A Fast Linkage Detection Scheme for Multi-Source Information Integration

Akiko Aizawa
National Intsitute of Informatics /
The Graduate University for Advanced Studies
2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo 101-8430, Japan
aizawa@nii.ac.jp

Keizo Oyama
National Intsitute of Informatics /
The Graduate University for Advanced Studies
2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo 101-8430, Japan
oyama@nii.ac.jp

Abstract

Record linkage refers to techniques for identifying records associated with the same real-world entities. Record linkage is not only crucial in integrating multi-source databases that have been generated independently, but is also considered to be one of the key issues in integrating heterogeneous Web resources. However, when targeting large-scale data, the cost of enumerating all the possible linkages often becomes impracticably high. Based on this background, this paper proposes a fast and efficient method for linkage detection. The features of the proposed approach are: first, it exploits a suffix array structure that enables linkage detection using variable length n-grams. Second, it dynamically generates blocks of possibly associated records using 'blocking keys' extracted from already known reliable linkages. The results from our preliminary experiments where the proposed method was applied to the integration of four bibliographic databases, which scale up to more than 10 million records, are also reported in the paper.

1. Introduction

Record linkage is a problem of identifying a group of records from different sources that refer to the same real-world entity. The term 'record linkage' was used as early as the 1940s by Dunn [7] and Marshall [20], followed by the pioneering work by Newcombe et al. in 1959 [26]. Recently, the explosive growth of database resources, particularly the increase of legacy databases on the World Wide Web, has made the problem one of the central issues in data warehousing and data quality management.

Many recent studies, such as [3], have pointed out that similar types of problem have been described differently in different research fields, for example, *duplicate detection* [24, 2] or *deduplication* [27], which emphasizes the elimination of duplications in a database; *record matching* [31] or *approximate string join* [10], which is an operation to integrate two separate tables; *entity-name clustering and matching* [4], *object consolidation* [23] or *proper noun coreference analysis* [22], which concerns the coreference determination problem in text processing; and *entity identification* [18], *object identification* [30], *entity matching* [5] or *entity reconciliation* [6], which concerns heterogeneous data integration.

Table 1 illustrates the case where two bibliographic records refer to the same article, but with different attribute values. As shown in this example, real-world records usually contain 'noise', caused by such factors as notational variations, human input errors, or differences in editing policies. Therefore, constructing a practical linkage system often means the implementation of domain-specific knowledge to normalize the attribute values, or to evaluate the degree of match between the two records. In addition, the cost often becomes impracticably high, particularly for large databases that have been generated independently.

1. TAGUCHI Isao *Observation of Nonlinear Conduction in (NbSe_4)_3I : II. LOW TEMPERATURE PROPERTIES OF SOLIDS : Charge Density Waves* Japanese Journal of applied physics. Supplement vol. 26(3.pt1), pp.619-620 (1987)
2. TAGUCHI I *Observation of nonlinear conduction in (NbSe4)3I.* Jpn J Appl Phys Suppl no.26-3 Pt.1, pp.619-620 (1987)

Table 1. Two bibliographic records that refer to the same article.

There has been remarkable progress in recent years in approaches from the information retrieval (IR) and machine

learning (ML) fields [14]. Examples include the use of bigrams or Q-grams for possible linkage detection [3, 12, 11], ranking and scoring using the *tf-idf* similarity measure [4], applying learnable edit distance in matching score calculation [2], and the active learning adaptation to improve iteratively the performance of the linkage system [27]. It has been reported that these techniques enable less domain-dependent linkages while maintaining performance.

Our research was originally motivated by our experience with CiNii (Citation Information provided by National Institute of Informatics, http://ci.nii.ac.jp/index-e.html), where several bibliographic databases are integrated into a unified nationwide service for academic papers and their citations. Although the current implementation of CiNii employs the most modern techniques, including bigram-based search and support vector machines (SVMs), what has been learned in previous operations is that the system still requires considerable human review to maintain the quality of linkages. We seek a quality of: (i) less than 0.1% false matches and (ii) 1% missing linkages for databases with $10^5 - 10^7$ records.

Based on this background, this paper aims to establish a methodology to use and exploit recent IR and ML techniques for large-scale linkage problems with human review. Although our final goal is the linkage of general Web resources, for experimental purposes our focus in this paper is primarily on the linkage of legacy databases. In the future, we plan to combine the proposed method with existing text segmentation techniques, such as [29] and deal with a broader range of data resources.

The remainder of the paper is organized as follows: In section 2, a general procedure for record linkage is introduced and, focusing on the preselection process (referred to as 'blocking' in this paper), a brief overview of existing methods is given. In section 3, we propose a new blocking scheme that is characterized as follows. First, it exploits a suffix array structure to enumerate possible linkage candidates rapidly using variable length n-grams. Second, it dynamically generates blocks of records that are possibly linked to each other using 'blocking keys' extracted from the already known reliable linkages. Section 4 presents the results from our preliminary experimental studies, where the proposed method was applied to the integration of four bibliographic databases that scale up to more than 10 million entries. Section 5 presents our conclusions.

2. Background Issues

2.1. General Procedure for Record Linkage

The 'records' in this paper are the entries in databases with known, but not necessarily common, attributes. The two general requirements for record linkage systems are: (i) efficiency and scalability to deal with large-scale data, and (ii) accuracy to maintain the high quality of the data. However, these two requirements are contradictory and cannot be satisfied easily using a single-path method. It follows that record linkage systems in general employ multi-stage processing that includes (i) selection of possible linkages, (ii) comparison and judgment, and (iii) human review (Figure 1). Each is briefly explained below.

(1) Selection

Given a target dataset, record pairs that possibly refer to the same entities are enumerated first. This stage is also called *blocking* in this paper.

(2) Comparison and judgment

All the selected candidate pairs are labeled either as '*match*,' '*possible match*,' or '*non-match*' based on the matching score calculated using a predefined matching function. Here, '*match*' and '*non-match*' are the decisions with high confidence, and '*possible match*' indicates that the classifier is uncertain whether they represent correct linkages.

(3) Human review

Record pairs that are labeled as '*possible match*' are examined by human reviewers.

After stage (3), the detected duplicates are either the pairs that are automatically labeled as '*match*' at stage (2) or the pairs that are identified as positive by human reviewers at stage (3).

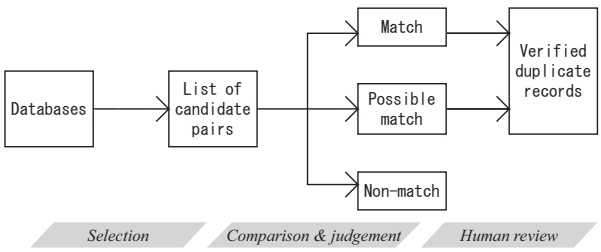
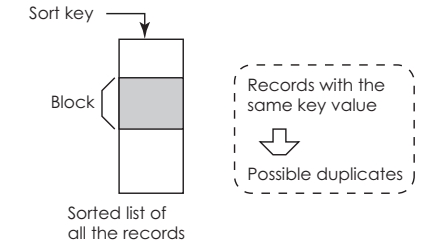
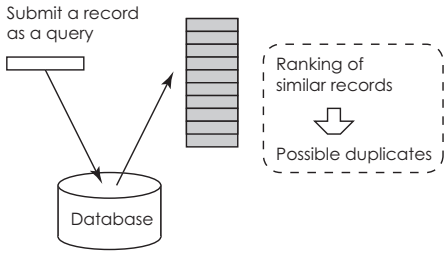


Figure 1. General record linkage procedure.

Throughout the process, the most time-consuming stage is the human review, stage (3). For example, in our system, a decision requires 10–20 seconds for a trained reviewer whether the result is positive or negative. It follows that the previous stages should be carefully designed so that they can significantly reduce the number of human judgments. For stage (2), to determine the optimal decision boundaries application of conventional machine learning techniques, such as SVM, is straightforward. Then, the critical issue becomes reducing the number of candidate linkages at stage (1) while maintaining as many positives as possible. Note



(a) Sorting method



(b) Ranking method

Figure 2. Sorting and ranking methods.

that applying the classifier to all the possible combinations of records is obviously not feasible. For example, record linkage between two databases each with 10^6 records yields $10^6 \times 10^6 = 10^{12}$ times comparisons of record pairs. Based on this, we specifically focus on the selection strategy at stage (1).

2.2. Overview of Existing Selection Methods

The basic strategy for candidate selection is to divide the target records into small blocks so that only pairs in the same block are considered at the later stages. The existing selection methods can be categorized into two: (a) sorting methods and (b) ranking methods. Each is briefly reviewed below.

Sorting methods Sorting methods are the traditional blocking schemes where the sorted list of records is first generated and then decomposed into different blocks (Figure 2-(a)). The simplest is *standard blocking* [17] where records are first sorted using a user-specified key (for example, 'the top four characters of one's family name'), and then, records with the same key value are grouped into a single block for further consideration. *Multiple pass blocking* [17] is where multiple keys are used for sorting. *SNM* (sorted neighborhood method) is a method that slides a fixed size window on the sorted list assuming that neighbors on the sorted list are also similar [15][16]. SNM has variations such as *clustering SNM* or *multi-pass SNM*. Other sorting methods include *priority queue method* [25], *k-way sorting method* [9], or *adaptive filtering method* [13].

Ranking methods Ranking methods consider the records as plain text and generate clusters of similar records by applying conventional information retrieval techniques (Figure 2-(b)). *TI-similarity* [28] first calculates the number of common characters for each attribute and then uses the summed value as a similarity measure. *Bigram indexing* [3] converts the key value into a set of character-based bigrams and then uses the permutations of subsets as keys for generating blocks. The lower bound of the subset size is determined by a specified threshold value. Positional *q-gram* [10, 12, 11] enables fast approximate string joins between records with a specified edit distance threshold value. *Canopy clustering* [4, 21] randomly selects a new record and generates a cluster of similar records using the *tf-idf* distance measure, widely used in IR. The generated clusters are called *canopies* and constitute blocks for further examination.

2.3. Observations

The sorting methods require domain-dependent knowledge to select appropriate keys and, for *SNM*, the optimal window size. Alternatively, the ranking methods do not require specific domain knowledge because they treat all the key attribute values as text. Nevertheless, the performance of the sorting methods are not necessarily better than the ranking methods because neighbors on the sorted list do not usually cover all the similar records even when multiple pass sorting is applied. A previous study [1] compared the performance of *standard blocking*, *SNM*, *bigram indexing*, and *canopy clustering*. It reported that the latter two, today's representative ranking methods, significantly outperform the former, the traditional sorting methods.

The computational cost for the sorting methods is simply that of sorting all the target records. Moreover, the sorting

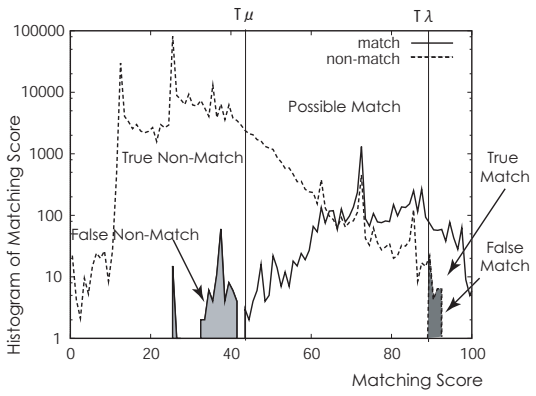


Figure 3. Distribution of the matching score.

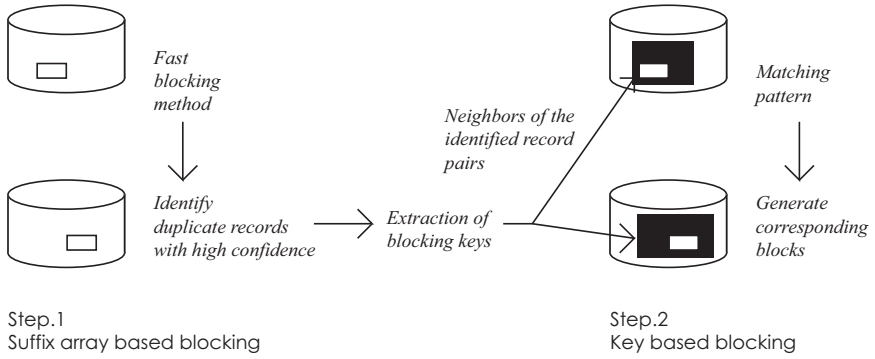


Figure 4. Basic idea of the proposed method.

operation is applied only once for the entire set of records. On the other hand, the cost for the ranking methods is the cost of indexing, retrieving, and ranking the records. The retrieval operation must be applied iteratively for all the possible blocks, the number of which is approximately in proportion to the total number of records. Therefore, even with today's search engine techniques, the cost for ranking methods becomes significant for large-scale data, due to the exponential property of frequencies known as Zipf's Law.

Another important issue is the cost of human review. As an illustrative example, Figure 3 shows the histogram of the matching scores obtained from the actual data used in our experiments. Assume that we choose T_μ and T_λ as boundaries for three decision regions, '*non-match*', '*possible match*', and '*match*.' As has been already described, the candidates labeled as '*possible match*' require human review, and the left- and right-side colored regions represent different types of linkage errors, i.e., '*false non-match*' and '*false match*,' respectively. From the figure, it becomes clear that the best performance we can expect from a classifier is determined a priori by the score distribution after the selection. Obviously, the total number of human judgments is most sensitive to the number of less-promising negative candidates (note that the vertical axis of Figure 3 is log-scaled and the majority of the samples are negative). However, neither conventional sorting nor ranking methods explicitly considers the elimination of these negative candidates.

3. The Proposed Method

3.1. Basic Idea

To summarize the above observations, the advantages and disadvantages of the existing methods are as follows.

- The sorting methods are scalable to the data size, but the ranking methods perform better in terms of the quality of the detection.
- For both methods, the cost of human review cannot be significantly reduced using conventional classifiers.

Based on this, we propose a new selection method that combines two newly developed blocking schemes: *suffix array-based blocking* and *key-based blocking* (Figure 2.2).

- *Suffix array-based blocking* provides a fast and domain-independent selection function by exploiting variable length n-grams on a suffix array representation. This enables us to identify, quickly though not exhaustively, linkage candidate pairs. Then, the detected pairs are evaluated using a specified scoring function to identify reliable positive pairs.
- *Key-based blocking* provides a way to resample the candidates using matching patterns, called '*blocking keys*', obtained from the reliable pairs. Records that match the same blocking key are grouped into the same block for further comparison and judgment.

In the proposed method, suffix array-based blocking is used to extract domain-specific transformation rules for the key-based blocking, and key-based blocking is used to reactivate candidate pairs that were overlooked by the suffix array-based blocking. By this combination, the majority of the negative candidates shown in Figure 2.2 can be eliminated. Each blocking scheme is explained in the following.

3.2. Suffix Array-Based Blocking

A suffix array is a sorted list of all the subsequences of tokens that appear in the text (Figure 3.1). Given a set of records for linkage, the corresponding suffix array structure is generated as follows.

(1) Tokenize all the key attribute values. Tokens are either characters or terms, depending on the types of the attributes and/or languages.

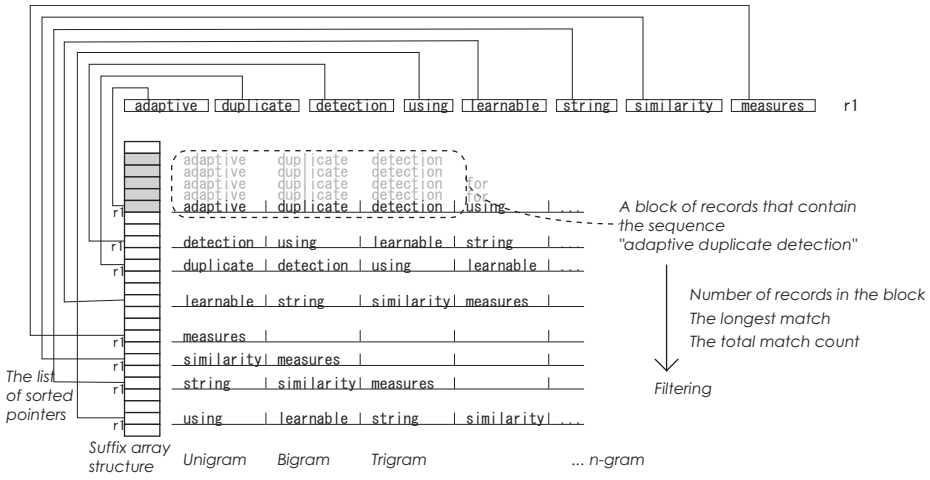


Figure 5. Suffix array-based blocking.

(2) Represent each record as a single token sequence. Here, the key attributes are arranged in a specified order with delimiters between them. Start and stop symbols are added at the beginning and the end of the sequence.

(3) Sort all the pointers of the tokens in alphabetical order of the token sequences.

Let $(t_1, \cdots, t_n)$ be a token sequence of length n (i.e., n-gram). Then, all the records that contain $(t_1, \cdots, t_n)$ are expressed as adjacent members of the suffix array. Conversely, the adjacent members of the suffix array who have the same top n tokens $(t_1, \cdots, t_n)$ catalog all the records in the database that contain $(t_1, \cdots, t_n)$. It follows that we can exhaustively check arbitrary length token sequences simply by iteratively scanning the suffix array structure. With the proposed *suffix array-based blocking* (SAB), such adjacent members of the suffix array are considered 'blocks.' We select only such blocks that satisfy the following conditions.

(1) The size of the block is between α_l and α_u.

(2) The length of the longest common token sequence is not less than β.

(3) The total number of common tokens is not less than γ.

Here, (3) is calculated by applying pairwise DP matching to the members of the block. The parameters α_l, α_u, β, and γ were determined heuristically in later experiments. The computation time of the proposed SAB is basically that needed for the sort operation to generate the suffix array structure. That is, with the built-in standard quick sort function, $O(n\ log(n))$, for text with n tokens.

There are several features of SAB. First, unlike existing ranking methods, which require an index search for every block generation, SAB collectively generates the entire blocks simultaneously. Next, unlike existing sorting methods that use predetermined sorting keys, SAB can dynamically select keys as arbitrary length n-grams that satisfy the conditions above. These features are particularly advantageous when dealing with large-scale data.

3.3. Key-Based Blocking

Key-based blocking (KB) collects neighbors of the reliable linkages using '*blocking keys*' that represent correspondence between specific attribute values. Given a list of key attributes, those blocking keys are automatically extracted from the pairs detected by SAB (Figure 4).

The KB method can be considered to be a variation of knowledge-based methods [15, 16, 19] that employ equivalence rules between the corresponding attribute values. The difference is that KB automatically extracts the blocking keys, whereas most of the knowledge-based methods use equivalence rules given by human experts. Because each blocking key is connected to a distinctive block, they can be justified easily and evaluated based on the matching scores of the supporting record pairs, or verified by human reviewers. The framework of KB is also similar to that of the active learning proposed in [27]. The difference is that KB utilizes highly reliable positive examples, whereas active learning investigates neutral examples so that the boundary of the classifier can be refined further.

4. Experimental Results

4.1. Conditions for the Experiments

In our experiments we used four bibliographic databases that were provided in our service operation in cooperation

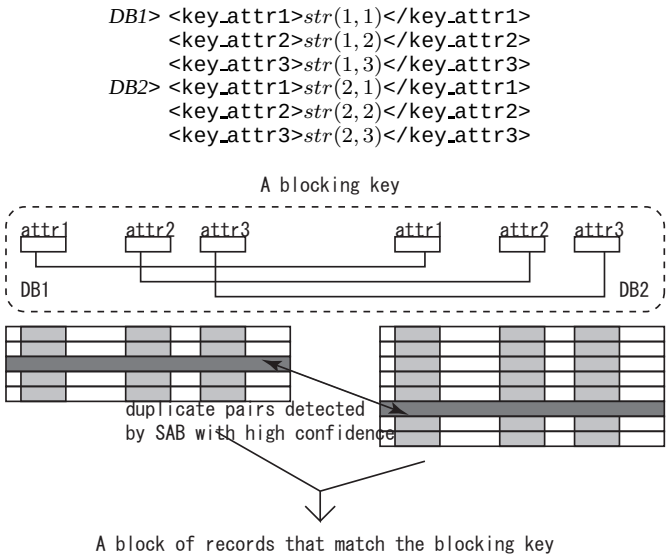


Figure 4. Key-based blocking.

with other institutions. The databases contain periodicals published by Japanese academic societies. Table 2 shows the total number of records used in the experiments. Because the four databases come from different sources, a considerable number of records were registered in the multiple databases. The objective of the experiment was to detect those duplicated entries so that the databases can be integrated with a unified service framework. The linkage process follows the general procedure that has been shown in Figure 1. Namely, the proposed blocking methods were first applied, and next, the detected candidate pairs were evaluated using a given matching function. Then, the candidate pairs were automatically labeled as either '*match*', '*possible match*', or '*non-match*.' Finally, all the pairs labeled as '*possible match*' were examined by human reviewers to ascertain whether they refer to the same article. In the experiments, the linkage was applied to the following six datasets: {(*DB1-DB2*), (*DB1-DB3*), (*DB1-DB4*), (*DB2-DB3*), (*DB2-DB4*), (*DB3-DB4*)}. [1]

Database ID	Number of records
DB1	623,996
DB2	1,345,570
DB3	5,977,839
DB4	15,464,052

Table 2. Target databases.

1 Part of the *DB1-DB2*dataset (43,618 records) was used for initial check and tuning and was subtracted from the remaining dataset.

When preparing the data, the attribute values were first normalized using general substitution rules (for example, uppercase to lowercase conversion), and then segmented into tokens, either characters or words, depending on the attribute types and the description languages (for example, names written in Japanese characters were segmented into characters, and the titles were segmented into words). In our experiments, we selected the following attributes: the name of the first author, the title of the article, the title of the journal, the volume and the number of the issue, the pages, and the year. For the first three, both the English and the Japanese descriptions were considered. In addition to these attributes, human reviewers may refer to other information sources as well, including the electronic or printed distribution of the articles. Also considered in human judgment were such factors as changes in journal titles, proceedings of the joint workshops, or reprinted articles. The matching function used for comparison, as well as the threshold values of the classifier, was determined heuristically because the dominant factor for performance in this case was the quality of the selection rather than the performance of the classifier.

Because different databases follow different notations and conventions, the target problem is not an easy task. Taking a real example, of all the record pairs identified as duplicates by human reviewers, only 10% had their authors and titles match exactly after normalizing. Conversely, when the normalized authors and titles were used as keys for sorting, only 30% of the known duplicates were detected and 7% of these were false matches. Considering that our target performance value is 0.1% false-match errors and 1% false-non-match errors, we concluded that simple conventional sorting methods were not applicable to the datasets. In addition, due to the large scale of the problem, applying conventional ranking methods was not feasible with the limited computing resources available in our experimental environment. For these reasons, in this paper we only give the preliminary results that show the effect of the proposed SAB and KB. Detailed comparisons have been left for future study.

4.2. Selection Methods

In the experiment, we investigated three alternative methods: *Exact&KB*, *SAB&KB*, and *SAB-alone* (Figure 5). Each method is described briefly in the following.

Exact&KB With this method, all the pairs whose normalized authors and titles exactly match were first identified. Then, corresponding blocking keys were extracted. As the key attributes for KB, 'the journal title' and 'volumes and numbers of the issue' were used. Because only limited pairs were identified by *Exact*, we applied a simple and ad hoc

generalization to the extracted keys; that is, to substitute the volumes and numbers by variables. Using this generalization, the *Exact* method could compensate for the scarceness of the samples.

Example of a generalized blocking key:

DB2> `<journal>JSME international journal. Series C, Mechanical systems, machine elements and manufacturing</journal><voln>$X ($Y)</voln>`
DB4> `<journal>JSME Int Journal. Ser C. Mech Systems, Mach Elem Manuf</journal><voln>$X ($Y)</voln>`

Later, the instantiated variables of the generalized blocking keys were checked and inappropriate correspondences were excluded. The threshold values for *KB* to decide '*match*', '*possible match*', and '*non-match*' were adjusted so that: (i) all the '*match*' cases were correct, and (ii) as many candidate pairs as possible were labeled as '*possible-match*' so long as they were in the same block.

SAB&KB With this method, the pairs extracted by *SAB* were also used to generate blocking keys. The parameters for SAB were $\alpha_l = 2$, $\alpha_u = 4$, $\beta = 4$, and $\gamma = 12$. The following is an example of a blocking key obtained for (*DB2-DB4*).

Example of a blocking key:

DB2> `<journal>The Journal of Toxicological Sciences</journal><voln>21 (Supplement I)</voln>`
DB4> `<journal>J Toxicol Sci</journal><voln>21 (Suppl 1)</voln>`

Note that there exist approximately 100,000 journal titles in total, and even for a single title, manually encoding all the correspondence patterns sometimes incurs considerable costs. For example, the exceptional cases include proceedings of joint workshops, special and seasonal editions, and supplementary issues. The correspondences between the journal issues specified by the generated blocking keys were verified later by human reviewers.

SAB-alone For comparison purpose, we have also examined the case where the candidate pairs were directly extracted using SAB. The parameters for SAB were, again, $\alpha_l = 2$, $\alpha_u = 4$, $\beta = 4$, and $\gamma = 12$.

In the experiment, *SAB&KB* was applied complementarily to *Exact&KB*, which means only those pairs that were not detected by *Exact&KB* were reported for human review. On the other hand, the results from *SAB-only* contained so many negative pairs that human review was not feasible. Consequently, *SAB&KB* covers almost all the known duplicates. As part of the CiNii operation, bigram-based blocking using an XML database engine had been separately carried out for the dataset (*DB2-DB4*), the result of which is available for comparison.

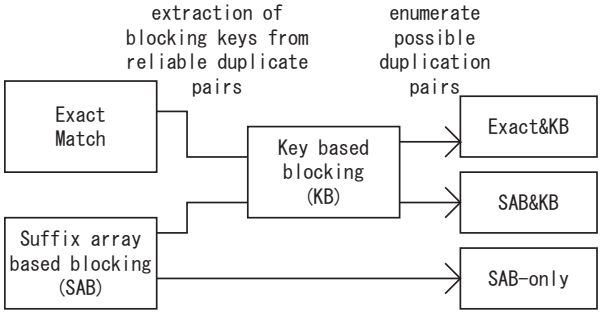


Figure 5. Methods used in the experiments.

4.3. Results

In our evaluation, we examined: (i) the execution time for SAB, (ii) the number of blocking keys automatically extracted by KB, and (iii) the number of pairs that required human review for each method as well as the errors in the detection.

Execution time for SAB Table 3 shows the total number of records, the number of detected candidate record pairs, and the execution time of *SAB*, for the six datasets. The execution was on-memory using a single CPU of Sun Fire V880 (900MHz, 64GBytes memory). The *Qsort* function of the standard C library was used. The relationship between the execution time and the total number of records is also shown in Figure 6.

Dataset name	Total number of records	Number of candidate pairs	Time (min.)
DB1-DB2	1,969,566	1,168,521	72
DB1-DB3	6,601,835	994,272	142
DB1-DB4	16,088,048	311,951	469
DB2-DB3	7,323,409	1,228,011	198
DB2-DB4	16,809,622	2,295,823	576
DB3-DB4	21,441,891	5,086,244	591

Table 3. Execution time for SAB.

As a comparison, it took approximately 0.2 seconds with (*DB3-DB4*) dataset for the XML database engine with bigram index to generate a candidate block given a target record (using HPC2500 32CPU/128GB cluster PC). This means that the blocking time totals approximately two weeks when all the target record is examined. Because the blocking time of SAB for the same dataset was approximately 10 hours, we concluded that the computation cost of SAB was significantly reduced.

Keys extracted by KB Table 4 compares the number of extracted blocking keys using *Exact&KB* and *SAB&KB*, re-

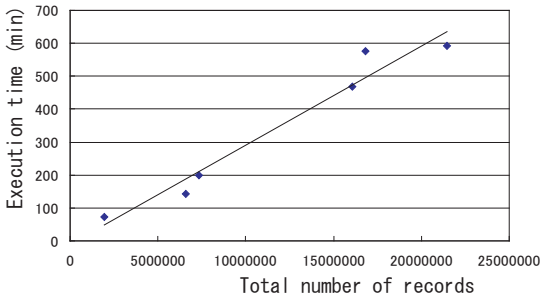


Figure 6. Execution time for SAB for different dataset sizes.

spectively. The figures in the parentheses for *Exact&KB* are the numbers of manually defined patterns for the key generalization. A considerable number of additional keys were obtained by *SAB*.

Dataset Name	Keys obtained by Exact		Additional keys obtained by SAB
DB1-DB2	148	(4)	428
DB1-DB3	612	(6)	5,552
DB1-DB4	806	(8)	2,189
DB2-DB3	464	(10)	5,172
DB2-DB4	356	(9)	900
DB3-DB4	6,016	(12)	38,614

Table 4. Number of blocking keys.

Performance comparison Table 5 summarizes the performance of *Exact&KB*, *SAB&KB*, and *SAB-only*. The table shows: (i) the number of the pairs that were identified as duplicates by the classifier, (ii) the number of the pairs that required human review, (iii) the number of the pairs that were falsely identified as duplicates, and (iv) the total number of correctly identified duplicate pairs.

The figures in the parentheses for (iii) and (iv) represent the detected candidates as a percentage of the known duplicates found by *SAB&KB*. Note that human reviews were applied only to those pairs that had been found by SAB&KB. For *Exact&KB* and *SAB&KB*, all the pairs labeled as '*match*' by the classifier were automatically considered as correct. The justification was also confirmed by the human reviewers. For these cases, (iii) also includes the cost of checking the correspondences of blocking keys automatically extracted. For *SAB-only*, (iii) represents the number of records with at least one duplication candidate. This is because human reviewers check those candidates simultaneously on the same screen.

For *SAB-only*, the following three ideal cases were compared: first, all the detected candidates were considered as '*possible match*.' The ratio of falsely identified pairs (p_1), and the ratio of missed duplications (p_2) were not specified. The threshold values for '*match*', '*possible match*', and '*non-match*' were simply determined as the highest and the lowest scores of the negative and the positive examples, respectively. In the second case, p_1 was set to 0.1% and the threshold value for '*match*' and '*possible match*' were optimized so that the number of '*possible match*' was minimized given p_1. p_2 was not specified. In the third case, p_1 was maintained as 0.1%, and p_2 was set to 5%. In this case, the threshold values for '*possible match*' and '*non-match*' were optimized so that the number of '*possible match*' was minimized given p_1 and p_2.

To summarize the results in Table 5, *SAB&KB* showed the best performance, showing the effectiveness of combining these two blocking schemes. In addition, the following observations were made.

- *SAB&KB* successfully found additional duplicates that could not be found by *Exact&KB*. This is also shown in Table 4 where considerable numbers of additional blocking keys were obtained by *SAB&KB*. Note that the performance of *Exact&KB* depends on the ad hoc generalization of the blocking keys. Without this generalization, the performance of *Exact&KB* was not comparable.
- The cost of human review was much smaller with *SAB&KB* and *Exact&KB* than with *SAB-only*. *SAB-only* by itself can detect approximately 95% –99% candidate pairs found by *SAB&KB*. However, the number of human reviews was not drastically reduced with *SAB-only*, even when the quality requirement was relaxed.

Finally, the existing CiNii system found an additional 476 duplicates for the dataset (*DB2-DB4*) after 262,829 human judgments. This means that the estimated false-non-match ratio for *SAB&KB* was approximately 0.2%. Conversely, *SAB&KB* found 1,471 duplications that were not identified by the existing CiNii system only after 15,715 human judgments. The figures demonstrated the effectiveness of the proposed *SAB&KB*. In the future, we plan to carry out further comparison with existing blocking methods using more comprehensive human review results.

5. Conclusion

In this paper, we propose a new blocking scheme for large-scale record linkage problems. Through the preliminary experiments using real-world bibliographic databases, the effectiveness of the proposed method both in computa-

Method	Dataset name	Automatically identified duplicates	Total number of human judgments	Falsely identified pairs (%)	Total number of correctly identified duplicates (%)
SAB-only	DB1-DB2	1,058	308,746	0 (0.0)	88,655 (97.3)
p_1 = *	DB1-DB3	1,230	500,890	0 (0.0)	265,175 (98.9)
p_2 = *	DB1-DB4	152	505,367	0 (0.0)	325,198 (99.2)
	DB2-DB3	2,226	622,587	0 (0.0)	184,390 (95.6)
	DB2-DB4	871	661,728	0 (0.0)	199,441 (98.4)
	DB3-DB4	1,500	2,981,888	0 (0.0)	1,339,823 (99.4)
SAB-only	DB1-DB2	60,503	249,301	91 (0.1)	88,655 (97.3)
p_1 = 0.001	DB1-DB3	21,260	480,860	268 (0.1)	265,175 (98.9)
p_2 = *	DB1-DB4	3,072	502,447	327 (0.1)	325,198 (99.2)
	DB2-DB3	3,502	621,311	192 (0.1)	184,390 (95.6)
	DB2-DB4	44,544	618,055	202 (0.1)	199,441 (98.4)
	DB3-DB4	14,736	2,968,652	1,348 (0.1)	1,339,823 (99.4)
SAB-only	DB1-DB2	60,503	27,864	91 (0.1)	86,564 (95.0)
p_1 = 0.001	DB1-DB3	21,260	235,922	268 (0.1)	254,722 (95.0)
p_2 = 0.05	DB1-DB4	3,072	326,023	327 (0.1)	311,324 (95.0)
	DB2-DB3	3,502	189,935	192 (0.1)	183,230 (95.0)
	DB2-DB4	44,544	153,370	202 (0.1)	192,475 (95.0)
	DB3-DB4	14,736	1,295,445	1,348 (0.1)	1,280,674 (95.0)
Exact&KB	DB1-DB2	71,856	4,561	0 (0.0)	76,124 (83.5)
	DB1-DB3	216,155	14,525	0 (0.0)	225,899 (84.3)
	DB1-DB4	299,356	13,863	0 (0.0)	312,810 (95.5)
	DB2-DB3	157,304	14,105	0 (0.0)	169,996 (88.1)
	DB2-DB4	179,920	13,134	0 (0.0)	192,729 (95.1)
	DB3-DB4	1,068,498	96,473	0 (0.0)	1,164,971 (86.4)
SAB&KB	DB1-DB2	86,051	5,931	0 (0.0)	91,121 (100.0)
	DB1-DB3	254,713	21,877	0 (0.0)	268,129 (100.0)
	DB1-DB4	314,453	20,318	0 (0.0)	327,710 (100.0)
	DB2-DB3	182,779	17,418	0 (0.0)	192,874 (100.0)
	DB2-DB4	190,637	15,715	0 (0.0)	202,606 (100.0)
	DB3-DB4	1,256,090	129,507	0 (0.0)	1,348,078 (100.0)

Table 5. Performance comparison.

tion time and in the cost of human review has been demonstrated. Although the experiments in this paper mainly focus on records of traditional databases, we expect that the proposed method will be an initial step towards practical Web resources exploitation, in that:

(i) the matching results enabled us to generate large-scale dictionaries of entities with their notational variations and acronyms. In addition, heuristic normalization rules can automatically be identified. Such resources are in most cases crucial in identifying and analyzing descriptions of entities that appear in unformatted text on the Web.

(ii) because the proposed SAB method considers each record as plain text, the method is directly applicable to semi-structured or unstructured data, the examples of which include home pages of researchers or research organizations, references to full-text papers, and other bibliographic collections that constitute the academic infrastructure on the Web.

Future work includes incorporating machine learning techniques to the extraction of blocking keys and automatic acquisition of normalization dictionaries from the verified linkages.

Acknowledgments

The work presented here has been partially supported by the Grant-in-Aid Scientific Research on Priority Area “Informatics” (Area #006) funded by MEXT, Japan. The authors would like to thank Jun Adachi, Atsuhiro Takasu and other colleagues at NII for their useful comments and discussions.

References

[1] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. *Proc. of ACM SIGKDD’03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (KDD2003)*, pages 39–48, 2003.

[3] P. Christen and T. Churches. Febrl – freely extensible biomedical record linkage. *Computer Science Technical Reports, TR-CS-02-05, Australian National University*, 2000.

[4] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (KDD2002)*, pages 475–480, 2002.

[5] D. Dey, S. Sarkar, and P. De. A probabilistic decision model for entity matching in heterogeneous databases. *Management Science*, (10):1379–1395, 1998.

[6] D. Dey, S. Sarkar, and P. De. A distance-based approach to entity reconcilation in heterogeneous databases. *IEEE Transactions on Knowledge and Data Engineering*, (3), 2002.

[7] H. L. Dunn. Record linkage. *American Journal of Public Health*, (36):1412–1416, 1946.

[8] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.

[9] A. Feekin and Z. Chen. Duplicate detection using k-way sorting method. *Proceedings of the ACM Symposium on Applied Computing (SAC) 2000*, pages 323–327, 2000.

[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 491–500, 2001.

[11] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins for data cleansing and integration in an rdbms. *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE 2003)*, pages 729–731, 2003.

[12] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. *Proceedings of the 12th ACM International Conference on World Wide Web (WWW 2003)*, pages 90–101, 2003.

[13] L. Gu and R. Baxter. Adaptive filtering for efficient record linkage. *Proceedings of the 2004 SIAM International Conference on Data Mining (SDM04)*, 2004.

[14] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. *CMIS Technical Report 03/83*, 2003.

[15] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD 1995)*, pages 127–138, 1995.

[16] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Journal of Data Mining and Knowledge Discovery*, (1):9–37, 1998.

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

[18] E. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. *Proceedings of the 9th IEEE International Conference on Data Engineering (ICDE 1993)*, pages 294–301, 1993.

[19] W. L. Low, M. L. Lee, and T. W. Ling. A knowledge-based approach for duplicate elimination in data cleaning. *Information Systems*, pages 585–606, 2001.

[20] J. T. Marshall. Canada's national vital statistics index. *Population Studies*, (2):204–211, 1947.

[21] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2000)*, pages 169–178, 2000.

[22] A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. *Proceedings of the 2003 IJICAI Workshop on Information Integration on the Web (IIWeb-03)*, page 2003.

[23] M. Michalowski, S. Thakkar, and C. A. Knoblock. Exploiting secondary sources for automatic object consolidation. *Proceedings of the ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[24] A. E. Monge. Matching algorithms within a duplicate detection system. *IEEE Transaction on Data Engineering*, (4):14–20, 2000.

[25] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. *Proceedings of the ACM-SIGMOD Workshop on Research Issues on Knowledge Discovery and Data Mining*, 1997.

[26] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, (3381):954–959, 1959.

[27] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (KDD2002)*, pages 269–278, 2002.

[28] S. Y. Sung, Z. Li, and P. Sun. A fast filtering scheme for large database cleansing. *Proceedings of the 2002 ACM Conference on Information and Knowledge Management (CIKM 2002)*, pages 76–83, 2002.

[29] A. Takasu. Bibliographic attribute extraction from erroneous references based on a statistical model. *Proceedings of the Joint Conference on Digital Libraries (JCDL) 2003*, pages 49–60, 2003.

[30] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 350–359, 2002.

[31] V. S. Verykios, A. K. Elmagarmid, and E. N. Houstis. Automating the approximate record matching process. *Journal of Information Sciences*, (1-4):83–98, 2000.