**Progressive Duplicate Detection**

**A PROJECT WORK**

**Submitted in fulfilment of the award of Degree of**

**BACHELOR OF TECHONOLGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted**

**By**

**Aditya Goindi 14951A0501**

**Chigurupati Harshitha 14951A0512**

**Akhil Uday D 14951A0505**

**Under the Supervision**

**of**

**Dr. K Suvarchala**

Professor, CSE



**COMPUTER SCIENCE AND ENGINEERING**

**INSTITUTE OF AERONAUTICAL ENGINEERING**

**(Autonomous)**

Dundigal, Hyderabad - 500043, Telangana

**April, 2018**

**INSTITUTE OF AERONAUTICAL ENGINEERING**

**(Autonomous)**

Dundigal, Hyderabad - 500043, Telangana

**COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that work embodied in this dissertation entitled “**Progressive Duplicate Detection”** being submitted by

**Aditya Goindi 14951A0501**

**Chigurupati Harshitha 14951A0512**

**Akhil Uday D 14951A0505**

for partial fulfilment of the requirements for the award of **Bachelor of Technology** in **Computer Science and Engineering** discipline to Institute of Aeronautical Engineering, Dundigal, Hyderabad during the academic year 2017-2018 as a record of bona fide work, undertaken by her the supervision of the undersigned.

**Approved and Supervised by**

**Dr. K Suvarchala**

Professor, CSE

**Forwarded by**

**Dr. L V Narasimha Prasad Dr. K Rajendra Prasad**

**Principal**  **Professor and Head**

**EXTERNAL EXAMINER**

**INSTITUTE OF AERONAUTICAL ENGINEERING**

**(Autonomous)**

Dundigal, Hyderabad - 500043, Telangana

**COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

We the students of **Bachelor of Technology in Computer Science and Engineering**, **session: 2017 - 2018**, Institute of Aeronautical Engineering, Dundigal, Hyderabad, hereby declare that the work presented in this project work entitled **“Progressive Duplicate Detection”** is the outcome of our own bonafide work and is correct to the best of our knowledge. This work follows engineering ethics and contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

**Aditya Goindi 14951A0501**

**Chigurupati Harshitha 14951A0512 Akhil Uday D 14951A0505**

**Date:**

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whole constant guidance and encouragement crown all the efforts with success. We thank our college management and respected **Sri M. Rajashekar Reddy**, Chairman, IARE, Dundigal for providing us the necessary infrastructure to carry out the project work.

We express our sincere thanks to **Dr. L.V. Narasimha Prasad**, Professor, Principle, and who has been a great source of information for our work, and **Dr. K. Rajendra Prasad**, Professor and Head, Department of CSE, for extending his support to carry on this project work.

We are especially thankful to our internal supervisor **Dr. k Suvarchala**, Professor, Department of CSE, for his internal support and professionalism who helped us in shaping the project into successful one.

We take this opportunity to express our thanks to one and all who directly or indirectly helped us in bringing this effort to present form.

**Aditya Goindi 14951A0501**

**Chigurupati Harshitha 14951A0512**

**Akhil Uday D 14951A0505**

**ABSTRACT**

Duplicate detection is the process of identifying multiple representations of same real world entities. Today, duplicate detection methods need to process ever larger datasets in ever shorter time: maintaining the quality of a dataset becomes increasingly difficult. We present two novel, progressive duplicate detection algorithms that significantly increase the efficiency of finding duplicates if the execution time is limited. They maximize the gain of the overall process within the time available by reporting most results much earlier than traditional approaches. Comprehensive experiments show that our progressive algorithms can double the efficiency over time of traditional duplicate detection and significantly improve upon related work. Two approaches that we follow are progressive duplicate detection algorithms namely progressive sorted neighbourhood method (PSNM), which performs best on small and almost clean datasets, and progressive blocking (PB), which performs best on large and very dirty datasets. Both enhance the efficiency of duplicate detection even on very large datasets. In comparison to traditional duplicate detection, progressive duplicate detection satisfies two conditions viz. Improved early quality and same eventual quality. We introduce a concurrent progressive approach for the multi-pass method and adapt an incremental transitive closure algorithm that together forms the first complete progressive duplicate detection workflow. We make these findings through our project. A user has only limited, maybe unknown time for data cleansing and wants to make best possible use of it. Then, simply start the algorithm and terminate it when needed. The result size will be maximized. A user has little knowledge about the given data but still needs to configure the cleansing process. Then, the progressive algorithm choose window/block sizes and keys automatically to detect duplicates.

**TABLE OF CONTENTS**

1. Introduction 1

1.1 Introduction 1

1.2 Existing system 1

1.3 Proposed system 2

1.3.1 Proposed system 2

1.3.2 Advantages of Proposed system 2

1.4 Problem Definition 2

1.5 Software and Hardware Requirements 3

1.5.1 Hardware requirements 3

1.5.2 Software requirements 3

1.6 Overview of the report 3

2. Literature Review 4

2.1 Neighborhood blocking with MapReduce 4

2.2 Indexing techniques for scalable record linkage 4

2.3 Windowing algorithms for duplicate detection 4

2.4 Clustering algorithms in duplicate detection 5

2.5 Data cleansing and the merge/purge problem 5

3. System Design 7

3.1 Architecture 7

3.2 Modules

3.3 Data Flow Diagram 7

3.4 UML diagrams 9

3.4.1 Class diagram 10

3.4.2 Use case diagram 12

3.4.3 Sequence diagram 13

3.4.4 Activity diagram 14

3.4.5 Component Diagram 15

3.4.6 Deployment Diagram 16

4. Implementation 17

4.1 Implementation of Code 17

4.2 Testing 22

4.2.1 Unit testing 22

4.2.2 Integration testing 24

4.2.3 Acceptance testing 26

4.2.4 Test Cases

5. Result analysis 27

5.1 Output Screenshots 28

6. Conclusion and Future Work 38 References 40

**LIST OF FIGURES**

Figure 1.1 Structure of Data Mining 1

Figure 3.1 Architecture 7

Figure 3.2 Data Flow Diagram 8

Figure 3.3.1 Class diagram 9

Figure 3.3.2 Use case diagrams 10

Figure 3.3.3 Sequence diagram 12

Figure 3.3.4 Activity diagram 14

Figure 3.3.5 Deployment Diagram 15

Figure 3.3.6 Component Diagram 16

Figure 5.1 Home screen 28

Figure 5.2 Menu Screen 29

Figure 5.3 Data Loader 29

Figure 5.4 Ordered Data 30

Figure 5.5 Selection Window 30

Figure 5.6 Ordered Data 31

Figure 5.7 Data Seperation 31

Figure 5.8 Selection Window 32

Figure 5.9 DataSet Detection 32

Figure 5.10 Duplicate Detection 1 33

Figure 5.11 Duplicate Detection 2 34

Figure 5.12 Clustering 34

Figure 5.13 Full Data 35

Figure 5.14 Blocking Applied 35

Figure 5.15 Blocks 36

Figure 5.16 Representation Window 36

**CHAPTER 1 - INTRODUCTION**

* 1. **INTRODUCTION**

**What is Data Mining?**

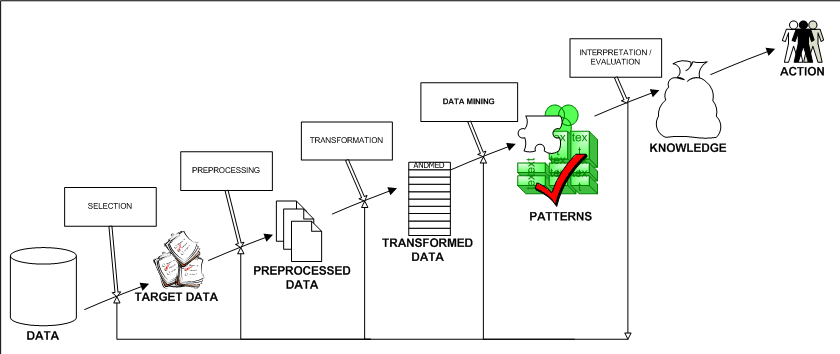


Figure 1.1 Structure of Data Mining

Generally, data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.

* 1. **EXISTING SYSTEM**

Much research on duplicate detection, also known as entity resolution and by many other names, focuses on pairselection algorithms that try to maximize recall on the one hand and efficiency on the other hand. The most prominent algorithms in this area are Blocking and the sorted neighborhood method (SNM).

Xiao et al. proposed a top-k similarity join that uses a special index structure to estimate promising comparison candidates. This approach progressively resolves duplicates and also eases the parameterization problem.

Pay-As-You-Go Entity Resolution by Whang et al. introduced three kinds of progressive duplicate detection techniques, called “hints”

* 1. **PROPOSED SYSTEM**

**1.3.1 PROPOSED SYSTEM**

In this work, however, we focus on progressive algorithms, which try to report most matches early on, while possibly slightly increasing their overall runtime. To achieve this, they need to estimate the similarity of all comparison candidates in order to compare most promising record pairs first.We propose two novel, progressive duplicate detection algorithms namely progressive sorted neighborhood method (PSNM), which performs best on small and almost clean datasets, and progressive blocking (PB), which performs best on large and very dirty datasets. Both enhance the efficiency of duplicate detection even on very large datasets.

We propose two dynamic progressive duplicate detection algorithms, PSNM and PB, which expose different strengths and outperform current approaches.We introduce a concurrent progressive approach for the multi-pass method and adapt an incremental transitive closure algorithm that together forms the first complete progressive duplicate detection workflow.We define a novel quality measure for progressive duplicate detection to objectively rank the performance of different approaches.We exhaustively evaluate on several real-world datasets testing our own and previous algorithms

**1.3.2 ADVANTAGES OF PROPOSED SYSTEM**

Improved early quality

Same eventual quality

Our algorithms PSNM and PB dynamically adjust their behavior by automatically choosing optimal parameters, e.g., window sizes, block sizes, and sorting keys, rendering their manual specification superfluous. In this way, we significantly ease the parameterization complexity for duplicate detection in general and contribute to the development of more user interactive applications.

* 1. **PROBLEM DEFINITION**

A user has only limited, maybe unknown time for data cleansing and wants to make best possible use of it. Then, simply start the algorithm and terminate it when needed. The result size will be maximized.

A user has little knowledge about the given data but still needs to configure the cleansing process.A user needs to do the cleaning interactively to, for instance, find good sorting keys by trial and error. Then, run the progressive algorithm repeatedly; each run quickly reports possibly large results.

All presented hints produce static orders for the comparisons and miss the opportunity to dynamically adjust the comparison order at runtime based on intermediate results.

**1.5 SOFTWARE AND HARDWARE REQUIREMENTS**

**1.5.1 HARDWARE REQUIREMENTS:**

System : Pentium IV 2.4 GHz.

Hard Disk        : 40 GB.

Floppy Drive : 1.44 Mb.

Monitor : 15 VGA Colour.

Mouse : Logitech.

Ram : 512 Mb.

**1.5.2 SOFTWARE REQUIREMENTS:**

Operating system : Windows XP.

Coding Language :  JAVA

Data Base :  MYSQL

**1.6 OVERVIEW OF REPORT**

In the next chapter, the collection of ideas behind the project development is discussed. Chapter 3, Methodology explains the designing of the system via UML diagrams, the algorithms used in the software modules and the software modules themselves. Chapter 4, Implementation, is all about the architecture of the system and why such architecture was chosen, the APIs required for authentication. In the last chapter, the results of the application are analysed and the future directions for improving the system is discussed

**CHAPTER 2 – LITERATURE REVIEW**

**2.1 Progressive Duplication Detection Approach**

**AUTHORS:**  Thorsten Papenbrock, Arvid Heise, and Felix Naumann

Duplicate detection is the process of identifying multiple representations of same real world entities. Today, duplicate detection methods need to process ever larger datasets in ever shorter time: maintaining the quality of a dataset becomes increasingly difficult. We present two novel, progressive duplicate detection algorithms that significantly increase the efficiency

of finding duplicates if the execution time is limited: They maximize the gain of the overall process within the time available by reporting most results much earlier than traditional approaches.

**2.2 A survey of indexing techniques for scalable record linkage and deduplication**

**AUTHORS:** P. Christen

Record linkage is the process of matching records from several databases that refer to the same entities. When applied on a single database, this process is known as deduplication. Increasingly, matched data are becoming important in many application areas, because they can contain information that is not available otherwise, or that is too costly to acquire. Removing duplicate records in a single database is a crucial step in the data cleaning process, because duplicates can severely influence the outcomes of any subsequent data processing or data mining. With the increasing size of today's databases, the complexity of the matching process becomes one of the major challenges for record linkage and deduplication. In recent years, various indexing techniques have been developed for record linkage and deduplication. They are aimed at reducing the number of record pairs to be compared in the matching process by removing obvious nonmatching pairs, while at the same time maintaining high matching quality. This paper presents a survey of 12 variations of 6 indexing techniques. Their complexity is analyzed, and their performance and scalability is evaluated within an experimental framework using both synthetic and real data sets. No such detailed survey has so far been published.

**2.3  A generalization of blocking and windowing algorithms for duplicate detection**

**AUTHORS:**  U. Draisbach and F. Naumann

Duplicate detection is the process of finding multiple records in a dataset that represent the same real-world entity. Due to the enormous costs of an exhaustive comparison, typical algorithms select only promising record pairs for comparison. Two competing approaches are blocking and windowing. Blocking methods partition records into disjoint subsets, while windowing methods, in particular the Sorted Neighborhood Method, slide a window over the sorted records and compare records only within the window. We present a new algorithm called Sorted Blocks in several variants, which generalizes both approaches. To evaluate Sorted Blocks, we have conducted extensive experiments with different datasets. These show that our new algorithm needs fewer comparisons to find the same number of duplicates.

**2.4  Framework for evaluating clustering algorithms in duplicate detection**

**AUTHORS:**  O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller

The presence of duplicate records is a major data quality concern in large databases. To detect duplicates, entity resolution also known as duplication detection or record linkage is used as a part of the data cleaning process to identify records that potentially refer to the same real-world entity. We present the Stringer system that provides an evaluation framework for understanding what barriers remain towards the goal of truly scalable and general purpose duplication detection algorithms. In this paper, we use Stringer to evaluate the quality of the clusters (groups of potential duplicates) obtained from several unconstrained clustering algorithms used in concert with approximate join techniques. Our work is motivated by the recent significant advancements that have made approximate join algorithms highly scalable. Our extensive evaluation reveals that some clustering algorithms that have never been considered for duplicate detection, perform extremely well in terms of both accuracy and scalability.

**2.5 Real-world data is dirty: Data cleansing and the merge/purge problem**

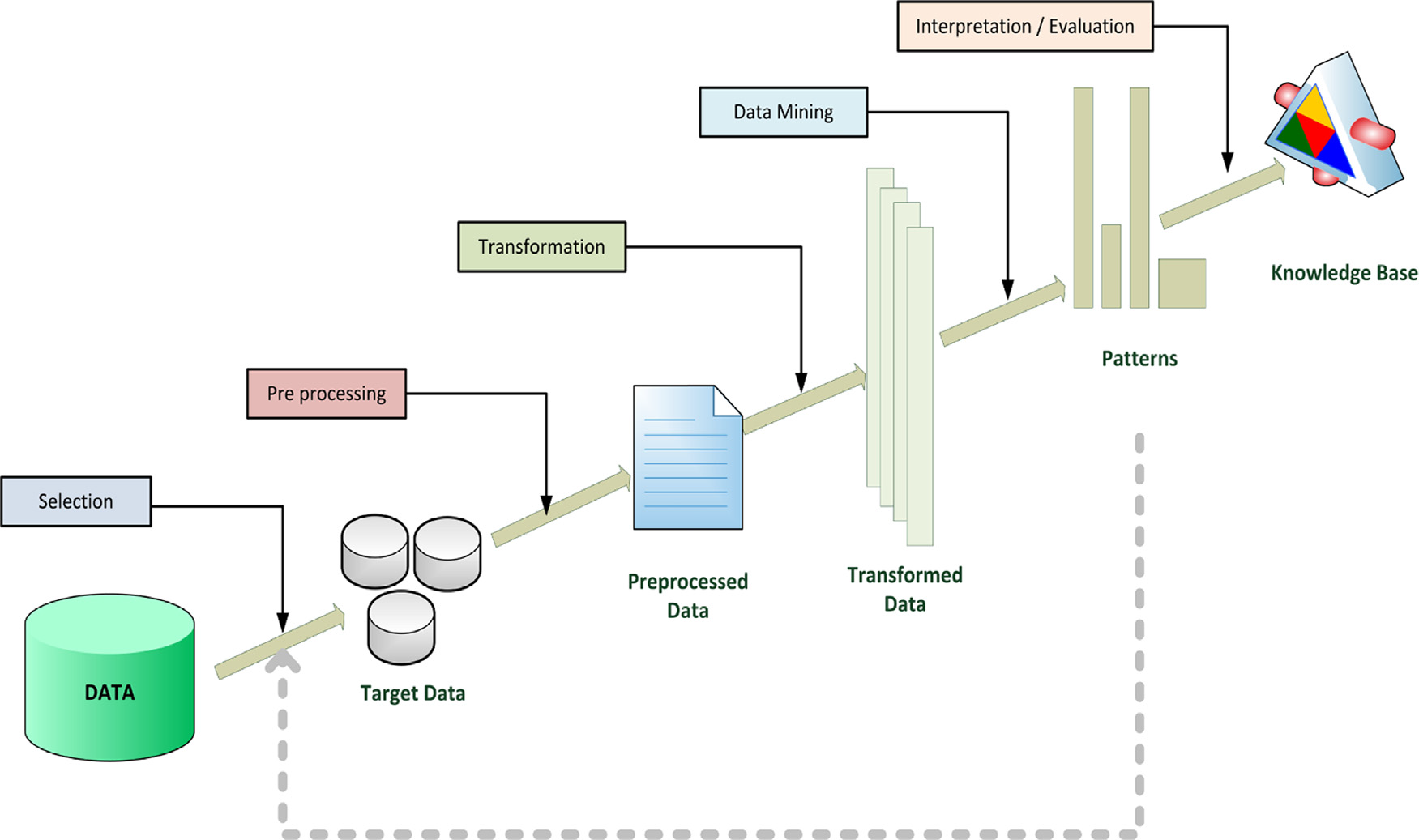
**AUTHORS:**  M. A. Hernandez and S. J. Stolfo,

The problem of merging multiple databases of information about common entities is frequently encountered in KDD and decision support applications in large commercial and government organizations. The problem we study is often called the Merge/Purge problem and is difficult to solve both in scale and accuracy. Large repositories of data typically have numerous duplicate information entries about the same entities that are difficult to cull together without an intelligent “equational theory” that identifies equivalent items by a complex, domain-dependent matching process. We have developed a system for accomplishing this Data Cleansing task and demonstrate its use for cleansing lists of names of potential customers in a direct marketing-type application. Our results for statistically generated data are shown to be accurate and effective when processing the data multiple times using different keys for sorting on each successive pass. Combing results of individual passes using transitive closure over the independent results, produces far more accurate results at lower cost. The system provides a rule programming module that is easy to program and quite good at finding duplicates especially in an environment with massive amounts of data. This paper details improvements in our system, and reports on the successful implementation for a real-world database that conclusively validates our results previously achieved for statistically generated data.

**CHAPTER 3 – SYSTEM DESIGN**

Systems design implies a systematic approach to the design of a system. It may take a bottom-up or top-down approach, but either way the process is systematic wherein it takes into account all related variables of the system that needs to be created—from the architecture, to the required hardware and software, right down to the data and how it travels and transforms throughout its travel through the system. Systems design then overlaps with systems analysis, systems engineering and systems architecture.The systems design approach first appeared right before World War II, when engineers were trying to solve complex control and communications problems. They needed to be able to standardize their work into a formal discipline with proper methods, especially for new fields like information theory, operations research and computer science in general.

**3.1 SYSTEM ARCHITECTURE:**



**Data Separation**

**Duplicate Detection**

Figure 3.1 Architecture

**3.2 MODULES:**

Dataset Collection

Preprocessing Method

Data Separation

Duplicate Detection

Quality Measures

**MODULES DESCSRIPTION:**

**Dataset Collection:**

To collect and/or retrieve data about activities, results, context and other factors. It is important to consider the type of information it want to gather from your participants and the ways you will analyze that information. The data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable. after collecting the data to store the Database.

**Preprocessing Method:**

Data Preprocessing or Data cleaning, Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data. And also used to removing the unwanted data. Commonly used as a preliminary data mining practice, data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the user.

**Data Separation:**

After completing the preprocessing, the data separation to be performed. The blocking algorithms assign each record to a fixed group of similar records (the  
blocks) and then compare all pairs of records within these  
groups. Each block within the block comparison matrix represents the comparisons of all records in one block with all records in another block, the equidistant blocking, all blocks have the same size.

**Duplicate Detection:**

Theduplicate detection rules set by the administrator, the system alerts the user about potential duplicates when the user tries to create new records or update existing records. To maintain data quality, you can schedule a duplicate detection job to check for duplicates for all records that match a certain criteria. You can clean the data by deleting, deactivating, or merging the duplicates reported by a duplicate detection.

**Quality Measures:**

The quality of these systems is, hence, measured using a cost-benefit calculation. Especially for traditional duplicate detection processes, it is difficult to meet a budgetlimitation, because their runtime is hard to predict. By delivering  
as many duplicates as possible in a given amount of time, progressive processes optimize the cost-benefit ratio. In manufacturing, a measure of excellence or a state of being free from defects, deficiencies and significant variations. It is brought about by strict and consistent commitment to certain standards that achieve uniformity of a product in order to satisfy specific customer or user requirements.

**3.3 DATA FLOW DIAGRAM:**

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.

1. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
2. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
3. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

Input

Pre-processing

Data Separation

Cluster Block Size

Duplicate Detection

Figure 3.2 Data Flow Diagram

**3.4 UML DIAGRAMS**

Design is the place where quality is fostered in software development. Design is the only way that we can accurately translate a user’s requirements into a finished software product or system. Software design serves as the foundation for all software engineers and software maintenance steps that follow. Without design we risk building an unstable design one that may be difficult to test, and one whose quality cannot be accessed until late in the software engineering process.

We seem to understand complexity better when it is displayed to us visually as opposed to written textually. By producing visual models of a system, we can show how system works or several levels. We can model and the interaction between the users and the system.

There are many problem-solving paradigms or models in Computer Science, which is the study of algorithms and data. There are four problem-solving model categories: imperative, functional, declarative and object-oriented languages (OOP).  In object-oriented languages, algorithms are expressed by defining ‘objects’ and having the objects interact with each other. Those objects are things to be manipulated and they exist in the real world. They can be buildings, widgets on a desktop, or human beings.

**3.4.1 Class Diagram**

Class diagrams are the most common diagrams found in the modelling object-oriented systems. A class diagrams shows a set of classes, interfaces and collaborations and their relationships. Graphically a class diagram is a collection of arcs and vertices.

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of execution.

Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

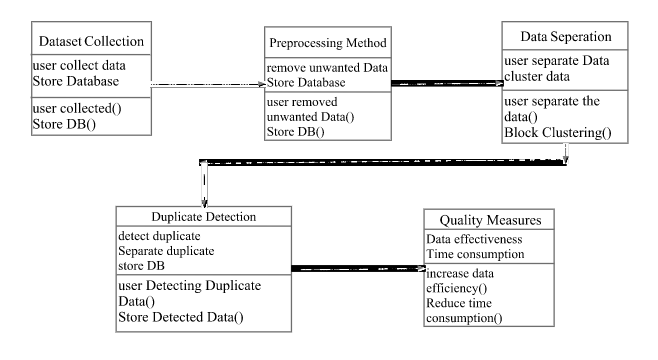


Figure 3.3.1 Class diagram

**3.4.2 Use case Diagram**

Usecase diagram are one of the five diagrams in the UML for modelling the dynamic aspects of systems . A use case is a functionality the users need from the system. A use case diagram depicts the relationships among the actors and use cases. It is usually used for requirements analysis.

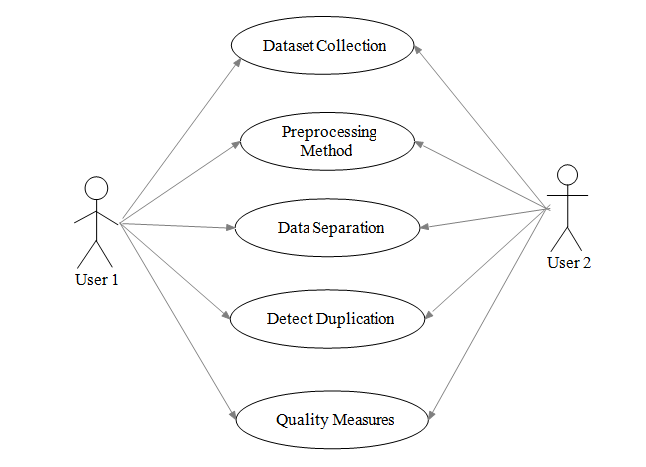


Figure 3.3.2 Use case diagram

**3.4.3 Sequence Diagram**

A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline.

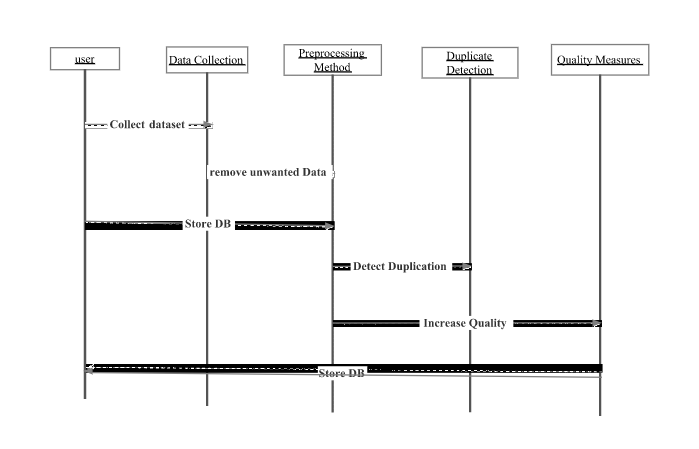


Figure 3.3.3 Sequence diagram

**3.4.4 Activity Diagram**

Activity diagram is another diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc. activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

Dataset

Collection

Preprocessing Method

Cluster Block Size

Duplicate Detection

Improve efficiency

Activity

Quality Measures



Figure 3.3.4 Activity diagram

**3.4.5 Component Diagram**

Purpose of ComponentDiagrams. Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

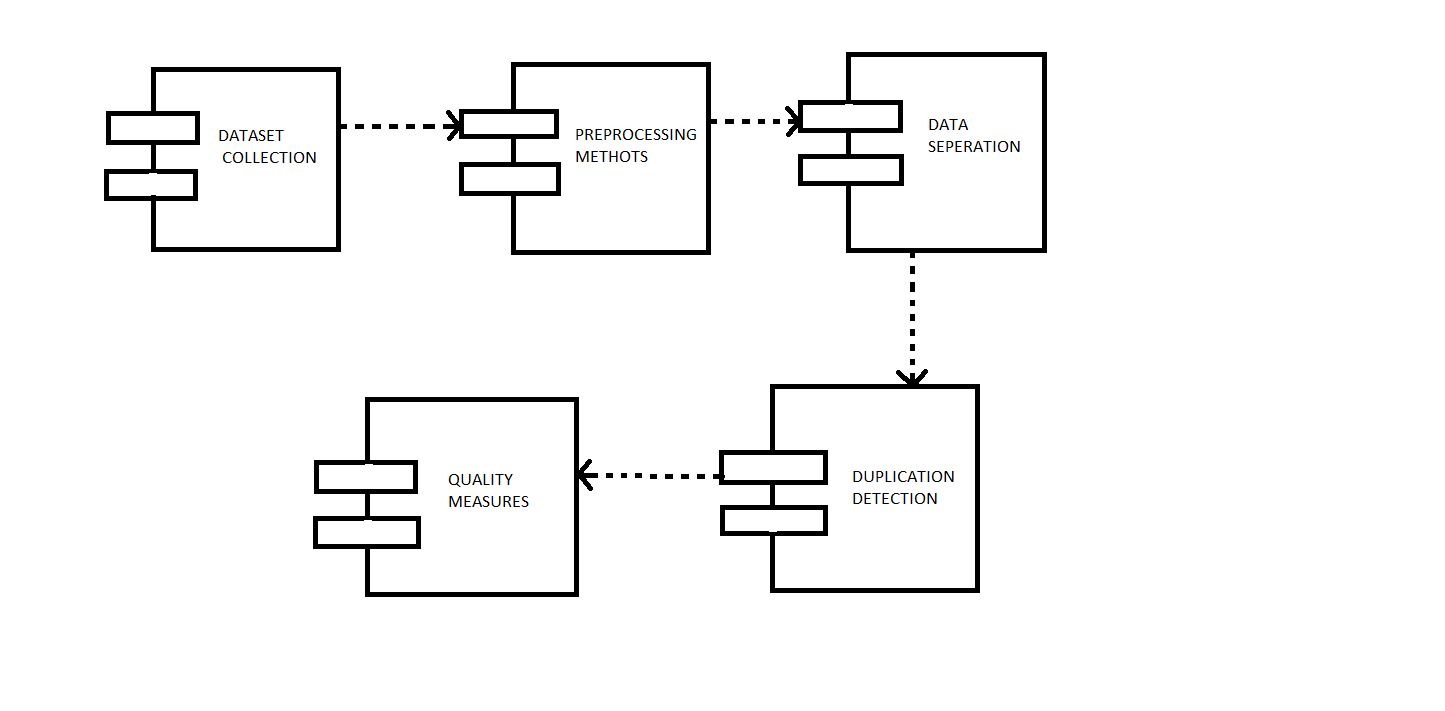


Fig 3.3.6 Component Diagram

**3.4.6 Deployment Diagram**

Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets. Arti facts represent concrete elements in the physical world that are the result of a development process.

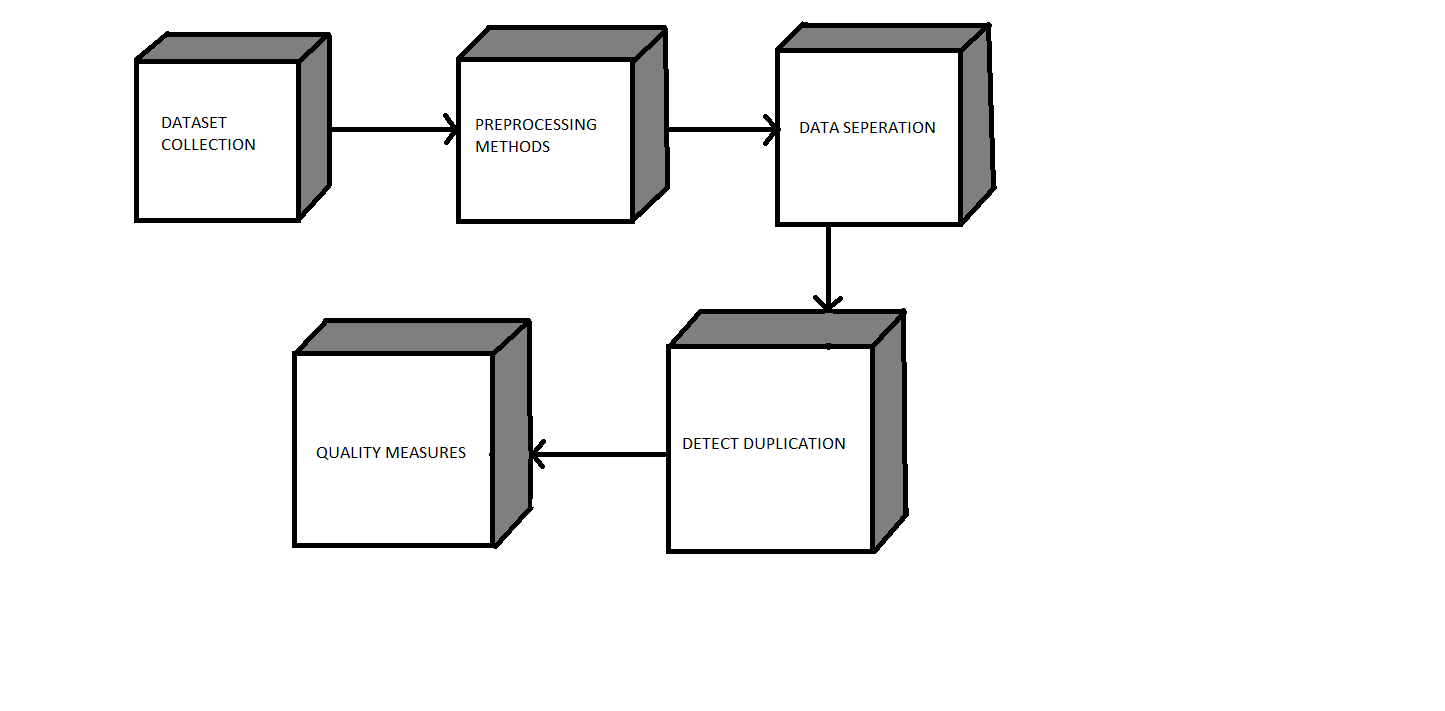


Fig 3.3.5 Deployment Diagram

**CHAPTER 4 – IMPLEMENTATION**

**4.1 IMPLEMENTATION OF CODE**

Input csv file is given as an input to the dataset loader class. After the input file successfully loads for detection of duplicates it is then preprocessed to remove data which is not required. Then the data is chronologically ordered and viewed. Now the term for its duplication detection is taken in as input by the Evaluation.java class. The data is then divided into blocks for progressive detection of data. Each block of data is now given as an input to Progressive.java class and finally the duplicates are detected.

Database is connected to java using the code. DButils.java is the source file for connecting to the database containing datasets that we use.

**DButils.java**

package progressive;

import java.sql.ResultSet;

import javax.swing.table.TableModel;

class DbUtils {

static TableModel resultSetToTableModel(ResultSet rs) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

}

**Evaluation.java**

The evaluation is a sub step of preprocessing of the raw dataset provided as an input to detect duplicates progressively. The similarity between records is measured to arrange the look alike record together.

public static void main(final String[] args) {

final Evaluation p = new Evaluation("EFFECTIVENESS");

p.pack();

RefineryUtilities.centerFrameOnScreen(p);

p.setVisible(true);

}

**Progressive.java**

The below code is used to detect duplicates progressively as the block splitting is used in detection of duplicates

package progressive;

@author admin

public class Progressive {

@param args the command line arguments

public static void main(String[] args) {

// TODO code application logic here

}

}

**4.2 TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**TYPES OF TESTS**

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**4.2.1 Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases. Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

* Dataset loader module is working and taking in csv file as an input.
* Pre-processor module processes the data and works successfully.
* Data seperation module divides the data into given numbers of blocks.
* Duplication Detection module is working.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

# 4.2.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error. Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

All the dataset loader , pre-processing methods, data separation, duplication detection and quality measure modules integrates with each other properly

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**4.2.3 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**4.2.4 Test Cases**

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application.

|  |  |
| --- | --- |
| **Figure Name** | **Figure Number** |
| 5.1. Loading the dataset   1. On the main menu. 2. Click Browse. 3. Select open. 4. Csv File containing datasets is selected. | Dataset loaded – PASSED |
| 5.2. Data is loaded | Dataset loaded – PASSED |
| 5.3 Data is ordered chronologically | Dataset ordered – PASSED |
| 5.4 PSNB and SNB input data | Input csv file is accepted |
| 5.5 Term for Duplication detection is selected | PASSED |
| 5.6 Data separation Window  1. Block size as input.  2.The dataset is divided into blocks of data. | Block size given as input - PASSED |
| 5.7 Parent data displayed and duplicates are detected | Viewed - PASSED |
| 5.8 Block values calculated and displayed | Displayed |
| 5.9 Efficiency, Duplicates graph representation | PASSED |

**CHAPTER 5 – RESULT ANALYSIS**

**5.1 OUTPUT SCEERNSHOTS**

The screenshots of our implemented application are taken and are shown as follows

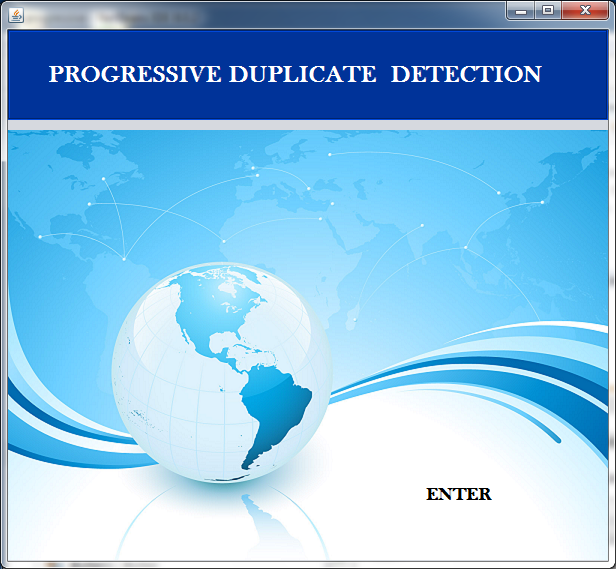


Figure 5.1 Home screen

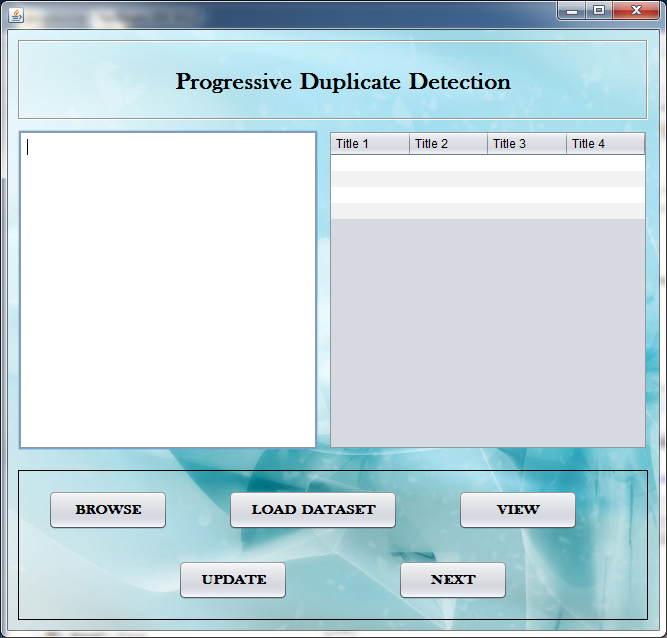


Figure 5.2 Menu Screen

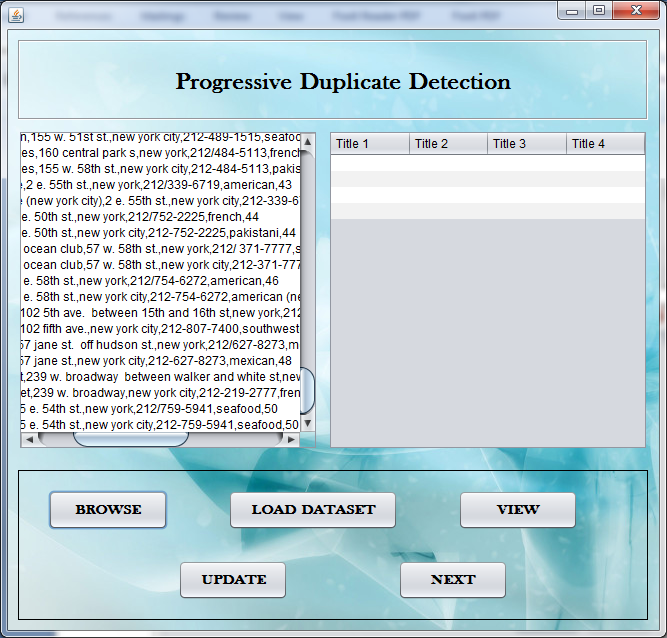
******

Figure 5.3Data Loader

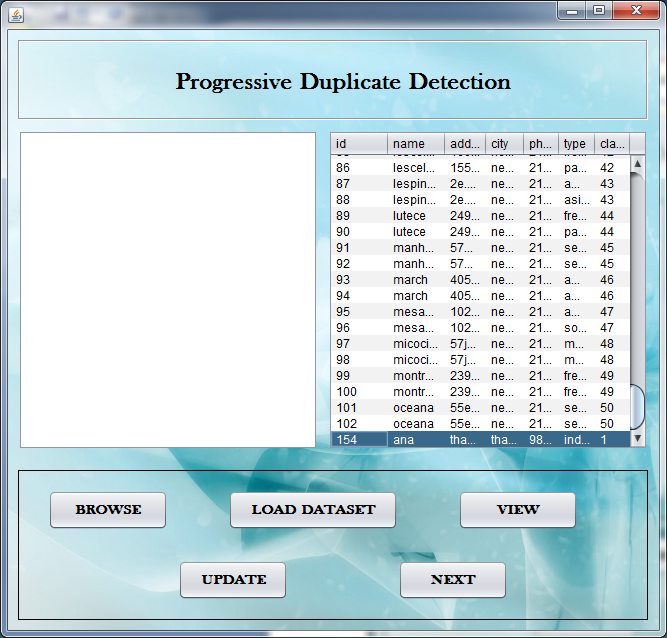


Figure 5.4 Ordered Data

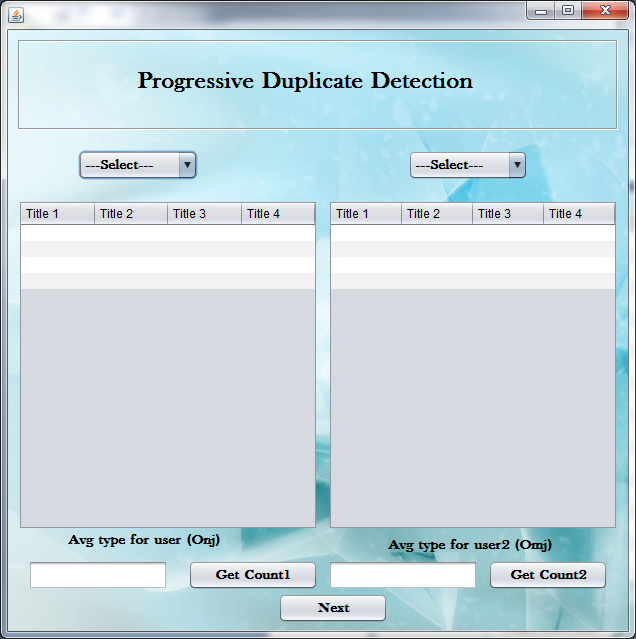


Figure 5.5 Selection Window



Figure 5.6 Ordered Data

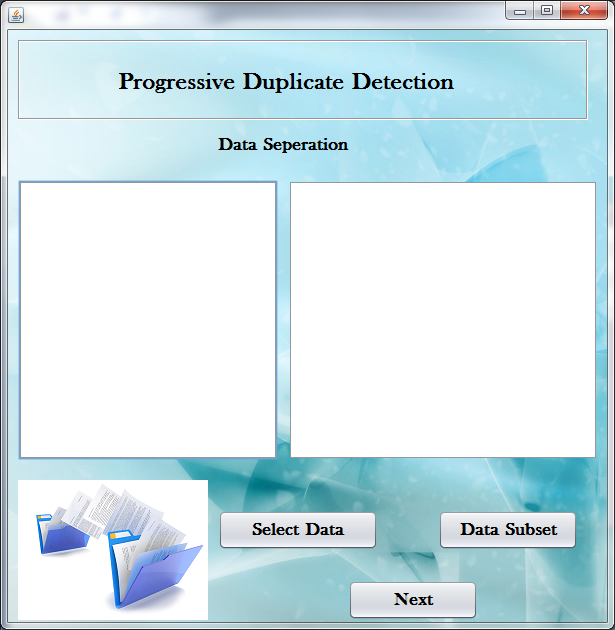


Figure 5.7 Data Seperation

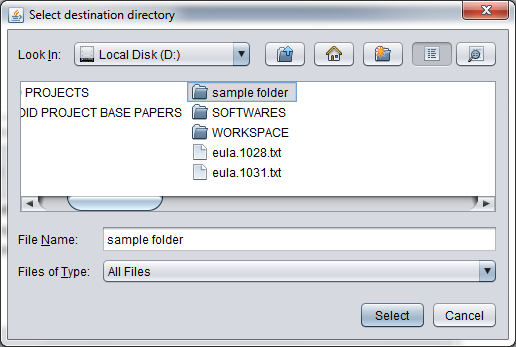


Figure 5.8 Selection Window

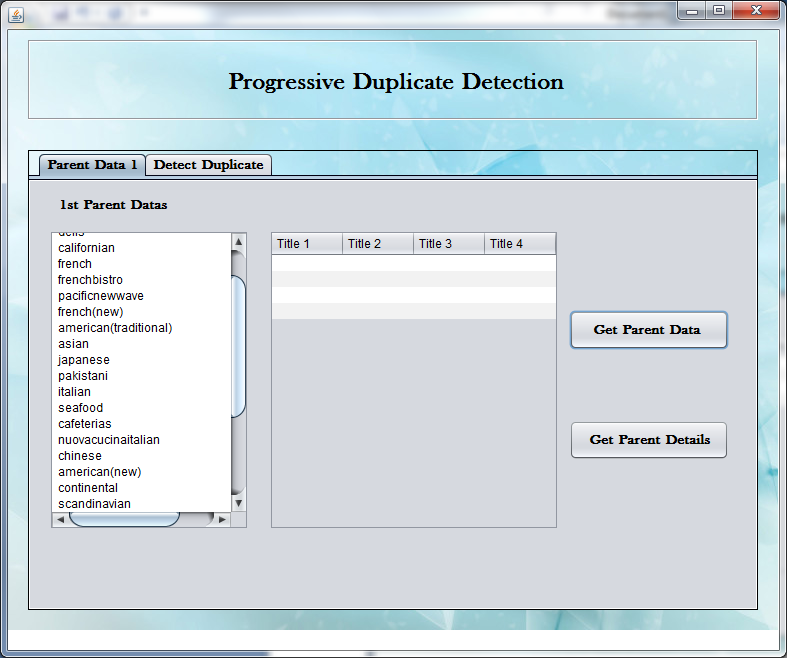


Figure 5.9 Dataset Detection

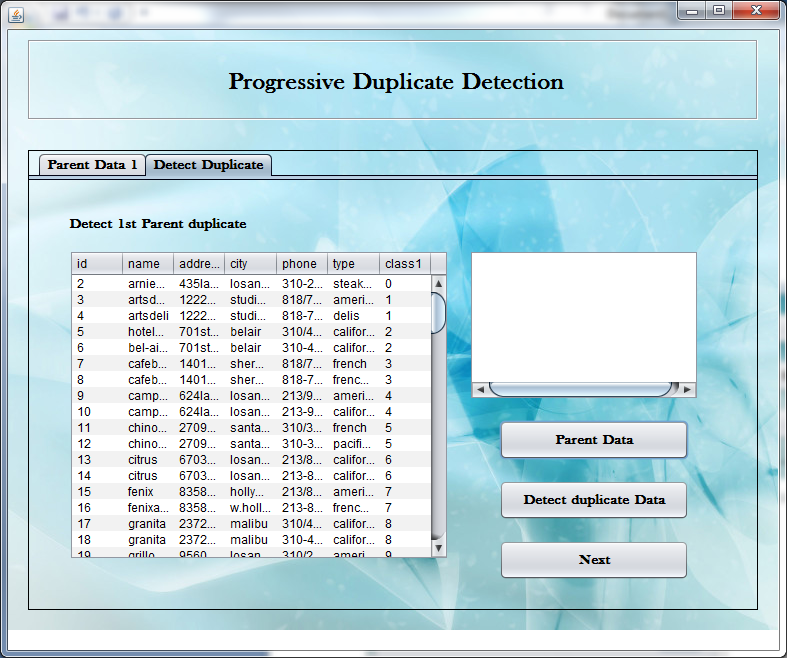


Figure 5.10 Duplication Detection 1

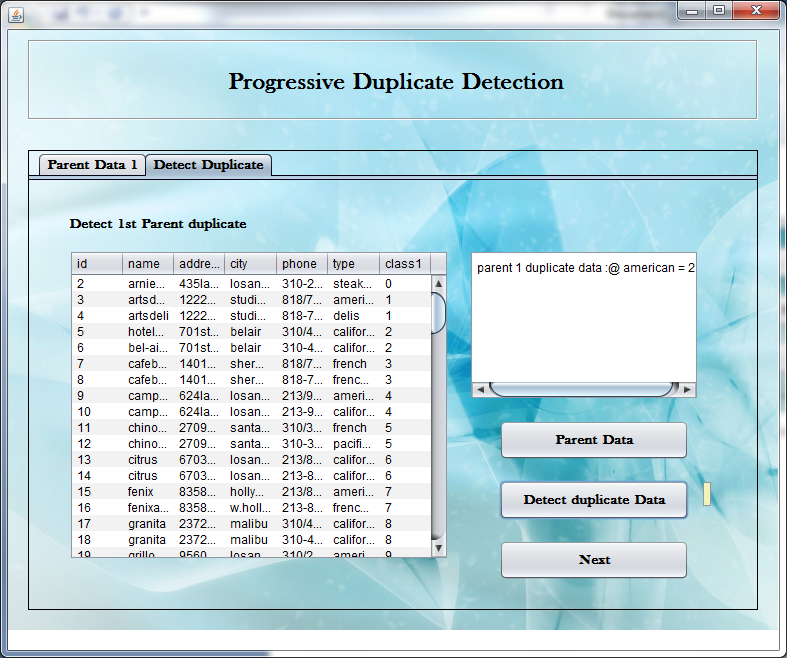


Figure 5.11 Duplication Detection 2

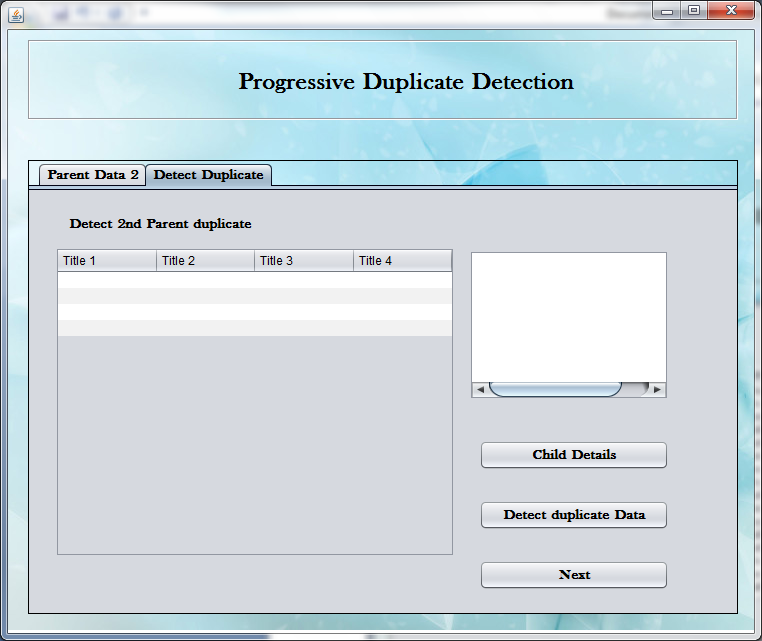


Figure 5.12 Clustering

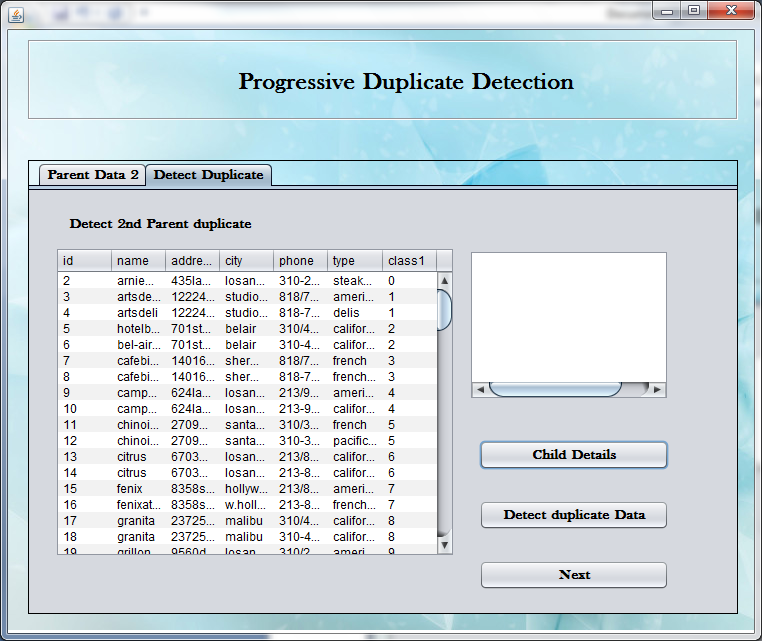


Figure 5.13 Full Data

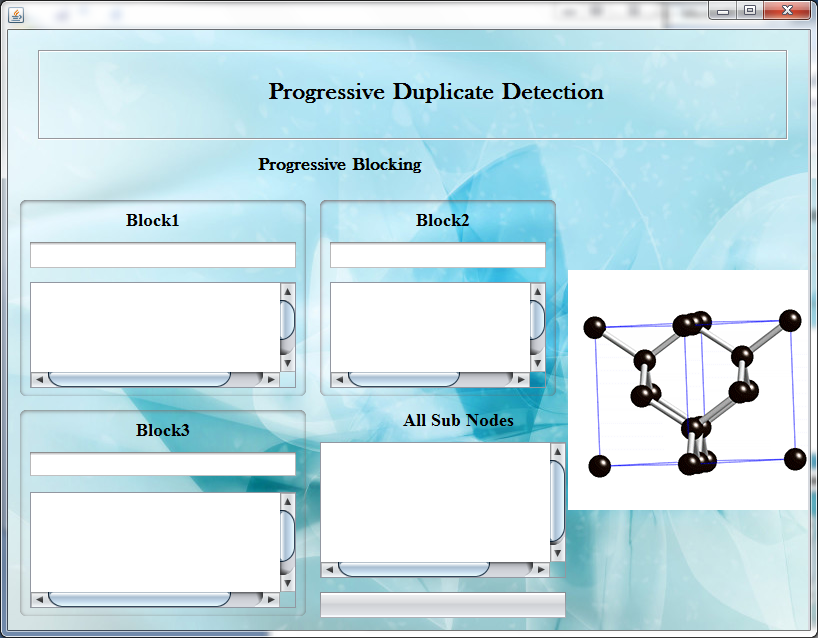


Figure 5.14 Blocking Applied

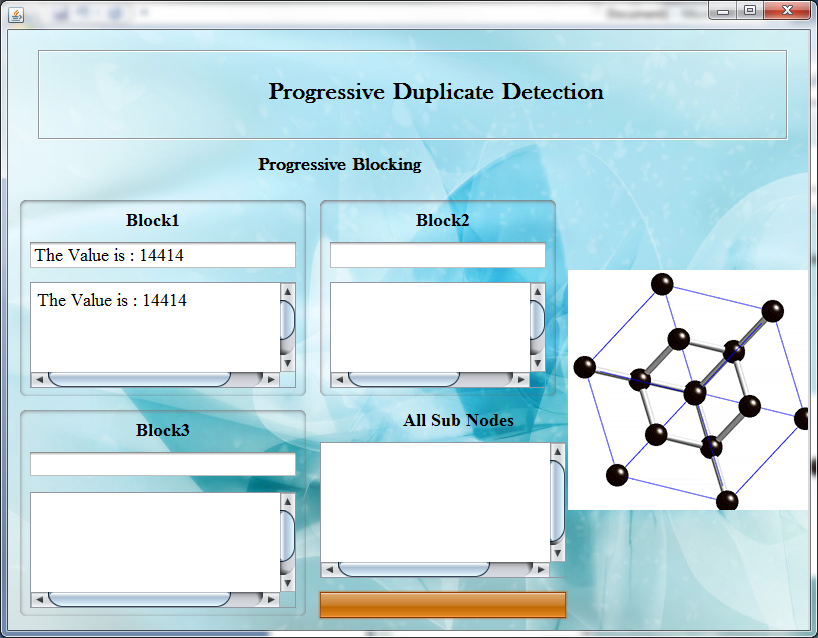


Figure 5.15 Blocks



Figure 5.16 Representation Window

**6. CONCLUSION AND FUTURE WORK**

**6.1 CONCLUSION**

We implemented the progressive sorted neighborhood method and progressive blocking. Both algorithms increase the efficiency of duplicate detection for situations with limited execution time; they dynamically change the ranking of comparison candidates based on intermediate results to execute promising comparisons first and less promising comparisons later. To determine the performance gain of our algorithms, we proposed a novel quality measure for progressiveness that integrates seamlessly with existing measures. Using this measure, experiments showed that our approaches outperform the traditional SNM by up to 100 percent and related work by up to 30 percent  For the construction of a fully progressive duplicate detection workflow, we proposed a progressive sorting method, Magpie, a progressive multi-pass execution model, Attribute Concurrency, and an incremental transitive closure algorithm. The adaptations AC-PSNM and AC-PB use multiple sort keys concurrently to interleave their progressive iterations. By analyzing intermediate results, both approaches dynamically rank the different sort keys at runtime, drastically easing the key selection problem

**6.2 FURUTE WORK**

In future work, we can combine our progressive approaches with scalable approaches for duplicate detection to deliver results even faster. In particular, Kolb et al. introduced a two phase parallel SNM , which executes a traditional SNM on balanced, overlapping partitions.

Here, we can instead use our PSNM to progressively find duplicates in parallel.

**REFERENCES**

[1] S. E. Whang, D. Marmaros, and H. Garcia-Molina, “Pay-as-you-go entity resolution,” IEEE Trans. Knowl. Data Eng., vol. 25, no. 5, pp. 1111–1124, May 2012.

[2] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, “Duplicate record detection: A survey,” IEEE Trans. Knowl. Data Eng., vol. 19, no. 1, pp. 1–16, Jan. 2007.

[3] F. Naumann and M. Herschel, An Introduction to Duplicate Detection. San Rafael, CA, USA: Morgan & Claypool, 2010.

[4] H. B. Newcombe and J. M. Kennedy, “Record linkage: Making maximum use of the discriminating power of identifying information,” Commun. ACM, vol. 5, no. 11, pp. 563–566, 1962.

[5] M. A. Hern\_andez and S. J. Stolfo, “Real-world data is dirty: Data cleansing and the merge/purge problem,” Data Mining Knowl. Discovery, vol. 2, no. 1, pp. 9–37, 1998.

[6] X. Dong, A. Halevy, and J. Madhavan, “Reference reconciliation in complex information spaces,” in Proc. Int. Conf. Manage. Data, 2005, pp. 85–96.

[7] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller, “Framework for evaluating clustering algorithms in duplicate detection,” Proc. Very Large Databases Endowment, vol. 2, pp. 1282– 1293, 2009.

[8] O. Hassanzadeh and R. J. Miller, “Creating probabilistic databases from duplicated data,” VLDB J., vol. 18, no. 5, pp. 1141–1166, 2009.