Fast Matching for All Pairs Similarity Search

Amit Awekar and Nagiza F. Samatova
North Carolina State University, Raleigh, NC, USA.
Oak Ridge National Laboratory, Oak Ridge, TN, USA.
Email: acwekar@ncsu.edu, samatovan@ornl.gov

Abstract

All pairs similarity search is the problem of finding all pairs of records that have a similarity score above the specified threshold. Many real-world systems like search engines, online social networks, and digital libraries frequently have to solve this problem for datasets having millions of records in a high dimensional space, which are often sparse. The challenge is to design algorithms with feasible time requirements. To meet this challenge, algorithms have been proposed based on the inverted index, which maps each dimension to a list of records with non-zero projection along that dimension. Common to these algorithms is a three-phase framework of data preprocessing, pairs matching, and indexing. Matching is the most time-consuming phase. Within this framework, we propose fast matching technique that uses the sparse nature of real-world data to effectively reduce the size of the search space through a systematic set of tighter filtering conditions and heuristic optimizations. We integrate our technique with the fastest-to-date algorithm in the field and achieve up to $6.5X$ speed-up on three large real-world datasets.

1. Introduction

Many real-world systems frequently have to search for all pairs of records with similarity above the specified threshold. This problem is referred as the *all pairs similarity search* ($APSS$). Similarity between two records is defined via some similarity measure, such as the cosine similarity or the Tanimoto coefficient. For example, similar web pages are suggested by Web search engines to improve user experience [1]; and similar users in an online social network are potential candidates for new friendship and collaboration [2].

$APSS$ is a compute-intensive problem. Given a dataset with n records in a d-dimensional space, where $n \leq d$, the naïve algorithm for the $APSS$ will compute the similarity between all pairs in $O(n^2 \cdot d)$ time. Such a solution is not practical for datasets with millions of records, which are typical in real-world applications. Therefore, many $heuristic$ solutions based on hashing [3], or dimensionality reduction [4] have been proposed for $APSS$.

Nagiza F. Samatova is the corresponding author.

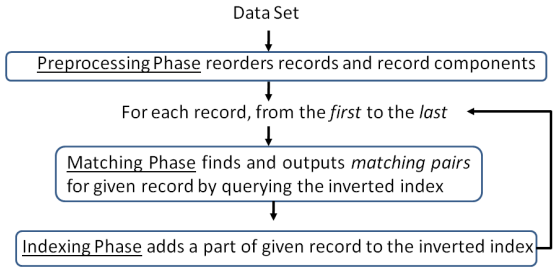

Figure 1. Overview of the framework common across recent exact algorithms for the APSS

However, recent $exact$ algorithms for the $APSS$ [5], [6], [2], [1] have performed even faster than the heuristic methods because of their ability to significantly prune the similarity score computation by taking advantage of the fact that only a small fraction of all $\Theta(n^2)$ pairs typically satisfy the specified similarity threshold. These exact algorithms depend on the *inverted index*, which maps each dimension to the list of records with non-zero projection along that dimension.

We observe that the exact algorithms based on the inverted index share a common three-phase framework of data preprocessing, pairs matching and indexing (please, refer to Figure 1 for the framework overview). The matching phase dominates the computational time. It searches for similar pairs in the inverted index and computes similarity score of pairs found. Therefore, improving the performance of any solution to the $APSS$ would require optimization of these two tasks in the matching phase.

In this paper, within the observed framework, we present the fast matching technique that reduces the search space; the size of the search space is defined as the actual number of record pairs evaluated by the algorithm. The proposed matching incorporates (a) the lower bound on the number of non-zero components in any record and (b) the upper bound on the similarity score for any record pair. The former allows for reducing the number of pairs that need to be evaluated, while the latter prunes, or only partially computes, the similarity score for many candidate pairs. Both bounds require only constant computation time. We integrate our fast matching technique with the fastest-to-date All_Pairs

algorithm [2] to derive the proposed $AP\_Time\_Efficient$ algorithm.

Finally, we conduct extensive empirical studies using three real-world million record datasets derived from information on: (a) scientific literature collaboration in Medline indexed papers[1], (b) social networks from Flickr[2], and (c) social networks from LiveJournal[3]. We compare the performance of our $AP\_Time\_Efficient$ algorithm against the All_Pairs algorithm [2] using the well-known cosine similarity. In our experiments, our fast matching technique reduces the search space by at least an order of magnitude, and achieve up to $6.5X$ speed up.

The rest of the paper is organized as follows. Definitions and notations are stated in Section 2. The common framework and related work are explained in Section 3. The fast matching technique and the corresponding $AP\_Time\_Efficient$ algorithm are described in Section 4. The datasets and experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Definitions and Notations

In this section, we define the problem and other important terms referred throughout the paper (please, see Table 1 for the summary of notations).

Definition 1 (*All Pairs Similarity Search*): The all pairs similarity search ($APSS$) problem is to find all pairs (x, y) and their exact value of similarity $sim(x, y)$ such that $x, y \in V$ and $sim(x, y) \geq t$, where

- V is a set of n real valued, non-negative, sparse vectors over a finite set of dimensions D; $|D| = d$;
- $sim(x, y) : V \times V \rightarrow [0, 1]$ is a symmetric similarity function; and
- t, $t \in [0, 1]$, is the similarity threshold.

Definition 2 (*Inverted Index*): The inverted index maps each dimension to a list of vectors with non-zero projection along that dimension. A set of all d lists $I = \{I_1, I_2, \ldots, I_d\}$, i.e., one for each dimension, represents the inverted index for V. Each entry in the list has a pair of values (x, w) such that if $(x, w) \in I_k$, then $x[k] = w$. The inverse of this statement is not necessarily true, because some algorithms index only a part of each vector.

Definition 3 (*Candidate Vector* and *Candidate Pair*): Given a vector $x \in V$, any vector y in the inverted index is a candidate vector for x, if $\exists\ j$ such that $x[j] > 0$ and $(y, y[j]) \in I_j$. The corresponding pair (x, y) is a candidate pair.

Definition 4 (*Matching Vector* and *Matching Pair*): Given a vector $x \in V$ and the similarity threshold t, a candidate vector $y \in V$ is a matching vector for x, if $sim(x, y) \geq t$. We say that y matches with x, and vice versa. The corresponding pair (x, y) is a matching pair.

1. http://www.nlm.nih.gov/pubs/factsheets/medline.html
2. http://www.flickr.com
3. http://www.livejournal.com

Table 1. Notations Used

Notation	Meaning
Given a dimension j	
$density(j)$	the number of vectors in V with non-zero projection along the dimension j
$global\_max\_weight[j]$	$x[j]$ such that $x[j] \geq y[j]$ for $\forall y \in V$
Given a vector x	
$x.max\_weight$	$x[k]$ such that $x[k] \geq x[i]$ for $1 \leq i \leq d$
$x.sum$	$\sum_{i=1}^{d} x[i]$
x'	the unindexed part of x
x''	the indexed part of x
$\|x\|$ (size of x)	the number of nonzero components in x
$\|\|x\|\|$ (magnitude of x)	$\sqrt{\sum_{i=1}^{d} x[i]^2}$
Given a pair of vectors (x, y)	
$dot(x, y)$	$\sum_{i} x[i] \cdot y[i]$
$cos(x, y)$	$dot(x, y)/(\|\|x\|\| \cdot \|\|y\|\|)$

During subsequent discussions we assume that all vectors are of unit length ($||x|| = ||y|| = 1$), and the similarity function is the cosine similarity. In this case, the cosine similarity equals the dot product, namely:

$$sim(x, y) = cos(x, y) = dot(x, y).$$

Our algorithm can be extended to the Tanimoto coefficient using simple conversion derived by Bayardo *et al.* [2].

3. Common Framework

The basic idea behind the exact $APSS$ algorithms based on the inverted index is similar to the way information retrieval systems answer queries [7]. Every vector in the dataset is considered as a query and the corresponding matching pairs are found using the inverted index. Most of the time, however, the information retrieval system requires only top-k similar pairs, while the $APSS$ requires all matching pairs. The framework can be broadly divided into three phases: data preprocessing, pairs matching, and indexing (please, refer to Algorithm 1 for details).

3.1. Preprocessing

The preprocessing phase reorders vectors using a permutation Ω defined over V (lines 1–5, Algorithm 1). Bayardo *et al.* [2] and Xiao *et al.* [1] sorted vectors on the maximum value within each vector. Sarawagi *et al.* [5] sorted vectors on their size. The components within each vector are also rearranged using a permutation Π defined over D. Bayardo *et al.* [2] observed that sorting the dimensions in D based on

Algorithm 1: Inverted index based framework common across recent exact $APSS$ algorithms

Input: V, t, D, sim, Ω, Π
Output: $MATCHING\_PAIRS\_SET$

1 $MATCHING\_PAIRS\_SET = \emptyset$;
2 $I_i = \emptyset \ , \forall \ 1 \leq i \leq d$;
3 Arrange vectors in V in the order defined by Ω;
4 Arrange components in each vector in the order defined by Π;
5 Compute $summary\_statistics$;
6 **foreach** $x \in V$ *using the order defined by* Ω **do**
7 C = set of candidate pairs corresponding to x, found by querying and manipulating the inverted index I;
8 **foreach** *candidate pair* $(x, y) \in C$ **do**
9 $sim\_max\_possible$ = upper bound on $sim(x, y)$;
10 **if** $sim\_max\_possible \geq t$ **then**
11 $sim\_actual = sim(x, y)$;
12 **if** $sim\_actual \geq t$ **then**
13 $MATCHING\_PAIRS\_SET = MATCHING\_PAIRS\_SET \cup (x, y, sim\_actual)$
14
15
16 **foreach** i *such that* $x[i] > 0$ *using the order defined by* Π **do**
17 **if** $filtering\_condition(x[i])$ *is true* **then**
18 Add $(x, x[i])$ to the inverted index;
19
20
21 **return** $MATCHING\_PAIRS\_SET$

vector density speeds up the $APSS$. The summary statistics about each record, such as its size, magnitude, and maximum component value are computed during the preprocessing phase. They are used later to derive filtering conditions during the matching and indexing phases to save time and memory. The time spent on preprocessing is negligible compared to the time spent on matching.

3.2. Matching

The matching phase scans the lists in the inverted index that correspond to the non-zero dimensions in x, for a given vector $x \in V$, to find candidate pairs (lines 7–15, Algorithm 1). Simultaneously, it accumulates a partial similarity score for all candidate pairs. Bayardo *et al.* [2] and Xiao *et al.* [1] used the hash-based map, while Sarawagi *et al.* [5] used the heap-based scheme for score accumulation.

Given t, Ω, Π and $summary\_statistics$, various filtering conditions are derived to eliminate candidate pairs that will definitely not satisfy the required similarity threshold; these pairs are not added to the set C (line 7, Algorithm 1). Sarawagi *et al.* [5] identified the part of the given vector $x \in V$ such that for any candidate vector $y \in V$ to have $sim(x, y) \geq t$, the intersection of y with that part must be non-empty. Bayardo *et al.* [2] computed a lower bound on the size of any candidate vector to match with the current vector as well as any remaining vector. Our fast matching technique further tightens this lower bound.

Some of the candidate pairs can be safely discarded by computing an upper bound on the similarity score. Xiao *et al.* [1] used the Hamming distance based method for computing such an upper bound. But their technique and formulation of the APSS problem is specific to binary vectors only. Bayardo *et al.* [2] used the vector size and maximum component value to derive a constant time upper bound. We further tighten this upper bound in our fast matching technique. Finally, the exact similarity score is computed for the remaining candidate pairs, and those having scores above the specified threshold are added to the output set.

3.3. Indexing

The indexing phase adds a part of the given vector to the inverted index so that it can be matched with any of the remaining vectors (lines 16–20 Algorithm 1). Sarawagi *et al.* [5] unconditionally indexed every component of each vector. Instead of building the inverted index incrementally, they built the complete inverted index beforehand. Bayardo *et al.* [2] and Xiao *et al.* [1] used the upper bound on the possible similarity score with only the part of the current vector. Once this bound reached the similarity threshold, the remaining vector components were indexed.

4. Fast Matching

In the framework described in Algorithm 1, the matching phase searches and evaluates $O(n^2)$ candidate pairs. This phase is critical for the running time. The computation time of the matching phase can be reduced in two ways:

- generating fewer candidate pairs for evaluation, and
- reducing the number of candidate pairs that are evaluated completely.

Our tighter lower bound on the size of the candidate vector reduces the number of candidate pairs that are being generated. Our tighter upper bound on the similarity score reduces the number of candidate pairs that are being evaluated completely. First, we will prove the correctness of these bounds, and later we will show that they are tighter than the existing bounds.

4.1. Upper Bound on the Similarity Score

Given a candidate pair (x, y), the following constant time upper bound on the cosine similarity holds:

$$cos(x, y) \leq x.max\_weight * y.sum. \quad (1)$$

The correctness of this upper bound can be derived from:

$$x.max\_weight * y.sum \geq dot(x, y) = cos(x, y).$$

Similarly following upper bound can be derived:

$$cos(x, y) \leq y.max\_weight * x.sum. \quad (2)$$

Combining upper bounds in 1 and 2, we propose following upper bound on the cosine similarity score:

$$min(x.max\_weight * y.sum,\ y.max\_weight * x.sum), \quad (3)$$

where the min function selects the minimum of the two arguments.
We can safely prune the exact dot product computation of a candidate pair, if it does not satisfy the similarity threshold even for this upper bound.

4.2. Lower Bound on the Candidate Vector Size

For any candidate pair (x, y), the following is true:

$$x.max\_weight * y.sum \geq dot(x, y).$$

For this candidate pair to qualify as a matching pair, the following inequality must hold:

$$y.sum \geq t/x.max\_weight.$$

For any unit-length vector y, the following is true:

$$y.sum \geq k \rightarrow |y| \geq k^2.$$

This gives us the following lower bound on the size of any candidate vector y:

$$|y| \geq (t/x.max\_weight)^2. \quad (4)$$

The lower bound on the size of the candidate vector avoids generating the candidate pairs that will not satisfy the similarity threshold.

4.3. AP_Time_Efficient Algorithm

The proposed $AP\_Time\_Efficient$ algorithm integrates both optimizations for the lower and upper bounds with the fastest-to-date All_Pairs algorithm [2] (please, refer to Algorithm 2). The vectors are sorted in decreasing order of their max_weight, and the dimensions are sorted in decreasing order of their vector density. For every $x \in V$, the algorithm first finds its matching pairs from the inverted index (Matching Phase, Lines 6–22) and then adds selective parts of x to the inverted index (Indexing Phase, Lines 23–29). The difference between the algorithms $AP\_Time\_Efficient$ and All_Pairs is in the tighter bounds on filtering conditions.

4.4. Tighter Bounds

The All_Pairs algorithm [2] uses $t/x.max\_weight$ as the lower bound on the size of any candidate vector. The $AP\_Time\_Efficient$ algorithm tightens this bound by squaring the same ratio (Line 8, Algorithm 2).

The All_Pairs algorithm uses the following constant time upper bound on the dot product of any candidate pair (x, y):

$$min(|x|, |y|) * x.max\_weight * y.max\_weight. \quad (5)$$

We will consider following two possible cases to prove that our bound proposed in (3) is tighter than this bound.

- Case 1: $|x| \leq |y|$
  The upper bound in 5 reduces to:
  $$|x| * x.max\_weight * y.max\_weight.$$
  The upper bound proposed in (2) is tighter than this bound because:
  $$x.sum \leq |x| * x.max\_weight.$$
- Case 2: $|x| > |y|$
  The upper bound in 5 reduces to:
  $$|y| * x.max\_weight * y.max\_weight.$$
  The upper bound proposed in (1) is tighter than this bound because:
  $$y.sum \leq |y| * y.max\_weight.$$

4.5. Effect on Matching Phase

For a given vector x, our proposed lower bound on the size of a candidate vector is inversely proportional to $x.max\_weight$. The vectors are sorted in decreasing order by max_weight. Hence, the value of the lower bound on the size of the candidate increases monotonically as the vectors are processed. $AP\_Time\_Efficient$ generates fewer candidate pairs as fewer vectors qualify to be candidate vectors because of the tighter lower bound on their size.

Computing the exact dot product for a candidate pair requires linear traversal of both vectors in the candidate pair (Line 20, Algorithm 2). The tighter constant time upper bound on the similarity score of a candidate pair prunes the exact dot product computation for a large number of candidate pairs. In our experiments we observed that $AP\_Time\_Efficient$ reduces the search space by at least an orders of magnitude (please, refer to Figure 2).

5. Experiments

We empirically evaluate the effectiveness of our fast matching technique. We use the following abbreviations in figures for each algorithm:

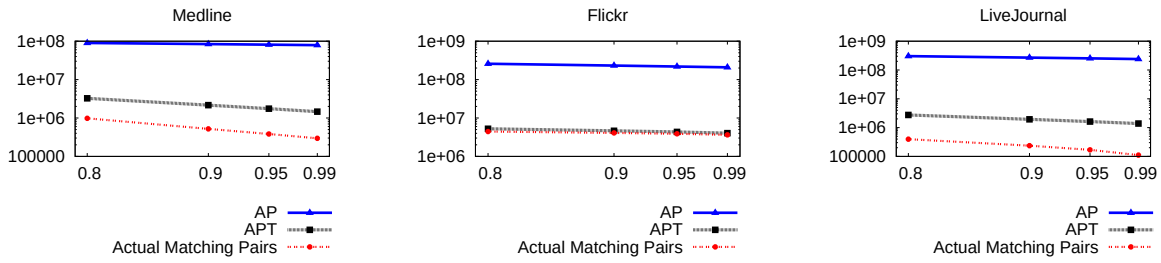

Figure 2. Candidate Pairs Evaluated vs Similarity Threshold for Cosine Similarity

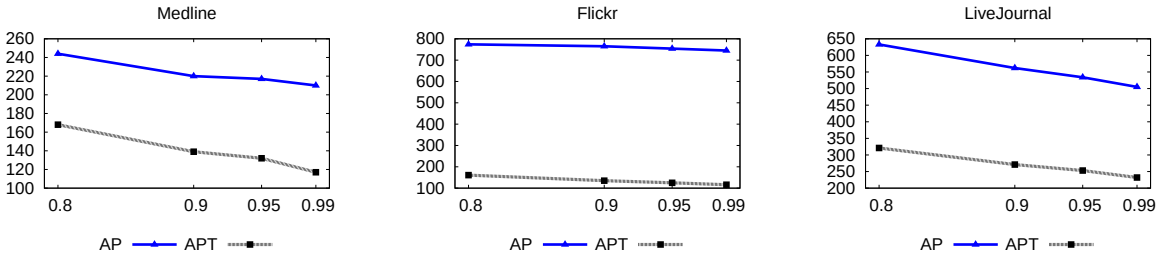

Figure 3. Runtime in Seconds vs. Similarity Threshold for Cosine Similarity

- **AP** : All_Pairs algorithm by Bayardo *et al.* [2]; and
- **APT** : $AP\_Time\_Efficient$ algorithm.

Many practical applications need pairs with relatively high similarity values [2], [1]. Hence, we varied the similarity threshold from 0.8 to 0.99 in 0.05 increments. The code, the datasets and additional experimental results are available for download on the Web [8].

5.1. Datasets

Medline dataset was selected to investigate possible applications for large web-based scientific digital libraries like PubMed and CiteSeer. We used the dataset prepared by the Auton Lab of Carnegie Mellon University. We are interested in finding pairs of authors that have similar collaboration patterns. Each vector represents the collaboration pattern of an author over the space of all authors. We use the weighing scheme of Newman [9] to derive the weight of collaboration between any two authors. If k authors have co-authored a paper, then it adds $1/(k-1)$ to the collaboration weight of each possible pair of authors of that paper. All vectors are then normalized to unit-length.

Flickr and LiveJournal datasets were selected to explore potential applications for large online social networks. We are interested in finding user pairs with similar social networking patterns. We use the dataset prepared by Mislove *et al.* [10]. Every user in the social network is represented by a vector over the space of all users. A user's vector has non-zero projection along those dimensions that correspond to the users in his/her friend list. But the weights of these social network links are unknown. So, we applied the weight distribution from the Medline dataset to assign the weights to these social network links in the two datasets.

5.2. Fast Matching Performance

We evaluate the performance of fast matching in the $AP\_Time\_Efficient$ algorithm based on two parameters: the size of the search space and the end-to-end run-time. The $AP\_Time\_Efficient$ reduces the search space by at least an order of magnitude (please, refer to Figure 2). The end-to-end speed up is between $1.4X$ and $6.5X$ (please, refer to Figures 3). Even though the search space is reduced by an order of magnitude, we get comparatively less end-to-end speed up, because some time is still spent traversing the inverted index to find candidate pairs.

The best speed up is obtained for the Flickr dataset, because it has a heavy tail in the distribution of vector sizes. Vectors having the long size typically generate a large number of candidate pairs. But $AP\_Time\_Efficient$ effectively avoids their generation and evaluation. In fact, for the Flickr dataset, the number of candidate pairs evaluated by $AP\_Time\_Efficient$ almost equals the actual number of matching pairs (please refer to Figure 2).

6. Conclusion and Future Work

We described the inverted index based framework common across recent exact algorithms for $APSS$. Within this framework, we presented tighter bounds on the candidate

Algorithm 2: $AP\_Time\_Efficient$ Algorithm

Input: V, t, d, $global\_max\_weight$, Ω, Π
Output: $MATCHING\_PAIRS\_SET$

$MATCHING\_PAIRS\_SET = \emptyset$;
$I_i = \emptyset \ , \forall \ 1 \leq i \leq d$;
Ω sorts vectors in decreasing order by max_weight;
Π sorts dimensions in decreasing order by density;
foreach $x \in V$ *in the order defined by* Ω **do**
    $partScoreMap = \emptyset$;
    /* Empty map from vector id to partial similarity score */
    $remMaxScore = \sum_{i=1}^{d} x[i] * global\_max\_weight[i]$;
    $minSize = (t/x.max\_weight)^2$;
    foreach i: $x[i] > 0$, *in the reverse order defined by* Π **do**
        Iteratively remove $(y, y[i])$ from front of I_i while $|y| < minSize$;
        foreach $(y, y[i]) \in I_i$ **do**
            if $partScoreMap\{y\} > 0$ *or* $remMaxScore \geq t$ **then**
                $partScoreMap\{y\} = partScoreMap\{y\} + x[i] * y[i]$;
        $remMaxScore = remMaxScore - global\_maximum\_weight[i] * x[i]$;
        /* Remaining maximum score that can be added after processing current dimension */
    foreach y: $partScoreMap\{y\} > 0$ **do**
        if $partScoreMap\{y\} + min(sum(y') * x.max\_weight, sum(x) * y'.max\_weight) \geq t$ **then**
            /* Tighter upper bound on the similarity score */
            $s = partScoreMap\{y\} + dot(x, y')$;
            if $s \geq t$ **then**
                $MATCHING\_PAIRS\_SET = MATCHING\_PAIRS\_SET \cup (x, y, s)$
    $maxProduct = 0$;
    foreach i: $x[i] > 0$, *in the order defined by* Π **do**
        $maxProduct = maxProduct + x[i] * min(global\_max\_weight[i], x.max\_weight)$;
        if $maxProduct \geq t$ **then**
            $I_i = I_i \cup \{(x, x[i])\}$;
            $x[i] = 0$;
return $MATCHING\_PAIRS\_SET$

Table 2. Datasets Used

Dataset	$n = d$	Total Non-zero Components	Average Size
Medline	1565145	18722422	11.96
Flickr	1441433	22613976	15.68
LiveJournal	4598703	77402652	16.83

size and similarity score. These bounds reduced the search space by at least an order of magnitude and provided significant speed up for three real-world datasets.

Incremental APSS where APSS is performed multiple times over the same dataset while varying the similarity threshold, is an interesting problem for future work.

Acknowledgments

We thank Paul Breimyer of NCSU for thoughtful comments and suggestions. This work is performed as part of the Scientific Data Management Center (http://sdmcenter.lbl.gov) under the Department of Energy's Scientific Discovery through Advanced Computing program (http://www.scidac.org). Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DEAC05-00OR22725.

References

[1] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *WWW '08*.

[2] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *WWW '07*.

[3] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC '02*.

[4] R. Fagin, R. Kumar, and D. Sivakumar, "Efficient similarity search and classification via rank aggregation," in *ACM SIGMOD '03*.

[5] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," in *ACM SIGMOD '04*.

[6] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *VLDB '06*.

[7] H. Turtle and J. Flood, "Query evaluation: strategies and optimizations," *Inf. Process. Manage.*, vol. 31, no. 6, pp. 831–850, 1995.

[8] "Code and data sets for our algorithms : www4.ncsu.edu/~acawekar/wi/."

[9] M. E. J. Newman, "Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality," *Physical Review*, vol. 64, no. 016132, 2001.

[10] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *ACM IMC '07*.