Locality Sensitive Hashing for Scalable Structural Classification and Clustering of Web Documents

Christian Hachenberg
Institute for Web Science and Technologies,
University of Koblenz-Landau, Germany
hachenberg@uni-koblenz.de

Thomas Gottron
Institute for Web Science and Technologies,
University of Koblenz-Landau, Germany
gottron@uni-koblenz.de

ABSTRACT

Web content management systems as well as web front ends to databases usually use mechanisms based on homogeneous templates for generating and populating HTML documents containing structured, semi-structured or plain text data. Wrapper based information extraction techniques leverage such templates as an essential cornerstone of their functionality but rely heavily on the availability of proper training documents based on the specific template. Thus, structural classification and structural clustering of web documents is an important contributing factor to the success of those methods. We introduce a novel technique to support these two tasks: *template fingerprints*. Template fingerprints are locality sensitive hash values in the form of short sequences of characters which effectively represent the underlying template of a web document. Small changes in the document structure, as they may occur in template based documents, lead to no or only minor variations in the corresponding fingerprint. Based on the fingerprints we introduce a scalable index structure and algorithm for large collections of web documents, which can retrieve structurally similar documents efficiently. The effectiveness of our approach is empirically validated in a classification task on a data set of 13,237 documents based on 50 templates from different domains. The general efficiency and scalability is evaluated in a clustering task on a data set retrieved from the Open Directory Project comprising more than 3.6 million web documents. For both tasks, our template fingerprint approach provides results of high quality and demonstrates a linear runtime of $O(n)$ w.r.t. the number of documents.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

template detection, template fingerprints, locality sensitive hashing

CIKM'13 October 27 - November 01 2013, San Francisco, CA, USA

1. INTRODUCTION

Web content management systems (CMS) of different origin and type are widely available nowadays. Accordingly, more and more websites are run by open source CMS (e. g. Typo3, Plone, Joomla), blog environments (e. g. WordPress, TypePad, Moveable Type), enterprise CMS (e. g. RedDot, Cascade Server, EPiServer), online shop systems (e. g. Zen Cart, KonaKart, Batavi) or are hosted in a Software-as-a-Service setup (e. g. blogger.com, Business Catalyst). Furthermore, several websites use tailor made CMS to suit individual needs. A typical feature of CMS systems is the use of templates: predefined document structures with placeholder slots, which are filled and completed with content from backend databases. The intended effect is to implement a common look and feel in combination with a corporate design. A side effect is, that the documents have a highly similar structure in which the same type of information is usually located at the same position in the document's layout. This effect is heavily exploited in the field of web content mining and information extraction. Wrapper induction tools are able to *learn* the underlying template structure from example documents and identify the positions where information rich items can be found and how they relate to attributes of the entities in the backend database. This leads to extraction rules which can be applied to new documents based on the same template in order to discover and extract new information.

Given the central role of the template structures in this setting, several publications have addressed the task of template detection for the purpose of structural classification or clustering of web documents (e. g. [5, 1, 14, 4, 7, 6]). *Structural classification* is the task of identifying whether a document is based on one of k known templates, typically specified via training examples. The cost for this task is generally of $O(mn)$ depending on the number n of documents which need to be classified and the number m of training examples. Differences in runtime usually result only from higher computational costs for the comparison of two individual documents. *Structural clustering* of a collection of n documents, instead, is the task of forming groups of documents which are all based on the same template. So far, the solutions addressing this task require a pairwise comparison of all documents in the collection. This leads to an inherent quadratic complexity of $O(n^2)$ w.r.t. the number of web documents. Thus, these approaches for structural clustering are not really applicable for large document collections. Alternative approaches using the URL of a document [2, 9] as a non-template specific feature have been investigated but depend on the assumption of a strong correlation between the use of templates and the URL patterns of final documents. There are cases where this assumption does not hold. For instance, popular platforms such as Mediawiki[1] are installed on hundreds of web

[1] `http://www.mediawiki.org/` accessed: 2 August, 2013

sites and exhibit very different URLs although the structure of the web documents follows the same template on most of these installations. Vice versa, documents with similar URLs can be based on very different templates, e.g. when switching from the default version of a document for desktop browsers to its print version or a presentation style which suits mobile clients.

In the related context of near duplicate detection for the content of text documents, solutions based on *fingerprinting* techniques have proven to work well for an efficient identification of highly similar text. Fingerprints are usually based on a hash function which maps similar objects onto the same or highly similar hash values. Such locality sensitive hash values allow for a lookup of similar entities in constant runtime. Thus, a decision whether the content of a given document is highly similar to the content of another document can be made very fast.

In this paper, we take up this idea and present a *template fingerprint* method for web documents where we want to identify a high similarity in the structure of documents rather than the contents. For any given web document, the method computes a fingerprint of 25 characters to capture the structural information of the document and to represent template specific features in a very effective way. The fingerprint computation is inspired by compression techniques used for identifying redundant structures. In our case the redundancies correspond to recurring patterns in the document structure and this is attributed to an underlying template. The template fingerprints can be computed very efficiently. Furthermore, the decision, whether or not two documents are structurally similar can be answered with high accuracy based on the computation of the string edit distance between their corresponding fingerprints. We will present an index structure and efficient algorithm which is specifically designed towards our template fingerprints and allows to retrieve structurally similar documents in effectively constant runtime. [2]

We apply and evaluate our template fingerprint method and the index structure for retrieval of structurally similar documents in the two settings already mentioned before:

Structural Classification. For a given set of k templates provided by a set of training examples each, we analyse a corpus of n web documents and decide for each of the documents which of the k templates it is based on. Thus, we have to perform a classification task with k+1 classes. The additional class is necessary to cover the case that a document in the corpus is not based on any of the k known templates. Template fingerprints allow for a very efficient check whether or not a document belongs to any of the k classes. This leads to an overall complexity of $O(n)$ for classifying n documents, thus, being independent of the number of training documents.

Evaluation of the structural classification task is performed on a collection of 13,237 template-based documents from 50 different domains. We show that our fingerprint approach provides highly accurate results with perfect precision and very high recall of 0.9877. We compare our approach to six approaches from related work as well as to two approaches based on MD5 hash values computed over document structures. The fingerprint templates show competitive performance when being compared to the state-of-the-art approach under the aspect of classification accuracy. The theoretic advantage in runtime performance is reflected in practice by a speed-up factor of three in the classification task compared to the state-of-the art approach.

[2]The source code and the evaluation data of our experiments are released at `http://west.uni-koblenz.de/Research/DataSets/template-detection`.

Structural Clustering. For a collection of n web documents, we form groups of documents in which all documents are based on the same template. Thus, within the set of documents we have to identify all the templates and assign each document to one of these templates. Template fingerprints can be used here to build clusters of structurally similar documents in a single pass over the document collection. Our approach to template clustering has a linear complexity of $O(n)$.

We evaluate the task of structural clustering on two data sets. On the one hand we use again the data set of 13,237 documents for which a gold standard is known. The gold standard is used to estimate the quality of the obtained cluster configuration. On the other hand we demonstrate scalability using a data set of more than 3.6 million documents obtained from the Open Directory project (DMOZ)[3]. This large-scale experiment confirms empirically the theoretically motivated linear runtime for our clustering approach.

We proceed as follows: After reviewing related work in Section 2 we introduce our template fingerprints in Section 3. There, we also describe the index structure and algorithms for efficient retrieval of structurally similar documents. In Section 4 we demonstrate the effectiveness and efficiency of the template fingerprint approach for structural classification and compare its performance with state-of-the-art methods. The task of structurally clustering web documents is discussed in Section 5. We conclude the paper with a discussion of our findings and future work.

2. RELATED WORK

Structural comparisons of HTML documents have been investigated for more than a decade. Initial approaches used HTML elements to create tag feature vectors [5]. A first approach to template identification was proposed in 2002 by Bar-Yossef and Rajagopalan [1]. They defined a template as a collection of HTML documents which have the same look and feel and which are controlled by a single authority. Identification of templates is achieved via the detection of recurrent sub-document elements which can be identified in a collection of documents. The recurrent sub-document elements are extracted via DOM tree segmentation.

Subsequently, the topic was addressed by other publications. One common approach to the computation of structural comparisons is to perform alignments on the DOM tree structure in order to calculate tree edit costs [14, 16, 19]. The common idea of these approaches is, that the structural changes between documents based on the same template are relatively small. Hence, also the tree edit costs for transforming DOM trees of two documents into each other is significantly lower if the documents are based on the same template. The major drawback of these approaches is the high complexity of the tree alignment step. This also applies to approaches using tag sequence alignments [7].

In order to overcome this drawback alternative approaches try to reduce complexity of the comparison between two documents. One solution is to use path based similarity metrics [10, 19]. Here, instead of the full tree structure, documents are compared on the basis of the paths from the root to the leaf nodes in the DOM tree. A further reduction of the dimensionality can be obtained by shingling [3]. Surprisingly, document comparisons based on path shingling or tag sequence shingling are not only more efficient in the pairwise comparison of documents, but have shown to produce qualitatively better cluster configurations, too [7]. This approach has also been applied successfully in a setting of structural classification [8].

[3]`http://www.dmoz.org/` accessed 2 August, 2013

Yet another approach is to entirely ignore structural properties of web documents and focus on other features that have a tendency to correlate with the usage of the same template instead. Quite often, the URL can be a good proxy feature for the use of the same template [9]. This observation is used by Blanco et. al. [2] in a template clustering approach which leverages terms extracted from the URL of documents. The approach is based on a greedy algorithm to find a minimum set of generative patterns to explain a collection of observed URLs. Each generative pattern corresponds to a template. The algorithm scales well and is of linear complexity. Even though the approach is said to be easily extensible to content terms, URLs remain a weak and potentially misleading pointer to corresponding template structures (cf. example in section 1).

A related problem to template detection is near duplicate detection for the content of text documents. Here, fingerprint techniques have already been developed successfully [18]. The general idea of fingerprint techniques is to reduce similar objects onto the same hash value. In this way, all similar objects can be sorted into the same *bucket* of hash values and retrieval of similar objects can be achieved in constant time. Important to mention is the inherent notion of *similarity* which bears the challenge. Not only identical objects, but also objects with minor differences should be identifiable via the same fingerprint hash value. A typical solution with fingerprints from plain text is to use multiple implementations of the hash function which groups objects around different centroids of similarity together in overlapping buckets. A review of the principles behind fingerprint construction for near duplicates in text is given in [17]. However, near duplicate detection in text documents is an orthogonal problem to template detection. For template detection we intentionally want to allow different textual contents, as long as the document structure is highly similar.

An efficient data structure for finding bit vector fingerprints allowing a maximum hamming distance of k is discussed in [12]. The bit vector fingerprints are distributed randomly over several tables of approximately equal size. Searching in parallel in several tables allows for finding a perfect match in a fragment of the permutated fingerprint and, thereby, narrowing down search space for comparing the entire fingerprint. Our approach follows a very similar motivation, but is optimized towards the specific shape of our template fingerprints.

3. TEMPLATE FINGERPRINTS FOR WEB DOCUMENTS

The template fingerprints we present in this paper are based on the structure of a document and computed from a compression schema over the document's structure which is then encoded as a hash value. The fingerprints form a locality sensitive hash function, i.e. minor variations in a document's structure are reflected in very small variations of the hash value. Thus, we can identify structural similarity of web documents via observing high similarity of the hash values. In the following, we will describe the process of constructing template fingerprints along with an illustrative example. We also present an index structure which allows for fast retrieval of documents being structurally similar to a given query document.

3.1 Computation of Template Fingerprints

At the core of our template fingerprints lies the computation of a compression schema for the structure of web documents. This means, for a given document we derive a compression pattern encoding the redundancies in the document's structure. Our hypothesis is, that structurally similar documents based on the same template exhibit consistent redundancies. Thus, the compression schemata of structurally similar documents are highly similar themselves. A second hypothesis is, that the initial segment of template based web documents is structurally more stable. This segment typically comprises the HTML `head` element and the initial part of the `body` element. Together they yield a reasonable fragment of the template layout covering meta data, the outer document structure as well as characteristic design elements located at the top of a document. The structure of the "middle" part of a template based document, instead, is more noisy and filled with contents which are potentially weakly structured themselves. Thus, this part is less suitable to capture relevant template information.

Based on these two hypothesis we design our template fingerprints by applying *Lempel-Ziv-Welch* (LZW) compression [20] to the initial sequence of tokens from a document's tag sequence. The LZW algorithm offers an efficient on-the-fly compression of a token sequence. Prior to applying the algorithm we pre-process the HTML documents to discard all information which is not related to document structure. To this end, text and comment elements are removed from the document as well as any attributes from the remaining HTML elements. The remainder of the document is represented as a sequence of tag tokens. As a further generalization, we do not differentiate between start and end tags, but represent the tags solely by their element name.

Example: We will illustrate the pre-processing and fingerprint computation using the following simple HTML document:

```
<html>
  <body>
  <!-- Example web document -->
    <p><b>This text is bold</b></p>
    <p><strong>This text is strong</strong></p>
    <p><big>This text is big</big></p>
    <p><em>This text is emphasized</em></p>
    <p><i>This text is italic</i></p>
    <p><small>This text is small</small></p>
    <p>This is<sub> subscript</sub> and
      <sup>superscript</sup></p>
  </body>
</html>
```

After removing text nodes and comments the document's representation as a sequence of tag tokens has the following form:

```
html, body, p, b, b, p, p, strong, strong, p, p,
big, big, p, p, em, em, p, p, i, i, p, p, small,
small, p, p, sub, sub, sup, sup, p, body, html
```

The next step in the computation of the template fingerprint is to compress this tag sequence using the LZW algorithm. In the following, we will briefly describe the parts of the algorithm which are relevant in the context of this paper. For details on the correctness of the algorithm, the storage schema and the decompression algorithm we refer to the original publication [20].

The LZW compression algorithm builds up both a dictionary and a compressed sequence during a single pass over the input data. The dictionary interlinked with the resulting compressed sequence can be encoded as a list of consecutive tuples. The first component in a tuple is a reference to another dictionary entry which provides the prefix of a token sequence. The second component in a tuple stores the single token which extends the prefix in a unique way. The general idea of LZW is to linearly read the input data and extend the token sequences currently read as long as it matches an entry in the dictionary. This is achieved by adding a single token to the observation window and probing the dictionary for this token sequence. If the latest added token extends the sequence in a way that is not yet encoded in the dictionary it is used to construct a new tuple as entry in the dictionary. This new entry references the index of the old entry matching the prefix up to the last but one token of the current sequence and the last token as the extension.

Table 1: LZW compression dictionary for the example web document.

Dictionary Index	Reference to Prefix	New Token
1	–	`html`
2	–	`body`
3	–	`p`
4	–	`b`
5	4 (`b`)	`p`
6	3 (`p`)	`strong`
7	–	`strong`
8	3 (`p`)	`p`
9	–	`big`
10	9 (`big`)	`p`
11	3 (`p`)	`em`
12	–	`em`
13	8 (`p p`)	`i`
14	–	`i`
15	8 (`p p`)	`small`
16	–	`small`
17	8 (`p p`)	`sub`
18	–	`sub`
19	–	`sup`
20	19 (`sup`)	`p`
21	2 (`body`)	`html`

If no prefix is available (because no part of the token sequence has been observed so far), the reference part of the entry remains empty. Then LZW continues to read the next tokens in the data sequence starting again with an empty observation window.

For compressing the structure of a web document, we use the tag tokens as alphabet. Continuing the above example results in the dictionary depicted in Table 1. For deriving the actual fingerprint we only make use of the prefix references in this dictionary, as they reflect the patterns of redundancy in a document we are interested in. Each reference is translated into a single character (byte) in the fingerprint. The dictionary entries without a prefix reference can be represented by a value of 0. In this way we obtain the following fingerprint for our example document:

```
0,0,0,0,4,3,0,3,0,9,3,0,8,0,8,0,8,0,0,19,2
```

To restrain the fingerprint computation to the stable initial segment of a web document, we stop the computation of the LZW algorithm as soon as the dictionary contains 25 entries. This value was determined empirically on a small set of 100 documents based on ten different templates. On this small set of documents, a length of 25 entries for the fingerprint has proven to be sufficient to identify the stable part of structurally similar web documents. On this sample we also observed that the 25 entries in the dictionary practically always included the HTML `head` element and some parts of the beginning `body` fragment. The complete process for the computation of template fingerprints is described in algorithm 1.

The fingerprints obtained in this way satisfy our need for a locality sensitive hash function. First of all, they mainly encode the structurally stable part of template based web documents. Thus, structural differences that occur later on in the more diversified part of the document do not affect the fingerprint. Second, small changes at the beginning of the tag sequence cause only minor changes in the compression pattern indicated by the prefix references. Exchanging, adding or removing single elements in the document structure affects the fingerprint only little (e.g. removing the `sub` element in our example) or not at all (e.g. when replacing the `i` element in our example with a `strong` element).

Algorithm 1: Computation of template fingerprint FP.

Input: Web document d
Output: The fingerprint FP; a sequence of 25 byte characters
begin
  $tokenSeq = \mathsf{tokenizeWebDocument}(d)$;
  $dict = \mathsf{new\ Dictionary}()$;
  $fp = \varepsilon$, $buffer = \varepsilon$, $dictEntryId = 0$;
  while $dictEntryId < 25$ **do**
    $token = tokenSeq.\mathsf{next}()$;
    if $dict.\mathsf{containsPrefix}(\mathsf{concat}(buffer, token))$ **then**
      $buffer = \mathsf{concat}(buffer, token)$;
    else
      $refPrefixId = dict.\mathsf{refId}(buffer)$;
      $fp = \mathsf{concat}(fp, \mathsf{byte}(refPrefixId))$;
      $dict.\mathsf{put}(dictEntryId, (refPrefixId, token))$;
      $dictEntryId = dictEntryId + 1$;
      $buffer = \varepsilon$;
  return fp;
end

3.2 Detecting Structurally Similar Documents

As mentioned above, small variations in the document structure may cause changes in the template fingerprint. Thus, we need to define a function which can cope with these small changes and provides a decision, whether or not two documents are structurally similar. On our small sample of 100 documents we observed, that the fingerprints of document pairs originating from the same template hardly ever vary by more than one character. Thus, we derive the following rule for the decision of the SAME TEMPLATE task:

DEFINITION 3.1 (SAME TEMPLATE). *Two web documents d_1 and d_2 are considered to be structurally similar, iff. $\mathsf{sed}(\mathsf{FP}(d_1), \mathsf{FP}(d_2)) \leq 1$, where sed defines the Levenshtein string edit distance [11], and FP is the function computing the template fingerprint of a document.*

Note, that checking two sequences of equal length for having a Levenshtein distance of at most one can be computed in linear runtime w.r.t. the length of the sequences. Given that we use sequences of fixed length, the overall runtime is effectively constant.

3.3 Index for Efficient Retrieval of Structurally Similar Documents

The template fingerprints and the definition of the function for identifying structurally similar documents (i.e. documents based on the same template) can be used for pairwise comparisons of documents. In order to operate on large collections of web documents we now introduce an efficient index structure which is able to retrieve documents that are structurally similar (according to definition 3.1) to a given query document.

The essential idea of the index structure is to decompose a template fingerprint fp into two parts fp_{prefix} and fp_{suffix}. This split is performed always at the same position l of the fingerprint. We now formulate a simple lemma which is fundamental to the operating mode of the index structure and the retrieval algorithms:

LEMMA 3.1. *Let d_a and d_b be two documents and fp^a and fp^b their respective template fingerprints. Furthermore, let fp^a_{prefix} correspond to the first l characters of the fingerprint of fp^a and fp^a_{suffix}*

to the remaining characters, so that $fp^a = \mathsf{concat}(fp^a_{prefix}, fp^a_{suffix})$. *Equivalently,* fp^b_{prefix} *and* fp^b_{suffix} *are the decomposition of* fp^b.

If fp^a *and* fp^b *have a Levenshtein distance of at most one (i.e. are structurally similar according to definition 3.1), then at least one of the following two cases is valid: (a)* $fp^a_{prefix} = fp^b_{prefix}$ *or (b)* $fp^a_{suffix} = fp^b_{suffix}$.

Effectively, the lemma states that for two structurally similar documents the one change that is tolerated to occur in the characters of the respective template fingerprints has to occur either among the first l characters (i.e. in fp_{prefix}) or in the characters following afterwards (in fp_{suffix}).

This lemma motivates the index structure for efficient retrieval of structurally similar documents. For each web document to be indexed by this data structure we split up its template fingerprint into two parts fp_{prefix} and fp_{suffix}. We then build two hash table structures h_{prefix} and h_{suffix} which store the respective fingerprint parts of all indexed documents. Thus, in h_{prefix} we use the first part fp_{prefix} as key value and store the second part fp_{suffix} as entry together with a reference to the full document. Vice versa we use fp_{suffix} as key value in h_{suffix} and store fp_{prefix} and a reference to the document as entry. Thereby, the entries in both hash tables actually represent a list of documents sharing the same specific part fp_{prefix} or fp_{suffix}.

The process of retrieving documents that are all structurally similar to a query document d_q is shown in algorithm 2 and illustrated in Figure 1. The algorithm starts by computing the template fingerprint fp^q for document d_q and decomposing fp^q into the two parts fp^q_{prefix} and fp^q_{suffix} based on the split parameter l. We start the retrieval by looking for matches in hash table h_{prefix}. The obtained matches provide a list of documents $list_1$ including their full fingerprint where each fingerprint's first part exactly matches fp^q_{prefix}. In this way we can massively narrow down the search space to a few candidate documents, for which we iteratively (**foreach** loop) compare the second part of the fingerprint w.r.t. a Levenshtein distance of at most one. If this criterion is satisfied, we add the document to the result set. The same process is performed for the second part fp^q_{suffix} of the fingerprint of the query document. In this way we can identify all structurally similar documents according to definition 3.1. Moreover, if there is no matching entry for $fp_{q,prefix}$ in h_{prefix} and no match for $fp_{q,suffix}$ in h_{suffix} we can immediately conclude that there is no structurally similar document in the index. In summary, the data structure described gives us the advantage of leveraging a lookup of candidate documents in constant time. The candidate documents are typically very few (especially compared to the number of all indexed documents) and comprise mainly those documents, that are in fact structurally similar.

3.4 Tuning the Split Parameter

The parameter l can be used to smooth imbalances in the distribution of the characters at the different positions in a fingerprint. Previous work [12] considered the values of the entries in a fingerprint to be randomly distributed. While for many bit vector based fingerprints this might be a valid approximation, it certainly does not hold for our template fingerprints. Given that our fingerprints are based on the prefix references in the LZW dictionary, the first fingerprint characters show generally only little variation. Accordingly a higher value of l can be used to reduce the number of hash collisions for fp_{prefix} that occur by chance and, thereby, improve runtime performance[4]. However, setting l too high will increase the risk of hash collisions for fp_{suffix}, simply because this subsequence becomes too short to contain much information.

[4]Note, that the accuracy of the method is not affected by this parameter.

Algorithm 2: Retrieval of structurally similar documents.

Input: Query document d_q, index structure consisting of the two hash tables h_{prefix} and h_{suffix}
Output: Set *result* of documents structurally similar to d_q
begin
  $fp^q = \mathsf{FP}(d_q)$;
  $(fp^q_{prefix}, fp^q_{suffix}) = \mathsf{split}(fp^q, l)$;
  $result = \emptyset$;
  // check for matches of fp^q_{prefix}
  $list_1 = h_{prefix}.\mathsf{get}(fp^q_{prefix})$;
  foreach $x \in list_1$ **do**
    if $sed(fp^q_{suffix}, x.getFp_{suffix}) \leq 1$ **then**
      $result.\mathsf{add}(x.\mathsf{getDocument})$;
  // check for matches of fp^q_{suffix}
  $list_2 = h_{suffix}.\mathsf{get}(fp^q_{suffix})$;
  foreach $y \in list_2$ **do**
    if $sed(fp^q_{prefix}, y.getFp_{prefix}) \leq 1$ **then**
      $result.\mathsf{add}(y.\mathsf{getDocument})$;
end

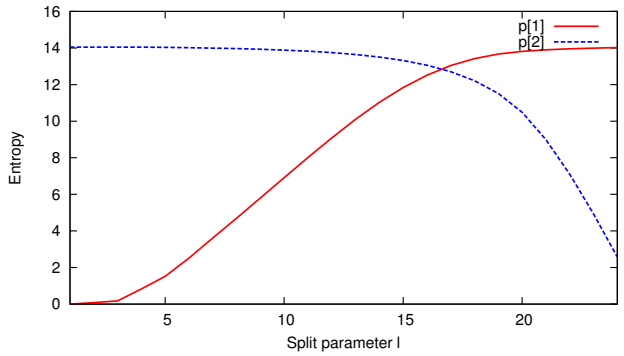

Figure 2: Entropy of fp_{prefix} and fp_{suffix} depending on the choice of the split parameter l.

In order to find a good setting for l we computed the fingerprints for a random sample of documents and compared the entropy of the fragments fp_{prefix} and fp_{suffix} depending on different values for the split parameter l. The result of this analysis is shown in Figure 2. We observed a break-even point at a value of $l = 17$, indicating an empirical optimum. This means that when setting the split parameter to this value we can expect the distribution of the values of the two fragments to be comparable and provide equally much information about a document's template.

4. STRUCTURAL CLASSIFICATION

In the following we describe the experiments and complexity estimation for the task of *structural classification*. In this task we operate on a collection of documents and a known set of templates. Each template is represented by a set of training documents based on this template. The training documents are not part of the document collection under investigation. The task of template classification is to assign each of the documents in the collection to the correct template or alternatively to conclude that it is not based on any of the known templates.

More formally, we deal with a set T of templates $t_1, \ldots, t_k$ which are given implicitly by sets of training documents D_i each.

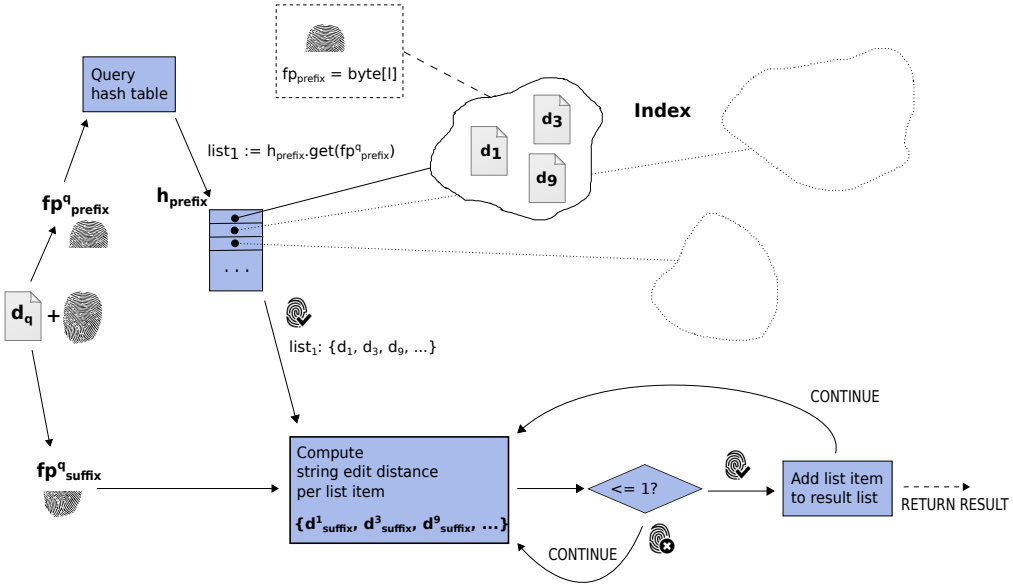

Figure 1: A query document d_q is matched against the indexed documents using hash tables h_{prefix} and h_{suffix}. Entries in the tables are associated with candidate documents and refer to the respective part of the fingerprint being equal among these documents. For the sake of clarity, only the computation with fp^q_{prefix} and $list_1$ is depicted here (the process for fp^q_{suffix} is equivalent).

That is, all documents $d_j \in D_i$ are based on template t_i. Furthermore, we have a collection D of web documents with $D \cap D_i = \emptyset$, $\forall i$. We are now looking for a function $f_T : D \rightarrow \{1, \ldots, k, none\}$ which assigns any given document $d \in D$ to the template t_i it is based on. A result of *none* means that a document is not based on any of the templates in T. This corresponds to a structural classification task with k+1 classes in which the last class is defined as an "other" class (i.e. not based on any known template).

4.1 Using Template Fingerprints for Structural Classification

We implemented an instance based template classifier using our template fingerprint method and the index structure for efficient retrieval of structurally similar documents. This means that we collect the fingerprints of all training documents but do not use any further method to perform feature generalization. Instead, we map all fingerprints directly to the template class they have been observed with. For the classification task it is then sufficient to compare a given document d to the fingerprints of the training documents. Note, that since we use the efficient index for template retrieval as described in Section 3.3 we do not have to compare the new document to all training documents. It is also sufficient to retrieve one structurally similar document in order to assign the given document d to the correct class.

By using this approach in the retrieval lookup, we can decide in constant runtime for a given document whether it falls into the class of one particular template. In conclusion, we have an overall complexity of $O(n)$ for the entire classification task, where n is the total number of documents to be classified.

4.2 Baseline Algorithms

As baseline in the structural classification task we use the methods discussed in Section 2. The baseline covers approaches based on DOM tree edit distance (RTDM) [14], tag sequence alignment (LCTS) [7], tag feature vectors (TV) [5], overlap of common DOM tree paths (CP) [3] or common path shingles (CPS) [3] as well as common tag sequence shingles(CTSS) [7]. Each of these approaches provides a similarity metric to structurally compare two documents. We implemented structural classifiers which assigned a document d to a template classes t_i if the similarity between d and any of the training examples $d_j \in D_i$ was above a threshold θ_i. For RTDM, TV, CP and CPS we determined this threshold by taking the minimal similarity observed among the training documents, i.e. $\theta_i = \min_{d_a, d_b \in D_i}(\mathsf{sim}(d_a, d_b))$. For CTSS, instead, we used a threshold of 0.8 as given in [7]. For the baseline algorithms, this leads to an overall complexity of $O(m \cdot n)$ for the classification task, where $m \geq k$ is the total number of training documents and n is again the number of documents to be classified.

In order to investigate how well our template fingerprints comply to the property of being locality sensitive we also implemented a non-locality sensitive hash based classifier. To this end, we use the MD5 hash function [15] to encode the tag sequence of documents: once the entire tag sequence (MD5) and once only the HTML header (MDH5). The latter approach is motivated again by our hypothesis, that the initial segment of a document is structurally more stable. For both approaches, MD5 and MDH5, two documents are then considered to be structurally similar if they have an identical MD5 hash value. In total this gives us the eight baseline methods listed in Table 2.

4.3 Experiments

The task we address is a typical classification task. Thus, we need a set of training and test documents for which the gold standard is known. We assembled such a training set from publicly available web documents. To this end, we manually collected a total of 13,237 template-based documents from 50 different websites and domains. The documents were taken from specific subsections of the websites and were inspected visually to be based on the same template. Table 3 gives an overview of this *50 templates data set* regarding the original website, the template domain the documents were taken from and the number of documents. The performance of all approaches was evaluated using established metrics such as

Table 2: Overview of compared approaches.

Approach	Method	Data model	Reference
FP	Fingerprints	Tag sequence	(this paper)
RTDM	Alignment	DOM tree	[14, 19]
LTCS	Alignment	Tag sequence	[7]
TV	Feature vector	Tags	[5]
CP	Feature set	DOM paths	[3]
CPS	Shingling	DOM paths	[3]
CTSS	Shingling	Tag sequence	[7]
MD5	Hashing	Tag sequence	(this paper)
MDH5	Hashing	Header tag sequence	(this paper)

Table 3: Characteristics of our 50 templates data set.

Website	Template	Documents
apple.com	bags & cases	218
bbc.co.uk	world news articles	402
bundestag.de	representatives	648
cnet.com	reviews laptops	106
cyberport.de	camcorder & mobile phones	70
datev.de	corporate portal	216
deliciousdays.com	recipes	102
dlr.de	corporate news	1,048
dm.de	drug store products	660
dradio.de	news articles	71
electricliterature.com	blog articles	100
europarl.europa.eu	EP representatives	552
fbi.gov	terrorists	30
fifa.com	worldcup teams	97
garden.com	pest library	77
govisithawaii.com	blog articles	108
guardian.co.uk	news articles (issue 07/08/12)	123
heise.de/mobil	mobile phone specs	106
huffingtonpost.com	news articles (issue 26/07/12)	85
ibm.com/developerworks	developer wiki	120
ikea.com	candle products	99
imdb.com	top 250 movies	250
koblenz.de	sport clubs	79
lego.com	corporate news articles	120
mattel.com	online games	174
msn.de	news articles politics & economy	159
nba.com	player profiles	509
neckermann.de	adidas clothing products	849
notebooksbilliger.de	ultrabooks	82
nytimes.com	news articles (issue 08/08/12)	116
openarchive.icomos.org	publications urbanism	82
politics.co.uk	news articles	1,406
q-cells.com	products	587
reichelt.de	electronics products	135
sap.com	corporate news articles	99
semanticweb.com	articles	111
shopping.com	mp3 player products	114
skoda.de	cars	195
spiegel.de	news articles	103
stanleygibbons.com	chinese stamps	1,037
sueddeutsche.de	olympic news	92
tagesschau.de	news articles	136
telekom.de	business customer products	100
tomshardware.com	reviews hardware	103
vestas.com	corporate news	733
vindor.de	red vine products	133
walmart.com	top 200 books	200
web.de	news articles	90
zalando.de	loose fit jeans products	108
zeit.de	news articles economy	298

accuracy, precision, recall and F1. As evaluation methodology we used a standard ten-fold cross validation procedure.

Given that the runtime of all baseline methods depends on the number of training documents we first investigated their runtime requirements in practice, as this affects their ability to handle a ten-fold cross validation on a data set of this size. Therefore, we randomly sampled ten documents for each of the 50 templates, trained a classifier for each of the methods and used it as a binary classifier on the remaining documents. Table 4 shows the overall time needed for each approach to perform this task as well as the average number of documents processed per second[5]. As described in previous papers and observed in practice as well as under the theoretic analysis it turns out that DOM tree alignment (RTDM) and tag sequence alignment (LTCS) are too expensive to be carried out in a full ten-fold cross validation. This can be explained by the quadratic complexity of the alignment computation for each document pair. Scaling up the time requirements from the single run using ten training documents to the full setup we estimated, that RTDM would have needed approximately 334 days and LTCS approximately 30 days to complete the evaluation. Since their quality has already been demonstrated to be inferior to methods such as CP, CPS and CTSS[6] we discarded these methods from further evaluation.

Table 4: Runtime assessment of all evaluated classification approaches on the 50 templates data set using ten training documents for each template.

Approach	Docs/sec	Overall time
FP	594.90	18.53 min
RTDM	0.61	18,206.94 min
LTCS	6.74	1,636.24 min
TV	261.57	42.14 min
CP	46.46	237.26 min
CPS	37.37	295.02 min
CTSS	419.68	26.27 min
MD5	611.18	18.04 min
MDH5	8,643.51	1.28 min

With the remaining approaches we then performed the full ten-fold cross validation on 13,237 documents and 50 template classes. The results are listed in Table 5. As can be seen our template fingerprint approach (FP) performs very well compared to all other methods. FP has a perfect precision, i.e. all documents assigned to a class originate from the appropriate template. Also CTSS and MDH5 achieve a precision equal to one, all other methods perform lower under this metric. FP scores a high recall as well, though, CTSS achieves a slightly better result among the methods with perfect precision. TV, CP, CPS and MD5 all perform much lower w.r.t precision or recall. This is also reflected in the values for the F1 metrics. Overall, we can identify CTSS, FP and MDH5 as the top ranking approaches under F1.

In order to investigate the differences between these three approaches we applied a significance test regarding the metrics recall and F1.We used a standard two-tailed, paired Student's t-test with $p = 0.05$. We observed that the difference in recall is significant between MDH5 and FP as well as for MDH5 and CTSS. The same applies to the F1 metric. Thus, we can state that MDH5 is not of the same quality as the other two methods. The difference between FP and CTSS, instead, is not statistically significant.

Considering runtime, though, MDH5 is by far the fastest method. This is not surprising, since MDH5 considers only the header of a web document. In particular, this is a tremendous advantage with

[5]All experiments were run on the same Ubuntu Linux Server 10.04 machine (Intel Core 2 Duo T7700 @2,40GHz, 16 GB RAM).

[6]The lower quality of the approaches was also confirmed on the classification setup with ten training documents we used to evaluate the runtime requirements.

Table 5: Qualitative results for the classification task.

Approach	Prec.	Recall	F1	Accuracy	Docs/sec
FP	1.0000	0.9877	0.9930	1.0000*	488.18
TV	0.1868	0.9996	0.1913	0.4593	14.94
CP	0.7809	0.9979	0.7836	0.8369	2.78
CPS	0.7147	0.9986	0.7147	0.8242	1.95
CTSS	1.0000	0.9912	0.9950	1.0000*	141.00
MD5	0.5760	0.2250	0.2786	0.9987	454.03
MDH5	1.0000	0.9729	0.9831	1.0000*	6,438.51

* results rounded

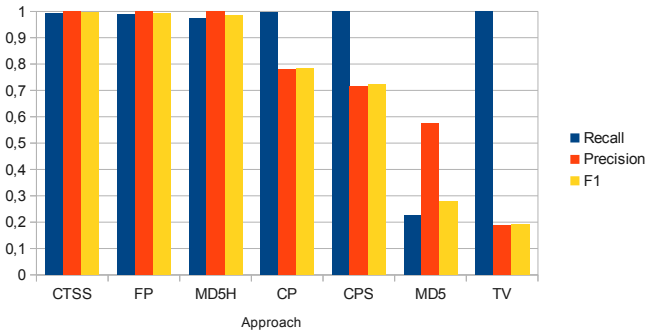

Figure 3: Approaches and metrics from the cross validation.

long web documents, as MDH5 does not require to parse the full tag sequence. This is exactly the reason, why MD5 and FP have a slower performance. They parse the complete document in order to compute the hash values. CTSS is notably slower by a factor of three. TV falls behind CTSS by an order of magnitude, the path-based methods CP and CPS even by two orders of magnitude.

In conclusion, we can state that FP is of comparable quality with regards to the best performing structural classification algorithm CTSS. However, FP is significantly faster than CTSS. As it is additionally independent of the number of training documents, FP is conceptually more suitable in particular for settings with many training documents.

5. STRUCTURAL CLUSTERING

Clustering web documents differs from the classification task we addressed in the previous section. Given a collection of documents based on an unknown number of templates the task is to form groups of structurally similar documents, i.e. cluster all documents together that are based on the same template. Formally, we are given a collection D of web documents. These documents have to be grouped in subsets $D_i \subset D$ for which all of the documents $d_j \in D_i$ are structurally similar.

5.1 Using Template Fingerprints for Structural Clustering

Ideally, we want to achieve a linear approach for structural clustering in which each document in D needs to be considered only once. During this single pass over the document set we want to iteratively build the clusters. For each document we encounter, we have to decide whether it belongs to an already existing cluster or whether it is not similar to any previously seen document and, thus, constitutes its own novel cluster.

Similar to the classification task our clustering algorithm is a truly iterative process which allows for successive assignment of new web documents to clusters. We utilize again the index for template retrieval (cf. Section 3.3) for the decision if a document can be assigned to an existing cluster or if a new cluster with this document as first member has to be created. Please note, that our algorithm is independent from the sequence in which the documents are observed and will provide the same result for arbitrary permutations of the document sequence.

The assignment of a document d to a cluster is based on a lookup in the index structure. Splitting up again the fingerprint of d into two parts fp_{prefix} and fp_{suffix}, we can efficiently retrieve all structurally similar documents from the index and, thereby, also the clusters they belong to. There are three possible cases: (a) we find no clusters and create a new cluster for d, (b) we find exactly one cluster c and assign d to this cluster, or (c) we find two or more clusters c_i which all contain documents that are structurally similar to d. The first two cases can be handled straight forward adding the document as new entry in our index structure or by extending an existing entry. The last case can occur when the considered document has a Levenshtein distance of one to more than one document each belonging to a different cluster. In this case document d forms a link between the identified clusters and so we merge the retrieved c_i into one cluster[7].

Again, given the constant runtime complexity for the lookup of candidate documents and clusters in the index, we have an overall linear runtime of $O(C \cdot n)$, where C is a constant related to the total number of clusters with a complying fingerprint regarding each document and n the total number of documents to be clustered. More precisely, C covers the two possible hash table lookups plus a loop iteration over a small set of documents in case there is a mismatch for at least one of the fingerprint parts. The actual value of the constant depends on the number of templates in the document collection but as we will see in Section 5.2 from our experiments we can empirically conclude that clustering has linear runtime in scenarios of web quality data.

5.2 Experiments

For our clustering experiments we used two different experimental setups. First, we employed again the 50 template data set from the classification task consisting of 13,237 web documents. As this data set provides a gold standard we can use it to evaluate cluster quality. However, it is too small to evaluate the scalability of our approach. To evaluate scalability, we looked for a large-scale, representative set of web documents. A good source is the Open Directory (DMOZ) which provides a list of URLs referencing more than 3.6 million web documents. We retrieved all these documents directly from the web in order to construct our *DMOZ data set*.

To evaluate cluster quality we used again MDH5 as baseline. By forming groups of all documents with the same hash value, it is the only algorithm capable of performing structural clustering in linear runtime. Furthermore, MDH5 demonstrated an acceptable accuracy in the classification task. We applied well-established metrics: Rand Index (RI) [13], Adjusted Rand Index (ARI) [13] and average cluster Purity[8]. As gold standard we used the known templates in our 50 templates data sets as reference clusters. The results of this experiment are shown in Table 6.

The purity of the obtained clusters is perfect for both approaches (i.e. each clusters contains only documents based on the same

[7]This causes chaining effects, which have proven to provide good results in the context of structural clustering [7].

[8]We also considered and computed Mutual Information, but obtained highly similar values which did not provide useful insights.

Table 6: Qualitative results for the clustering task on the 50 templates data set.

Approach	Clusters	RI	ARI	Purity	Docs/sec
FP	274	0.9866	0.8218	1.0	487.30
MDH5	410	0.9844	0.7862	1.0	4,263.12

Table 7: Statistics for clustering the DMOZ collection using the FP approach.

Documents	Clusters	Overall Runtime	Docs/sec
3,657,105	2,604,653	50.91 min	1,197.32

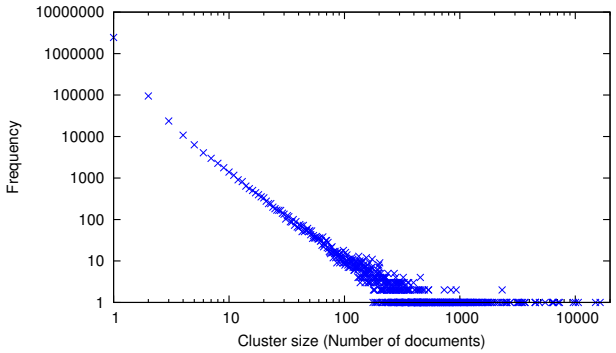

Figure 4: Clusters size distribution on the DMOZ data set.

template according to the reference cluster in the gold standard). However, the template fingerprint approach dominates the MDH5 method in the other quality metrics RI and ARI. FP leads to a lower number of clusters in comparison with MDH5. Given the still perfect purity of the clusters, this means, that FP manages better to group together documents which are structurally similar but not equal. In fact, many of the clusters align very well with the gold standard. Only a few outlier documents form own singleton clusters. This is also reflected by the relatively high adjusted Rand Index of 0.8218. In conclusion, we have reached not a perfect but a qualitatively good structural clustering of the documents. The processing speed of FP is the same as for the classification task—demonstrating the same linear runtime.

In our second experiment, we additionally measured runtime over the large DMOZ data set to empirically underline the statement on linear runtime complexity for the clustering task using our fingerprint approach. For this reason, we used the DMOZ data set with more than 3.6 million documents as input to our fingerprint clustering method. The statistics are given in Table 7.

For the larger DMOZ data set we obtained approximately 2.6 million clusters. This number seems reasonable given that the Open Directory typically lists individual entry points of websites which are not structurally similar. In fact, about 2.4 million clusters contain only one single document. The overall distribution of the cluster size seems to follow a power law, as indicated in Figure 4. The largest cluster contained 16,411 documents which after visual inspection of a random sample seemed to consist of documents from Wikipedia or other Mediawiki installations.

Regarding the processing performance, we can observe a high value of 1,197 docs/sec, so even above the 487 docs/sec in the previous experiment on the 50 templates data set. This can be explained by the different average size of the documents in DMOZ

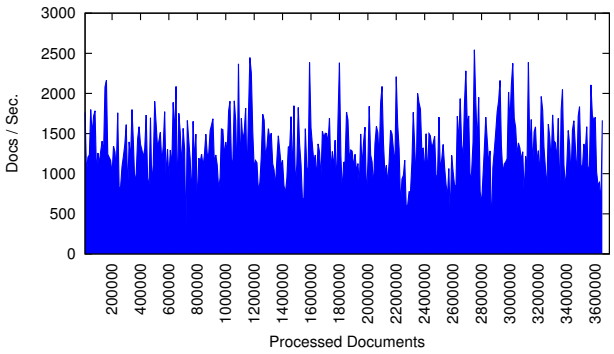

Figure 5: Data throughput in documents per second over the number of processed documents in the DMOZ data set.

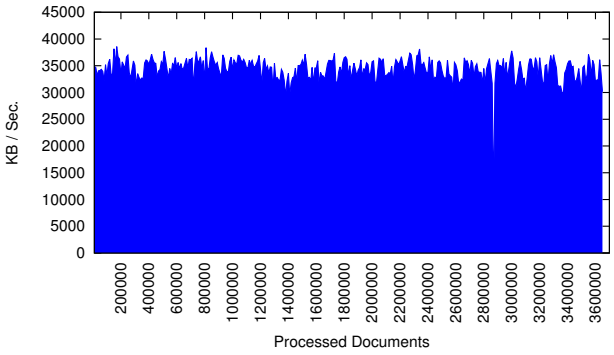

Figure 6: Data throughput in KB per second over the number of processed documents in the DMOZ data set.

and the 50 template data set. The average size of documents in the latter collection is 73.29 KB compared to 28.44 KB in DMOZ. As already mentioned, the FP approach in its current implementation tokenizes the entire documents into a tag sequence, prior to the application of LZW compression algorithm. Thus, the average reduction of the input data by a factor of approximately 2.58 leads to a speedup at nearly the same scale (2.46).

Figures 5 and 6 show the runtime performance in processed documents per second and processed kilobytes per second over the number of documents processed[9]. Some minor variation in the performance can be observed mainly in the throughput of processed documents. These fluctuations can be attributed to the varying size of documents. Regarding the processed data volume in Figure 6, the performance is very stable and—most important—constant over the entire data set. This observation empirically confirms our theoretic considerations of the linear runtime complexity regardless of the number of clusters. Also the overall performance is very satisfactory. Clustering the entire DMOZ data set was completed in less than 51 minutes (excluding overhead caused by I/O access).

In conclusion, we can state that the template fingerprints in combination with the efficient index structure for structurally similar documents provide a scalable and high quality algorithm for structural clustering. Given its linear runtime over the number of analysed web documents it is suitable even for web-scale applications.

[9]For the sake of clarity in the illustration we aggregated numbers to bins of 10.000 documents

6. CONCLUSION

In this paper we presented a novel method for structural classification and clustering of web documents based on a template fingerprint technique. The fingerprints represent a locality sensitive hash function which is based on a compression schema computed from the initial segment of a web document's tag sequence. Our implementations of classification and clustering approaches use an efficient index structure for retrieving structurally similar web documents from a previously analysed and indexed collection. This data structure leverages the template fingerprints to avoid the costly pairwise comparisons of all considered documents. We evaluated our fingerprint method for the task of structural classification on a gold standard data set of more than 13,000 web documents. The approach proved to achieve perfect precision and high recall values and is competitive with state-of-the-art algorithms. At the same time its runtime complexity is independent of the size of the training set. In the evaluation setting we observed a speedup by a factor of three. Also in the context of structural clustering we evaluated the quality and the runtime performance of our approach. We obtained a perfect purity in the clusters and a high adjusted Rand Index of 0.82 with respect to a ground truth of template classes. Finally, we demonstrated that our method scales linearly regardless of the actual number of clusters and only depends on the amount of document data to be processed. We can therefore conclude our fingerprint method FP to be suitable for web-scale structural classification and clustering tasks.

Future work will cover optimizations of the fingerprint computation. In our current implementation we tokenize documents entirely into token sequences regardless of how much of this data is actually required for the computation of the fingerprint. We estimate that coupling the tokenisation procedure closely to the compression algorithm can improve the runtime performance by a factor between five to ten. Furthermore, considering only a subset of structurally relevant tags might improve robustness and runtime performance.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 257859, ROBUST.

7. REFERENCES

[1] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *WWW '02: Proceedings of the 11th International Conference on World Wide Web*, pages 580–591. ACM Press, 2002.

[2] Lorenzo Blanco, Nilesh Dalvi, and Ashwin Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 437–446. ACM, 2011.

[3] David Buttler. A short survey of document structure similarity algorithms. In *IC '04: Proceedings of the International Conference on Internet Computing*, pages 3–9. CSREA Press, 2004.

[4] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. Page-level template detection via isotonic smoothing. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 61–70. ACM Press, 2007.

[5] Isabel F. Cruz, Slava Borisov, Michael A. Marks, and Timothy R. Webbs. Measuring structural similarity among web documents: preliminary results. In *EP '98: Proceedings of the 7th Int. Conference on Electronic Publishing, Artistic Imaging, and Digital Typography*, pages 513–524, 1998.

[6] Thomas Gottron. Bridging the Gap: From Multi Document Template Detection to Single Document Content Extraction. In *EuroIMSA '08: Proceedings of the IASTED Conference on Internet and Multimedia Systems and Applications 2008*, pages 66–71, 2008.

[7] Thomas Gottron. Clustering template based web documents. In *ECIR '08: Proceedings of the 30th European Conference on Information Retrieval*, pages 40–51. Springer, 2008.

[8] Thomas Gottron. Detecting website redesigns via template similarity on streams of documents. In *ITA'09: Proceedings of the 3rd International Conference on Internet Technologies and Applications*, pages 35–43, 2009.

[9] Thomas Gottron and Roman Schneider. A hybrid approach to statistical and semantical analysis of web documents. In *EuroIMSA'09: Proceedings of 5th European Conference on Internet and Multimedia Systems and Applications*, pages 115–120, 2009.

[10] Sachindra Joshi, Neeraj Agrawal, Raghu Krishnapuram, and Sumit Negi. A bag of paths model for measuring structural similarity in web documents. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 577–582. ACM Press, 2003.

[11] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[12] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 141–150. ACM, 2007.

[13] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[14] D. C. Reis, P. B. Golgher, A. S. da Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, pages 502–511. ACM Press, 2004.

[15] R. Rivest. The md5 message-digest algorithm, 1992.

[16] Lei Shi, Cheng Niu, Ming Zhou, and Jianfeng Gao. A DOM tree alignment model for mining parallel data from the web. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 489–496. Association for Computational Linguistics, 2006.

[17] Benno Stein. Principles of Hash-based Text Retrieval. In *SIGIR'07: Proceedings of the 30th International ACM Conference on Research and Development in Information Retrieval*, pages 527–534. ACM, July 2007.

[18] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 563–570, New York, NY, USA, 2008. ACM.

[19] Karane Vieira, André da Costa Carvalho, Klessius Berlt, Edleno de Moura, Altigran da Silva, and Juliana Freire. On finding templates on web collections. *World Wide Web*, 12:171–211, 2009. 10.1007/s11280-009-0059-3.

[20] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.