**CHAPTER 1: INTRODUCTION**

* 1. **PROBLEM DEFINITION**

Detection of distinct URL and removing DUST using multiple alignment of sequences.

* 1. **MOTIVATION**

The main challenge for these methods is to derive general rules with a reasonable cost from the available training sets. As observed in, many methods derive rules from pairs of duplicate URLs. Thus the quality of these rules is affected by the criterion used to select these pairs and the availability of specific examples in the training sets. To avoid processing large numbers of URLs, most of the methods employ techniques such as random sampling or by looking for DUST only within sites, preventing the generation of rules involving multiple DNS names. Because of these issues, current methods are very susceptible to noise and, in many cases, 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. New method DUSTER, that takes advantage of multiple sequence alignment in order to obtain a smaller and more general set of normalization rules.

The proposed system aims at detection and removal of distinct URLs with the help of normalization Rules.

* 1. **OBJECTIVE**

The objectives of this system are:

- To learn normalization rules to transform all duplicate URLs into the same canonical form.

- DUSTER, new approach addresses the DUST problem, i.e. detection of distinct URLs that correspond to pages with duplicate or near-duplicate content. DUSTER learns normalization rules that are very precise in converting distinct URLs which refer the same content to a common canonical form, making it easy to detect them.

* 1. **BASIC CONCEPTS**
     1. **DUST**

DUST means duplicate URLs with similar text. There are the different sites where same page may be found under different URLs. Because of this search engine face different problems like- (1) Search Engine may take large amount of time to visit each page, (2) Search Engine may treat each duplicate page as different. Crawling this redundant content leads to drawbacks like waste of resources. So, detection of DUST is the important task for all search engines.

* + 1. **WEB CRAWLER**

To find information on millions of Web pages that exist, a search engine employs special software robots, called spiders, to build lists of the words found on Web sites. When a spider is building its lists, the process is called Web crawling. A Web crawler is an [Internet bot](https://en.wikipedia.org/wiki/Internet_bot) which systematically browses the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), typically for the purpose of [Web indexing](https://en.wikipedia.org/wiki/Web_indexing). A Web crawler may also be called a Web spider, or an automatic indexer, or Web scutter. [Web search engines](https://en.wikipedia.org/wiki/Web_search_engine) and some other sites use Web crawling software to update their [web content](https://en.wikipedia.org/wiki/Web_content) or indexes of others sites' web content. Web crawlers can copy all the pages they visit for later processing by a search engine which [indexes](https://en.wikipedia.org/wiki/Index_(search_engine)) the downloaded pages so the [users](https://en.wikipedia.org/wiki/User_(computing)) can search much more efficiently.

* + 1. **DUSTER**

DUSTER is the solution to avoid the presence of DUST in search engines. When URLs is crawled, it is merged with the already known URLs to form a new set of known URLs. During crawling, the crawler is also able to identify examples of DUST by different 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. The two main phases of DUSTER are the generation of candidate rules, where a multi-sequence alignment algorithm generates candidate rules from dup-clusters, and rules validation, where DUSTER filters out candidate rules according to their performance in a validation set.

**CHAPTER 2: LITERATURE SURVEY**

Current research on DUST detection can be classified in two main families of methods: content-based and URL based. In content-based DUST detection, to infer if two distinct URLs correspond to duplicates, or near duplicates, it is necessary to fetch and inspect the whole content of their corresponding pages. In order to avoid such a waste of resources, several URL-based methods have been proposed to determine duplicate URLs without examining the associated contents.

The methods previously described use a bottom-up approach in which normalization rules are learned by inducing local duplicate pairs to more general forms. The main drawback of these strategies is the difficulty to induce general rules starting from pairs of duplicate URLs.

The main differences between this work and the previous one are (1) the handling of large dup-clusters; (2) the adoption of new methods for intra-cluster generalization and alignment penalization; (3) the elimination of a hierarchical clustering step with the reduction of the number of generated rules; and (4) the simplification of the algorithm, by supporting fewer kinds of tokens.

[1] **Z . Bar-Yossef, I. Keidar, and U. Schonfeld, “Do not crawl in the**

**DUST: Different URLs with similar text”.**

It presents Dust Buster algorithm for mining DUST very effectively from a URL list. The first URL-based method proposed was Dust-Buster. In this, the authors addressed the DUST detection problem as a problem of finding normalization rules able to transform a given URL to another likely to have similar content. The rules consist of substring substitutions learned from crawl logs or web logs. Rules are selected if (a) they have large support, (b) they do not come from large groups and (c) URLs matched by them have similar sketches or compatible sizes in the training log. Redundant rules are eliminated based on their support information. Knowledge of DUST rules is essential for canonizing URL names. Canonical names are very important for statistical analysis of URL popularity based on Page Rank or traffic.

**[2] A. Dasgupta, R. Kumar, and A. Sasturkar, “De-duping URLs via**

**Rewrite Rules”.**

Since substitution rules were not able to capture many common duplicate URL transformations on the web, Dasgupta etc. all presented a new formalization of URL rewrite rules. The new formulation was expressive enough to capture all previous substitution rules as well as more general patterns, such as the presence of irrelevant substrings, complex URL token transpositions and session-id parameters. The authors use some heuristics to generalize the generated rules. In particular, they attempt to infer the false-positive rate of the rules in order to select the most precise ones.

A large number of URLs on the web contain duplicate or near-duplicate content. De-duping URLs is very important problem for search engines and it is addressed by fetching and examining the content of the URL. The rewrite rules can be applied to eliminate duplicates among URLs that are encountered for the first time during crawling, even without fetching their content. It helps in trapping duplicates easily in search-engine workflow. It improves efficiency of entire process and effective in a large-scale experiment.

**[3]H. S. Koppula, K. P. Leela, A. Agarwal, K. P. Chitrapura, S. Garg, and A. Sasturkar, “Learning URL patterns for webpage De-duplication”.**

The authors Hema Koppula and Amit Agarwal extended the work presented by Dasgupta to make their use feasible at web scale. They observed that the quadratic complexity of the rule extraction performed in is prohibitive for very large dup-clusters. Thus, they proposed a method for deriving rules from samples of URLs. In addition, they used a decision tree algorithm to learn a small number of higher precision rules to minimize the number of rules deployed to the crawler.

**[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 this paper, a pattern tree-based approach is used for learning URL normalization rules. Pattern trees based on the training set is created first, and then generate normalization rules via identifying duplicate nodes on the pattern tree. The computational cost is also low as rules are directly induced on patterns, rather than on every duplicate URL pair. The pattern tree helps select deployable rules by removing conflicts and redundancies.

**CHAPTER 3: PROBLEM STATEMENT**

**Detection of distinct URL and removing DUST using multiple alignment of sequences.**

Different URLs that have similar content is a common phenomenon on the Web. Besides plagiarism, these duplicate URLs, generically known as DUST (Duplicate URLs with Similar Text), occur for many reasons. For instance, in order to facilitate the user’s navigation, many web sites define links or redirections as alternative paths to reach a document. In addition, webmasters usually mirror content to balance load and ensure fault tolerance. Other common reasons for the occurrence of duplicate content are the use of parameters placed in distinct positions in the URLs and the use of parameters that have no impact on the page content, such as the session\_id attribute, used to identify a user connection. Detecting DUST is an extremely important task for search engines since crawling this redundant content leads to several drawbacks such as waste of resources (bandwidth and disk storage, for example); disturbance in results of link analysis algorithms; and poor user experience due to duplicate results. To overcome these problems, several authors have proposed methods for detecting and removing DUST from search engines. Whereas first efforts focused on comparing document. We present DUSTER, a new method that takes advantage of multiple sequence alignment in order to obtain a smaller and more general set of normalization rules. Multiple sequence alignment is traditionally used in molecular biology as a tool to find similar patterns in sequences. Its importance in biology has motivated many efforts concerning the proposition of optimized algorithms. By applying multiple sequence alignment we are able to identify similarities and differences among strings. We show that these similarities and differences can be explored to determine fixed and mutable substrings in URLs, facilitating the derivation of normalization rules. As multiple sequence alignment methods find patterns involving all the available strings, the method is able to find more general rules avoiding Removing DUST using Multiple Alignment of Sequences problems related to pairwise rule generation, and the problem related to finding rules across sites. We show that a full multi-sequence alignment of duplicate URLs, performed before rules are generated, can make the learning process more robust and less susceptible to noise when compared to previous work in the literature. Furthermore, we show that 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. 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 unfeasible in practice. Unlike other methods, 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. Rules are then 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.

**CHAPTER 4: PROJECT REQUIREMENT**

**4.1 Nonfunctional Requirements**

* Maintainability

All the modules must be clearly separate to allow different user interfaces to be developed in future, through thoughtful and effective software engineering; all steps of the software development process will be well documented to ensure maintainability of the product throughout its life time. All development will be provided with good documentation.

* Performance

The response time, utilization and throughput behavior of the system are a part of the performance. Care is taken so as to ensure a system with comparatively high performance. The proposed system is designed with aim of providing relevant documents within short period of time.

* Usability

The ease of use and training the end users of the system is usability. System should have qualities like- learning ability and efficiency. The proposed system provides high usability through attractive, easy to use GUI and providing classification and justification by minimal user efforts.

* Modifiability

The ease with which a software system can accommodate changes to its software is modifiability. Our project is easily adaptable for changes that is useful for the application to withstand the needs of the users.

* Reusability

The extent to which the existing application can be reused for further versions of android is reusability. Our application can be reused a number of times without any technical difficulties.

**4.2 Software requirements**

Operating System  **:** Windows XP, Windows7, Windows 8

Software **:** Net Beans,JDK

Front End **:** JAVA

// Back End **:** MySQL

* 1. **Hardware requirements**

Processor **:** 80x86/Pentium

RAM **:** 4GB

Hard Disk **:** 10 GB

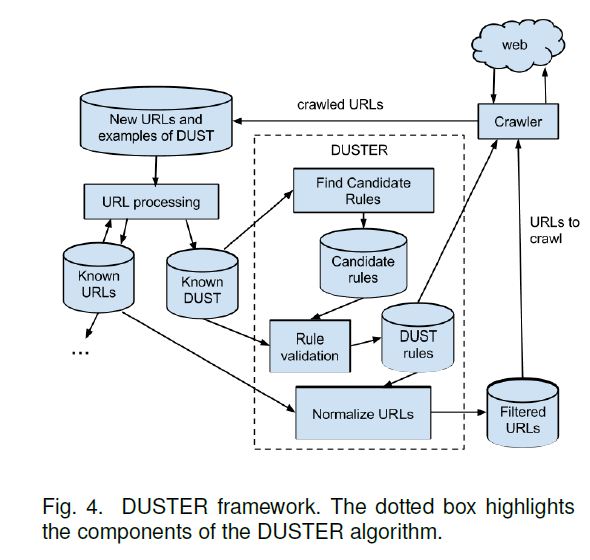
Keyboard **:** 104 Keys

Monitor **:**SVGA (Color)

Mouse **:** Logitech (Recommended**)**

**CHAPTER 5: SYSTEM ARCHITECTURE**

**5.1 System architecture**

****

**\*Fig: System Architecture\***

**DUSTER**

In this section, we describe in detail our solution to avoid the presence of DUST in search engines. Fig. 4 depicts the framework of our algorithm, DUSTER. As we can see in this figure, once a new set of URLs is crawled, it is merged with the already known URLs to 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.

The two main phases of DUSTER are the generation of candidate rules, where a multi-sequence alignment algorithm generates candidate rules from dup-clusters, and rules validation, where DUSTER filters out candidates rules according to their performance in a validation set. These two phases are described in the next sections.

**Phases of Duster**

* Generation of candidate rules –

Multiple sequence alignment algorithm generate candidate rules from dup-clusters

* Rules validation

DUSTER filters out candidate rules according to their performance in a validation set.

**5.3 Modules**

**Step 1:URL Processing**

* Find Candidate rules.
* Find Dust Rules
* Rules Validation

**Step 2:** **Normalize URL**

**Step 3: Filtered URL’S**

**CHAPTER 6: HIGH LEVEL DESIGN**

**6.1 Unified Modeling Design**

Design modeling uses a combination of text and diagrammatic forms to depictthe requirements for data, function and behavior in a way that s relatively easy tounderstand and more importantly straightforward to review for correctness,completeness and consistency. The Unified Modeling Language (UML) is agraphical language for visualization, specifying, constructing and documenting theartifacts of a software intensive system. The UML gives a standard way to writesystems blue prints, covering conceptual things, such as business processes andsystem functions, as well as concrete things, such as classes written in a specificprogramming language, database schemas and reusable software components.

Here we have included the following UML diagrams:

* Use Case Diagram
* Activity Diagram
  + 1. **Use Case Diagram**

A use case diagram is a type of behavioral diagram defined by the UnifiedModeling Language (UML) and created from a Use-case analysis. Its purpose is topresent a graphical overview of the functionality provided by a system in terms ofactors, their goals (represented as use cases), and any dependencies between thoseuse cases. The main purpose of a use case diagram is to show what system functions are performed for which actors. Roles of the actors in the system can be depicted.

It represents the activity or functionality of the system. It is the dialogue between the user and system.

**Actor**: It represents the user, machine which gives the input or gets the output to and from the system. An actor which provides inputs are called as **primary actors** and actors which receives the output from the system are called as **secondary actors.**

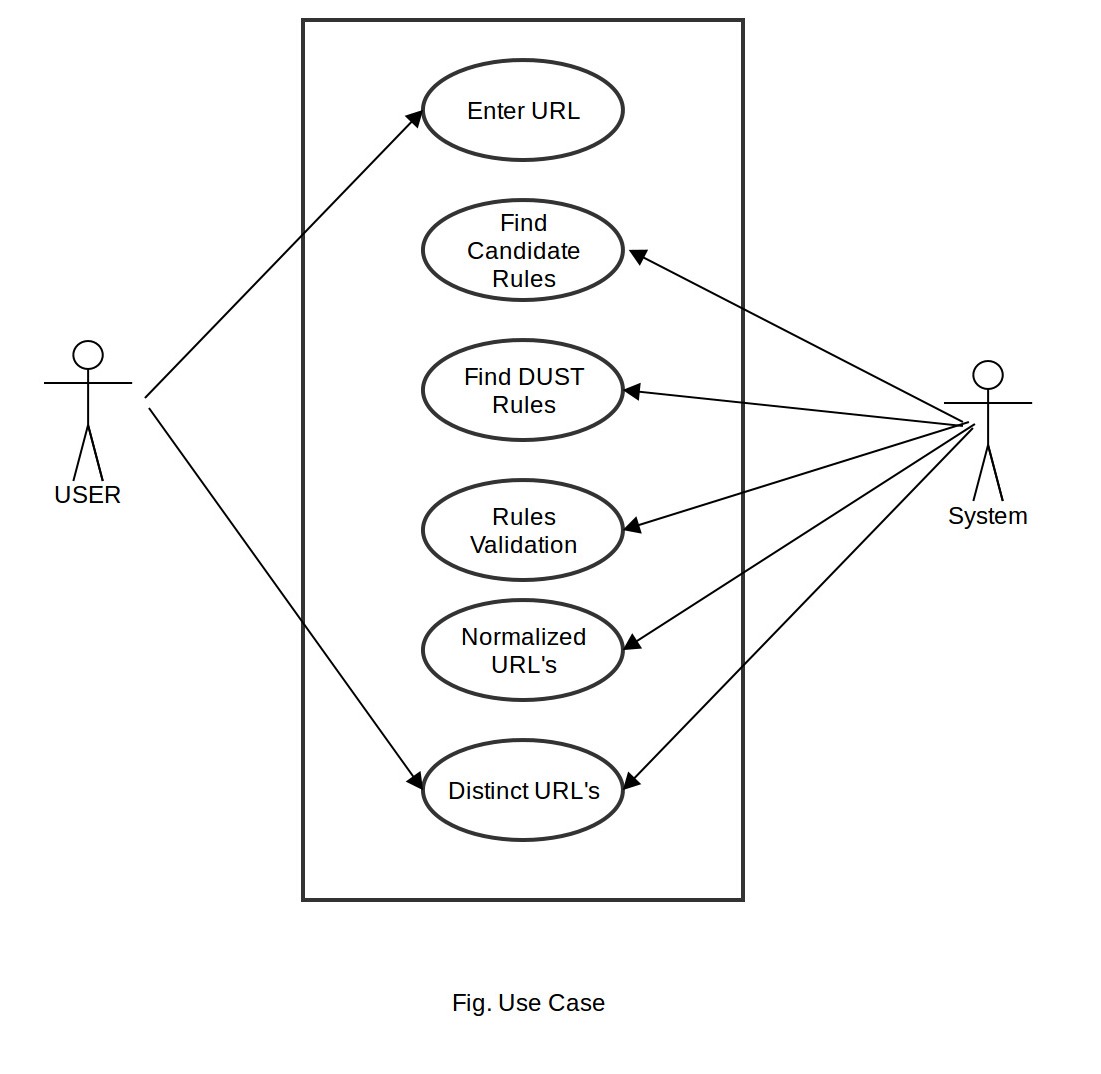
**Association**: connects/ sends messages from actor to the system

**Dependency:** <<includes>> and <<extends>>



**\*Fig: Usecase Diagram notations\***

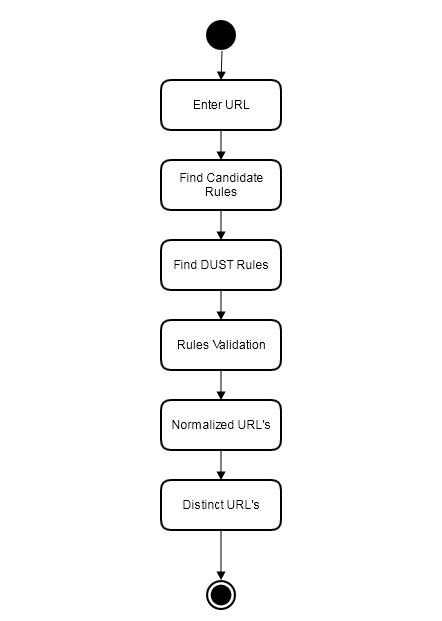
* **Use Case**

****

* + 1. **Activity diagram**

Activity diagrams are graphical representations of [workflows](https://en.wikipedia.org/wiki/Workflow) of stepwise activities and actions with support for choice, iteration and concurrency. In the [Unified Modelling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language), activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control. Activity diagrams may be regarded as a form of [flowchart](https://en.wikipedia.org/wiki/Flowchart).

* **Activity diagram**

****

**\*Fig: Activity diagram\***

**CHAPTER 7: PROJECT PLANNING AND MANAGEMENT**

**7.1 Project Process Management**

The project progresses through three main phases. Initially it starts with the problem definition where the exact goal or the aim objective is decided. It is concerned with what is to be done. The exact problem to be addressed is defined and then the requirements are analyzed. Here the requirements do not only deal with the hardware software resources but it also considers the various inputs at each stage of the project. Then the most important part that is project plan is developed. The plan covers the various tasks, goals, objectives, roles and responsibilities to be carried out etc. The implementation phase is the phase where the actual work is done. It is the central part of the project. The actual designing and coding activities are carried out in this phase.

The data set of various similar URLs is encrypted using the various rules. The rules are applied to find the duplicates and near duplicates. Then Graphical User Interface (GUI) is prepared. The output of this phase is user interface for sending and receiving URLs. Finally testing the application’s portability analysis and the recovery capabilities of the application takes place in this phase.

**7.2 Feasibility Analysis**

Whatever we think need not be feasible. It is wise to think about the feasibility of any problem we undertake. Feasibility is the study of impact, which happens in the organization by the development of a system. The impact can be either positive or negative. When the positives nominate the negatives, then the system is considered feasible. Here the feasibility study can be performed in two ways such as **Technical Feasibility** and **Economical Feasibility.**

* **Technical Feasibility**

The system’s technical feasibility is very high as:

* DUST removal is primary requirement. Resources are widely available.
* Almost every person face the problems occurred due to DUST.
* **Economical Feasibility**

Development of this application is highly economically feasible. The only thing

is to be done is making an environment for the development with an effective supervision. If we are doing so, we can attain the maximum usability of the corresponding resources. Even after the development, the organization will not be in a condition to invest more in the organization. The system uses these basic components for development and testing:

* DUST rule formation.
* Availability of large scale data sets.
* Collection of URLs.
* Duplicate URLs, which are readily available.
* **Time Feasibility**

Time feasibility is mainly concerned with how the time management of the project work is distributed and how much time is given to particular section of the project. It gives information about the time utilization while developing the system project.

This system can be completed easily within the stipulated time and hence is very feasible.

* **Resource Feasibility**

Resource feasibility consists of all the resources which are required for the proposed system. In this section we consider whether the resources required is easily available and its costing is affordable or not. The system requires no extra resources than those which are available.

**7.3 Risk Analysis**

**Project Risk**

The process of identification, analysis and either acceptance or mitigation of uncertainty in investment decision-making is called Risk Management. Essentially, risk management occurs anytime an investor or fund manager analyses and attempts to quantify the potential for losses in an investment and then takes the appropriate action (or inaction) given their investment objectives and risk tolerance. Inadequate risk management can result in severe consequences for companies as well as individuals.

A risk is any factor that may potentially interfere with successful completion of the project. A risk is the recognition that a problem may occur. By recognizing potential problems they can be avoided.

The risks that the system may face are:

* Incompatibility with large scale URLs.
* Transmission risks.
* Mining of similar URLs.

**7.4 Project Management**

The efficient use of resources and time for creating the system is an essential part of project management. Planning forms an integral part. Planning includes the distribution of modules and using software engineering techniques to manage the project.

**Project Plan**

Our project plan schedule for the following year will be as follows:

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Month-Year** | **Work to be done** |
| 1. | June | Study of Multiple sequence alignment method and Basics of URL normalization. |
| 2. | July | Study of JAVA, OpenCV and Matlab |
| 3. | August | Validation of Candidate Rule |
| 4. | September | Experimentation with different data sets. |
| 5. | October | DUST rule formation. |
| 6. | November | DUST rule testing and evaluation. |
| 7. | December | Normalization & Filtering of URLs. |
| 8. | January | Collection and Experimentation of real data type set. |
| 9. | February | Analyze and evaluate method by taking different web collections. |

**CHAPTER 8: CONCLUSION**

In this, we presented DUSTER, a new method to address the DUST problem, the detection of distinct URLs that correspond to pages with duplicate or near-duplicate content. DUSTER learns normalization rules that are very precise in converting distinct URLs which refer the same content to a common canonical form, making it easy to detect them. To achieve this, DUSTER applies a novel strategy based on a full multisequence alignment of training URLs with duplicate content. By analyzing the alignments obtained, accurate and general normalization rules can be generated, as demonstrated in our experiments. We evaluated the method in a set of duplicate URLs extracted from the TREC GOV2 collection and found a reduction in the number of duplicate URLs that is 82% larger than the one achieved by our best baseline. When evaluating a Brazilian web sample, we obtained a gain of 140.74% over the same baseline.

**CHAPTER 9: REFERENCES**

[1] A. Agarwal, H. S. Koppula, K. P. Leela, K. P. Chitrapura, S. Garg,P. K. GM, C. Haty, A. Roy, and A. Sasturkar. URL normalization for de-duplication of web pages. In Proceedings of the 18th ACM conference on Information and knowledge management, CIKM ’09, pages 1987–1990, New York, NY, USA, 2009. ACM

[2] 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.

[3] 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.

[4] 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.

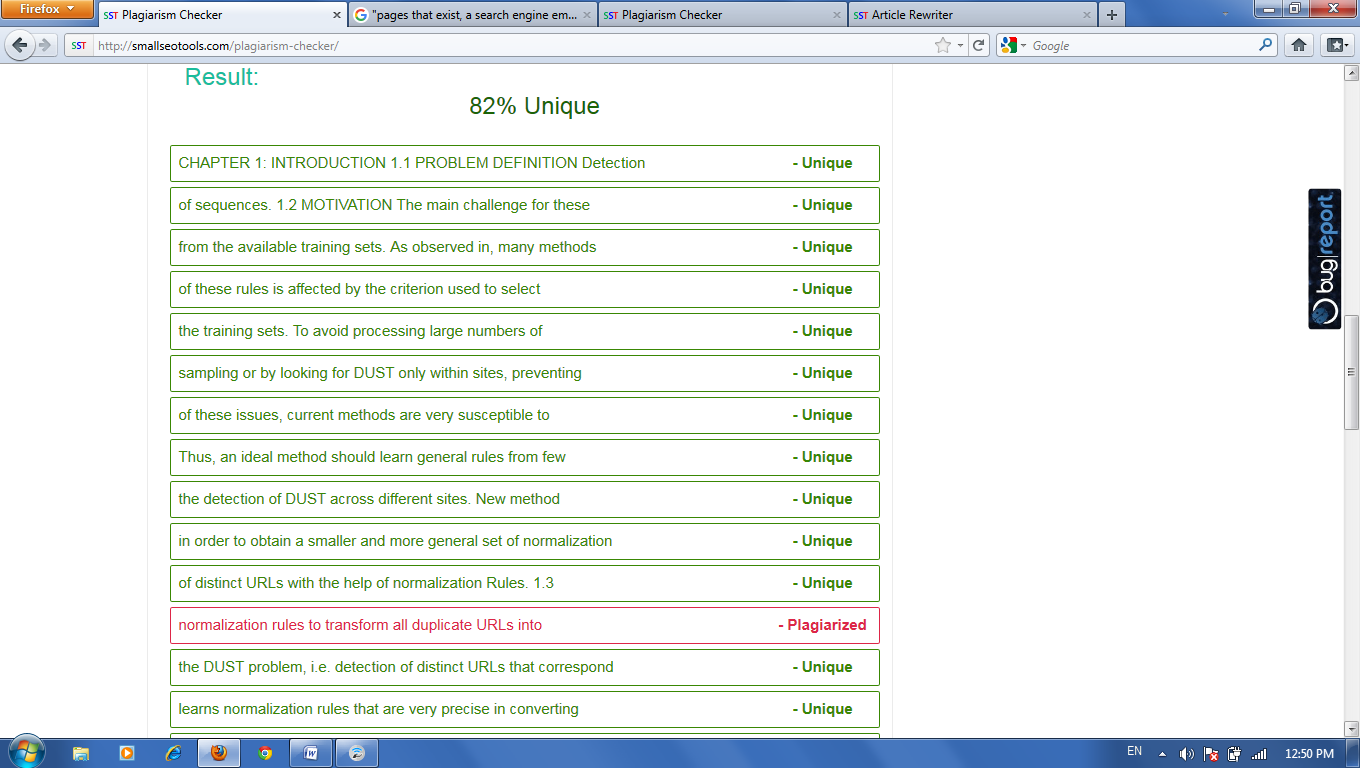
[5] T. Lei, R. Cai, J.-M. Yang, Y. Ke, X. Fan, and L. Zhang. A pattern tree-based approach to learning URL normalization rules. In Proceedings of the 19th international conference on World wide web, WWW ’10, pages 611–620, New York, NY, USA, 2010. ACM.

**APPENDICES**

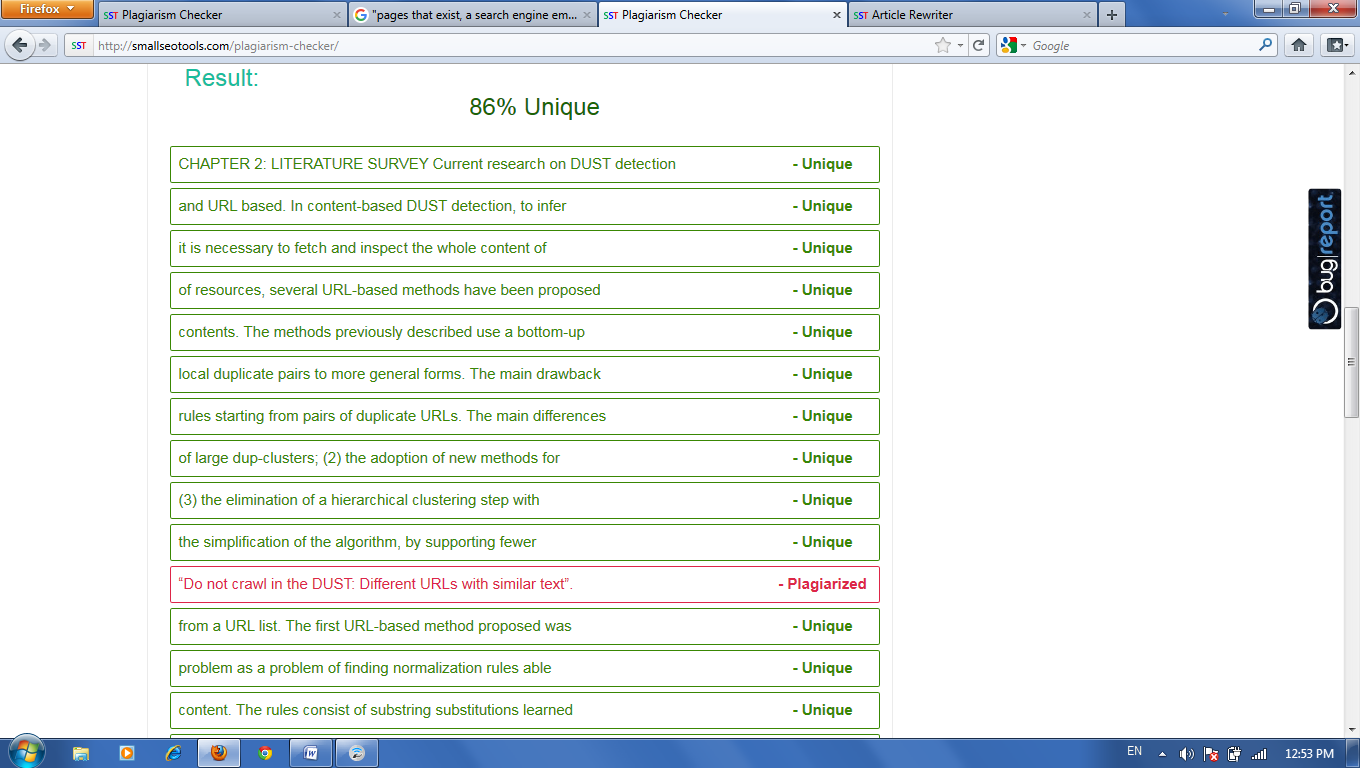
**A. BASE PAPER**

**B. PLAGARISM RESULTS**

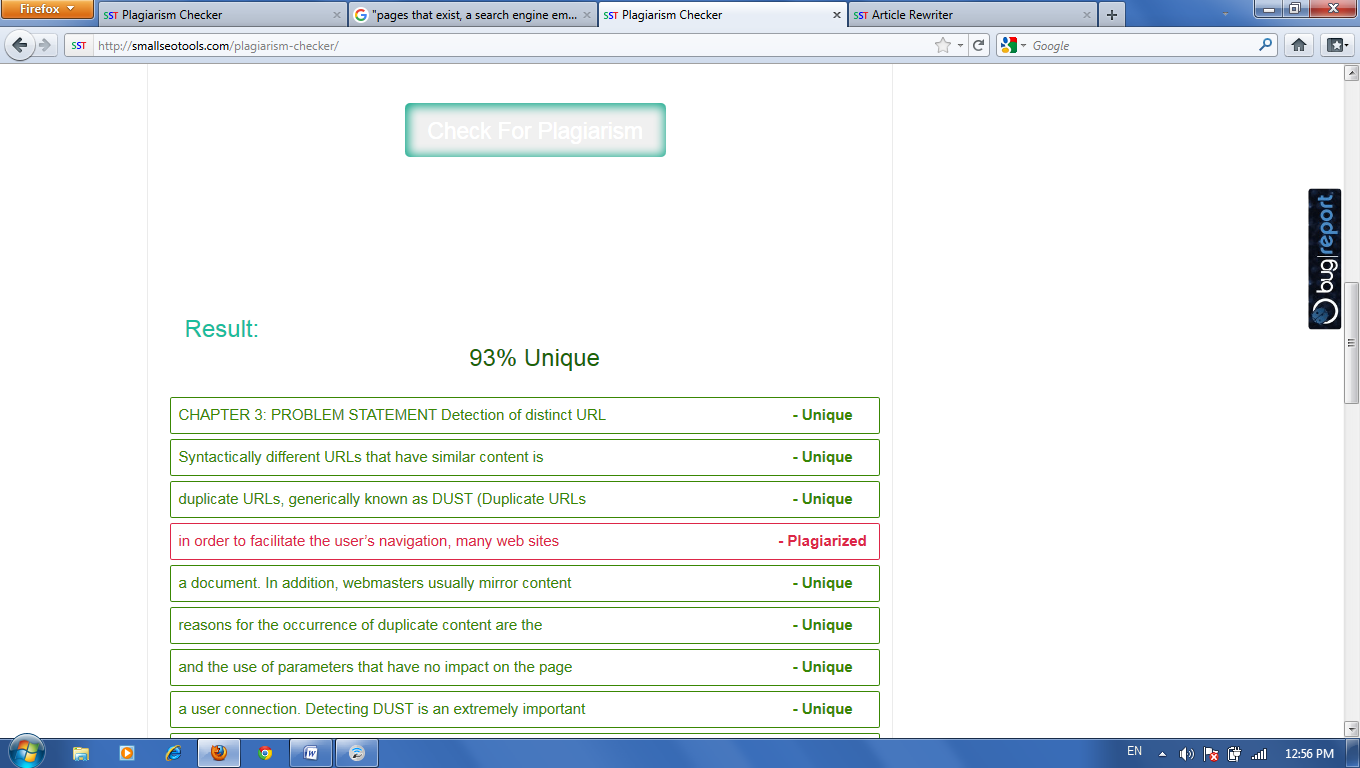
* **Chapter 1 (Introduction)**

****

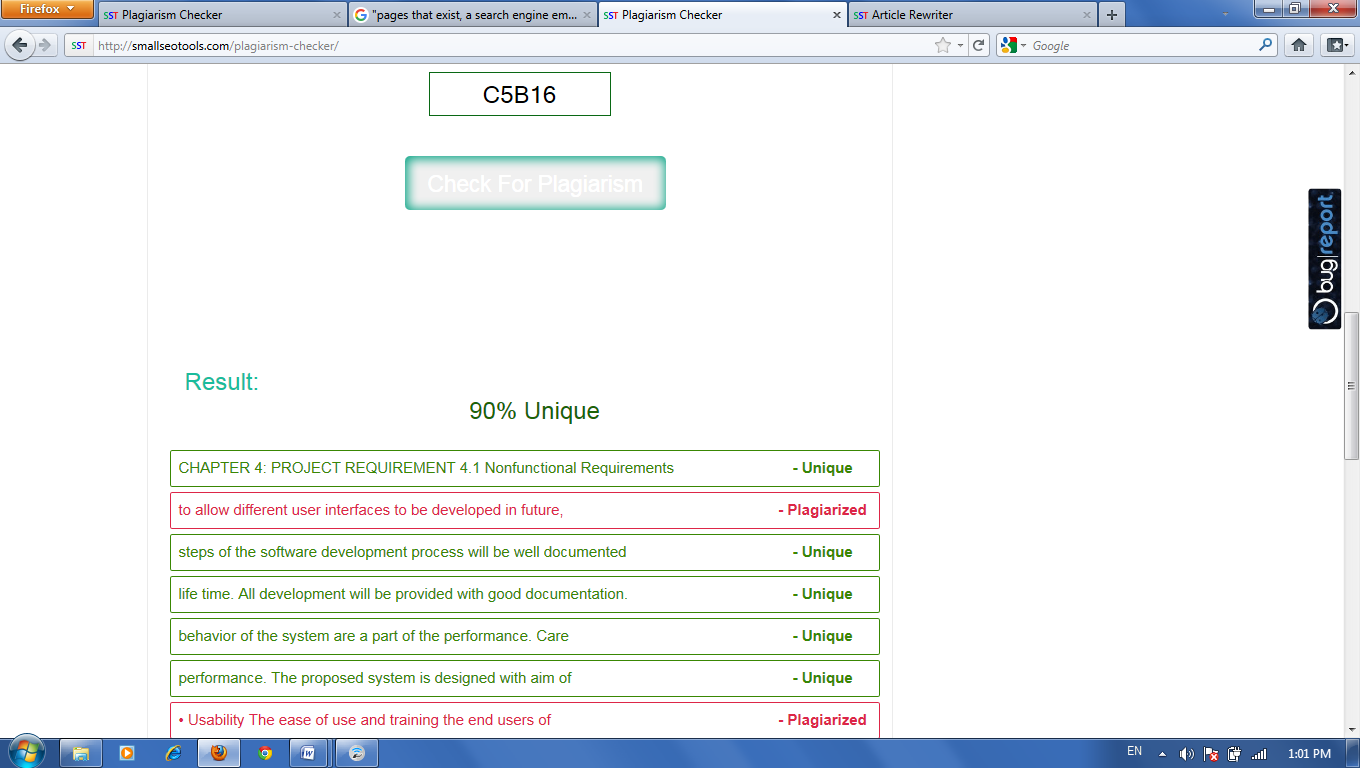
* **Chapter 2( Literature Survey)**

****

* **Chapter 3 ( Problem Statement)**

****

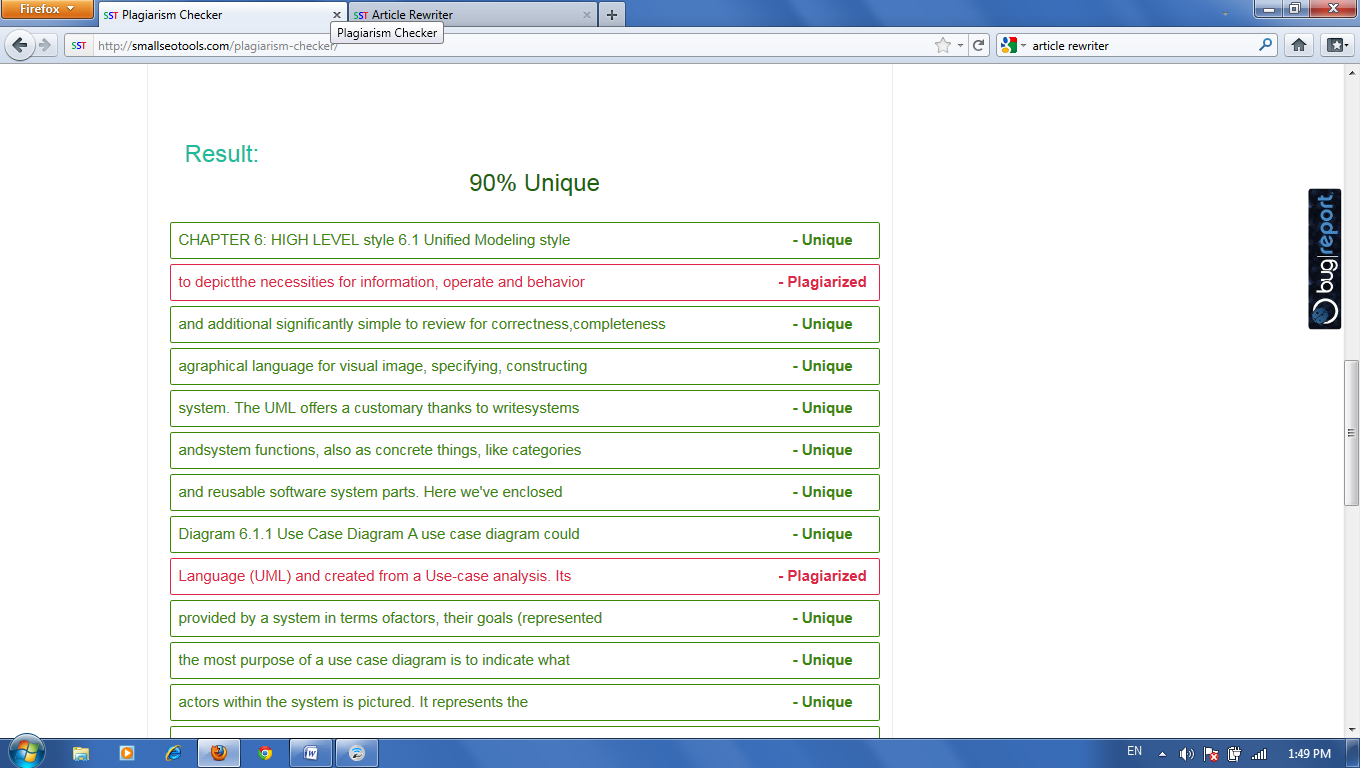
* **Chapter 4 ( Project Requirement )**

****

* **Chapter 5 (System Architecture)**

****

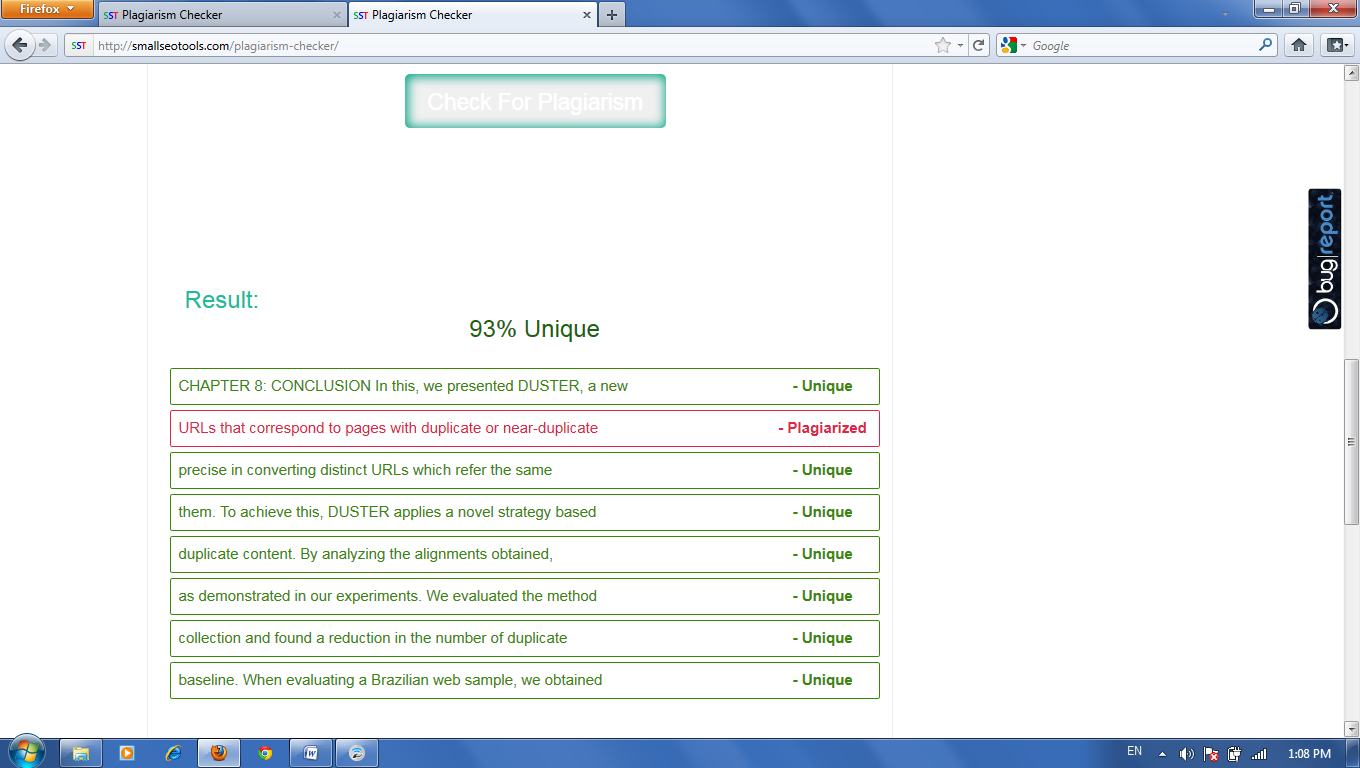
* **Chapter 6( High Level Design)**

****

* **Chapter 7 ( Project Planning and Management )**

****

* **Chapter 8( Conclusion)**

****