# Detection of Distinct URL and Removing DUST Using Multiple Alignments of Sequences

**Prof. Sandhya Shinde[1], Ms. Rutuja Bidkar[2],Ms. Nisha Deore[3], Ms. Nikita Salunke[4] , Ms. Neelay Shivsharan[5]**

*[1] Professor, Department of Information Technology, PCCOE, Pune*
*[2345] Student, Department of Information Technology, PCCOE, Pune*

---

**Abstract -** ***As we know large numbers of URLs collected by web crawlers correspond to pages with duplicate or near-duplicate contents. Result of this, to crawl, to store, and use such duplicated data implies the building of low quality rankings, a waste of resources and poor user experiences. To deal with such problem, several studies have been proposed. This studies focus on, to detect and remove duplicate documents without fetching their contents. To derive this, the proposed methods learn the normalization rules to transform all duplicate URLs into the same canonical form. A challenging aspect of this strategy is to derive a set of general and precise rules. Here we are going to present DUSTER, a new approach to derive quality rules, which that take advantage of a multi-sequence alignment strategy. Before the generation of the rules takes place, a full multi-sequence alignment of URLs with duplicated content is demonstrated that can lead to the deployment of very effective rules. By evaluating this method, we observed it achieved larger reductions in the duplicate URLs than our best baseline, with gains of 82% and 140.74% in two different web collections.***

***Key Words:*** *Search engine, web Crawler, duplicate detection, URL Normalization.*

## 1. INTRODUCTION

The results which comes after the crawling the corresponding pages, contains the duplicate or near-duplicate contents. Hence these duplicate URLs commonly known as DUST (Duplicate URLs with Similar Text). It can be effectively explained using following example, the URLs http://google.com/news and http://news.google.com return the same content. That means while searching for the news using any of above URL result the same content. This DUST can be created for many reasons. For instance, to facilitate the user's navigation, many web site developers define links or redirection as alternative path to find or search a document. Secondly it is used by the webmasters for load balancing and to ensure fault tolerance. Thus detecting such DUST (Duplicate URLs with Similar Text) is an extremely important task for search engines, as crawling this redundant content tends to several drawbacks. Drawbacks can be waste of resources (e.g. bandwidth and disk storage); disturbance in results of link analysis algorithms; Due to duplicate results we get poor user experience. To overcome such problem several methods are proposed for detecting and removing DUST. Recent studies focused on comparing URLs rather than whole document or document content. These studies are classified as Content Based and URL Based DUST detection. By analyzing these technique we conclude that URL based DUST detection is efficient. Also known as URL- based de- duping.

The main challenge for these methods is to derive general rules from available training sets. Some methods make use of derive rules from pairs of duplicate URLs. The quality of rules is affected by the criterion used to select these pairs. Current methods are less affected by noise and derive rules that are very specific. Thus, an ideal method should learn general rules from few training examples, taking maximum advantage, without sacrificing the detection of DUST across different sites. The DUSTER is introduced, motived by these issues. DUSTER takes advantages of multiple sequence alignment (MSA) in order to obtain a smaller and more general set of normalization rules. Traditionally the multiple sequence alignment is used in molecular biology as a tool. The tool, find out the similar pattern in sequences. By applying these we are able to identify similarities and differences among strings. As the methods find patterns involving all the available strings, the method is able to find more general rules avoiding problems related to pairwise rule generation and problem related to finding rules across sites.

To make learning process more robust and less susceptible to noise, we show that the multiple sequence alignment of duplicate URLs, performed before rules are generated. Our proposed method is able to generate rules involving multiple DNS names. And has an acceptable computational cost even when crawling in large scale scenarios. Here its complexity is proportional to the number of URLs to be aligned, unlike other methods where the complexity is proportional to the number of specific rules generated from all clusters, which can be not feasible in practice. In this method, we do not derive candidate rules from URL pairs within the dup-cluster. We first align all the URLs in the dup-clusters obtaining consensus sequences for each dup-

cluster. Then rules are generated from these sequences. For large clusters, which are rare, we adopt a heuristic similar to the one proposed by to ensure the efficiency of the method. Evaluating our method, we observed it diminished the number of duplicate URLs achieving gains of 82% and 140% over the best baseline found in literature, when considering two different web collections.

## 2. RELATED WORK

DUST (Duplicate URLs with Similar Text) occurs for many reasons like to facilitate user's navigation. Usually webmasters mirror the content to balance load and ensure fault tolerance. Detecting DUST is extremely important problem for search engines. Since all the principle functions of search engine including indexing, crawling, ranking, presentation etc. impacted by presence of DUST. DUST detection can be classified into two types of methods – Content based and URL based. In Content based DUST detection, it is necessary to fetch the whole content of the corresponding pages of distinct URL's present. In order to avoid such waste of resources several URL based methods have been proposed to find out DUST without fetching associated contents of URL's. In following paragraphs, we focus on different URL-based methods, as far as we know, reported the best results in the literature.

Dust-Buster [1] was the first URL Based proposed method. Dust-Buster algorithm is used for mining DUST very effectively from a URL list. In this, author addressed DUST detection problem as a problem of finding normalization rules able to transform given URL to another likely to have similar content. It can reduce crawling overhead by up to 26% , increases crawl efficiency and reduces indexing overhead.

As the substitution rules were not able to capture many common duplicate URL transformations on web, so the author Dasgupta presented a new formalization of URL Rewrite rules [2]. Author uses some heuristics to generalize the generated rules. Rewrite rules applied to remove the duplicates. In this, set of URL's classified into classes and according to classes of similar content clusters are formed. Then the Rewrite rules are applied to clusters. It helps in trapping duplicates easily in search engine workflow. It also improves efficiency of entire process and effective in large scale experiment. As a result, learned rule can't obtain maximum compression expected for this.

The authors Hema Koppula and Amit Agarwal extended the work presented by Dasgupta to make their use feasible at web scale. They presented machine learning technique to generalize the set of rules that reduces resource footprint to be usable at web scale. In this paper[3], the major focus is an efficient and large scale de-duplication of documents on world wide web. Authors evaluated the method with 3 billion URL's showing it's scalability. Different pair-wise rules are generated from pairs of URL's in dup-cluster. Rule generation technique uses efficient ranking methodologies for reducing number of pair-wise rules. Pair-wise rules thus generated are consumed by the decision tree algorithm to generate highly precise generalized rules. In this paper [3], deep tokenization is used. URL's divided into number of tokens. Based on tokens, content and transactions are calculated. As it is not publically available and don't describe enough data to experiment. So, it has limitations.

As these all previous approaches are inefficient and very sensitive to noise. Thus authors in [4] proposes top-down approach. In this paper [4], a pattern tree based approach is used for learning URL normalization rules. In this first, training data set is created. A pattern tree is generated on basis of training data set. Then the duplicate nodes are identified from the pattern tree. And at last, the normalization rules are generated. As these normalization rules are directly induced on pattern rather than on every URL pair, the respective computational cost is low.

## 3. PROBLEM FORMULATION

Training set i.e. set of URLs is given input to this problem. The URLs are partitioned into group of similar pages (called as dup-cluster) from one or more domain. URL-based de-duping methods, strategy is to learn, by mining these dup-clusters, rules that transform duplicate URLs to the same canonical form.

Example of URLs to be de-duplicated and possible canonical forms. URLs of a same dup-cluster point to the same or similar content. URLs from different dup-clusters likely correspond to different content. Thus, in this example, whereas contents of $u_0$ and $u_2$ are the same, contents of $u_0$ and $u_5$ are different.

| dup-cluster | URL |
|---|---|
| $C_1$ | $u_0$ = http://britney.com.br/?id=5 |
| | $u_1$ = http://britney.com.br/index.php?id=5 |
| | $u_2$ = http://Britney.com.br/?id=5 |
| | $u_3$ = http://www.britney.com.br/?id=5 |
| | $n_1$ = http://www.britney.com.br/index.php?id=5 |
| $C_2$ | $u_4$ = http://britney.com.br/?id=7 |
| | $u_5$ = http://Britney.com.br/index.php?id=7 |
| | $n_2$ = http://www.britney.com.br/index.php?id=7 |

In above table, U = {u1, u2, u3, u4, u5} is partitioned in dup-clusters C1 and C2. The canonical form of the URLs in
C1 and C2 are given by n1 and n2, respectively. This process, called as URL normalization, identifies, at crawling time, whether two or more URLs are DUST without fetching their contents. As crawlers have resources constraints, the best methods are those that achieve larger reductions with smaller false positive rates using the minimum number of normalization rules. A normalization rule is a description of the conditions and operations necessary to transform a URL into a canonical form.

## 4. BACKGROUND

In this we see the problem of Sequences alignment, it show us How to apply Sequences Alignment to URL and we study Why use of URL alignment in URL de-duplication.

### 4.1 Sequences Alignment

In Sequences Alignment is a method to arrange multiple sequences in order to identify the similar parts among them. The main aim of alignment process is to inserting space into the sequences so the similar sequences are aligned in the same position. In the this case the alignment of similar tokens gives the rules that transform DUST into a canonical form.

#### 4.1.1 Pairwise Sequences Alignment

The alignment of any two Sequences is known as pairwise sequence alignment, the initial step for alignment sequences this issues can be solve using the dynamic programming to calculate all the sub problems. It possible when given sequences is less than two then we implemented the pairwise sequence alignment.

#### 4.1.2 Multiple sequences Alignment

A multiple sequences alignment of sequences can be the taken as a natural generalization of the above method .This method is taken into consideration when sequences greater than two no of sequences are present in the training dataset, In this gaps are inserted at arbitrary position in any number of sequences to be aligned ,so the result we get have the same size l. the sequences are arrange in k lines and l column.so that spaces of sequences can be occurred in single column.

The Multiple Sequences alignment method is known as NP-hard problem there are different solution have been invented to find a heuristic solution and to overcome this problem. So we use the method called as Progressive Alignment to align cluster of duplicate URL. In dupcluster there are multiple URL are present .in this method perform the alignment from already selected sequences and then after that new sequences is aligned with pervious alignment, from that we get the final multiple sequences alignment. This process is repeated until get all sequences been aligned

The progressive alignment uses the method which is known as greedy policy, in that once gaps is given, which is not removed for an sequences alignment. So gaps are preserved until the final solution .the most similar sequences are selected due to error rate in alignment at every stage seen to be decrease and increase if most similar sequences are selected. It determine the best order of sequences for the alignment .Mostly similar Sequences are alignment first and at end most different Sequences due to this reduces the error which introduced by this heuristic solution.

### 4.2 URL Alignment

Our method takes the advantages of multiple sequences Alignment to obtain smaller and more general set of normalization rules. In the method each dupcluster generate the consensus sequences from the given input set and generate the rule out of them. The task is aligning the URLs in each cluster and then consensus sequences give result alignment of URLs. In next we see the how we represent URLs.

#### 4.2.1 URL Tokenization

We use the easier representation of URL .in this process decomposition the URL into a sequences of URL tokens, used as tokenization. Each URLs are represent as a singleton set. For Example URL s=http://www.google.com/news.html is represented by the following sequences of 13 token set.

S=<{http},{:},{/},{/},{www},{.},{google},{.},{com},{/},{news},{.},{html}>.

#### 4.2.2 Pair-Wise URL Alignment

The consensus sequences is the output of alignment process of sequences set. Which give result of the alignment. The consensus sequences of n sequences is composed by the union of the token in the corresponding of the n aligned sequences .it help the user for better understanding the complete flow

#### 4.2.3 Multiple URL Alignment

In this we show how to align a dupcluster lager than two URLs. To overcome we use the progressive alignment strategy in that align the most similar sequences at each stage and infer a new token set sequences from them this process is repeated until all sequences have been aligned, result into the final multiple alignment.

We see in Multiple sequences alignment is used to select the most similar sequences based on Multiple sequences alignment algorithim.by aligning all urls we obtain a consensus sequences show the entire cluster .we can derive a general rule from consensus sequences. Analysis of the Algorithm we consider the no of alignment between sequences as the revanlt cost measure to determine the running time .so firstly a priority queue is created from all pair of URLs in C so all the has to be aligned (line 3-11).By using priority queue it is possible to find the tuple with most similar pair of URLs(line 13.).these sequences are moved form set of sequences to be aligned and added to the set of aligned sequences(line 15-16).then align the consensus sequences in tuple ϭ with all the reaming sequences be aligned(line 17-20).the progressive alignment is carried out until final multiple alignment is done(line12-23).at end of this process we have aligned all URL in cluster C and reduced them to a single sequences of token sets.

So the complexity of algorithm is O(P*$n^2$).where P is pair-wise URL alignment .It is more efficient algorithm and we adapted for future work

**Algorithm 1 MultipleURLAligment ($\mathcal{C}$)**

**Input:** A dup-cluster $\mathcal{C} = \{u_1, ..., u_n\}$ with $n$ duplicate URLs
**Output:** A tuple $\pi = (consensus, domains, support)$.
1: Let $\mathcal{Q}$ be a priority queue in which tuples $\sigma = (x, y, consensus_{xy}, scoring)$ are sorted in descending order according to the alignment $scoring$.
2: $D_{omains} = \emptyset$; $S_{equences} = \emptyset$; $S_{upport} = \emptyset$; $A_{ligned} = \emptyset$;
3: **for all** pairs of distinct urls $u_1, u_2$ in $\mathcal{C}$ **do**
4: $S_{upport} = S_{upport} \cup \{(u_1, u_2)\}$
5: $D_{omains} = D_{omains} \cup \{domain(u1)\} \cup \{domain(u2)\}$
6: $x = tokenize(u1)$
7: $y = tokenize(u2)$
8: $S_{equences} = S_{equences} \cup \{x\} \cup \{y\}$
9: $\sigma = PairURLAlignment(x, y)$
10: add $\sigma$ to $\mathcal{Q}$
11: **end for**
12: **while** $\mathcal{Q}$ is not empty **do**
13: Pop the first tuple $\sigma$ from $\mathcal{Q}$
14: **if** $\sigma.x \notin A_{ligned}$ and $\sigma.y \notin A_{ligned}$ **then**
15: $A_{ligned} = A_{ligned} \cup \{\sigma.x\} \cup \{\sigma.y\}$
16: $S_{equences} = S_{equences} - A_{ligned}$
17: **for all** sequences $s$ in $S_{equences}$ **do**
18: $\epsilon = PairURLAlignment(\sigma.consensus, s)$
19: add $\epsilon$ to $\mathcal{Q}$
20: **end for**
21: $S_{equences} = S_{equences} \cup \{\sigma.consensus\}$
22: **end if**
23: **end while**
24: Let $s$ be the unique consensus sequence in $S_{equences}$
25: **return** $\pi = (s, D_{omains}, S_{upport})$

## 5. DUSTER

Here, in this section we describe in detail our solution to avoid the presence of DUST in search engine.

In this figure, once a new set of URLs is crawled, it is merged with the already known URLs, Which form a new set of known URLs. During crawling, the crawler is also able to identify examples of DUST by following canonical tags. As a result, a new set of known DUST is also available. This set can be still enriched by processes such as those based on content signature, followed by manual inspection. Given the final set of known DUST, DUSTER can use it to find and validate rules, by splitting it in training and validating sets. The resulting rules are then used to normalize the known URLs yielding a new (and reduced) set of URLs to be crawled. By using this set and the set of DUST rules, the crawler can gather new URLs, closing the cycle.

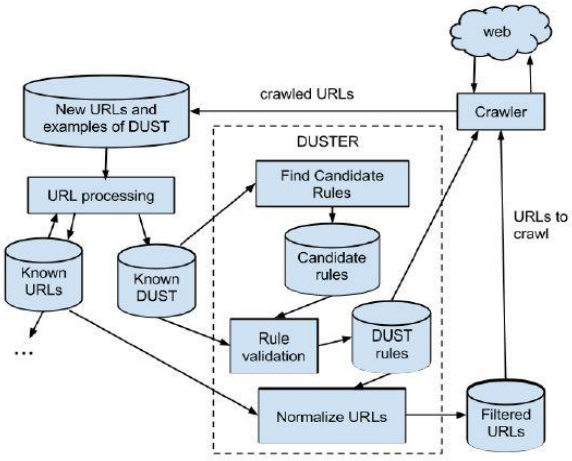


Fig. 4. DUSTER framework. The dotted box highlights the components of the DUSTER algorithm.

The two main phases of DUSTER are the generation of candidate rules, where our algorithm i.e. multiple sequence alignment generate rules from dup-clusters, and rules validation. In rules validation DUSTER filters out candidates rules according to their performance in a validation set.

### Phase 1: Candidate Rules Generation

This consists of two steps: (1) for each dup-cluster in the training set, we obtain a rule. To accomplish this, we first align a set of K URLs randomly selected within the cluster in order to obtain a consensus sequence to represent these URLs. Note that if the cluster has less than K URLs, all of them are involved in the alignment process. We adopt this sampling strategy because, even with few URL training examples, it is still possible to generate the correct rule for the cluster. In this way, we also avoid the alignment of large clusters which could be very expensive. Also note that, since the cluster size distribution is much skewed, our sampling strategy affects only the larger dup-clusters, that is, the minority of the dup-clusters. (2) From the generated rules, we discard the ones with frequency less than min freq. Thus, very specific rules, with few occurrences in the training set, are discarded. Algorithm presents GenerateCandidateRules which takes a set of dup-clusters as input and generates a set of candidate rules as output. In this, two tables are created: RT (Rules Table) which stores the candidate rules generated for each cluster, and CRT (Candidate Rules Table) which stores rules which exceed the frequency threshold minfreq. After that K URLs are randomly selected from ci and aligned by algorithm MultipleURLAligment (see Algorithm 1). In lines 6 to 7, candidate rules are generated and added to CRT .

**Algorithm 2** GenerateCandidateRules ($\mathcal{TS}$)

**Input:** Training Set $\mathcal{TS} = \{c_1, ..., c_n\}$ with $n$ duplicate clusters
**Output:** Set of $m$ candidate rules $\mathcal{CR} = \{r_1, ..., r_m\}$
1: Create table $\mathcal{RT}$ (*context, transformation, domains, support*)
2: Create table $\mathcal{CRT}$ (*context, transformation, domains, support*)
3: **for all** cluters $c_i \in \mathcal{TS}$ **do**
4: $T$ = selectKRandomlyURLsFrom($c_i$)
5: $\pi$ = MultipleURLAligment($T$)
6: r = generateRule ($\pi$.consensus)
7: add ($r.context$, $r.transformation$, $\pi.domains$, $\pi.support$) to $\mathcal{RT}$
8: **end for**
9: group tuples in $\mathcal{RT}$ into buckets by (*context, transformation*)
10: **for all** buckets B **do**
11: **if** ($|B| >= min_{freq}$) **then**
12: $D_{omains} = \emptyset$; $S_{upport} = \emptyset$;
13: **for all** tuples $t \in B$ **do**
14: $D_{omains} = D_{omains} \cup t.domains$
15: $S_{upport} = S_{upport} \cup t.support$
16: **end for**
17: $\alpha$ = the first tuple in B
18: add ($\alpha.context$, $\alpha.transformation$, $D_{omains}$, $S_{upport}$) to $\mathcal{CRT}$
19: **end if**
20: **end for**
21: **return** a set $\mathcal{CR}$ of rules created from $\mathcal{CRT}$

## Phase2: Validating Candidate Rules

The goal of this phase is to consider as valid or refute the candidate rules generated in the previous phase. This filter selects the more effective rules by two pre-defined thresholds: false-positive rate (fprmax) and minimum support (minsupp). If the false-positive rate is larger than fprmax or the support of the rule is smaller than minsupp, the rule is discarded. Otherwise, the rule is added to the set of valid rules. Note that rules with small support values are not desirable anyway, because the reduction gained by applying them can be insignificant. Thus, the support value is indicative of the possible compression that a rule can achieve, whereas the false-positive rate corresponds to the precision of the rule in the task of DUST detection.

**Algorithm 3** ValidateRules ($\mathcal{VS}$, $\mathcal{CR}$, $fpr_{max}$, $min_{supp}$)

**Input:** $\mathcal{VS}$: validation set, $\mathcal{CR}$: Set of $n$ candidate rules, $fpr_{max}$: maximum false-positive rate that can be tolerated, $min_{supp}$: minimum number of instances required.
**Output:** Set of $n$ valid rules $\mathcal{VR} = \{r_1, ..., r_n\}$
1: Create table $\mathcal{CT}$ (*canonical, url*)
2: Create table $\mathcal{RT}$ (*context, transformation, domains, support*)
3: **for all** *candidate rules* $r$ in $\mathcal{CR}$ **do**
4: $Nsupp = 0$; $Nfpp = 0$; $S_{upport} = \emptyset$;
5: $\mathcal{U} = \cup_{\forall d \in r.D_{omains}}$ URLs from domain $d$ in $\mathcal{VS}$.
6: **for all** urls $u$ in $\mathcal{U}$ **do**
7: **if** $r.context$ matches with $u$ **then**
8: $canonical$ = NormalizeURL ($u$, $r$)
9: add ($canonical$, $u$) to $\mathcal{CT}$
10: **end if**
11: **end for**
12: group tuples in $\mathcal{CT}$ into buckets by (*canonical*)
13: **for all** buckets B **do**
14: **if** ($|B| > 1$) **then**
15: **for all** pairs of distinct tuple $t_1$,$t_2 \in B$ **do**
16: $Nsupp = Nsupp + 1$
17: $S_{upport} = S_{upport} \cup \{(t_1.url, t_2.url)\}$
18: **if** ($t_1$.url and $t_2$.url *are not* DUST) **then**
19: $Nfpp = Nfpp + 1$
20: **end if**
21: **end for**
22: **end if**
23: **end for**
24: **if** ($Nsupp >= min_{supp}$) **then**
25: $fpr = Nfpp/Nsupp$
26: **if** ($fpr <= fpr_{max}$) **then**
27: add ($r.context$, $r.transformation$, $r.domains$, $S_{upport}$) to $\mathcal{RT}$
28: **end if**
29: **end if**
30: Clear table $\mathcal{CT}$
31: **end for**
32: **return** a set of all rules in $\mathcal{RT}$

## CONCLUSION AND FUTURE WORK

In our system we can use the framework which is known as DUSTER to report the DUST problem i.e detection the number of differnt URLs which have the duplicate or near duplicate contents. In DUSTER framework we had study URL normalization, is used to transform the number of URLs into a canonical form. Using this form it becomes to detect distinct URLs. DUSTER method used the generalization and validating the candidate rules the main goal behind that to select the effective rules and filter the URLs effectively and efficiently. Accuracy level is increases using this approach.

In future work ,our aim to effective use of resources in term of bandwidth and disk storage and improve the scalability as well as user experience.

## REFERENCES

[1] Z. Bar-Yossef, I. Keidar, and U. Schonfeld. Do not crawl in the dust: Different urls with similar text. ACM Trans. Web, 3(1):3:1–3:31, Jan 2009.

[2] A. Dasgupta, R. Kumar, and A. Sasturkar. De-duping urls via rewrite rules. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, pages 186–194, New York, NY, USA, 2008. ACM.

[3] H. S. Koppula, K. P. Leela, A. Agarwal, K. P. Chitrapura, S. Garg, and A. Sasturkar. Learning URL patterns for webpage deduplication. In Proceedings of the third ACM international conference on Web search and data mining, WSDM '10, pages 381–390, New York, NY, USA, 2010. ACM.

[4] T. Lei, R. Cai, J.-M. Yang, Y. Ke, X. Fan, and L. Zhang. A pattern tree-based approach to learning URL normalization rules. In Proceedings of the 19th international conference on World wide web, WWW '10, pages 611–620, New York, NY, USA, 2010. ACM