# Highly Efficient Algorithms for Structural Clustering of Large Websites

Lorenzo Blanco
Università degli
Studi Roma Tre
Rome, Italy
blanco@dia.uniroma3.it

Nilesh Dalvi
Yahoo! Research
Santa Clara, CA, USA
ndalvi@yahoo-inc.com

Ashwin Machanavajjhala
Yahoo! Research
Santa Clara, CA, USA
mvnak@yahoo-inc.com

## ABSTRACT

In this paper, we present a highly scalable algorithm for structurally clustering webpages for extraction. We show that, using only the URLs of the webpages and simple content features, it is possible to cluster webpages effectively and efficiently. At the heart of our techniques is a principled framework, based on the principles of information theory, that allows us to effectively leverage the URLs, and combine them with content and structural properties. Using an extensive evaluation over several large full websites, we demonstrate the effectiveness of our techniques, at a scale unattainable by previous techniques.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Data Mining

## General Terms

Algorithms

## Keywords

information extraction, structural clustering, minimum description length

## 1. INTRODUCTION

Virtually any website that serves content from a database uses one or more script to generate pages on the site, leading to a site considering of several *clusters* of pages, each generated by the same script. Since a huge number of surface-web and deep-web sites are served from databases, including shopping sites, entertainment sites, academic repositories and library catalogs, these sites are natural targets for information extraction. Structural similarity of pages generated from the same script allows information extraction systems to use simple rules, called *wrappers*, to effectively extract information from these webpages. Wrapper systems are commercially popular, and the subject of extensive research over the last two decades wrappers [2, 3, 6, 10, 17, 18, 20, 19, 24, 25, 26]. While the original goal and an important application of wrapper techniques is the population of structured databases, our research goal goes beyond this to the production of a sophisticated web of linked data, a web of concepts [11]. A key challenge to fulfill this vision is the need to perform web-scale information extraction over domains of interest.

The key difference between wrapper induction and web-scale wrapper induction is the form of the input. For a traditional wrapper induction task, a schema, a set of pages output from a single script, and some training data are given as input, and a wrapper is inferred that recovers data from the pages according to the schema. For web-scale extraction, a large number of *sites* are given as input, with each site comprising the output of an unknown number of scripts, along with a schema. A clear result of the new problem definition for web-scale extraction is that per-site training examples can no longer be given, and recent work on unsupervised extraction seeks to meet this challenge [12, 15, 16, 23].

An equally important, but less-recognized result of the new problem definition is the need to *automatically* organize pages of the site into clusters, such that a single, high-quality wrapper can be induced for each cluster. Conceptually, each cluster corresponds to the output of one of the scripts that created the site. Alternatively, if manual work is done to select which pages to wrap, the benefit of unsupervised extraction techniques is effectively lost, since non-trivial editorial work must still be done per site. (Even though techniques with the limited scope of extracting from lists [16, 23] do not explicitly need such a clustering, the knowledge that many lists on a site have the same structure can substantially improve the extraction accuracy and recall of these techniques.) While substantially less-well-studied than wrapper induction, the resulting problem of *structurally clustering* web pages for extraction, has in fact been studied [7, 8], and summarized in a recent survey [13].

However, at the current state-of-the art, a fundamental issue remains: existing techniques do not scale to large websites. Database-generated websites suitable for extraction routinely have millions of pages, and we want the ability to cluster a large number of such websites in a reasonable amount of time. The techniques covered in a recent survey [13] do not scale beyond few hundred webpages. In fact, most of these techniques based on similarity functions along with agglomerative hierarchical clustering have a quadratic complexity, and cannot handle large sites. The XProj [1] system, which is the state of the art in XML clustering, has a linear complexity; however, it still requires an estimated time of more than 20 hours for a site with a million pages[1].

*WWW 2011*, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0632-4/11/03.

[1] It takes close to 1200 seconds for 16,000 documents from DB1000DTD10MR6 dataset, and the documents themselves are much smaller than a typical webpage.

**Our Contributions** In this work, we develop highly scalable techniques for clustering websites. We primarily rely on URLs, in conjunction with very simple content features, which makes the techniques extremely fast. Our use of URLs for structural clustering is novel. URLs, in most cases, are highly informative, and give lots of information about the contents and types of webpages. Still, in previous work [7], it was observed that using URLs similarity does not lead to an effective clustering. We use URLs in a fundamentally different way. We share the intuition in XProj [1] that pairwise similarity of URLs/documents is not meaningful (we illustrate this in Sec 2.2). Instead, we need to look at them holistically, and look at the patterns that emerge. In this work, we develop a principled framework, based on the principles of information theory, to come up with a set of scripts that provide the *simplest explanation* for the observed set of URLs/content.

Below, we summarize the contributions of our work.

1. We explore the idea of using URLs for structural clustering of websites.

2. We develop a principled framework, grounded in information theory, that allows us to leverage URLs effectively, as well as combine them with content and structural properties.

3. We propose an algorithm, with a linear time complexity in the number of webpages, that scales easily to websites with millions of pages.

4. We perform an extensive evaluation of our techniques over several entire websites spanning four content domains, and demonstrate the effectiveness of our techniques. We believe this is the first experimental evaluation of this kind, as all previous systems have either looked at small synthetic datasets, or a few small sample clusters of pages from websites. We find that, for example, we were able to cluster a web site with 700,000 pages in 26 seconds seconds, an estimated 11,000 times faster than competitive techniques.

## 2. OVERVIEW

In this section, we introduce the clustering problem and give an overview of our information-theoretic formulation. The discussion in this section is informal, which will be made formal in subsequent sections.

### 2.1 Website Clustering Problem

Websites use scripts to publish data from a database. A *script* is a function that takes a relation $R$ of a given schema, and for each tuple in $R$, it generates a webpage, consisting of a (*url*,*html*) pair. A website consists of a collection of scripts, each rendering tuples of a given relation. E.g., the website `imdb.com` has, among others, scripts for rendering *movie*, *actor*, *user*, etc.

In structured information extraction, we are interested in reconstructing the hidden database from published webpages. The inverse function of a script, i.e. a function that maps a webpage into a tuple of a given schema, is often referred to as a *wrapper* in the literature [2, 18, 17, 20, 25, 26]. The target of a wrapper is the set of all webpages generated by a common script. This motivates the following problem:

**Website Clustering Problem** : Given a website, cluster the pages so that the pages generated by the same script are in the same cluster.

The clustering problem as stated above is not yet fully-specified, because we haven't described how scripts generate the urls and contents of webpages. We start from a very simple model focusing on urls.

### 2.2 Using URLs For Clustering

A url tells a lot about the content of the webpage. Analogous to the webpages generated from the same script having similar structure, the urls generated from the same script also have a similar pattern, which can be used to cluster webpages very effectively and efficiently. Unfortunately, simple pairwise similarity measures between urls do not lead to a good clustering. E.g. consider the following urls:

$u_1$ : `site.com/CA/SanFrancisco/eats/id1.html`
$u_2$ : `site.com/WA/Seattle/eats/id2.html`
$u_3$ : `site.com/WA/Seattle/todo/id3.html`
$u_4$ : `site.com/WA/Portland/eats/id4.html`

Suppose the site has two kinds of pages : *eats* pages containing restaurants in each city, and *todo* pages containing activities in each city. There are two "scripts" that generate the two kind of pages. In terms of string similarity, $u_2$ is much closer to $u_3$, an url from a different script, than the url $u_1$ from the same script. Thus, we need to look at the set of urls holistically, and cannot rely on string similarities for clustering.

Going back to the above example, we can use the fact that there are only 2 distinct values in the entire collection in the third position, *todo* and *eats*. They are most like script terms. On the other hand, there are a large number of values for states and cities, so they are most likely data values. We call this expected behavior the *small cardinality* effect.

Data terms and script terms can occur at the same position in the url. E.g., the same site may also have a third kind of pages of the form: `site.com/users/reviews/id.html`. Thus, in the first position we have the script term *users* along with list of states, and in second position we have *reviews* along with cities. However, if one of the terms, e.g *reviews*, occurs with much higher frequency than the other terms in the same position, it is an indication that its a script term. We call this expected behavior the *large component* effect. We note that there are scenarios when a very frequent data item might be indistinguishable from script term according to the large component effect. We show how to disambiguate script terms and data terms in such cases using semantic constraints in Section 5.4.

In order to come up with a principled theory for clustering urls, we take an information theoretic view of the problem. We consider a simple and intuitive encoding of urls generated by scripts, and try to find an *hypothesis* (set of scripts) that offer the simplest explanation of the *observed data* (set of urls). We give an overview of this formulation in the next section. Using an information-theoretic measure also allows us to incorporate addition features of urls, as well as combine them with the structural cues from the content.

## 2.3 An Information-Theoretic Formulation

We assume, in the simplest form, that a url is a sequence of tokens, delimited by the "/" character. A *url pattern* is a sequence of tokens, along with a special token called "$*$". The number of "$*$" is called the *arity* of the url pattern. An example is the following pattern:

`www.2spaghi.it/ristoranti/*/*/*/*`

It is a sequence of 6 tokens: *www.2spaghi.it*, *ristoranti*, $*$, $*$, $*$ and $*$. The arity of the pattern is 4.

**Encoding URLs using scripts**

We assume the following generative model for urls: a script takes a url pattern $p$, a database of tuples of arity equal to $arity(p)$, and for each tuple, generates an url by substituting each $*$ by corresponding tuple attribute. E.g., a tuple $(lazio, rm, roma, baires)$ will generate the url:

`www.2spaghi.it/ristoranti/lazio/rm/roma/baires`

Let $\mathcal{S} = \{S_1, S_2, \cdots S_k\}$ be a set of scripts, where $S_i$ consists of the pair $(p_i, D_i)$, with $p_i$ a url pattern, and $D_i$ a database with same arity as $p_i$. Let $n_i$ denote the number of tuples in $D_i$. Let $U$ denote the union of the set of all urls produced by the scripts. We want to define an encoding of $U$ using $\mathcal{S}$.

We assume for simplicity that each script $S_i$ has a constant cost $c$ and each data value in each $D_i$ has a constant cost $\alpha$. Each url in $U$ is given by a pair $(p_i, t_{ij})$, where $t_{ij}$ is a tuple in database $D_i$. We write all the scripts once, and given a url $(p_i, t_{ij})$, we encode it by specifying just the data $t_{ij}$ and an index to the pattern $p_i$. The length of all the scripts is $c \cdot k$. Total length of specifying all the data equals $\sum_i \alpha \cdot arity(p_i) \cdot n_i$. To encode the pattern indexes, the number of bits we need equals the entropy of the distribution of cluster sizes. Denoting the sum $\sum_i n_i$ by $N$, the entropy is given by $\sum_i n_i log \frac{N}{n_i}$.

Thus, the description length of $U$ using $\mathcal{S}$ is given by

$$ck + \sum_i n_i \log \frac{N}{n_i} + \alpha \sum_i arity(p_i) \cdot n_i \qquad (1)$$

**The MDL Principle**

Given a set of urls $U$, we want to find the set of scripts $\mathcal{S}$ that best explain $U$. Using the principle of *minimum description length* [14], we try to find the shortest hypothesis, i.e. $\mathcal{S}$ that minimize the description length of $U$.

The model presented in this section for urls is simplistic, and serves only to illustrate the *mdl* principle and the cost function given by Eq .(1). In the next section, we define our clustering problem formally and in a more general way.

# 3. PROBLEM DEFINITION

We now formally define the MDL-based clustering problem. Let $W$ be a set of webpages. Each $w \in W$ has a set of terms, denoted by $T(w)$. Note that a url sequence

"site.com/a$_1$/a$_2$/..."

can be represented as a set of terms

$$\{(pos_1 = \texttt{site.com}), (pos_2 = \texttt{a}_1), (pos_3 = \texttt{a}_2), \cdots\}$$

In section 3.1, we will describe in more detail how a url and the webpage content is encoded as terms. Given a term $t$, let $W(t)$ denote the set of webpages that contain $t$. For a set of pages, we use $script(W)$ to denote $\cap_{w \in W} T(w)$, i.e. the set of terms present in all the pages in $W$.

A *clustering* is a partition of $W$. Let $C = \{W_1, \cdots, W_k\}$ be a clustering of $W$, where $W_i$ has size $n_i$. Let $N$ be the size of $W$. Given a $w \in W_i$, let $arity(w) = |T(w) - script(W_i)|$, i.e. $arity(w)$ is the number of terms in $w$ that are not present in all the webpages in $W_i$. Let $c$ and $\alpha$ be two fixed parameters. Define

$$mdl(C) = ck + \sum_i n_i \log \frac{N}{n_i} + \alpha \sum_{w \in W} arity(w) \qquad (2)$$

We define the clustering problem as follows:

PROBLEM 1. (MDL-CLUSTERING) *Given a set of webpages $W$, find the clustering $C$ that minimizes $mdl(C)$.*

In Sec. 4, we formally analyze Eq. (2) and show how it captures some intuitive properties that we expect from URL clustering.

Eq. (2) can be slightly simplified. Given a clustering $C$ as above, let $s_i$ denote the number of terms in $script(W_i)$. Then, $\sum_{w \in W} arity(w) = \sum_{w \in W} |w| - \sum_i n_i s_i$. Also, the entropy $\sum_i n_i \log \frac{N}{n_i}$ equals $N \log N - \sum_i n_i \log n_i$. By removing the clustering independent terms from the resulting expression, the MDL-CLUSTERING can alternatively be formulated using the following objective function:

$$mdl^*(C) = ck - \sum_i n_i \log n_i - \alpha \sum_i n_i s_i \qquad (3)$$

## 3.1 Instantiating Webpages

The abstract problem formulation treats each webpage as a set of terms, which we can use to represent its url and content. We describe here the representation that we use in this work:

**URL Terms**

As we described above, we tokenize urls based on "/" character, and for the token $t$ in position $i$, we add a term $(pos_i = t)$ to the webpage. The sequence information is important in urls, and hence, we add the position to each token.

For script parameters, for each (*param*, *val*) pair, we construct two terms: (*param*, *val*) and (*param*). E.g. the url `site.com/fetch.php?type=1&bid=12` will have the following set of terms: { $pos_1$=`site.com`, $pos_2$=`fetch.php`, *type*, *bid*, *type*=1, *bid*=12}. Adding both (*param*, *val*) and (*param*) for each parameter allows us to model the two cases when the existence of a parameter itself varies between pages from the same script and the case when parameter always exists and its value varies between script pages.

Many sites use urls whose logical structure is not well separated using "/". E.g., the site `tripadvisor.com` has urls like `www.tripadvisor.com/Restaurants-g60878-Seattle_Washington.html` for restaurants and has urls of the form `www.tripadvisor.com/Attractions-g60878-Activities-Seattle_Washington.html` for activities. The only way to separate them is to look for the keyword "*Restaurants*" vs. "*Attractions*". In order to model this, for each token $t$ at position $i$, we further tokenize it based on non-alphanumeric characters, and for each subterm $t_j$, we add $(pos_i = t_j)$ to the webpage. Thus, the restaurant webpage above will be represented as { $pos_1$=*tripadvisor.com*, $pos_2$=*Restaurants*, *pos*$_2$=*g60878*, *pos*$_2$=*Seattle*, *pos*$_2$=*Washington*}. The idea is that the term *pos*$_2$=*Restaurants* will be inferred as part of the script, since its frequency is much larger than other terms in co-occurs with in that position. Also note that we treat the individual subterms in a token as a set rather than sequence, since different urls can have different number of subterms in general, and we don't have a way to perfectly align these sequences.

**Content Terms**

We can also incorporate content naturally in our framework. We can simply put the set of all text elements that occur in a webpage. Note that, analogous to urls, every webpage has some content terms that come from the script, e.g. "*Address:*" and "*Opening hours:*" and some terms that come from the data. By putting all the text elements as webpage terms, we can identify clusters that share script terms, similar to urls. In addition, we want to disambiguate text elements that occur at *structurally different positions* in the document. For this, we also look at the html tag sequence of text elements starting from the root. Thus, the content terms consist of all $(xpath, text)$ pairs present in the webpage.

# 4. PROPERTIES OF MDL CLUSTERING

We analyze some properties of MDL-CLUSTERING here, which helps us gain some insights into its working.

**Local substructure**

Let $opt(W)$ denote the optimal clustering of a set of webpages $W$. Given a clustering problem, we say that the problem exhibits a *local substructure* property, if the following holds : for any subset $S \subseteq opt(W)$, we have $opt(W_S) = S$, where $W_S$ denotes the union of webpages in clusters in $S$.

LEMMA 4.1. MDL-CLUSTERING *has local substructure.*

Local substructure is a very useful property to have. If we know that two sets of pages are not in the same cluster, e.g. different domains, different filetypes etc., we can find the optimal clustering of the two sets independently. We will use this property in our algorithm as well as several of the following results.

**Small Cardinality Effect**

Recall from Sec. 2.2 the *small cardinality* effect. We formally quantify the effect here, and show that MDL-CLUSTERING exhibits this effect. We denote by $W(f)$ the set of webpages in $W$ that contain term $f$.

THEOREM 1. *Let $F$ be a set of terms s.t. $C = \{W(f) \mid f \in F\}$ is a partition of $W$ and $|F| \leq 2^{\alpha-c}$. Then, $mdl(C) \leq mdl(\{W\})$.*

A corollary of the above result is that if a set of urls have less than $2^{\alpha-c}$ distinct values in a given position, it is always better to split them by those values than not split at all. This precisely captures the intuition of the small cardinality effect. For $|W| \gg c$, the minimum cardinality bound in Theorem 1 can be strengthened to $2^{\alpha}$.

**Large Component Effect**

In Sec. 2.2, we also discussed the *large component effect.* Here, we formally quantify this effect for MDL-CLUSTERING.

Given a term $t$, let $frac(t)$ denote the fraction of webpages that have term $t$, and let $C(t)$ denote the clustering $\{W(t), W - W(t)\}$.

THEOREM 2. *There exists a threshold $\tau$, s.t., if $W$ has a term $t$ with $frac(t) > \tau$, then $mdl(C(t)) \leq mdl(\{W\})$.*

For $|W| \gg c$, $\tau$ is the positive root of the equation $\alpha x + x \log x + (1-x)\log(1-x) = 0$. There is no explicit form for $\tau$ as a function of $\alpha$. For $\alpha = 2$, $\tau = 0.5$. Thus, for $\alpha = 2$, if a term appears in more than 0.5 fraction of URLs, it is always better to split the term into a separate component.

For clustering, $\alpha$ plays an important role, since it controls both the small cardinality effect and the large component effect. On the other hand, since the number of clusters in a typical website is much smaller than the number of urls, the parameter $c$ plays a relatively unimportant role, and only serves to prevent very small clusters to be split.

# 5. FINDING OPTIMAL CLUSTERING

In this section, we consider the problem of finding the optimal MDL clustering of a set of webpages. We start by considering a very restricted version of the problem: when each webpage has only 1 term. For this restricted version, we describe a polynomial time algorithm in Sec 5.1. In Sec 5.2, we show that the unrestricted version of MDL-CLUSTERING is NP-hard, and remain hard even when we restrict each webpage to have at most 2 terms. Finally, in Sec 5.3, based on the properties of MDL-CLUSTERING (from Section 4) and the polynomial time algorithm from Sec 5.1, we give an efficient and effective greedy heuristic to tackle the general MDL-CLUSTERING problem.

## 5.1 A Special Case : Single Term Webpages

We consider instances $W$ of MDL-CLUSTERING where each $w \in W$ has only a single term. We will show that we can find the optimal clustering of $W$ efficiently.

LEMMA 5.1. *In* OPT$(W)$*, at most one cluster can have more than one distinct values.*

Thus, we can assume that OPT$(W)$ has the form

$$\{W(t_1), W(t_2), \cdots, W(t_k), W_{rest}\}$$

where $W(t_i)$ is a cluster containing pages having term $t_i$, and $W_{rest}$ is a cluster with all the remaining values.

LEMMA 5.2. *For any term $r$ in any webpage in $W_{rest}$ and any $i \in [1, k]$, $|W(t_i)| \geq |W(r)|$.*

Lemma 5.1 and 5.2 give us an immediate PTIME algorithm for MDL-CLUSTERING. We sort the terms based on their frequencies. For each $i$, we consider the clustering where the top $i$ frequent terms are all in separate clusters, and everything else is in one cluster. Among all such clusterings, we pick the best one.

## 5.2 The General Case : Hardness

In this section, we will show that MDL-CLUSTERING is NP-hard. We will show that the hardness holds even for a very restricted version of the problem: when each webpage $w \in W$ has at most 2 terms.

We use a reduction from the 2-BOUNDED-3-SET-PACKING problem. In 2-BOUNDED-3-SET-PACKING, we are given a 3-uniform hypergraph $H = (V, E)$ with maximum degree 2, i.e. each edge contains 3 vertices and no vertex occurs in more than 2 edges. We want to determine if $H$ has a perfect matching, i.e., a set of vertex-disjoint edges that cover all the vertices of $H$. The problem is known to be NP-complete [4].

We refer an interested reader to Appendix B for further details about the reduction.

## 5.3 The General Case : Greedy Algorithm

**Algorithm 1** RECURSIVEMDLCLUSTERING

```
Input: $W$, a set of urls
Output: A partitioning $C$
$C_{greedy} \leftarrow$ FINDGREEDYCANDIDATE($W$)
if $C_{greedy}$ is not null then
    return $\cup_{W' \in C_{greedy}}$ RECURSIVEMDLCLUSTERING($W'$)
else
    return $\{W\}$
end if
```

In this section, we present our scalable recursive greedy algorithm for clustering webpages. At a high level our algorithm can be describe as follows: we start with all pages in a single cluster. We consider, from a candidate set of refinements, the one that results is the lowest *mdl* score. Then, we look at each cluster in the refinement and apply the greedy algorithm recursively.

The following are the key steps of our algorithm:

- (**Recursive Partitioning**) Using the local substructure property (Lemma 4.1), we show that a recursive implementation is sound.
- (**Candidate Refinements**) We consider a set of candidate refinements, and pick the one with lowest *mdl.* Our search for good candidates is guided by out intuition of *large component* and *small cardinality* properties. We show that our search space is complete for single term web pages, i.e. the recursive algorithm returns the optimal clustering of single term web pages as given in Sec 5.1.
- (**Efficient MDL Computation**) The key to efficiency is our technique that can compute the *mld* scores of all candidate refinements in linear time using a single scan over webpages. To achieve this, we analyze the *functional dependencies* between terms in different clusters.

We give details for each of the key steps below.

**1. Recursive Partitioning**

Let $W$ be a set of input webpages to our clustering algorithm. If we know that there is a partition of $W$ such that pages from different partitions cannot be in same cluster, then we can use the *local substructure* property (Lemma 4.1) to independently cluster each partition. We call any partition a *refinement* of $W$. We consider a set of candidate refinements, chosen from a search space of "*good*" refinements, greedily pick the one that results in the highest immediately reduction in *mdl*, and recursively apply our algorithm to each component of the refinement. We stop when no refinement can lead to a lower *mdl.*

**Algorithm 2** FINDGREEDYCANDIDATE

```
Input: $W$, a set of urls
Output: A greedy partitioning $C$ if mdl cost improves, null otherwise
$T = \cup_{w \in W} T(w) - script(W)$
Set $\mathcal{C} \leftarrow \emptyset$ // set of candidate partitions

// Two-way Greedy Partitions
for $t \in T$ do
    $C_t = \{W(t), W - W(t)\}$, where $W(t) = \{w | t \in T(w)\}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_t\}$
end for

// k-way Greedy Partitions ($k > 2$)
Let $T_s = \{a_1, a_2, \ldots\}$ be an ordering of terms in $T$ such that $a_i$ appears in the most number of urls in $W - \cup_{\ell=1}^{i-1} W(a_\ell)$.
for $2 < k \leq k_max$ do
    $U_i = W_{a_i} - \cup_{\ell=1}^{i-1} W_{a_\ell}$, $W_{rest} = W - \cup_{\ell=1}^{k} W_{a_\ell}$
    $C_k = \{U_1, U_2, \ldots, U_k, W_{rest}\}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup C_k$
end for

// return best partition if mdl improves
$C_{best} \leftarrow \arg\min_{C \in \mathcal{C}} \delta_{mdl}(C)$
if $\delta_{mdl}(C_{best}) > 0$ then
    return $C_{best}$
else
    return null
end if
```

**2. Candidate Refinements**

Our search for good candidate refinements is guided by our intuition of the *large component* and the *small cardinality* properties.

Recall that if a term appears in a large fraction of webpages, we expect it to be in a separate component from the rest of the pages. Based on this, for each term $t$, we consider the refinement $\{W(t), W - W(t)\}$. We consider all terms in our search space, and not just the most frequent term, because a term $t_1$ might be less frequent than $t_2$, but might functionally determine lots of other terms, thus resulting in a lower *mld* and being a better indicative of a cluster.

A greedy strategy that only looks at two-way refinements at each step may fail to discover the small cardinality effect. We illustrate this using a concrete scenario. Suppose we have $3n$ webpages in $W$, $n$ of which have exactly one term $t_1$, $n$ others have $t_2$ and the final $n$ have a single term $t_3$. Then,

$$mdl^*(\{W\}) = c - 3n\log(3n) - \alpha \cdot 0$$

since a single cluster has no script terms. Any two-way refinement has cost

$$mdl^*(\{W(a_i), W - W(a_i)\}) = 2c - n\log n - 2n\log 2n - \alpha n$$

It is easy to check that $mdl^*$ of any two-way refinement is larger than $mdl^*(\{W\})$ for a sufficiently large $n$ and $\alpha = 1$. Hence, our recursive algorithm would stop here. However, from Lemma 5.1, we know that the optimal clustering for the above example is $\{W(a_1), W(a_2), W(a_3)\}$.

Motivated by the small cardinality effect, we also consider the following set of candidate refinements. We consider a *greedy set cover* of $W$ using terms defined as follows. Let $a_1, a_2, \ldots$ be the ordering of terms such that $a_1$ is the most frequent term, and $a_i$ is the most frequent term among webpages that do not contain any $a_l$ for $l < i$. We fix a $k_{max}$ and for $2 < k \leq k_{max}$, we add the following refinement to the set of candidates : $\{U_1, U_2, \cdots, U_k, W - \cup_{i=1}^{k} U_i\}$, where $U_i$ denotes the set of web pages that contain $a_i$ but none of the terms $a_\ell$, $\ell < i$.

We show that if $k_{max}$ is sufficiently large, then we recover the algorithm of Sec 5.1 for single term web pages.

LEMMA 5.3. *If $k_{max}$ is larger than the number of clusters in $W$, Algorithm 5.3 discovers the optimal solution when $W$ is a set of single term web pages.*

### 3. Efficiently MDL Computation

In order to find the best refinement of $W$ from the candidate set, we need to compute the *mdl* for each refinement. If we compute the *mdl* for each refinement directly, the resulting complexity is quadratic in the size of $W$. Instead, we work with the *mdl* savings for each refinement, which is defined as $\delta_{mdl}(C) = mdl^*(C) - mdl^*(W)$.

We show that, by making a single pass over $W$, we can compute the *mdl* savings for all the candidate refinements in time linear in the size of $W$.

If $C = \{W_1, W_2, \ldots, W_k\}$, then it is easy to show that

$$\delta_{mdl} = -c + \sum_i |W_i| \log |W_i| + \sum_i |W_i| \cdot (s_i - s)$$

where, $s_i$ is the size of *script*$(W_i)$ and $s$ is the size of *script*$(W)$. Since every script term in $W$ is also a script term in $W_i$, note that $(s_i - s)$ is the number of *new* script terms in $W_i$. We now show how to efficiently compute $(s_i - s)$ for all clusters in every candidate partition in a single pass over $W$. Thus if the depth of our recursive algorithm is $\ell$, then we make at most $\ell$ passes over the entire dataset. Our algorithm will use the following notion of *functional dependencies* to efficiently estimate $(s_i - s)$.

DEFINITION 1 (FUNCTIONAL DEPENDENCY). *A term $x$ is said to* functionally determine *a term $y$ with respect to a set of web pages $W$, if $y$ appears whenever $x$ appears. More formally,*

$$x \rightarrow_W y \;\equiv\; W(x) \subseteq W(y) \tag{4}$$

*We denote by $FD_W(x)$ the set of terms that are functionally determined by $x$ with respect to $W$.*

First, let us consider the two-way refinements $\{W(t), W - W(t)\}$. Since $t$ appears in every web page in $W(t)$, by definition a term $t'$ is a script term in $W(t)$ if and only if $t' \in FD_W(t)$. Similarly, $t$ does not appear in any web page in $W - W(t)$. Hence, $t'$ is a script term in $W - W(t)$ if and only if $t' \in FD_W(\neg t)$; we abuse the FD notation and denote by $FD_W(\neg t)$ the set of terms appear whenever $t$ *does not* appear. Therefore, *script*$(W(t)) = FD_W(t)$, and *script*$(W - W(t)) = FD_W(\neg t)$.

The set $FD_W(t)$ can be efficiently computed in one pass. We compute the number of web pages in which a single term $(n(t))$ and a pair of terms $(n(t, t'))$ appears.

$$FD_W(t) = \{t' | n(t') = n(t, t')\} \tag{5}$$

To compute $FD_W(\neg t)$, we find some web page $w$ that does not contain $t$. By definition, any term that does not appear in $T(w)$ can not be in $FD_W(\neq t)$. $FD_W(\neg t)$ can be computed as

$$\{t' | t' \in T(w) \,\wedge\, n - n(t) = n(t') - n(t, t')\} \tag{6}$$

where, $n = |W|$.

Now, look at $k - way$ refinements. Given an ordering of terms $\{a_1, a_2, \ldots, a_{k_{max}}\}$, our $k$-way splits are of the form $\{U_1, U_2, \ldots, U_{k-1}, W - \cup_i U_i\}$, where $U_i$ denotes the set of web pages that contain $a_i$ but none of the terms $a_\ell$, $\ell < i$. Therefore (again abusing the FD notation), *script*$(U_i) = FD_W(\neg a_1 \wedge \neg a_2 \neg \ldots \neg a_{i-1} \wedge a_i)$. The final set does not contain any of the terms $a_\ell$, $\ell < k$. Hence, *script*$(W - \cup_i U_i) = FD_W(\wedge_{i=1}^{k-1} \neg a_i)$.

The $FD$ sets are computed in one pass over $W$ as follows. We maintain array $C$ such that $C(i)$ is the number of times $a_i$ appears and none of $a_\ell$ appear $1 \leq \ell < i$. For each non script term in $W$, we maintain an array $C_t$ such that $C_t(i)$ is the number of times $t$ appears when $a_i$ appears and none of $a_\ell$ appear $1 \leq \ell < i$. Similarly, array $R$ is such that $R(i) = |W| - \sum_{\ell=1}^{i} C(\ell)$. For each non script term $t$ in $W$, $R_t$ is an array such that $R_t(i) = |W(t)| - \sum_{\ell=1}^{i} C_t(\ell)$. The required $FD$ sets can be computed as:

$$FD_W((\wedge_{i=1}^{\ell-1} \neg a_i) \wedge a_\ell) \;=\; \{t | C(\ell) = C_t(\ell)\} \tag{7}$$

$$FD_W(\wedge_{i=1}^{\ell} \neg a_i) \;=\; \{t | R(\ell) = R_t(\ell)\} \tag{8}$$

## 5.4 Incorporating additional knowledge

Our problem formulation does not take into account any semantics associated with the terms appearing in the urls or the content. Thus, it can sometime choose to split on a term which is "clearly" a data term. E.g. Consider the urls $u_1, u_2, u_3, u_4$ from Section 2.2). The split $C_{\texttt{eats}} = \{W(\texttt{eats}), W - W(\texttt{eats})\}$ correctly identifies the scripts `eats` and `todo`.

However, sometimes, there are functional dependencies in the URLs that can favor data terms. E.g. there is a functional dependency from *Seattle* to *WA*. Thus, a split on *Seattle* makes two terms constant, and the resulting description length can be smaller than the correct split. If we have regions and countries in the urls in addition to states, the *Seattle* split $C_{\texttt{Seattle}}$ is even more profitable.

If we have the domain knowledge that *Seattle* is a city name, we will know that its a data term, and thus, we won't allow splits on this value. We can potentially use a database of cities, states, or other dictionaries from the domain to identify data terms.

Rather than taking the domain centric route of using dictionaries, here we present a domain agnostic technique to overcome this problem. We impose the following *semantic script language constraint* on our problem formulation : *if $t$ is a script term for some cluster $W$, then it is very unlikely that $t$ is a data term in another cluster $W'$.* This constraint immediately solves the problem we illustrated in the above example. $C_{\texttt{Seattle}}$ has one cluster $(W(\texttt{Seattle}))$ where `WA` is a script term and another cluster where `WA` is a data term. If we disallow such a solution, we indeed rule out splits on data terms resulting from functional dependencies.

Hence, to this effect, we modify our greedy algorithm to use a term $t$ to create a partition $W(t)$ if and only if there does not exist a term $t'$ that is a script term in $W(t)$ and a data term is some other cluster. This implies the following. First, if $t' \in script(W(t))$, then $t' \in FD_W(t)$. Moreover, both in the two-way and $k$-way refinements generated by our greedy algorithm, $t'$ can be a data term in some other cluster if and only if $t'$ is not in $script(W)$. Therefore, we can encode the *semantic script language constraint* in our greedy algorithm as:

$$\text{split on } t \text{ if and only if } FD_W(t) \subseteq script(W) \qquad (9)$$

In Algorithm 5.3, the above condition affects line number 5 to restrict the set of terms used to create two-way partitions, as well as line number 11 where the ordering is only on terms that satisfy Equation 9.

# 6. EXPERIMENTS

We first describe the setup of our experiments, our test data, and the algorithms that we use for evaluation.

**Datasets** As we described in Sec. 1, our motivation for structural clustering stems from web-scale extraction. We set up our experiments to target this. We consider four different content domains : (a) Italian restaurants, (b) books, (c) celebrities and (d) dentists. For each domain, we consider a seed database of entities, which we use, via web search to discover websites that contain entities of the given types. Fig. 1 shows the websites that we found using this process. E.g. for Italian restaurants, most of these are websites specialize in Italian restaurants, although we have a couple which are generic restaurant websites, namely `chefmoz.com` and `tripadvisor.com`. Overall we have 43 websites spanning the 4 domains. For each website, we crawl and fetch all the webpages from those sites. The second column in the table lists the number of webpages that we obtained from each site. Every resulting site has several clusters of pages. E.g, for restaurant websites have, along with a set of restaurant pages, a bunch of other pages that include users, reviews, landing pages for cities, attractions, and so on. Our objective is to identify, from each website, all the pages that contain information about our entities of interest, which we can use to train wrappers and extraction.

For each website, we manually identified all the webpages of interest to us. Note that by looking at the URLs and analyzing the content of each website, we were able to manually identify keywords and regular expressions to select the webpages of interest from each site. We use this golden data to measure the precision/recall of our clustering algorithms. For each clustering technique, we study its accuracy by running it over each website, picking the cluster that overlaps the best with the golden data, and measuring its precision and recall.

**Algorithms** We will consider several variants of our technique: Mdl-U is our clustering algorithm that only looks at the urls of the webpages. Mdl-C is the variant that only looks at the content of the webpages, while Mdl-UC uses both the urls and the content.

In addition to our techniques, we also look at the techniques that are described in a recent survey [13], where various techniques for structural clustering are compared. We pick a technique that has the best accuracy, namely, which uses a *Jaccard similarity* over path sequences between webpages, and uses a single-linkage hierarchical clustering algorithm to cluster webpages. We call this method CP-SL.

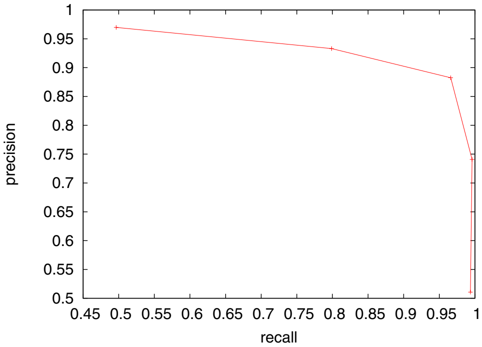

**Figure 2: Precision-Recall of Mdl-U by varying $\alpha$**

## 6.1 Accuracy

Fig. 1 lists the precision/recall of various techniques on all the sites, as well as the average precision and recall. We see that Mdl-U has an average precision of 0.95 and an average recall of 0.93, supporting our claim that urls alone have enough information to achieve high quality clustering on most sites. On some sites, Mdl-U does not find the perfect cluster. E.g., in `chefmoz`, a large fraction of restaurants (around 72%), are from *United States*, and therefore Mdl-U thinks its a different cluster, separating it from the other restaurants. Mdl-UC, on the other hand, corrects this error, as it finds that the content structure in this cluster is not that different from the other restaurants. Mdl-UC, in fact, achieves higher average precision and recall than Mdl-U. On the other hand, Mdl-C performs slightly worse that Mdl-U, again confirming our belief that urls are often more informative and noise-free than the content.

Fig. 1 also includes the precision/recall numbers for CP-SL. CP-SL algorithm is really slow, so to keep the running times reasonable, we sampled only 500 webpages from each website uniformly at random, and ran the algorithm on the sample. For a couple of sites, the fraction of positives pages was so small that the sample did not have a representation of positives pages. For these sites, we have not included the precision and recall. We see that the average precision/recall, although high, is much lower that what we obtain using our techniques.

**Dependency on $\alpha$:** Recall that the $\alpha$ parameter controls both the small cardinality and large compenent effect, and thus affects the degree of clustering. A value of $\alpha = 0$ leads to all pages being in the same cluster and $\alpha = \infty$ results in each page being in its own cluster. Thus, to study the dependency on $\alpha$, we vary *alpha* and compute the precision and recall of the resulting clustering. Fig. 2 shows the resulting curve for the Mdl-U algorithm; we report precision and recall numbers averaged over all Italian restaurant websites. We see that the algorithm has a very desirable *p-r* characteristic curve, which starts from a very high precision, and remains high as recall approaches 1.

## 6.2 Running Times

Figure 3 compares the running time of Mdl-U and CP-SL. We picked one site (`tripadvisor.com`) and for $1 \leq \ell \leq$

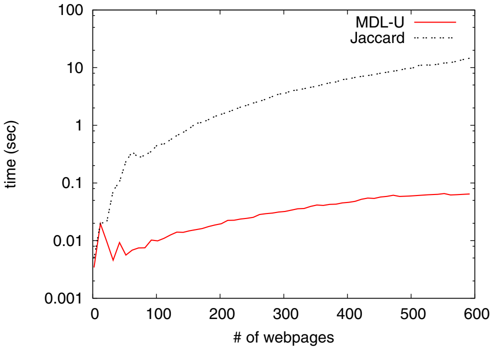

**Figure 3: Running Time of** Mdl-U **versus** CP-SL

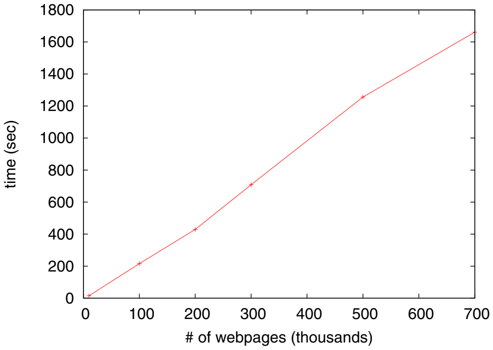

**Figure 4: Running Time of** Mdl-U

60, we randomly sampled $(10 \cdot \ell)$ pages from the site and performed clustering both using Mdl-U and CP-SL. We see that as the number of pages increased from 1 to 600, the running time for Mdl-U increases from about 10 ms to about 100 ms. On the other hand, we see a quadratic increase in running time for CP-SL (note the log scale on the y axis); it takes CP-SL about 3.5 seconds to cluster 300 pages and 14 $(= 3.5 * 2^2)$ seconds to cluster 600 pages. Extrapolating, it would take about 5000 hours ($\approx$ 200 days) to cluster 600,000 pages from the same site.

In Figure 4 we plotted the running times for clustering large samples of $100k, 200k, 300k, 500k$ and $700k$ pages from the same site. The graph clearly illustrates that our algorithm is linear in the size of the site. Compared to the expected running time of 200 days for CP-SL, Mdl-U is able to cluster 700,000 pages in just 26 minutes.

## 7. RELATED WORK

There has been previous work on structural clustering. We outline here all the works that we are aware of and state their limitations. There is a line of work [1, 5, 9, 21, 22] that looks at structural clustering of XML documents. While these techniques are also applicable for clustering HTML pages, HTML pages are harder to cluster than XML documents because there are more noisy, do not confirm to simple/clean DTDs, and are very homogeneous because of the fixed set of tags used in HTML. At the same time, there are properties specific to HTML setting that can be exploited, e.g. the URLs of the pages. There is some work that specifically target structural clustering of HTML pages [7, 8]. Several measures of structural similarity for webpages have been proposed in the literature. A recent survey [13] looks at many of these measures, and compares their performance for clustering webpages.

However, as mentioned in Section 1, current state-of-the art structural clustering techniques do not scale to large websites. While, one could perform clustering on a sample of pages from the website, as we showed in Section 6, this can lead to poor accuracy, and in some cases clusters of interest might not even be represented in the resulting sample. In contrast, our algorithm can accurately cluster sites with millions of pages in a few seconds.

## 8. CONCLUSIONS

In this work, we present highly efficient and accurate algorithms for structurally clustering webpages. Our algorithms use the principle of minimum description length to find the clustering that best explains the given set of urls and their content. We demonstrated, using several webpages, that our algorithm can run at a scale not previously attainable, and yet achieves high accuracy.

## 9. REFERENCES

[1] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki. Xproj: a framework for projected structural clustering of xml documents. In *KDD*, pages 46–55, 2007.

[2] T. Anton. Xpath-wrapper induction by generating tree traversal patterns. In *LWA*, pages 126–133, 2005.

[3] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *VLDB*, pages 119–128, 2001.

[4] M. Chlebík and J. Chlebíková. Inapproximability results for bounded variants of optimization problems. *Fundamentals of Computation Theory*, 2751:123–145, 2003.

[5] G. Costa, G. Manco, R. Ortale, and A. Tagarelli. A tree-based approach to clustering xml documents by structure. In *PKDD*, pages 137–148, 2004.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001.

[7] V. Crescenzi, G. Mecca, and P. Merialdo. Wrapping-oriented classification of web pages. In *Symposium on Applied computing*, pages 1108–1112, 2002.

[8] V. Crescenzi, P. Merialdo, and P. Missier. Clustering web pages based on their structure. *Data and Knowledge Engineering*, 54(3):279 – 299, 2005.

[9] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis. A methodology for clustering xml documents by structure. *Inf. Syst.*, 31(3):187–228, 2006.

[10] N. Dalvi, P. Bohannon, and F. Sha. Robust web extraction: An approach based on a probabilistic tree-edit model. In *SIGMOD*, pages 335–348, 2009.

[11] N. N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu. A web of concepts. In *PODS*, pages 1–12, 2009.

[12] H. Elmeleegy, J. Madhavan, and A. Y. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2(1):1078–1089, 2009.

[13] T. Gottron. Clustering template based web documents. In *ECIR*, pages 40–51, 2008.

[14] P. D. Grünwald. *The Minimum Description Length Principle.* MIT Press, 2007.

[15] P. Gulhane, R. Rastogi, S. Sengamedu, and A. Tengli. Exploiting content redundancy for web information extraction. In *VLDB*, 2010.

[16] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. In *VLDB*, 2009.

[17] W. Han, D. Buttler, and C. Pu. Wrapping web data into XML. *SIGMOD Record*, 30(3):33–38, 2001.

[18] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[19] U. Irmak and T. Suel. Interactive wrapper generation with minimal user effort. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 553–563, New York, NY, USA, 2006. ACM.

[20] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *IJCAI*, pages 729–737, 1997.

[21] M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. Xclust: clustering xml schemas for effective integration. In *CIKM*, pages 292–299, 2002.

[22] W. Lian, D. W.-l. Cheung, N. Mamoulis, and S.-M. Yiu. An efficient and scalable algorithm for clustering xml documents by structure. *IEEE Trans. on Knowl. and Data Eng.*, 16(1):82–96, 2004.

[23] A. Machanavajjhala, A. Iyer, P. Bohannon, and S. Merugu. Collective extraction from heterogeneous web lists. In *WSDM*, 2010.

[24] I. Muslea, S. Minton, and C. Knoblock. Stalker: Learning extraction rules for semistructured. In *AAAI: Workshop on AI and Information Integration*, 1998.

[25] J. Myllymaki and J. Jackson. Robust web data extraction with xml path expressions. Technical report, IBM Research Report RJ 10245, May 2002.

[26] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy web data-sources using W4F. In *VLDB*, pages 738–741, 1999.

# APPENDIX

## A. PROOFS OF LEMMAS

PROOF. OF LEMMA 4.1
Let $W$ be any set of pages, $S_0 \subset opt(W)$ and $S_1 = opt(W) - S_0$. Let $N_0$ and $N_1$ be the total number of urls in all clusters in $S_0$ and $S_1$ respectively. Using a direct application of Eq. (2), it is easy to show the following:

$$mdl(opt(W)) = N_1 \log \frac{N}{N_1} + N_2 \log \frac{N}{N_2} + mdl(S_0) + mdl(S_1)$$

Thus, if $opt(W_0) \neq \mathcal{S}_0$, we can replace $\mathcal{S}_0$ with $opt(W_0)$ in the above equation to obtain a clustering of $W$ with a lower cost than $opt(W)$, which is a contradiction. □

PROOF. OF LEMMA 5.1
Suppose there are two clusters $C_1$ and $C_2$ in $\text{OPT}(W)$ with more than 1 distinct values. Let there sizes be $n_1$ and $n_2$ with $n_1 \leq n_2$ and let $N = n_1 + n_2$. By Lemma 4.1, $\{C_1, C_2\}$ is the optimal clustering of $C_1 \cup C_2$. Let $ent(p_1, p_2) = -p_1 \log p_1 - p_2 \log p_2$ denote the entropy function. We have

$$mdl(\{C_1, C_2\}) = 2c + N \cdot ent(\frac{n_1}{N}, \frac{n_2}{N}) + \alpha N$$

Let $C_0$ be any subset of $C_1$ consisting of unique tokens, and consider the clustering $\{C_0, C_1 \cup C_2 - C_0\}$. Denoting the size of $C_0$ by $n_0$, the cost of the new clustering is

$$2c + N \cdot ent(\frac{n_0}{N}, \frac{n_1}{N}) + \alpha(N - n_0)$$

This is because, in cluster $C_0$, every term is constant, so it can be put into the script, hence there is no data cost for cluster $C_0$. Also, since $n_0 < n_1 \leq n_2 < n_2$, the latter is a more uniform distribution, and hence $ent(\frac{n_0}{N}, \frac{n_1}{N}) < ent(\frac{n_1}{N}, \frac{n_2}{N})$. Thus, the new clustering leads to a lower cost, which is a contradiction. □

PROOF. OF LEMMA 5.2
(Sketch) Suppose, w.l.o.g, that $|W(t_1)| \leq |W(r)|$ for some term $r \in W_{rest}$. Lemma 4.1 tells us that $C_0 = \{W(t_1), W_{rest}\}$ is the optimal clustering of $W_0 = W(t_1) \cup W_{rest}$. Let $C_1 = \{W(v), W_0 - W(v)\}$ and let $C_2 = \{W_0\}$. From first principles, it is easy to show that

$$max(mdl(C_1), mdl(C_2)) < mdl(C_0)$$

This contradicts the optimality of $C_1$. □

## B. NP-HARDNESS OF THE MDL-CLUSTERING PROBLEM

Given an instance $H = (V, E)$ of the 2-BOUNDED-3-SET-PACKING, we create an instance $W_H$ of MDL-CLUSTERING. For each vertex $v$, we create a webpage $v_w$ whose terms consists of all the edges incident on $v$. We call these the *vertex-pages*. Also, For each edge $e \in E$, we create $\beta$ webpages, each having a single term $e$, where $\beta$ is a constant whose values we will choose later. We call these the *edge-pages* and denote the edge-pages of $e$ by $e_W$. We set $c = 0$, and we will choose $\alpha$ later.

The set of unique terms in $W_H$ is precisely $E$. Also, since $H$ has maximum degree 2, each webpage has at most 2 terms. Let $C = \{W_1, \cdots, W_k\}$ be an optimal clustering of $W_H$. Let $E_i$ denote $script(W_i)$, i.e. the set of terms that are constant in $W_i$.

LEMMA B.1. *For all $e \in E$, there is a $i$ s.t. $E_i = \{e\}$.*

PROOF. (Sketch) Suppose there is an $e$ for which the lemma does not hold. Let $W_i$ be the cluster that contains the edge-pages for $e$. We have $|e_W| = \beta$ and $|W_i| \leq |W_H| = |E|\beta + |V| \leq |E|\beta + 3|E| \leq 2|E|\beta$, assuming $\beta > 3$. Thus, $|W_i|/|e_W| \geq 1/2|E|$. We set $\alpha$ to a large value such that $1/2|E|$ is greater than the threshold $\tau$ in Theorem 2. For such an $\alpha$, we get that $\{e_W, W_i - e_W\}$ is a better clustering for $W_i$, which is a contradiction. □

LEMMA B.2. *There is no $i$ for which $|E_i| > 1$.*

PROOF. Since each webpage has at most 2 edges, $|E_i| \leq 2$. Suppose there is a cluster $W_i$ with $|E_i| = 2$. Let $E_i = \{e_1, e_2\}$. Clearly, $n_i = |W_i| \leq 3$, since $w \in W_i$ implies $w$ is a vertex-page and there are at most 3 vertices containing $e_1$ (or $e_2$). Let $W_j$ be the cluster s.t. $E_j = \{e_1\}$, which exists according to Lemma B.1. We will show that $C_1 = \{W_i \cup W_j\}$ is a better clustering that $C_2 = \{W_i, W_j\}$. We have $n_j = |W_j| \geq \beta$. Let $n = n_i + n_j$. $mdl^*(C_2) - mdl^*(C_1) = n_i \log \frac{n}{n_i} + n_j \log \frac{n}{n_j} - \alpha * n_i \geq \log \frac{\beta}{3} - 3\alpha$. For sufficiently large values of $t$, this is positive. □

Lemma B.1 and B.2 tells us that, for a suitably chosen $\alpha$ and $\beta$, the optimal clustering of $W_H$ has exactly $|E|$ clusters, one corresponding to each edge. Each cluster contains the $\beta$ edge-pages of the corresponding edge. Every vertex-page belongs to the edge cluster of one of its adjacent edge. We want to find the assignment of vertex-pages to edge clusters that minimizes the mdl. The number of clusters and the script terms in each clusters is constant. Thus, we want the assignment that minimizes the entropy. When there exists a perfect matching, the entropy is minimized when $|V|/3$ edge clusters contain 3 vertex-pages each and rest do not contain any vertex-page. Thus, we can check if $H$ has a perfect matching by examining the optimal clustering of $W_H$

From this we get the following result.

THEOREM 3. MDL-CLUSTERING *is* NP*-hard.*

| **Website** | **Pages** | Mdl-U | | | Mdl-C | | | Mdl-UC | | | CP-SL | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **p** | **r** | **t(s)** | **p** | **r** | **t(s)** | **p** | **r** | **t(s)** | **p** | **r** |
| **Restaurants in Italy** | | | | | | | | | | | | |
| 2spaghi.it | 20291 | 1 | 1 | 2.67 | 0.99 | 0.34 | 128.79 | 1 | 1 | 182.03 | 1 | 0.35 |
| cerca-ristoranti.com | 2195 | 1 | 1 | 1.17 | 1 | 0.91 | 7.39 | 1 | 0.91 | 8.01 | 0.99 | 0.74 |
| chefmoz.org | 37156 | 1 | 0.72 | 16.18 | 1 | 0.98 | 75.54 | 1 | 0.98 | 116.73 | 1 | 0.93 |
| eristorante.com | 5715 | 1 | 1 | 2.07 | 1 | 1 | 12.62 | 1 | 1 | 13.63 | 0.43 | 1 |
| eventiesagre.it | 48806 | 1 | 1 | 15.96 | 1 | 1 | 484.28 | 1 | 1 | 799.79 | 1 | 1 |
| gustoinrete.com | 5174 | 1 | 1 | 1.04 | 1 | 1 | 15.03 | 1 | 1 | 16.84 | - | - |
| ilmangione.it | 18823 | 1 | 1 | 2.08 | 1 | 0.29 | 214.24 | 1 | 1 | 262.44 | 1 | 0.63 |
| ilterzogirone.it | 6892 | 1 | 0.26 | 1.32 | 1 | 1 | 103.22 | 1 | 1 | 108.93 | 1 | 0.44 |
| iristorante.it | 614 | 1 | 0.54 | 0.49 | 1 | 0.96 | 25.12 | 1 | 0.96 | 26.45 | 1 | 0.95 |
| misvago.it | 14304 | 0.36 | 1 | 3.66 | 0.99 | 0.93 | 297.72 | 0.99 | 0.93 | 387.13 | 1 | 1 |
| mondochef.com | 1922 | 1 | 0.79 | 1.04 | 1 | 0.79 | 10.79 | 1 | 0.79 | 11.9 | 0.23 | 0.89 |
| mylunch.it | 1500 | 0.98 | 0.94 | 1.41 | 0.98 | 1 | 3.82 | 0.98 | 1 | 4.26 | 0.98 | 0.97 |
| originalitaly.it | 649 | 1 | 0.96 | 0.48 | 0.97 | 0.85 | 31.95 | 0.97 | 0.85 | 37.67 | 0.49 | 0.93 |
| parks.it | 9997 | 1 | 1 | 1.67 | 1 | 0.5 | 14.91 | 1 | 1 | 15.28 | - | - |
| prenotaristorante.com | 4803 | 1 | 0.5 | 1.33 | 1 | 0.63 | 14.05 | 1 | 0.63 | 16.62 | 1 | 0.66 |
| prodottitipici.com | 31904 | 1 | 1 | 4.58 | 0.72 | 0.68 | 465.39 | 0.72 | 0.68 | 522.79 | 0.49 | 0.51 |
| ricettedi.it | 1381 | 1 | 1 | 0.88 | 0.6 | 0.94 | 5.29 | 0.6 | 0.94 | 5.63 | 1 | 0.74 |
| ristorantiitaliani.it | 4002 | 0.99 | 0.82 | 1.28 | 0.62 | 0.64 | 12.31 | 0.99 | 0.92 | 15.63 | 0.77 | 0.5 |
| ristosito.com | 3844 | 1 | 1 | 1.37 | 1 | 1 | 17.36 | 1 | 1 | 19.91 | 1 | 0.97 |
| tripadvisor.com | 10000 | 0.96 | 1 | 15.01 | 0.12 | 0.98 | 1527.7 | 1 | 0.82 | 1974.58 | 1 | 0.64 |
| zerodelta.net | 191 | 1 | 1 | 0.21 | 0.85 | 1 | 102.16 | 1 | 1 | 96.21 | 0.03 | 1 |
| **Books** | | | | | | | | | | | | |
| borders.com | 176430 | 0.95 | 1 | 8.5 | 0.97 | 0.65 | 896.99 | 1 | 0.93 | 1055.29 | 0.97 | 0.94 |
| chegg.com | 8174 | 0.95 | 0.99 | 2.04 | 1 | 0.59 | 25.79 | 0.99 | 0.95 | 30.7 | 1 | 0.53 |
| citylights.com | 3882 | 1 | 0.63 | 1.65 | 0.98 | 0.59 | 18.10 | 1 | 0.99 | 21.3 | 1 | 0.95 |
| ebooks.com | 51389 | 1 | 1 | 4.96 | 1 | 0.74 | 1181.78 | 0.95 | 0.99 | 1406.89 | 1 | 0.87 |
| houghtonmifflinbooks.com | 23651 | 0.76 | 1 | 3.41 | 0.76 | 0.97 | 204.83 | 0.92 | 0.86 | 240.97 | 0.41 | 1 |
| litlovers.com | 1676 | 1 | 1 | 1.09 | 1 | 1 | 4.41 | 0.92 | 0.92 | 5.25 | 1 | 0.93 |
| readinggroupguides.com | 8587 | 0.88 | 1 | 2.19 | 0.89 | 1 | 67.83 | 0.92 | 0.85 | 79.8 | 0.5 | 1 |
| sawnet.org | 1180 | 1 | 1 | 0.61 | 0.75 | 1 | 2.50 | 1 | 0.85 | 2.97 | 1 | 0.61 |
| **Celebrities** | | | | | | | | | | | | |
| television.aol.com | 56073 | 1 | 1 | 11.97 | 0.98 | 0.8 | 508.76 | 1 | 1 | 605.67 | 0.71 | 1 |
| bobandtom.com | 1856 | 1 | 0.89 | 1.07 | 0.82 | 0.96 | 7.87 | 0.96 | 0.82 | 9.04 | 1 | 0.82 |
| dead-frog.com | 2309 | 1 | 1 | 1.45 | 0.72 | 0.88 | 31.91 | 1 | 0.95 | 37.98 | 1 | 0.93 |
| moviefone.com | 250482 | 1 | 1 | 8.19 | 0.91 | 0.59 | 3353.17 | 0.97 | 1 | 3854.21 | 1 | 0.94 |
| tmz.com | 211117 | 1 | 0.88 | 10.74 | 0.87 | 0.88 | 1712.31 | 0.93 | 0.82 | 2038.46 | - | - |
| movies.yahoo.com | 630873 | 0.26 | 1 | 9.39 | 0.99 | 0.79 | 11250.44 | 0.98 | 0.94 | 12931.55 | 0.38 | 0.36 |
| **Doctors** | | | | | | | | | | | | |
| dentistquest.com | 2414 | 1 | 1 | 0.97 | 1 | 1 | 7.08 | 1 | 1 | 12.15 | 1 | 0.33 |
| dentists.com | 8722 | 0.99 | 1 | 1.69 | 0.69 | 0.99 | 12.89 | 1 | 1 | 43.27 | 0.23 | 1 |
| dentistsdirectory.us | 625 | 0.97 | 0.99 | 0.37 | 0.95 | 0.99 | 2.53 | 0.95 | 0.99 | 2.78 | 0.96 | 0.75 |
| drscore.com | 14604 | 1 | 1 | 3.53 | 1 | 0.72 | 124.92 | 1 | 1 | 199.57 | 1 | 0.67 |
| healthline.com | 98533 | 1 | 1 | 23.33 | 1 | 0.85 | 2755.18 | 1 | 1 | 1624.53 | 1 | 0.54 |
| hospital-data.com | 29757 | 1 | 1 | 4.91 | 1 | 1 | 344.82 | 1 | 1 | 143.6 | 1 | 0.79 |
| nursinghomegrades.com | 2625 | 1 | 1 | 1.32 | 0.9 | 1 | 15.08 | 0.98 | 1 | 17.68 | 1 | 0.45 |
| vitals.com | 34721 | 1 | 1 | 7.46 | 0.99 | 0.92 | 422.26 | 0.99 | 0.92 | 793.1 | 1 | 0.5 |
| **Average** | | **0.95** | **0.93** | | **0.91** | **0.84** | | **0.97** | **0.93** | | **0.84** | **0.77** |
| **Total** | **1849843** | | | **186.74** | | | **26521.13** | | | **29799.22** | | |

**Figure 1: Comparison of the different clustering techniques**