CONSTRUCTING FEATURES AND PSEUDO-INTERSECTIONS TO MAP UNRELIABLE DOMAIN SPECIFIC DATA ITEMS FOUND IN DISJOINT SETS

By

BILLY D'ANGELO GASTON

Bachelor of Science
Langston University
Langston, Oklahoma
1998

Master of Science
Oklahoma State University
Stillwater, Oklahoma
2001

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 2007

CONSTRUCTING FEATURES AND PSEUDO-INTERSECTIONS

TO MAP UNRELIABLE DOMAIN SPECIFIC DATA ITEMS

FOUND IN DISJOINT SETS

Dissertation Approved:

Dr. K. M. George

Dissertation Adviser

Dr. George Hedrick

Dr. Marilyn Kletke

Dr. Nohpill Park

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to thank Jesus Christ, the Lord and Savior of my life. He blessed me with the opportunity to learn from some of the best and brightest minds in the field of computer science as well as the opportunity to help others to learn and accomplish their goals in the process. He was and is my strength and present help.

I would like to thank Dr. K. M. George, my head dissertation chairperson. He has played a vital role in my educational pursuits since my undergraduate studies at Langston University. I thank him for accepting me and working with me as his student from the Masters through the Doctoral level. I will be forever grateful for his guidance and commitment to my success. From witnessing my wedding to the birth of several children, I thank him for his patience and willingness to adapt to my individual needs and circumstances.

I would like to thank Mr. Kenneth Simmons for trusting Dr. George in giving me the employment opportunity to learn in a fast paced and real-world research environment at Tinker Air Force Base. He believed in my potential and allowed me to play an integral part in research and work that affected the lives of many, both civilian and military. I thank Ken for his dedicated walk of faith. It is truly an encouragement.

I would like to thank Dr. Park, Dr. Hedrick and Dr. Kletke, my dissertation committee members, for their constant and needed encouragement and commitment throughout this process. Thank you for your openness and availability. At times even without knowing my financial struggles, God used Dr. Hedrick on several occasions to bless me with financial opportunities. I thank Dr. Hedrick for being a blessing.

I would like to thank the Computer Science Department for the opportunity to be a student of computer science at Oklahoma State University. I would like to thank Mrs. Beau Turner, Computer Science Department Supervising Secretary. She has been a great help throughout my studies at OSU.

I would like to thank Dr. Chandler and Dr. Mayfield for making themselves available when needed and for their words of encouragement.

I would like to thank Dr. Ernest L. Holloway, a friend and encourager. He was instrumental in bringing me to Langston University from Hanau, Germany. I thank him for his confidence and compassion for educating youth of all ages. He truly believes in second chances.

I would like to thank Anautics, Inc. for their encouragement and commitment to education.

I would like to thank my wife, Erma, for believing in me and for just loving and listening to me in my times of discouragement and confusion. She has played a vital role in the care of our family throughout this long and strenuous process.

I would like to thank my mother Marie and my sister Africa for their constant encouragement and support. In closing, I would also like to thank my friends and family for their constant prayer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Throughout history information has been a critical component in decision making processes for entities. Without accurate and timely information, today's average business would quickly close its doors as business revenue would decrease for lack of appropriate business strategy. The ability to manage, retrieve, and capture knowledge from documents and/or data repositories is important to the viability of any business. Discovering and defining relationships (e.g. trends, patterns, anomalies) across sets of information has become a rather common and needed task within all areas of business and industry. Entities needing this capability range from potentially small scale systems, such as a small elementary school library book management systems, to large scale systems, such as military government information systems servicing several international logistic depots. Presently several methods have been designed to capture relationships across a collection of documents or within a single relation (table). This is to assist the analyzing entity in discovering new correlations or anomalies that may prove beneficial in business decisions, medical discoveries, advances in information retrieval and organization, environmental management, aircraft maintenance management and the like. The methods designed involve techniques based on clustering analysis, prediction models, anomaly detection algorithms and decision rules. The demand for such techniques as well as the optimization of them have increased substantially as the volume that data entities are required to manage has increased over time. Although several techniques have been explored, the task of making sense of data (i.e. uncovering relationships), whether the data is historical or currently acquired, still remains an issue. Making sense of data, suggests both capturing and understanding what the data communicates either explicitly or implicitly. This research addresses this issue as well as another unique problem. Perhaps just as critical, yet increasingly more difficult to achieve, is the task of linking data sets that are considered disjoint.

Definition 1.0: A *data set* is defined as a collection of data items; such as, tuples, strings, numbers, etc., which represent some container of information.

As a practical application, consider a fictitious company, Part-Mart, that sells auto parts. This company contracts two different companies to supply their auto parts. In making these parts, Company A refers to a "screw" as a "bolt" while Company B refers to a "screw" as a "metal component". Note both companies refer to the same object however with a different name assigned to it. In addition let each company assign their own part numbers to the parts. Company A eventually goes out of business and all its inventory documentation is lost. Part-Mart Company, in an attempt to create a database to store the name associations of the parts from Company A and Company B in a table, is now presented with a problem. In an effort to merge the data, Part-Mart soon discovers that there do not exist unique identifiers (e.g. primary keys, foreign keys) in any database tables to attempt a data integration step such as a SQL JOIN. (The **SQL JOIN** clause is used whenever one has to select data from two or more tables in a relational database. To be able to use the **SQL JOIN** clause to extract data from two (or more) tables, one needs a relationship (common column value) between certain columns in these tables.)

Definition 1.1: A *Data Integration Step* is the process of merging multiple datasets (e.g. tables) into a single dataset.

Alternately, employees from either Part-Mart or Company B could perhaps sift through several part descriptions and associated schematics in attempts to associate part names and numbers. Unfortunately, performing this entire task manually is not feasible when there are several parts in question (i.e. order of millions). Such a task could take years. How does one solve such a problem? How does one automate such a process? Unfortunately many entities, particularly part manufacturers and logistic depots, are left in this dilemma, researching solutions to reconcile multiple sets of data that possess the characteristic of being disjoint.

In this research, we introduce data mining and record linkage strategies to resolve the issue of discovering and identifying relationships between data items in disjoint sets. Data mining can be defined as the process

of uncovering relevant information in structured data resident in large data repositories [36]. Although the data is found in large data repositories consisting of multiple tables generally the actual information processed by data mining algorithms are single tables of data. In figure 1 an example of two tables with common field values is seen. It is noted, the tables are not identical in size and information.

CustomerID	CarType
10325	Chevy
43565	Ford
34245	Nissan

CustomerID	First	Last	PhoneNo
10325	Jack	Frost	1234567
43565	Sally	Mae	7654321
34245	John	Doe	1234765

Figure 1. Multiple relations with common field

Conversely, multi-relational data mining (MRDM) uses multiple relations (tables) as datasets to discover or uncover patterns within the dataset. MRDM begins by performing a data integration step on the multiple relation dataset. The data integration step can be either a join or an aggregation operation. These commands are made possible because the relations have a field in common. Figure 1 illustrates two tables where the field *CustomerId* exists in both. After the data integration step, the knowledge discovery process begins. See figure 2 for an example of a traditional record linkage process.

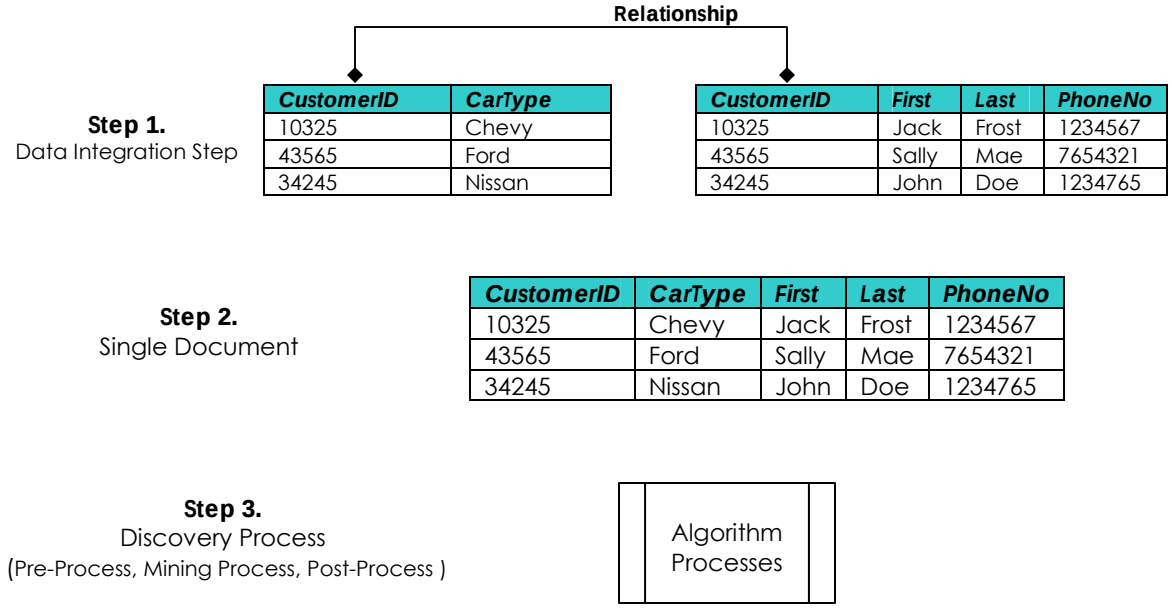


Figure 2. Traditional linkage process

As previously suggested, in attempting to apply data mining techniques to multiple relations, a significant problem presents itself if there does not exist a mutual field among the tables to apply a data integration step. Although there may be an implicit relationship between the relations, without the existence of a common field, the relations are still considered disjoint. An implicit relationship can be characterized by a prior knowledge that one or more fields in a set of relations are related however there exists no explicit identifier in which to connect the two. Figure 3 depicts a modern record linkage process.

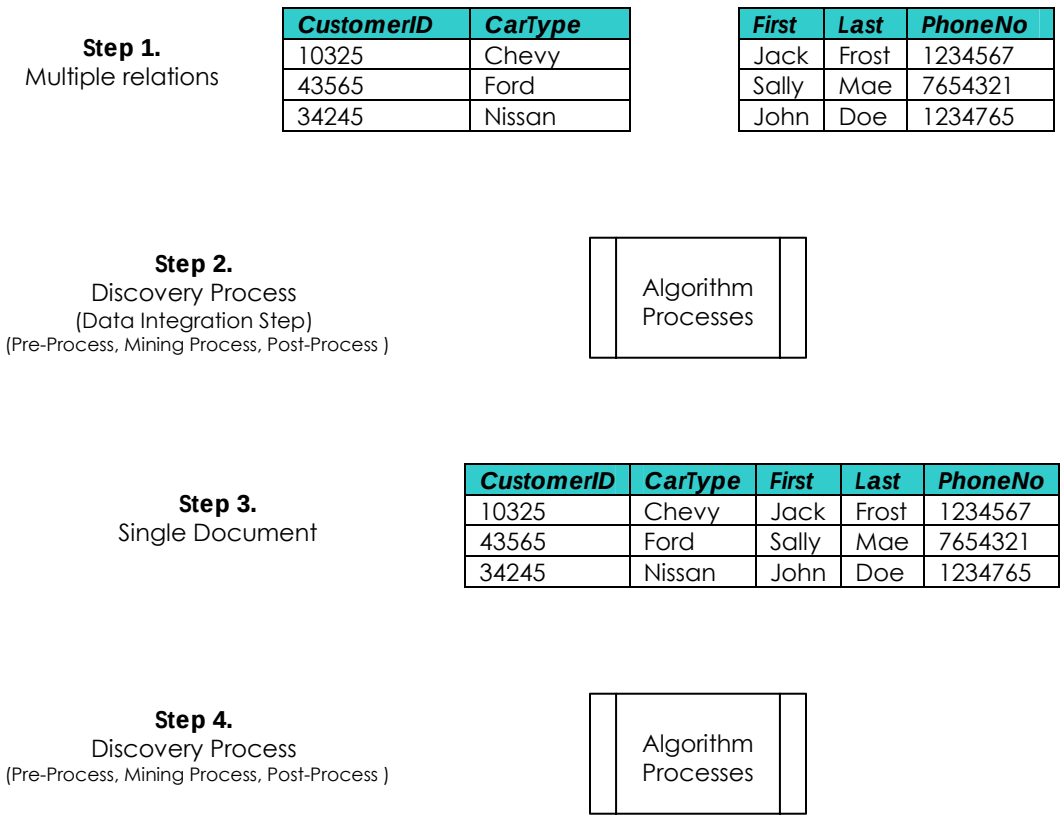


Figure 3. Modern linkage process

Definition 1.2: The process of integrating data sources (e.g. data records) that use different data formats and terminology to refer to the same entity (e.g. person, place, event) is referred to as *record linkage*.

This research presents a set of novel data mining and record linkage approaches to assist in establishing explicit links between disjoint data resident in multiple relations.

CHAPTER TWO

FORMAL PROBLEM

2.1 Problem Description

Although multi-relational data mining suggests the initial step of data integration in order to begin algorithmic application to discover new information, we present a unique data integration issue that remains prevalent in the information industry. In the event that a common field is not present between two relations, the researcher must identify a set of fields in each relation which will give the most optimal opportunity to initiate a data integration step. As the relations are disjoint, this implies that a data integration step based on a solution that solely compares relation field values for equivalency is not feasible. As in traditional record linkage and data mining, several obstacles may present themselves. This research deals primarily in the domain of machinery parts data, but it is noted that the problem can be generalized. The issues being addressed in this research is formally defined as the *MUDD classification problem*. We describe the problem as being distinguished by four vital elements,

Element 1: **M**apping processes,
Element 2: **U**nreliability of data,
Element 3: **D**omain specific data and
Element 4: **D**isjoint data sets.

M.U.D.D. is the acronym that best captures the four elements which exists together to characterize the core problem. For clarity, in future sections, references to the disjoint sets and all involved data items will be addressed as the *Data Field*.

Definition 2.0: Given the sets A, B, and C, where $A \cap B = \varnothing$ and C contains data elements that reference elements in A and B, a *Data Field* is defined as the set $D = (A \cup B \cup C)$.

In the sections to follow a detailed explanation of the four elements will be presented. A discussion of

element four, Disjoint data sets, will be discussed in section 2.2. In section 2.3 and section 2.4, element three, Domain specific data, and element two, Unreliability of data, will be discussed respectively. In section 2.5 we elaborate on element one, the Mapping processes. Finally in section 2.6 we present a formal problem statement.

2.2 Disjoint Property

We begin with the description of element four, the very foundation of the problem. Element four captures the central issue of the MUDD classification problem. It is denoted by two existing disjoint data sets A and B, each of which contain data components such as records in a data source (i.e. document or database).

Let an arbitrary data set, represented as a table T, be defined as,

T: $\{t \mid t = \langle a_1, a_2, \ldots a_k \rangle$ where a_i are fields within t$\}$

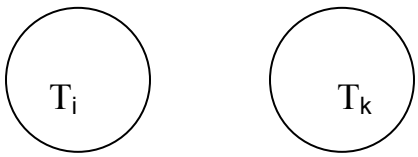


Figure 4. Disjoint Sets

In figure 4, the two sets A (e.g. T_i) and B (e.g. T_k) are considered to be disjoint as they have no records t_i or elements $\langle a_k \rangle$ in common. Let $T_1 = \{t \mid t = \langle a_1, a_2, \ldots a_n \rangle\}$ and $T_2 = \{t` \mid t` = \langle b_1, b_2, \ldots b_m \rangle\}$. Then T_1 and T_2 are considered disjoint if for every $t \in T_1$ and $t` \in T_2$, $t \neq t`$ and $a_i \neq b_j$ for any i, j in any given t and t`. Table I and Table II show an example of the two disjoints sets A and B expressed as tables. Since the two sets are disjoint, the process of making the correct mapping between records $t_i$ in set A to records $t`_j$ in set B can become extremely complex. Referring to Table I and Table II, in the event that an analyst attempts to map a record from set A to a record in set B, the analyst soon establishes that the records do not have an existing common field $\langle a_i \rangle$ that can be used to create a direct record mapping or merge.

TABLE I. Sample Set A as a Table

Part Description	Quantity	Age	Part No.
steel rod	4	12	ab-723
forward covering	2	3	yu-343-33
left fuel assembly	2	5	34t-443-2

TABLE II. Sample Set B as a Table

Part Description	Quantity	Age	Part No.
fuel support	1	13	xyx-ab-123
front dump panel	7	9	89-43-wer
retractable arm	14	1	390-op3-78

Attempting to do such a task manually could prove exhaustive if the record count is large. A central issue lies in designing and implementing a process by which associations between records in both data sets can be efficiently and accurately acquired in an effort to determine a new set of probable record mappings which will ultimately yield a third set X. The third set X can be likened to that of a set established after a (SQL) JOIN operation is applied between two tables with a common field. However unlike a SQL JOIN where the operation is executed using at least one common field value found in both tables, this issue calls for a similar operation or operations that must be executed using at least one form of a relationship/association discovered and defined by the analyst as a result of careful analysis of the data.

2.3 Domain Specific Property

The third element is denoted as the domain specific property. This property suggests that the data being analyzed and processed is restricted to a certain corpus or related subject matter. This deals with particular words, jargon and documentation methodologies used in a domain specific context. Word preferences or a company's adopted word usage standard can also be attributed to the existence of this property. An example of a word in which its meaning is determined by the context or domain in which it is used is "nose". The term nose could identify a part of an airplane or a part of the face of an animal. This research deals primarily with data as it is related to machinery parts and systems.

Definition 2.1: We define the term *Domain Object* as being the primary entity that is described or referenced by elements in a Data Field. (See definition 2.0 for a formal definition of Data Field.) More specifically, the Data field is the collection of data items and the Domain Object is the entities the data items describe or reference. As an example if the Data Field consisted of part descriptions that referenced a

Ford Mustang, the Data Field would be all part information (i.e. strings, numbers) while the Domain Object would be denoted as a Ford Mustang.

2.4 Unreliability Property

The second element of the MUDD Classification problem is the property of unreliable data. This property suggests that the data in its natural state renders a high level of unreliability. Information in such a state may be unstructured and difficult to understand when first viewed by the human eye; thus meaningless to a computer algorithm. This second element also references a whole host of separate impediments that may otherwise be quickly eliminated by exhaustive human interaction. Such obstacles include:

1) misspelled words,
2) missing/incomplete terms
3) variations of word spellings,
4) added, irrelevant, or ambiguous words,
5) abbreviations of words,
6) acronyms,
7) synonyms,
8) multiple uses of the same word and
9) references to outside documents or figures.

It is already known that the process of attempting to discover synonyms alone poses an extremely difficult semantic challenge within itself [18]. With the aforementioned obstacles, attempting to match words within data sets becomes increasingly difficult. Figure 5 shows sample word variations and abbreviations of the words - building, message, housing and assembly.

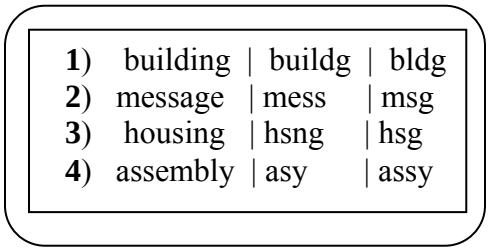


Figure 5. Sample word variations and abbreviations

We note that one reason various word variations exist in documents is because different authors of documents tend to use a variation of words and word phrases to articulate their point. Finally the categorization or structure of the data could prove unreliable as well. For instance, if two disjoint sets are thought to consist of items (e.g. records) related to the category of "Automobile Fuel System", the process

of relating the records from both sets would become even more difficult if 90% of the items in one of the sets were actually items which belonged to the "Automobile Electric System".

The English language and its word usages are extremely difficult to grasp in itself. Some of these issues and ambiguities as it relates to word usages are rather astutely characterized in [49] where comedian George Carlen is quoted as saying,

> "The English Language. Have you ever wondered why foreigners have trouble with the English Language?. Let's face it. English is a crazy language. There is no egg in the eggplant. No ham in the hamburger. And neither pine nor apple in the pineapple. English muffins were not invented in England. French fries were not invented in France. We sometimes take English for granted. But if we examine its paradoxes we find that Quicksand takes you down slowly, Boxing rings are square. And a guinea pig is neither from Guinea nor is it a pig. If writers write, how come fingers don't fing. If the plural of tooth is teeth. Shouldn't the plural of phone booth be phone beeth, If the teacher taught, Why didn't the preacher praught. If a vegetarian eats vegetables. What does a humanitarian eat? Why do people recite at a play, Yet play at a recital? Park on driveways and Drive on parkways. You have to marvel at the unique lunacy. Of a language where a house can burn up as it burns down. And in which you fill in a form by filling it out. And a bell is only heard once it goes! English was invented by people, not computers, and it reflects the creativity of the human race (Which of course isn't a race at all) That is why when the stars are out they are visible, but when the lights are out they are invisible, and why it is that when I wind up my watch it starts, but when I wind up this observation, it ends."

2.5 Mapping Property

Mapping is introduced as a property because to attempt to combat the MUDD Classification problem with a pure matching solution would soon prove useless as the Data Field is disjoint. For this reason instead of exact or approximate matching, a mapping process must be considered. The mapping property is categorized into four major areas of concern 1) mapping ambiguity, 2) many-to-one mapping, 3) hierarchical relationship mapping and 4) mapping validation.

2.5.1 Mapping Ambiguity

Mapping ambiguity occurs when two items appear to be related but there exists a level of uncertainty that prevents the mapping from being created. More specifically, in the event that one does discover a probable mapping between items in both data sets (e.g. items may have similar descriptions), do the descriptions

actually refer to the same item? This specific issue confronts companies like collection agencies. As an example, collection agencies must determine if "Eugene W. Foster and Gene Foster are in fact the same person? On the surface this seems like a simple question. But determining whether or not they are one and the same can require some comprehensive analysis. There are numerous Fosters in the U.S., so how do we determine if they are the same person? We must examine any and all common attributes available in the data store. This might include an account number, address, telephone number, birthday, or any other information available that might link them together. It is also important to note that any "similar" information found may not be stored in the same format. For example, a birthday on one file might be stored as mmddyy but as ddmmyyyy on another. The ability to dissect this information and boil it down to a common format is equally important as identifying it." [35] How does one solve such a problem if no common record attributes exist?

2.5.2 Many-to- One Mapping

The many-to-one property adds another dimension of difficulty. With the presence of this issue, there exists no "process of elimination" property associated with the items in the problem domain. More specifically, when and if an element τ_j of set A is found to map to an element τ_x in a set B, element τ_x can still be found to map to other items τ_i and τ_k in set A. Such a relationship is termed many-to-one and serves to further intensify the problem. See figure 6b. Ideally one would hope to eliminate a previously used item as to have a one-to-one mapping. See figure 6a.

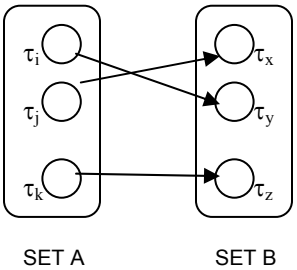


Figure 6a.

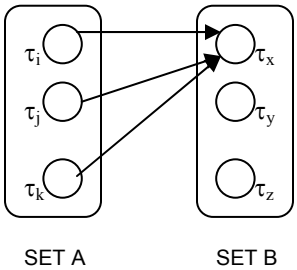


Figure 6b.

Definition 2.2: A *many-to-one mapping* is defined as, given two sets A and B, one or more elements in set A can map to at most one element in set B.

2.5.3 Hierarchical Relationship Mapping

Additionally, although it may be thought that records in a set A, which consist of words or phrases, may have an associated record located in the set B; the record description in set B may not be an actual synonym or direct reference to an identical element in set A. More specifically the correct record mapping may not be obtained simply by analyzing names or words for pure synonymous meanings. Rather the correctly associated record in set B could be a reference to a larger component or system where the component described in set A resides. Two records can reference the same object or both can potentially point to an item in an upper level of a relationship hierarchy. For instance a record denoted as "power button" in set B could actually be associated to a record denoted as "computer monitor". A computer monitor is clearly not a power button; however a power button is a component of a computer monitor. The hierarchy exists as power button is an item which can be found on a computer monitor. Obviously such mappings would depend on the domain and characteristics of the data.

2.5.4 Validation of Mapping

Of course another issue is determining which words in the sets exhibit the most relevance. Capturing the relevance of specific data items within the data sets places a data analyst or mapping algorithm in an optimal position to identify a correct mapping. Additionally there must be some approach identified to quantify the relevancy of any particular data component as to have some mathematical basis to validate the results.

2.6 Formal Problem Statement

We formally state the MUDD Classification problem:

Given two disjoint data sets A and B, consisting of domain specific and unreliable data, identify an automated process to link an element from set A to an element in set B.

CHAPTER THREE

RECORD LINKAGE AND DATA MINING

3.1 Overview of Record Linkage

Record linkage alternately denoted as data linkage is the integration of information from multiple independent data sources. Data linkage can be used to improve data quality and integrity, to allow analysis of data for new analytical studies, and to reduce costs and efforts in data acquisition [25,43,5]. Linkage techniques are used to link together records which relate to the same entity. An entity is denoted as a student, patient, customer or any type of information representing something deemed important by a data analyst. Record linkage techniques are extremely beneficial when links among one or more data sets are needed. Typically records are linked using a common identifier such as social security numbers, birth date, marital status or sex. The process of record linkage is simple when a common field exists among the data sets; however a unique identifier for each entity represented within the data sets is not always available. This makes linking data difficult if not impossible. Records are sometimes compared to determine the likelihood or probability of being a direct link. Many linkage techniques are founded in machine learning and data mining. These two tools are often used to improve upon the accuracy of the linkage as to significantly reduce the man hours required to link large data sets [25,5].

It is common for different organizations to employ different terminology such as codes or identifiers when referring to the same entity. Several factors make creating and maintaining links among data entities difficult. These factors include, lack of a known standardized means of identifying the types of terminology, progression of time almost always ensures changes in terminology, changes of which are not usually reflected in historical data. Record linkage is a reoccurring issue within many organizations. Large applications such as database management systems and expert systems are continuously utilized to

integrate multiple information sources to assist in information restructuring efforts, knowledge discovery, decision making processes and statistical analysis.

Linkage techniques are used in areas such as health administration, patient care, insurance claims and vital statistics data. Many of the previously developed record linkage systems depend heavily on rules associated with specific fields within the data field being considered. Furthermore, most of the rules are manually programmed for each type of field. Unfortunately such systems are difficult to create and maintain because with the addition or modification of fields new rules must be identified and applied to the new information.

There are two basic types of record linkage, deterministic and probabilistic.

3.1.1 Deterministic Linkage

Deterministic record linkage is a technique which attempts to link two or more files based on exact agreement of matching variables [43]. Examples of variable could be patient last name, marital status, and geographic location. Typically, a link created by a deterministic linkage algorithm is made when an entire set or subset of identifiers agree between two records. This method is said to decrease the number of uncertainties in the matches between two databases since only a complete match on a set of unique variables is accepted at the cost of lowering the linkage rate [25].

A common issue one finds with deterministic record linkage is that when two records agree on a specific field, there usually exists no additional information on whether that agreement increases or decreases the likelihood that the two records in fact refer to the same entity. An example can be seen when two records agree on the last name, ColdFoster, and two records agree where Smith matches Smith. Generally, these two sets of mappings would be treated with similar matching power, even though the likelihood of Smith matching is greater than that of ColdFoster.

3.1.2 Probabilistic Linkage

Probabilistic Record linkage is a technique which attempts to link two or more files by utilizing computed probabilities of agreement and disagreement between a range of matching variables [43] Probabilistic linkage was developed by researchers in response to an issue encountered in deterministic linkage. Researchers often encounter data sets where there exist no single set of identifiers which can be used to distinguish between truly linked records. Probabilistic record linking assumes multiple pieces of identifying data can be used to calculate the probability that two records refer to the same entity. Unlike its deterministic counterpart, this technique suggests that no single variable agreement between files can render a total match reliability. In general, a formula is derived which generates a score for each record pair and based on the score identifies record pairs as matches, potential matches, and non matches. The formula incorporates weights specific to each of the data elements and scaling factors for many of the data elements. The weights reflect the relative importance of specific data elements in predicting a match. The scaling factors adjust the weights for a given record pair based on the "rarity" of the data value [5]. As an example, the last name "ColdFoster" would have a much larger scale factor than that of the last name "Smith" [30]. Probabilistic linkage maximizes linkage theoretically but may result in uncertainty for some potential links [25].

Whether deterministic or probabilistic linkage techniques are used, errors denoted as false positives and false negatives can be encountered. A false positive error occurs when a link is identified between two records when the records do not actually refer to the same entity. A false negative error occurs when a link is not identified between two records when the records in fact refer to the same entity.

3.2 Overview of Data Mining

In solving the MUDD Classification problem, a form of knowledge capture method is explored. Knowledge capture can be described as the process of acquiring relevant nuggets of information by consulting computer programs, reference documents, databases, or human experts in an effort to obtain

knowledge that may prove vital to the function of current and future tasks. Knowledge capture can be costly in respect to both time and money as a result of error proneness and inconsistencies [32]. The primary method of knowledge capture explored in this research is data mining

Definition 3.0 : *Data mining* initially denoted as knowledge discovery in databases (KDD) is an automated knowledge capture process of analyzing, most notably, structured data in an effort to uncover hidden knowledge that is not otherwise captured from an in-depth reading of the data [36,38,41].

Data mining techniques have become increasingly popular methods of use by data analysts today. Data analysts attempt to discover new knowledge from past and present data sets in an effort to make preparations and choices that will ultimately affect future business decisions. It poses the simple question of "what knowledge can we gain by looking at the data differently?" The study of data mining involves techniques and algorithms that have been developed to identify patterns, trends, and relationships within single and/or multiple populations of data. Such discoveries are vital in a decision making processes. One begins with the raw data, processes it into relevant information, and stores it as to accumulate a store of knowledge. Utilization and application of the acquired knowledge allows for effective decision making. Data mining has become an increasingly popular arena of research. Business entities as well as research institutions have become trailblazers in this discipline. Since the number of potential customers are finite, competition among corporations is extremely fierce and companies continually look for the edge that will give them the upper hand [35]. The design and implementation of business advantages, marketing strategies, system security, customer analysis and security options and data security are just a few of the many areas data mining techniques are being utilized.

There are many necessary steps that enable the continuous flow of the data mining process. Perhaps the most important and time consuming aspect of data mining is data preprocessing or preparation. Data preprocessing involves data cleansing and the eliminating of inconsistencies among the data [38]. Capturing quality data is the first and perhaps, most important step a company must take in achieving an

accurate view of its customer population [35]. It is said that 75 to 85 percent of the work in building models using data mining relates to the cleaning and preparation of data prior to a specific analysis [24].

Once the data preparation phase has been completed, one must determine the desired outcome of the future analysis. This step is vital in determining which available data mining tool will be used. In the event that there exist no feasible data mining tool to give the desired results, a new technique must be identified. Furthermore as stated in [38], obtaining results is not the difficult part, rather validating and making sense of the data seems to be the more difficult part.

3.2.1 Challenges in Data Mining

Although widely used, data mining does not go without its challenges. Aforementioned in the section above, the challenge of validating and making sense of the resulting data presents some difficulties to analysts. Additionally, some of the most familiar and research worthy issues surrounding the area of data mining are scalability and performance, high dimensionality, data ownership, data security and heterogeneous data [36]. Scalability and performance suggests being able to capture and process data in response to the increase of data as well as having the computing power necessary to process the data. High dimensionality is an issue which deals with the large number of attributes associated with data. This issue begins with the question of "How does one capture and make sense of the data?" Data ownership and security involves issues which deal with distributed data that may not be readily accessible due to security constraints as well as a lack of data ownership. Furthermore in the event access is granted to obtain data, secure communication must be maintained, the number of data transactions must be reduced to maintain efficiency, and any distributed data must be eventually consolidated into a comprehendible form. Heterogeneous and complex data suggests that the attributes associated with the data may not be of the same type which makes it difficult to make decisions of similarities or associations between the data [36].

3.2.2 Data Mining Classifications

There are at least four classifications of data mining - association rule mining, classification and prediction, cluster analysis, sequential pattern and time series [36]. It is assumed that all data mining techniques fall into at least one of the four categories. Association rule mining serves to discover associations [correlations] between data that occur frequently within the data population being analyzed. Classification and prediction is a process of classifying or modeling a group of data based on some defined characteristic shared by some of the members of the data population. Once the model has been developed, it is then utilized to predict the classification of future data components where the classification remains unknown. Clustering analysis is a process which divides a population of data into subsets based on attributes of similarity. The items found in clusters are more like the items in the same set and more unlike the items in others sets. Sequential pattern and time-series consists of a collection of techniques which identify patterns or trends based on the occurrence of events represented by the data from a moment-to-moment or time-to-time bases.

3.3 Record Linkage and Data Mining Techniques

There are many techniques used in the disciplines of record linkage and data mining. We deal with the concept of matching terms as well as linking records. It is noted that part of the process of linking records is the linking of terms within the records. A few techniques that will be addressed deals with decision trees, statistical inference, neural networks, clustering and synonym discovery [36].

3.3.1 Blocking

Blocking is the process of dividing the records in the Data Field into individual blocks of records as to reduce the number of actual record pair comparisons. There are several blocking methods used. Standard Blocking is done by selecting a feature or combination of features to create what is called a block key. Blocks are then created and consist of records which have common block keys. Hash Blocking is an

extension of standard blocking but the block key is hashed and the hashed record is placed in the appropriate block. Canopy Clustering with TFIDF is a form of blocking where clusters are formed by randomly selecting a candidate record from the global set of candidate records and placing all other records within a certain threshold distance into the cluster or block. In sorted neighborhood blocking, records are sorted based on some pre-defined criteria, such as first name values, and then a window of size x, where x is an integer that moves along the records [2]. Only records within the size of the sliding window are considered for comparison.

There are defined metrics to measure the performance of blocking schemes. First, the Reduction Ratio (RR) is represent as RR=1 – s/N where s is the number of record pairs produced by the blocking method, and N is the total possible number of record pairs (N = n x n), where n is the total number of records. This metric measures the relative reduction in the number of record comparisons [2]. Another blocking performance metric is denoted as Pairs completeness (PC). The formula is PC = s_M/N_M, where s_M is the total number of matched record pairs in the set of records produced for comparison by the blocking method and N_M is the number of true match record pairs in the entire data [2].

3.3.2 Comparison Measures

Comparison measures or string comparators are techniques used to compare strings to determine a measure of similarity. They work very well when the intent is to identify two words that are the same despite minor typographical errors. They assist in identifying potential abbreviations and acronyms. They are normally based primarily on the measurement of character variations and positioning among multiple words. These measures are not applicable when attempting to identify words that may be synonyms or aliases for one another. Some comparison measures are discussed below.

A. Jaro's Algorithm

Jaro's algorithm attempts to determine the number of characters or transpositions between two strings. It uses formula 3.1 to compute a comparison score:

$$(c/l_1 + c/l_2 + (2c-t)/2c)/3 \quad \textbf{(3.1)}$$

where c is the number of common characters, l_1 and l_2 are the lengths of the two strings, and t is the number of transposed characters.

B. Edit Distance and Hamming Distance

There have been several approximate string matching algorithms contributed by researchers in the area of computer science [8]. Given two words of the same length, the number of positions with different characters is denoted as the Hamming distance. This distance often used to compute a quality of match. In addition the Levenshtein distance or editing distance measures the number changes required to change one word into the matching string. The term changes refer to insertions, deletions, substitutions and transpositions. Based on the number of changes required a quality of match value is computed. The Hamming Distance string comparator is best used when attempting to determine the quality of match between strings that have a fixed length, such as social security numbers. In contrast, the Edit Distance is best used with variable length strings such as first and last names.

C. N-grams

In [3, 31], the authors use bi-grams and trigrams in the comparison of field values in an attempt to measure the closeness of values. Bi-grams are two-letter consecutive combinations and trigrams are three letter consecutive combinations. As an example the word "decrease" has the following trigrams "dec", "ecr", "cre", "rea", "eas", and "ase". The number of bi-grams or trigrams common or uncommon among words is used to compute a measurement of quality of match. One representation of the N-gram formula is

$$1 - (2 * c)/l_1 + l_2 \quad \textbf{(3.2)}$$

where c is the number of letter combinations common to both strings, and l_1 and l_2 are the lengths of the two strings. It has been shown that the N-Qram string comparator is most effective when the strings have minor typographical errors with N=2.

D. Soundex

Phonetic matching attempts to associate words not necessarily by their spelling but by the way they sound [45]. It is conceivable that words that sound alike in many cases have a similar variation of spelling. The SOUNDEX phonetic index is probably one of the most well-known encoding schemes used. It was first used in an 1880 census [48]. Researchers, wanting to locate surnames quickly, encode names by assigning certain letters specific codes. The letters A, E, I, O, U, W, Y, and H are disregarded in the encoding

scheme. The final encoding of a word is called a SOUNDEX code. Each code begins with a letter followed by three digits. For example the code B-536 denotes the surname Bender. The letter represents the first letter of the surname [48]. In using SOUNDEX codes one is able to group words like Smith and Smythe in the same category in the event of needed quick information extraction. Such an encoding could be used to assist in determining various word usages and abbreviations. This method is most effective when dealing with terms that are similar in sound.

3.3.3 Other Techniques

Decision trees are rule based structures used to classify data using data attributes which are filtered by decision rules. The internal nodes of the structure denote the tests which are conducted on the input pattern and the leaf nodes denote the determined classification of the input pattern. Example inputs could be a data record or collection of information. Each test is determined to be mutually exclusive as to generate a definite outcome for a given input.

Statistical inference deals specifically with statistical computations based on quantified characteristics of the data such a word frequency. Latent Semantic Indexing (LSI) is an information retrieval method based on statistical mathematics. It was designed to overcome such obstacles as polysemy and synonyms [19]. LSI is capable of returning relevant information in cases where keywords have not been identified. Synonyms suggest the concept of using many names to represent a single object. For instance the term "car" can also be referred to as "vehicle" or "auto". Polysemy suggests the concept of a single word having more than one meaning. For example the word "Java" can refer to a computer programming language or "coffee". Whereas many common indexing algorithms attempt to locate a set of documents based on the occurrence of key words, LSI retrieves documents based on concepts (e.g. subject matter) contained in the documents [37]. By using concepts, LSI is able to retrieve documents based more on semantics versus pure occurrences of terms. LSI utilizes a numerical analysis technique known as Singular Value Decomposition (SVD) to create concepts. SVD decomposes a single matrix into three distinct matrices. The first matrix is a term by concept matrix. The second matrix is a concept by concept matrix while the

third matrix is a concept by document matrix [37]. From these matrices one can index all the documents given in a particular concept. Some disadvantages of Latent Symantec Indexing is that it does not scale well, time consuming in reprocessing matrices, and difficulty in interpreting underlying reduced term space [37]. In our case the overall concept is known already; however the meanings and inferences of various terms and phrases within the document of records are not always known.

Neural network algorithms are techniques which attempts to emulate decision making patterns of a human brain in an effort to learn vital information about the data being modeled. Methods in neural networks attempt to learn by using a learning method called back-propagation or feed-forward [47]. The user filters input data through the neural network obtaining a result and an associated error value. The input data is continuously filtered through the neural network, giving the network an opportunity to learn until either an output close to the known desired output is obtained or the algorithm ceases to learn additional information with each repeated run. This phase is normally called training. Once a network has been trained, the network is then ready to process data in which it has been trained for in the hopes of predicting accurate results. More specifically, a neural network is able to predict an output pattern when it recognizes a given input pattern [44]. Neural networks are composed of simple elements operating in parallel. The function of the network is determined largely by the connections between the input and output elements. A neural network can be trained to perform a particular function by adjusting the values of the connections (weights) between elements. The learning ability of neural networks is based on the adjustment of weights. A particular set of data is input into the network and the output is compared with the expected output. The difference between the two outputs is used to adjust the weights among the neurons. This process is usually repeated until the network output matches the target output. The quality of a neural network is determined by the quality of the data used to train it. Training data consists of cases where the target outcome is known before hand. As an example test data can be fed into a neural network such that the network will be able to distinguish matching records from those that do not. Unfortunately, when there is little to no data to train the network, the network's ability to recognize patterns is greatly decreased. In this work, knowledge of an exact mapping expected for a given record is unknown and with each new record being processed the data needed to create the mappings is seen as unique.

In [3], the authors compute a quality of match for each field in a record by multiplying a relative weight times a computed quality of match field value. This product is then added to the product of a second relative weight which is multiplied by a quality of match for phonetic matching. The final composite value is then used to determine the overall probability of a record match. The author suggests no systematic means of computing weight values only that the weights are relative to the string and phonetic matching of the field. Phonetic matching suggests matching words that sound alike [45, 3].

Clustering is the process of grouping data with similar characteristics or qualities. One goal of clustering is to find homogenous data subsets within a pool of data [30]. Measuring or quantifying the homogeneity is perhaps the vital ingredient necessary in clustering. How does one quantify a single word? What is the most optimal approach of doing so and does it change when the data set changes? Inverse Document Frequency (IDF) is a weighting procedure commonly used. It weights words based on their occurrences over a collection of documents. Unfortunately, the distribution of words across a cluster of documents can vary greatly, thus rendering the IDF less effective or credible when selecting keywords [14]. The authors in [30] present a variant of discriminative clustering which represents text documents as probability distributions. Discriminative clustering attempts to involve discriminative elements into the clustering process. The text documents are used to form clusters and auxiliary keywords are use to optimize the clustering process. In working with data in text documents the following definitions should be understood,

- Term document frequency – the number of documents in which a term appears.
- Term discrimination values – a measure of the effect of the addition of a term to a vector space on the similarities between documents
- Good discriminator –term that tends to increase the distance between documents
- Poor discriminator – term that tends to decrease the distance between documents
- Indifference discriminator – term that when added renders no change in the distance between documents.

The authors in [12] present what many suggest is the formal foundation for probabilistic record linkage [40]. They present a conditional probabilistic approach where they use the probability that a field agrees given that the record pair being evaluated is actually a match. This probability is denoted as m. Next they compute the probability that a field agrees given that the record pair being evaluated is not a match pair. This probability is denoted as u. The computation for the weight corresponding to a matching field is $\log(m_k/u_k)$ and $\log((1-m_k)/(1-u_k))$ for the weight corresponding to fields that do not match. The sum of the

weights for individual fields is computed and the larger the weight the greater the probability that the two records are the same. The values m_k and u_k are estimated. One popular method of estimating the unknown values is using algorithms like the Expectation-Maximization (EM) algorithm. The basic formula for the two unknown values is below.

$$m_k = \Pr\{c_k^{i,j} = 0 \mid r_{i,j} \in M\}\ , \qquad u_k = \Pr\{c_k^{i,j} = 0 \mid r_{i,j} \in U\}$$

The term $c_k^{i,j}$ denotes the k[th] feature of records i and j. The term $r_{i,j}$ denotes a comparison between records i and j. The terms M and U denote the set of matched records and non-matched records respectively. The EM Algorithm applies numerous expectation and maximization iterations until the desired precision of the estimated values is reached [4]. The values g_m and g_u are derived in the expectation step. These values correspond to whether two records represent a matched or unmatched record pair respectively.

$$g_m(c_1) = \frac{p\prod_{k=1}^{n} m_k^{c_k^l}(1-m_k)^{1-c_k^l}}{p\prod_{k=1}^{n} m_k^{c_k^l}(1-m_k)^{1-c_k^l} + (1-p)\prod_{k=1}^{n} u_k^{c_k^l}(1-u_k)^{1-c_k^l}}$$

The value g_u is computed in the same manner. The unknown parameters m_k, u_k and p are computed in the maximization step. The value p is defined as the proportion of the matched record pairs in the data set. The actual equations to estimate the unknown parameters are below.

$$m_k = \frac{\sum_{l=1}^{N} c_k^l * g_m(c_l)}{\sum_{l=1}^{N} g_m(c_l)} \qquad u_k = \frac{\sum_{l=1}^{N} c_k^l * g_u(c_l)}{\sum_{l=1}^{N} g_u(c_l)} \qquad p = \frac{\sum_{l=1}^{N} g_m(c_l)}{N}$$

In [25], the authors attempt to link data from three different databases using a deterministic linkage method. They analyzed three different combinations of four identifiers: (1) surname (i.e. surname at birth or marital surname as recorded in these three databases), sex and date of birth (i.e. month, day, and year of birth); (2)

first name, sex, and date of birth; (3) surname, first name, sex and date of birth. The common identifiers were formatted in the same way across the three databases. Structured formatting included capitalizing letters, and removing blanks and dashes. The authors employed a SOUNDEX coding method to identify linkage between names that fail to match due to variant spellings of the names in the two databases [45, 48]. Among the three approaches, approach one (surname, sex and date of birth) had the highest linkage rate (88.0%) compared to approach two (first name, sex and date of birth, 82.4%) and approach three (surname, first name, sex and date of birth, 79.5%) [25].

In [7], the authors begin with the premise that terms are similar if they tend to appear in the same documents within a corpora. This can be represented by a term document matrix where each term is a vector and each document within the corpora is a dimension with entry values denoting if the term was present in a specific document or not. The authors used the following similarity measure to compute the similarity between two words, $\cos(i, j) = \frac{i \cdot j}{\sqrt{i \cdot i \times j \cdot j}}$, where vector *i* represents the term vector for word *i* and vector *j* the term vector for word *j* and *i*·*j* is the inner product of *i* and *j* [7]. This measure is based on the assumption that the axes are orthogonal. The author also used the Cluster measure as a comparison to the cosine measure. This measure provides an asymmetrical similarity relationship between two terms. $cluster(i, j) = \frac{i \cdot j}{||i||_1}$, where $||i||_1$ is the sum of magnitudes of *i*'s coordinates [7]. Results indicated that the cluster measure performed better in proportion of relevant terms that weire selected (i.e. concept recall ratio), while the performances were similar when tested for the proportion of selected terms that were relevant (i.e. Concept precision ratio).

In [9], the author presents a method for automatic thesaurus construction in an effort to modify queries sent to an information retrieval system. They use a matrix similar to [7]. The matrix is structured such that the documents are vectors and the terms denote dimensions. Term Frequency Inverse Document Frequency (TF-IDF) was utilized as a weighting metric for terms. In [9] TF-IDF is described as the number of times a term appears in a document multiplied by the function of the inverse of the number of the documents the

term appears in. It was determined that terms, which appeared in a single document often and not in many documents, have important weights [9].

In [33] the authors use a graph constructed from a dictionary and assumes words are similar if they have common words in their definitions. In constructing the graph each word x represents a vertex and an edge (x, y) represents a word y in the x's definition.

The authors in [6] introduce a graph-based algorithm to search large volumes of unstructured data. A contextual network graph is created from a term-document matrix. The creation of a term-document matrix is similar to the first step in Latent Semantic Indexing. Each term represented in the matrix is said to be connected to all documents in which it occurs. The value zero is recorded in the appropriate matrix position if the word is not present and a value indicating the word frequency is recorded otherwise. The matrix is transformed into a bipartite graph connecting terms and documents with the word frequencies corresponding to the weights placed on the edges. The search algorithm begins with a query word. The node where the word resides is found in the contextual graph and an arbitrary energy amount is assigned to it. The energy amount is divided between the neighboring nodes which are connected to the initial query node. This process is repeated on each neighboring node until the energy amount reaches a pre-defined threshold. Once the energy amount reaches the threshold, all nodes with energy deposits are sorted according to the energy. The nodes with the most energy deposits are considered most relevant.

The authors in [18] review different methods used to capture synonymy within documents. Such methods include approximate matching algorithms, vocabulary mapping, editor-directed searches and exploiting source semantics (i.e. various naming conventions). All these approaches were considered less than optimal by the authors. Approaches which obtained author approval were the use of lexical algorithms, word level synonymy, and inferred phrase level synonymy. An example of inferred phrase level synonymy can be seen in the following example. Given two phrases "Relatives died" and "Relatives deceased", by removing the common word "Relatives" from each phrase a possible synonym match is implied between "died" and "deceased".

The authors in [13] present variations of approximate word matching as an approach to combat spelling variants, misspellings, etc. in the detection and categorization process of digital libraries. This process consists of the following steps, 1) extract strings, 2) cluster strings, 3) review of strings (e.g. by human expert) and 4) make modifications that correspond to the reviewers analysis of the data. The clustering algorithm used simply extracts some desired set of strings which meet an author-defined criterion and captures their frequency in the respective document. If the distance between two strings meet a certain threshold then the strings are added to the cluster, otherwise one string is discarded. The string in the cluster with the highest number of frequency is used as the description for the cluster. The algorithm then attempts to pair up words in a way which would allow the sum of the edit distances to be minimized.

CHAPTER FOUR

METHODOLGY

4.1 Significance of Research

In a typical classification problem there exists a set of defined classifications and a population of data elements that need to be classified into one of the defined classifications. In this work we deal with two sets of data denoted as A and B. We define the records in set B to be the unique classifications.

Our work was done in conjunction with a business organization which specialized in the maintenance and distribution of aircraft parts. See figure 7 for a general diagram of an aircraft [46]. Due to the significant decrease of aircraft parts across the U.S. military over past years, managing parts more efficiently has become a great necessity. The term 'managing' suggests having data resources available to predict events or determine a part's use, such as the mission capability of a part or weapon system (i.e. aircraft). The decrease of parts can be attributed to many factors, some of which are due to the downsize of military effort, use of aging aircraft where parts are no longer available, large financial cost associated with part manufacturing, and time required to manufacture new parts in a dynamic environment (i.e. war zone). As a result of the decrease in parts, parts themselves have to be differently managed, preserved and well maintained, and repaired and reused. To better illustrate the scenario, imagine if in the year 1980, a particular component we will denote as Widget180 was stocked heavily to the point that when this component failed on a vehicle XYZ, the mechanic could immediately throw it out and replace it with a new one. Fast forward to 1990's, the military is downsizing, the manufacturer of Widget180 is no longer in business and no historical performance data has been recorded over the years on the Widget180. All performance data has been captured on vehicle XYZ. Basically the fact that vehicle XYZ failed was

recorded but the exact reasons or parts that lead to the failure is not. Imagine if for instance if either of these items failed, the engine, Widget180, axle, or some electrical component, the only data recorded was that vehicle XYZ failed. Then instead of allocating money for the purchasing or repairing of a Widget180, a new vehicle XYZ is purchased. This scenario illustrates the poor recording and acquisition maintenance data, which prevents an optimal analysis of parts and vehicles. Furthermore millions of dollars are lost. In order to effectively manage, maintain and reuse aircraft parts one must be aware of part related information and metrics. As an example, part related information may consist of data associated with part breakdown, part supply, requisition, maintenance, reliability and sustainment. Such information is generally resident in multiple remote data systems in heterogeneous or unreliable formats accessible and plagued with the problems mentioned in section 2. There are several million parts associated with records and data elements that require individual observation in order for the mappings to be executed accurately. Once mappings are captured effective analysis of data can be initiated. Throughout the government and many businesses, there exists the need for a stable documented methodology expressed as a software solution that can capture input from various sources, analyze the information, discover new information, link related information, and from said information allow the necessary personnel to make intelligent decisions.

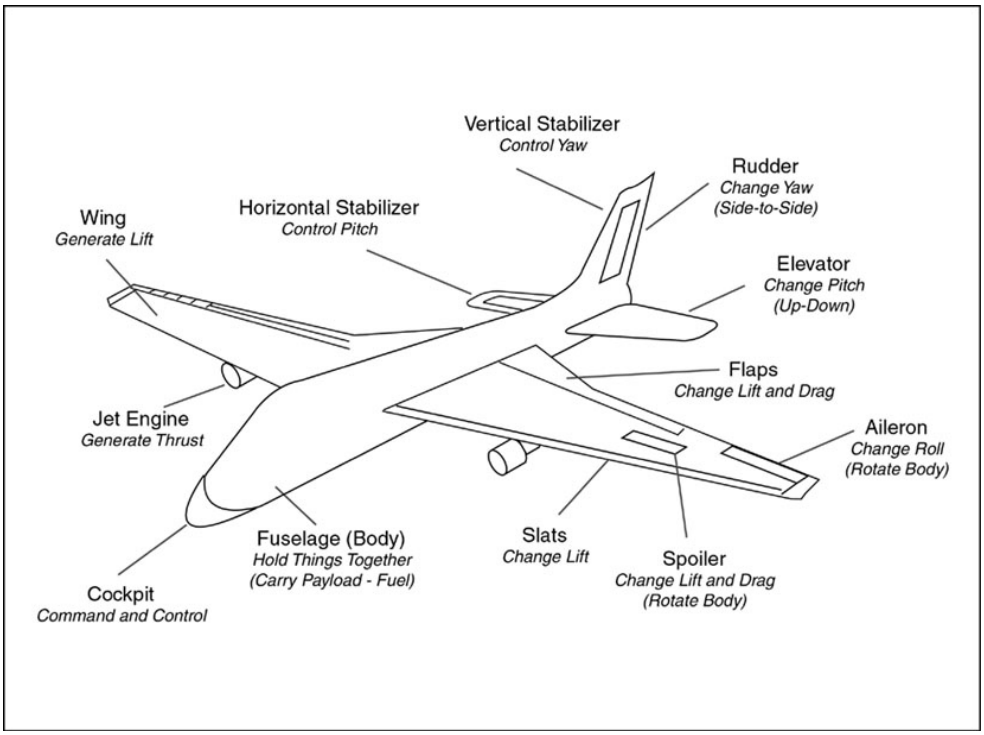


Figure 7. Diagram of airplane

4.1.1 Current Solution

Before presenting our work, we briefly describe the solution as it is presently being applied in the business work environment today. Presently, company employees are manually performing the mappings by analyzing record descriptions and associated figures and images. We note that this research is specific to the area of machinery parts. Using aircraft parts as an example, employees commissioned with the responsibility of mapping records from one table to that of another table are faced with the time consuming task of analyzing part descriptions (and any included reference information) from both tables to try to determine the best possible mapping. They must also refer to a part drawing or schematic to try to correlate the actual position of the part to that in which the description communicates. In interviewing some of these employees they stated that between seventy and eighty percent of their time is spent examining schematics while the other twenty to thirty percent of the time is spent analyzing the part description. Executing this process manually requires a huge number of man hours and more man hours suggests an increase in money and time to complete the effort. These are resources that could be saved or utilized in other areas, if the process were partially automated.

4.2 Disjoint Factorization and Discovery Items

In this section we present an approach to the selection and elimination of data elements denoted as Features. We also present an approach to the construction of data elements denoted as Discovery Items. Definitions of unfamiliar terms are given throughout the document. Table III depicts a sample record found in set B. It shows the features and the associated feature values.

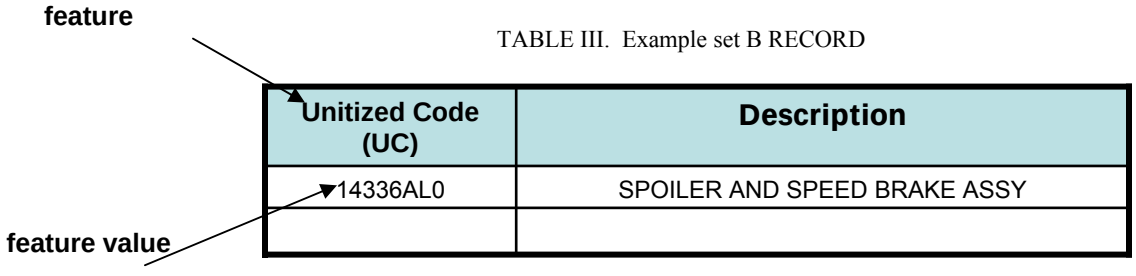


TABLE III. Example set B RECORD

Unitized Code (UC)	Description
14336AL0	SPOILER AND SPEED BRAKE ASSY

Definition 4.0: A *feature* is an attribute or field in a relation (i.e. table) identified as relevant to the discovery process and its values are processed during the data processing step [36, 39].

> "In credit card fraud, for instance, an important feature might be the location where a card is used. Thus, if a credit card is suddenly used in a country where it's never been used before, fraudulent use seems likely" [39].

In specific medical related cases important features might consist of age, weight, and ethnicity.

Definition 4.1: *Disjoint Factorization* (DF) is the process of decomposing objects (e.g. record fields) resident in data sets into smaller individualized elements.

Depending on the type of object, the concatenation (in the case of strings) or product (in the case of numerical values) of the smaller individualized elements will yield the original object. In this research we build upon the concept of Disjoint Factorization to develop methodologies to combat the MUDD classification problem. It is noted that several of the linkage solutions described above utilized multiple fields from a relation to determine record agreement. In figures 8a and 8b, we illustrate an application of DF using integer values.

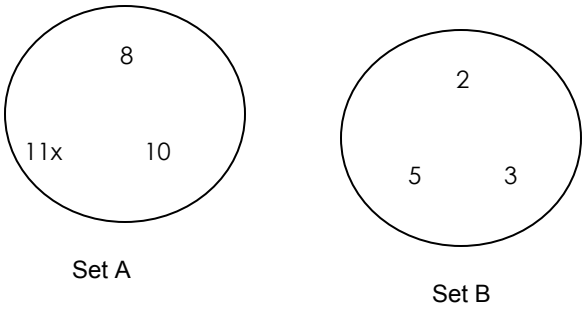


Figure 8a. Disjoint sets

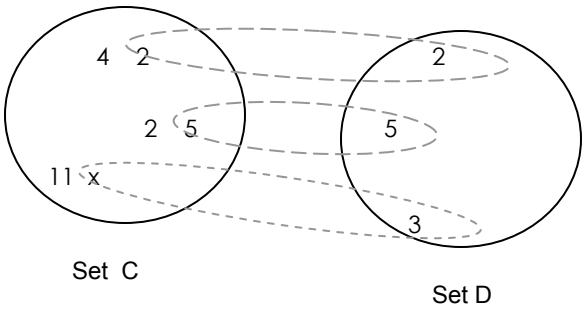


Figure 8b. Disjoint Factorization with Pseudo-intersections

In figure 8a, the sets A and B are clearly disjoint. With the application of disjoint factorization, we see in set C of figure 8b, the integer 8 is transformed into the values 4 and 2 and the integer 10 is transformed into the values 2 and 5. In addition 11x is transformed into the terms 11 and x. With these transformations pseudo-intersections can be created (see figure 8b).

Definition 4.2: Given two disjoint sets A and B, an application of disjoint factorization can be applied on the elements in either set A or set B or both set A and set B, to create two new sets C and D where $C \cap D$ is not empty. (Note: the elements of set A are used to create a set C and the elements of B are used to create a set D) The newly intersecting elements are denoted as *pseudo-intersections*. Pseudo-intersections are intersections of composite values or terms that are used to create legitimate relationships among the disjoint sets. A pseudo-intersection is denoted by the symbol $\underline{\cap}$. Using our example in figure 8b, we obtain $A \underline{\cap} B = C \cap D = \{2, 5, 3, x\}$ where 3 and x are unique values; however x is identified as a term that can be substituted for the value 3. Term substitutions can be realized in the case of synonyms or term aliases such as the two terms *bolt* and *screw*. Although these terms are unique, in some domains the two can be used interchangeably.

In this research, the principle of disjoint factorization is used with a record description consisting of one or more strings. Preliminary tests revealed that attempting to process a record description as a whole data object proved uneventful. By decomposing the description into subunits, pseudo-intersections among the data sets can be developed. Pseudo-intersections help to establish relationships between the two sets. They serve to increase the chances of creating mappings. We expound on this more in future sections.

In this research we deal specifically with data records with field values that describe aircraft machinery parts, namely part descriptions, part numbers, part quantity, etc. We select the part description field from both sets to apply a disjoint factorization operation. We eliminate the other fields and utilize only the description field as it is determined that the other fields yield little to no assistance in obtaining record mappings. This process is called feature elimination.

Definition 4.3. *Feature elimination* is the process of removing irrelevant features from a Data Field in an effort to eliminate redundant, ambiguous, irrelevant data for the data processing step [36].

A *data element* is a string. A record description consists of a set of explicitly ordered data elements.

Definition 4.4: A *classifier* is a term used to describe a collection of data elements. It is used to separate and organize data elements of a description into a distinct category.

The data elements extracted after a disjoint factorization application can be associated with one of N defined classifiers. Classifiers are defined by the data analyst. It is noted that classifiers are synonymous with features. The process of creating classifiers is done explicitly with the creation of the objects denoted as Discovery Items.

Definition 4.5*: Discovery items* are relevant criteria defined by the data analyst that must be identified during the mapping process in order to eliminate the number of comparisons needed to create accurate mappings among tuples in the disjoint data sets A and B.

The defined discovery items should be used in the creation of classifiers because during the mapping process when specific classifier values are obtained they will give reference to at least one of the defined discovery items. Each feature should map to one of the Discovery Items. Thus given a set of features $F = \{f_1, f_2, \ldots f_n\}$ and a set of Discovery Items $D = \{d_1, d_2, ..d_m\}$ there is a many to one relationship between sets F and D. Given the following relationships $\{f_1,f_2\} \rightarrow \{d_1\}$, $\{f_3\} \rightarrow \{d_2\}$, and $\{f_4\} \rightarrow \{d_3\}$, it can be understood that features f_1 and f_2 can be used to reference d_1 and f_3 can used to reference d_2. As an example, a record description of an automobile part is

Ford front driver side door window lever.

The individual terms that compose the record description are denoted as data elements. In the given record description there are seven data elements present. When observing automobile parts and system specifications, one quickly realizes that an automobile is composed of several systems such as a fuel system, engine system, structure and doors, electrical system, light system, instrument control system, and

so forth. Depending on the type of automobile, the system names or system part composition could vary as well. The automobile system and sub-system names qualify as types of information that can be used as discovery items. These are items one must identify in order to establish a mapping among two automobile parts records. Example discovery items could be 1) Automobile make, 2) System of part and 3) Part name. During the processing step, if one is able to identify these three items from the description then establishing a mapping will become less difficult. We define four classifiers for this example: Positional, Descriptive, Noun and Primary. (These classifiers/features will be used throughout the remainder of the paper.) Positional classifiers consist of values which refer or give some evidence to the position of the part on the automobile (e.g. back, rear, front). A Descriptive classifier is defined to capture any descriptive information about the part (e.g. red, aluminum). These values are normally adjectives. Noun classifiers are nouns within the description. The Primary classifier captures the item of primary importance within a given record. *This term corresponding to a Primary classifier is generally the last noun in the record description*. In the example record description above the term *lever* is the primary object of concern. This object is categorized as a Primary classifier value or Primary term. In attempting to create a mapping for the above record, one would not look to associate a *lever* with a *tire*; rather attempts should be made to identify another record which deals with a *lever* or has some association with a *lever*. All other terms in the record description serve as extra information that describes pertinent information about a part such as a part's location and use. When attempting to locate a record to link to this one, one would not want to begin analyzing records which have *door* or *window* as a Primary term, rather records having a Primary term *lever* or alternative terms for *lever*. Using table IV, the positional, descriptive and noun classifiers give some idea of the appropriate equipment and system to initiate the mapping process.

Table IV. Classifiers and Associated values

Positional Classifier	***Descriptive Classifier***	***Noun Classifier***	***Primary Classifier***
Front	side,driver,Ford	window,door	Lever

The Primary classifier value informs the specific type of part to reference (discovery item 3). The Noun Classifier values give some information which references discovery item two, the system of the

automobile. The noun classifier values also serve to eliminate other systems; for instance the fuel system, electric system, etc. The positional and descriptive classifier values give a reference to the make of the vehicle and the *type* of lever the algorithm should look for. In addition the classifiers serve to eliminate the irrelevant lever types and automobile models that do not meet the criterion communicated by the classifiers. *The features/classifiers are constructed by the researcher.* The term feature and classifier can be used synonymously once the features are created.

Definition 4.6: *Feature creation* is the process of creating features from data in the Data Field.

Determining the features of significance is vital in both record linkage and data mining. Features denote the data components of a record that are most relevant to ensure the effectiveness of the mapping or analysis algorithm. In [36], there are three basic ways noted to create features; construction, extraction and mapping. Feature construction is the process of forming new features by combining or manipulating other features. Feature extraction is the process of creating a new set of features from capturing data from the original data. A popular example of this is the process of extracting certain types of edge information from photographs as a means to assist in classifying photographs. Feature extraction techniques are regarded as domain-specific [36]. Feature mapping is the process of representing data differently to obtain new features. Applying a Fourier transform to times series data inundated by noise can help to reveal any previously undetected periodic patterns.

In the following section we present an approach to construct features based on a context free grammar.

4.3 Grammar-Based Feature Construction

The new features are constructed based on an analysis of the data population being processed. To analyze the data population one could study the entire population record by record. Alternately one could identify and study patterns communicated by the data population as well as relationships among various data elements. Grammars are tools that can be used to describe and analyze languages, generate languages, and

parse languages [26]. In our research, the set of all record descriptions in the Data Field is viewed as a language, L. The record descriptions are synonymous with sentences in a language. We develop an algorithm to generate a context free grammar (CFG) given a set of sentences as input. We look to study the syntax of a subset of records from the language L to understand the structure of the records in the Data Field better. The algorithm does not try to infer a grammar based on a finite set of strings S from a language. Such an algorithm attempts to generate a grammar, G, as well as making the claim that G will generate a general language, L, which includes the strings, S, and excludes the strings in a language, L′. Algorithms of this type generally take Positive samples and Negative samples as input. Positive samples are valid sentences in a language L. Negative samples are invalid sentences of a language L and can properly be denoted and sentences in a language L′. The algorithm uses the negative samples to assist it in generating valid production rules which will only form sentences similar to those found in the Positive sample.

In this research standardized negative samples are not available as the naming convention of the records in the Data Field communicates little to no information about a formal standard or structure. With little to no formal structure for the naming convention of the records, a reliable set of negative samples is difficult to obtain. As a result the algorithm designed and implemented in this study generates a grammar based solely on the positive samples it receives as input. For the language L consisting *only* of the strings,

```
ab
aabc
aaabbc
aaaabbbc
```

The generated grammar is seen in Figure 9.

```
START → S1 | S2 | S3 | S4
S1 → 9-3-11
9 → aaa
3 → ab
11 → bbc
S2 → 4-3-5
4 → aa
5 → bc
S3 → 0-3-2
0 → a
2 → c
S4 → 3
```

Figure 9. Generated Context Free Grammar

In this section we present an algorithm to generate a context free grammar (CFG) which takes a language L (i.e. finite set of sentences) as input. The algorithm was programmed in the Java programming language. The source code can be seen in Appendix A. In this section we present pseudo-code with an overview of its operation. In figure 10 a list of initial declarations are shown. The variable COMB is a string array that holds all valid combinations of alphabet obtainable by the input sentences. The string TMP_SENT_FORM is temporary storage that holds a newly constructed sentential form of a sentence being processed. NON_TERM and PRODUCTION are parallel arrays that hold the left and right side of a production rule respectively. The string START_PROD contains the right side of the production rule associated with the start symbol of the CFG. The structure SENTENCE is used to store three important pieces of information about a specific sentence in the input language L. Each SENTENCE object stores 1) an input sentence in the text string, 2) a list of sentential forms, which are placed in the SENT_FORM string array and 3) a list of indices that correspond to values in the COMB string array. The list of indices is stored in the P_LIST integer array.

```
String[] COMB
String TMP_SENT_FORM
String[] NON_TERM
String[] PRODUCTION
String START_PROD

struct SENTENCE{
String text
String[] SENT_FORM
int[] P_LIST
}
```

Figure 10. Algorithm declarations

In order to understand the algorithm we offer an explanation by example. The algorithm begins with the function *GenerateCFG()* shown in figure 11.

```
Line 1:   Read in sentences of language in object SENTENCE[i]
Line 2:   Sort sentences by length in ascending order
Line 3:   FOR i=1 to n, (n is total number of sentences)
Line 4:     Identify all possible pattern combinations in SENTENCE[i]
Line 5:     Insert pattern combinations in array COMB
Line 6:   Sort items in array COMB
Line 7:   CreateSententialForms()
Line 8:   BuildProductionList()
Line 9:   TranslateRecursion()
Line 10:  RemoveUnitProduction()
```

Figure 11. Algorithm: GenerateCFG()

As sample input we use the three sentences: *abc, bc* and *abcc.* In line 1 of figure 11, each sentence is read into a SENTENCE object. Line 2 calls for the sentences to be sorted by length from shortest to largest. The result of this operation yields the sentence order: *bc, abc, abcc.* In lines 3-5, for each input sentence all valid string combinations that can be obtained are identified and stored in the string array COMB. We denote the string combinations as patterns. No duplicate patterns are stored in COMB. In figure 12 it is seen that the input sentences yield corresponding patterns.

bc := b, c, bc
abc := a, b, c, ab, bc, abc
abcc := a, b, c, ab, bc, cc, abc, bcc, abcc

Figure 12. Resultant sentence patterns

In line 6, the patterns are sorted by length in ascending order. Using our example sentences the following patterns are stored in COMB{ a, b, c, ab, bc, cc, abc, bcc, abcc }. The function *CreateSententialForms()* in figure 13 is called next.

```
Line 1:   FOR i=1 to N
Line 2:     FOR k=COMB.size to SIZE_TWO
Line 3:       IF COMB[k] is substring of SENTENCE[i]
Line 4:         Replace all occurrences of COMB[k] in SENTENCE[i] with the
                value k (convert k to a string) and assign to TMP_SENT_FORM[x]
Line 5:         Insert value k in P_LIST array
Line 6:         FOR j=k-1 to SIZE_TWO
Line 7:           Replace all occurrences of COMB[j] in TMP_SENT_FORM[x] with
                  the value j (convert j to a string)
Line 8:           IF TMP_SENT_FORM[x] has been completely transformed
Line 9:             BREAK
Line 10:        IF TMP_SENT_FORM[x] has been completely transformed AND
                TMP_SENT_FORM is not in SENTENCE[i].SENT_FORM array
Line 11:          Insert TEMP_SENT_FORM into SENTENCE[i].SENT_FORM array
Line 12:          x++
```

Figure 13. Algorithm: CreateSententialForms()

This function is responsible for creating valid sentential forms of each input sentence by using the patterns stored in COMB. In specific for each input sentence all valid patterns found in COMB that can be used to create the current input sentence when a substitution operation is applied are identified. In line 1 a loop is used to process each of the N input sentences. In line 2 a loop is used to initiate the processing for each input sentence. The patterns in COMB having a length equal to the sentence being processed initiate this

process. The iterations continue until the last pattern in COMB having a size two is reached. Patterns in COMB of length one are not considered. Patterns having a size of two are the last patterns to be considered. This is seen in line 2 and 6 as represented by the variable SIZE_TWO. The variable SIZE_TWO holds the index of the last pattern of size two in the array. If the pattern being processed (referenced by COMB[k] is a substring of the sentence being processed (referenced by SENTENCE[i]), all occurrences of the pattern found in the sentence is replaced with the index of the pattern in the vector COMB (line 4). This process is called a transformation. The pattern initiating the transformation is termed the primary combination. The index of each primary combination is stored in the integer array P_LIST (line 5). When a transformation is initiated a sentential form of the sentence being processed is created. The temporary sentential form is stored in the TMP_SENT_FORM array. In line 6 a new loop is initiated starting with the next pattern that immediately follows the primary combination string retrieved from COMB[k]. This loop is started in order to maintain the position of the loop in line 2. The action in line 7 occurs until the sentence in TEMP_SENT_FORM has been completely transformed or until the loop ends (line 9). In the event that the input sentence has been completely transformed and the newly created sentential form has not already been seen, the sentential form is inserted in the SENT_FORM array found in the SENTENCE object associated with the input sentence being processed (line 11). This process continues for the current sentence being processed until all patterns in COMB have been processed. Once all patterns have been processed, the operations are repeated for the next input sentence. Below in table V, VI and VII are the results of the transformation for each of the sample input sentences. These new transformations are denoted as temporary sentential forms. Each sentence has a set of temporary sentential forms associated with them. Each set of sentential forms are stored in the SENT_FORM array in the SENTENCE object associated with the given input sentence. It is noted that the valid sentential forms are created using the indices of the patterns found in COMB.

Table V. Sentence ‘abcc’ sentential form

Input Sentence = ***abcc***		
SENT_FORM index	COMB index *(primary combination)*	SENT_FORM Value (No. denote COMB indices)
1	9	9
2	8	1-8
3	7	7-3
4	6	4-6
5	5	1-5-3

Table VI. Sentence ‘abc’ sentential form

Input Sentence = ***abc***		
SENT_FORM index	COMB index *(primary combination)*	SENT_FORM Value (No. denote COMB indices)
1	7	7
2	5	1-5
3	4	4-3

Table VII. Sentence ‘ab’ sentential form

Input Sentence = ***ab***		
SENT_FORM index	COMB index *(primary combination)*	SENT_FORM Value (No. denote COMB indices)
1	5	5

The primary combination indices associated with each sentence are stored in the integer array `P_LIST` for future use. See figure 14 for the contents of each sentence’s `P_LIST` array. Note the values in `P_LIST` are stored in descending order.

abcc.P_LIST ={ 9, 8, 7, 6, 5 }
abc.P_LIST= { 6, 5, 4 }
ab.P_LIST = { 5 }

Figure 14. Sentence P_LIST values

Next the function *BuildProductionList()* shown in figure 15 is called. This function builds the list of productions for the CFG. The list is created by comparing the `P_LIST` values of the current longest input sentence, denoted by variable `i`, to that of the next shortest, denoted by `j`. This executes by observing the

values in the respective P_LIST array in an effort to identify the smallest value common to both sentences. (NOTE: this is the process of looking for pattern commonality between multiple sentences). The variable j in line 6 keeps a reference to the sentence that occurs immediately after the sentence referenced by variable i. In line 3 the value at location index in P_LIST is assigned to the variable val_One. The values in P_LIST are accessed from the end of the array to the beginning. The variable k in line 7 is initially assigned the size of the P_LIST array in the SENTENCE[j] object and is used to index into its P_LIST. In line 9 the P_LIST values of SENTENCE[j] are compared to the values of SENTENCE[i]. If the values are equal the algorithm breaks out of the inner three loops (i.e. line 4, line 6, line 7) and begins creating productions in line 14. Otherwise if the P_LIST value of SENTENCE[j] is greater than that of SENTENCE[i], the algorithm breaks out of the innermost loop (line 7); the value of j is decremented and the P_LIST comparisons begin again.

```
Line 1:   FOR i=N to 1
Line 2:     index = SENTENCE[i].P_LIST.size
Line 3:     val_One = SENTENCE[i].P_LIST[index]
Line 4:     while(!FOUND OR index >0)
Line 5:     index--
Line 6:     FOR j=(i-1) to 1
Line 7:       FOR k=SENTENCE[j].P_LIST.size to 1
Line 8:         val_Two = SENTENCE[j].P_LIST[k]
Line 9:         if val_One == val_Two || val_Two > val_One
Line 10:          break
Line 11:      if val_One == val_Two
Line 12:        FOUND=true
Line 13:        break
Line 14:    NON_TERM[q]="S" + x
Line 15:    START_PROD=START_PROD + NON_TERM[q] + "|"
Line 16:    PRODUCTION[q]= SENTENCE[i].SENT_FORM[index]
Line 17:    q++
Line 18:    x++
Line 19:    WHILE Tokenize(SENTENCE[i].SENT_FORM[index]) != NULL
Line 20:      NON_TERM[q]=GetNextToken()
Line 21:      Value=ConvertToInteger(NON_TERM[q])
Line 22:      PRODUCTION[q]=COMB[Value]
Line 23:      q++
Line 24:    NON_TERM[q]="S" + x
Line 25:    START_PROD=START_PROD + NON_TERM[q] + "|"
Line 26:    PRODUCTION[q]= SENTENCE[j].SENT_FORM[k]
Line 27:    q++
Line 28:    x++
Line 29:    WHILE Tokenize(SENTENCE[j].SENT_FORM[k]) != NULL
Line 30:      NON_TERM[q]=GetNextToken()
Line 31:      Value=ConvertToInteger(NON_TERM[q])
Line 32:      PRODUCTION[q]=COMB[Value]
Line 33:      q++
Line 34:    j=0;
Line 35:    BREAK
```

Figure 15. Algorithm: BuildProductionList()

If the algorithm reaches line 14, this suggests that `SENTENCE[i]` and `SENTENCE[j]` both have a sentential form that have a combination pattern in common. In line 14 a non-terminal value is created. The sentential forms are assigned a non-terminal value (e.g. Sx, where x is an integer value). This is seen in lines 14 and 24. The non-terminals beginning with S are assigned to the right-side of the start symbol as seen in lines 15 and 25. The variable `q` keeps track of the number of productions, while the variable `x` keeps track of the number of non-terminals that begin with the letter S. In line 19 the sentential form in `SENTENCE[i]` that was found to have a pattern in common with a sentential form in `SENTENCE[j]` is tokenized. We note that all sentential forms associated with a sentence are stored as a string of numbers delimited by the '-' symbol. *Any symbol can be used as a delimiter. For explanatory purposes we use the '-' symbol.* In specific the numbers denote the indices of the patterns found in `COMB`. Each number tokenized becomes a new non-terminal and is inserted in the non-terminal array `NON_TERM` in lines 20 and 30. In lines 21 and 31 the values are converted into an integer and the value is used to index into the `COMB` array to obtain a pattern. This pattern string is assigned to the `PRODUCTION` array in lines 22 and 32.

Using our example, comparing the values of `P_LIST` associated with sentences '*abcc*' and '*abc*', 5 is the lowest value common to both. See table V and table VI for a view of the common sentential forms. Thus the temporary sentential form 1-5-3 is used for *abcc* and temporary sentential form 1-5 is used for *abc*. This assignment creates a set of productions, where the newly assigned non-terminals appear on the left side and the sentential forms appear on the right. In addition new productions rules are made using the non-terminals found in the sentential forms. The non-terminals in the sentential forms are placed on the left side of a production and the pattern associated with it becomes the right side. These new non-terminal values are actually the index values into `COMB`. For instance, using `COMB`, the pattern at index 1 is *a* and the pattern at index 5 is *bc*.

S1 → 1-5-3
1 → a
5 → bc
3 → c
S2 → 1-5
START → S1 | S2

Comparing the values of P_LIST associated with sentences '*abc*' and '*bc*', again 5 is the lowest value common to both, so the temporary sentential form 1-5 is used for *abc* and temporary sentential form 5 is used for *bc*. The sentential form 1-5 is already present in the production list so it is disregarded. We add the following productions,

S3 → 5
START → S1 | S2 | S3

Resulting productions are:

S1 → 1-5-3
1 → a
5 → bc
3 → c
S2 → 1-5
S3 → 5
START → S1 | S2 | S3

The function *TranslateRecursion()* shown in figure 16 processes the right side of all newly created productions to find forms common within other productions. It is noted that all right side values are stored in the `PRODUCTION` array. This step captures production rules that have a length greater that 1. In each production a single term is separated by a '-' symbol. Productions consisting of all terminal symbols are not considered. A list of all productions previously used in this step is maintained.

In the example productions, the right side of S2→1-5 is common in S1→ 1-5-3. The right side 1-5 is replaced with non-terminal S2 in production S1→1-5-3 to yield the following result.

S1 → S2-3
1 → a
5 → bc
3 → c
S2 → 1-5
S3 → 5
START → S1 | S2 | S3

```
Line 1:   FOR i=1 to PRODUCTION.size
Line 2:     if PRODUCTION[i].length > 2 AND PRODUCTION[i] != all terminals
Line 3:       FOR j=1 to PRODUCTION.size
Line 4:         if (i !=  j) AND PRODUCTION[i] is a substring of PRODUCTION[j]
Line 5:           Replace substring with NON_TERM[i]
```

Figure 16. Algorithm: TranslateRecursion()

This next function *RemoveUnitProductions()* is optional. The pseudo-code for this function is not presented in this work; however the source code is listed in Appendix A. This function removes unit productions. Unit productions are productions of the form $\alpha \rightarrow \beta$ where α and β are both non-terminals. In the example productions, S3→ 5 denotes a unit production so everywhere S3 is found the non-terminal 5 is substituted. We have the resulting productions.

S1 → S2-3
1 → a
5 → bc
3 → c
S2 → 1-5
START → S1 | S2 | 5

Once the grammar is constructed, we examine it in an attempt to identify patterns or correlations within the productions. Using a subset of the disjoint records to create a grammar eliminates the task of observing millions of records. Also when the data is limited in its reliability, this provides a means to create additional structure and reliability. Features are defined based on the discovered patterns and correlations.

4.4 Feature-Based Record Object

Data records from both sets (i.e. disjoint sets) are translated to associate their data elements to the new features and to exclude the features eliminated during the feature elimination step. All records are translated into a corresponding Feature-Based Record Object.

Definition 4.7: A *Feature-based Record Object* (RO) is an object corresponding to a data record, where the elements of the record are associated with specifically defined features or classifiers.

There are two basic types of ROs, target record objects (TRO) and candidate record objects (CRO). Let the data mapping algorithm be defined as a function F(x). Given the function F(x) = y, x is a domain element and y is a range element. The target record objects are elements of the domain and the candidate record objects are elements of the range. Record Objects consist of feature values with corresponding weights. The following binary operations can be performed on record objects, union, intersection, and difference.

Definition 4.8: A *Target Record Object* (TRO) is a domain item containing target record classifier data, that maps to a specific candidate record object.

Definition 4.9: A *Candidate Record* (CRO) Object is a range item which is mapped to a target record item.

Definition 4.10: The *Union* of two record objects S and T denoted by ($S \cup T$) is the process of combining the classifier values from two Record Objects.

Definition 4.11: The *Intersection* of two record objects S and T denoted by ($S \cap T$) is the process of capturing the common classifier values from two Record Objects; this process yields a score.

Definition 4.12: The *Difference* of two record objects S and T denoted by ($S - T$) is the process of removing from S the classifier items that are found in T.

Given two sets A and B, set A contains all target record objects and set B consists of all candidate records, $A=\{t_1,t_2,t_3,\ldots,t_n\}$ and $B=\{c_1,c_2,c_3,\ldots,c_m\}$. Each object t_i and c_k consists of data elements which correspond to specific classifier values. As an example if there were four defined classifiers each element t_i in A and c_k in B consists of values associated with one of the four classifiers E_1, E_2, E_3, and E_4. See Table III for an example. To access the list of values of a given classifier E_1 we use the dot reference operator. For example E_1.values[i] references the i^{th} value of classifier E_1.

We now discuss how to quantify the Record Objects so as to provide a means of evaluation and validation. Quantifying the contents of the record objects provides a mechanism for measuring and evaluating relationships among the different record objects. Let there be a set of classifiers $F=\{E_1,E_2,E_3,..,E_n\}$, each classifier has an associated weight value w_i. There exists a set of weights $W=\{w_1,w_2,w_3,..,w_n\}$ where each w_i corresponds to a classifier E_i in F. The weight values will be used in computing a quality of match value. Note the data elements associated with a specific classifier are denoted as classifier values or feature values. After an application of a binary operation on two objects, a new Record Object is produced. It is noted that each record object has a score which corresponds to the weights associated with the data

elements within the object. The score is computed using the function Score(x), where x is a record object. The classifier values within an RO, along with defined weight values can be used to compute a value denoted as an Actual score.

Definition 4.13: An *Actual score* is a numerical value assigned to a record object based on its feature values. It is the sum of the weights w_i times the number of classifier values associated with the classifier E_i, $\sum_{i=0}^{n}(w_i \times E_{i.size})$, where n equals the number of defined classifiers and $E_{i.size}$ denotes the total number of values within the record object which corresponds to a classifier E_i. A Target Actual score is associated with every TRO and a candidate actual score is associated with every CRO. In figure 17 we express the target and candidate record objects with a Venn Diagram.

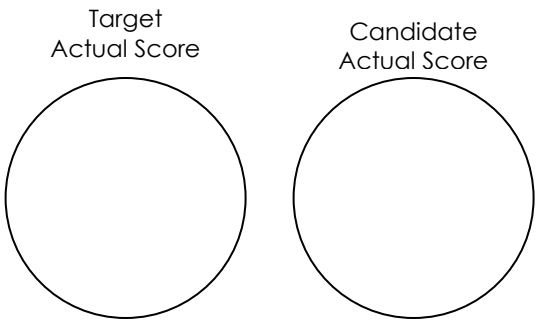


Figure 17. TRO and CRO represented as a Venn Diagram

For explanatory purposes, Venn Diagrams will be used to communicate aspects of the linkage strategy. The target actual score and candidate actual score is computed using the Actual score formula previously mentioned. A numerical value denoted as a *Candidate Match Score* is the total score associated with the classifier values in which a candidate record object has in common with the target record object. See figure 18 for an illustration. See equation (4.1).

Candidate Match Object (CMO) = Target Record Object (TRO) ∩ Candidate Record Object (CRO) **(4.1)**
Candidate Match Score = ***Score***(CMO)

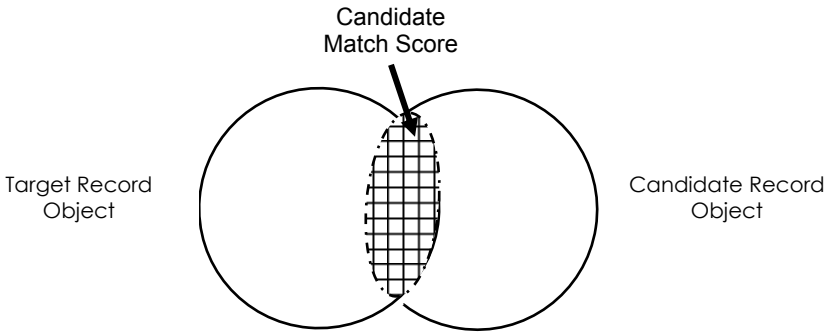


Figure 18. Candidate Match Score

The *Insertion Score* is the value associated with the remaining classifier items that must be added to the Candidate Match Score making the sum of the Insertion Score plus the Candidate Match Score equal to the Target Actual score. See figure 19 for an illustration. See equation (4.2).

Insertion Score = ***Score***(Target Record Object – Candidate Match Object) **(4.2)**

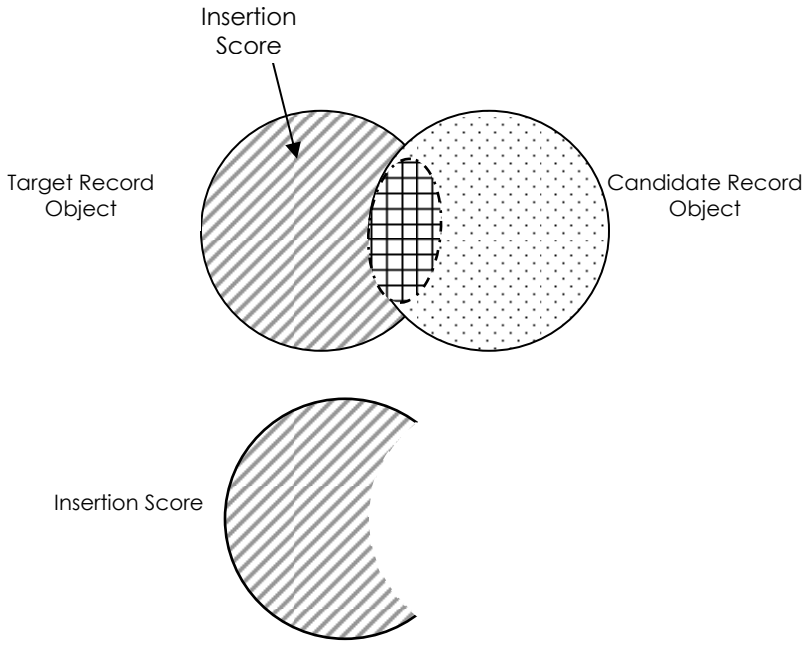


Figure 19. Insertion Score

The *Deletion score* is the value associated with the irrelevant classifier items that must be eliminated from a candidate record object. These items have nothing in common with the items in the Target Record Object. See figure 20 for an illustration. See equation (4.3).

Deletion Score = ***Score***(Candidate Record Object – Candidate Match Object) **(4.3)**

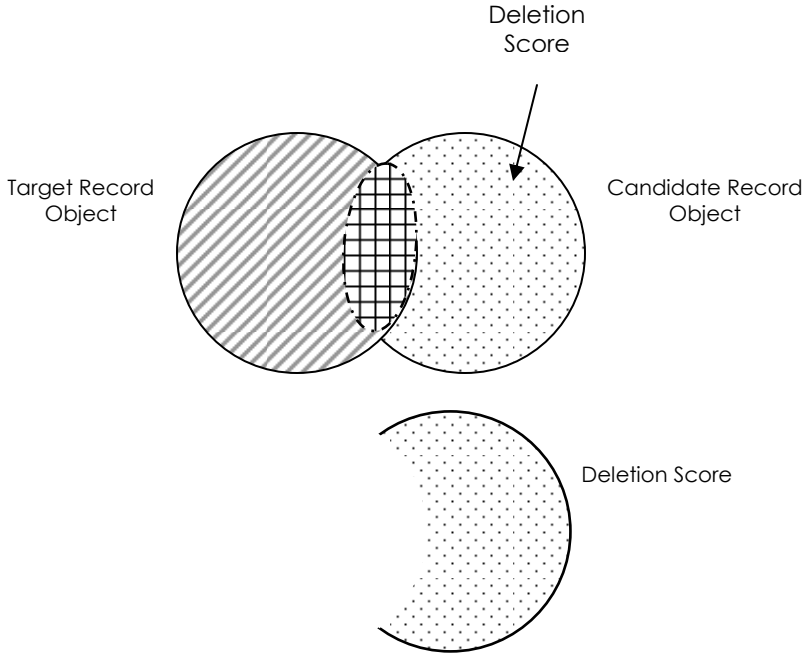


Figure 20. Deletion Score

A score denoted as the *Object Translation Score* is the sum of the Insertion Score and Deletion Score. See figure 21 for an illustration. Specifically, the value of insertion and deletion operations applied during the translation process is denoted as *Object Edit Distance or Object translation score*, much like that of edit distances described in an approximate string matching algorithm. Given a TRO with a target actual score a_i, a probable CRO mapping with a computed Object translation score of zero indicates a CRO that maps directly to the TRO. Furthermore, the closer a CRO's object translation score is to zero the more likely that the CRO is the better mapping/classification for the TRO.

Object Translation Score =***Score***((TRO – CMO) U (CRO – CMO)) **(4.4)**
= Insertion Score + Deletion Score

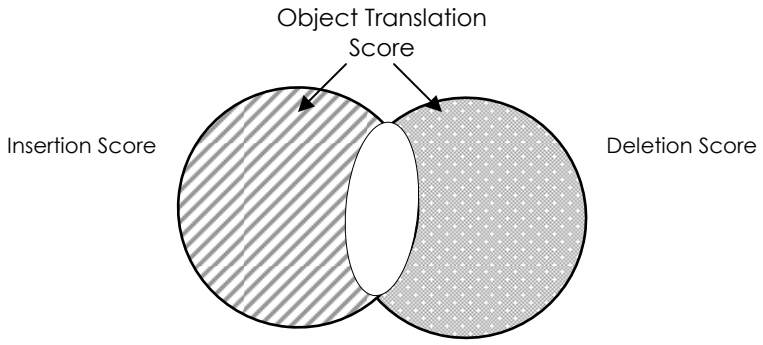


Figure 21. Object Translation Score

We present detailed explanations in two examples to follow. In this first example, assume that a target record from set A is selected. The record description is *CHORD, UPPER, RIB, AIL. OUTBD,.. WING STA. 729.00 (LH).* A candidate record from set B with the description *SPOILER AND SPEED BRAKE ASSY* is analyzed to determine its relevance. See tables VIII and IX for a description of the complete records from each set. An application of Disjoint Factorization is applied to the descriptions and the individual data elements are classified into one of the four defined features (i.e. classifiers). See table X and table XI.

TABLE VIII. Example set A RECORD

Figure Index	Part Number	IHS Num	Description	Units Per Assy	Use Code	XY Code
53	1239-33-2		CHORD, UPPER, RIB, AIL. OUTBD,.. WING STA. 729.00 (LH)	REF	A	12--9-

TABLE IX. Example set B RECORD

Unitized Code	Description
14336AL0	SPOILER AND SPEED BRAKE ASSY

TABLE X. Set A RECORD: *CHORD, UPPER, RIB, OUTBD AIL.,..WING STA. 729.00*

Primary Element	Positional Element	Descriptor Element	Noun Element	Number Element
CHORD	UPPER, OUTBOARD,LEFTH		RIB, AILERON	WING STATION 729.00

TABLE XI. Set B RECORD: *SPOILER AND SPEED BRAKE ASSEMBLY*

Primary Element	Positional Element	Descriptor Element	Noun Element	Number Element
BRAKE		SPEED	SPOILER	

The records are translated into Record Objects and their respective Actual scores computed. In figure 22 the number in the feature boxes denote the weight values associated with a specific feature. For example the Primary Element has a weight of 9 and a Noun Element has a weight of 4. The total value for the Noun Elements seen in the Target Record Object is 8. This value is computed by multiplying the weight value times the number of available Noun elements 2. In this example, the target actual score is 26 and the candidate actual score is 15. See figure 22 for an illustration of the record objects. The following scores are computed next.

- ***Candidate Match Score*** = ***Score***(Target ∩ Candidate) = 0
- ***Insertion Score = Score(***TRO – CMO) **=** 26
- ***Deletion Score*** = ***Score***(CRO - CMO) = 15
- ***Object Translation Score***= Insertion Score + Deletion Score = **41**

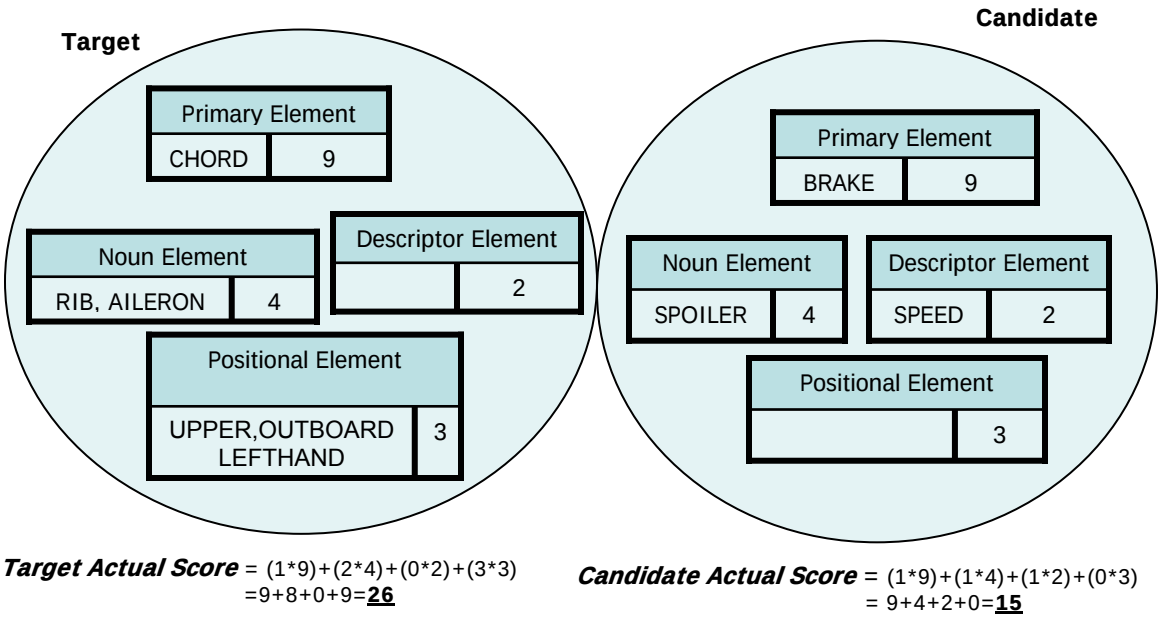


Figure 22. Target Actual Score and Candidate Actual Score Computation

The Candidate Record Object has no elements in common with the target record object so the Candidate Match score is zero. To compute the Insertion Score it is noted that in order for the candidate object to have all the elements of the target object all the elements must be added or inserted into the candidate object. The value of this operation is 26, yielding an insertion score of 26. The candidate object must

remove all of its initial data elements as they yield no relationship to the elements in the target object. The value of that operation is 15, yielding a deletion score of 15. The result of the operations (i.e. translation) produces Insertion Score + Deletion Score = 41, yielding an Object translation score of 41.

In the second example, we test the relevance of another record from set B denoted as *RIB*. The record description elements are associated with the corresponding feature. See table XII.

TABLE XII. Set B RECORD: *RIB*

Primary Element	Positional Element	Descriptor Element	Noun Element	Number Element
RIB			RIB*	

The element RIB appears in both the Primary element and Noun Element columns because we have identified a rule we denote as the One-Up Rule.

Definition 4.14: The *One-Up Rule* states that if the Primary term in a candidate record does not match the primary term in the target record, but matches a Noun term in the target record, then classify the Primary term as a Noun term instead.

The record is translated into a record object. See figure 23 for an illustration.

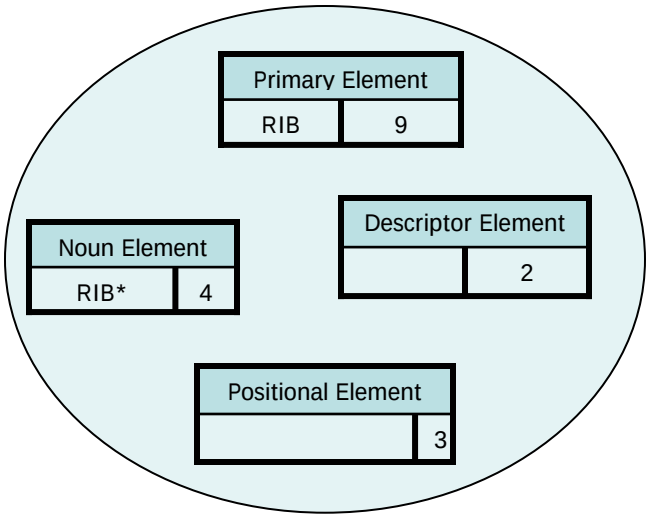


Figure 23. Candidate Record Object for record *RIB*

Alone, the new record object *RIB* does not have a high Actual score. It is possible to increase a Candidate Actual Score by adding additional feature values to a CRO. This is done by applying the union operation on a CRO and another record object. This is known as an Enhancement Step. We introduce the concept of an Enhancement Step.

Definition 4.15: An *Enhancement Step* is the process of adding additional information to a Candidate Object in an effort to boost its chance of being the top candidate for a mapping.

Definition 4.16: *Enhancement Data* is any data related to a candidate record (CR) being processed that would provide vital information about the CR to enhance the quality of match score.

Enhancement data can be located in the immediate data source or in an external data source. Examples of enhancement data are document section headers, figure titles/headers related to the candidate record, documents referenced by the CR, and synonyms. In figure 24 we provide an example of a scenario where the records are in a text document. The records are preceded by a section header which gives some general description of the section and throughout the document, figures are present.

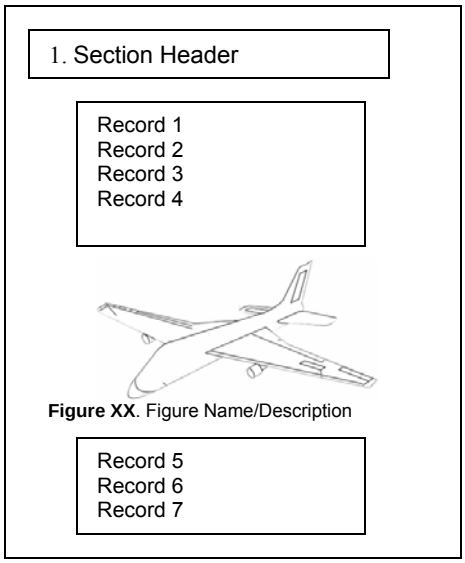


Figure 24. Example document with Enhancement Data

The records in the document reference different figures as well as parts within the figures. The figure header/description and section header can be used as enhancement data as they serve to add value or context to the records being processed. The features defined during the feature construction phase add extreme flexibility to the mapping process which enables a nonrestrictive data processing step as it allows for new data elements to be merged into the data records mid-stream in an effort to provide better quality of match scores and thus optimal mappings. Feature Enhancement Data can be identified both before and during the data processing step. *In order to be used the Enhancement Data is translated into a record object (RO) and merged with a candidate object during the discovery process*. The information that is identified beforehand is denoted as static checkpoints, while dynamic checkpoints are characterized by their ability to change over time.

Definition 4.17: *Static Check points* are pre-defined data elements in pre-defined locations within a data source (e.g. table, document) where the data elements are identified as additional evidential information that can be utilized to enhance a Candidate Object's probability of being mapped to a target object. After considerable analysis of the data, static check points are identified by the researcher before the data processing step as areas which hold additional information that could prove vital to the mapping process. Examples of static check points could be 1) section headers in documents, 2) figure names/headers and 3) system names.

Definition 4.18: *Dynamic Check points* are data elements that are discovered throughout the data processing step and are identified as additional evidential information to be utilized to enhance a Candidate Object's probability of being mapped to a target object. An example of this may be new terms that are learned over time during the processing step such as synonyms or alternate word terms.

An Enhancement step can be applied to the RIB record object. It is known beforehand that this particular part RIB, denoted by the record, belongs to the aircraft system denoted as OUTBOARD AILERON ASSEMBLY. We use the aircraft system information as Enhancement Data. The system information is translated into a RO and the associated actual scores are computed. See figure 25 for an illustration of the translation. The record object RIB is enhanced using its system information. See figure 26 for an illustration of the enhanced record object.

- ***Candidate Actual Score*** = ***Score***(CRO U System Object)
  = ***Score***(CRO_prime)
  = 8 + 3 = **11**
- ***Candidate Match Score*** = ***Score***(TRO ∩ CRO_prime) = 11
- ***Insertion Score = Score(***TRO – CRO_prime) **=** 26 – 11 = 15
- ***Deletion Score*** = ***Score***(CRO_prime - CMO) = 11 – 11 = 0
- ***Object Translation Score***= Insertion Score + Deletion Score = 15 + 0 = **15**

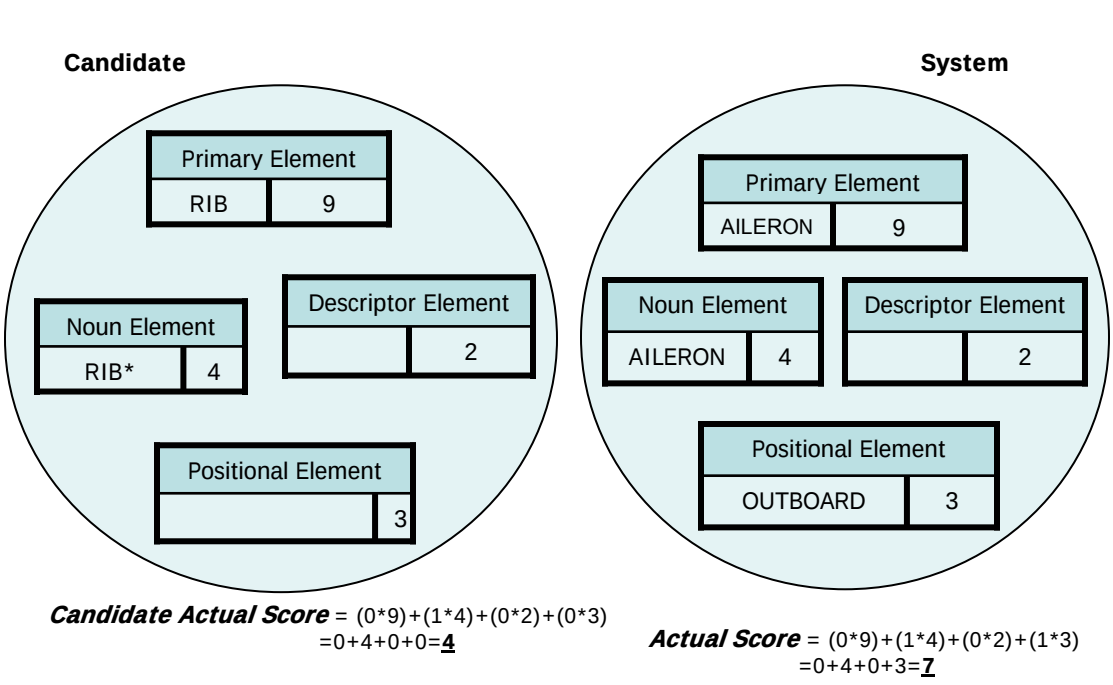
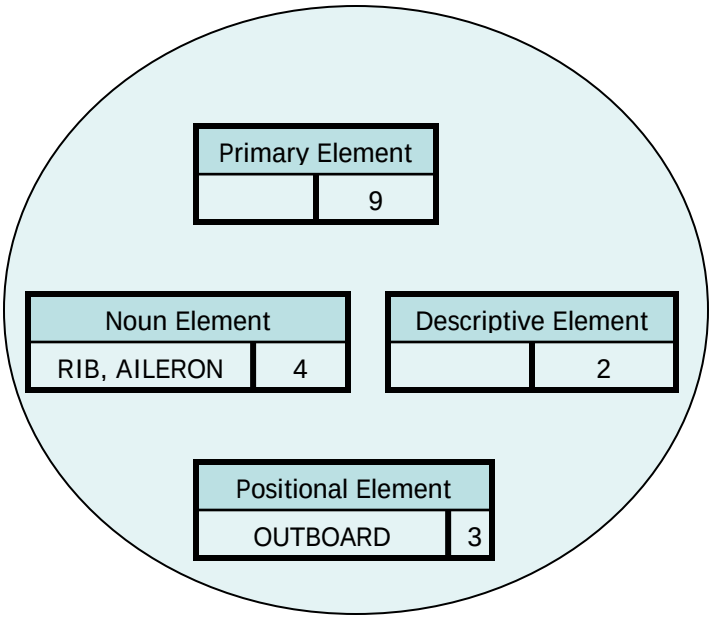


Figure 25. Actual Score Computation of associated System record



Candidate Actual Score = (0*9)+(2*4)+(0*2)+(1*3)
=0+8+0+3=**11**

Figure 26. Final Candidate Actual Score with the integrated Enhancement Data

The resultant scores from our second example are computed above. The Object Translation Score of the record RIB (i.e. 15) is closer to zero than that of the Object Translation Score of the record SPOILER AND SPEED BRAKE ASSY (i.e. 41). As a result the candidate record RIB is more likely to be the mapping to the target record LEFTHAND OUTBOARD AILERON RIB UPPER CHORD than candidate record SPOILER AND SPEED BRAKE ASSY.

4.5 Sum-Ordering Feature Weighting

In this section we introduce a technique denoted as sum-ordering feature weighting using a percentage value approach. It describes the computation of the weight values used in computing the Object values discussed earlier in section 4.4. Let $E_1E_2E_3E_4$ be four distinct features (e.g. Positional, Descriptive, Noun, and Primary). The researcher determines that the Primary feature should outweigh all others because it is the main term of importance in a record description. Based on an intimate knowledge of the data population, the researcher is able to determine constraints between the features. As an example we have identified that the following constraints should exist between the features. In specific, $E_{1.weight} < E_{2.weight} <$

$E_{3.weight} < E_{4.weight}.$, where $E_{1.weight}$ is the weight value assigned to the feature *E1*. The Primary feature would correspond to E_4. The main objective of this sum-ordering feature weighting process is to identify weight values that do not disrupt the defined feature constraints. When analyzing a group of potential candidate records, it is important to have an established rank among the features. The predefined feature ranking helps to establish implicit rules which address how the algorithm is to determine which candidate record is most significant given any combination of feature values. For example let E_3 correspond to the feature denoted as Noun and E_4 correspond to the feature denoted as Primary. Let there be a target record *door knob*, where data element *door* is associated with the feature: Noun and *knob* is associated with the feature: Primary. If two candidate records are encountered one being *door* and the other being *knob*, which record should be considered the most dominant? By observation, a human could determine that the record *knob* is the most dominant, but an algorithm must have some defined way of making the same observation. In analyzing the record *door*, the One-Up rule (see definition 4.14 for explanation) must be used, thus the data element door is categorized as a Noun feature term (i.e. E_3). In analyzing the record *knob*, the element knob is categorized as a Primary feature term (i.e. E_4). Using the predefined feature ranking order where $E_3 < E_4$, an algorithm has a quantitative means in determining the most dominant record, the record *knob*.

In Table XIII an example is given where $E_{1.weight}=1$, $E_{2.weight}=2$, $E_{3.weight}=3$ and $E_{4.weight}=4$. The Phrase column indicates all the possible combinations of the features. Each combination of features will yield a value when the associated weights are summed. Each combination is summed and the values are ordered in descending order.

Table XIII. Effects of incorrect feature weighting

$E_{1=1}$ $E_{2=2}$ $E_{3=3}$ $E_{4=4}$

Index	*Phrase*	*Value*
1	$E_1E_2E_3E_4$	*1+2+3+4 = 10*
2	$E_2E_3E_4$	*2+3+4 = 9*
3	$E_1E_3E_4$	*1+3+4 = 8*
4	E_3E_4	*3+4 = 7*
5	$E_1E_2E_4$	*1+2+4 = 7*
6	E_2E_4	*2+4 = 6*
7	E_1E_4	*1+4 = 5*
8	E_4	*4 = 4*
9	$E_1E_2E_3$	*1+2+3 = 6*
10	E_2E_3	*2+3 = 5*
11	E_1E_3	*1+3 = 4*
12	E_3	*3 = 3*
13	E_1E_2	*1+2 = 3*
14	E_2	*2 = 2*
15	E_1	*1 = 1*

From the constraints previously applied, the expected order of the feature combinations from highest priority to lowest priority is already known. See the Phrase Column 2 in table XIV for the correct ordering. The task in weight creation is to make sure that the relationships between the summed weight values maintain the predefined constraints. The Value column depicts the phrase combination values in descending order. In table XIII the phrase combinations in indices 4 and 5 and 12 and 13 reveal that equivalent values are present. This does not follow our constraints. The Phrase combination in indices 8 and 9 also reveal a phrase pair that does not follow the constraint rule as well. Let B be the set of all combinations of N items taken 1, 2, 3,…,N items at a time. The total number of different combination items can be determined by:

$$\text{Total number of possible combinations of N items} = \sum_{P=1}^{N} \left(\frac{N!}{P!(N-P)!} \right) = 2^N$$

For example in Column 1 of table XIV listed are all the combinations of 3 items taken 1, 2 and 3 items at a time. Given the defined relationship among the items (i.e. $E_{1.weight} < E_{2.weight} < E_{3.weight}$) the items can be arranged such that the values of the combination will be in descending order. See arrangement in Column 2 of table XIV. Using Column 2, the feature combination at index 1 should be greater than the combination at index 2, the combination at index 2 should be greater that the combination at index 3 and so forth.

Table XIV. Incorrect versus Correct feature ordering

Index	*Column 1*	*Column 2*
1	E_2E_3	$E_1E_2E_3$
2	$E_1E_2E_3$	E_2E_3
3	E_1E_2	E_1E_3
4	E_1E_3	E_1E_2
5	E_1	E_3
6	E_3	E_2
7	E_2	E_1

Definition 4.19: The *Sum Relationship Rule (SRR)* states that given a collection of N items $E_1E_2E_3...E_N$, with the strict constraints of $E_{1.weight} < E_{2.weight} < E_{3.weight} <....< E_{N.weight}$, where $E_{i.weight}$ is a weight value associated with item E_i, there is a set X consisting of all combinations of the N items, where all elements of X can be arranged, summed and sorted in descending order.

To satisfy the Sum Relationship Rule we introduce the Percentage Grid. See Table XV. We first let each $E_i.weight$ be represented by some constant C^N. At this point N is unknown. Selecting the appropriate constant C is done by first determining what percentage of the Record Object is to be represented by the Primary feature.

Table XV. Sample Percentage Grid

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	1	2	4	8	16
Row 2	1	3	9	27	81
Row 3	1	4	16	64	256
Row 4	1	5	25	125	625
Row 5	1	6	36	216	1296
Row 6	1	7	49	343	2801

For clarity we denote the Primary feature as P. We denote the remaining features as the set P`. In our case P`={Noun, Positional, and Descriptive}. The remaining features (e.g. Noun, Positional, and Descriptive) together will make up the residual percentage. As an example if one selects the Primary feature to be valued at approximately 67 percent of any given Record Object; the remaining features together would be worth the remaining 33 percent of the Record Object. See figure 27. The weight value assignment is done

by using the values in table XV. Using table XV, row 1 represents the constant C having the value 2. In table XV, each value in a given row denotes the value of C^N where N takes on the values 0 to 4.

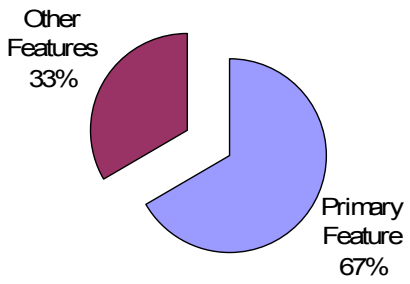


Figure 27. Record Object Percentage

For this example N ranges from 0 to 4, but practically N can be set to any integer value. Row 2 represents the constant C having the value 3. Row 3 represents the constant C having the value 4 and rows 4 and 5, the values 5 and 6 respectively. Table XV serves as a table look-up for percentage values as well as specific weight values. Next the values in column 1 are used as a numerator and the values of column 2 as a denominator to compute percentage values. For instance to assign the Primary feature the approximate percentage of 67 and the other features P` the approximate percentage of 33, the constant C would take the value 3. This is determined by Row 2 where column 1 has the numerator value 1 and column two has the denominator value of 3 which yields 1/3 or 33%. If the preferred value for the P` is to be 25% (i.e. ¼) and the value of the Primary feature is to be 75% then the constant C would take the value of 4. This strategy can be used when the total number of features constructed is greater than two. In table XVI, we present the feature weight values using the constant C. The weight values are E_1.weight=1, E_2.weight=2, E_3.weight=4 and E_4.weight=8. It is seen that the values satisfy the Sum Relationship Rule.

Table XVI. Sum Relationship Rule satisfied with valid weight values (C=2)

E_1=1 E_2=2 E_3=4 E_4=8

Index	*Phrase*	*Value*
1	$E_1E_2E_3E_4$	*1+2+4+8 =* ***15***
2	$E_2E_3E_4$	*2+4+8 =* ***14***
3	$E_1E_3E_4$	*1+4+8 =* ***13***
4	E_3E_4	*4+8 =* ***12***
5	$E_1E_2E_4$	*1+2+8 =* ***11***
6	E_2E_4	*2+8 =* ***10***
7	E_1E_4	*1+8 =* ***9***
8	E_4	*8 =* ***8***
9	$E_1E_2E_3$	*1+2+4 =* ***7***
10	E_2E_3	*2+4 =* ***6***
11	E_1E_3	*1+4 =* ***5***
12	E_3	*4 =* ***4***
13	E_1E_2	*1+2 =* ***3***
14	E_2	*2 =* ***2***
15	E_1	*1 =* ***1***

We present another example; let us assume a total of four features have been constructed and the data analyst wants to assign the Primary feature approximately 67% of the Record Object and P` the remaining 33%. To obtain 33% for the P` values, row 2 is used because 1 over 3 is approximately 33%. Using table XV it is seen that the features in P` are assigned the values 1, 3, and 9. The Primary feature is assigned the value of 27.

To ensure that percentage values are accurate we validate them by a simple computation. First we sum the four values, 1 + 3 + 9 + 27 = 40. Our total Record Object is valued at 40. The percentage of the RO that is occupied by the Primary feature is 27/40 = .675 or approximately 67%. The remaining features occupy 13/40 = .325 or approximately 33% of the RO. Table XVII shows the feature weight values when the constant C=3.

Table XVII. Sum Relationship Rule satisfied with valid weight values (C=3)

E_1=1 E_2=3 E_3=9 E_4=27

Index	*Phrase*	*Value*
1	$E_1E_2E_3E_4$	*1+3+9+27 =* ***40***
2	$E_2E_3E_4$	*3+9+27 =* ***39***
3	$E_1E_3E_4$	*1+9+27 =* ***37***
4	E_3E_4	*9+27 =* ***36***
5	$E_1E_2E_4$	*1+3+27 =* ***31***
6	E_2E_4	*3+27 =* ***30***
7	E_1E_4	*1+27 =* ***28***
8	E_4	*27 =* ***27***
9	$E_1E_2E_3$	*1+3+9 =* ***13***
10	E_2E_3	*3+9 =* ***12***
11	E_1E_3	*1+9 =* ***10***
12	E_3	*9 =* ***9***
13	E_1E_2	*1+3 =* ***4***
14	E_2	*3 =* ***3***
15	E_1	*1 =* ***1***

Formula 4.5 is used to compute the percentage value for P` and formula 4.6 is used to compute the percentage value for the Primary feature.

$$\frac{\sum_{i=0}^{N-1} C^i}{\sum_{i=0}^{N} C^i} \qquad \textbf{(4.5)}$$

$$\frac{C^N}{\sum_{i=0}^{N} C^i} \qquad \textbf{(4.6)}$$

In table XIII, there exist three instances that did not satisfy the Sum Relationship Rule, indices 4 and 5, 8 and 9, and 12 and 13. All three share a common characteristic.

Case 1: In indices 12 and 13, index 12, the weight value of E_3 should be greater than the sum of the weight values of E_2 and E_1. This suggests that C^n should be greater than $(C^{n-1}+C^{n-2})$.

Case 2: In indices 8 and 9, index 8, the weight value of E_4 should be greater that the sum of the weight values of E_3, E_2 and E_1. This suggests that C^n should be greater than $C^{n-1} + C^{n-2} + C^{n-3}$.

Case 3: In indices 4 and 5, index 4, the weight value of the sum of E_3 and E_4 should be greater that the sum of the weight values of E_4, E_2 and E_1. Since E_4 is in both indices, we can disregard that value so we end up with case 1 again. E_3 should be greater that the sum of the weights of E_2 and E_1.

All the above cases suggest that C^n should be greater than $C^{n-1} + C^{n-2} + \ldots + C^0$. We can prove that the following is true, $C^n > C^{n-1} + C^{n-2} + \ldots + C^0$ when $C \geq 2$. We begin first by defining the formula for Geometric Progression as: $C^0 + C^1 + C^2 + \ldots + C^{n-1} = \dfrac{C^n - 1}{C - 1}$.

Proof: If $C \geq 2$, *then*

$$C + \frac{1}{C^n} > 2$$

$$\Rightarrow C^{n+1} + 1 > 2C^n$$

$$\Rightarrow C^{n+1} - C^n > C^n - 1$$

$$\Rightarrow C^n (C - 1) > C^n - 1$$

$$\Rightarrow C^n > \frac{C^n - 1}{C - 1}$$

$$\Rightarrow C^n > C^{n-1} + C^{n-2} + \ldots + C^0$$

Proposition 1.0: Given N items (E_1, E_2, E_3,…,E_N) having the relationship $E_1 < E_2 < E_3 < \ldots < E_N$, weight values $C^0, C^1, C^2, \ldots, C^{N-1}$ (where C is an integer greater than 1) can be assigned to each item respectively such that the Sum Relationship Rule is satisfied.

4.6 Feature-Based Data Rule Generation

Data Rules are guidelines or condition statements, based on detailed or unique properties of data within a specific Data Space which help to govern the way an entity (e.g. humans or computer programs) can best process other data found in similar Data Spaces. Data rules are generally captured as phrases similar to IF THEN condition statements and then transformed into statements in which the processing algorithm can understand. A sample rule could be: "If the item description consists of the term 'screw' then add the item to set Y". Once characteristics and properties among the data within the Data Field are identified, developing data rules are made less difficult. In specific, in this work a record description having each data element properly associated with a feature is the most basic form of a data rule. As an example, given the record "*front door knob screw*", the term *front* is classified in the feature category denoted as Positional, the terms *door* and *knob* are classified in the feature category denoted as Noun, and the term *screw* is classified in the feature category denoted as Primary. In this example the feature category Description is left without a value. The rule in figure 28, suggests that each collection of feature values be multiplied by its associated weight value to obtain the corresponding scores.

[(**Positional:** front) (**Noun:** door AND knob) (**Primary:** screw)]

Figure 28. Example feature based data rule

Data rules are dynamically constructed as new information is learned about terms. For example due to learning from previously processed records the system identifies the term *forward* as an alternate word for *front* and the term *handle* is determined to be an alternate word for *knob*, the data rule in figure 29 would be generated.

[((**Positional:** front OR forward)) ((**Noun:** door AND (knob OR handle))) (**Primary:** screw)]

Figure 29. Example feature based data rule using learned information

The rule in figure 29 suggests that either the term *forward* or *front* can be used as a Positional term. The term *door* along with the term *knob* or *handle* can be used for the Noun term during the score computations. In this sense each data rule generated is different as it is dependant on two elements. First it is dependant on the record description and second it is dependant on the amount of knowledge learned to the present time of a record processing step. In specific the data rule generated for a record A at the beginning of a linking session at time γ will be different from a data rule generated for the same record A if processed towards the end of a linking session at time $\gamma`$. The difference is marked by the amount of knowledge obtained during the time $\gamma` - \gamma$.

4.6.1 Parts-Breakdown Blocking

In this section we present a Parts-Breakdown blocking procedure.

Definition 4.20 *Blocking* is the process of dividing the records in the Data Field into individual blocks of records as to reduce the number of actual record pair comparisons.

The primary goal of blocking is to remove candidate records that obviously should not be considered as a mapping during the data processing step [2]. As an example if a target record description describes a component of an aircraft *wing*, then candidate records describing components of the aircraft's *nose landing gear* should not be considered. An advantage of blocking is that it can decrease the amount of time to process records as a result of only observing "relevant" records. This leads to the decrease in the time it takes the linkage algorithm to make data mappings. To make the record blocks, the blocking procedure is performed using a set of record attributes or features common to records in the Data Field. As an example records could be separated and compared based on whether or not they have the same *city and state* value. We analyze the Domain Object in question; in this case an airplane. We identify k different regions of the aircraft. These k regions are identified by the researcher as main areas of the aircraft which are more likely to consist of parts that are restricted to the specific aircraft region. For example the nose of an aircraft does not consist of wings or wing-specific parts, so the nose and wing areas of the aircraft could be valid regions

used during blocking. In figure 30 a tree-like structure denoted as a Parts-Breakdown Tree (PBT) is shown. The leaf nodes (i.e. bottom nodes) consist of a set of strings which correspond to the k main areas/partitions of the Domain Object (i.e. aircraft). We denote this set as $M=\{m_1,m_2,m_3\ldots, m_k\}$ where m_i is a label corresponding to a partition and k is an integer value. A set M' represents a miscellaneous or unknown region. It denotes the other areas of the Domain Object not captured by M. The data rule describing the items in M' is denoted as $\neg (m_1 \wedge m_2 \wedge m_3 \wedge \ldots \wedge m_k)$. Using figure 30 M={*wing, engine, fuselage, tail, landing gear, nose*}. The branch nodes (internal nodes) in the tree diagram consist of a set of indicators (strings) used to reference a unique item in M. We denote this set as $S=\{s_1,s_2,s_3,\ldots, s_k\}$ where s_i is a set of strings that reference an element m_j in the set M where i=j. For example in figure 30 it is seen that the indicators *rudder, elevator, stabilizer, fin* were determined to refer to the Tail section of the aircraft. A set S' represents the set of indicators that are not captured in S. The data rule describing the items in S' is denoted as $\neg (s_1 \wedge s_2 \wedge s_3 \wedge \ldots \wedge s_k)$. It is noted that the elements in M and S exist in the set denoted by the union of the Noun features and Primary features. The blocking process used is initiated by selecting and categorizing records based on the k partitions identified in M. Record descriptions containing any of the terms found in label m_i or the indicator s_i corresponding to m_i are placed in a separate data block. The indicator values should be relatively specific to a main area in M otherwise some records could be neglected as a result of being in the wrong data block. For instance if there existed two blocks of data, block A and block B, and as a result of a weak blocking procedure, block A contains records that belong in block B. In the event that a target record requires comparisons of block B, based on indicators, then the records in block A belonging to block B will never be processed.

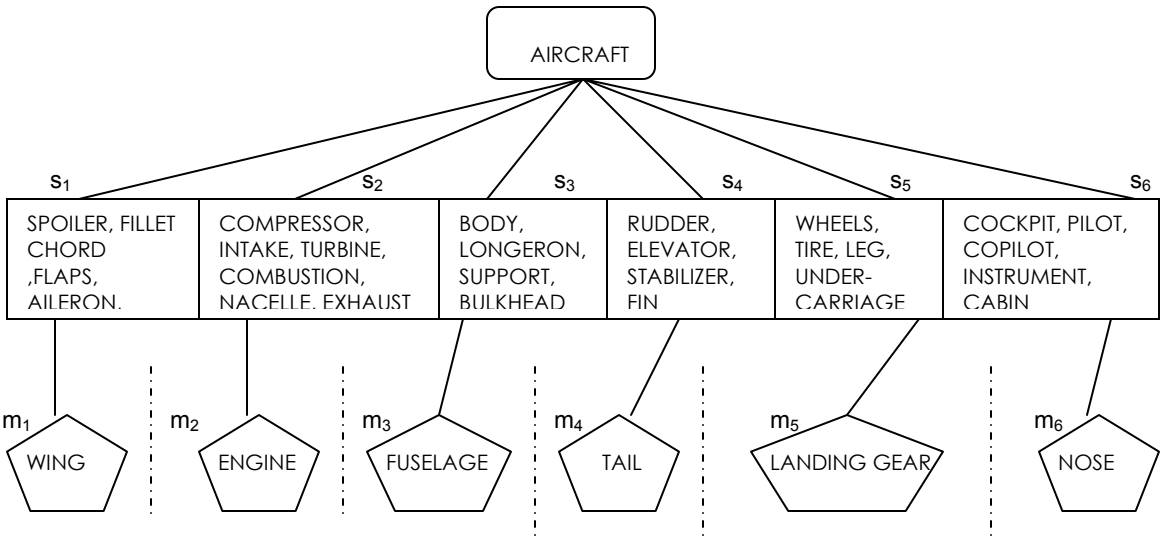


Figure 30. Parts-breakdown Tree Structure

Once the blocking process has been completed, the record comparisons are initiated. The feature values from a target record are processed to determine which data block the record comparisons should begin. As an example, given a record *Inboard Aileron Trailing Edge*, based on the Noun feature values *Aileron* and *Edge* it is determined that the candidate records should be obtained from the block associated with *Wing* area. All records within the block denoted by *Wing* that include the terms *Inboard, Aileron, Trailing, Edge* or known synonyms or alias terms will be retrieved and processed to obtain a quality of match score. This step identifies the data block as well as a list of potential candidate records within the data block. If by chance a candidate record *Elevator Trailing Rib* is placed in the *Wing* data block, the record will quickly be eliminated from the list of possible candidates because the PBT is also used in a filtering process.

Definition 4.21 *Filtering* is the process of using predefined rules to eliminate or limit the acquisition of unwanted information.

Filtering helps to refine the accuracy of the data processing step. Filtering is done by checking all relevant feature values of a candidate record to ensure that indicator values associated with other data blocks are not present. Using the target record *Inboard Aileron Trailing Edge*, belonging to the *Wing* section, and the

candidate record *Elevator Trailing Rib*, belonging to the *Tail* section, it is quickly determined that the candidate record should not be considered as a possible mapping because of the indicator value *Elevator*.

4.7 Learning and Alternate Term Discovery

In this work, a subject matter expert (domain expert) will validate the record links obtained in the linking process. Many times a domain expert will be able to say with some degree of certainty if a record link is accurate or not. The validated information will become historical data for use during future record linkages. Stored data can be meaningless if not used appropriately. In this section we present a means of capturing, recording, and utilizing the validated information from the record pairs to give the mapping algorithm additional information (i.e. dynamic check points) to be used in an enhancement step during the linking process. We modify the 3-Dimensional Torus Data Structure (3DTDS) presented in [15], to develop a data structure which implicitly captures conditional probabilities. A conditional probability is the probability of an event A given the occurrence of another event B. The 3DTDS is introduced in [15] as a new structure possessing characteristics similar to that of a three-dimensional torus network topology. It utilizes a special hash function to insert and retrieve elements. The hash function used is similar to the function used in locating computing nodes within a typical torus network topology. The data structure can be created using matrices, binary trees, or in a matrix-tree hybrid. Quality results have been seen when sorting and searching large data sets. Review [15] for further explanation of the 3DTDS. The structure presented in this research is denoted as Term Probability Structure. Formally the Term Probability structure Z is a five-tuple (F,Q,M,S,G,D),

F is a set which consists of values associated with relevant features (i.e. NOUN ∪ PRIMARY);
Q is a set of row locations q_i, where i=1,N;
M is a mapping on F, F**x**F$\rightarrow q_i$;
S is a mapping Q**x**F$\rightarrow$ I, where I is the set of integers;
G is a set of elements $\{g_1,g_2,\ldots g_N\}$ of type S;
D is a set of distance values $\{d_1,d_2,\ldots,d_N\}$ corresponding to a specific element g_i in G where $d_i \rightarrow g_j$, i=j.

We briefly present some benefits of the TPS then give examples of its function. In section 4.3 we introduced a methodology for constructing features based on the syntax of records in the Data Field by using a context free grammar. In contrast in this section we incorporate the concept of "context" to assist

in recording the frequency of word to word mappings. We want to know the relationship between three terms. In specific we want to know the probability of a term A mapping to a term B given that a term C appeared adjacent to term A in a record description. If we only record the number of times a term A mapped to a term B, this information is helpful but we want to know to what extent or what other parameters work together to make this a valid case. During the data processing step in the event that three candidate records have identical Object Translation scores, a decision must be made to determine which record should be selected. Based on a record of historical mappings and related parameters, such values can be used to assist the mapping algorithm in making optimal decisions when there is a need to make a determination between multiple records or in determining if a certain record should be selected over another. As result of mapping ambiguity, capturing the context is vital to this research (see section 2 for explanation of mapping ambiguity) as there are several records with identical or nearly identical Primary term values. For instance the target records *Inboard Aileron Rib* and *Horizontal Stabilizer Rib* have the same Primary term, *Rib*. Although these records have identical Primary terms, the candidate records that each one of the target records map to may not consist of the Primary term *Rib*. As an example let us use the record *Inboard Aileron Rib.* The purpose of the data structure is to capture the term that *Rib* maps to given that the term *Aileron* is adjacent to *Rib* in the description. The level of adjacency is measured by a distance value. As an example, the terms *Aileron* and *Rib* are considered to be at a distance of one unit apart. Using the record *Horizontal Stabilizer Rib*, the purpose of the structure is to capture the term that *Rib* maps to given that the term *Stabilizer* is adjacent to *Rib* in the description. The terms *Stabilizer* and *Rib* are considered to be at a distance of one unit apart. It is noted that all distance values being measured denote the values in set D in the Term Probability Structure. Each distance value is associated with an element g_i in the set G. In the previous examples the element g_1 would consist of all values associated with the measured terms that have an adjacency distance of one, denoted by d_1. The structure can be used to capture relationships at varying distances. For example to capture the relation at a distance of two units, let us use the record *Horizontal Stabilizer Rib.* We would then use the structure (e.g. denoted as $g_{2)}$ to capture the term that *Rib* maps to given that the term *Horizontal* is at a distance of two (i.e. d_2) from *Rib* in the description. The adjacency distance being measured is determined by the data analyst. Recording the frequency of such mappings serves as a knowledge base of historical data vital to assisting the data

mapping algorithm in making optimal mappings. The structure utilized implicitly captures conditional probabilities. This is extremely beneficial because it provides quantitative justification for making mapping decisions. Let X denote the probability that a term A will map to a term B and let Y be the probability that a term C occurs at a distance of N terms away from A, where N is an integer. As an example given the record *Door Knob Screw,* the terms *Knob* and *Screw* share a distance of one, thus N equals one. In using the structure to capture a set of previously processed mappings, the conditional probability P(X|Y) (based on historical data) can be obtained and used to make an optimal mapping decision in the future. As more data is processed the probability values associated with a specific term will become dominant.

In figure 31a the FxF mapping M is depicted. It is noted that the data researcher must select the features and associated feature values that will be represented by F. In this research, the Noun and Primary features were selected. The elements A, B, C, and D are representative of terms in both the set of Primary and Noun elements which are taken from the disjoint set in the Data Field. The value of location M[i][i] is equal to zero as it is assumed that a term will not be in the same record description. As an example in figure 31a the values at M[0][0], M[1][1], M[2][2] and M[3][3] all have the value zero. In figure 31a, the row denoted by M[1] corresponds to term A. Thus,

M[1][1] can be understood as an event when A is adjacent to A in a record;
M[1][2] can be understood as an event when B is adjacent to A in a record;
M[1][3] can be understood as an event when C is adjacent to A in a record;
M[1][4] can be understood as an event when D is adjacent to A in a record.

Noun Element

Primary Element	A	B	C	D
A	0	1	2	3
B	4	0	5	6
C	7	8	0	9
D	10	11	12	0

Figure 31a. Example component **M**

Term Mapping

Rows	A	B	C	D
1	1	4	2	3
2	4	1	2	0
3	1	2	1	0
4	3	8	3	1

d_1: Distance = 1 g_1 g_2 g_3

Figure 31b. Example components **S**

As illustrated in figure 31a M[i][j] will yield an integer value denoting a row location in S (i.e. QxF). We note there can be several elements S each of which is associated with a specific word distance between adjacent words. S where word distance is one is pictured as the first matrix in figure 31b. This matrix is denoted as g_1. The data analyst could define another element S in the set G to capture information associated with an adjacency word distance of 2. Pictured in gray is g_2 and g_3 where the distance matrix is equal to two and three respectively. Using figures 31a M[1][3] (i.e. event when term C is adjacent to term A in a record), the value references row location two in matrix g_1 (word adjacency occurs at a distance of 1). The integer values in matrix g_i indicate the number of times a Primary Element in M mapped to a specific Noun Element, where the set of Noun Elements compose the column values of g_i. See figure 31b for an illustration. As an example, from the previously processed data, it can be deduced that the probability that term A maps to A given that it appears adjacent to term C in a record at a distance of one (i.e. P(X|Y)) is

$$P(X|Y) = \frac{\text{\# of outcomes in X and Y}}{\text{\# of outcomes in Y}} = \frac{P(X \cap Y)}{P(Y)} = \frac{4/21}{7/21} = 4/7 = .57 = 57\%$$

where,

Event X = {Term A mapped to term A}.

Event Y = {Term C was adjacent to A}.

In essence this mechanism once applied will assist the linking algorithm to, 1) filter out records consisting of less dominate terms and 2) select records consisting of records deemed more dominate. This tool will allow for discovery of synonyms and/or alternate terms. Data from this mechanism will be used in the enhancement step as dynamic checkpoints. As word synonyms and alternate terms become available, they can be utilized to strengthen the pseudo-intersections between disjoint sets. Figure 32 illustrates a magnified view of the pseudo-intersection between the initial two disjoint sets.

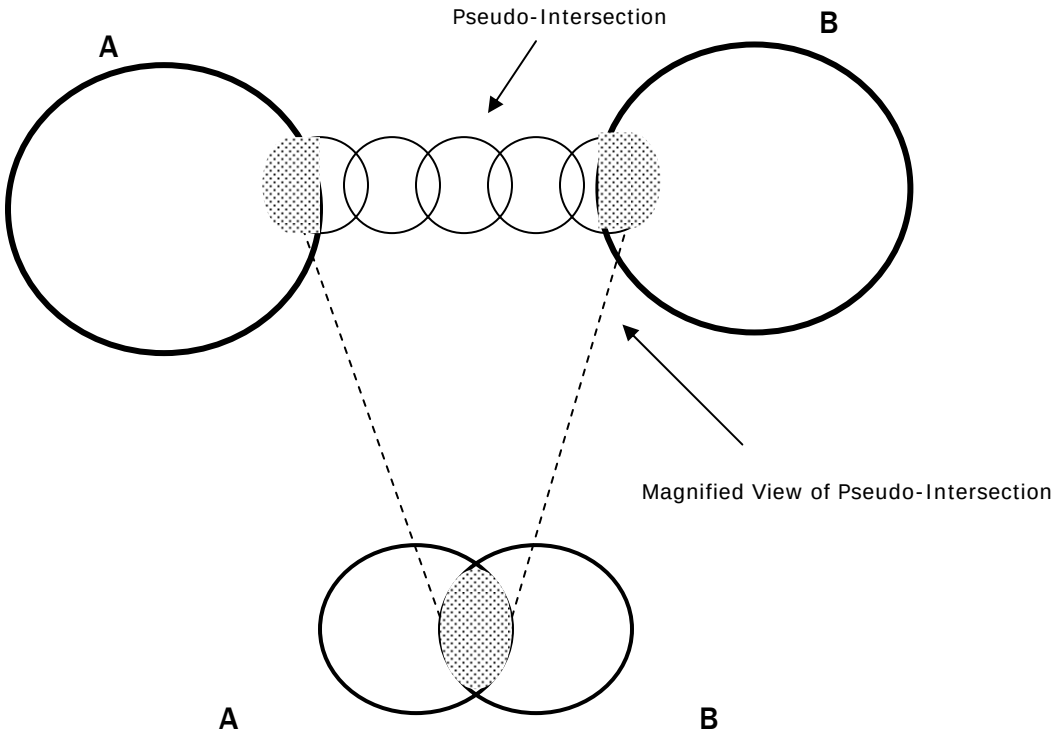


Figure 32. Items in initial disjoint sets mapped using a pseudo-intersection

4.8 MUDD Classification Algorithm

The MUDD Classification Algorithm is the result of the integration of the approaches described in the previous sections. A pseudo-algorithm is listed in figure 33.

WHILE LOOP 1: i=1,X (X=number of set A records to evaluate)
Step 1: Get set A Record
Step 2: Convert Record to *Record Object*
Step 3: Perform Enhancement step
Step 4: Compute Target Actual Value
Step 5: Feature Value Ordering
Step 6: Get List of potential mappings
Step 7: Compute Recommended Discovery Item (e.g. Equipment Code)
WHILE LOOP 2: j=1,Y (Y=number of Equipment codes to process)
WHILE LOOP 3: k=1,Z (Z=number of set B records to process)
IF (set B record_EquipmentID == EquipmentCode[j])
Step 8: Convert set B_Record[k] to Candidate Object (CO)
Step 9: Apply Rules
Step 10: Perform Candidate Object Enhancement Step
Step 11: Compute Object Translation Value
Step 12: Insert CO into **MAPPING_LIST** array
END IF
END WHILE LOOP 3
END WHILE LOOP 2
Step 13: Perform Minimum Sort on **MAPPING_LIST** array
Step 14: Output Top M records (M=5, user defined)
Step 15: Perform Learning Step*****
Step 16: Distance Tree*****
END WHILE LOOP 1

Figure 33. MUDD Classification Algorithm

CHAPTER FIVE

RESULTS

5.1 Grammar-based Featured Construction

In our research, a subset of the records in the Data Field were processed to generate a grammar in an attempt to discover any knowledge about the Data Field. It was determined that in order to better observe the data, production rules could be analyzed versus attempting to analyze the entire Data Field manually record by record. A grammar captures any patterns or relationships within the records. These patterns and relationships can then be used to assist a data analyst to construct features. Based on several thousand records processed, the following sentences were obtained,

```
ananndnnnng
anannnng
nannnnn
n
annnnnng
annn
an
aannn
nannn
anndanann
nndcdanann
aanng
aann
ananng
nnnn
andnng
ngnnn
anndng
anndnng
annnng
nnnnnnn
annnn
aaanan
anndnn
annnnng
aaannng
nnn
aanannn
```

These sentences make up the language P. In the sentences, the letter ‘a’ denotes an occurrence of an adjective and the letter ‘d’ denotes the occurrence of a digit. The letter ‘n’ denotes a noun and the letter ‘g’ denotes a type of noun that encompasses multiple parts/items. For example the term ‘system’ is a noun but the term itself suggests the possible inclusion of multiple components. In contrast, the term ‘wing’ is a noun that alone, does not suggest the inclusion multiple parts. The grammar generated by our algorithm for language P can be found in figure 34.

```
START → S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | S13 | S14 | S15 |
S16 | S17 | S18 | S19 | S20 | S21 | S22 | S23 | S24 | S25 | S26 | S27 | S28 | S29 | S30

S1 → 82 5 5 0
82 → nndcd
5 → an
0 → n
S2 →  5 11 5 5 0
11 → nd
S3 →  5 5 49
49 → nnng
S4 →  1 S27 6 2
1 → a
6 → nn
2 → g
S5 →  23 S27 2
23 → ana
S6 →  1 5 S29
S7 →  7 5 20
7 → aa
20 → nng
S8 →  0 5 37
37 → nnnn
S9 →  1 S27 9
9 → ng
S10 →  S27 S30
S11 →  1 6 3 6 2
3 → d
S12 →  8 S27 0
8 → na
S13 →  7 5 5
S14 →  1 6 3 6
S15 →  1 S27 2
S16 →  5 5 9
S17 →  1 6 28
28 → dng
S18 →  24 6 2
24 → and
S19 →  23 6 2
S20 →  5 16
16 → nnn
S21 →  1 5 9
S22 →  9 S30
S23 →  7 6 2
S24 →  0 S29
S25 →  1 S29
S26 →  1 5 0
S27 →  6 6
S28 →  1 S30
S29 →  5 6
S30 →  6 0
```

Figure 34. Generated CFG

The grammar generated helps us to isolate patterns within the population of data in order to make an informed analysis of the Data Field. Some of the characteristics analyzed are as follows,

- frequency of specific patterns and sub-patterns
- composition of patterns
- relationship of terms within patterns (terms beginning or end)

We found that each record had to have at least one noun as 'n' was the most common pattern. Each record ended with a noun denoted by either letter 'g' or 'n'. The total patterns ending in 'n' were greater than the number of patterns ending in 'g'. The pattern consisting of 'nn' was the second most common pattern found throughout the grammar. The pattern 'nn' suggests that throughout the Data Field nouns are used to modify other nouns; they often function like adjectives. The third most common pattern found was 'an' which suggests that in the records, adjectives are used to describe nouns although not as often a nouns are used. We were able to construct new features based on these observations. From these observations as well as an analysis/awareness of how different adjectives were used throughout the Data Field, new features were constructed. The features denoted as Primary, Noun, Positional, and Descriptive were constructed. Each feature is given a weighted value. The Primary feature indicates a set of noun values and was constructed based on the observation that all records must end with a noun term. This term is considered the Primary Term of Observation (PTB). The Noun feature indicates an additional set of noun values. This feature was constructed on the basis that a larger number of nouns were used throughout the Data Field to add meaning to the PTB and/or other nouns within a record description. The Positional and Descriptive features both indicate a set of adjectives. Based on an analysis of some of the adjectives, the set of adjectives we divided into these two categories. It was found that some adjectives such as "outboard" and "left-hand", described the *location* or *position* of items on an aircraft whereas other terms like "hexagonal" and "diagonal" described specific characteristics of items. Based on the observation of frequency within patterns as described above, weight values were assigned.

5.2 Sum-Ordering Weighting

In this research we selected the Primary feature to represent approximately 50% of the RO and the other features to represent the other 50%. The values in row 1 were used as weights for the features (i.e. values

1, 2, 4, 8). The percentage value of 50% was selected because we wanted the weight of a single Primary term to be as important as the appearance of a candidate object that did not have a Primary term but only had a Noun term, Positional term, and Descriptive term. It was observed that if the weights are not allocated appropriately, a number of false positives will be obtained throughout the data processing step. We demonstrate a scenario of the acquisition of a false positive below. Assume we have a target record object with the name *Outboard Trailing Edge Rib* and two candidate record objects (CRO), one with the name *Inboard Rib* (CRO_1) and the second *Outboard Trailing Edge (CRO_2)*. Let P` be valued at 25% (i.e. row 3) and the Primary feature be valued at 75%. Using row 3 in the Percentage Grid, this suggests that the features have the following values, Primary feature: 64, Noun feature: 16, Positional feature: 4 and the Descriptive feature: 1. Taking the terms in all the record objects and categorizing them into there appropriate feature we obtain the following list:

Primary: *Rib*
Noun: *Edge*
Positional: *Outboard, Inboard*
Descriptive: *Trailing*

Now we proceed with the steps to compute the Object Translation Scores. The actual value for the Target Record Object having the name *Outboard Trailing Edge Rib* is 85 (i.e. 64+16+4+1). Candidate Record Object 1 (CRO_1) has an actual value of 68 (i.e. 64+4) because it has one primary term, *Rib*, and one Positional term, *Inboard*. Candidate Record Object 2 (CRO_2) has an actual value of 21 (i.e. 16+4+1). We note the One-up Rule was used. (*Definition 4.14*). The Object Translation score (OTS) for CRO_1 is 25 and 64 for CRO_2. These OTS's suggest that CRO_1 is more of a match than CRO_2, where although CRO_1 suggests that it is a Rib structure; however one of its description terms *Inboard* communicates that the particular structure *Inboard Rib* is in a different location on the aircraft than that of the *Outboard Trailing Edge Rib*. As a result the algorithm would have identified the wrong type of Rib as the correct mapping. CRO_2 should have rendered the better quality of match score because although the record does not communicate it is specifically a Rib structure, it does reference the appropriate area and position of the aircraft.

5.3 Algorithm Results

Our experiments were conducted using a PC with a processor speed of 1.73 GHz and 2.0 GB of RAM. The data was managed by the Oracle 9i Database Management System on a remote server. See figure 35. All programming was done in Java programming language because of its object oriented nature, portability and built-in functionality.

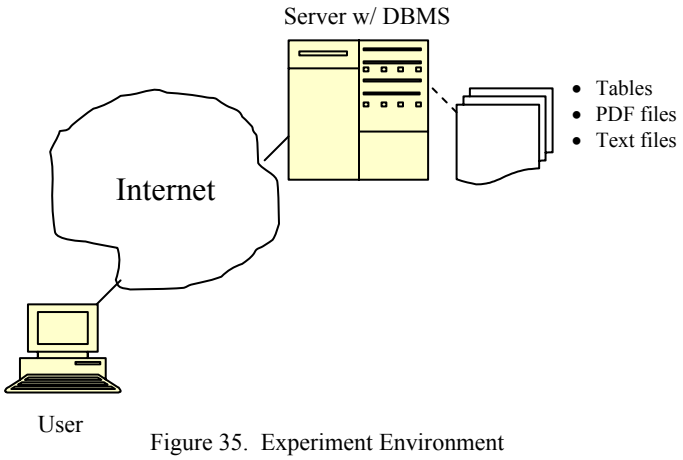


Figure 35. Experiment Environment

A number of tables were used to help prepare the data before it was processed by the data mapping algorithm. The tables of major interest consisted of a General Dictionary, Knowledge Base, Figures, Section Headers. A brief description of each is listed below.

- General Dictionary - consists of words and major parts of speech.
- Knowledge Base – consists of words and their known acronym, abbreviation and alias.
- Figures – consists of figure ids and figure descriptions
- Section Headers – section id and section descriptions

The tables holding data from set A and B were similar in make-up. Sample columns include part description, part number and a column corresponding to each feature (e.g. Primary, Positional, Description). See table XVIII for a view of a partial table.

Table XVIII. Partial Table

Part Id	Part No.	Figure Id	Part Description	Primary (feature 1)	Noun (feature 2)	Positional (feature 3)	Descriptive (feature 4)	Number (feature 5)

There were minor variations in the tables. The table containing the items from set A consisted of an additional field that corresponded to a figure item in the Figures table. The records in set B did not have associated figure data. There was part system information corresponding to the records in set B available. An additional field was created to capture this information in the table holding the set B records. System information for the items in set A was not available. All data is made available from several PDF documents. The PDF files were converted to text files and cleaned and formatted in order to parse and populate the database tables. The features were defined based on data rules discovered from the feature creation process and other data analysis. The feature columns from both tables were populated by applying a Disjoint Factorization on the Part Description field and classifying each data component to a feature based on the data rules. The data was normalized using information from the Knowledge Base (KB). This KB table was populated using information obtained from subject matter experts (SME), analyzed aircraft manuals, and other documents.

This research addresses several data issues, most of which are mentioned in chapter two; however, there was another obstacle that made the evaluation of the mapping process difficult. There was not any training data or other reliable data available where with to compare our results. For accuracy and verification we relied on a group of Air Force SMEs to observe the mapping results. These persons observed the data results and gave an estimate as to percent of accuracy. The SMEs estimated the accuracy of the algorithm to be between 70 to 80 percent. This is extremely promising as there existed no sound algorithm to automate this process.

For experimental purposes the MUDD algorithm was compared to the Expectation-Maximization-Based Probabilistic (EMP) algorithm (See Appendix B). In comparing the two algorithms we wanted to see if the algorithms would agree on the mappings they selected and/or to what extent of agreement. In the event of mapping disagreements, we wanted to identify the reasons for the disagreements. We determined that by identifying these reasons we would discover information that would aid in making the MUDD algorithm better.

In conducting the experiment, the same blocking scheme was used for both algorithms and the same record pair combinations were processed by both algorithms. For each record pair comparison a quality of match score was computed by each algorithm. As an example in figure 36 the target record a_1 is being compared to four candidate records, b_1, b_3, b_7 and b_{11}. For each comparison two scores are computed one for the MUDD algorithm and one for the EMP algorithm. Once computed both scores were stored.

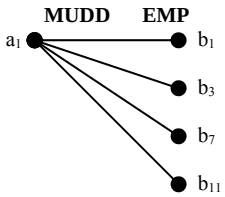


Figure 36. Record Comparison

The top five scores from each mapping iteration were saved. A mapping iteration is defined as the single event of comparing the appropriate candidate records to a single target record. As an example, if there are twenty-five target records then there would be twenty-five mapping iterations. The resulting Object translation scores from the MUDD algorithm were sorted in ascending order first. Note the lower the score the better the quality of match. The lowest five scores were considered the top five most probable to map to the target record. After the results for the MUDD algorithm were stored the scores for the EMP algorithm was sorted in ascending order. We used the insertion sort algorithm for its stable characteristic. The term stable suggests that the algorithm maintains the relative order of the objects while the objects are being processed. A stable algorithm was necessary because we realized that there were several record mappings with identical low scores. In capturing records that were common to both algorithms we wanted to ensure that in the event that two candidate records having the same EMP quality of match score, where one is also one of the top five records selected for the MUDD algorithm and the other is not, that the right record was selected. In other words a record with the same score doesn't get placed in front of it. The candidate records with their corresponding scores are shown in table XIX. The scores shown in this table are a result of one mapping iteration with a single target record. The table shows the MUDD algorithm results sorted in ascending order. The records b_{23}, b_1, b_{34}, b_5 and b_7 are the top records for the MUDD algorithm. Sorting the scores for the EMP algorithm yields the record order b_5, b_7, b_{23}, b_{90}, b_1. There are

multiple records with the value 18.345. Using a stable sorting algorithm maintains the relative order of the records to ensure that other records with the value 18.345 do not get placed before the record b_1.

Table XIX. Comparison of MUDD and EMP algorithm scores

Record	MUDD Score	EMP Score
b_{23}	10	13.5
b_1	15	18.345
b_{34}	35	25.343
b_5	40	5.23
b_7	80	5.23
b_{90}	100	13.5
b_{22}	140	56.77
b_2	200	26.09
b_{56}	220	18.345
b_9	240	18.345

In table XX we show the number of resulting records the algorithms had in common. The column headers indicate the number of records common to both. As an example the column labeled with the number five denotes the number of times that both algorithms had all five mapping results in common. During this experiment, the algorithms had the same five resultant mappings 271 times or 3.7 percent of the time. 73.22 percent of the time the algorithms had zero records in common. The algorithms did not share a large percentage of the same results.

Table XX. Agreement Table

	Number of Common Records					
	0	1	2	3	4	5
Total	5359	593	510	300	286	271
Percentage	73.22%	8.10%	6.96%	4.09%	3.90%	3.70%

In tables XXI and XXII we capture three categories of mappings,

1. Part-to-same part,
2. Part-to-system and
3. Part-to-different part.

The *part-to-same part* mappings denote the number of resultant candidate records that had the same part name in the Primary feature value as the target record. An example of this mapping is a target record *Left Rib Bolt* mapping to a candidate record *Rudder Bolt*. The Primary feature values in both records were identical (i.e. Bolt). The *part-to-system* mappings denote the number of resultant candidate records that

actually referenced a part system. This type of mapping indicates that the target record mapped to a candidate record with a system description. The *part-to-different part* mappings denote the number of resultant candidate records that had a different part name in the Primary feature value from that of the target record.

Table XXI. Empirical results for Category mappings (actual)

	Part-to-Same Part	Part-to-System	Part-to-Different Part
Gaston	16787	10306	9319
EM-Based	1323	14977	20112

Table XXII. Empirical results for Category mappings (percentages)

	Part-to-Same Part	Part-to-System	Part-to-Different Part
Gaston	46.10%	28.30%	25.59%
EM-Based	3.63%	41.13%	55.23%

In our algorithm 46.10 percent of the mappings were to candidate records having the same part description in the Primary feature field whereas the EMP algorithm only mapped 3.63 percent of it results to candidate records with the same part. 55.23 percent of the results from the EMP algorithm mapped to candidate records with a different part description in the Primary feature field while this category yielded 25.59 percent, the lowest percentage value, for our algorithm. There was a significant difference in the mapping results of both algorithms. We were able to identify at least five factors:

1. Inappropriate weighting,
2. Lack of enhancement data,
3. Poorly cleaned data,
4. Wrong feature classification and
5. Insufficient String comparator.

Applying the appropriate weight to the appropriate features proved extremely important. From section 5.2 we described why weighting is vital to the mapping process. The weights serve to identify the degree of significance a feature has across a data set. Misapplying weights causes a feature or a combination of features to have more significance than necessary. This is what happened with the weighting of the EMP algorithm. A subset of mappings from the new algorithm was used as input to the Expectation-

Maximization (EM) algorithm to estimate the unknown value m_k and u_k. Based on the EM algorithm the Noun, Positional and Descriptive features required the most weight. This conclusion was made because in the majority of the mappings the Positional and Descriptive values matched despite the absence of the matching Primary values. This left less weight to be distributed to the Primary feature.

Unlike the Enhancement component of the MUDD algorithm, the EMP algorithm does not have a component where with to bring in extra data related to records. This prevents the EMP algorithm from making optimal decisions in determining if a specific part should be mapped. As an example there are multiple candidate records with the same description *Rib*; however each record corresponds to a specific system such as Outboard Aileron System or Inboard Aileron System. Without the use of the associated system information an optimal match is only by happenstance. The EMP algorithm will view the descriptions *Rib*, *Outboard Aileron System*, *Inboard Aileron System* as three different records instead of descriptions that complement each other. There is no way for it to distinguish between records having identical part names but belonging to two entirely different aircraft systems.

Poorly cleaned data caused noticeable problems during the mapping process. Before processing the data, we attempted to remove all typographical errors and normalize the data by using standard names and formatting. We discovered that some of the wrong mappings by the MUDD algorithm were a result of the poorly cleaned data. The MUDD algorithm's inability to combat typographical errors was due to the exact matching string comparison measure used.

Incorrectly classifying terms in the wrong feature category causes significant problems because the incorrect weight value is assigned to the wrong term. This can cause records to be ranked higher or lower than actual. This issue is an extension of issue three, poorly cleaned data.

We discovered that using the wrong string comparator prevented the MUDD algorithm from making some correct mappings. For the MUDD algorithm a binary string comparator was used. In specific if the feature values were an exact match then the weights associated with the feature was computed in the Object

Translation value calculation otherwise no value was attributed to the terms. The Exact Matching technique was used because of the applications of data cleaning and data normalization to both data sets during the data preprocessing step. The importance of the type of comparison measure used is realized when records appear as 'WINGFLAP' instead of as 'WING FLAP'. During the data preprocessing stage description one would be processed as one word. In our research if a candidate record was 'WING FLAP', this phrase would get split into two words and placed into distinct feature categories. Since the MUDD algorithm uses a binary comparison measure scheme 'FLAP' compared to 'WINGFLAP' would be considered unequal, whereas if a categorical scheme like Bi-grams were used a better measure of similarity could be obtained. In the EMP algorithm we used the bi-gram string comparison measure. The bi-gram measure computes a value between zero and one. The closer the value is to zero the better the quality of match. The threshold value of .375 was used to determine the measure of similarity between two terms. If the bi-gram value was less than or equal to .375 then the two terms were consider equal else they were considered unequal.

In an effort to optimize the mapping results the Conditional Probability structure was used. We ran the experiment twice using the same population of records, one with the use of the structure and again without its use. An apparent issue from the onset was the memory required for the utilization of the structure. The components M and D required memory allocations proportionate to the total number of distinct values associated with the union of all Primary feature values and Noun Feature values. Recall the component M is a matrix-type structure which holds the values of the states that serve as mappings into the Distance vector D. The state values can be emulated using the *GetState(PrimIndex, NounIndex)* algorithm. See figure 37.

```
Algorithm GetState(int PrimIndex ,int NounIndex)
state=(row*PLANE_SZ + col);
IF (row!=0)
  dec=row;
  IF col > row AND col != row
     state=state-dec;
  Else
    state=state-(dec -1 );
END IF
return state;
```

Figure 37. Component M Transformation

The formal parameters PrimIndex and NounIndex denote the indices corresponding to the index of the Primary feature term and the index of the Noun feature term adjacent to the Primary term. Using these indices the value of the state location is returned.

The structure performed as anticipated. It recommended certain records over others based on the learned information from previous mappings. We first captured the results of 1500 record mappings. We set 1500 as a threshold value. When this threshold value was reached, records processed after that point were given additional weight based on its probability of being a match. As an example given the target record *Outboard Aileron Seal* and candidate records *Outboard Seal* and *Inboard Rib*, the Primary feature values in the candidate records *Seal* and *Rib* each have probability of being a mapping. This probability is computed using the historical data in the structure. If for instance records having a Primary feature value of *Seal* have a 70% probability and those having *Rib* as a Primary value have a 25%, then additional weight values corresponding to these percentages will be applied to each record's Object translation score. The additional weight is computed by multiplying the Primary feature weight by the probability value. For the above example the added weight for the record having *Seal* as a Primary value is computed as (.70 * 8), where 8 is the Primary feature weight and .70 is the probability. Although the structure was able to recommend certain records over others, it was not able to distinguish between records with similar Primary feature values. For instance given the target record *Outboard Aileron Seal* and candidate records *Outboard Seal* and *Outboard Aileron System*, *Inboard Rib, Inboard Seal, Spoiler Seal and Rudder Seal*, the Primary feature values in the candidate records *Seal, System,* and *Rib* each have probability of being a mapping. From our previously processed data, the records ending in *Seal* have the higher probability of being a mapping. After applying the additional weight value, all the records ending in *Seal* are placed at the top of the list as most likely to be the match. Using our example *Outboard Seal, Inboard Seal, Spoiler Seal* and *Rudder Seal* will be placed at the top. Although these records refer to *Seal*, based on the other components of their descriptions the chance of *Inboard Seal, Spoiler Seal* and *Rudder Seal* being an actual mapping is unlikely. This is a characteristic that the EMP algorithm suffered from. As a result of using the structure, there was a mapping accuracy decrease. More work must be done to the weighting process to address this issue.

CHAPTER SIX

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

In summary a record linkage approach was designed and implemented to address the four primary characteristics of the MUDD Classification problem,

- Mapping processes,
- Unique and unreliable data,
- Domain specific data and
- Disjoint data sets.

Two disjoint sets A and B containing aircraft part data are identified. The items in set B are defined as unique classifications in which elements from set A must map to. In this research we introduced the concept of identifying discovery items which assist the data analyst in constructing relevant features from a single field in a relation for the data processing step. In outlining the discovery items, the analyst was able to establish ranking order of importance among the features. We introduced a new sum-order weighting approach to systematically compute weight values that correspond to the analyst-defined ranks and constraints established by the analyst. We also introduced a learning mechanism that assists in recording and maintaining dynamic relationships among the data. Learned data as well as the notion of checkpoint data was introduced which provides a means of enhancing the record data to generate a more dominate mapping candidate. To establish a form of measurement and validation, a method of quantifying the relationships is presented. In addition, a rule generation, blocking and filtering approach is presented. Below is an itemized list of methodologies and approaches proposed in order to combat the issues described in section 2.

- New data linkage algorithm
- Approach to Enhance mapping data using record objects
- Grammar-based Feature Construction approach
- Sum-Ordering Weighting approach
- Part-Breakdown Blocking and Filtering approach
- Conditional Probability Structure

For experimental purposes the MUDD algorithm was compared to the popular EM-based probabilistic record linkage algorithm. The MUDD algorithm outperformed the EM-based algorithm; however our algorithm made incorrect mappings as a result of poorly cleaned data, incorrectly classified terms and the use of an inefficient string comparison model. One difference between our approach and most traditional approaches is that each feature contained multiple values whereas in traditional record linkage solutions, per record there is normally a single value associated with each value. As an example if one of the features of a record was *Social Security Number* (SSN) there is generally only one number associated with this feature instead of multiple numbers. Our approach creates features from one record field; in this case the part description field. In addition no training data was needed and external data was used to make optimal record mappings.

For future efforts, we recommend using a categorical comparison measure such as bi-grams or Jaro's algorithm. Additional research must also be done in the area of the Conditional Probability Structure to ensure optimal decision making.

References

[1] Al-Lawati, A., Lee, D., and McDaniel, P. (2005). Blocking-Aware Private Record Linkage. *In ACM SIGMOD Workshop on Information Quality in Information Systems (IQIS)*, Baltimore, MD., pg. 59-68.

[2] Baxter, R., Christen, P. and Churches, T. (2003). A Comparison of Fast Blocking Methods for Record Linkage, *First Workshop on Data Cleaning, Record Linkage and Object Consolidation, KDD*, Washington D.C., pg. 25-27.

[3] Bell, G. and Sethi, A. (2001). Matching Records in a National Medical Patient Index, *Communications of the ACM*, Vol. 44. No. 9, ACM Press, pg. 83-88.

[4] Bhattacharya, I. and Getoor, L. (2004). Iterative Record Linkage for Cleaning and Integration. *9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, ACM Press, pg. 11-18.

[5] Campbell, Keven, (2006). Rule Your Data with The Link King, http://www2.sas.com/proceeding/sugi30/020-30.pdf

[6] Ceglowski, M., Coburn, A. and Cuadrado, J. (2001), Semantic Search of Unstructured Data Using Contextual Network Graphs. IEEE Computer Society.

[7] Chen, C. and Lynch, K. (1992). Automatic Construction of Networks of Concepts Characterizing Document Databases. *IEEE Transaction on Systems, Man and Cybernetics*, Vol. 22, pg. 885-902.

[8] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2003). *Introduction to Algorithms*, McGraw Hill Book Co., 2nd Edition

[9] Crouch, C.J. (1990). An Approach to the Automatic Construction of Global Thesauri. *Information Processing and Management*, pg. 629-640.

[10] Culotta, A. and McCallum, A. (2005). Joint Deduplication of Multiple Record Types in Relational Data, *Conference on Information and Knowledge Management Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM Press, pg. 257-258.

[11] Elfeky, M., Verykios, V. and Elmagarmid, A. (2002). TAILOR: A Record Linkage Toolbox, *IEEE Proceedings of the 18th International Conference on Data Engineering*, IEEE Computer Society.

[12] Fellegi, P. and Sunter, A. (1969). A Theory for Record Linkage, *Journal of the American Statistical Association,* pg. 1183-1210.

[13] French, J., Powell, A., and Schulman, E. (1997). Applications of Approximate Word Matching in Information Retrieval. *Proceedings of the sixth international conference on Information and knowledge management*, ACM Press. pg. 9-15.

[14] Frigui, Hichem and Nasraoui, O. (2002). Simultaneous Categorization of Text Documents and Identification of Cluster-dependent Keywords. *In Proceedings of FUZIEEE.* pg. 158-163.

[15] Gaston, B.D. (2001). *Sorting and Searching using a Data Structure Based on the 3-Dimensional Torus Network.* Masters Thesis., Oklahoma State University.

[16] Gelbukh, A., Lopez, A., Baeza-Yates, R. (2001). Text Mining with Conceptual Graphs, *IEEE nternational Workshop on Natural Language Processing & Knowledge Engineering (NLPKE)*. pg. 893-903.

[17] Grefenstette, G. (1992). SEXTANT: Exploring Unexplored Contexts for Semantic Extraction from Syntactic Analysis. *Proceedings of the 30th Annual meeting of Association of Computational Linguistics*. pg. 324-326.

[18] Hole, W. and Srinivasn, S. (2000). Discovering Missed Synonymy in a Large Concept-Oriented Metathesaurus, *Proceedings of AMIA Symposium,* pg. 354-358.

[19] Hong, J. (2006). *An Overview of Latent Semantic Indexing*, http://www.cs.berkeley.edu/~jasonh/classes/sims240/sims-240-final-paper-lsi.htm

[20] Jin, L., Li, C., and Mehrotra, S. (2003). Efficient Record Linkage in Large Data Set., *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, IEEE Computer Society.

[21] Kamath, C. (2000). *Mining Data for Gems of Information*. http://www.llnl.gov/str/Kamath.html.

[22] Kawahara, M. and Kawano, H. (2000). An Application of Text Mining: Bibliographic Navigator Powered by Extended Association Rules, *In Proceedings of the 33rd Hawaii Inter. Conference of System Sciences*. pg. 2-10.

[23] Kummamuru, K., Dhawale, A. and Krishnapuram, R. (2003). *The IEEE International Conference on Fuzzy Systems*, St. Louis, MO. pg. 772-777.

[24] Labovitz, M. (2003). What is Data Mining and What Are Its Uses?, *Darwin Magazine*

[25] Li, Bing, Quan, H., Fong, A. and Lu, M. (2006). Assessing record linkage between health care and vital statistics databases using deterministic methods, *BMC Health Services Research 2006*, Vol. 6.

[26] Martin, J. (1997). *Introduction to Languages and the Theory of Computation*, McGraw Hill, 2nd Edition.

[27] Minton, S. and Nanjo, C. (2005). A Heterogeneous Field Matching Method for Record Linkage, *Proceedings of the Fifth IEEE International Conference on Data Mining*, IEEE Computer Society, pg.1-8

[28] Monostori, K., Zaslavsky, A., Schmidt, H. (2001). Efficiency of Data Structures for Detecting Overlaps in Digital Documents. 24th *Australian Computer Science Conference (ACSC)*, IEEE Computer Society, pg. 140-147.

[29] Pekar, V., Krkoska, M., Staab, S. (2004). Feature Weighting for Co-occurrence-based Classification of Words, *In Proceedings of the 20th International Conference on Computational Linguistics, Geneva*, pg. 799-805.

[30] Peltonen, J., Sinkkonen, J., and Kaski S. (2002). Discrimininative Clustering of Text Documents. *Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02)*, Vol. 4, pg. 1956-1960.

[31] Pfeifer, U., Poersch, T., and Fuhr, N. (1996). Retrieval Effectiveness of Proper Name Search Methods, *Information Processing and Management*, pg. 667-679.

[32] Rhem, A.J. (2001). A Framework for Knowledge Acquisition. *White Paper*, A.J. Rhem and Associates Inc., pg. 3-14.

[33] Senellart, P. and Blondel, V. (2002). Automatic Extraction of Synonyms in a Dictionary. *Text-Mining Workshop,* Arlington ,VA.

[34] Sharman, R. A., Jelinek, F., and Mercer, R. (1990). Generating a Grammar for Statistical Training*. In Proceedings of the June 1990 DARPA Speech and Natural Language Workshop*. Hidden Valley, Pennsylvania., pg.267-274.

[35] Swartz, R. (2004). *The Importance of Quality Data in Building Strong Relations*. http://www.crm2day.com/library/EEpyyuAZlEZtoxvwjB.php.

[36] Tan, P., Steinback, M. and Kumar V. (2005). *Introduction to Data Mining*, Pearson-Addison Wesley, Boston, MA.

[37] Tanenbaum, A. (1996), *Computer Networks*, Prentice Hall., 3rd Edition.

[38] Thuraisingham, B. and Ceruti, M. (2000). Understanding Data Mining and Applying it to Command, Control, Communications and Intelligence Environments*, 24th International Computer Software and Applications Conference*, IEEE Computer Society, pg. 171-175.

[39] Turney, P. (2001). Mining the Web for Synonyms. *Proceedings of the Twelfth European Conference on Machine Learning (ECML-2001)*, pg. 491-502.

[40] Winkler, W. (2000). *Machine Learning, Information Retrieval, and Record Linkage*, http://www.amstat.org/Sections/Srms/Proceedings/papers/2000_003.pdf.

[41] Zhang, D. and Zhou, L. (2004). Discovering Golden Nuggets: Data Mining in Financial Application, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 34, No. 4., pg. 513-522.

[42] http://www.wired.com/wired/archive/12.01/stuntbots.html, (2004). *Attack of the Stuntbots*

[43] http://amchp.org, (2006). *Advanced Data Records Linkage for MCH Professional*

[44] http://www.cormactech.com/neunet/whatis.html, (2005). *What is a Neural Net*

[45] http://www.cs.colorado.edu/~martin/SCP/slp_ch1.pdf, (2006). *Speech and Language Processing: Introduction to Natural Language Processing, Speech Recognition*

[46] http://www.leedscarroll.com/Graphics/Tech_Illustration/PlaneParts.shtml, (2002). *Plane Parts: Technical Illustrations*

[47] http://www.neurosolutions.com/products/ns/whatisNN.html, (2006). *What is a Neural Network*

[48] http://www2.lib.udel.edu/subj/hist/resguide/soundex.htm, (2006). *SOUNDEX*

Appendix A

```
/************************************************************
/** The following program automatically generates a context
/** free grammar based on sample input sentences of a Language.
/**The sentences are stored in a file designated by the user. The programs
/**prints the original sentences, valid sentence combination patterns
/** and the list of valid productions as output.  The file name of the sentences
/** can be modified in the getLanguage() method.
/**
/** SAMPLE INPUT SENTENCES
/**      ab
/**      aabc
/**      aaabbc
/**      aaaabbbc
/**
/**
/** SAMPLE PRODUCTIONS
/**      S1->-9-3-11
/**      9->aaa
/**      3->ab
/**      11->bbc
/**      S2->-4-3-5
/**      4->aa
/**      5->bc
/**      S3->-0-3-2
/**      0->a
/**      2->c
/**      S4->-3
/**      START->S1 | S2 | S3 | S4 |
/************************************************************
```

```
import java.io.*;
import java.util.*;

public class CFG_Grammar{
        public static void main(String[] args)
        {
        int index=0;
        String[] text;
        Vector text1=new Vector();
         String[] perm=new String[1500];    //holds all possible pattern combinations of size N_size
         sentenceNode[] sn=null;
        getLanguage(text1);                 //call to read sentences into Vector array
         text=sortLanguage(text1);          //call to sort sentences by sentence length
        printLanguage(text);                 //call to print sentences of language
         Vector comb=new Vector();          //holds all possible patterns assoicated with text
         index=getLongSentence(text);       //get index of Longest sentence O(s)
```

```
 int N_size=text[index].length();     //get size value
 sn=createSentenceObject(sn,text);  //create Sentence Objects

 /*submit longest sentence*/
 combination(N_size,perm);              //get all possible combinations of arbitrary string of size N_size and
put in perm-gets TEMPLATE

 for(int i=0;i<text.length;i++){
 getCombTransform(sn[i].sentence,perm,comb);   //based on combination(sub-patterns) template from
previous function get all combinations of sentence 'text'
 }//end LOOP

 String[] text2=sortLanguage(comb);  //sorts total number of combinations
 comb.clear();
 copyVector(comb,text2);            //copies combinations to  text string array
 getProductions(sn,comb);
 buildProductions(sn,comb);         //build production list
}//END main

public static sentenceNode[] createSentenceObject(sentenceNode[] sn,String[] text){
            sn=new sentenceNode[text.length];   //create nodes associated with each sentence
    for(int i=0;i<sn.length;i++){
            sn[i]=new sentenceNode(text[i]); //insert sentences in structure O(s)
    }
    return sn;
}//END method

/**
 * method reads in sentences of a Language L from a file
 * O(s) s is the number of sentences
 * @param text
 */
public static void getLanguage(Vector text){
            String sent="";
            try{
                        File fp=new File("language1.txt");        //read file with Sample sentences
                        FileReader fr=new FileReader(fp);
                        BufferedReader br=new BufferedReader(fr);
               sent=br.readLine();
               while(sent != null){
                        text.add(sent);
                        sent=br.readLine();
               }
            }catch(IOException e){}
}//END method


public static void Recursive(Vector p,Vector non){
  int flag=1;
  int index=-1;

  Vector list=new Vector();
            while(index != -999){
                        flag=0;
                        if((index=findCommon(p,list,index+1)) == -999)break;
                        String nt=((String)non.elementAt(index));
```

```
                String pat=((String)p.elementAt(index));
            for(int i=0;i<p.size();i++){
                if(i != index){
                    String word=((String)p.elementAt(i));
                    if(word.indexOf(pat)!= -1){
                        word=word.replaceAll(pat,"-"+nt);
                        p.setElementAt(word,i);
                flag=1;
                        }
                }

            }
            if(flag==1)index=-1;
        }
            printProductions(non,p);
} //END method


        /**
         * method searches the list of productions to find the minimum most specific pattern that has more than
         * one nonTerminal element but is .  If the pattern found has already been used the search continues on for
         * the next best production.
         * O(p) where p is the number of discovered productions
         * @param a
         * @param list
         * @param start
         * @return
         */
        public static int findCommon(Vector a,Vector list,int start){
        int min=99;
        int index=-999;
        int i=0;
         for(i=start;i<a.size();i++){
         String pattern=((String)a.elementAt(i));
         if(list.contains(pattern))continue;
         StringTokenizer st=new StringTokenizer(pattern,"-");
         int val=st.countTokens();
         if(val > 1 && val < min){
                 min=val;
                 index=i;
         }
         }

         if(index != -999)
         list.add((String)a.elementAt(index));
           return index;
        }//END method


        /**
         * method to copy contents of a String array into a vector array
         * O(b), where b is length of String array
         * @param a
         * @param b
         */
        public static void copyVector(Vector a,String[] b){
```

```
		for(int i=0;i<b.length;i++){
			a.add(b[i]);
			System.out.println("["+i+"]"+b[i]);
		}
	}//END method


	public static String[] sortLanguage(Vector text){
		String temp="";
		int len=0;
		int len1=0;
		String[] words=new String[text.size()];
		for(int i=0;i<text.size();i++){		//sentences in temp structure
			words[i]=((String)text.elementAt(i));
		}

		for(int i=1;i<words.length;i++){ //O(s-1) where a is list of sentents
			len=words[i].length();
			for(int j=0;j<i;j++){		//O(s)
				len1=words[j].length();
				if(len < len1){ temp=words[i];
				/*SHIFT REMAINING VALUES*/
				for(int k=i;k>j;k--){	//O(s-i)
					words[k]=words[k-1];
				}
				words[j]=temp;
				break;
				}
			}
		}

		return words;
	}//END method

 public static void printLanguage(String[] words){
	System.out.println("****BEGIN LANGUAGE******");
		for(int i=0;i<words.length;i++){
			System.out.println("SENTENCE["+i+"]"+words[i]);
			}
		System.out.println("****END LANGUAGE******\n");
}//END method


	/**
	 * method searches for the first occurance of a pattern combination of size two
	 * O(c) where c is the total possible number of pattern combinations computed based on
	 * the sentences
	 *
	 * @param comb
	 * @return
	 */
	public static int getSizeTwoPat(Vector comb){
	  int index=0;
		for(int i=0;i<comb.size();i++){
			 if(((String)comb.elementAt(i)).length() > 1){
				index=i;
```

```
                                break;
                        }

                        }
        return index;
}//END method

/**
 * method to find: given a sentence size X find the next sentence in the list that is lower
 * O(s) where s is the total number of sentences
 * @param sn
 * @param index
 * @return
 */
public static int getNextSizeDown(sentenceNode[] sn,int index){
   int size=sn[index].sent_len;
   index--;
        for(int i=index;i>=0;i--){//starts from position value denoted by index and decrements
           int nextSize=sn[i].sent_len;
                if(nextSize < size) return nextSize;   //returns the size of the sentence that is the next
sentence size lower that the current sentence
        }
        return -1;
}//END method


/**
 * method to find the first pattern combination of size sz, starts from the end of the list and goes backward
 * O(c) where c is the number of pattern combination
 *
 * @param comb
 * @param sz
 * @return
 */
public static int getCombIndex(Vector comb,int sz){
        for(int i=comb.size()-1;i>=0;i--){
                String pat=(String)comb.elementAt(i);
                if(pat.length()==sz)return i;
        }
        return -1;
}//END method


public static void buildProductions(sentenceNode[] sn,Vector comb){
   int len=sn.length;
   int beginN_TermCt=1;
   Vector NONTERM=new Vector();
   Vector prod1=new Vector();
   String start="";
   int nextInd=0;
   int nextProd=0;
   int val=0;
   len--;
        for(int i=len;i>0;i--){   //get sentence node  O(s)
         int lastInd=sn[i].last.size();      //get size of storage which hold combination indices
```

```
            boolean found=false;
           /*get last value*/
                    while(!found && lastInd > 0){
                            lastInd--;
                             val=((Integer)sn[i].last.elementAt(lastInd)).intValue();
                            for(int j=i-1;j>=0;j--){  //get sentence node below       it with the same value in
its storage
                                    int lastInd1=sn[j].last.size();  //get size of below node's storage
                                    lastInd1--;
                                    int val1=0;
                                    for(int k=lastInd1;k>=0;k--){   //check to see if value VAL is in this
storage
                                    val1=((Integer)sn[j].last.elementAt(k)).intValue();
                                    if(val == val1 || val1 > val){nextInd=j; nextProd=k; break; } //either
found match or
                                                                                    // match is not in current
                                                                            sentence node
                                    }

                                    if(val == val1){found=true; break;}
                            nextInd=0;
                            nextProd=0;
                            }
                    }
                    String pd="";
                    if(sn[i].prod.size()>=1) pd=(String)sn[i].prod.elementAt(lastInd);   //get curretnly
processed production
                    else continue;
                    /**prod1 holds global productions*/
                    if(!prod1.contains(pd)){
                            NONTERM.add("S"+String.valueOf(beginN_TermCt));
                            start=start+("S"+String.valueOf(beginN_TermCt)) +" | ";   //save all letter
NONTERM for the
                                                                                    //START symbol

                    prod1.add(pd);
                    StringTokenizer st=new StringTokenizer(pd,"-");
                    while(st.hasMoreTokens()){                      //make additional productions from
production pd i.e. 0-0-2-4
                            int value=Integer.parseInt(st.nextToken());
                            if(!NONTERM.contains(String.valueOf(value))){
                                    NONTERM.add(String.valueOf(value));
                                    prod1.add(comb.elementAt(value));
                            }
                    }

                    beginN_TermCt++;
                    }

                    /*do the samething for the common sentence NODE*/
                    String pd1="";
                    if(sn[nextInd].prod.size()>=1) pd1=(String) sn[nextInd].prod.elementAt(nextProd);
                    else continue;
                    if(!prod1.contains(pd1)){
                            NONTERM.add(("S"+String.valueOf(beginN_TermCt)));
                            start=start+("S"+String.valueOf(beginN_TermCt)) +" | ";
```

```
                prod1.add(pd1);
                StringTokenizer st=new StringTokenizer(pd1,"-");
                while(st.hasMoreTokens()){
                        int value=Integer.parseInt(st.nextToken());
                        if(!NONTERM.contains(String.valueOf(value))){
                                NONTERM.add(String.valueOf(value));
                                prod1.add(comb.elementAt(value));
                        }
                }
                beginN_TermCt++;
                }
        }
        NONTERM.add("START");
        prod1.add(start);
        Recursive(prod1,NONTERM);
}//END method


public static void printProductions(Vector nTerm,Vector pd){
        System.out.println("\n****PRODUCTION LIST******");
        for(int i=0;i<nTerm.size();i++){
                System.out.println((String)nTerm.elementAt(i)+"->"+(String)pd.elementAt(i));
        }
}//END method

/**
 * method to get all possible valid productions of a sentence, productions are placed in a production vector
 * and the index of the main combination pattern of the production is stored
 * @param sn
 * @param comb
 */
public static void getProductions(sentenceNode[] sn,Vector comb){
        int len=sn.length;
        len--;
        int index=getSizeTwoPat(comb);      //index of first pattern with length of 2
        int sz=0;
        int ind=0;
        for(int i=len;i>=0;i--){       //current sentenceNOde being processed  O(s)
                String sentence1=sn[i].sentence;
                if((sz=getNextSizeDown(sn,i))!=-1) ind=getCombIndex(comb,sz);  //get index of first
pattern in comb of
                                                                        // lenght sz
                if(ind == -1) continue;//String sentence2=sn[i].sentence;
                for(int j=ind;j>=index;j--){         //loop to traverse through patterns
                        String pat=(String)comb.elementAt(j);       //gets pattern from combination
                        String word="";
                        if((word=checkPat(sentence1,pat,ind,comb,word)) != null){    //start from the
first pattern of length
                                                                        // sz as not to leave out
                                                                any
                                if(!sn[i].prod.contains(word)){
                                        sn[i].prod.add(word);
                                        sn[i].setLast(j);                       //store the index of the
combination pattern
                                                                    // that was found in the sentence
                        }
```

```
                                        }
                                }

                        }
        }//END method


public static String checkPat(String sentence,String pattern,int index, Vector comb,String newComb){

        int ind1=comb.indexOf(pattern);
        int ind=sentence.indexOf(pattern);  //find out if pat is in sent
        if(ind != -1)sentence=sentence.replaceAll(pattern,("-"+String.valueOf(ind1)));
        else return null;  //pattern not found

        for(int i=index;i>=0;i--){                      //scan remaining combination patterns
                String p=((String)comb.elementAt(i)).trim();
                String replace="-"+String.valueOf(i);
                sentence=sentence.replaceAll(p,replace);        //find and replace all occurances of the pattern
        with a symbol
        }
        newComb=sentence;

        /*make sure new sentence has been completely transformed by checking to see if any letters remain*/
        for(int i=0;i<newComb.length();i++){
                if(isAlpha(newComb.charAt(i))) return null;
        }
        return newComb;
}//END method


        /**
         * method to determine is char is a LETTER
         * O(1)
         * @param c
         * @return
         */
        public static boolean isAlpha(char c){
                char start='a';
                char end='Z';
                if(((int)c) > ((int)start) && ((int)c)<((int)end))return true;
                else return false;
        }//END method


        /**
         * method to determine longest sentence in the list
         * O(s) where s is the number sentences being processed
         * @param a
         * @return
         */
        public static int getLongSentence(String[] a){
        int max=0;
        int ind=0;
                for(int i=0;i<a.length;i++){
                        int len=a[i].length();
```

```
                    if( len > max){max=len; ind=i;}
            }
            return ind;
}//END method

/**
 * method to get valid combinations of a sentence
 * @param text
 * @param perm
 * @param comb
 */
public static void getCombTransform(String text,String[] perm, Vector comb){
String word="";
int index=0;
int old=0;
int textA_len=text.length();
 for(int i=0;i<perm.length;i++){  //O(p) where p is the list of valid combinations
            if(perm[i] == null)break;
            String curr=perm[i].trim();
            word="";

            for(int j=0;j<curr.length();j++){  //O(pl) where pl is the length of the current combination
                        char c=curr.charAt(j);
                        if(c != ' '){
                                    index=Integer.parseInt(String.valueOf(c));       //get integer value of character
                                    if(j==0)old=index;                          //assign value of index to old if j==0
                                    else if(index != old+1){word=""; break;}         //index values (index and old)
should be 1 unit
                                                                                    //value apart otherwise invalid
                                                                                combination
                                    old=index;
                                   //assign new value of index to old

                                    if(index <= textA_len)                      //make sure value of index is not
longer that the actual
                                                                                // value of the text
                                    word=word+text.charAt(index-1);
                                    else word="";

                        }
            }
            if(!comb.contains(word) && !word.equals("")){          //insert combination word into vector if
not a duplicate
                        comb.addElement(word);
            }
 }

}//END method



/**
 * method to get valid combinations of a sentence
 * @param text
 * @param perm
 * @param comb
```

```
 */
public static void GetCombTransform(String text,Vector perm, Vector comb){
String word="";
int index=0;
int old=0;
int textA_len=text.length();
 for(int i=0;i<perm.size();i++){  //O(p) where p is the list of valid combinations
            if(perm.elementAt(i) == null)break;
            String curr=((String)perm.elementAt(i)).trim();
            word="";

            for(int j=0;j<curr.length();j++){  //O(pl) where pl is the length of the current combination
                        char c=curr.charAt(j);
                        if(c != ' '){
                                    index=Integer.parseInt(String.valueOf(c));       //get integer value of character
                                    if(j==0)old=index;                              //assign value of index to old if j==0
                                    else if(index != old+1){word=""; break;}         //index values (index and old)
should be 1 unit
                                                                                     // value apart otherwise invalid
                                                                                combination
                                    old=index;
                                   //assign new value of index to old

                                    if(index <= textA_len)                          //make sure value of index is not
longer that the actual
                                                                                    // value of the text
                                    word=word+text.charAt(index-1);
                                    else word="";

                        }
            }
            if(!comb.contains(word) && !word.equals("")){          //insert combination word into vector if
not a duplicate
                        comb.addElement(word);
            }
 }

}//END method

/**
 * method to get combination template by position values
 * O(s'^2)
 *
 * @param MAX
 * @param perm
 * @return
 */
public static int combination(int MAX, String[] perm){
int CurSize=1;
int code=0,test=0;
int[] template=new int[MAX];
int permInt=0;

/*loop to get single values in combination (e.g. 1, 2, 3, 4)*/
/**O(s') where s' is the size of the longest sentence being processed*/
for(int i=0;i<MAX;i++){
```

```
                template[i]=i+1;
                perm[permInt]=String.valueOf(template[i]);
                permInt++;
        }

        /**O(1)*/
        for(int i=2;i<=MAX;i++){                        //size of current permutation
                /*O(s')*/
                for(int j=0;j<MAX;j++){                 //No. of items in array
                        CurSize=1;                      //size of items in current permutation
                        Integer dum=new Integer(template[j]);   //obtain first item of the permutation
                  code=0;                               //FLAG
                  /*O(s')*/
                 for(int k=0;k<MAX;k++){                //no. to choose from
           if(template[k] > template[j]){       //current value should be greater than num in currently in permuation
                  if(code==0){                  //first iteration
                        perm[permInt]=dum.toString();   //concat digit onto permutation string
                        test=k;                 //*captures index of first obtained index value in permutation
                        code=1;
                  }
                  Integer dum2=new Integer(template[k]);
                  perm[permInt]=perm[permInt]+" "+dum2.toString();      //*capture next value in permutation
                  CurSize++;
           }
           if(CurSize == i){
                  CurSize=1;
                  if((MAX-test)+1 >= i){k=test; code=0; }               //reset variables
                  permInt++;                                    ///increment permutation count

           }

                        }
                }

        }
        return permInt;
   }

}//END method


class sentenceNode{
        Vector permArray;
        Vector mapArray;
        Vector buffer;
        Vector prod;
        String sentence;
        int sent_len;
        Vector last=new Vector();
        sentenceNode() {
                permArray=new Vector();
                mapArray=new Vector();
                buffer=new Vector();
                last=new Vector();
                prod=new Vector();
        }
```

```
        sentenceNode(String sentence) {
                permArray=new Vector();
                mapArray=new Vector();
                buffer=new Vector();
                prod=new Vector();
                last=new Vector();
                this.sentence=sentence;
                this.sent_len=sentence.length();
        }
        protected void setLast(int val) {
                last.add(new Integer(val));
        }

}
```

Appendix B

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class EM_Driver{
        static double[][] TrainingData;
        static int featureCt;
        static BufferedReader in;

        public static void main(String[] args){
                String filename="StatFile.txt";
                readTEXTData(filename);
                EM_Linkage em=new EM_Linkage(TrainingData,5) /* 5=feature count*/
                System.out.println(" STARTING ITERATIONS!!");
                em.iteration();
                System.out.println(" ENDING ITERATIONS!!");
        }

        public static void readTEXTData(String filename)
          {
           try {
                File inFile =new File(filename);
                in = new BufferedReader(new FileReader(inFile));
            } catch (Exception e) {}
            int N = 0,d = 0;
            try {
              N = Integer.valueOf(in.readLine()).intValue();
              d = Integer.valueOf(in.readLine()).intValue();
              featureCt=d;
            } catch (Exception e) {}

            System.out.println("DIM: "+N+" "+d);
            TrainingData = new double[N][d];
            try {
              ParseData(in,TrainingData);
              in.close();
            } catch (Exception e) {}
          }


        public static void ParseData(BufferedReader in,double[][] Data) throws Exception
          {
           int i,j,k;
           String str;
           StringBuffer word = new StringBuffer();

           for(i=0;i<Data.length;i++)
```

```
            {
             str = in.readLine(); k = 0;
             System.out.println();
             for(j=0;j<Data[i].length;j++)
             {
              word.setLength(0);
              for(;k<str.length(); k++)
                if(str.charAt(k) != ' ') break;
              for(;k<str.length(); k++)
              {
               char c = str.charAt(k);
               if(c == ' ') break;
               word.append(str.charAt(k));
              }
              Data[i][j] = Double.valueOf(word.toString()).doubleValue();
             //  if(Data[i][j] >= 0.0)Data[i][j]=.1;
              System.out.print(Data[i][j]+ " ");
             }
            }
           }
}


class EM_Linkage{
        double p;   //proportion of the matched record pairs |M|/N
        double[] m; //match
        double[] u; //non-match
        double p_old;   //proportion of the matched record pairs |M|/N
        double[] m_old; //match
        double[] u_old; //non-match
        int k;      //total number of features
        int N;                  //total number of record pairs
        double[] g_m;
        double[] g_u;
        double[][] data;

        EM_Linkage(double[][] Data,int featureCt){
                data=Data;
                k=featureCt;
                N=Data.length;
                System.out.println("DATA SIZE "+data.length);
                initialize();
        }

        public void initialize(){
                m=new double[k];
                u=new double[k];
                m_old=new double[k];
                u_old=new double[k];
                g_m=new double[N];
                g_u=new double[N];
                for(int i=0;i<k;i++){
                        m[i]=0.8;
                        u[i]=0.2;
                }
```

```
        p=0.5;
        printParameters(1);
}

public void iteration(){
        double threshold=100.0;
        int interate=0;
        while(threshold > 0.0000000005 ){
        interate++;
//      System.out.println("getOldValue");
        getOldValue();
//      System.out.println("expectation");
        expectation();
        //System.out.println("maximization");
        maximization();
        //System.out.println("compare");
        threshold=compareOldNewValue();
//      System.out.println("print");
        printParameters(interate);
        }
}
public void printParameters(int iteration){
        System.out.println("ITERATION: "+iteration);
 System.out.println("\nM");
 for(int i=0;i<k;i++){
         System.out.print(m[i]+" ");
 }

 System.out.println("\nU");
 for(int i=0;i<k;i++){
         System.out.print(u[i]+ " ");
 }
 System.out.println("\nP:" +p);

}

public double compareOldNewValue(){
        double error=0.0;
        for(int i=0;i<this.k;i++){
                error=error+(Math.abs(m[i] - m_old[i]));
                error=error+(Math.abs(u[i] - u_old[i]));
        }
        error=error/10.0;
        return error;
}

public void getOldValue(){
        for(int i=0;i<k;i++){
                m_old[i]=m[i];
                u_old[i]=u[i];
        }
        p_old=p;
}

public void expectation(){
        /*compute G_m[]*/
```

```
		for(int i=0;i<N;i++){
			g_m[i]=computeG_m(data[i]);
		}

		/*compute G_u[]*/
		for(int i=0;i<N;i++){
			g_u[i]=computeG_u(data[i]);
		}
	}

	public void maximization(){
		/*compute M_k[]*/
		for(int i=0;i<k;i++){
			m[i]=computeM_k(data,g_m,i);
		}

		/*compute U_k[]*/
		for(int i=0;i<k;i++){
			u[i]=computeU_k(data,g_u,i);
		}

		/*compute P*/
		p=computeP(g_m);


	}


	public double computeG_m(double[] c_vector){
		double G_m=0.0;
		double numerator=1.0;
		double denominator1=1.0;
		double denominator2=1.0;

		/*compute numerator portion*/
		for(int i=0;i<k;i++){
		numerator=numerator * (Math.pow(m[i],c_vector[i]) * Math.pow((1.0-m[i]),(1.0-
c_vector[i])));
		}
		numerator=p*numerator;

		/*compute 1st part of denominator portion*/
		for(int i=0;i<k;i++){
			denominator1=denominator1 * (Math.pow(m[i],c_vector[i]) * Math.pow((1.0-
m[i]),(1.0-c_vector[i])));
		}
		denominator1=p*denominator1;

		/*compute 2nd part of denominator portion*/
		for(int i=0;i<k;i++){
			denominator2=denominator2 * (Math.pow(u[i],c_vector[i]) * Math.pow((1.0-
u[i]),(1.0-c_vector[i])));
		}
		denominator2=(1.0 - p)*denominator2;

		G_m=numerator/(denominator1 + denominator2);
```

```
		return G_m;
	}

	public double computeG_u(double[] c_vector){
		double G_u=0.0;
		double numerator=1.0;
		double denominator1=1.0;
		double denominator2=1.0;

		/*compute numerator portion*/
		for(int i=0;i<k;i++){
			numerator=numerator * (Math.pow(u[i],c_vector[i]) * Math.pow((1.0-u[i]),(1.0-
c_vector[i])));
		}
		numerator=p*numerator;

		/*compute 1st part of denominator portion*/
		for(int i=0;i<k;i++){
			denominator1=denominator1 * (Math.pow(u[i],c_vector[i]) * Math.pow((1.0-
u[i]),(1.0-c_vector[i])));
		}
		denominator1=p*denominator1;

		/*compute 2nd part of denominator portion*/
		for(int i=0;i<k;i++){
			denominator2=denominator2 * (Math.pow(m[i],c_vector[i]) * Math.pow((1.0-
m[i]),(1.0-c_vector[i])));
		}
		denominator2=(1.0 - p)*denominator2;

		G_u=numerator/(denominator1 + denominator2);
		return G_u;
	}



	public double computeM_k(double[][] c_vector,double[] g_m,int k){
		double value=0.0;
		double value1=0.0;
		double M_k=0.0;
		for(int i=0;i<N;i++){
			value=value+(c_vector[i][k] * g_m[i]);  /*compute numerator*/
			value1=value1+g_m[i];  /*compute denominator*/
		}
		M_k=value/value1;
		return M_k;
	}

	public double computeU_k(double[][] c_vector,double[] g_u,int k){
		double value=0.0;
		double value1=0.0;
		double U_k=0.0;
		for(int i=0;i<N;i++){
			value=value+(c_vector[i][k] * g_u[i]);  /*compute numerator*/
			value1=value1+g_u[i];  /*compute denominator*/
		}
```

```
            U_k=value/value1;
            return U_k;
        }

        public double computeP(double[] g_m){
            double value=0.0;
            double P=0.0;
            for(int i=0;i<N;i++)
                value=value+g_m[i];

            P=value/(double)N;
            return P;
        }
}
```

VITA

Billy D'Angelo Gaston

Candidate for the Degree of

Doctor of Philosophy

Thesis: CONSTRUCTING FEATURES AND PSEUDO-INTERSECTIONS TO MAP UNRELIABLE DOMAIN SPECIFIC DATA ITEMS FOUND IN DISJOINT SETS

Major Field: Computer Science

Biographical:

Personal Data: Born in Montgomery, Alabama on July 7, 1976, the son of Mr. Billy Gaston and B. Marie McDaniel

Education: Graduated from Hanau American High School, Hanau, Germany, 1994; received Bachelor of Science in Computer Science/Mathematics from Langston University, Langston, Oklahoma in May 1998. Received the Master of Science in Computer Science from Oklahoma State University in May 2001. Completed the requirements for the Doctor of Philosophy in Computer Science at Oklahoma State University in May 2007.

Experience: *Research Assistant*, Langston University, Department of Mathematics/Physics 1995 to 1997. *Teaching Assistant*, Langston University, Department of Computer Science 1997 to 1998. *Head Teaching Assistant*, Oklahoma State University, Department of Computer Science 1998 to 1999. *Research Associate*, Oklahoma State University/Tinker Air Force Base 1999 to 2005. *Adjunct Instructor*, Langston University, Department of Mathematics 2004. *Senior Software Engineer*, Anautics, Inc. 2005 to 2007. *President/CEO*, Anautics, Inc. 2007 to Present.