Optimize Document Identifier Assignment for Inverted Index Compression

Chong Chen  
Department of Information Management  
Beijing Normal University  
Beijing, China  
chenchong@bnu.edu.cn

Jing He  Dongdong Shan  Hongfei Yan  
Department of Computer Science  
Peking University  
Beijing, China  
{hj, sdd, yhf}@net.pku.edu.cn

Abstract—Document identifier assignment is a technique for inverted file index compression, by reducing d-gap value of posting lists. It was approached by either TSP or clustering methods in existing study. However, there is no proper formulation for this problem and the existing approaches has no theory guarantee to be good approximations. In this paper, we first formulate document identifier assignment problem as an optimization problem, and then propose a new method to solve it approximately. Our method first clusters the documents by URL information and then rearranges the documents and clusters with benefit function, which is derived by minimizing posting space directly. TSP method can be considered as one simple special case of our method. The experiments shows that it achieves a good trade-off between efficiency and effectiveness.

Keywords—document identifier, cluster, inverted index compression, optimization

I. INTRODUCTION

Inverted file index is an important component for IR system. When the dataset becomes larger, index efficiency a very challenging problem. Compression is one technique to improve inverted file index efficiency for two reasons: 1) it reduces the disk space for inverted index; 2) in query evaluation it reduces up loading time, which is the most expensive phase in searching.

Inverted file index is composed by a collection of posting lists, each of which stores the identifiers of documents containing one term. The documents in a posting list is sorted by either document identifiers or impact. For the first type, in a posting list, the first posting stores original document identifier and the subsequent postings store d-gaps[10].

Inverted file index compression is to compress the collection of d-gap values. There are two categories of compression algorithms: bitwise[10] and bytewise[1]. Bitwise algorithms compress a value to a number of bits and byte-wise algorithms require the compressed data to be aligned by byte. The first class algorithm has better compression ratio but the second class has faster speed. There are still some research on how to change the d-gap distribution to improve compression ratio.

Almost all compression algorithms have the property that it takes smaller space for smaller value. Therefore, the compression ratio can be improved by decreasing d-gap values. One effective approach for this task is to reassign document identifier. For example, for documents A, B, C, D and terms t1, t2, t3, the document-term matrix is shown in Table I. In the original document assignment d1 = A, d2 = B, d3 = C, d4 = D, the posting lists are l1 = (1, 3), l2 = (2, 4), l3 = (1, 3), for term t1, t2, t3 separately. After converting original document identifier to d-gap, the posting lists become l1 = (1, 2), l2 = (2, 2), l3 = (1, 2). However, if we reassign the document identifiers to be d1 = A, d2 = C, d3 = B, d4 = D, the d-gap posting lists become l1 = (1, 1), l2 = (3, 1), l3 = (1, 1). It is obvious that the overall values in the posting lists become smaller. In this example, we assign close identifier for document pairs (A, C) and (B, D), whose content are very similar. More generally, there are two similar documents d and d′, which share a large number of words. If it assigns document identifiers i and i + 1 to them separately, there would be a d-gap 1 for each common word. If assigned identifiers far away, d-gap becomes larger. The property is named as clustering property[9], and document identifier assignment is motivated by this intuition.

The document identifier assignment problem is either approximated with travelling salesman problem(TSP) or clustering problem. Both approaches are inspired by the clustering property. However, there is no proper formulation for this problem so that it’s difficult to determine which approximation is better. In this paper, we first formulate this problem to be an optimization problem. With the objective function, we derive an approximation algorithm for it. It’s a hybrid algorithm, combining clustering and benefit function based solutions. TSP method can be considered as one special simple case of the benefit based solution. The result
shows that it is a good and flexible trade-off between efficiency and effectiveness. The contribution of this paper includes:

- propose a proper formulation to the document identifier problem
- propose a URL tree resort algorithm to achieve good and flexible trade-off between efficiency and effectiveness
- derive a benefit based weight function from posting storage space directly
- systematically compare the existing weight functions with significance test

II. RELATED WORK

Shieh[7] approximated document identifier assignment problem as TSP. TSP is defined on a graph and the solution is a cycle path with maximal weight sum. It models a document as a vertex and between any pair of vertexes there is an edge, whose weight is defined by the document similarity. With the maximal weight path, the document identifier is assigned to the document on the path in sequence. This approach only considers the similarity of direct neighbor documents whose identifiers differ by 1. However, the exact solution of TSP is not necessary the optimal solution to document identifier assignment problem, because it ignores the documents whose identifiers differ more than 1. Because TSP is a NP-hard problem, Shieh approximates it by greedy-NN algorithm. In addition, this approach also has efficiency problem, because it is too expensive to compute and store the whole document-document similarity. To overcome the efficiency problem, Blanco and Barreiro[4], [3] proposed two variance methods for TSP problem. The first method is to just store a compressed document similarity graph by Singular Value Decomposition(SVD) approach. The second method(c-block) splits the document collection into blocks, and then runs greedy-NN algorithm both intra-block and inter-blocks. These two method reduced computation time a lot, and effectiveness is affected not so much. But these two methods are still not very scalable.

Another effort is to solve this problem by clustering approach. Blelloch and Blandford[5] used clustering techniques firstly. As the TSP approach, it constructs a document similarity graph, and splits the graph into smaller graphs. With a document tree, whose leaf node is document and inner node is document set, it assign document identifier by depth-first visiting the tree. Like Shieh’s[7] method, it’s also not scalable because it needs to store a complete document similarity graph. Some work followed to make it more efficient. [9] proposed a series of one-pass clustering techniques and it doesn’t require a pre-built index. The most efficient method in this category is URL-sorting method[8]. In this method, it sorts the documents according to their URLs and assigns document identifiers according the URL order. This method performs well and scalable. However, as TSP problem, there is no guarantee that good clustering result necessarily leads good compression result. Specifically, it’s not reasonable to employ simple depth-first visiting for document assignment after clustering.

The only formulation work on this problem is by Blanco and Barreiro[2], formulating it as Pattern Sequence Problem(PSP). Its objective function is to minimize the sum of d-gap values, so it is equal to minimize the sum of document identifier difference between the minimal and maximal document identifiers in a posting list. However, such formulation cannot present the real required space, because compression algorithm usually compress a value in much fewer number of bits than the actual value. We should reformulate it with a more reasonable objective function.

III. OPTIMIZATION PROBLEM

For a document collection, the inverted file index is a data structure to find out the document subset containing a term fast. It is composed by a collection of posting lists, each one of which is a list of document identifiers in increasing order.

The document-term information is always presented by a matrix $M$. Each row of it presents a document and the $i$-th row is for document with identifier $i$. Each column presents a term. $M_{i,j}$ is a binary value, indicating whether document $i$ contains term $j$. One example is shown in Table I. The document identifier problem is to find of row permutation $\pi$ of the matrix $M$, so that the inverted file index space required for permuted matrix $M_\pi$ is minimal. The optimization problem can be formulated as below:

$$\pi_0 = \arg\min_\pi \sum_{j=0}^{\vert T \vert} (s(\pi(d_{j1})) + \sum_{k=2}^{f_j} s(\pi(d_{jk}) - \pi(d_{j_{k-1}})))$$

where $\vert T \vert$ is total number of unique terms in the collection, $f_j$ is document frequency for term $t_j$, and $(\pi(d_{j1}), \ldots, \pi(d_{j_{f_j}}))$ is the identifiers of documents who contain term $t_j$ after permutation. $s$ is a storage space function presents how many bits required to a integer value. Here we name $\pi(d_{jk}) - \pi(d_{j_{k-1}})$ to be distance between neighbor document in posting list. In [2], it has not introduced storage space function $s$ in its formulation, and default uses unary encoding for their presentation. However, actually no inverted file index is compressed with unary encoding because it spends too much space. Our formulation, however, can be more generally adapted to various of compression algorithms. With this formulation, we have the principle to select the weight function and approximate algorithm.

IV. APPROXIMATE ALGORITHM

In this section, we propose an algorithm to solve the optimization problem approximately. First, we propose the framework of URL tree resort algorithm. And then we introduce two important components: document collection
ordering algorithm and document subsets ordering algorithm. We also derive the weight function based on posting storage space.

A. URL Tree Resort Algorithm

For Web scale dataset, it is too expensive to get exact optimal solution for the problem, so we approximate it. TSP and clustering methods are two examples of approximate algorithms, but neither of them connect with the optimization problem, especially the storage space function directly. URL sorting algorithm[8], which is most efficient algorithm for this problem, can extend to Web scale, but it has the same drawback of other clustering methods. Specifically, it is not reasonable to assign the order of all pages or directories with same path prefix according to their alphabet order. To overcome these deficiencies, we propose URL tree resort algorithm: it uses URL information to build a URL tree as first step for its efficiency and employs content similarity based method to boost its effectiveness. The pseudo code for this algorithm is presented in Algorithm 1.

This algorithm firstly builds a document tree(buildTree) by URL information. The root of the tree is a pseudo node and the second-level nodes are all sites. The other inner nodes are directories and the leaf nodes are pages. The document tree keeps only clustering information but not ordering information, because clustering by site/directory is effective and efficient but ordering by alphabet is not reasonable. The ordering information is achieved by function orderDocument and orderSubset. While pre-order visiting the tree, it orders documents for small document collection, whose size is less than threshold \( t \), and then orders subsets for large node. Because it is pre-order visiting, the documents in subsets must be in order before it orders these subsets. For example, one document tree is presented in Figure 1. The document tree is composed by three sites: \( A, B \) and \( C \). \( A \) has three directories: \( A_1, A_2, A_3 \). \( B \) has two directories \( B_1, B_2 \) and \( C \) has no directory. It assumes that all the directories has less documents than threshold \( t \). After building this tree as first step, for site \( A \), it orders the documents in directories \( A_1, A_2 \) and \( A_3 \) by orderDocument function separately, and then orders subsets \( A_1, A_2, A_3 \) in sites \( A \) by orderSubset function. It works similarly for sites \( B \) and \( C \). After all documents in site \( A, B, C \) are ordered, it orders \( A, B, C \) by orderSubset function.

```
Input: document collection \( D \), document group size threshold \( t \)  
Output: ordered document \( (d_1, \ldots, d_N) \)  
begin  
\( T \leftarrow \text{buildTree}(D) \)  
while node \( \leftarrow \text{preorderVisit}(T) \) do  
if node.documentNum() < \( t \) then  
  \( \text{orderDocument}(\text{node}) \)  
else  
  \( \text{orderSubset}(\text{node}) \)  
end  
end  
Algorithm 1: URL Tree Resort Algorithm
```

One approach to order the document collection and subset is to employ TSP method multiple times. This approach is similar to c-blocks algorithm[4]. There are two differences between our algorithm and c-block:

- c-block method divides the document collection to be fixed size blocks arbitrarily. However, our blocks are divided by URL clusters and the numbers of document in blocks are not fixed.
- c-block method has a set of flat blocks, but our clusters are organized hierarchically, so it requires to run greedyNN algorithm at each level.

A drawback of TSP method is that it only considers the documents whose identifier differs by 1. However, in our problem formulation, the space required is determined by neighbor documents of posting lists, and the neighbor documents in posting list are not necessary have identifier difference 1. Intuitively, it can improve compression performance by considering more documents whose identifiers differ larger than 1. For example, for a document sequence \( d_1, d_2, \ldots, d_n \), the TSP method sums up all the similarity weights between \( d_i \) and \( d_{i+1} \), \( 1 \leq i \leq n - 1 \). We can extend to consider weights between two arbitrary \( d_i \) and \( d_j \), \( 1 \leq i, j \leq n \). One problem is that the weight between two documents is affected not only by the content of two documents, but also the document identifier difference and content of documents between them. The algorithm and the weight function is discussed in section IV-B, also based on benefit function.

The subset ordering method in c-block is simply by ordering samples selected from subsets. It has no guarantee

Figure 1. Document Tree
that a good order of samples leads good order of subsets, so we propose another ordering algorithm for subsets in section IV-C.

B. Order Document Collection

We propose a greedy algorithm to approximate document identifier assignment problem, and this algorithm is the implementation of orderDocument function in Algorithm 1.

<table>
<thead>
<tr>
<th>Input:</th>
<th>unassigned identifier document collection D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>ordered document queue q = (d_1, ..., d_N)</td>
</tr>
<tr>
<td>begin</td>
<td>remove document d_0 randomly from collection D</td>
</tr>
<tr>
<td>q.append(d_0)</td>
<td>while D is not empty do</td>
</tr>
<tr>
<td>d_0 = argmax_d weightFunction(d, q, D)</td>
<td>remove d_0 from collection D</td>
</tr>
<tr>
<td>q.append(d_0)</td>
<td>end</td>
</tr>
</tbody>
</table>

Algorithm 2: Order Document Collection

This algorithm selects a document randomly from the collection as the first document, assigning identifier 1. With documents d_1, ..., d_k, which have been assigned identifiers 1, ..., i, it assigns identifier i + 1 to the document which leads maximal storage space benefit. The pseudo code for this algorithm is presented in Algorithm 2. The first document is randomly selected, so it may not be very good. One variance is to run this algorithm for several times and select the run with maximal overall benefit as the final ordering.

Weight function is a very important component for this algorithm and other existing methods. There are various of weight functions used, most of which are based on content similarity of two documents. In [9], it used Jaccard similarity coefficient to measure the similarity. The Jaccard similarity coefficient is a common used metric to measure set similarity, defined as:

$$w_{A,B} = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are term sets for two documents. Another common used similarity measure is cosine coefficient[5], defined as:

$$w_{(X_1, X_2)} = \frac{X_1 \cdot X_2}{|X_1| \cdot |X_2|}$$

where X_1 and X_2 are two documents’ vector presentation, each element of the vector is 0 or 1 for a term, indicating whether it is contained in the document. Cosine is the one of most popular measures for documents in information retrieval. However, these weight functions are just “borrowed” from other applications. It’s lack of validation that they are effective for document identifier assignment problem.

We derive a good weight function from the optimization problem defined in section III. Moreover, as we state in section IV-A, in our algorithm, we would consider not only the neighbor documents whose document identifiers differ by 1, but extend to longer-distance documents. However, the existing weight function above cannot consider the identifier distance.

We propose a benefit based weight function. The main idea of this function is to measure how much “benefit” we can achieve by assigning a document identifier. For example, if we have assigned document identifier for d_1, ..., d_i, and D is documents which are not assigned identifiers, we will calculate how much benefit we can achieve by assigning identifier i + 1 to document d (d ∈ D).

Firstly, let’s consider last assigned document d_i and document to be assigned d. There are three types of terms in d_i or d: the common terms t ∈ d_i ∩ d, the terms belonging to d_i only t ∈ d_i − d) and those belonging to d only t ∈ d − d_i). If assigned identifier i + 1 to d, for a common term t, the storage space in the posting list after d_i is composed by two parts: one posting for storing d-gap between d_i and d, and number of postings for storing the d-gaps between the unassigned documents containing term t. The first part is simply s(1), and the second part is (df(t, D) − 1) · s(df(t, D) − 1, |D| − 1), where s(n) is storage space for value n, df(t, D) is document frequency for t in collection D and π(df, dn) is a function to calculate the space expectation for term t with document frequency df and total document number dn. If assigning document identifier randomly, the d-gap distribution should be a geometric distribution, whose mean is \( \frac{dn}{df+1} \). Therefore the posting space expectation for a term t should be:

$$\pi(df(t), dn) = \sum_{k=1}^{dn-df(t)+1} Pr(dgap = k) \cdot s(k)$$

$$= \sum_{k=1}^{dn-df(t)+1} (1 - p(t))^{k-1} \cdot p(t) \cdot s(k)$$

$$p(t) = \frac{df(t) + 1}{dn}$$

(1)

If not assigning identifier i + 1 to document d, for a common term t, the space expectation for a posting list after d_i is \( \pi(df(t, D), |D|) \). In summary, the benefit from a common term is presented in Equations 2. Similarly, we can also calculate the benefit from other two types of terms, but we ignore them because it is much smaller.

$$b(t, D) = s_{u(t, D)} - s_{a(t, D)}$$

$$s_{u(t, D)} = df(t, D) \cdot \pi(df(t, D), |D|)$$

$$s_{a(t, D)} = s(1) + (df(t, D) − 1) \cdot \pi(df(t, D) − 1, |D| − 1)$$

So the overall benefit from d_i and d is the sum of all common words of d_i and d:

$$b(d_i, d, D) = \sum_{t \in d_i \cap d} b(t, D)$$
More generally, there is still benefit from \( d_{i-k} \) and \( d \) by assigning identifier \( i+1 \) to document \( d \). Similarly to the derivation above, the benefit can be summed up by all common words of them, but the benefit function for each common word is different from that of \( d_i \) and \( d \). First, it should consider the distance between the documents: the longer distance leads less benefit because it costs more storage space for longer distance; Second, it should consider the documents \( \{d_{i-k}, \ldots, d_i\} \), which are between \( d_{i-k} \) and \( d \); if any of these documents contain the common word \( t \), the benefit is “blocked” by the document, because there is no direct \( d \)-gap between \( d_{i-k} \) and \( d \). For a list of assigned identifier documents \( q = (d_1, \ldots, d_i) \), the benefit of assigning identifier \( i+1 \) to document \( d \) is defined the sum of benefits between \( d \) and all assigned identifiers documents.

\[
b(d, q, D) = \sum_{1 \leq j \leq i} b(d_j, d, q, D) \tag{3}
\]

\[
b(d_j, d, q, D) = \sum_{t \in d_j \cap d} b(t, j - i, D) \tag{4}
\]

\[
b(t, k, D) = s_u(t, k, D) - s_a(t, k, D)
\]

\[
s_u(t, k, D) = \pi(df(t, D), |D|, k) + (df(t, D) - 1) \cdot \pi(df(t, D), |D|)
\]

\[
s_a(t, k, D) = s(k + 1) + (df(t, D) - 1) \cdot \pi(df(t, D) - 1, |D| - 1)
\]

This benefit function is more powerful and effective than existing weight function for two reasons: 1) it is derived directly from the storage space, but not just employing other measure for content similarity arbitrarily; 2) it considers document identifier difference, which is required in our URL resort algorithm. Now we are going to specifically define the storage space function \( s \).

The upper bound for lossless compression algorithm for a positive integer \( n \) is \( \log_2(n) \). Currently, there are many state-of-art algorithms approaching this upper bound, so we approximate space function as \( \log_2 \). Another alternative method is to use a compression algorithm to estimate the space required directly. So our method can adapt to some specific compression algorithm in this way.

In this formulation, it calculates the benefit by summing up benefits from all assigned documents. If the document collection is large, the document far away affects little to the overall benefit, because more common words are blocked and benefit decreases as distance increases. One alternative approach is to set a threshold \( dist \), and only the documents whose identifier distance is smaller than this threshold are considered. When the threshold \( dist \) is to set the threshold to be 1, it is same as the greedyNN algorithm for TSP. So we can see that the existing TSP method is a simple special case of our approach.

### C. Order Document Subsets

As presented in Algorithm 1, the document subsets are ordered after the documents in the subsets are ordered. The pseudo code for this algorithm is presented in Algorithm 3. The weight between a subset \( D \) and ordered subsets \( q \) can be evaluated by the benefit to put the subset next the ordered subsets, presented in Equation 4, where \( D' \) is all documents in the unsigned identifier subsets and benefit function \( b(d, q, D) \) from document \( d \), document list \( q \) and unsigned document collection \( D \) is defined in Equation 3. In the equation, the benefit is summed up by all benefit from all documents in subset \( D \) and the documents in \( q \). The weight is summed up by all weights for common terms in both any of document of subset \( D \) and any of document in ordered subsets \( q \). So it selects subset with least weight each time. Similar to Algorithm 2, this algorithm can also run several times and select the ordering with the maximal overall benefit.

```
Algorithm 3: Order Document Subsets

\[ b(q, D, D') = \sum_{d_i \in D} b(d_i, q + D_{1,i-1}, D') \tag{4} \]
```

### V. Experiment AND Result Analysis

#### A. Document Identifier Assignment Algorithms

We perform experiments on WT2g dataset, which is one TREC standard document collection for Web information retrieval evaluation. It contains 250,000 documents and takes 2 gigabytes. As the hardware platform, we use a Pentium IV 4.0GHz, with 2GB of RAM and Redhat Linux operating system. For encoding algorithm, we consider Oracle, Gamma and Simple. Oracle is an optimal encoding ratio, requiring \( \log_2(n) \) storage space for integer \( n \). Gamma and Simple9 presents the result for bitwise and bytewise result separately.

[8] shows that the URL sorting, content clustering and TSP methods can achieve comparable, so we use URL sorting method as our baseline. The result is presented in table II. The best performance of document assignment algorithm for each compression algorithm is in bold. This result shows
that URL tree resort algorithm can achieve much better compression ratio than the URL sorting algorithms for bitwise compression. Specifically, it can see that requires 8.4%-9.8% less space for oracle encoding and 3.9%-4.9% less space for Gamma encoding. However, it doesn’t perform much better with bytewise algorithm, because it is affected by more other factors than values.

For computation complexity, URL tree resort algorithm takes longer time than URL sorting algorithm because it needs to analyze the content of Web page. However, as we stated, the complexity of the algorithm is also $O(n\log n)$, which is same as URL sorting algorithm. It can be validated by the time change with size in Table II. Furthermore, we argue that: 1) the compression ratio, which has effect online retrieval performance, is much more important than offline indexing performance; 2) the identifier assignment is not the bottleneck of indexing because it only takes a small ratio of preprocessing/indexing time. In the 7th column of Table II, we show the total time of indexing plus document identifier assignment. Random algorithm doesn’t need extra time for document identifier assignment, so its total time is just indexing time. Compared to it, both URL sorting and tree resorting algorithm do not increase a lot of indexing cost.

### B. Weight Functions

The existing work has no empirical comparison between different weight functions. Here we validate the effectiveness of four types of weight functions: common word weight function, Jaccard weight function, cosine weight function and benefit based weight function.

The first experiment is to use URL resort algorithm, but with different weight functions. The first three algorithms, however, can not extend to longer distance benefit as order-Document and orderSubset function do, so it just consider the closest document for approximation. The results are reported in Table III.

<table>
<thead>
<tr>
<th>Weight Function</th>
<th>Oracle</th>
<th>Gamma</th>
<th>Simple9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cosine</td>
<td>2.55</td>
<td>2.02</td>
<td>8.51</td>
</tr>
<tr>
<td>Jaccard</td>
<td>2.14</td>
<td>6.97</td>
<td>8.47</td>
</tr>
<tr>
<td>Benefit</td>
<td>2.42</td>
<td>6.89</td>
<td>8.44</td>
</tr>
</tbody>
</table>

Table III

**RESULTS FOR WEIGHT FUNCTIONS**

We formulate document identifier assignment problem and present **URL tree resort algorithm** to solve it approximately. This algorithm firstly clusters the documents by their URL information and then resort them hierarchically. In the algorithm, we derive the benefit based weight function directly from the space storage function and can adapt to specific compression algorithm. The experiment results show that **URL tree resort algorithm** is a good and flexible trade-off between effectiveness and efficiency. Benefit based weight function outperforms the other weight functions by significance test, and we also show the empirical comparison result for the other weight functions, showing that common-word weight function is also a good choice for both effectiveness and efficiency.

### VI. Conclusion

We formulate document identifier assignment problem and present **URL tree resort algorithm** to solve it approximately. This algorithm firstly clusters the documents by their URL information and then resort them hierarchically. In the algorithm, we derive the benefit based weight function directly from the space storage function and can adapt to specific compression algorithm. The experiment results show that **URL tree resort algorithm** is a good and flexible trade-off between effectiveness and efficiency. Benefit based weight function outperforms the other weight functions by significance test, and we also show the empirical comparison result for the other weight functions, showing that common-word weight function is also a good choice for both effectiveness and efficiency.

### ACKNOWLEDGMENT

This work is partially supported by NSFC under the grant number 70903008 and 60933004, CNGI grant number 2008-122 and 863 Program No. 2009AA01Z143.

### REFERENCES


<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Oracle</th>
<th>Gamma</th>
<th>Simple9</th>
<th>AssignTime(s)</th>
<th>TotalTime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50K Random</td>
<td>3.91</td>
<td>9.22</td>
<td>9.64</td>
<td>-</td>
<td>-707.4</td>
</tr>
<tr>
<td>50K URL Sorting</td>
<td>2.49</td>
<td>6.85</td>
<td>8.15</td>
<td>18.0</td>
<td>605.3</td>
</tr>
<tr>
<td>50K URL Tree Resort</td>
<td>2.28</td>
<td>6.58</td>
<td>8.08</td>
<td>165.6</td>
<td>752.9</td>
</tr>
<tr>
<td>100K Random</td>
<td>4.14</td>
<td>9.59</td>
<td>10.10</td>
<td>-</td>
<td>1188.9</td>
</tr>
<tr>
<td>100K URL Sorting</td>
<td>2.54</td>
<td>7.10</td>
<td>8.50</td>
<td>25.2</td>
<td>1218.1</td>
</tr>
<tr>
<td>100K URL Tree Resort</td>
<td>2.37</td>
<td>6.75</td>
<td>8.41</td>
<td>347.0</td>
<td>1535.9</td>
</tr>
<tr>
<td>150K Random</td>
<td>4.19</td>
<td>9.78</td>
<td>10.21</td>
<td>-</td>
<td>1807.8</td>
</tr>
<tr>
<td>150K URL Sorting</td>
<td>2.46</td>
<td>7.17</td>
<td>8.52</td>
<td>41.7</td>
<td>1832.6</td>
</tr>
<tr>
<td>150K URL Tree Resort</td>
<td>2.43</td>
<td>6.88</td>
<td>8.44</td>
<td>498.8</td>
<td>2306.6</td>
</tr>
<tr>
<td>200K Random</td>
<td>4.22</td>
<td>9.82</td>
<td>10.17</td>
<td>-</td>
<td>2425.3</td>
</tr>
<tr>
<td>200K URL Sorting</td>
<td>2.64</td>
<td>7.17</td>
<td>8.45</td>
<td>61.9</td>
<td>2487.2</td>
</tr>
<tr>
<td>200K URL Tree Resort</td>
<td>2.42</td>
<td>6.87</td>
<td>8.42</td>
<td>702.8</td>
<td>3128.1</td>
</tr>
<tr>
<td>250K Random</td>
<td>4.32</td>
<td>10.02</td>
<td>10.33</td>
<td>-</td>
<td>3223.6</td>
</tr>
<tr>
<td>250K URL Sorting</td>
<td>2.65</td>
<td>7.18</td>
<td>8.58</td>
<td>82.5</td>
<td>3306.1</td>
</tr>
<tr>
<td>250K URL Tree Resort</td>
<td>2.42</td>
<td>6.89</td>
<td>8.44</td>
<td>941.3</td>
<td>4164.9</td>
</tr>
</tbody>
</table>

Table II

DOCUMENT IDENTIFIER ASSIGNMENT ALGORITHMS


