (ICDM)*, pages 340–349.

Madian Khabsa, Pucktada Treeratpituk, and C. Lee Giles. 2014. Large scale author name disambiguation in digital libraries. In *IEEE International Conference on Big Data*, pages 41–42.

Madian Khabsa, Pucktada Treeratpituk, and C. Lee Giles. 2015. Online person name disambiguation with constraints. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries(JCDL'15)*, pages 37–46.

Kunho Kim, Madian Khabsa, and C. Lee Giles. 2016. Random forest dbscan clustering for uspto inventor name disambiguation and conflation. In *IJCAI-16 Workshop on Scholarly Big Data: AI Perspectives, Challenges, and Ideas*.

Kunho Kim, Athar Sefid, and C Lee Giles. 2017. Scaling author name disambiguation with cnf blocking. *arXiv preprint arXiv:1709.09657*.

Kunho Kim, Athar Sefid, and C. Lee Giles. 2018. A web service for author name disambiguation in scholarly databases. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 265–273.

Michael Levin, Stefan Krawczyk, Steven Bethard, and Dan Jurafsky. 2012. Citation-based bootstrapping for large-scale author disambiguation. *Journal of the American Society for Information Science and Technology*, 63(5):1030–1047.

Wanli Liu, Rezarta Islamaj Doğan, Sun Kim, Donald C Comeau, Won Kim, Lana Yeganova, Zhiyong Lu, and W John Wilbur. 2014. Author name disambiguation for pubmed. *Journal of the Association for Information Science and Technology*, 65(4):765–781.

Andrew McCallum, Kamal Nigam, and Lyle H Ungar. 2000. 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*, pages 169–178.

Matthew Michelson and Craig A Knoblock. 2006. Learning blocking schemes for record linkage. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pages 440–445.

Raymond J Mooney. 1995. Encouraging experimental results on learning cnf. *Machine Learning*, 19(1):79–92.

George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42.

George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695.

Guolong Su, Dennis Wei, Kush R Varshney, and Dmitry M Malioutov. 2016. Learning sparse two-level boolean rules. In *Proceedings of the IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6.

Vetle I Torvik and Neil R Smalheiser. 2009. Author name disambiguation in medline. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(3):11.

Su Yan, Dongwon Lee, Min-Yen Kan, and Lee C Giles. 2007. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 185–194.