Faculty of Library, Information and Media Science, University of Tsukuba
Technical Report

Efficient Filtering and Ranking Schemes for Finding Inclusion Dependencies on the Web

Atsuyuki Morishima*, Erika Yumiya**, Masami Takahashi**,
Shigeo Sugimoto*, Hiroyuki Kitagawa***
*Faculty of Library, Information and Media Science, University of Tsukuba
**Graduate School of Library, Information and Media Studies, University of Tsukuba
***Faculty of Engineering, Information and Systems, University of Tsukuba

SLIS-TR-2014-001

Abstract

Data integrity constraints are fundamental in various applications, such as data management, integration, cleaning, and schema extraction. This paper presents the results of a first comprehensive study on finding *inclusion dependencies* on the Web. The problem is important because (1) applications of inclusion dependencies, such as data quality management, are beneficial in the Web context, and (2) such dependencies are not explicitly given in general. In our approach, we enumerate pairs of HTML/XML elements that possibly represent inclusion dependencies and then rank the results for verification. First, we propose a bit-based signature scheme to efficiently select candidates (element pairs) in the enumeration process. The signature scheme is unique in that it supports Jaccard containment to deal with the incomplete nature of data on the Web, and preserves the semiorder inclusion relationship among sets of words. Second, we propose a ranking scheme to support a user in checking whether each enumerated pair actually suggests inclusion dependencies. The ranking scheme sorts the enumerated pairs so that we can examine a small number of pairs for simultaneously verifying many pairs. Finally, we prove that there exist efficient algorithms for the ranking scheme. In addition to the theoretical results for the signature and ranking schemes, we present a comprehensive set of experimental results using various real Web sites. The results show that in the enumeration process the signature scheme reduces the number of candidate pairs *by orders of magnitude*, and that the ranking scheme allows a small number of higher ranked results to cover many other pairs.

1 Introduction

Data integrity constraints are fundamental in computer data management. They have been used in many applications, such as data integrity management, data integration, data cleaning, and schema extraction [7] [10]. In particular, two types of constraints, *functional and inclusion dependencies*, are widely used. A functional dependency states that a set of values that appear in some part determines values in other places. An inclusion dependency states that a set of data items must be a subset of the set of data items in another place. In other words, it states that if a data item appears in a place, the same item must appear in another place.

We focus on the problem of supporting the discovery of inclusion dependencies among HTML/XML elements on the Web. Inclusion dependencies are important both from practical and theoretical viewpoints. First, we see many inclusion dependencies among data in Web sites, such as among lists of publications or members, and those with sets of sentences taken from an original document. For example, Figure 1 shows a pair of Web pages that show an inclusion dependency on the Web. They are taken form ACM SIGWEB and SIGIR Web sites, each of which lists the winners of Vannevar Bush best paper award. The SIGWEB page on the left has a (nested) element, which contains a part of the complete list shown in the SIGIR page on the right. The elements are structured with different tags (table and items) but they are similar to each other in the hierarchical structure. It is common to see such inclusion dependencies appear in the Web sites maintained by different administrators. Second, an inclusion dependency is theoretically important because it is a generalization of the equivalence constraint, which is a universal constraint that pervasively appears in many applications involving Web sites having the same data but maintained by different administrators. The problem is that the data often violate the dependencies because of ill maintenance. In fact, the existence of related data maintained by different administrators is one of the causes to degrade the data quality on the Web. For example, [18] reports that in a set of real estate Web sites, they found about 60% of data items with some inconsistency.

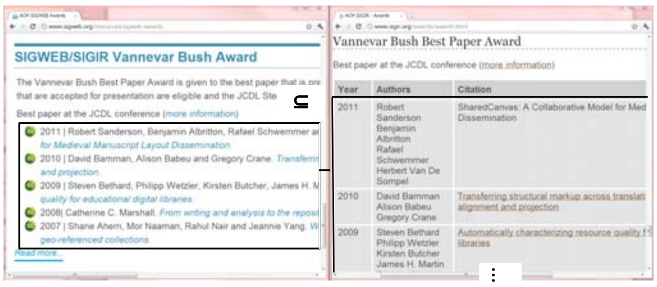
Applying data integrity constraints to fix such inconsistencies have been widely discussed in various data integration and management problems [6]. Assume that we know inclusion dependencies among the Web data that are related to each other but are maintained by different administrators. Then, we can use them to maintain the quality of the contents by (1) finding the portions that violate the inclusion dependency, (e.g., a name does not appear in another place) and (2) fixing the violations by updating the values. Therefore, inclusion dependencies can be one of the key technologies to improve data quality on the Web.

Since data integrity constraints have such important applications, there have been many attempts to explicitly deal with data integrity constraints on the Web. For example, XML Schema [9] introduces the key and foreign key constraints in XML elements, which are variations of functional and inclusion dependencies.

A widely known problem related to data integrity constraints is that they are not always given in an explicit manner [11]. Therefore, many studies have addressed the problem of helping users find integrity constraints from an existing data instance. However, most existing techniques address the problem of supporting the discovery of data integrity constraints in the context of relational databases [1] [2]. To the best of our knowledge, only a few papers address the problem of supporting the discovery of data integrity constraints in the Web context. One such study discusses how to find functional dependencies in an XML document [13].

This paper is the first to show the results of a comprehensive study on finding inclusion dependencies on the Web. Because it is inevitable that finding all inclusion dependencies in a given data set yields false positives, a common approach is first to enumerate all possible candidates (including false positives) [2] and then to verify the enumerated candidates. This paper discusses algorithms for this two-phase approach in the Web context.

Our challenge is to develop efficient schemes that can deal with the characteristics of Web content, i.e., we need to address *both* of the following requirements: (1) **Efficiency.** We want the scheme to be efficient, because the number of Web pages can be large. (2) **Dealing with the characteristics of Web content.** We want the scheme to be able to deal with the characteristics of Web content, because Web pages have hierarchical structures and their data are not necessarily clean. To our



SIGWEB/SIGIR Vannevar Bush Award

The Vannevar Bush Best Paper Award is given to the best paper that is pre
that are accepted for presentation are eligible and the JCDL Ste

Best paper at the JCDL conference (more information)

- 2011 | Robert Sanderson, Benjamin Albritton, Rafael Schwemmer ar *for Medieval Manuscript Layout Dissemination.*
- 2010 | David Bamman, Alison Babeu and Gregory Crane. *Transferrin and projection.*
- 2009 | Steven Bethard, Philipp Wetzler, Kirsten Butcher, James H. M *quality for educational digital libraries.*
- 2008| Catherine C. Marshall. *From writing and analysis to the reposi*
- 2007 | Shane Ahern, Mor Naaman, Rahul Nair and Jeannie Yang. *W geo-referenced collections.*

Read more...

$\subseteq$

Vannevar Bush Best Paper Award

Best paper at the JCDL conference (more information)

Year	Authors	Citation
2011	Robert Sanderson Benjamin Albritton Rafael Schwemmer Herbert Van De Sompel	SharedCanvas: A Collaborative Model for Med Dissemination
2010	David Bamman Alison Babeu Gregory Crane	Transferring structural markup across translati alignment and projection
2009	Steven Bethard Philipp Wetzler Kirsten Butcher James H. Martin	Automatically characterizing resource quality f libraries

Figure 1: Example of inclusion dependency on the Web

knowledge, there have been no schemes that address both of the requirements.

The contributions of this paper are as follows. First, this paper introduces a bit-based signature scheme to efficiently deal with *Jaccard containment* [1], which is an asymmetric version of the ordinary Jaccard coefficient, in order to enumerate inclusions with the incomplete data on the Web. In general, a bit-based signature is a bit sequence associated to each data item, and has been used for efficiently determining whether each item satisfies a given condition. In our context, a bit-based signature is associated to each HTML element to concisely represent information required to compute the inclusion relationship with other elements. The proposed mechanism for Jaccard containment is unique in that the signature has a fixed length and is designed to preserve a semiorder represented by the inclusion relationship. To our knowledge, there have been no such signature schemes.

Second, we discuss a ranking scheme to aid in verifying the candidate inclusion dependencies. We introduce the notion of *covers* to efficiently examine the enumerated inclusions, and prove that there is an efficient algorithm to compute probabilities of inclusions that are compatible with the definition of covers. Then, we propose a re-ordering scheme for the algorithm's outputs in order to obtain better rankings.

Finally, we discuss the results of a comprehensive set of experimental evaluations using various real Web sites. The results show that the signature scheme reduces the search space by orders of magnitude and the ranking scheme allows us to examine only a small number of candidates for verifying many candidates.

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 defines our problem. Section 4 develops the bit-based signature scheme for dealing with the Jaccard containment to enumerate inclusions as candidates of inclusion dependencies. Section 5 discusses how to rank the enumerated candidates. Section 6 evaluates the proposed schemes, and finally, Section 7 concludes the study.

2 Related Work

In the context of relational databases, there are already numerous studies about computing inclusions, i.e., asymmetric set containments. Bauckmann and others [2] proposed an algorithm that takes as input a set of relations and efficiently enumerates all pairs of relational attributes one of which includes the other. The algorithm is designed to minimize the amount of I/O over the sets of attribute values. Sergey Melnik and others [12] proposed two hash-based partitioning algorithms called the Adaptive Pick-and-Sweep Join (APSJ) and the Adaptive Divide-and-Conquer Join (ADCJ), to efficiently compute set containment joins. The algorithms above are designed to compute strict inclusions in flat relations under the assumption that the data is clean.

Recently, finding inclusions based on Jaccard containment is attracting attention in the research community. This is because there are many applications in which we need to relax the assumption that the data is clean. An approach is to use the prefix filtering [16] [4], which is a run-time optimization technique for similarity joins. It uses the prefix of each data set sorted in some order in the join process. [1] points out that the prefix filtering is not appropriate to be used as the basis of the design of structured index for Jaccard containment and proposes to use the notion of minimal infrequent sets to construct the index. The size of the index can be exponential in the record size, but it is reported that the size is often much smaller in practical applications. To our knowledge, there are no signature schemes to deal with Jaccard containments although both of the structured and signature-based indices are known important to support various types of applications. Again, the prefix filtering is not appropriate to be used as the basis of the design of the signatures, since it requires the threshold for Jaccard containment to determine the size of prefix. Another approach is to use estimators. For example, in [21], bottom-k sketches are used as estimators for Jaccard containment and then, foreign key constraints were determined using a criterion called randomness. The bottom-k sketches are similar to the prefix filtering but do not guarantee 100% recall because k is determined independent of Jaccard containment.

Bit-based signatures to efficiently support (exact) set-containment queries were studied in [8]. However, they can deal only with exact set containments, and as discussed in Section 4, it is not trivial to develop a signature scheme to deal with non-exact set containments. Our bit-based signature scheme is the first one that can deal with Jaccard containment and is unique in that it has all the following properties: (1) It employs fixed-length signatures, (2) the signature is general enough to support any Jaccard containments, and (3) it guarantees 100% recall.

Although asymmetric measures like Jaccard containment have been discussed mainly in the database context, *symmetric* similarities have been discussed in the literature in the context of the Web. [15] showed that Charikar's simhash [3] is practically useful for identifying near-duplicates in Web pages. [19] proposes the positional filtering to support efficient similarity joins for near duplicate detection, which can be used to find near-duplicate Web pages. However, symmetric similarity cannot capture asymmetric measures (such as containment) in general (Note that there are many cases in which X is contained in Y but X and Y do not have a high similarity score.) As suggested in [1], there are scenarios in which an asymmetric measure is more appropriate. Finding inclusion dependencies is one such scenario.

There are studies on the ranking of XML elements in the XML search context. In general, XML search needs to take into consideration the hierarchical structure of XML elements in the ranking, because XML elements of any granularity are potential answers to a query [14]. Our ranking scheme is unique in that the ranking is for pairs of HTML/XML elements. However, the idea of the cover in our context can be considered as a generalization of the removal of overlapping answers in XML search [5].

3 Preliminaries and the Problem

3.1 Inclusion Dependencies among Web Page Elements

This paper deals with inclusion dependencies among Web page elements. We model the target Web data as a triple $(P, elem, words)$. Here, P $(= \{p_1, p_2, \ldots\})$ denotes a set of Web pages, and $elem$ and $words$ are functions to represent components of each Web page; $elem(p_k)$ $(= \{e_1, e_2, \ldots\})$ defines the set of *page elements* contained in Web page p_k, and $words(e_j)$ $(= \{|w_1, w_2, \ldots|\})$ defines the multiset $words(e_j)$ of words contained in the element e_j. We need one constraint to represent the hierarchical structure of page elements; If e_i is a sub-element of e_j, $words(e_i)$ has to be a subset of $words(e_j)$. For example, assume that $elem(p_k)$ represents a set of HTML elements in Web page p_k and let $words(e_j)$ be a multiset of words in each element $e_j \in elem(p_k)$. Then, the mapping satisfies the constraint. Note that as long as the constraint is satisfied, the following discussion is independent of the mapping. For example, each p_k does not necessarily have to be an actual Web page; it can be an XML document created from an HTML page by a wrapping process.

Now, we define an inclusion dependency among page elements as follows: Let e_i and e_j be page elements. Then, $e_i \subseteq_{ind} e_j$ is an inclusion dependency between e_i and e_j meaning that $words(e_i) \subseteq words(e_j)$ should always be satisfied on the Web. In the rest of the paper, we often use e_i to denote $words(e_i)$ in set operations, when the meaning is clear from the context.

3.2 Jaccard Containment and Weak Inclusions

To find inclusion dependencies, we need to find *inclusions*, i.e., pairs of page elements that have the inclusion relationship in the current data instance. However, automatic discovery of such inclusions poses a problem, especially in the Web context. In general, Web content is *error-prone* and has *variations in expression*. Therefore, if we search for *exact* inclusions, we would miss many inclusion dependencies.

To deal with such a situation, we employ an approach based on Jaccard containment. Jaccard containment [1] is an asymmetric version of the Jaccard coefficient, which is a measure of set similarities. Given two sets s_1 and s_2, the Jaccard containment of s_1 in s_2, denoted by $JaccCont(s_1, s_2)$, is defined as follows.

$$JaccCont(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1|} \tag{1}$$

For example, Jaccard containment for $s_1 = \{\texttt{apple}, \texttt{peach}\}$ and $s_2 = \{\texttt{apple}, \texttt{banana}, \texttt{grape}\}$ is 0.5. If s_1 and s_2 are bags, the bag intersection is used to compute their Jaccard containment.

Let e_i and e_j be page elements and c be a value s.t. $0 \leq c \leq 1$. Then, we define $e_i \subseteq_c e_j$ as follows.

$$e_i \subseteq_c e_j \text{ iff } JaccCont(words(e_i), words(e_j)) = c \tag{2}$$

We read $e_i \subseteq_c e_j$ as "e_i is included in e_j with the inclusion ratio c," and call the pair (e_i, e_j), s.t. $e_i \subseteq_c e_j$, *an inclusion*. Note that when $c = 1$, it equals to $e_i \subseteq e_j$. In particular, an inclusion with a ratio $c < 1$ is called *a weak inclusion* when we need the distinction.

We define $e_i \subseteq_{\geq c} e_j$ as a natural extension; the pair (e_i, e_j) is an inclusion s.t. $e_i \subseteq_{\geq c} e_j$, if $JaccCont(words(e_i), words(e_j))$ in Equation (2) is greater than or equals to c. Inclusions with ratios for other inequalities are defined in a similar manner.

3.3 The Problem

Our problem is to first enumerate every inclusion (e_i, e_j) s.t. $e_i \subseteq_{\geq c} e_j$ for a given c, and then sort the enumerated inclusions for verification to find the actual inclusion dependencies among page elements on the Web.

Formally, let a set E of all page elements be $\bigcup_{p_k \in P} elem(p_k)$, $pairs = \{(e_i, e_j) | e_i, e_j \in E\}$, and c be a value s.t. $0 \leq c \leq 1$. Then, the first step is to compute and output a set of inclusions, denoted by $inclusions(pairs, c)$, where $inclusions(pairs, c) = \{(e_i, e_j) | (e_i, e_j) \in pairs \wedge e_i \subseteq_{\geq c} e_j\}$. In the first step, because the size of $pairs$ can be large ($O(|E|^2)$) and checking if a pair (e_i, e_j) is an inclusion s.t. $e_i \subseteq_{\geq_c} e_j$ requires costly strict comparisons, we want to filter out irrelevant pairs first and then conduct strict comparisons. The first step will be addressed in Section 4.

Although the first step enumerates all inclusions, they are candidates for inclusion dependencies; each of them does not necessarily imply the existence of inclusion dependency. Therefore, the second step is to sort the inclusions in $inclusions(pairs, c)$ for verification to find the actual inclusion dependencies. However, ordering the inclusions enumerated in the first step to support the efficient verification and finding efficient algorithms for the sorting are challenges. This will be addressed in Section 5.

The problem setting reflects how inclusion dependencies appear on the Web in practice. We model each element as a multiset of words and find inclusion dependencies among them, taking into consideration the fact that the Web content is error-prone and has variations in expressions. The model can exploit the hierarchical structure of HTML/XML elements but are tolerant to minor structural differences of elements. As shown in the examples in Section 1, it is common on the Web that the same data appears in different elements with similar hierarchical structures but the elements are different in their tags inside. Note that we do not deal with the case where two elements are semantically equivalent but much different in their structure or expressions. Dealing with such cases requires a different (and much complex) approach and is out of the scope of the paper. However, we believe that our approach can deal with many practical cases.

4 Filtering with bit-based signatures

In the first step, for the efficient computation of inclusions we first use a *filter* that removes the pairs of page elements that are guaranteed not to be inclusions. Then, we apply strict comparisons to the set of pairs that survived the filter for finding all the inclusions

4.1 Filters

Let $filter(e_i, e_j, c)$ be a predicate that returns false only when $e_i \subseteq_{\geq c} e_j$ is guaranteed not to hold. Assume that we compute the following set of pairs:

$$pairs' = \{(e_i, e_j) | (e_i, e_j) \in pairs \land filter(e_i, e_j, c)\}. \tag{3}$$

Then, $filter(e_i, e_j, c)$ should be designed to lead to the following results: (1) $|pairs'| \leq |pairs|$ and (2) $inclusions(pairs, c) = inclusions(pairs', c)$. This means that for a given c, the results with $pairs$ and $pairs'$ are identical.

4.2 Composing Bit-based Signatures

The idea of the bit-based signature scheme is as follows: First, we compute a *bit-based signature* $b(e)$ for each $e \in E$. Next, given $b(e_i)$, $b(e_j)$, and c, we perform a simple computation to decide whether $filter(e_i, e_j, c)$ holds.

The question is whether it is possible to develop a scheme to evaluate $e_i \subseteq_{>c} e_j$ for any given c. This section shows that there exists such a signature scheme. Let $sigsize$ be the (fixed) size of the signature. Given $e_i \in E$ and an integer $t > 0$, let $b(e_i)$ be the bit sequence computed in the following manner.

1. For each word $w \in words(e_i)$, compute the hash value $h(w) \in [0, sigsize - 1]$.
2. If the number of words having the same $h(w)$ is greater than t, set the $h(w)$-th bit of $b(e_i)$ to 1, otherwise set it to 0.

Here, t should be chosen such that the distribution of signatures is not skewed. Note that because the set of page elements constitute a tree structure, we have a large number of small elements and a smaller number of large elements. In other words, the distribution of $|words(e_i)|$ for $e_i \in E$ is biased to smaller values. Assuming that the distribution of word occurrences is uniform, it is reasonable to define t so that 1s and 0s are uniformly distributed in the signatures of small elements. Consequently, we define t as follows. Let $minsize$ be the smallest size of $words(e_i)$ for $e_i \in E$, i.e.,

$$minsize = \begin{matrix} min \\ e_i \in E \end{matrix} \Big|words(e_i)\Big|. \tag{4}$$

Then,

$$t = \begin{cases} 1 & (minsize \leq sigsize) \\ \lfloor \frac{minsize}{sigsize} \rfloor & otherwise \end{cases} \tag{5}$$

Note that if we define t on the basis of the larger $words(e_i)$, many bits of the signatures of small elements would be 0, which means that the distribution of bits is more skewed in many signatures.

Example. Assume that we have the following set E of page elements.

$$E = \{e_1, e_2, e_3\}$$
$$words(e_1) = \{|a1, a2, b1, c1, d1|\}$$
$$words(e_2) = \{|a1, a2, b1, b2, c1, d1, d2, e1, e2, f1|\}$$
$$words(e_3) = \{|b1, b2, b3, b4, d1, d2, e1, f1, g1, h1|\}$$

Let $sigsize = 8$ and $h(w)$ be the code of the first character of w modulo eight. Then, $b(e_1)$, $b(e_2)$, and $b(e_3)$ are computed as follows.

1. Let $H(e_i)$ be the multiset of hash values for $words(e_i)$, i.e., $H(e_i) = \{|h(w)|w \in words(e_i)|\}$.
$$H(e_1) = \{|0, 0, 1, 2, 3|\}$$
$$H(e_2) = \{|0, 0, 1, 1, 2, 3, 3, 4, 4, 5|\}$$
$$H(e_3) = \{|1, 1, 1, 1, 3, 3, 4, 5, 6, 7|\}$$
2. Because $minsize = |words(e_1)| = 5 \leq sigsize, t = 1$. Therefore, we set the $h(w)$-th bit of $b(e_i)$ to 1 if there is at least one $h(w)$ in $H(e_i)$. We get
$$b(e_1) = 00001111$$
$$b(e_2) = 00111111$$
$$b(e_3) = 11111010$$

4.3 Filter Evaluation

Here, we define $b\text{-}filter(e_i, e_j, c)$, which is a kind of $filter(e_i, e_j, c)$ that uses $b(e_i)$, $b(e_j)$, and c to remove irrelevant page element pairs.

The idea is to compute the possible maximum inclusion ratio $\mathtt{c_{max}}$ for $e_i \subseteq_c e_j$ on the basis of $b(e_i)$ and $b(e_j)$. Then, we can define $b\text{-}filter(e_i, e_j, c) \equiv true$ if $\mathtt{c_{max}} \geq c$. For this purpose, we introduce a theorem on $b(e_i)$ and $b(e_j)$.

Theorem 1 *Let e_i and e_j be page elements. Then, the possible maximum inclusion ratio $\mathtt{c_{max}}$ for $e_i \subseteq_c e_j$ is computed as follows.*

$$\mathtt{c_{max}} = \frac{|words(e_i)| - X}{|words(e_i)|} \tag{6}$$

Here, X is the number of integers k s.t. the k-th bit of $b(e_i)$ is 1 and the k-th bit of $b(e_j)$ is 0. □

Proof. Let k be an integer s.t. the k-th bit of $b(e_i)$ is 1 and that of $b(e_j)$ is 0. By definition, X denotes the number of such bits. Although the k-th bit of $b(e_j)$ is 0, it could have been 1 if there was one more occurrence of word w with $h(w) = k$ in $words(e_j)$ that reaches the threshold t. In other words, it is guaranteed that there is at least one word that is included in $words(e_i)$ and not in $words(e_j)$. Therefore, given $b(e_i)$ and $b(e_j)$, c is maximum for $e_i \subseteq_c e_j$ when for every bit of X such bits we need only one word to set the bit to 1. Equation (3) computes c for the case. □

Given Theorem 1, we define $b\text{-}filter(e_i, e_j, c)$, that produces no false negatives, as follows.

$$b\text{-}filter(e_i, e_j, c) = \begin{cases} true & \mathtt{c_{max}} \geq c \\ false & otherwise \end{cases} \tag{7}$$

Example. Assume that we have the page elements shown in Section 4.2. Then, $b\text{-}filter(e_1, e_2, 0.7)$ is evaluated as follows. First, $\mathtt{c_{max}} = \frac{5-0}{5} = 1$. Because $\mathtt{c_{max}} \geq 0.7$, it is possible that the inclusion ratio exceeds 0.7. Therefore, $b\text{-}filter(e_1, e_2, 0.7)$ is true. As another example, $b\text{-}filter(e_1, e_3, 0.7)$ is false, because $\mathtt{c_{max}} = \frac{5-2}{5} = 0.6 < 0.7$. Therefore, the element pair (e_1, e_3) does not survive the filter.

4.4 Strict Comparison

For each (e_i, e_j) that survived the filter, the first step conducts a strict comparison to compute Jaccard containment and determine whether it is an inclusion. To the best of our knowledge, the complexity of the fastest algorithm to check if a pair (e_i, e_j) is an inclusion is in $O(n)$ for the size of the sets [2] under the assumption that we sort the words in e_i and e_j before the calculation.

4.5 Computational Complexities

Let $m = |E|$. Then, $|pairs|$ is ${}_mC_2$ (i.e., $O(m^2)$). Let n be the average of $|words(e)|$ for all $e \in E$. The simple computation needs to (1) sort the words included in every element in E, whose computational complexity is $O(mn \log n)$, and (2) conduct strict comparisons to find inclusions ($O(m^2 n)$).

If we use $b\text{-}filter$, we need an additional computation of the bit signatures for all page elements, whose computational complexity is $O(n)$ for each signature. However, the additional computation reduces other costs. We do not need to sort the words eliminated by $b\text{-}filter$. The cost of the filtering is cheap (its computational complexity is $O(1)$).

Let $m_{\mathtt{b}} (<< m)$ be the number of pairs that are not eliminated by $b\text{-}filter$. Let $n_{\mathtt{b}}$ be the average of $|words(e)|$ for all elements of the survived pairs. Then, the computation needs to (1) create the signatures for every $e \in E$ ($O(nm)$), (2) apply $b\text{-}filter$ to all pairs ($O(m^2)$), (3) sort $words(e_i)$ and $words(e_j)$ of (e_i, e_j) that survived $b\text{-}filter$ ($O(m_{\mathtt{b}} n_{\mathtt{b}} \log n_{\mathtt{b}})$), and (4) conduct strict comparisons to find inclusions ($O(m_{\mathtt{b}}^2 n_{\mathtt{b}})$). Because $m_b << m$ as shown in Section 6, $b\text{-}filter$ dramatically reduces the cost of inclusion computation compared with the simple computation without the filter, although $n_{\mathtt{b}}$ is often larger than n.

5 Ranking Inclusions

Because all inclusions enumerated for a data instance do not necessarily result in actual inclusion dependencies, the next step is to verify whether each inclusion implies an inclusion dependency. The method to rank the enumerated inclusions is important because typically, the verification is performed manually. This section discusses the method to rank the element pairs in $inclusions(pair, c)$ (Section 3.3), which represent the enumerated inclusions. For the discussion, we ignore the inclusion ratio in each $\subseteq_{\geq c}$, since the purpose of introducing the ratio is to consider the problem as an approximation of finding exact inclusions. Now, the problem can be modeled as how to rank the inclusions in $inclusions(pairs) = \{(e_i, e_j) | (e_i, e_j) \in pairs, e_i \subseteq e_j\}$.

Let α and β be inclusions s.t. $\alpha, \beta \in inclusions(pairs)$. We say that α *covers* β, if we can verify β in parallel with the verification of α. We write $\alpha \geq \beta$ to denote that α covers β.

The notion of covers can be considered as a generalization of the removal of overlapping answers in XML search. In XML search, it is often the case that the element hierarchy of XML data allows us to see an answer e_1 in parallel with seeing e_2, because some of the elements (e.g. e_1) satisfying a query condition are often included in the others (e_2). We extend the idea to deal with the relationships among element pairs in the context of the verification of inclusion dependencies. We define two types of cover relationships, namely, *deductive covers* that represent logical overlaps and *regional covers* that represent physical overlaps. Note, our purpose is to verify whether each inclusion represents an inclusion dependency, not whether each element pair is an inclusion. We also note that the two relationships are not intended to constitute a complete set of cover relationships, although we have not noticed any other types of covers.

In the following, we first define two types of cover relationships. Cover relationships define the semiorder among inclusions (element pairs). However, how to efficiently compute the relationships is not straightforward. Interestingly, there exists an efficient algorithm based on the probabilities of word occurrences that produces the total order among inclusions compatible with both cover relationships (i.e., the output is one of its topological sorts). We present the algorithm. Finally, we show that re-ordering of the outputs of the probability-based algorithm gives better ranking results.

5.1 Cover Relationships among Inclusions

The cover relationship ($\geq$) consists of *deductively cover* relationship ($\geq_d$) and *regionally cover* relationship ($\geq_r$). Formally, $\alpha \geq \beta$ iff $\alpha \geq_d \beta \vee \alpha \geq_r \beta$.

Definition 1 *When we have two inclusions $\alpha = (e_1, e_4)$ and $\beta = (e_2, e_3)$, we say α deductively covers β (denoted by $\alpha \geq_d \beta$) if and only if the following conditions hold.*

- *e_2 is a descendent element of e_1 in the element hierarchy of a page (i.e., $words(e_2) \subseteq words(e_1)$), and*
- *e_4 is a descendent element of e_3 in the element hierarchy of a page (i.e., $words(e_4) \subseteq words(e_3)$).* □

This is illustrated by Figure 2, in which two element hierarchies are shown: (1) e_1 and e_2 and (2) e_3 and e_4. Then, $e_1 \subseteq e_4$ deductively covers $e_2 \subseteq e_3$ because the latter is deduced from the former and the inclusions $e_2 \subseteq e_1$ and $e_4 \subseteq e_3$, which are derived from the hierarchical structure.

When α deductively covers β, we can easily check whether β suggests an inclusion dependency in parallel with checking whether α does. This is because (1) the elements in β overlap those in α, and (2) the existence of inclusion $\alpha = (e_1, e_4)$ suggests the place of inclusion $\beta = (e_2, e_3)$. Namely, (a) e_2 exists inside e_1, and (b) e_3 can be every ancestor of e_4. The existence of β, however, does not imply that of α. Therefore, the user who was first told that β exists, could not know where and even whether related inclusions exist.

Definition 2 *When we have two inclusions $\alpha = (e_1, e_3)$ and $\beta = (e_2, e_4)$, we say α regionally covers β (denoted by $\alpha \geq_r \beta$) if and only if the following conditions hold.*

- *e_2 is a proper descendent element of e_1 in the element hierarchy of a page, and*
- *e_4 is a proper descendent element of e_3 in the element hierarchy of a page.* □

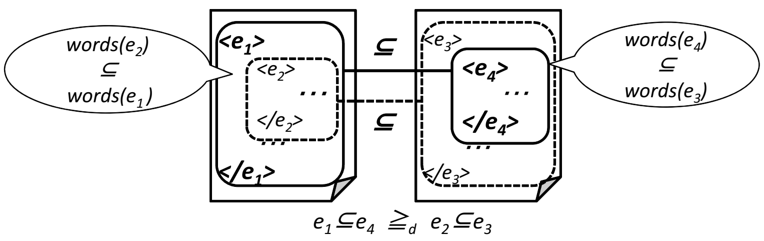


Figure 2: Example of $\geq_d$

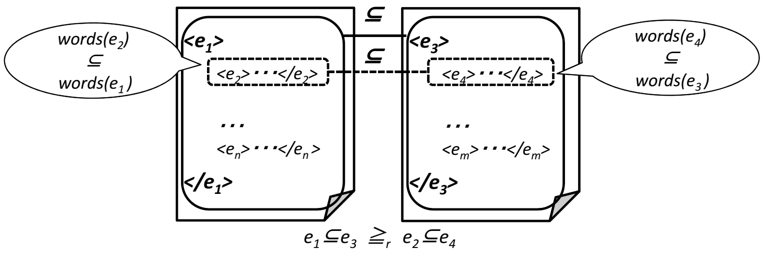


Figure 3: Example of $\geq_r$

Figure 3 shows an example in which there are two element hierarchies: (1) e_1 and e_2 and (2) e_3 and e_4. Then, $e_1 \subseteq e_3$ regionally covers $e_2 \subseteq e_4$, because e_2 is a descendent element of e_1 and e_4 is a descendent element of e_3.

A typical scenario encountered is the situation where there are two pages maintaining two lists of items and one list is a sublist of the other; for example, a publication list of a lab and one of its members. In the example, e_1 and e_3 are lists of publications, and e_2 and e_4 represent publications.

When α regionally covers β, we can easily check whether β suggests an inclusion dependency in parallel with checking whether α does, for the same reason as the deductive covers: (1) the elements in β are overlapped to those in α, and (2) the existence of inclusion $\alpha = (e_1, e_3)$ suggests the place of inclusion $\beta = (e_2, e_4)$. Namely, (a) e_2 exists inside e_1, and (b) e_4 exists in e_3.

5.2 Probabilities of Inclusions

A question arises as to whether there exist algorithms to efficiently compute the deductive and regional covers among inclusions. Our finding is that there exists such an algorithm to compute them.

As proved later (Lemmas 2 and 3), we found that the notion of deductive and regional covers is compatible with the probabilities of occurrences of inclusions. Formally, let α (β) be an inclusion and $P(\alpha)$ ($P(\beta)$) be the probability that the inclusion appears in page elements. Then, we prove that $P(\alpha) \leq P(\beta)$ if $\alpha \geq \beta$. Because of the antisymmetric nature of the order, this implies that $\alpha \geq \beta$ if $P(\alpha) \leq P(\beta)$ when $\alpha \geq \beta$. Therefore, when we sort the inclusions according to their probabilities, the result becomes a topological sort of inclusions with the cover relationship.

We compute the probabilities on the basis of a simplified model. In Section 6, we show that the model works well even in the real Web setting. Let $WORDS$ denote the set of all words that can appear in Web pages. We assume that each word independently occurs in page elements and the distribution of word occurrence is uniform. In addition, we show that the size of $WORDS$ is sufficiently large s.t. $|WORDS| >> |words(e)|$ for every $e \in E$.

Then, given elements e_1 and e_2 (s.t. $|words(e_1)| \leq |words(e_2)|$), the probability that $e_1 \subseteq e_2$ appears, denoted by $P(e_1 \subseteq e_2)$, is computed as follows.

$$P(e_1 \subseteq e_2) \quad = \quad \frac{{}_{|words(e_2)|}C_{|words(e_1)|}}{{}_{|WORDS|}C_{|words(e_1)|}} \tag{8}$$

The expression computes the probability that $words(e_1)$ is a subset of $words(e_2)$ in the simplified model. Note that the definition is very simple and does not require complex computations.

Algorithm FindGroups(dataSeq)

```
Input: dataSeq = [$pair_1$, $pair_2$, ...]
Output: dataSeq with group identifiers
 1 let firstElem = NULL;
 2 let currentGroupID = 0;
 3 for each pair∈dataSeq{
 4 if (firstElem == NULL || firstElem.postOrder < pair.leftElem.postOrder){
 5         currentGroupID = currentGroupID + 1;
 6         firstElem = pair.leftElem;
 7 }
 8 pair.groupID = currentGroupID;
 9 }
10 return dataSeq;
```

Figure 4: Algorithm to find inclusion groups.

5.3 Probabilities and the Cover Relationships

The following theorem holds.

Theorem 2 *For any two inclusions α and β, $P(\alpha) \le P(\beta)$ if $\alpha \ge \beta$.* □

We prove the theorem by showing three lemmas and present some of their proofs.

Lemma 1 *Let two inclusions α and β be (e_1, e_2) and (e_3, e_4), respectively, Then, $P(\alpha) < P(\beta)$ if (a) $|words(e_1)| > |words(e_3)|$, or (b) $|words(e_1)| = |words(e_3)|$ and $|words(e_2)| < |words(e_4)|$.* □

Crux. The only subtle case is when $|words(e_1)| > |words(e_3)|$ and $|words(e_2)| > |words(e_4)|$. Assuming that $|WORDS| > |words(e_2)|^N$ where $N = |\frac{words(e_1)}{words(e_1)-words(e_3)}|$, we can derive $P(\alpha) < P(\beta)$ even in this case. □

The complete proof of Lemma 1 is found in [20].

Lemma 2 *For any two inclusions α and β, $P(\alpha) \le P(\beta)$ if $\alpha \ge_d \beta$.* □

Proof. Let two inclusions α and β be (e_1, e_2) and (e_3, e_4), respectively. By the definition of $\ge_d$, $e_3 \subseteq e_1$ and $e_2 \subseteq e_4$ because $e_1 \subseteq e_2 \ge_d e_3 \subseteq e_4$. Therefore, $|words(e_3)| \le |words(e_1)|$ because $e_3 \subseteq e_1$. Similarly, $|words(e_2)| \le |words(e_4)|$ because $e_2 \subseteq e_4$.

There are three cases: (1) If $|words(e_3)| < |words(e_1)|$, condition (a) of Lemma 1 guarantees that $P(\alpha) < P(\beta)$ holds. (2) If $|words(e_1)| = |words(e_3)|$ and $|words(e_2)| < |words(e_4)|$, condition (b) of Lemma 1 guarantees that $P(\alpha) < P(\beta)$ holds. (3) If $|words(e_1)| = |words(e_3)|$ and $|words(e_2)| = |words(e_4)|$, it is obvious that $P(\alpha) = P(\beta)$ and thus $P(\alpha) \le P(\beta)$. □

Lemma 3 *For any two inclusions α and β, $P(\alpha) \le P(\beta)$ if $\alpha \ge_r \beta$.* □

Proof. Let α and β be (e_1, e_2) and (e_3, e_4), respectively. By the definition of $\ge_r$, $e_3 \subset e_1$ and $e_4 \subset e_2$ because $e_1 \subseteq e_2 >_r e_3 \subseteq e_4$. Therefore, $|words(e_3)| < |words(e_1)|$ because $e_3 \subset e_1$. Similarly, $|words(e_4)| < |words(e_2)|$ because $e_4 \subset e_2$.

Since $|words(e_3)| < |words(e_1)|$, condition (a) of Lemma 1 guarantees that $P(\alpha) < P(\beta)$ and thus $P(\alpha) \le P(\beta)$. □

5.4 Algorithms for Ranking

In the ranking result of inclusions we want the top-ranked inclusions to cover as many other inclusions as possible. A simple approach is to sort the inclusions in the ascending order of their probabilities because the sorting result is guaranteed to be a topological sort of inclusions with the cover relationship $\ge$.

Interestingly, the simple approach does not necessarily yield a good ranking and other possibilities exist. For instance, assume that we have four inclusions α, β, γ and δ s.t. $\alpha \ge \beta$ and $\gamma \ge \delta$.

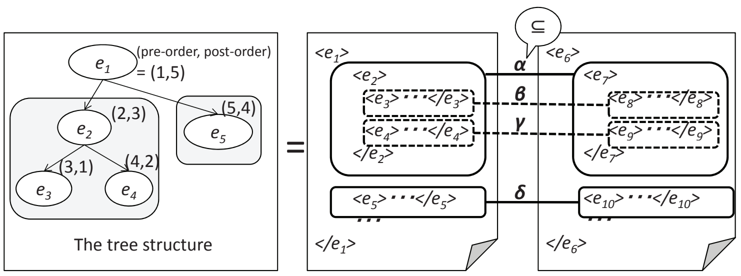


Figure 5: Example of the group-conscious algorithm.

Obviously, the good rankings would have α and γ in the first two inclusions, and β and δ in the last two inclusions, since we need to verify only the first two inclusions that cover the remaining inclusions. However, the simple ranking does not necessarily yield such a ranking result, because it is possible that we have such probabilities that $P(\alpha) < P(\beta) < P(\gamma) < P(\delta)$. In that case, we need to examine the first three inclusions to cover all the inclusions.

Group-Conscious Algorithm. In the above example, a better ranking result has α and γ in the first two inclusions and β and δ in the last two inclusions, because we need to verify only the first two inclusions that cover the remaining inclusions. The group-conscious algorithm generalizes the idea. First, we define a group of inclusions as follows:

Definition 3 *A group of inclusions on $\geq$ is a set of inclusions, for any two inclusions α and β of which, either of $\alpha \geq \beta$ or $\beta \geq \alpha$ holds.* □

The group-conscious algorithm first divides inclusions into groups, then sorts inclusions in each group by the probabilities and finally merges the sorted results on the basis of the rank in each group. In the final step, the inclusions with the same rank in their groups are ordered according to their probabilities. Note that the algorithm outputs a sequence [α, γ, β, δ] for the above example, in which the first two inclusions cover the remainder.

Theorem 3 *The group-conscious algorithm sorts inclusions in a topological order with the cover relationship.* □

Proof. By the definition of a group, inclusions from different groups do not have the cover relationships to each other. The inclusions in each group are sorted to form a topological sort, and the final re-ordering preserves the ordering in each group. □

It is not easy to find an efficient algorithm to group inclusions. We developed an efficient group-conscious algorithm, that exploits the following theorem.

Theorem 4 *For any inclusions $\alpha = (e_1, e_2)$ and $\beta = (e_3, e_4)$, e_1 is an ancestor of e_3 if $\alpha \geq \beta$.* □

Proof. There are two cases. (1) $\alpha \geq_d \beta$: By the definition of $\geq_d$, $e_3 \subseteq e_1$ when $e_1 \subseteq e_2 \geq_d e_3 \subseteq e_4$. (2) $\alpha \geq_r \beta$: By the definition of $\geq_r$, $e_3 \subset e_1$ when $e_1 \subseteq e_2 \geq_r e_3 \subseteq e_4$. □

Figure 4 shows the efficient algorithm to identify inclusion groups. For simplicity, the algorithm deals with inclusions related to only one Web page, i.e., we assume that we have one element tree. It is easy to extend it to the case where we have more than one element tree.

The prerequisite of the algorithm is that the inclusions (element pairs) are sorted by the depth-first order of the left element (e_i of (e_i, e_j)) in the element tree. Note that we do not need explicit sorting if the algorithm to produce element pairs traverses the element tree in the depth-first order.

Interestingly, Theorem 4 guarantees that the sequence of inclusions that are sorted in the order already clusters the inclusion groups. In addition, it is guaranteed that the left element e_i of the first inclusion (e_i, e_j) in each group (in the sequence) is an ancestor of every left element of the inclusions in the same group.

The algorithm in Figure 4 exploits the property and scans the sequence of inclusions from the beginning, assigning an incremental group identifier to each group. In the scan, the algorithm maintains (1) the group identifier kept in the variable $currentGroupID$, and (2) the left element of the first inclusion of each group kept in the variable $firstElem$. The latter is used to determine when the scan enters the next group. It is easy to determine when the scan enters the next group

Set	WebSite	# WebPages	$\|pairs_i\|$
Set 1	The School of Informatics (inf)	42	4,975,217,840
	College of Knowledge and Library Sciences (klis)	131	
	College of Media Arts, Science and Technology (mast)	156	
	College of Information Science (coins)	672	
Set 2	SIGMOD	2994	29,440,755,931
	SIGIR	72	
	SIGWEB	644	
Set 3	Tokyo Electric Power Company /nu (tepco)	956	7,948,539,479
	Nuclear Safety Commission of Japan (nsc)	561	
	Nuclear and Industrial Safety Agency /genshiryoku (nisa)	212	

Table 1: Datasets.

if we use the post-order assigned to each element. When each group ends in the scan, it updates $currentGroupID$ and $firstElem$ for the next group.

For example, assume that we have the four inclusions $\alpha = (e_2, e_7)$, $\beta = (e_3, e_8)$, $\gamma = (e_4, e_9)$ and $\delta = (e_5, e_{10})$ shown in Figure 5 (right). Each element has a pair of pre and post-order numbers (Figure 5 (left)). By the definition of the cover relationships, $\alpha \geq_r \beta$ and $\alpha \geq_r \gamma$. Then, we can find the groups for the four inclusions as follows:

1. Assume that the inclusions are sorted by the depth-first order (pre-order) of the left element of each inclusion.
$$\alpha : e_2 \subseteq e_7$$
$$\beta : e_3 \subseteq e_8$$
$$\gamma : e_4 \subseteq e_9$$
$$\delta : e_5 \subseteq e_{10}$$
2. At first, $currentGroupID = 0$ and $firstElem = Null$ (Lines 1, 2), and in the first iteration of the loop, they are updated so that $currentGroupID = 1$ and $firstElem = e_2$.
3. Until the iteration reaches δ, $currentGroupNo$ remains 1, and therefore, we find that α, β, and γ are in group 1.
4. When δ is reached, the post-order of node e_5 is 4 and we again update $currentGroupNo$ and $firstElem$ to enter the next group.
5. Finally, we find that α, β, γ are in the group 1 and δ is in the group 2.

5.5 Computational Complexities

The group-conscious algorithm is as efficient as the simple algorithm. Let n be the length of the inclusion sequence. The simple algorithm first computes the probabilities of inclusions whose computational complexity is $O(n)$ and then sorts the inclusions according to the probabilities. Overall, the computational complexity is $O(n \log n)$.

Although the group-conscious algorithm needs to identify inclusion groups, the scan can be performed in parallel with the computation of probabilities ($O(n)$). Next, it sorts inclusions in each group and finally merges the results of all sorts. Therefore, the computational complexity is again $O(n \log n)$.

6 Evaluation

We evaluated the effects of the proposed filter and ranking schemes with various real Web sites. All experiments were performed with a PC with a Core 2 DUO 2.13 GHz CPU and 16 GB RAM. The codes were written in Java 1.6.0.

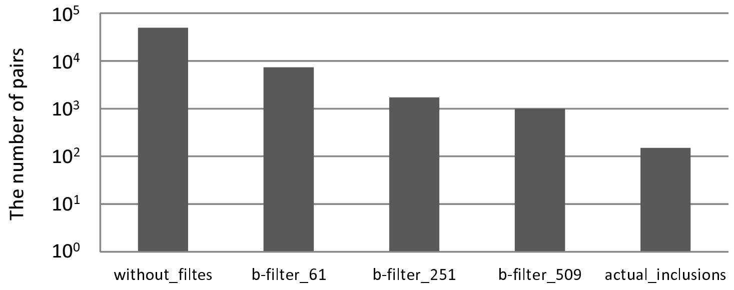


Figure 6: Experiment 1: Effects of the signature size ($pairs_1$).

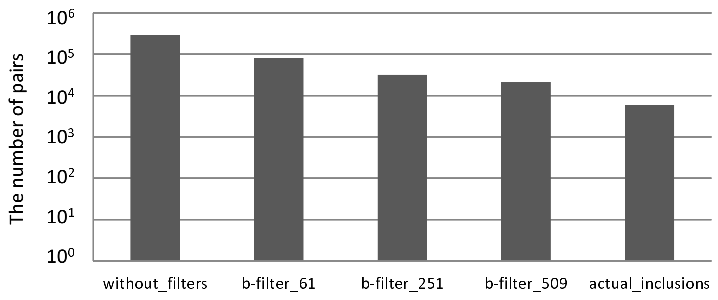


Figure 7: Experiment 1: Effects of the signature size ($pairs_2$).

6.1 Datasets

We created three sets of data for the experiments involving 11 Web sites. There are many situations in which the contents of Web sites are closely related to each other but are managed by different organizations. We constructed three sets to include Web sites from academia, industry, and governmental organizations.

Set 1 consists of four web sites from a university. One is the Web site of the School of Informatics (inf) [22] of the university, and the other threes (mast [24], klis [23], coins [25]) are those of its departments.

Set 2 consists of three Web sites from ACM special interest groups: the web sites of SIGMOD [26], SIGWEB [28], and SIGIR [27].

Set 3 consists of the Web sites of three organizations related to the nuclear power plant accident in Japan. They include Tokyo Electric Power Company (tepco) [29], Nuclear Safety Commission of Japan (nsc) [30] and Nuclear and Industrial Safety Agency (nisa) [31]. The latter two are governmental organizations that administer nuclear policies and power plants in Japan.

The data for the experiments were constructed as follows. First, We crawled Web sites from their root pages except for Set 3. Since most of contents of tepco and nisa are not related to nuclear power plants, we crawled them from the root pages of their nuclear plant directories.

After the harvest of Web pages, we extracted three types of words from page elements, namely, natural language words (`words`), 2-grams (`2-grams`), and 3-grams (`3-grams`). For Web pages written in Japanese, we used Sen [32], which is a tool for the morphological analysis to extract natural language words. Finally, we constructed $pairs_i$ which is a set of element pairs, for each Set i. We did not pair up elements in the same Web page, so that each set contains no trivial inclusion, i.e., any pair of elements one of which is a descendant of the other. Table 1 summaries the numbers of Web pages and the size of $pairs_i$.

6.2 Preliminary Experiments

First, we conducted a preliminary experiment to determine the appropriate threshold for the weak inclusion relationship $\subseteq_{\geq c}$. We randomly selected 20 pairs from the site `inf` of Set 1; it is known that 10 of the selected pairs actually have inclusion relationships and the others do not. Our finding is that there is a clear difference between the inclusion ratio of the set of inclusions and that of the

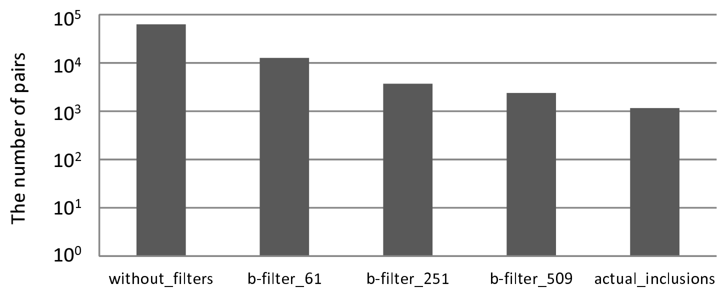


Figure 8: Experiment 1: Effects of the signature size ($pairs_3$).

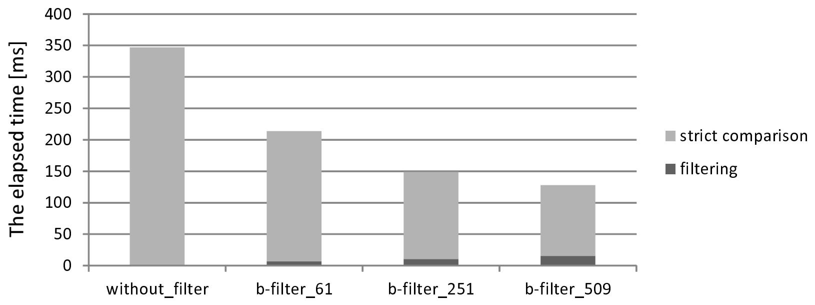


Figure 9: Experiment 1: Elapsed time ($pairs_1$).

set of pairs that are not inclusions. For every pair that we know has an inclusion relationship, the inclusion ratio is greater than 0.7, whereas for the others, it is smaller than 0.2. The fact that there is a huge gap agrees with our another result based on a different set of data [17], and confirms that the weak inclusions are appropriate for finding inclusions on the Web. Therefore, in the experiments, we use $c = 0.7$ as a threshold for the weak inclusion relationships (i.e., $\subseteq_{\geq 0.7}$).

Second, we conducted an experiment to check whether the hash function $h(w)$ used to construct the signatures in the experiments is appropriate. The hash function computes the reminder of the sum of $bytecode(c_i) \times 10^{i-1}$ divided by the size of signature, where c_i is the i-th byte character of the word. The variation coefficients of the size of equivalence classes of words by $h(w)$ with the data sets are from 4.4% (for 61-bit signatures) to 15.6% (for 509-bit signatures), which means that all equivalence classes are relatively similar to each other in size.

6.3 Filtering with the Bit-based Signatures

In summary, our results show that the bit-based signatures reduce the size of the set of possible pairs by orders of magnitude. We explored the influence of the signature size and the type of extracted words on the performance. In this paper, some of the results will be shown only for Set 1 if the remaining results are similar.

6.3.1 Experiment 1: Effects of the Signature Size

We verified the number of irrelevant pairs that can be removed from $pairs_i$ by $b\text{-}filter$. We set the inclusion ratio to 0.7 according to the result of our preliminary experiment. We used `words` as a base set of words and $sigsize$ for $b\text{-}filter$ was set to 61, 251 or 509. They were chosen since they are prime numbers close to 64, 256 and 512.

Result. Figures 6, 7, and 8 show the results with the three $pairs_i$'s. The horizontal axis shows the signature size with the exception that the left most bar represents the case without filters, and the right most bar represents the actual inclusions with inclusion ratios of greater than 0.7. The vertical axis shows the number of pairs (per left element) that survived each filter.

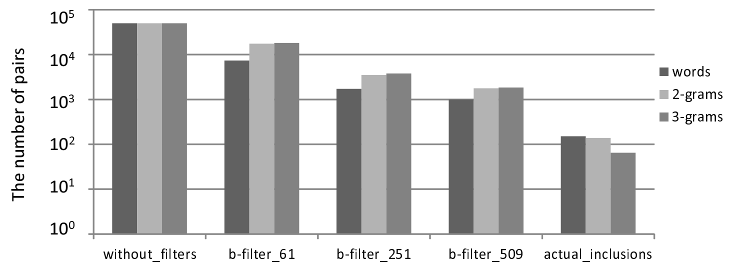


Figure 10: Experiment 2: Effects of the word type.

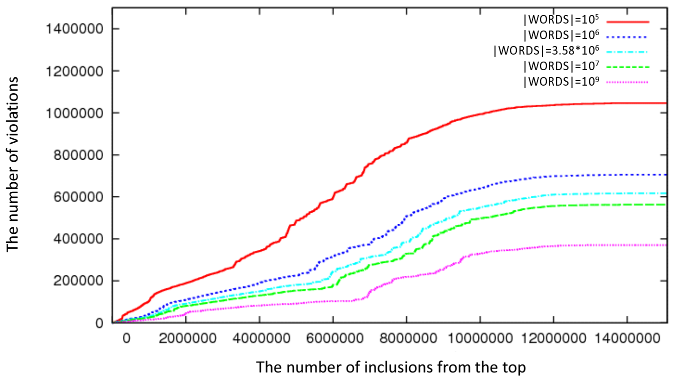


Figure 11: Experiment 3: Violations for $inclusions_1$

The result demonstrates two important facts. First, the bit-based signature can reduce the size of the set of possible pairs by an order of magnitude. Second, the larger the size of the signature, the better the result. For Set 1, the 509-bit signature reduced the set of possible pairs to about 1/50 in size. We obtained similar results for the other sets. Importantly, for all data sets, the number of inclusions is reduced to less than 10 times the number of the actual inclusions. Because the Web sites for the experiments are chosen to give many related elements, the data sets are not favorable to the reduction ratio. The ratio is expected to be larger in general cases.

Next, Figure 9 shows the elapsed time per element, i.e., the total time divided by the number of elements. The time shown is the average of the results of the five trials. In the experiment, the 509-bit signature reduced the elapsed time by 2.5 times compared with that without the filter. The results all agree with our theoretical results on computational complexities. The cost of the signature creation and filtering is cheap. The reduction ratio of strict comparisons by the filter is $\frac{m_\mathfrak{b}^2 n_\mathfrak{b}}{m^2 n}$, where $m_\mathfrak{b}^2 = 1/50m^2$ and $n_\mathfrak{b} = 20n$ in the result of Experiment 1.

Note that the performance of the filter is not the only factor to affect the elapsed time. In general, the higher the cost of strict evaluation of inclusions is, the better the filter reduces the elapsed time. Our experiment loaded the data into the disk before all the computation and the elapsed time does not include the disk access time. In practice, however, it is often the case that the data is too large to be stored on the memory and we can store only the signatures on the memory to avoid loading unnecessary data. In such scenarios, the signature-based filter is more effective.

6.3.2 Experiment 2: Effects of the Word Type

In the experiment, we compared the results of the filters using different sets of base words (`words`, `2-grams`, and `3-grams`), with Set 1 ($pairs_1$).

Result. Figure 10 shows the results of the experiments with `words`, `2-grams` and `3-grams`, respectively. The result using `words` is slightly better than the others, since the number of n`-grams` is larger than that of `words` and the average size of equivalence classes of words by $h(w)$ is larger. An important fact, however, is that the filter is effective with all word sets and the same discussions are applicable to all the three cases. In the following experiments, we use `words` as the set of words.

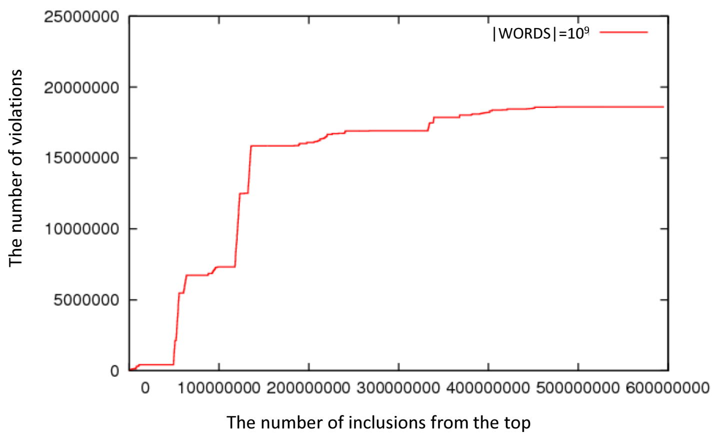


Figure 12: Experiment 3: Violations for $inclusions_2$

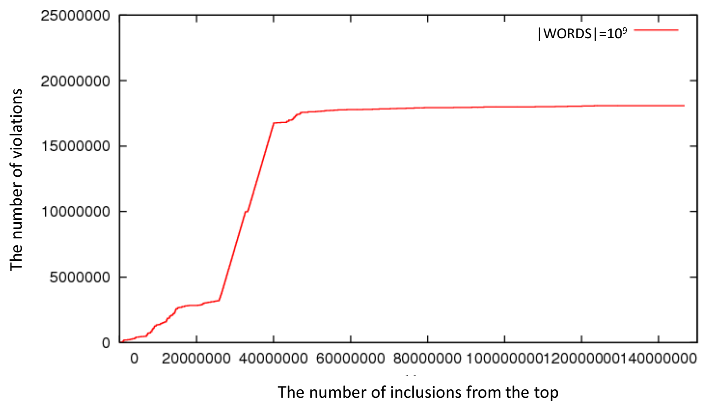


Figure 13: Experiment 3: Violations for $inclusions_3$.

6.4 Ranking Schemes

6.4.1 Experiment 3: Cover Relationships and Probabilities in Real Settings

Although we have proven that the probabilities of inclusion occurrences are compatible with the cover relationship, we assumed in the proof a simplified model for word occurrences. The experiment explores how the theoretical results can be applied to real data. We first constructed the set $inclusions_i = inclusions(pairs_i, 0.7)$ for each $pairs_i$.

Then, for every $inclusions_i$, we compared the probabilities in real Web data with the ideal results. For calculating probabilities of inclusions, we need parameter $|WORDS|$. In the experiment, we took the parameter from $\{100{,}000, 1{,}000{,}000, 3{,}580{,}000, 10{,}000{,}000, 1{,}000{,}000{,}000\}$, in which 3,580,000 is the number of words that actually appear in Set 1.

Result. For the first analysis, we counted the number of *violations*. We define a violation as the probability of an inclusion that is incompatible with the cover relationship. Formally, $P(\alpha)$ is a violation if there is an inclusion β s.t. $\beta \geq \alpha$ but $P(\beta) \geq P(\alpha)$.

In the experiment, there was no violation concerning $\geq_d$. This is because by the definition, the $\geq_d$ relationship is not related to the subtle case in the crux of Lemma 1. However, there were violations concerning $\geq_r$. Figure 11 shows the result with $inclusions_1$. The horizontal axis shows the number of inclusions from the top and the vertical axis shows the accumulated number of violations. As the figure shows, the rate of violations decreases when we increase the number of $|WORDS|$. The rate is only 2.5% when $|WORDS| = 10^9$. The reason is that our model assumes that $|WORDS| >> words(e_2), words(e_1)$, when we calculate $P(e_1 \subseteq e_2)$.

An interesting finding is that the result with $|WORDS| = 10^9$ is better than that with the actual number of $|WORDS|$ (3,580,000). The actual number is not consistent with our assumption in

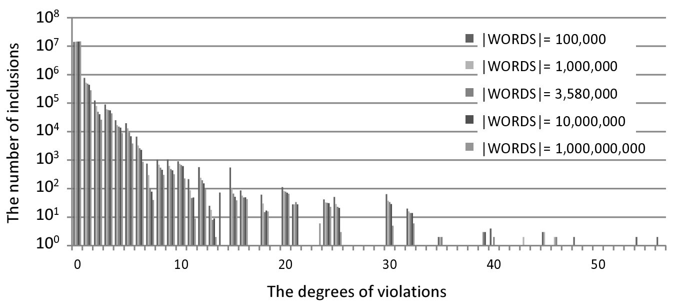


Figure 14: Experiment 3: Violation degrees for $inclusions_1$.

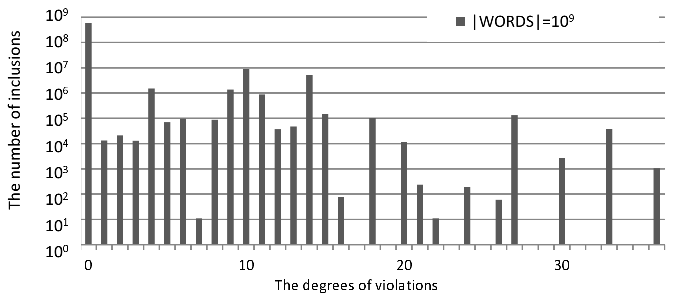


Figure 15: Experiment 3: Violation degrees for $inclusions_2$.

the proofs; because there are elements having more than 2000 different words, it is possible that $|WORDS| < words(e_2)^N$ for such elements (see Theorem 2). Therefore, the question is whether we should set the parameter according to the real data or our assumption. The result suggests that we do not have to stick to the real data when setting the parameter and should use a large number. Given the results, the remaining experiments were conducted with $|WORDS| = 10^9$ for the calculation of probabilities.

Similarly, the violation rates with $|WORDS| = 10^9$ were 3.1% and 12% for $incusions_2$ and $inclusions_3$, respectively. The rate for $inclusions_3$ was higher because the size of words in elements was larger. As shown next, however, the influence by the violations were limited.

In the second analysis, we examined the degree of difference owing to each violation. We define the *degree* of a violation as the number of inclusions that are placed at incorrect positions in the ranking because of the violation. For example, let $P(\alpha)$ be a violation. Then, if there are two inclusions s.t. $\gamma \geq \beta \geq \alpha$ but $P(\beta) > P(\gamma) > P(\alpha)$, the degree of the violation is 2.

Figures 14, 15 and 16 show the results. The horizontal axis shows the degrees of violations, except that zero denotes the number of probabilities that are *not* violations. The vertical axis represents the number of violations for each degree. The result shows that for 99.0% - 99.9% of the inclusions that are placed wrongly in the ranked results, the distance from the correct position is within at most 20.

6.4.2 Experiment 4: The Simple and Group-Conscious Methods

Although both simple and group-conscious methods yield topological sorts of inclusions with $\geq$, the latter was designed to produce better results. The experiment clarifies the difference between the two methods with the real data sets.

Result. Figures 17, 18, and 19 demonstrate the results. The horizontal axis shows the position in the ranking. The vertical axis shows the accumulated number of covered inclusions among the inclusions ranked so far. The figures clearly demonstrate the effectiveness of the group-conscious algorithm. For example, Figure17 shows that for Set 1, the first 33% of inclusions cover 90% of inclusions with the group-conscious algorithm, whereas only 26% inclusions are covered with the simple algorithm. The advantage is more evident for the other sets. For Set 2, the first 3% of inclusions cover 90% of inclusions, whereas only 40% of inclusions are covered with the simple

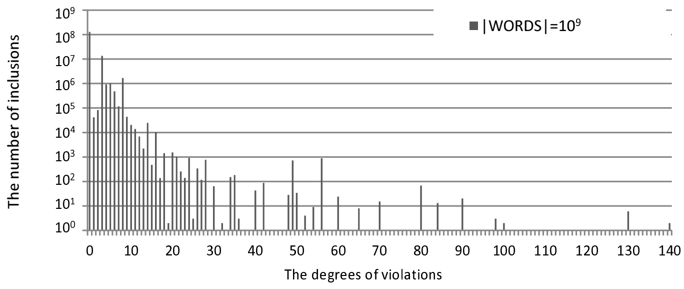


Figure 16: Experiment 3: Violation degrees for $inclusions_3$.

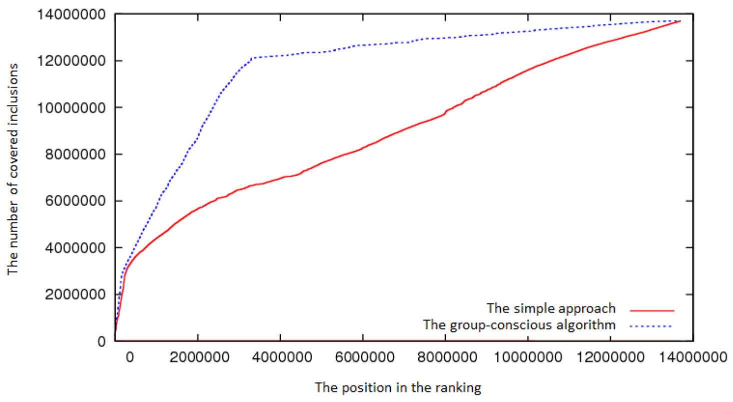


Figure 17: Experiment 4: Results with $inclusions_1$.

method (Figure 18). For Set 3, the top 5% of inclusions cover 90% of inclusions, while only 41% of inclusions are covered with the simple method (Figure 19).

7 Conclusion

This paper showed the results of a comprehensive study on finding inclusion dependencies on the Web. First, we introduced a bit-based signature scheme to efficiently reduce the number of pairs of page elements that are irrelevant to inclusion dependencies. The signature scheme is unique in that it can serve as an efficient filter to deal with Jaccard containment, in order to cope with the incomplete nature of Web data. Second, we discussed ranking schemes for the enumerated inclusions to support the verification process to find inclusion dependencies. We introduced the notion of covers to efficiently look through the enumerated inclusions, and proved that there are efficient algorithms to compute probabilities that are compatible with the definition of covers. In addition to the theoretical results, the paper presents a comprehensive set of experimental results with a variety of real Web sites, that showed the effectiveness of the proposed techniques.

Acknowledgments

The authors are grateful to Prof. Tetsuo Sakaguchi and Prof. Mitsuharu Nagamori for the discussion in seminars. This research was partially supported by the Grant-in-Aid for Scientific Research (#22240023) from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan.

References

[1] Parag Agrawal, Arvind Arasu, Raghav Kaushik. On Indexing Error-Tolerant Set Containment. SIGMOD 2010, 927-938.

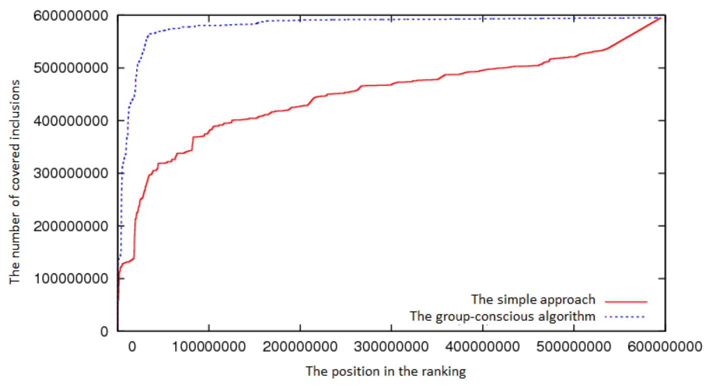


Figure 18: Experiment 4: Results with $inclusions_2$.

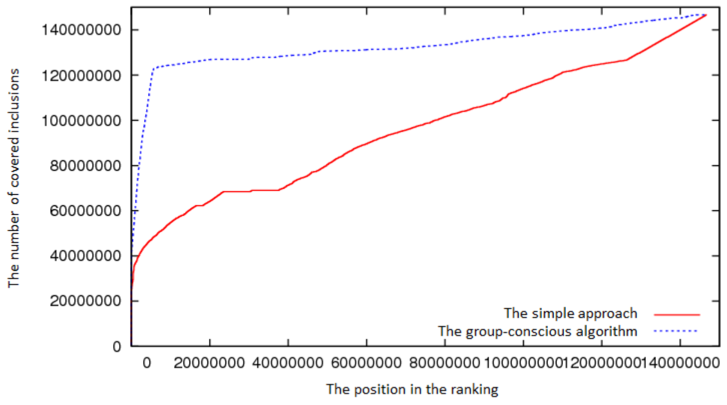


Figure 19: Experiment 4: Results with $inclusions_3$.

[2] Jana Bauckmann, Ulf Leser, Felix Naumann. Efficiently Computing Inclusion Dependencies for Schema Discovery. ICDE Workshops 2006, 2.

[3] Moses Charikar. Similarity estimation techniques from rounding algorithms. STOC 2002, 380-388.

[4] Surajit Chaudhuri, Venkatesh Ganti, Raghav Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning. ICDE 2006, 5.

[5] Charles L. A. Clarke. Controlling overlap in content-oriented XML retrieval. SIGIR 2005, 314-321.

[6] Wenfei Fan. Dependencies revisited for improving data quality. PODS 2008, 159-170.

[7] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Wenyuan Yu. Towards Certain Fixes with Editing Rules and Master Data. VLDB 2010, vol.3, No.1, 173-184.

[8] Yoshiharu Ishikawa, Hiroyuki Kitagawa, Nobuo Ohbo. Evaluation of Signature Files as Set Access Facilities in OODBs. SIGMOD 1993, 247-256.

[9] Michael Karlinger, Millist Vincent and Michael Schrefl. Inclusion Dependencies in XML. Extending Relational Semantics. DEXA 2009, 23-37.

[10] Dongwon Lee, Murali Mani,Frank Chiu, Wesley W. Chu. NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints. CIKM 2002, 282-291.

[11] Fabien De Marchi, Stephane Lopes, Jean-Marc Petit. Unary and n-ary inclusion dependency discovery in relational databases. J. Intell. Inf. Syst. Vol.32, No.1, 53-73, 2009.

[12] Sergey Melnik, Hector Garcia Molina. Adaptive Algorithms for Set Containment Joins. ACM Trans. Database Systems, 2003, Vol.28, No.1, 56-99.

[13] Hang Shi, Toshiyuki Amagasa, Hiroyuki Kitagawa. Fast Detection of Functional Dependencies in XML Data. XSym 2010, 113-127.

[14] Sihem Amer-Yahia, Mounia Lalmas: XML search: languages, INEX and scoring. SIGMOD Record, Vol.35, No.4, 16-23, 2006.

[15] Gurmeet Singh Manku, Arvind Jain, Anish Das Sarma. Detecting near-duplicates for web crawling, WWW 2007, 141-150.

[16] Sunita Sarawagi, Alok Kirpal. Efficient set joins on similarity predicates. SIGMOD 2004, 743-754.

[17] Masami Takahashi, Natsumi Sawa, Atsuyuki Morishima, Shigeo Sugimoto, Hiroyuki Kitagawa. Development of a Support Tool for the Maintenance of Web Content Integrity. ipsj 70, 189-190, 2008.

[18] Ningning Wu, Irit Askira Gelman, Isaac O. Osesina. How consistent is web information - A case study on online real estate databases, AMCIS 2009, 437.

[19] Chuan Xiao,Wei Wang, Xuemin Lin, Jeffrey Xu Yu. Efficient Similarity Joins for Near Duplicate Detection. WWW 2008, 131-140.

[20] Erika Yumiya, Atsuyuki Morishima, Shigeo Sugimoto, Hiroyuki Kitagawa. A Ranking Method for The Discovery of Inclusion Dependencies in Web Contents. DEIM 2011.

[21] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, Divesh Srivastava. On Multi-Column Foreign key Discovery. VLDB 2010, vol.3, No 1, 805-814.

[22] The School of Informatics, http://inf.tsukuba.ac.jp, (accessed 2010-07-13).

[23] College of Knowledge and Library Sciences, http://klis.tsukuba.a-c.jp/, (accessed 2010-07-13).

[24] College of Media Arts, Science and Technology, http://www.mast.ts-ukuba.ac.jp/, (accessed 2010-07-13).

[25] College of Information Science, http://www.coins.tsukuba.ac.-jp/, (accessed 2010-07-13).

[26] ACM SPECIAL INTEREST GROUP ON MANAGEMENT OF DATA, http://www.sigmod.org/, (accessed 2011-06-20).

[27] ACM Special Interest Group on Information Retrieval, http://www.sigir.org/, (accessed 2011-06-24).

[28] ACM Special Interest Group on Hypertext, Hypermedia and the Web, http://www.sigweb.org/, (accessed 2011-06-24).

[29] TOKYO ELECTRIC POWER COMPANY, http://www.tepco.co.jp/index-j.html, (accessed 2011-06-29).

[30] Nuclear Safety Commission of japan, http://www.nsc.go.jp/index.htm, (accessed 2011-06-30).

[31] Nuclear and Industrial Safety Agency, http://www.nisa.meti.go.jp/, (accessed 2011-07-03).

[32] Sen, http://www.mlab.im.dendai.ac.jp/ yamada/ir/MorphologicalAnalyzer/Sen.html, (accessed 2010-05-20).