Adaptive Sorted Neighborhood Methods for Efficient Record Linkage

Su Yan*, Dongwon Lee*, Min-Yen Kan†, C. Lee Giles*
* College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802
†School of Computing, National University of Singapore, Singapore
{syan, giles}@ist.psu.edu, dongwon@psu.edu, kanmy@comp.nus.edu.sg

ABSTRACT

Traditionally, record linkage algorithms have played an important role in maintaining digital libraries – i.e., identifying matching citations or authors for consolidation in updating or integrating digital libraries. As such, a variety of record linkage algorithms have been developed and deployed successfully. Often, however, existing solutions have a set of parameters whose values are set by human experts off-line and are fixed during the execution. Since finding the ideal values of such parameters is not straightforward, or no such single ideal value even exists, the applicability of existing solutions to new scenarios or domains is greatly hampered. To remedy this problem, we argue that one can achieve significant improvement by adaptively and dynamically changing such parameters of record linkage algorithms. To validate our hypothesis, we take a classical record linkage algorithm, the sorted neighborhood method (SNM), and demonstrate how we can achieve improved accuracy and performance by adaptively changing its fixed sliding window size. Our claim is analytically and empirically validated using both real and synthetic data sets of digital libraries and other domains.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Record Linkage, Citation Matching, Entity Resolution, Sorted Neighborhood

1. INTRODUCTION

A critical task in Digital Libraries (DL) is to identify similar or matching digital library entities such as citations or authors, and consolidate them into a single canonical entity. For instance, CiteSeer [16] routinely integrates daily-crawled citations into its existing citation index. If the same citation already exists in the database, then newly-crawled citations should not be treated separately. However, due to diverse formats of citations found on web pages, this task is not always straightforward. Similarly, it is common to find the occurrence of an author in different spellings – i.e., “Jeffrey D. Ullman” vs. “Ullman, J.”. This problem, known as the *name authority control*, has been traditionally handled by librarians such that a list of canonical and variant entities are manually recorded in a catalog for resolution. However, as the size of document entities in modern DLs increases dramatically and their update becomes frequent, name authority control needs to be automated. A core technique to automate this task is the *record linkage* [21] or *entity resolution* [24] algorithm – i.e., identifying matching records or entities in a large collection fast and accurately, and the technique has been extensively studied in the context of DL [22] as well as other areas [2, 12, 24].

Although it works well in many specific scenarios, existing record linkage solutions developed for DLs are *not* very flexible. They employ many parameters (e.g., attributes to use for blocking, choice of blocking algorithms, window size, choice of similarity functions), the values of which are often set by human experts. Finding the ideal values of such parameters is not straightforward. It may also be the case that no single optimal value of a parameter exists for a given problem. Rather, it is more plausible to believe that values for such parameters must be dynamically adjusted to achieve the best performance. An example of such a problem is *blocking* [14], a popular filtering technique in record linkage that groups input entities into a set of clusters such that similar entities are put into the same cluster. More expensive record linkage solutions are applied to entities only within the same cluster. Thus, blocking saves computational cost significantly.

As reported in [22], a variety of blocking methods exist with different pros and cons. Since the same blocking method can yield bipolar results against different datasets, selecting a suitable blocking method for the given record linkage algorithm and dataset requires significant domain knowledge. A natural question to ask is, then, whether an algorithm can dynamically choose different blocking meth-

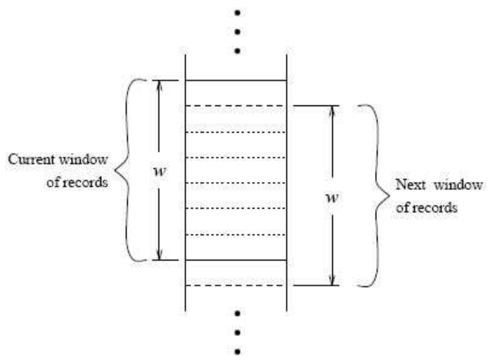

Figure 1: The illustration of the SNM method.

ods and parameters for different data sets and configurations with little human intervention. Recent work such as [4, 19], for instance, study how to dynamically select and combine a set of attributes for the blocking key and report good results. As digital library entities that the record linkage problem must deal with become more heterogeneous and large-scaled, we argue that statically-set and never-changing parameters of record linkage solutions do not handle diverse scenarios. In order to make existing solutions more flexible and applicable, we envision that "*adaptivity*" will play a critical role. Therefore, in this paper, we argue that one can achieve significant improvement in record linkage by adaptively and dynamically changing such parameters of record linkage algorithms during the execution time.

1.1 Motivation

To demonstrate our key ideas, let us use the well-known record linkage algorithm, the *sorted neighborhood method* (SNM) [12]. The SNM scheme first sorts all the entities using the pre-selected key attributes and heuristics. Then, it slides a fix-sized window from the beginning to the end of the list. In each sliding window, the first entity is compared to the rest of the entities within the same window to identify pairs with small distances. Then the window slides down by one entity and the process repeats until it reaches the end of the list. Thus, the sliding window works as a *blocking* scheme with the premise that matching entities are lexicographically similar so that they tend to be located within the same window. Figure 1 illustrates SNM. Single pass of SNM over n records with w records per window yields $n - w + 1$ blocks. Since each block incurs $w - 1$ number of record comparisons, overall, the running time of SNM becomes $O(nw)$. The baseline approach without any blocking requires all pair-wise comparisons among records, yielding $O(n^2)$. Due to the relatively faster running time compared to the baseline approach and the simple implementation, the SNM approach has been a popular choice of record linkage algorithm in many data applications.

Note that this simple algorithm uses several user-determined parameters:

- Choice of attributes for a key
- Size of the sliding window
- Distance threshold

In the original work of SNM [12] and all its variations, these parameters are pre-set once and never change during the execution of the algorithm (e.g., use the first 4 characters of the attribute "lastname" as a key and set the default window size as 3). However, such fixed parameters cannot cope with the fluctuating data characteristics. For instance, Figure 2 shows the varying sizes of the ideal blocks in the Cora data set, where 1,295 citations of 116 Computer Science articles are mixed [1]. If the ideal blocking is applied, then one will have 116 blocks. However, each block has different number of records as shown in Figure 2, from as few as one to as many as 55 records. Therefore, choosing the right window size (, which is also the block size in SNM) for this data set to ensure high precision and recall requires dynamic solution.

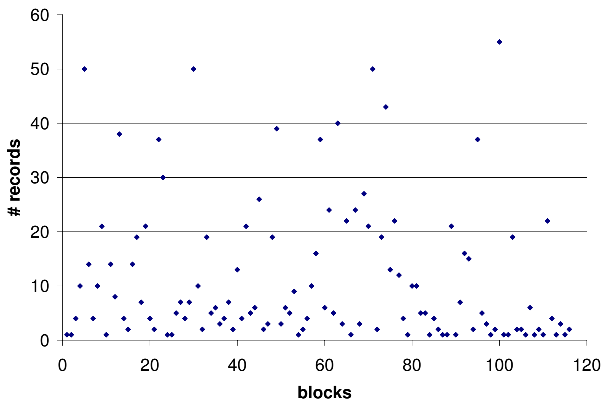

Figure 2: Ideal block sizes in the Cora dataset (116 blocks, alphabetically ordered).

1.2 Sketch of Our Solution

Now, let us see how we can adaptively change the window size. Intuitively, the idea is similar to how people watch videos – i.e., if two subsequent scenes are similar, people often press fast-forward to skip frames to arrive at new scenes quickly, and press fast-backward to go back if too many frames are skipped. Similarly, by monitoring how close or far two neighboring entities are, one can extend or shrink (i.e., fast-forward and fast-backward) the sliding window adaptively. To exactly increase or decrease the window size, one can use various heuristics (e.g., following linear, exponential, or prime number). In addition, more complex schemes, which are similar to cardinality estimation techniques in databases, can be used to estimate the density of duplicates.

Overall, we study this adaptivity issue of record linkage algorithms in depth. Among many parameters of record linkage algorithms, in this paper, we focus on the size of the sliding window in SNM and propose the adaptive version of SNM, named as the *adaptive sorted neighborhood method (A-SNM)*. Note that the size of the window in SNM amounts to the size of the block, which in turn is related to the *aggressiveness* of a blocking method. That is, by exploiting the characteristics of datasets, we propose to adaptively control the aggressiveness of a blocking method. Our contributions are:

- We advocate the importance of adaptivity in record

[1]We will introduce this dataset in detail in section 4.

linkage problems, and study the problem in detail using the classical SNM record linkage algorithm.

- We present two adaptive versions of the SNM algorithm, named as the *incrementally-adaptive SNM* (IA-SNM) and
  the *accumulatively-adaptive SNM* (AA-SNM). Although quite simple, these adaptive versions show significant improvements over the original SNM in our experimental study.
- We study various effects of adaptivity in A-SNM in several dimensions: (1) distributions – Uniform vs. Poisson vs. Zipf, (2) data sets – real bibliographic citations vs. real restaurant addresses vs. synthetic mailing list, (3) adaptiveness – linear vs. geometric vs. full.
- Since what we advocate in this paper is the adaptive framework, the same idea can be applied to other well-known record linkage algorithms. We leave this application as a future work.

2. RELATED WORK

The general record linkage problem has been known as various names in different disciplines – record linkage (e.g., [9, 5]), citation matching (e.g., [17]), identity uncertainty (e.g., [23]), merge-purge (e.g., [12]), object matching (e.g., [6]), entity resolution (e.g., [2, 24]), authority control (e.g., [13, 25]), and approximate string join (e.g., [10]) etc. In this paper, we focus on the record linkage techniques in the context of digital libraries (DLs).

Bilenko et al. [5] have studied name matching for information integration using string-based and token-based methods. Cohen et al. [7] have also compared the efficacy of string-distance metrics, like JaroWinkler, for the name matching task. [15] experimented with various distance-based algorithms for citation matching, concluding that word based matching performs well. [22] conducted an in-depth study on the split entities case from the blocking point of view. The ALIAS system in [24] proposes a framework to detect duplicate entities such as citations or addresses, but its focus is on the learning aspect.

[4] treats the adaptive record linkage problem from the aspect of learning an optimal blocking strategy using training data. A blocking strategy states the attributes to be used for blocking, as well as the method that combines the attributes (e.g. conjunction, disjunction). The work reports that the adaptive framework for training blocking strategies is efficient and accurate for a given domain. [19] solves the same problem with a different machine learning method, and also reports good results. Our work in this paper is greatly inspired by these two works.

3. ADAPTIVE SORTED NEIGHBORHOOD

When applied to n records, the original SNM approach slides a fix-sized window $n - w + 1$ times, and generates $n-w+1$ number of blocks. Note that these generated blocks are *overlapping* each other, and *fix-sized.* However, in the adaptive SNM, the "window" becomes only an interim tool to generate final blocks, and all generated blocks will have different (thus adaptive) sizes.

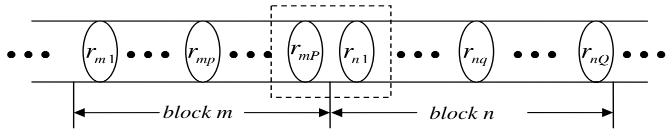

Figure 3: An example of boundary pairs.

Definition 1 (Block) *A* block *is a collection of records that are potentially duplicates each other.* □

The basic premise of the SNM approach is *sorting* – i.e., when records are lexicographically sorted using some attributes, duplicate records tend to locate in the neighborhood. The underlying hypothesis is the following:

Hypothesis 1 *The n sorted records, $r_i \leq \cdots \leq r_{i+n}$, may satisfy:*

$$dist(r_i, r_{i+1}) \leq dist(r_i, r_{i+2}) \leq \cdots \leq dist(r_i, r_{i+n})$$

where $dist(r_i, r_j)$ is the distance between records r_i and r_j.□

Needless to say, Hypothesis 1 does not always hold because sorting is done lexicographically, but not by distance. The rate that Hypothesis 1 does not hold depends on the error rate of the blocking field and varies for different datasets. To overcome this limitation, one can run SNM multiple times using different attributes to sort each time, or apply transitive closure rules (i.e., if $r_i \approx r_j$ and $r_j \approx r_k$, then $r_i \approx r_k$) to both single pass and multiple pass results.

The reason that SNM uses blocks on top of sorting is based on the following Hypothesis:

Hypothesis 2 *Suppose two records, r_i and r_j, are the first and last records within the same block while the third record r_k is outside the block. Then, the following may hold:*

$$dist(r_i, r_j) \leq \phi < dist(r_i, r_k)$$

where ϕ is the minimum distance threshold. □

That is, if blocking is done ideally, then the records outside a block must be *significantly* different from the records within the block. To find blocks, one only needs to find the boundary records between adjacent blocks. For example, in Figure 3, record r_{mP} is the last record of block m, and record r_{n1} is the first record of block n. According to Hypothesis 2, $dist(r_{mP}, r_{n1}) > \phi$. This suggests that we only need to find those records whose distance to its next adjacent record is larger than a threshold ϕ. Such a record and its next adjacent record (e.g. $\{r_{mP}, r_{n1}\}$) define the boundary between two adjacent blocks, and thus the pair of records is called a *boundary pair.*

To find boundary pairs, a naive way is to compare every adjacent record pairs. In the blocking phase, we need not to compare records in detail by incorporating all attributes of the records, but only approximately compare them on blocking fields. If the distance between any two adjacent records is larger than the threshold, we find a boundary pair. After all boundary pairs found, the whole sorted list can be divided into non-overlapping blocks of various sizes. Note that, such division is adaptive to the duplicate distribution of the specific database. If the size of a database is n, then there are $n - 1$ approximate comparisons during the blocking phase. Therefore, this method is inefficient for large databases.

To reduce the number of comparisons in the blocking phase, We propose a set of adaptive window setting schemes to find the boundary pairs with much less approximate comparisons. Every scheme uses a sequence of windows to aggressively approach the position of boundary pairs, but each with different window setting methods in two phases: (1) "enlargement" phase to gather as many potential duplicates together with as few cheap distance calculations as possible, and (2) a "retrenchment" phase to find boundary pairs within the final window from the first phase.

Algorithm 1 $block$ = IA-SNM($records, key, \phi$)

```
1: sort $records$ on $key$
2: /* Initialization */
3: $n \leftarrow$ number of records
4: $w \leftarrow 2$
5: $block \leftarrow [\,]$
6: posit window to the initial position $w.first \leftarrow record(1)$
7: /* Searching for blocks */
8: while $index(w.last) < n$ do
9:     $block \leftarrow [index(w.first)]$ /* save the starting index of the current block */
10:    /* enlargement */
11:    while $dist(w.first, w.last) < \phi$ do
12:        $w = \frac{\phi \cdot w}{dist(w.first, w.last)}$
13:    end while
14:    /* retrenchment */
15:    if $dist(w.first, w.last) > \phi$ then
16:        $w = \frac{\phi \cdot w}{dist(w.first, w.last)}$
17:        while $w > 2$ do
18:            binary comparison
19:        end while
20:    end if
21:    $block \leftarrow [index(w.last)]$/* save the ending index of the current block */
22:    $w \leftarrow 2$ /* reset window to minimum size */
23:    reposition the window, s.t. $index(w.first) = index(w.last) + 1$
24: end while
25: $block \leftarrow [n]$
26: return $block$
```

3.1 Incrementally-Adaptive SNM (IA-SNM)

The basic idea is to measure if records within a small neighborhood are close/sparse and if there are rooms to grow/shrink in the window, then the window size is increased/decreased dynamically. In order to measure the record distribution within a window, it seems that we need to measure the distances between all the records in the window. However, If we consider a small neighborhood defined by a window in the sorted list, then we can assume that Hypothesis 1 holds in the window. Therefore, we only need to measure the distance between the first and last record in the window, and this distance can represent the overall record distribution within the window.

In the ideal case, the distance between the first and last record in a block equals to the distance threshold ϕ, that is, there is no room to enlarge the block to incorporate more potential duplicate records or to retrench the block to remove incorrectly identified potential duplicate records.

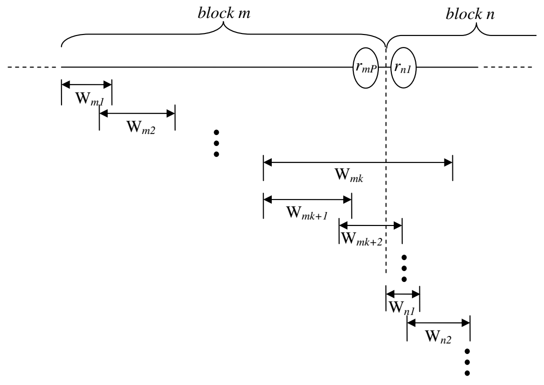

Figure 4: A running example of AA-SNM.

Suppose W_1 is the current window under consideration. The distance between the first and last records satisfies that, $dist(r_1, r_{W_1}) < \phi$, where ϕ is the distance threshold. This distance indicates that records within the current window are close to each other, so there is still room to enlarge the window size to find more potential duplicate records. Otherwise, the window should be retrenched. Now we need to measure the extent of the window adjustment. $\frac{dist(r_1, r_{W_1})}{W_1}$ can be understood as the "average distance" between records within window W_1. We choose the next step window size by $\frac{dist(r_1, r_{W_1})}{W_1} W_2 = \phi$. In this way, the window size of the next step is set to be the ideal window size estimated from the current window. By using the above formula as an invariant, one can adaptively change the window size W to maximize recall (at the risk of increased running time). The window size adjustment ends when $dist(r_1, r_W) \approx \phi$. Note that in each step, we only adjust the window size, and the starting point of a window is not changed. For example, the first record of W_2 is the same as that of W_1. Therefore, we use incrementally adjusted windows to approach the optimal block sizes, and we call this method incrementally adaptive.

When applying the above adaptive window setting idea to a real blocking problem, we start from the minimum window size of 2 and continue the window size adjustment process (enlargement and retrenchment). Depending on the key selection and the distance measure used, a vibration in the window size adjustment may happen after the first retrenchment. To prevent the potential vibration in the window size adjustment that may happen in this method, after the first retrenchment, we do the binary comparison within the final window from the enlargement phase to find the boundary pair. In other words, in the binary comparison stage, we reduce the window size by half each time when adjusting window sizes and restrict the search scope to the last window from the enlargement phase. We found a boundary pair when we reached a window of size 2 and the distance between the two records is larger than the threshold ϕ.

The pseudo code of the IA-SNM method is shown in Algorithm 1.

3.2 Accumulatively-Adaptive SNM (AA-SNM)

In this section, we show another scheme that can adaptively find various sized blocks. Unlike IA-SNM, which does not explicitly search for boundary pairs to find blocks, the goal of this method is to quickly and accurately find all boundary pairs. We introduce this method with a running example as shown in Figure 4. In this example, we scan the sorted list of records to search for the boundary pair between block m and block n, $\{r_{mP}, r_{n1}\}$.

- **Window Enlargement**: Suppose we start from a small window $W_{m1} = 2$ that locates at the beginning of block m (this position is known because we should have found the boundary position between block m and its preceding block in this stage). As in IA-SNM, we measure the distance between the first and the last record in the current window W_1. If the distance is less than or equal to a threshold, $dist(r_1, r_{W_{m1}}) < \phi$, according to Hypothesis 1, all the records in window W_{m1} can be determined as potential duplicates to each other and should be grouped into the same block. It also suggests that the boundary pair is not in the current window and there are more potential duplicates outside window W_{m1}. Therefore, We need to enlarge the neighborhood of comparison to include more potential duplicates and to search for the boundary pair. This can be achieved by either enlarging the window size, as done in IA-SNM, or by moving the window forward and connecting the results of consecutive windows later. We choose the second option for this algorithm (to be explained later). Thus, we slide the window forward and get a new window W_{m2}. If all the records in W_{m2} are also potential duplicates, we continue to slide the window forward to get window W_{m3} and so on, until we reach the window W_{mk} that contains the boundary pair. In order to approach the boundary pair aggressively, we enlarge window sizes as we move the window forward, i.e. $|W_{m1}| < |W_{m2}| < \cdots < |W_{mk}|$.

  We slide a window forward such that the first record of the new window is the last record of the preceding window. Thus, the two consecutive windows share one overlapping record. See Figure 5 as an example. Furthermore, since we have found that r_{m1} and r_{mi} are potential duplicates, and r_{mi} and r_{mk} are potential duplicates, by transitivity, we assume that r_{m1} and r_{mk} are also potential duplicates. Therefore, all the records within W_{m1} and W_{m2} are potential duplicates to each other and should be grouped into one block. Notice that window W_x is the combination of W_{m1} and W_{m2}. The reason that we do not use the larger window W_x but compare r_{m1} and r_{mk} directly is that, the distance order as stated in Hypothesis 1 may only holds in a small neighborhood. In the larger neighborhood W_x, it may happen that r_{m1} and r_{mk} are in fact potential duplicates, but $dist(r_{m1}, r_{mk}) > \phi$, due to the errors in the blocking fields. By using a sequence of overlapping small windows instead of one growing large window, we expect to diminish the influence of small blocking-field-errors to the blocking result. This is why we call this method accumulatively adaptive.

- **Window retrenchment**: When reaching window W_{mk}, we first measure the distance between the first and last record of W_{mk} and suppose we find that the distance is greater than the threshold ϕ. This implies that at least one boundary pair is in W_{mk}. We begin to decrease window size at each step to search W_{mk} in detail and to find the position of the boundary pair $\{r_{mP}, r_{n1}\}$. We start the detailed search from the former part of W_{mk}

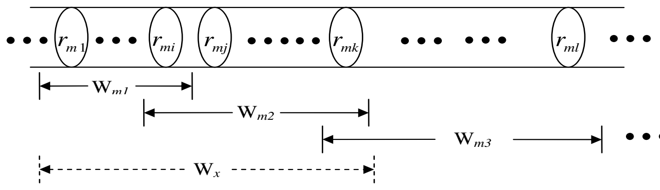

Figure 5: An example of setting window positions in AA-SNM.

  (to find the first boundary pair) by retrenching W_{mk} to W_{mk+1} and let both have the same starting record. Suppose we find that all records in W_{mk+1} are potential duplicates. Then, we move the window forward by the same scheme used in the enlargement phase, with smaller window size ($|W_{mk+2}| < |W_{mk+1}|$). This is because we want to search in more detail. The process of detailed search continues until we reach the window that contains only two records and the distance between the two records is larger than the threshold ϕ. In our example, this window contains $\{r_{mP}, r_{n1}\}$. Then the record index mP indicates the ending position of *Block m* and the record index $n1$ indicates the starting position of *Block n*.

  To search the boundary pair between *Block n* and its next adjacent block, we start from a small window W_{n1} of size 2 positioned at the beginning of *Block n* (r_{n1}), and the above process of window enlargement and retrenchment is repeated.

Varying Window Sizes: In both enlargement and retrenchment phases, window sizes are increased or decreased. For this purpose, various policies can be used as follows.

1. **Linear Adjustment:** The size of a new window is adjusted linearly by adding or subtracting a constant value α to or from the preceding window size.
2. **Geometric Adjustment:** The size of a new window is adjusted geometrically by multiplying or dividing a constant value α to or from the preceding window size.
3. **Full Adjustment:** The extent of the window adjustment is not fixed. Rather, the extent of the adjustment for the current step is decided using several former windows.

Since linear and geometric adjustment policies are straightforward, let us elaborate on full adjustment policy here. The idea behind the adjustment policy is that, in the window enlargement phase, we use the number of records already found for the current block to estimate the number of records to be found in the next step. For example in Figure 4, we estimate the size of window W_{m2} by $|W_{m2}| = |W_{m1}|$, and estimate the size of W_{m3} by $|W_{m3}| = |W_{m1}| + |W_{m2}|$ (if omit the one overlapping record). Therefore, the size of the last window in the enlargement phase, W_{mk} is estimated as $|W_{mk}| = \sum_{i=1}^{k-1} |W_{mi}|$. In the window retrenchment phase, we do binary comparison. Each time we decide to retrench a window, we decrease its size by half. In terms of our running example in Figure 4, $|W_{mk}| = 2|W_{mk+1}| = 4|W_{mk+2}| = \cdots$.

The pseudo code of AA-SNM using the full adjustment policy is shown in Algorithm 2.

Algorithm 2 *block* = *Full* AA-SNM(*records*, *key*, ϕ)

```
1: sort records on key
2: /* Initialization */
3: $n \leftarrow$ number of records
4: $w \leftarrow 2$
5: $block \leftarrow [\,]$
6: posit window to the initial position $w.first \leftarrow record(1)$
7: /* Searching for blocks */
8: while $index(w.last) < n$ do
9:     $block \leftarrow [index(w.first)]$ /* save the starting index of the current block */
10:    /* enlargement */
11:    while $dist(w.first, w.last) \leq \phi$ do
12:        move window forward s.t. $w.first = w.last$
13:    end while
14:    /* retrenchment */
15:    while $w > 2$ do
16:        binary comparison
17:    end while
18:    $block \leftarrow [index(w.last)]$ /* save the ending index of the current block */
19:    $w \leftarrow 2$
20:    reposition the window, s.t. $index(w.first) = index(w.last) + 1$
21: end while
22: $block \leftarrow [n]$
23: return block
```

4. EXPERIMENTAL EVALUATION

We use two real data sets and several synthetic data sets, all of which have been used in the existing work on blocking, to test the performance of adaptive sorted neighborhood methods. We also do controlled experiments to evaluate the adaptive methods from different aspects.

4.1 Evaluation Metrics

We use *pairs completeness* (PC), *reduction ratio* (RR), and *F-score* as defined blew to evaluate the performance of blocking schemes. These three metrics have been widely used in the existing blocking related work.

$$PC = \frac{\#Correctly\ Indentified\ Duplicate\ Pairs}{\#True\ Duplicate\ Pairs} \quad (1)$$

$$RR = 1 - \frac{\#Identified\ Potential\ Duplicate\ Pairs}{\#All\ Pairs} \quad (2)$$

$$F\text{-}score = \frac{2 * RR * PC}{RR + PC} \quad (3)$$

PC evaluates a blocking scheme based on the number of true duplicate pairs in the candidate set versus those in the entire set. Thus it measures the coverage of true positives. RR measures how well the blocking scheme reduces the number of record pairs to be compared in detail in the candidate set. Both PC and RR should be maximized, but there is a tradeoff between them. *F-score* captures this tradeoff by combining PC and RR via a harmonic mean.

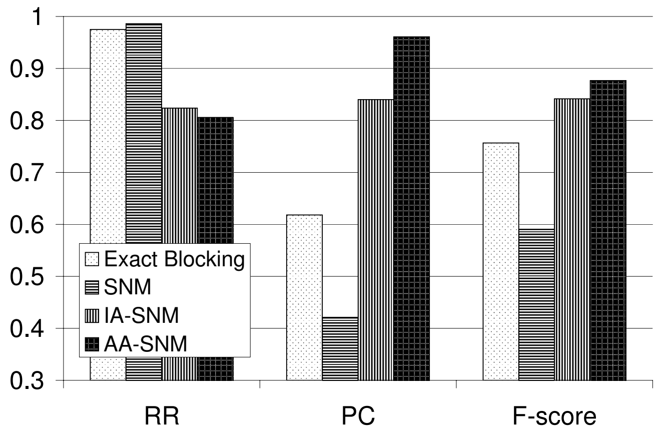

Figure 6: Blocking schemes comparison with the Cora dataset (data set: citation, size: 1,295).

4.2 Data sets

We use both real and synthetic datasets to test our adaptive blocking schemes. The first real data set we use is the hand-labeled Cora dataset, which contains 1295 12-field citations of 116 computer science papers (we relabeled the original dataset to correct some labeling error and results in 116 papers). The author fields are standardized by automatically abbreviating each name into the first initial concatenated with the last name, and the blocking key is chosen as the concatenation of all the standardized author names of a citation. Note that, the blocking performance is influenced by the choice of the blocking key. Usually the blocking key is chosen by domain experts and requires domain knowledge. However, it does not matter how we choose the blocking key for the purpose of our study, as long as all the blocking schemes are tested and compared by using the same blocking key. The blocking fields of the Cora dataset are not error-free due to citation segmentation errors and spelling mistakes.

The second real datbase we use is the 4-field Restaurant data set. It contains 864 restaurant names and addresses with 112 duplicates, composed of 533 and 331 restaurants assembled from Fodor's and Zagat's restaurant guides. The individual databases are duplicate free. The blocking key we use is the first 4 characters of the restaurant name concatenated with the first 4 characters of the city name.

The synthetic databases for our experiments are generated by DBGen [12]. DBGen is a database generator that artificially generates 9-field mailing list data. A large number of parameters of the generated database can be adjusted, including the size of the database, the percentage of duplicate records in the database, and the amount of error introduced in the duplicated records in each attribute field. For the databases generated by DBGen, the blocking key was specified to be the last name attribute truncated to 4 characters concatenated with the first name attribute truncated to 4 characters.

A summarization of the databases used in our experiments are outlined in Table 1.

4.3 Experiments with real data

We first test blocking schemes with Cora, the real citation database. For the SNM blocking, we set the window size to

Table 1: Summary of databases for experiments

Database name	size	property	#field	content	#blocks	maximum #records per block
Cora	1295	real	12	citation	116	55
Restaurant	864	real	4	restaurant addresses	112	2
DBGen	varies	synthetic	9	mailing list	varies	varies

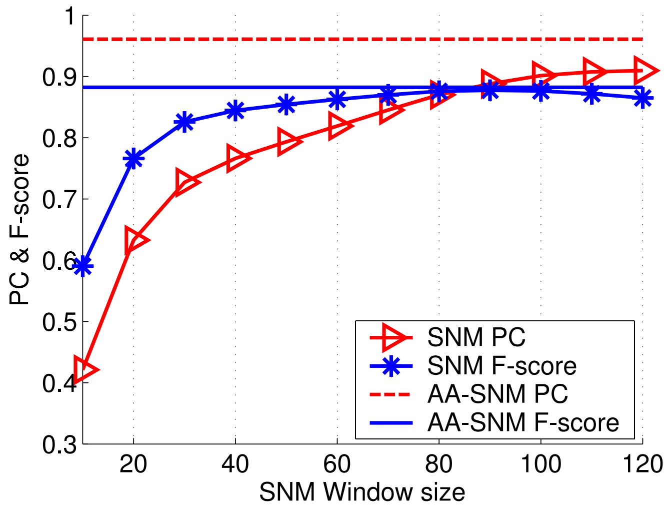

Figure 7: Detailed comparison between AA-SNM and SNM with Cora dataset (data set: citation, size: 1,295).

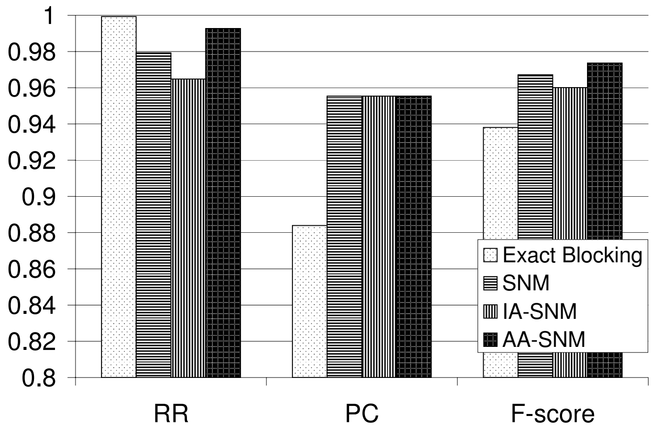

Figure 8: Blocking schemes comparison with the Restaurant dataset (data set: restaurant addresses, size: 864).

10, the value chosen in most experiments performed by the original work of SNM [12]. For adaptive sorted neighborhood schemes, any choice of the distance metric for the approximate record distance comparison is fine, and we choose to use *edit distance*. We follow the work in [18] and use one fourth of the original dataset to compose a validation set and tune the parameter with it. We also implemented the *Exact Blocking* method as a baseline. For this method, records with exactly the same key value are assigned to the same block. It is also known as "blocking" [8], and "standard blocking" [1, 3].

Experiment results with the Cora dataset are shown in Figure 6. Adaptive sorted neighborhood methods significantly outperform the original SNM method. AA-SNM has better performance than IA-SNM. The F-score of AA-SNM is 49% larger than that of SNM. Note that, the F-score difference between IA-SNM and AA-SNM is about 4%, but the PC difference is around 13%. This shows that AA-SNM is a better blocking method than IA-SNM since it finds more potential duplicate pairs with similar F-score.

The poor performance of SNM in the experiment is because the number of records that can be determined as potential duplicates of each other is larger than the window size. Figure 2 shows that many records have more than 10 duplicates. This is the key problem of SNM that has been reported by many other work [3, 11, 20]. We run SNM over the Cora dadaset with increasing window sizes until the highest F-score is achieved, and we compare the results with AA-SNM, as shown in Figure 7. The window sizes for SNM range from 10 to 120, with the best F-score achieved when the window size is 90. After this point, the PC value keeps increasing, but the F-score begins to decrease. From the figure, AA-SNM produces the highest F-score that can be achieved by SNM. However, even the window size of SNM can be accurately set as 90, the PC value of AA-SNM is still 8% larger than that of SNM, which shows the advantage of AA-SNM.

We then use another quite different real dataset to test our adaptive blocking schemes. In the Restaurant dataset, a record has at most 1 duplicate. In other words, the maximum block size in the optimal case should be 2. We again set the window size of SNM to 10. Figure 8 shows the experimental results. All the methods make better performance with the Restaurant dataset than with the Cora dataset. This time, SNM makes very good performance, better than IA-SNM, but still worse than AA-SNM. The reason for the better performance of SNM is that, 10 is a large enough window size to incorporate almost all potential duplicates of a record. Experimental results with the two real datasets show that the choice of the window size for SNM depends on the specific duplicate distribution of a dataset. Adaptive methods make better blocking performance because of the adaptivity to the dataset.

4.4 Varying the error rate of duplicates

A record and its duplicates refer to the same real entity but are textually different. The extend of the difference between a record and a duplicate is called the *error rate of duplicates* in our study. A good blocking scheme should not be greatly influenced by changing the error rate of duplicates. In this section, we do controlled experiments to test how different blocking schemes behave with the varying error rate of duplicates.

The errors introduced in the duplicates by DBGen range

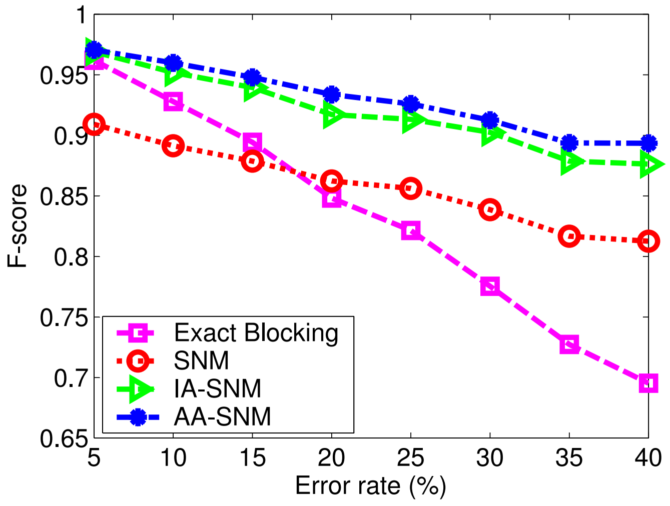

Figure 9: Influence of error rate to blocking schemes (data set: mailing list, size: 10,000).

from small typographical mistakes, to complete change of last names and addresses. The generator introduces typographical errors according to frequencies known from precious research on spelling correction algorithms. Since our blocking key contains only the last name and the first name attributes, we only consider the parameters of these two attributes and use DBGen default parameters for all the other attributes. 8 databases each of size around 10,000, and each containing uniformly sized blocks are generated, with the error rates of the "person names" field ranging from 5% $\sim$ 40%. We fixed the proportion of each individual error (insertions, deletions, replacements, and swappings) to the total error by using the default parameters of DBGen, but only change the total error rate of "person names". We set the average number of duplicates per record as 10, and the window size for SNM as 10.

Experimental results are shown in Figure 9. The performance of all the blocking schemes decrease as the error rate in the blocking field increases, with the performance degeneration of Exact Blocking the largest. AA-SNM again makes the best performance. The performance difference between IA-SNM and AA-SNM is small but gets larger as the error rate increases. It shows that AA-SNM has better resistance to the error in the blocking field than IA-SNM. The reason for the poor performance of Exact Blocking is that, Exact Blocking requires all the records within the same block to share exactly the same blocking key. However, when the error rate increases, the disambiguation power of a blocking key decreases. As a result, schemes based on exact match of blocking keys will produce poor performance.

4.5 Varying the size of blocks

A good adaptive blocking method should perform consistently when the number of duplicates per record varies. In this section, we do controlled experiments to test the consistency of adaptive blocking methods to the varying block sizes.

We vary the sizes of blocks of a dataset according to three distributions, which are the Uniform distribution, the Poisson distribution, and the Zipf distribution. With these three distributions, the variance of block sizes range from moderate to sever. In generating databases, we set the *mean* of the Uniform distribution and the Poisson distribution as 20, and the Zipf parameter θ ($0 \leq \theta \leq 1$) is set to 0.25. Error rate in the blocking field is fixed to 20%. 3 databases, one for each distribution are generated, each of size around 1,000. The variance of the block sizes of the generated databases are shown in Figure 11. For SNM blocking, we test two cases. We first set the window size to the conventional value 10 and refer to it as SNM-10. We also run SNM over the three datasets with increasing window sizes and pick the values that let SNM achieve the highest F-scores, and refer to this method *Optimal SNM*. According to experimental results, 35, 70, and 130 are the optimal window sizes for SNM when the duplicates of the databases follow the Uniform, Poisson, and Zipf distribution respectively.

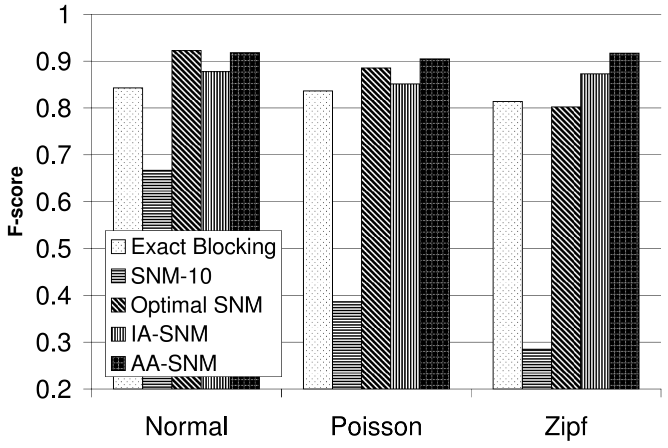

Figure 10: Influence of the variance in block sizes to blocking schemes(data set: mailing list, size: 1,000).

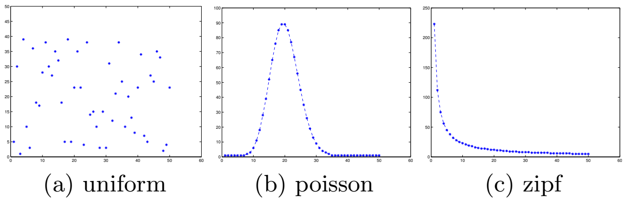

Figure 11: Variance in block sizes (data set: mailing list, size: 1,000).

Experiment results are shown in Figure 10. IA-SNM, AA-SNM, and Exact Blocking are robust to the variation in the distribution of duplicates. For all the three distributions, AA-SNM achieves the best performance. Although Optimal SNM has better performance than SNM, and achieves the best performance in the Uniform case, it is not adaptive to the variance of block sizes, and degenerates fast as the variation in blocks sizes becomes more sever.

4.6 Complexity of varying window sizes

In this section, we try to find out which method of varying window sizes is more efficient for the AA-SNM blocking scheme. We implement 7 methods to adaptively adjust window sizes for AA-SNM and compare the number of

Table 2: Complexity comparison of policies that set window sizes in AA-SNM

size	Linear-1	Linear-3	Linear-5	Linear-10	Linear-20	Geometric-2	Full
1k	537	422	**396**	421	540	550	432
2k	955	762	**737**	820	967	940	765
5k	2456	**1939**	2153	2162	2444	2394	2012
10k	5242	3867	4267	**3073**	4364	3852	3228

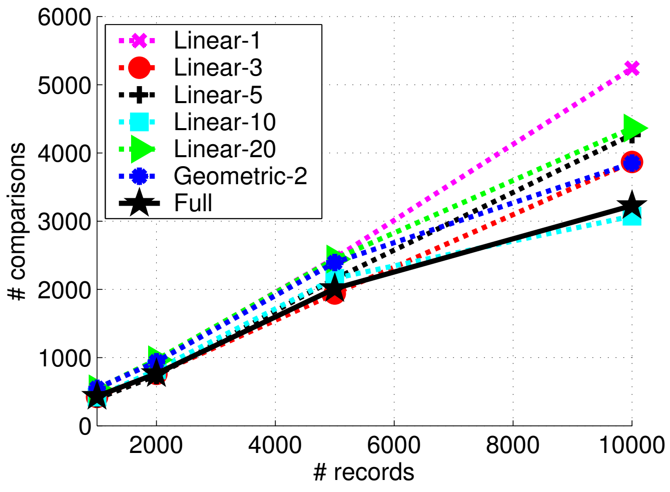

Figure 12: Complexity comparison of policies that set window sizes in AA-SNM (data set: mailing list).

times of approximate distance comparison required in the blocking phase. 5 schemes follow the linear adjustment policy, denoted as "Linear-α". We vary the aggressiveness of linear adjustment method by varying the constant value. We implement 1 geometric adjustment method, denoted as "Geometric-2", which means the window size is adjusted geometrically by multiplying or dividing by 2 to or from the preceding window. The last scheme is full adjustment, denoted as "full". 4 databases with sizes ranging from 1K to 10K are generated. Duplicates follow the Zipf distribution, and the error rate of the blocking field is fixed to 20%. Comparison results are shown in table 2. The boldface numbers in the table indicate the minimum times of comparison (best result) achieved for a given database. Figure 12 is drawn based on the table.

The experimental results confirm our consideration about the extent of adjusting the window size. When the extent is too small, e.g. 1 record each time (linear-1), the number of computations is high. On the contrary, when the extent is too large, e.g. 20 records each time (linear-20), the number of computations is still high. Besides, the optimal extent of window adjustment, which achieves the minimum number of comparisons, changes when the data size changes. An extent that achieves the minimum for one data size may produce very large number of computations for other data sizes (e.g., linear-3,linear-5, linear-10). Therefore, it is not clear how to choose the optimal extent such that it produces the minimum number of comparisons regardless of the data size. As shown in Figure 12, although full adjustment does not achieve the minimum for all the 5 data sizes, it produces the near optimal performance in all the cases.

5. CONCLUSION

In this paper, we advocate an *adaptive* framework to adaptively and dynamically adjust parameters of record linkage algorithms during the execution time. To demonstrate key ideas, we use a well-known record linkage algorithm, sorted neighborhood method (SNM), as a working example. We propose two adaptive versions of SNM, both of which dynamically adjust the sliding window size, a key parameter used in SNM, during the blocking phase to adaptively fit the duplicate distribution. Comprehensive experiments with both real and synthetic data sets of three domains validate the effectiveness and the efficiency of the proposed adaptive schemes. The adaptive schemes are robust to the variance in the size of each individual block, which can range from moderate to sever. Besides, the adaptive schemes show better resistance to the errors in the blocking fields. Several methods for adjusting window sizes, which are used in the adaptive methods, are proposed and compared. Among them, the full adjustment method is shown to be near optimal.

However, this paper is only the first step toward the fully adaptive framework where any parameter of record linkage algorithms can be dynamically adjusted to maximize some objective functions. Some key problems still need to be solved to achieve "full" adaptivity. Besides, many blocking methods other than SNM have been proposed, such as the bigram indexing, the canopy clustering, etc. To test how the adaptive idea can improve such methods is also interesting. As we stated before, many researchers work toward the same goal of adaptive record linkage, however, each on a specific aspect. To build a comprehensive framework that incorporate all the aspects is necessary. Finally, at this point, it is not entirely clear whether it is always good to have some "knobs" to tune parameters or everything is set dynamically. We argue that, when expert knowledge is available, tuning "knobs" may yield better performance. However, in most real world problems where expert knowledge are hard to obtain, it is helpful to have methods that can automatically choose reasonable parameters for us.

6. ACKNOWLEDGEMENT

The research of Dongwon Lee was partially supported by Microsoft SciData Award (2005) and IBM Eclipse Innovation Award (2006). Authors sincerely thank anonymous reviewers for their helpful comments.

7. REFERENCES

[1] A. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI '05: Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration*, pages 30–39, Washington, DC, USA, 2005. IEEE Computer Society.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. "Eliminating Fuzzy Duplicates in Data Warehouses". In *VLDB*, 2002.

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

[4] M. Bilenko, B. Kamath, and R. J. Mooney. "Adaptive Blocking: Learning to Scale Up Record Linkage". In *Workshop on Information Integration on the Web*, 2006.

[5] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. "Adaptive Name-Matching in Information Integration". *IEEE Intelligent System*, 18(5):16–23, 2003.

[6] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. "Robust and Efficient Fuzzy Match for Online Data Cleaning". In *ACM SIGMOD*, 2003.

[7] W. Cohen, P. Ravikumar, and S. Fienberg. "A Comparison of String Distance Metrics for Name-matching tasks". In *IIWeb Workshop held in conjunction with IJCAI*, 2003.

[8] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. Tailor: A record linkage tool box. *icde*, 00:0017, 2002.

[9] I. P. Fellegi and A. B. Sunter. "A Theory for Record Linkage". *J. of the American Statistical Society*, 64:1183–1210, 1969.

[10] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. "Text Joins in an RDBMS for Web Data Integration". In *Int'l World Wide Web Conf. (WWW)*, 2003.

[11] L. Gu and R. A. Baxter. Adaptive filtering for efficient record linkage. In *SDM*, 2004.

[12] M. A. Hernandez and S. J. Stolfo. "The Merge/Purge Problem for Large Databases". In *ACM SIGMOD*, 1995.

[13] Y. Hong, B.-W. On, and D. Lee. "System Support for Name Authority Control Problem in Digital Libraries: OpenDBLP Approach". In *European Conf. on Digital Libraries (ECDL)*, Bath, UK, Sep. 2004.

[14] R. P. Kelley. "Blocking Considerations for Record Linkage Under Conditions of Uncertainty". In *Proc. of Social Statistics Section*, pages 602–605, 1984.

[15] S. Lawrence, C. L. Giles, and K. Bollacker. "Digital Libraries and Autonomous Citation Indexing". *IEEE Computer*, 32(6):67–71, 1999.

[16] C. S. L. D. Library. http://citeseer.ist.psu.edu/.

[17] A. McCallum, K. Nigam, and L. H. Ungar. "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching". In *ACM KDD*, Boston, MA, Aug. 2000.

[18] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM Press.

[19] M. Michelson and C. A. Knoblock. "Learning Blocking Schemes for Record Linkage". In *AAAI*, 2006.

[20] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, Tuscon, AZ, May 1997.

[21] H. B. Newcombe and J. M. Kennedy. "Record Linkage". *ACM Comm. ACM*, 5:563–566, 1962.

[22] B.-W. On, D. Lee, J. Kang, and P. Mitra. "Comparative Study of Name Disambiguation Problem using a Scalable Blocking-based Framework". In *ACM/IEEE Joint Conf. on Digital Libraries (JCDL)*, Jun. 2005.

[23] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. "Identity Uncertainty and Citation Matching". In *Advances in Neural Information Processing Systems.* MIT Press, 2003.

[24] S. Sarawagi and A. Bhamidipaty. "Interactive Deduplication using Active Learning". In *ACM KDD*, 2002.

[25] J. W. Warnner and E. W. Brown. "Automated Name Authority Control". In *ACM/IEEE Joint Conf. on Digital Libraries (JCDL)*, 2001.