5-2011

Clustering digital ink content to assist with the grading of student work.

Jared Hatfield
University of Louisville

CLUSTERING DIGITAL INK CONTENT TO ASSIST WITH THE GRADING OF STUDENT WORK

By

Jared Hatfield
B.S., University of Louisville, 2010

A Thesis
Submitted to the Faculty of the
University of Louisville
J. B. Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Computer Engineering and Computer Science

May 2011

CLUSTERING DIGITAL INK CONTENT TO ASSIST WITH THE GRADING OF STUDENT WORK

Submitted by: _____
Jared Hatfield

A Thesis Approved on

April 21 2011
(Date)

by the following Thesis Committee

_____
Dr. Ahmed Desoky, ~~Co-~~Director

_____
Dr. Jeffery Hieb, Co-Director

_____
Dr. Ibrahim Imam

_____
Dr. Patricia Ralston

ACKNOWLEDGEMENTS

I would like to express my gratitude and thanks to my co-directors, Dr. Jeffery Hieb and Dr. Ahmed Desoky, for their guidance and support. I am also grateful to my other members of my thesis committee: Dr. Ibrahim Imam and Dr. Patricia Ralston. My work was inspired by the Engineering Fundamentals Department. Their continued support by providing an outlet for my work is greatly appreciated. I would be amiss if I did not extend my gratitude to the hundreds of calculus students in Dr. Tyler's and Dr. Ralston's ENGR 101 class that originally motivated me to design a system for automatically processing student work that was collected using DyKnow. The seeming endless amount of in-class problems they produced provided the necessary motivation to create the software to process their work and this eventually led me to where I am today. As I continued to explore what was possible for automated analysis of student work using DyKnow Vision, Dr. James Lewis and Dr. Jeffery Hieb were eager to point me in a new direction whenever I was struggling.

ABSTRACT

In recent years the use of Tablet PCs in education has received a great deal of attention. Improved note-taking and lecture presentation have been the main focus of Tablet PCs in education. Currently, grading of student work collected as digital ink using Tablet PCs is still done mostly as it would be done if collected as pencil on paper. However, having content stored as digital ink provides an opportunity to perform analysis that is neither practical nor possible with pen and paper student content. Once student work is collected using DyKnow Vision, an interactive Tablet PC based classroom software tool, the student names and specific answers to questions can be extracted for analysis. One type of analysis is clustering of students' answers. For short answer English words, clustering of answers can be automated using handwriting recognition algorithms, existing clustering techniques and string distance calculations. The clustering of answers will be an automated process that forms sets, but could be supplemented with human feedback to further refine the result. A software system to implement this approach was designed and developed. Multiple string distance measures were used to implement an agglomerative clustering algorithm that brought together similar answers. Initial evaluation of this approach on short, English word, answers showed that student answers can be clustered in such a way that produces useful results for a human grader. The algorithm has been found to be useful at creating groups of answers which a grader would consider to be identical with the most logical merges occurring early. However, the heuristic used by the algorithm has been found to both stop grouping similar answers too early in some cases and too late in others. Further refinement of this heuristic is needed to produce ideal clusters. While human processing is still required, more development in this area and the use of more advanced techniques would be highly valuable as technology becomes more tightly integrated in the classroom.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Background

As technology has become more integrated in the classroom setting, many different platforms have been adopted by educators in attempts to improve learning outcomes. The use of Microsoft based Tablet PCs in classrooms in combination with software specifically designed for classroom presentation has been adopted by several institutions including the J.B. Speed School of Engineering at the University of Louisville. A digital classroom unlocks new possibilities and opportunities that are not possible in a traditional pencil and paper environment. While typed digital work has been used in education and submitted in printed or electronic forms for many years, the new trend is towards electronic content that remains in digital form from student to instructor and back to the student.

The amount of data being generated by individuals and collected by businesses has been increasing exponentially as the digital frontier continues to expand. This enormous amount of data has proven to be impossible to analyze by hand and a wide range of algorithms and approaches have been developed to tackle the problem of analyzing and understanding this mountain of data. The software developed for collecting student work in an instructional environment has a variety of possible forms. Highly structured input such as responding to a true / false question or multiple choice questions can allow for automated and instantaneous grading and analysis along with immediate student feedback. Less structured forms of input such as text boxes tend to require human interpretation, and feedback is typically not instantaneous unless exact phrases are expected. Hand written input captured through an active

digitizer is the least structured, but is much more flexible in the type of responses that can be accepted.

The ability to collect, digital, hand written student work is well established; making it possible to begin investigating the opportunities afforded by the capture of digital ink applied by students to their Tablet PCs. Early digital ink software systems applications simply mirror the well-known processes and workflows that teachers and students are familiar with. It is only after the technology is adopted by a critical mass that the technology can break free of the artificial limitations and conventions imposed by its creators. Pen based technology in the classroom has reached this threshold and needs to move beyond mimicking analog processes and embrace new possibilities or step aside for a new technology to takes its place.

The application of data analysis to grading student work in digital ink form is an example of moving beyond mimicking analog processes. While some benefits are gained from adopting a digital ink based approach, such as near instantaneous transmission and unlimited copying, the real benefits have yet to be realized. The depth of information gathered, such as deleted pen strokes, is often ignored or simply too much information for the average grader to process. The field of computer science has produce a staggering number of algorithms and methods for processing digital data, by focusing these techniques and developing innovative user interfaces, the true benefits of a digital classroom can be realized. The approach for clustering student answers described here is one simple example on how this approach can be implemented and used.

The potential benefits of this specific implementation are less important than the conceptual approach it demonstrates: computational analysis of digital ink content used to assist

in the grading process. More advanced techniques such as machine learning, neural networks, and natural language processing could be applied to this domain and broaden the types of student work to which it could be applied. While not focused directly on student learning outcomes, it is possible and expected that the secondary effects of implementing such a system would be profound.

1.2 Organization of Thesis

Chapter two presents a literature review of the current state of Tablet PCs in education along with the important concepts of string distance and clustering used in the implementation of the clustering algorithm described in this paper. Chapter three discusses the possible applications for clustering ink based student work in addition to how existing handwriting recognition algorithms can provide a starting point for this work. Chapter four discusses the specific algorithm used for clustering and provides detailed analysis of applications results on various data sets. Conclusions and directions for future research are presented in Chapter five.

CHAPTER II

LITERATURE REVIEW

With Tablet PCs and the connected digital-classroom becoming more common in instructional settings, new educational practices and opportunities are arising. The current state of Tablet PC use in educational settings is discussed in Section 2.1 along with the lack of software tools for analyzing ink based student work. Clustering algorithms that might be applicable to analyzing student work are discussed in Sections 2.2. One domain of digital ink to which clustering could be applied is short English words. String distance algorithms that can be used as distance measures for clustering digital ink answers that represent short answers (interpreted as strings) are discussed in Section 2.3.

2.1 Use of Tablet PCs in Education

Windows based Tablet PCs have been used in the classroom setting as part of an attempt to improve the education of students. This technology utilizes pen based input to provide a more natural interface for both the instructor and the student compared to a standard keyboard and mouse. This has been shown to be very effective in both math and science classes because of the limitations of a standard keyboard. The use of this technology, even in a large classroom environment, can have a positive impact on learning outcomes. [1] The benefits of transitioning away from traditional paper based systems for collecting student work for in-class assignments include decreasing the amount of paper, decreasing the amount of time required to collect student work, and decreasing the amount of time to process student work. [2]

The focus of software development in this field has been centered on interactions that occur in the classroom. Software such as DyKnow Vision [3] and Classroom Presenter [4]

provide a platform for classroom presentations and interactivity. Both software suites provide a mechanism for wirelessly collecting student work in the instructional setting. Providing an open source foundation, Classroom Presenter has been expanded to further automate the collection of student work. [1]

At the Speed School of Engineering in the fall of 2008, the decision was made by the Engineering Fundamentals Department to transition the collection of in-class student work from paper to digital by utilizing the capabilities of DyKnow Vision. The task of compiling the weekly grade reports based off of the electronically collected submissions was given to Hatfield. While DyKnow provided a mechanism for collecting the student work, the facility to generate grade reports was absent from the software. The end result of the work was the development of DPX Manager, a software tool that produces grade reports based on the panels collected from students. [2]

For many schools, such as Grove City College, DyKnow was the preferred choice for interactive learning software, but the feedback from students was mixed with 20% of students having a negative opinion of the software's use in the classroom. [5] However, the use of interactive learning software, under controlled conditions, was shown to provide a significant improvement in student test scores in an undergraduate computer science course at MIT. [6] Even in the face of mixed feedback from students, many school administrators have recognized the potential benefits of Tablet PC deployments and have continued to experiment with the technology.

The unique benefits offered by Tablet PCs provide opportunities for the classroom that are not possible in a traditional classroom. The Virginia Tech College of Engineering, in an

attempt to "better facilitate pedagogical practices," starting in 2006, required all incoming freshmen to purchase a Tablet PC, replacing the previous laptop requirement. [7] Similarly, the J.B. Speed School of Engineering began requiring all incoming freshmen to purchase a Tablet PC starting in the fall of 2007. Speed's deployment focused on integrating the technology into the engineering analysis and mathematics classes and used multiple software packages including DyKnow. [8] These large scale Tablet PC deployments support the need for additional software development in the field of classroom management and automation that utilize the unique features of Tablet PCs.

While DyKnow is a proprietary piece of software and cannot be modified, it is possible to use the files that contain student work collected using DyKnow. Through the use of additional software, the content of these files, and therefore the student work itself, can be analyzed using custom tools. The ability to collect student work using DyKnow, combined with the use additional software, provides the opportunity to automate tasks such as grading and organizing student work. [9] It is possible, through the continuation of earlier work, to further improve the analysis techniques that are being used on the content of files that can be opened using the open source DPX Reader library. [10]

2.2 Clustering Algorithms: K-Means and Neighbor-joining

There are a variety of clustering algorithms that exist for automatically discovering patterns that exist in data sets. K-means clustering is a popular technique that divides a data set into a desired number of clusters by dividing the data into groups using the average value of subsets of the data. It has been found that a bisecting k-means approach can yield better results than an agglomerative approach that combines clusters. The difference in approaches can result

in different outcomes that are more or less desirable depending on the data set that is being manipulated. [11]

The k-means clustering algorithm is based on the concept that values from each cluster would be grouped together into a predetermined number of groups. This algorithm requires that the number of groups be known in advance and then associates every data point with the nearest mean. To begin, the initial data set is arbitrarily divided into the number of desired groups. The algorithm recalculates the group mean and every data point re-associates with the nearest mean for each group. After running for several iterations, the groups will converge and the data points will be divided into the desired number of groups.

A common approach used in the reconstruction of polygenetic trees is neighbor-joining. This approach produces an un-rooted tree that connects all of the nodes while dividing the structure of the tree based on similarity. A major difference in this approach as compared to k-means is that each node does not have an absolute value; rather the distance between the nodes is used to perform the clustering. [12]

The neighbor-joining algorithm is useful when comparing the genomes of different species where a common ancestor can be identified. This algorithm produces a tree that identifies which species or nodes are more closely related and then continues with the comparison to the other nodes. This algorithm is implemented using a matrix based comparison that iterates through the data set identifying the groups that are most closely related. While this algorithm can be applied to English words, the behavior of the algorithm is to first associate the node that has the largest distance value with the closest node since the intrinsic assumption that a common ancestor is made.

2.3 String Distance Algorithms

Many algorithms have been developed for the purposes of processing text and documents. For the purposes of correcting typed words, there are a handful of mistakes that humans are prone to making. Characters may be inserted, deleted, substituted, or transposed during the construction of the word. The number of operations required to transform one string into another denotes the Damerau-Levenshtein distance which provides a measure of similarity. [13] This algorithm depends on the tendencies of humans to make the same type of mistakes which can be easily identified when comparing text to a known dictionary. Microsoft's handwriting recognition engine utilizes a dictionary to assist in the handwriting recognition process to increase the overall accuracy. [14] Therefore the output string from handwriting recognition algorithms may have been refined based on the available dictionary.

The distance of two strings is an integer value that is the minimum number of transformations required to transform one string into another. For simple cases such as comparing "ran" and "run" it is obvious that a single substitution, changing the "a" into a "u" is the one substation required and the string distance is only one. Deletions, substitutions, and transpositions are also considered by the algorithm as it finds the minimum number of operations to transform one string into another. The greater the Demarau-Levenshtein is between two strings the more likely these strings are not related.

The longest common subsequence is a well-established problem in computer science and is the foundation for commonly used technologies such as diff. Several solutions to this problem exist and attempt to reduce the complexity of finding the longest subsequence as part of the comparison of long character sequences. [15] The subsequence problem is distinct from the longest substring problem. With a subsequence it is possible to have characters inserted between

other characters. This approach has a number of applications in the domain of DNA processing. However, the longest common subsequence can also be applied to short phrases to provide a measure of their similarity.

CHAPTER III

DESIGN

The workflows used by instructors to grade student work collected using digital ink have been constructed to mimic the well-established workflows of pencil and paper grading. Three different categories of responses are described in considering the possibility of clustering student responses: free form sketches, mathematical expressions, and handwritten short answers. Sketches provide an unstructured input that provides a difficult problem for clustering similar shapes and figures. Mathematical equations are more structured and technologies are emerging that support the conversion of digital ink mathematical expressions to standard representations such as MathML [16], but these algorithms are still being developed. For hand-writing short answers, handwriting recognition algorithms provide a mature technology that dramatically simplifies the problem of clustering short answers by providing well established algorithms for transforming the digital ink content into strings. Given the existence of mature handwriting recognition algorithms, student responses in the form of short phrases provide a problem domain suitable for further analysis.

Section 3.1 discusses the existing workflows used by instructors and the artificial limitations placed on electronic software packages for processing student work. Section 3.2 discusses the specifics for collecting student work using DyKnow Vision. The various domains of ink based content there were identified are explored in depth in Section 3.3. Lastly, Section 3.4 discusses the application of existing handwriting recognition technologies.

3.1 Optimizing Instructor Workflows Outside of the Classroom

In a traditional paper-based classroom setting, student work is collected on sheets of paper that may or may not have questions and regions for answers printed in advanced. In a paperless classroom that uses Tablet PCs, student work collected in a digital form provides a significant amount of flexibility in how the questions are delivered, the student work is collected, and how the instructor processes student responses. Additionally, there are a number of file formats and software packages that can be used for transmitting information back and forth from instructor and student. Once such classroom software package is DyKnow Vision discussed in Section 2.1. Other software packages such as Classroom Presenter and Microsoft Interactive Classroom provide alternative interactive classroom learning systems that vary slightly in the features they offer to instructors and students and the file format in which collected work is stored. Previous work on processing DyKnow files, specifically the DPX Reader library [9] discussed in Section 2.1, makes DyKnow the obvious choice for this investigation, but the approach described here can be extended and applied to these other systems within the constraints and limitations of each system.

DyKnow Vision is a proprietary software package that provides a robust Interactive Learning System that allows synchronized whiteboard space for instructors and students. An important feature of the software is that it allows instructors to collect student work. This collected work can then be viewed, modified, and eventually returned to the students using only the DyKnow Vision software. However, in the present release of DyKnow this remains a manual and time consuming process, even more so for large classes. By strictly imitating the workflows that were well established in the pre-digital classroom, several opportunities for possible automation are missed and artificial limitations are imposed.

The opportunity for automation, specifically in large classes, has not been the focus for software developed for instructional purposes. As discussed in section 2.1, software development has focused on tools that increase instructor and student interaction with the specific goals of improving the educational experience. Software that focuses primarily on workflows and instructor efficiency outside of the classroom is less obviously connected to student learning outcomes, but still offer potential benefits and possibly measurable improvements. Tools that assist in the grading process can provide more equitable grading by: reducing the turnaround time on grading assignments, providing the instructor with more time to focus lesson plans, reducing the variability of grades between students with similar or identical mistakes, preventing grader biases by not revealing student names during the grading process, and many other benefits.

3.2 Electronically Collecting Student Responses using DyKnow Vision

In DyKnow, all content is contained within DyKnow notebooks and student responses are collected or submitted as individual panels which are collectively stored in a DyKnow notebook. Individual panels are the digital equivalent of a piece of paper and along with digital ink is defined by a bounded area where the instructor and student are capable of creating content. While it is possible to have typed content, from students or instructors, on panels; this discussion will ignore this possibility for the purpose of focusing on ink based content. The standard layout assumed for the panel is a question and response type structure as show in Figure 3.1. The dark shaded region represents an *Answer Box*[1] where the student is prompted to provide a response.

[1] An *Answer Box* is a region on a DyKnow Panel that is depicted by a grey region on the panel. While this region appears to just be a simple shaded region, the file format sores the information about an *Answer Box* in a unique format that allows the area to be identified and therefore the ink content contained within that region.

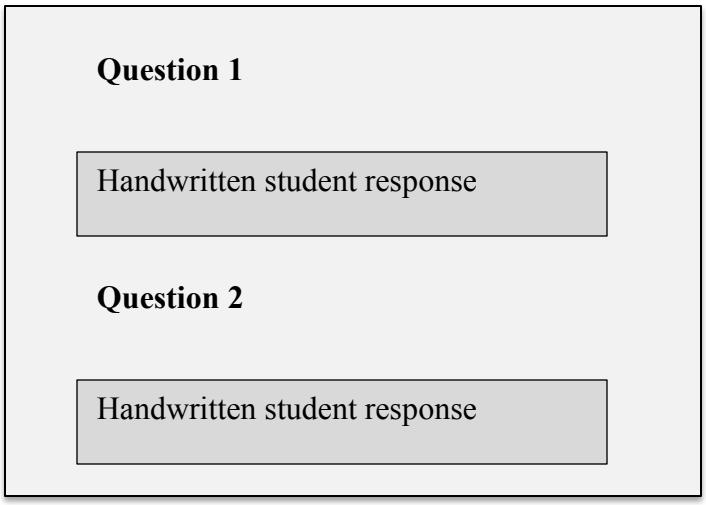


Figure 3.1 Panel Layout with *Answer Box*

The digital ink contained within each *Answer Box* can be isolated and extracted from the panel as an independent component. This isolated ink can then be displayed and analyzed independent of the other ink content on the panel. Since the ink has been isolated from the other content that is present on the panel, it can then be analyzed using established data analysis algorithms.

3.3 Applications for Clustering Digital Ink Student Responses

The process of grading student work can be a time consuming process. The pencil and paper approach is limited by the physical constraints of the medium. Student work collected using a Tablet PC and digital ink does not suffer the same limitations of paper, but current practice often imposes these constraints artificially. The digital nature of the content allows for infinite copies and unlocks an existing set of algorithms that can be used to analyze this content. However, the full range of data analysis techniques available in the field of computer science has not been applied to solve this problem.

Student work collected on a Tablet PC also provides more information than work written on an ordinary piece of paper. This information exists in several obvious and a few non-obvious forms. A subtle bit of information that is collected is the pressure of the pen to the screen as each individual stroke is made. While this information may not directly provide useful information, it demonstrates that with the transition to Tablet PCs to capture content, substantially more information is gathered. A piece of paper provides a static snapshot of the student work when it was finally submitted. A Tablet PC is capable of capturing not only the order in which pen strokes were made, but even the pen strokes that were deleted. While not all ink enabled Tablet PC software supports a history, this feature is part of DyKnow Vision.

This substantial amount of information contained in the ink content stored as part of a student response can be meticulously analyzed by hand, but this task quickly becomes unwieldy as the number of students and amount of work from each student increases. Since this content originates in a digital form, it is an ideal candidate for automated analysis. One type of analysis would be to cluster students' inked answers that were collected digitally.

The most general example of clustering student work would be the analysis and clustering of similar drawings or sketches. Even young children possess the ability to identify and classify objects even though they are not identical. It is effortless for a human to identify an animal as a cat or a dog, but this process is still difficult task for a computer. Restricting the domain to sketches, content drawn on a Tablet PC, this problem is simplified, but the problem is still very complex.

The figures depicted in Figure 3.2 show two types of shapes, a smiley face and a sail boat. While these figures are not identical, it is possible to divide them into categories. Using

the shape, all of the smiley faces are placed in one cluster and all of the sail boats are placed in another cluster, as shown in Figure 3.3. Other factors such as color and the scale of the sketch may be ignored and only the shape is considered for this specific problem.

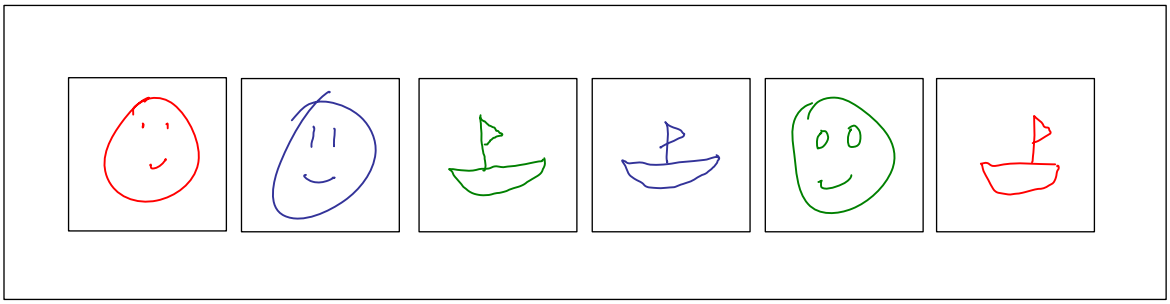


Figure 3.2 Sketch Example: Unclustered sketches

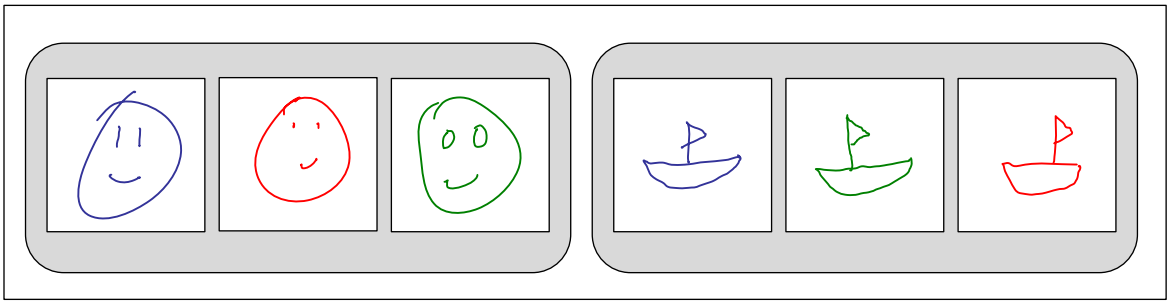


Figure 3.3 Sketch Example: Clustered sketches

A more structured application for clustering student work would be the organization of equivalent mathematical expressions. Mathematical expressions are highly structured and there are existing approaches for comparing and simplifying mathematical expressions using a computer algebra system. The novel component of this application is the input being in the form of digital ink instead of precise user input through a keyboard and mouse. Reliably converting digital ink to a mathematical expression would allow clustering to group the same or possibly similar answers with a high degree of accuracy. Microsoft's Math Input Panel, which is included as part of Windows 7, provides users with a means to create mathematical expressions through

the input of digital ink. However, this technology is still in its infancy and mature libraries are not available to convert existing ink directly into a mathematical representation as part of custom applications. The existing approaches depend on a significant amount of user feedback to obtain a high level of accuracy, which eliminates any potential time gains that are sought through the automated analysis.

The hand written mathematical equations shown in Figure 3.4 depicts several possible solutions to a mathematical problem. The previous work as part of the problem is ignored and only the final answer is used for this analysis. The trivial case of grouping together the identical answers is not difficult once the ink has accurately been transformed into a mathematical representation. With the assumption of accurately transformed ink content, it is also possible to identify those expressions that are not identical, but are equivalent using valid algebraic rearrangements. Figure 3.5 shows the equivalent expressions clustered into two groups. The difficulty of implementing a system that can organize ink based mathematical expressions is the ability to accurately convert the ink content into a standardized mathematical form that can be manipulated. While existing software allows for such a conversion, they depend on a large amount of supplemental information from the user to achieve a high level of accuracy. Once the conversion from ink has been performed, it is possible to identify equivalent expressions. So long as the question was not to simplify an expression, the equivalent answers could be clustered.

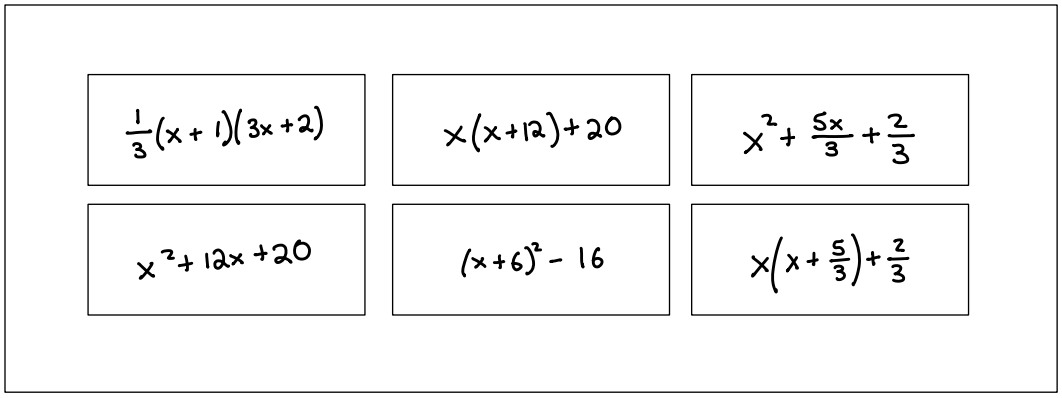


Figure 3.4 Math Example: Unclustered expressions

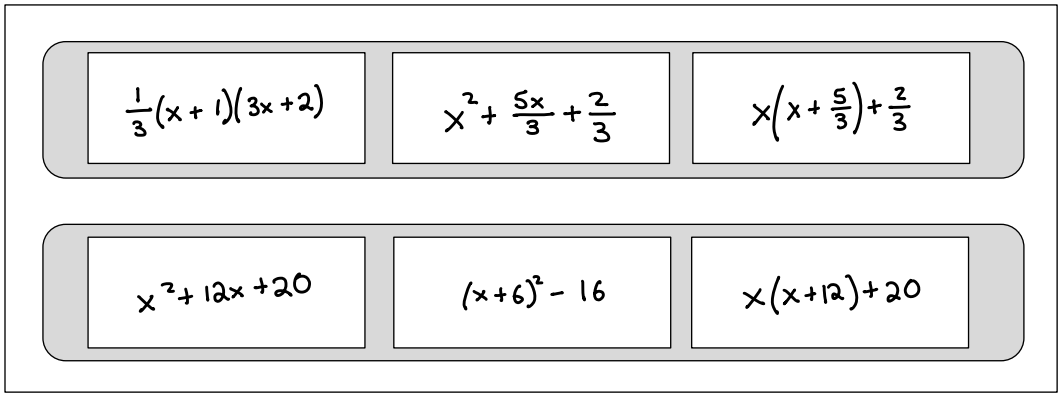


Figure 3.5 Math Example: Clustered expressions

Recognizing that the limiting factor of implementing an ink based mathematical answer clustering system is the recognition algorithm, a system that uses short English phrases as input can utilize the mature handwriting recognition algorithms that exist on Tablet PCs. Short English phrases provide a problem domain that can benefit from highly accurate handwriting recognition algorithms that can transform the student's ink into text based strings. These existing algorithms are highly accurate even with no additional user feedback into the system. Once the ink is converted to strings it can be easily compared and manipulated as part of a clustering algorithm to organize similar student responses. The difficulty of implementing this approach is finding an algorithm that can properly identify similar, but non identical, answers that the instructor would denote as equivalent.

A simple comparison can again be performed by comparing "smiley faces" and "sail boats," but this time in word form. The input values seen in Figure 3.6 can be easily divided into the two groups seen in Figure 3.7. Even though the phrases are not equivalent, each group has a root word, "smile" and "boat" respectively, that makes it possible to separate the objects into two groups even without the aid of natural language processing. A possible solution to this problem and specific test cases will be explored further in the next chapter.

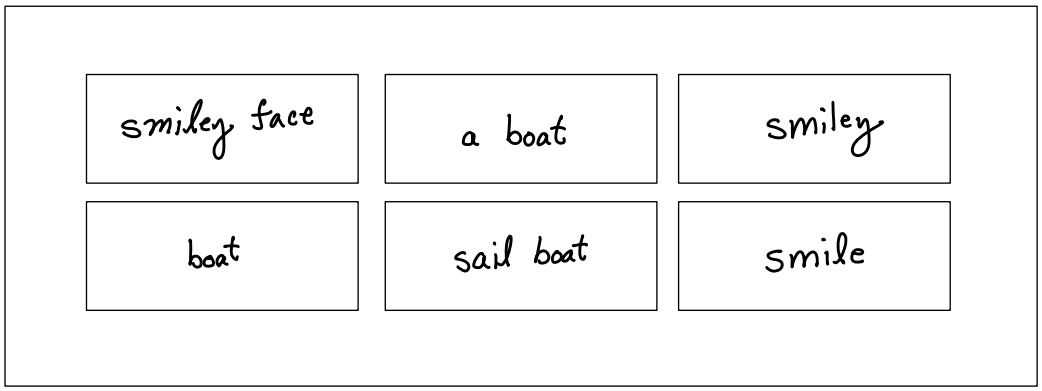


Figure 3.6 Example phrases: Unclustered phrases

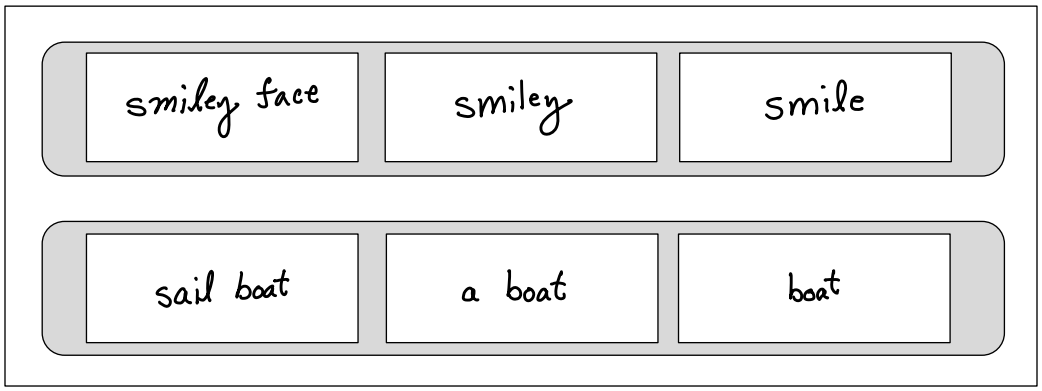


Figure 3.7 Example phrases: Clustered phrases

3.4 Handwriting Recognition

Handwriting recognition is an important component of pen based technology. However, these algorithms are not necessary for pen based technology to function. The algorithms

involved in handwriting recognition have undergone extensive development to improve the accuracy of the technology. It has been shown that on-line recognition systems provide much higher accuracy. [17] Tablet PCs provide more input than simply scanning existing text because the computer is capable of capturing information about the pressure being applied and the order of the strokes. For Windows based applications, handwriting recognition is predominately used for search, which does not require a high level of accuracy. The Microsoft handwriting recognition also provides a list of alternate strings that have been recognized. As long as the algorithm identifies the desired term as a potential candidate, it will appear in the search results. Microsoft has favored interfaces that allow the user to provide input and make corrections on the fly with an overall high level of accuracy. Since it is not possible to perform machine handwriting recognition with 100% accuracy, even in the most optimistic setting, these alternates may have the actual string that was written. The highest accuracy is achieved when the user provides input during the recognition process. [14] However, when it is assumed that we will be clustering similar strings small mistakes in the recognition algorithm will not result in major problems.

CHAPTER IV

IMPLEMENTATION

This chapter describes the implementation of a tool that applies the clustering approach described in Chapter 3 to short handwritten English word answers. To simplify the process of collecting student answers, DyKnow Vision was used as a foundation to collect and store the ink based content. The retrieved files can then be analyzed and the content of each *Answer Box* can be extracted. This ink content is then analyzed using Microsoft's handwriting recognition algorithm and the resulting strings can be clustered. This clustering is based on established string distance algorithms and the desired result is to organize the responses into groups that would be useful during the grading process. Section 4.1 describes the software used to open and process the DyKnow files and extract the content from the *Answer Boxes* while section 4.2 provides a detailed description of the clustering algorithm used to group similar answers. A detailed mathematical example of the clustering algorithm is presented in section 4.3. Known interesting failures cases are presented in section 4.4 and an analysis of clustering answers collected from students is presented in section 4.5.

4.1 A System for Processing Student Responses

The first task in investigating clustering of student responses is to get the ink associated with each student's response into a data structure appropriate for analysis. While the DyKnow client provides a robust mechanism for collecting student work in the classroom, there is no official mechanism to open a DyKnow file using anything other than the original application used to create the content. Therefore, exporting the data directly from DyKnow was not an option. Since the DyKnow client application is built on Microsoft's .NET platform and the same

developer tools that were used to create the DyKnow client are widely available it was possible to write custom code capable of extracting the desired ink stroke data. A DyKnow file is simply a compressed XML serialized object that has been stored to a file. Knowing this, it is possible to work backwards and create the necessary objects to de-serialize a DyKnow file, without access to the original executable or source code that was used to create it. With this knowledge and through a process of trial and error along with extensive testing, a C# library called DPXReader was created that is able to read a DyKnow file into memory.

The next step in the process is taking the content of the DyKnow file and rendering each panel on an InkCanvas[2]. This process is complicated by the fact that DyKnow stores a complete history of all changes made to a file. However, the DPXReader library is able to render the ink content of a DyKnow file with high accuracy assuming the panel history has been removed using the DyKnow client. It is also possible to display the *Answer Boxes* and extract the exact region on the panel. With the ability to render the ink content and the ability to identify the region that is designated to be an *Answer Box*, it is trivial to extract only the ink for each answer.

The application responsible for processing student responses contained within *Answer Boxes* is called DPX Answers [10] and is shown in Figure 4.1. At the core this application is able to open a DyKnow file and displays its contents. The content of each *Answer Box* is extracted and handwriting recognition is performed. Since this is a very computationally intensive process, this task is performed using a queue that has multiple workers allowing for increased performance in computers with multiple cores. The handwriting recognition must be performed on every *Answer Box* in the file so the more panels and the more boxes per panel, the

[2] An InkCanvas is the graphical interface that is part of the Windows operating system that is capable of displaying and receiving digital ink based content.

longer this process will take. The time required to render each panel and perform handwriting recognition on each *Answer Box* is negligible and is linearly related to the total number of panels and *Answer Boxes* that make up a DyKnow notebook.

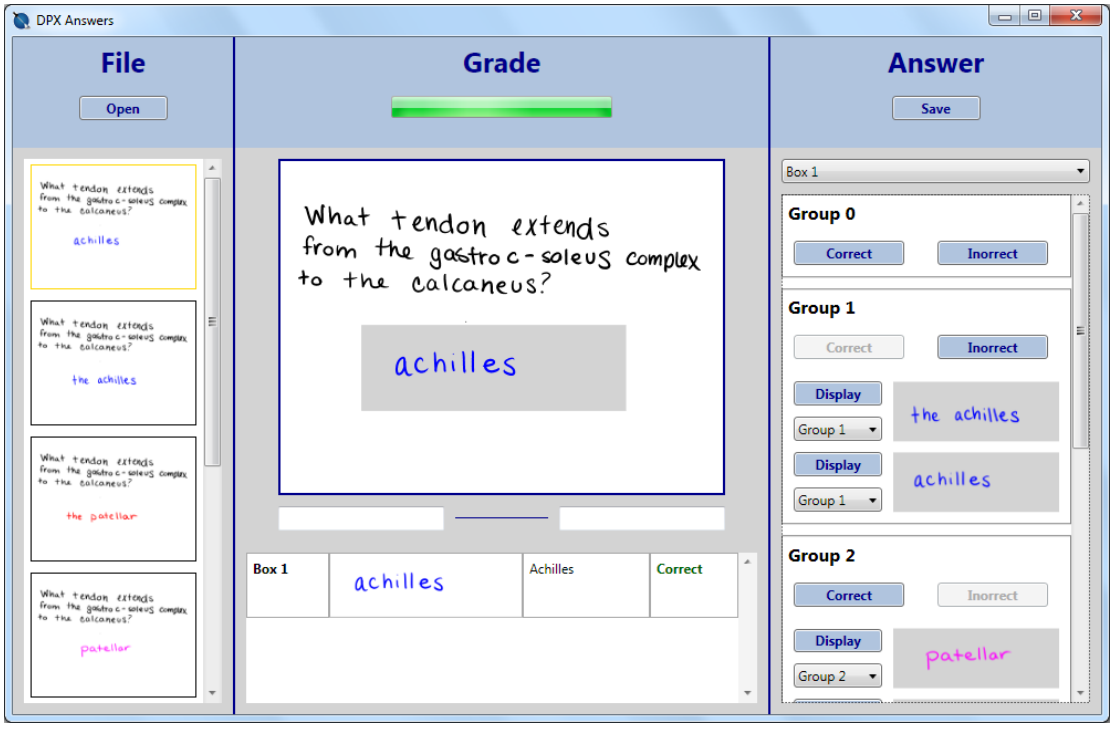


Figure 4.1 DPX Answers GUI

The input from the user is centered on each *Answer Box*. We can assume all *Answer Boxes* that occupy the same region contain inked answers for the same question. This is a valid assumption since instructors would not put multiple overlapping *Answer Boxes* on the same panel. The extracted answers can be placed into a various groups so clustering can be performed on the content. It is also possible for the user to manually manipulate the groups that result after the clustering. This provides a means to correct any mistakes that may have resulted. The user is then able to indicate if each answer is correct or incorrect and that value will be assigned to every answer within the group.

4.2 Analysis of Student Answers

Handwriting recognition provides a mechanism to convert the ink-based content into a text-based string. As part of the process of organizing student answers into groups, the handwriting recognition provided by Windows is used to recognize short answers written into the *Answer Box*. The simplifying assumption is made that the answer is a short text based string, a portion of which has a high likelihood of being repeated by many students. While it would be possible to extend grouping to complex mathematical formula based ink content or even hand drawn sketches, this is beyond the scope of the approach employed in this context. The possibilities of such an approach are discussed in chapter 5.

The process of truly understanding the answers provided by students and accurately organizing them into groups is trivial for a human, but very difficult for a computer. Linguistic processing would be required to understand answers that are different words meaning the same thing. Utilizing handwriting recognition does not eliminate the possibility for misspelled words and actually amplifies the possibility of a misrecognized word. The underlying goal of clustering similar student answers can reduce the amount of time required by a human to process the grading. By extracting the various answers this is partially accomplished. With the addition of a rudimentary clustering algorithm that can group similar answers, the amount of time required to process and grade student responses can be reduced.

Two well established string comparison algorithms can be used to measure the similarity and difference of two strings. The Damerau-Levenshtein distance algorithm measures how many deletions, insertions, substitutions, and transpositions of characters are required to

transform one string into another, providing a measure of how different two strings are from one another. In contrast, the longest common subsequence algorithm provides a measure of how similar two strings are to one another. While there are other established algorithms for comparing strings, these two algorithms in particular captured the distance and similarity measures that targeted the desired string mutations. An additional comparison can be performed that computes the number of common string tokens, or words, that two strings share in common. Since words can be arranged in varying orders and extraneous words can be added to answers, this measure provides a balanced comparison of the difference between two strings.

To perform the clustering of the various answers, an agglomeration approach is used to propose possible merges of the various groups of answers and then merge the two groups with the lowest cost. The cost measure calculated for each group is a numerical representation of the amount of variance between all answers that are contained within a single group. The cost calculation is performed through a collection of string comparisons and a formula that produces the overall cost of each proposed merge. A threshold is then calculated and the lowest cost merge that falls below the threshold is performed and the process is repeated until no merges fall below the threshold or a single merge is proposed. The goal of each merge is to reduce the overall variance of the system, however each iteration recalculates all of the group variances and threshold used to identify the best possible merge.

A total of five different measures are used in combination to calculate the cost of a potential merger of two groups. Two of the input parameters are trivial to calculate, one being the total number of strings and the other being the average length of the strings in the group. The three remaining inputs are calculated by performing all possible comparisons using the list of

provided strings and each of the comparison algorithms. The complete list of inputs is defined in Table 4.1 as the component measures of internal group variability.

Table 4.1 Definitions for Measures of Variability

Cost Measure	Formula Symbol	Description
Average Tokenized String Distance	$ATSD$	The average of the tokenized string distance algorithm applied to all combination of strings
Average Damerau-Levenshtein String Distance	$ADLS$	The average of the Damerau-Levenshtein string distance algorithm applied to all combination of strings
Average Longest Common Subsequence	$ALCSS$	The average of the longest common subsequence algorithm applied to all combination of strings
Average String Length	ASL	The average length of the all of the strings
Number of Strings	NS	The number of strings in the group

Once the values are calculated for each potential group merger, an overall cost for each group can be calculated using equation I. The equation combines multiple measures that are computed from the collection of strings. This specific equation was arrived at after experimentation on synthetic data sets and is only one of many possible equations that could be used to calculate a cost. Weights were not used in the equation as a simplifying assumption. The negative sign before the ALCSS term was included because the longest common subsequence was used as a measure of similarity while the other terms were used as measures of difference. This overall cost can then be used to compute the average and standard deviation for all of the potential merges. The threshold value used to test if a merger should be performed is one standard deviation below the average. A single merger takes place for the proposed merge that has the lowest computed cost. The algorithm continues until the terminating condition is reached.

$$cost = (ATSD + ADLS - ALCSS + ASL) \times NS \quad \text{(I)}$$

The algorithm used for clustering is shown in Figure 4.2 and the code used to calculate the group variance is shown in Figure 4.3. This algorithm uses an agglomerative approach; every iteration performs the necessary calculations to identify two groups that can be merged with the minimum cost. An unabbreviated version of the source code is available in Appendix A. The implementation show below is simplified to emphasize the important components of the algorithm.

```
void Cluster()
{
       while(number of clusters > 2 and candidate count > 1)
       {
              for each combination of cluster x_i and y_j
              {
                     calculate cost of merging x_i and y_j
              }

              calculate cost threshold // average – standard deviation of costs
              count number of candidates below threshold

              if min cost of merging groups x_i and y_j < threshold
              {
                     merge groups x_i and y_j
              }
       }
}
```

Figure 4.2 Clustering algorithm pseudo code

```
GroupAnalysis(group1, group2)
{
    tokenDistance = TokenizedStringDistance.Compute(group1, group2);
    damerauLevenshtineDistance = DamerauLevenshteinDistance.Compute(group1, group2);
    longestSubstringDistance = LongestSubstring.Compute(group1, group2);
    averageLength = CombinedAverageLength(group1, group2);
    size = group1.Count + group2.Count;

    calculatedValue = (tokenDistance + damerauLevenshtineDistance –
                       longestSubstringDistance + averageLength) * size;
}
```

Figure 4.3 Pseudo code for calculating the cost of merging two groups

4.3 Example of Clustering Student Answers

To demonstrate the algorithm, the list of words shown in Figure 4.4 is used as input to the clustering algorithm. The responses show would be a possible output from the handwriting recognition algorithm performed on the appropriate hand written student answers. The expected clusters that would result from a human performing simple answer clustering are shown in Figure 4.5. For humans, in most circumstances, the question that was asked to produce a set of responses is not needed group similar answers. The result arrived in Figure 4.5 is correct because a human recognizes that the addition of an "a" to the beginning of "cat" and the addition of an "s" to the end of "dog "does not change the meaning of the word. However, even without such understanding as in the case of a computer program, it is possible to arrive at this same result.

Figure 4.4 Unclustered example phrases

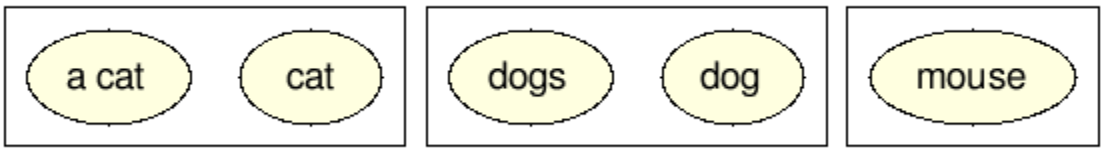


Figure 4.5 Expected clustering of example phrases

The calculations shown in Table 4.2 depict the first pass of the algorithm calculating all of the necessary input values. For the five groups listed there are $\frac{5^2-5}{2} = 10$ possible merges that need to be considered. In this first iteration of the algorithm there is only one item in each group so the average is the same as simply comparing the two items.

Table 4.2 Example calculation components: First pass

	cat					a cat					dog					dogs					mouse				
	$ATSD$	$ADLS$	$ALCSS$	ASL	NS	$ATSD$	$ADLS$	$ALCSS$	ASL	NS	$ATSD$	$ADLS$	$ALCSS$	ASL	NS	$ATSD$	$ADLS$	$ALCSS$	ASL	NS	$ATSD$	$ADLS$	$ALCSS$	ASL	NS
cat						1	3	3	4	2	2	3	0	3	2	2	3	0	3.5	2	2	4	0	4	2
a cat											3	4	0	4	2	3	4	0	4.5	2	3	5	0	5	2
dog																2	1	3	3.5	2	2	3	1	4	2
dogs																					2	2	1	4.5	2
mouse																									

Table 4.3 Example calculations: First pass

	cat	a cat	dog	dogs	mouse
cat		$(1+3-3+4)\times 2=10$	$(2+3-0+3)\times 2=16$	$(2+3-0+3.5)\times 2=17$	$(2+4-0+4)\times 2=20$
a cat			$(3+4-0+4)\times 2=22$	$(3+4-0+4.5)\times 2=23$	$(3+5-0+5)\times 2=26$
dog				$(2+1-3+3.5)\times 2=7$	$(2+3-1+4)\times 2=16$
dogs					$(2+2-1+4.5)\times 2=15$
mouse					

The cost calculation for the first iteration is show in Table 4.3. Using these calculated values we find the average to be $\mathbf{17.2}$ with a standard deviation of $\mathbf{5.52810}$. This puts the threshold at $\mathbf{11.67189}$ and the two groups that fall below the threshold are highlighted in Table 4.3. The minimum was the merger of "dog" with "dogs" having a calculated cost of 7. These two items are merged together and the algorithm is run again and the new cost calculations are show in Table 4.4. Since some possible merges have more than two items in the group, the average of all possible merges must be calculated. This computation is not shown, but can be derived from the computations found in Table 4.2. For a potential group with three elements a total of $\frac{3^2-3}{2}=3$ comparisons must be computed and then averaged.

Table 4.4 Example calculation components: Second pass

	cat					a cat					dog dogs					mouse				
	ATSD	*ADLS*	*ALCSS*	*ASL*	*NS*	*ATSD*	*ADLS*	*ALCSS*	*ASL*	*NS*	*ATSD*	*ADLS*	*ALCSS*	*ASL*	*NS*	*ATSD*	*ADLS*	*ALCSS*	*ASL*	*NS*
cat						1	3	3	4	2	2	$2.\overline{3}$	1	$3.\overline{3}$	3	2	4	0	4	2
a cat											$2.\overline{6}$	3	1	4	3	3	5	0	5	2
dog dogs																2	2	$1.\overline{6}$	4	3
mouse																				

Table 4.5 Example calculations: Second pass

	cat	a cat	dog dogs	mouse
cat		$(1+3-3+4)\times 2 = 10$	$(2+2.\overline{3}-1+3.\overline{3})\times 3 = 20$	$(2+4-0+4)\times 2 = 20$
a cat			$(2.\overline{6}+3-1+4)\times 3 = 26$	$(3+5-0+5)\times 2 = 26$
dog dogs				$(2+2-1.\overline{6}+4)\times 3 = 19$
mouse				

Using the calculated values shown in Table 4.5 we get the average to be $20.1\overline{6}$ with a standard deviation of 5.367080. This puts the threshold at 14.799585 and only one groups falls below the threshold. This means the group with "cat" and the group with "a cat" will be merged. Because only one candidate was found, the algorithm will terminate.

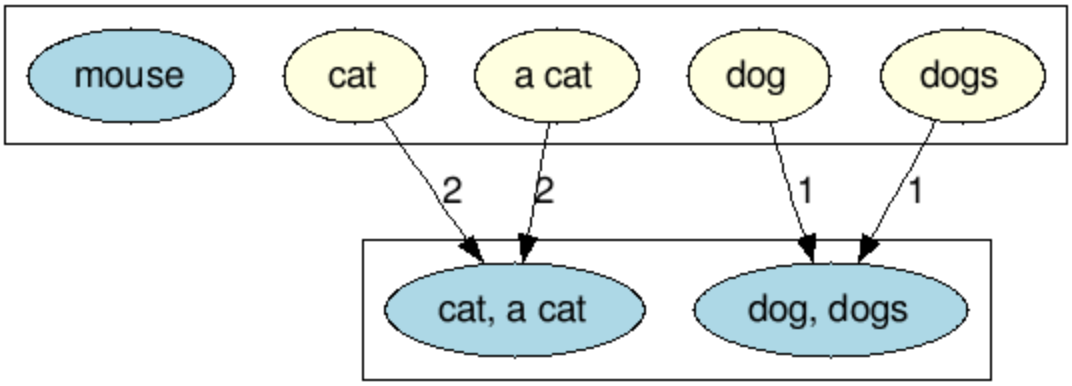


Figure 4.6 Clustering output form algorithm

The result of the algorithm is visualized in Figure 4.6, which shows the starting groups and the final groups. The original groups are located at the top of the diagram and the successive merges are indicated using number to denote the order of the merges. In this trivial example, "mouse" is both an original and final group to itself, but it is not shown grouped with the final clusters. The blue color of the group denotes that it is present when the algorithm terminates. The software generated report that contains the same calculations is found in Appendix B.

4.4 Analysis of Errors

There are several examples where the clustering approach employed by this algorithm fails to produce the desired result. The algorithm attempts to merge phrases that are similar and depending on the input, this may result in final clusters that are not the desired result. The assumption is there are groups present in the data, when there are no clusters present, it is possible for artificial groups to emerge. The example shown in Figure 4.7 performs clustering on the words one, two, three, up through ten. It is possible to recognize that there are no clusters found in this data set, but the somewhat random clusters that result are a product of the algorithm. Since these specific English words share a certain amount of similarity, and there are no merges present that actually make sense, the algorithm results in this undesirable output.

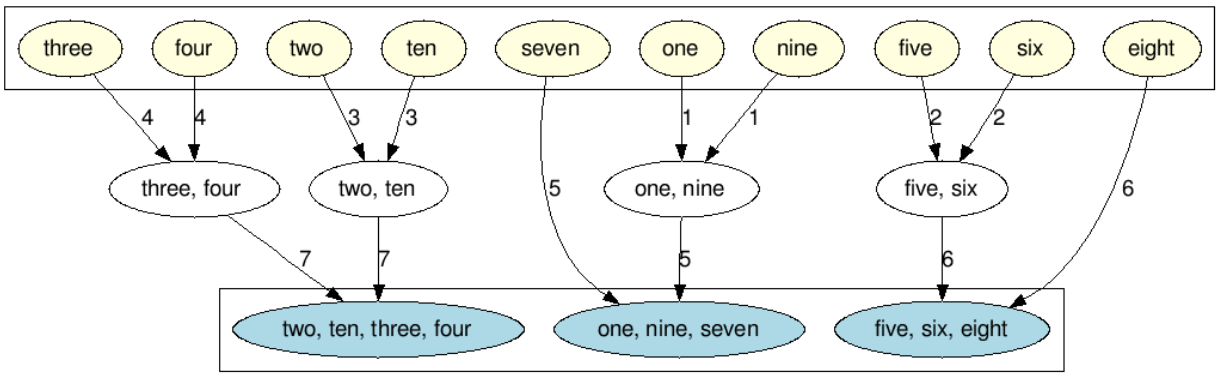


Figure 4.7 Incorrect clusters resulting from data with no pattern

The algorithm also is not able to understand synonyms. This is best demonstrated in the somewhat worst case example shown in Figure 4.8 where numerical numbers are used and the equivalent words. The measure of similarity is very high with just single character input and the first match found results in a merger. While it is obvious that the group "three" should be put together with "3" along with all of the other numbers, but this does not result. More advanced linguistic processing would be required to properly cluster this type of input. Unfortunately, the seemingly random clusters that result from the algorithm are more harmful than no clustering at all.

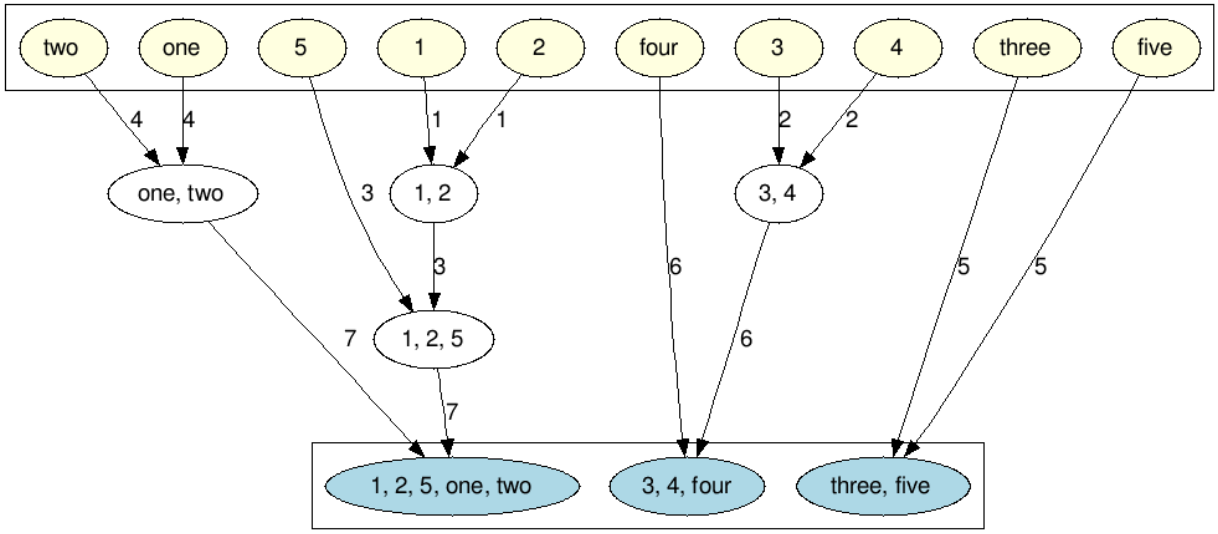


Figure 4.8 Incorrect clusters resulting from data with identifiable groups

While it is possible for the algorithm to properly cluster the input phrases, the clustering algorithm may continue to merge groups resulting in an incorrect final result. The clustering shown in Figure 4.9 has an intermediate step that contains the desired answer, but the final result is incorrect. The fourth merge is correct, but the fifth merge results in an incorrect groups. The significance here is that the incorrect merges occur after a certain point. This is not a failure to recognize the correct phrases to merge, rather a failure to recognize when the optimal result has been reached.

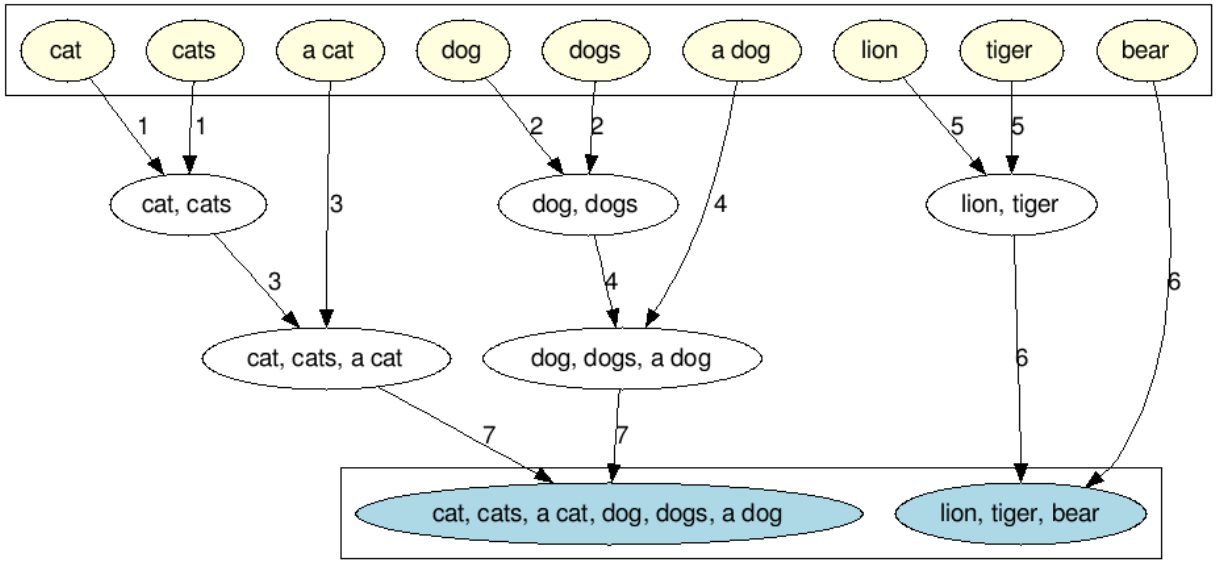


Figure 4.9 Incorrect clustering resulting from too many merges

4.5 Analysis of Student Answers

DPX Answers was deployed to analyze sample questions answered by students. The questions were answered using a provided DyKnow panel. A sample student response is shown in Figure 4.10. The panel contained four questions each with a corresponding answer box. The panel was distributed to students and collected using DyKnow Vision's retrieve panel feature. A

total of 32 students participated in the trial run providing answers to each of the four questions. The questions were selected to provide various distributions of answers to test the effectiveness of the clustering algorithm. The extracted ink for each student that was contained in an *Answer Box* as shown in the clustered results is provided in Appendix C.

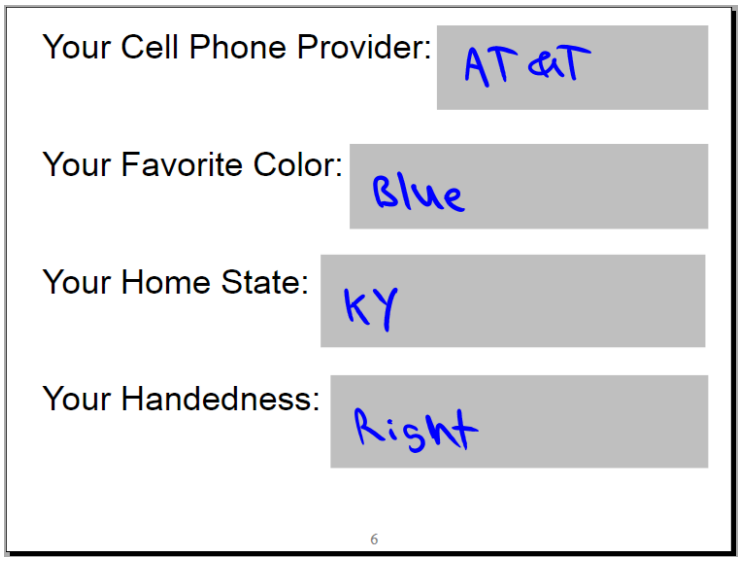


Figure 4.10 Student response collected using DyKnow

The results provided several insights into the effectiveness of the software and the clustering algorithm. The various responses to each question evoked different behaviors from the clustering algorithm. The results from the algorithm for each question are visualized in Appendix D. The original clusters along with the order of the cluster merge and final result is shown in a graph structure for the strings that were recognized from the digital ink answers.

The handwriting analysis performed on the data set provided very accurate string representations of the hand written answers. There were several occurrences where case of a letter, especially the first letter of a word, was possibly recognized incorrectly. This is not a major concern as there is definite ambiguity in the student's handwriting. The major failure in the algorithm was in the recognition of the ampersand found in the word "AT&T." The

handwriting recognition algorithm failed to recognize the entire word reverting to a string of "Other" indicating that the algorithm recognized an unknown shape. The recognition algorithm also made interesting mistakes such as recognizing "Verizon" as "venison" or "vector", "Kentucky" as "fantasy", and "KY" as "Rut." These mistakes represent only a small percentage of the overall answers and were expected due to the imperfect nature of the handwriting recognition algorithm.

The first question that asked about cell phone provider produced a large number of answer clusters. The algorithm, after identical strings were paired, resulted in a total of 17 distinct clusters. The clustering algorithm's heuristic determined that no clusters should be merged based on the original data set. This is an obvious failure of the heuristic being overly cautious since the total number of distinct answers was six. However, the results only matching on identical string put "Sprint" together three time, "AT&T" together five times, "ATT" together twice, and "Verizon" together eight times making the trivial clustering still useful.

The second question asked about favorite color and the answers included: black, blue, green, n/a, navy blue, orange, pink, purple, red, teal, and yellow. Ideally there should have been a total of 11 clusters, but the algorithm ended with only four. This is an instance where the heuristic failed to stop the algorithm soon enough. The ideal result was reached at an interim step, after three iterations, but the algorithm continued for a total of ten iterations.

The third question asked for the student's home state. Initially the identical "KY" answers produced a group with 7 responses and the identical "Kentucky" answers produced a group of 12 responses. The other responses and the merges that were performed were also reasonably accurate, although not perfect, making the results of this clustering very useful to a

possible grader. This also represents the use case where a question to a class produces two distinct groups. Since the majority of the responses fell into these groups, the amount effort required to grade is reduced and the interesting responses stand out from the crowd.

The fourth question asked about handedness. This question has two expected and distinct answers: left or right. However, this section of students only contained one left handed student so the clustering algorithm was overpowered by the number of right handed students. However, the results did produce in only a few numbers of clusters based on the very high number of identical answers. The lack variability in the answers does not provide additional insights into the clustering algorithm.

The analysis of the student answers shows that the approach employed by DPX Answers can provide useful results. The major shortcoming is in the heuristic for determining when to stop clustering. The examples presented in section 4.4 suffered from similar problems. A possible solution would be to use a more sensitive heuristic or simply prompt the user to identify the correct stopping point. Creating a user interface that combines the clustering algorithm results with human decision making would provide more accurate results. Since the individual answers require analysis for accuracy it is reasonable to prompt the user for additional feedback. The string distance algorithms deployed as a distance measure provided consistently accurate recommendations for clusters that should be merged during the first few iterations of the algorithm.

CHAPTER V

CONCLUSION AND FUTURE DIRECTIONS

This thesis presents a system for automated analysis of clustering digital ink answers based on established string distance measurements in combination with an agglomerative clustering algorithm. The approach presented here provides a means to automatically open, render, analyze, cluster, and manage groups of student answers along with producing a final grade report for all students. The clustering algorithm was tailored and optimized to work on designed sets of data that provided a representational sample of the problem statement. It has been shown that this approach can successfully group similar student responses in a number of test cases. Having shown that this workflow can be automated, the next step is to refine this automation process after gathering feedback and analyzing real data.

While DPX Answers provides the basic functionality for clustering student answers, there is still significant room for improvement. There are several directions future research could take, falling into three basic categories: expanding to other software systems, improving the clustering techniques used for short answers, or expanding into other types of input such as mathematical expressions. The software is designed to specifically read in DyKnow files. The library used to read in these files, DPX Reader, is not perfect because of the closed source nature of DyKnow. More importantly, the way the ink is rendered on the canvas is not always accurate. Moving and resizing ink on a panel from within DyKnow causes panels to not be rendered properly inside of DPX Answers. While there is a workaround for this problem by removing history, it is not a user friendly solution. An alternative direction would be to add support for reading files created

by Classroom Presenter. While this software package is open source, it provides its own set of challenges for reading and interpreting the content.

The approach used for clustering was based on string distance. While the algorithm outlined in section 4.2 has been show to produce useful clusters, a more advanced technique would be required to generalize the algorithm. Natural language processing would likely provide significantly more accurate and useful clusters, but the implementation would be significantly more complex. The most effective approach would likely be achieved by combining automated processing with human feedback. It would be almost impossible to solve the generic problem of clustering short answers in every possible knowledge domain without computational knowledge that approaches human levels.

A promising approach would be to offload the difficult decisions to the human user, but would still use the computational power of the computer to reduce the amount of effort required as part of the grading process. The complexity in this approach is primarily a human interface problem. What types of questions do we want to ask the user to insure the data is properly clustered? How do you pose the question so that you gain the maximum amount of knowledge about the data set? Can the user verify the accuracy of the clustering? While these problems are manageable under specific assumptions and circumstances, a generic solution would provide a significant challenge to implement and test.

Alternative approaches that use natural language processing, neural networks, or more complex clustering algorithms would likely provide improved results. The direction this research should take would be best directed under a supervised learning approach. The data sets used in the development of the clustering algorithm presented in this thesis were primarily

synthetic. A large, diverse, real data set collected using DyKnow Vision and contained within *Answer Boxes* would provide the best starting point for additional work. Useful statistics and knowledge could be extracted from such a data set, such as how common are identical answers, and would provide the necessary guidance in selecting the best approach. Collecting this data set would be time consuming, but the possible rewards would be great.

The problem of clustering short answers may have already been sufficiently solved by simply grouping identical answers. While it is not possible to automate the task of grouping identical answers using analog paper, the trivial implementation of the clustering used by DPX Answers has this ability. This trivial approach may be enough for many users in a variety of situations. However, different domains still provide very challenging problems.

Tablet PCs provide an attractive means of computer input for mathematics. The unrestricted input is easy for humans, but is not precise when compared to MathML. As the algorithms for transforming inked mathematical content approach the accuracy of handwritten language, the possibility of automated clustering opens up. Since this field is still in its infancy, the possibilities and potential impact are wide reaching. While the architecture of DyKnow favors student work that is analogous to quiz length assignments, the possibility for analyzing entire groups of tests and longer types of content will open up in the future. As technology continues to pervade the classroom, more advanced software solutions will be used to assist with the teaching process.

REFERENCES

[1] Koile, Kimberle and Singer, David., "Development of a Tablet-PC-based System to Increase Instructor-Student." The Impact of Pen-based Technology on Education: Vignettes, Evaluations, and Future Directions.

[2] Hatfield, Jared, Hieb, Jeffery and Lewis, James., "Scoring DyKnow Retrieved Panels for Large Classes." s.l. : Purdue University Press, 2009. The Impact of Tablet PCs and Pen-based Technology on Education: New Horizons. pp. 47-55.

[3] DyKnow., *DyKnow - Classroom Management and Interactive Education Software.* [Online] http://dyknow.com/.

[4] University of Washington., *UW Classroom Presenter.* [Online] http://classroompresenter.cs.washington.edu/.

[5] DiStasi, Vincent F., Birmingham, William P. and Welton, Gary L., "Evaluating Learning Software in the Classroom: A Continuing Study." s.l. : Purdue University Press, 2008. The Impact of Tablet PCs and Pen-based Technology on Education: Evidence and Outcomes. pp. 39-45.

[6] Koile, Kimberlie and Singer, David., "Assessing the Impact of Tablet-PC-based Classroom Interactive System." s.l. : Purdue University Press, 2008. The Impact of Tablet PCs and Pen-based Technology on Education: Evidence and Outcomes. pp. 73-80.

[7] Tront, Joseph G., "Facilitating Pedagogical Practices through a Large-Scale Tablet PC Deployment." Computer, September 2007, Issue 9, Vol. 40, pp. 62 - 68 .

[8] Hieb, Jeffery L. and Ralston, Patricia A. S., "Tablet PCs in Engineering Mathematics Courses at the J.B. Speed School of Engineering." International Journal of Mathematical Education in Science and Technology, 2010.

[9] Hatfield, Jared., "A Method for Automating the Analysis of Tablet PC Ink Based Student Work Collected Using DyKnow Vision." s.l. : Purdue University Press, 2010. The Impact of Tablet PCs and Pen-based Technology on Education: Going Mainstream. pp. 57-64.

[10] jjhatf02., dyknow-panel-extractor. *dyknow-panel-extractor - Google Project Hosting.* [Online] http://code.google.com/p/dyknow-panel-extractor/.

[11] Steinbach, Michael , Karypis, George and Kumar, Vipin., "A Comparison of Document Clustering Techniques." 2000. KDD Workshop on Text Mining.

[12] Saitou, Naruya and Nei, Masatoshi., "The neighbor-joining method: a new method for reconstructing phylogenetic trees." 1987. Molecular Biology and Evolution.

[13] Damerau, Fred J., "A technique for computer detection and correction of spelling errors." Communications of the ACM, March 1964, Issue 3, Vol. 7.

[14] Pittman, James A., "Handwriting Recognition: Tablet PC Text Input." Computer, September 2007, Issue 9, Vol. 40, pp. 49 - 54.

[15] Hirschberg, Daniel S., "Algorithms for the Longest Common Subsequence Problem." Journal of the ACM (JACM), October 1977, Issue 4, Vol. 24.

[16] Panic, Marko., "Math Handwriting Recognition in Windows 7 and Its Benefits." 2009. INTELLIGENT COMPUTER MATHEMATICS. pp. 29-30.

[17] Plamondon, R. and Srihari, S.N., "Online and off-line handwriting recognition: a comprehensive survey." IEEE Transactions on Pattern Analysis and Machine Intelligence, Jan 2000, Issue 1, Vol. 22, pp. 63 - 84.

APPENDIX A

This appendix contains the common source code used by DPX Answers and the clustering algorithm along with an implementation of the clustering algorithm and all of the associated distance calculations. The code is written in C# and was compiled using Visual Studio 2010.

```
// <copyright file="Cluster.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The Cluster.cs file.</summary>
namespace ClusterLibraryCore
{
    using System;
    using System.Collections.ObjectModel;
    using System.Linq;
    using System.Text;
    using ClusterLibraryCore.Exceptions;

    /// <summary>
    /// The class that contains all of the data that is being clustered.
    /// </summary>
    /// <typeparam name="T">The type of object that is being clustered.</typeparam>
    /// <typeparam name="L">The label that is applied to the group.</typeparam>
    public class Cluster<T, L> : ICluster<T, L>
    {
        /// <summary>
        /// The list of groups that are contained within the universe.
        /// </summary>
        private ObservableCollection<IClusterGroup<T, L>> groups;

        /// <summary>
        /// The master group for the cluster that can not be removed.
        /// </summary>
        private ClusterGroup<T, L> masterGroup;

        /// <summary>
        /// The default label used when a new group is created.
        /// </summary>
        private L defaultLabel;

        /// <summary>
        /// Initializes a new instance of the <see cref="Cluster&lt;T, L&gt;"/> class.
        /// </summary>
        /// <param name="defaultLabel">The default label.</param>
        public Cluster(L defaultLabel)
        {
            this.groups = new ObservableCollection<IClusterGroup<T, L>>();
            this.defaultLabel = defaultLabel;

            // Adds the first cluster group that is the master which can not be deleted.
            this.masterGroup = new ClusterGroup<T, L>(this.defaultLabel, false);
            this.groups.Add(this.masterGroup);
        }

        /// <summary>
        /// Gets the read only list of groups.
        /// </summary>
```

```
/// <value>The list of groups.</value>
public ReadOnlyObservableCollection<IClusterGroup<T, L>> Groups
{
    get { return new ReadOnlyObservableCollection<IClusterGroup<T, L>>(this.groups); }
}

/// <summary>
/// Adds the specified value to the master group.
/// </summary>
/// <param name="value">The value of the node.</param>
/// <returns>The node that was added.</returns>
public IClusterNode<T> AddValue(T value)
{
    ClusterNode<T> node = new ClusterNode<T>(value);
    this.masterGroup.InternalNodes.Add(node);
    return node;
}

/// <summary>
/// Adds the specified value to a group.
/// </summary>
/// <param name="group">The group to be appended.</param>
/// <param name="value">The value of the node.</param>
/// <returns>The node that was added.</returns>
public IClusterNode<T> AddValue(IClusterGroup<T, L> group, T value)
{
    if (this.groups.Contains(group))
    {
        ClusterGroup<T, L> g = group as ClusterGroup<T, L>;
        ClusterNode<T> node = new ClusterNode<T>(value);
        g.InternalNodes.Add(node);
        return node;
    }
    else
    {
        throw new InvalidClusterGroupException();
    }
}

/// <summary>
/// Adds the value dynamically to the cluster.
/// If the value is the same as all values in a group it will be added to that group.
/// If the value is unique a new group will be added.
/// </summary>
/// <param name="value">The value of the node.</param>
/// <returns>The node that was added.</returns>
public IClusterNode<T> AddValueDynamic(T value)
{
    ClusterNode<T> node = new ClusterNode<T>(value);
    bool added = false;

    // Identify the group to add
    for (int i = 0; i < this.groups.Count && !added; i++)
    {
        // Lets inspect this group to see if it works
        ClusterGroup<T, L> g = this.groups[i] as ClusterGroup<T, L>;

        // The group can't be empty
        if (g.InternalNodes.Count > 0)
        {
            // Assume the group is a match
            bool match = true;
            for (int j = 0; j < g.InternalNodes.Count; j++)
            {
                // If the group doesn't match give up
                if (!g.InternalNodes[j].Value.Equals(value))
                {
                    match = false;
                    break;
```

```
                }
            }

            // If the group is a match, use it
            if (match)
            {
                g.InternalNodes.Add(node);
                added = true;
            }
        }
    }

    if (!added)
    {
        ClusterGroup<T, L> g = this.AddGroup() as ClusterGroup<T, L>;
        g.InternalNodes.Add(node);
    }

    return node;
}

/// <summary>
/// Adds a new group to the cluster universe.
/// </summary>
/// <returns>The group that was added.</returns>
public IClusterGroup<T, L> AddGroup()
{
    L val = this.defaultLabel;
    try
    {
        val = (L)(this.defaultLabel as ICloneable).Clone();
    }
    catch
    {
        val = this.defaultLabel;
    }

    ClusterGroup<T, L> group = new ClusterGroup<T, L>(val);
    this.groups.Add(group);
    return group;
}

/// <summary>
/// Removes the specified node.
/// </summary>
/// <param name="node">The node to remove.</param>
/// <returns>
/// True if the node was successfully removed; otherwise false.
/// </returns>
public bool RemoveNode(IClusterNode<T> node)
{
    for (int i = 0; i < this.groups.Count; i++)
    {
        ClusterGroup<T, L> g = this.groups[i] as ClusterGroup<T, L>;
        ClusterNode<T> n = node as ClusterNode<T>;
        if (g.InternalNodes.Contains(n))
        {
            g.InternalNodes.Remove(n);
            return true;
        }
    }

    return false;
}

/// <summary>
/// Removes the specified group.  All of the nodes in the group will be moved to the master group.
/// </summary>
/// <param name="group">The group to remove.</param>
```

```
/// <returns>
/// True if the group was successfully removed; otherwise false.
/// </returns>
public bool RemoveGroup(IClusterGroup<T, L> group)
{
    if (this.groups.Contains(group))
    {
        ClusterGroup<T, L> g = group as ClusterGroup<T, L>;

        // Make sure that the cluster can be deleted
        if (g.IsDeletable)
        {
            // Move all of the nodes that are in this group into the master group
            while (g.InternalNodes.Count > 0)
            {
                ClusterNode<T> n = g.InternalNodes[0] as ClusterNode<T>;
                g.InternalNodes.Remove(n);
                this.masterGroup.InternalNodes.Add(n);
            }

            // Remove the group
            this.groups.Remove(group);
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        throw new InvalidClusterGroupException();
    }
}

/// <summary>
/// Moves the specified node to the specified destination.
/// </summary>
/// <param name="node">The node to move.</param>
/// <param name="destination">The destination group.</param>
/// <returns>
/// True if the move was successful; otherwise false.
/// </returns>
public bool Move(IClusterNode<T> node, IClusterGroup<T, L> destination)
{
    if (this.groups.Contains(destination))
    {
        ClusterGroup<T, L> dest = destination as ClusterGroup<T, L>;

        // Locate the source cluster
        ClusterGroup<T, L> source = null;
        for (int i = 0; i < this.groups.Count; i++)
        {
            if (this.groups[i].Nodes.Contains(node))
            {
                source = this.groups[i] as ClusterGroup<T, L>;
                break;
            }
        }

        // Test if the source pile was located
        if (source != null)
        {
            // Remove from the source and add to the destination
            source.InternalNodes.Remove(node);
            dest.InternalNodes.Add(node);
            return true;
        }
        else
```

```
                {
                    return false;
                }
            }
            else
            {
                throw new InvalidClusterGroupException();
            }
        }

        /// <summary>
        /// Merges the specified group into the destination group.  The source group will be deleted.
        /// </summary>
        /// <param name="source">The source group.</param>
        /// <param name="destination">The destination group.</param>
        /// <returns>True if the merge was successful; otherwise false.</returns>
        public bool Merge(IClusterGroup<T, L> source, IClusterGroup<T, L> destination)
        {
            Collection<IClusterNode<T>> list = new Collection<IClusterNode<T>>();
            foreach (IClusterNode<T> node in source.Nodes)
            {
                list.Add(node);
            }

            foreach (IClusterNode<T> node in list)
            {
                this.Move(node, destination);
            }

            this.RemoveGroup(source);
            (destination as ClusterGroup<T, L>).RegenerateGroupId();
            return true;
        }

        /// <summary>
        /// Gets the group that contains the specified node.
        /// </summary>
        /// <param name="value">The value of the node to locate.</param>
        /// <returns>The group containing the node.</returns>
        public IClusterGroup<T, L> GetGroup(T value)
        {
            for (int i = 0; i < this.groups.Count; i++)
            {
                ClusterGroup<T, L> g = this.groups[i] as ClusterGroup<T, L>;
                for (int j = 0; j < g.Nodes.Count; j++)
                {
                    IClusterNode<T> node = g.Nodes[j] as IClusterNode<T>;

                    // It is very important that we check to make sure the objects are the same, not that
they are equal.
                    if (Object.ReferenceEquals(value, node.Value))
                    {
                        return g;
                    }
                }
            }

            throw new InvalidClusterGroupException();
        }
    }
}



// <copyright file="ClusterGroup.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The ClusterGroup.cs file.</summary>
```

```
namespace ClusterLibraryCore
{
    using System;
    using System.Collections.ObjectModel;
    using System.ComponentModel;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The group of objects within a cluster.
    /// </summary>
    /// <typeparam name="T">The type of object that is being clustered.</typeparam>
    /// <typeparam name="L">The label that is applied to the group.</typeparam>
    internal class ClusterGroup<T, L> : IClusterGroup<T, L>
    {
        /// <summary>
        /// The index used to generate unique numbers.
        /// </summary>
        private static int index = 0;

        /// <summary>
        /// The list of objects that are within the cluster.
        /// </summary>
        private ObservableCollection<IClusterNode<T>> nodes;

        /// <summary>
        /// A flag indicating if this cluster can be deleted.
        /// </summary>
        private bool deletable;

        /// <summary>
        /// The uid for the group.
        /// </summary>
        private int uid;

        /// <summary>
        /// The label for the group.
        /// </summary>
        private L label;

        /// <summary>
        /// Initializes a new instance of the <see cref="ClusterGroup&lt;T, L&gt;"/> class.
        /// </summary>
        /// <param name="label">The label for the group.</param>
        internal ClusterGroup(L label)
        {
            this.nodes = new ObservableCollection<IClusterNode<T>>();
            this.deletable = true;
            this.uid = ClusterGroup<T, L>.index++;
            this.label = label;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="ClusterGroup&lt;T, L&gt;"/> class.
        /// </summary>
        /// <param name="label">The label for the group.</param>
        /// <param name="deletable">if set to <c>true</c> [deletable].</param>
        internal ClusterGroup(L label, bool deletable)
        {
            this.nodes = new ObservableCollection<IClusterNode<T>>();
            this.deletable = deletable;
            this.uid = ClusterGroup<T, L>.index++;
            this.Label = label;
        }

        /// <summary>
        /// Event for updating properties.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
```

```
/// <summary>
/// Gets a value indicating whether this instance is deletable.
/// </summary>
/// <value><c>true</c> if this instance is deletable; otherwise, <c>false</c>.</value>
public bool IsDeletable
{
    get { return this.deletable; }
}

/// <summary>
/// Gets the unique id.
/// </summary>
/// <value>The unique id.</value>
public int Uid
{
    get { return this.uid; }
}

/// <summary>
/// Gets or sets the grouplabel.
/// </summary>
/// <value>The group label.</value>
public L Label
{
    get
    {
        return this.label;
    }

    set
    {
        this.label = value;
        this.NotifyPropertyChanged("Label");
    }
}

/// <summary>
/// Gets the read only list of nodes.
/// </summary>
/// <value>The list fo nodesnodes.</value>
public ReadOnlyObservableCollection<IClusterNode<T>> Nodes
{
    get { return new ReadOnlyObservableCollection<IClusterNode<T>>(this.nodes); }
}

/// <summary>
/// Gets the modifible list of  nodes.
/// </summary>
/// <value>The list of nodes.</value>
internal ObservableCollection<IClusterNode<T>> InternalNodes
{
    get { return this.nodes; }
}

/// <summary>
/// Label value was updated.
/// </summary>
public void UpdateLabelValue()
{
    this.NotifyPropertyChanged("Label");
}

/// <summary>
/// Regenerates the group id.
/// </summary>
internal void RegenerateGroupId()
{
    this.uid = ClusterGroup<T, L>.index++;
```

```
        }

        /// <summary>
        /// Clusters the group label property changed.
        /// </summary>
        /// <param name="sender">The sender.</param>
        /// <param name="e">The <see cref="System.ComponentModel.PropertyChangedEventArgs"/> instance
containing the event data.</param>
        private void LabelPropertyChanged(object sender, PropertyChangedEventArgs e)
        {
            this.NotifyPropertyChanged("Label");
        }

        /// <summary>
        /// Signals that a property of this object has changed.
        /// </summary>
        /// <param name="info">The property that is being affected.</param>
        private void NotifyPropertyChanged(string info)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(info));
            }
        }
    }
}



// <copyright file="ClusterNode.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The ClusterNode.cs file.</summary>
namespace ClusterLibraryCore
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// A node that contains a value used within the cluster.
    /// </summary>
    /// <typeparam name="T">The type of object that is being clustered.</typeparam>
    internal class ClusterNode<T> : IClusterNode<T>
    {
        /// <summary>
        /// The value contained within this node.
        /// </summary>
        private T value;

        /// <summary>
        /// Initializes a new instance of the <see cref="ClusterNode&lt;T&gt;"/> class.
        /// </summary>
        /// <param name="value">The value.</param>
        internal ClusterNode(T value)
        {
            this.value = value;
        }

        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <value>The value.</value>
        public T Value
        {
            get { return this.value; }
        }
```

```
    }
}
```

```
// <copyright file="ICluster.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The ICluster.cs file.</summary>
namespace ClusterLibraryCore
{
    using System;
    using System.Collections.ObjectModel;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The interface for the possible manipulations that can be made to the cluster.
    /// </summary>
    /// <typeparam name="T">The type of object that is being clustered.</typeparam>
    /// <typeparam name="L">The label that is applied to the group.</typeparam>
    public interface ICluster<T, L>
    {
        /// <summary>
        /// Gets the read only list of groups.
        /// </summary>
        /// <value>The list of groups.</value>
        ReadOnlyObservableCollection<IClusterGroup<T, L>> Groups
        {
            get;
        }

        /// <summary>
        /// Adds the specified value to the master group.
        /// </summary>
        /// <param name="value">The value of the node.</param>
        /// <returns>The node that was added.</returns>
        IClusterNode<T> AddValue(T value);

        /// <summary>
        /// Adds the specified value to a group.
        /// </summary>
        /// <param name="group">The group to be appended.</param>
        /// <param name="value">The value of the node.</param>
        /// <returns>The node that was added.</returns>
        IClusterNode<T> AddValue(IClusterGroup<T, L> group, T value);

        /// <summary>
        /// Adds the value dynamically to the cluster.
        /// If the value is the same as all values in a group it will be added to that group.
        /// If the value is unique a new group will be added.
        /// </summary>
        /// <param name="value">The value of the node.</param>
        /// <returns>The node that was added.</returns>
        IClusterNode<T> AddValueDynamic(T value);

        /// <summary>
        /// Adds a new group to the cluster universe.
        /// </summary>
        /// <returns>The group that was added.</returns>
        IClusterGroup<T, L> AddGroup();

        /// <summary>
        /// Removes the specified node.
        /// </summary>
        /// <param name="node">The node to remove.</param>
        /// <returns>True if the node was successfully removed; otherwise false.</returns>
        bool RemoveNode(IClusterNode<T> node);
```

```
        /// <summary>
        /// Removes the specified group.  All of the nodes in the group will be moved to the master group.
        /// </summary>
        /// <param name="group">The group to remove.</param>
        /// <returns>True if the group was successfully removed; otherwise false.</returns>
        bool RemoveGroup(IClusterGroup<T, L> group);

        /// <summary>
        /// Moves the specified node to the specified destination.
        /// </summary>
        /// <param name="node">The node to move.</param>
        /// <param name="destination">The destination group.</param>
        /// <returns>True if the move was successful; otherwise false.</returns>
        bool Move(IClusterNode<T> node, IClusterGroup<T, L> destination);

        /// <summary>
        /// Merges the specified group into the destination group.  The source group will be deleted.
        /// </summary>
        /// <param name="source">The source group.</param>
        /// <param name="destination">The destination group.</param>
        /// <returns>True if the merge was successful; otherwise false.</returns>
        bool Merge(IClusterGroup<T, L> source, IClusterGroup<T, L> destination);

        /// <summary>
        /// Gets the group that contains the specified node.
        /// </summary>
        /// <param name="value">The value of the node to locate.</param>
        /// <returns>The group containing the node.</returns>
        IClusterGroup<T, L> GetGroup(T value);
    }
}



// <copyright file="IClusterGroup.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The IClusterGroup.cs file.</summary>
namespace ClusterLibraryCore
{
    using System;
    using System.Collections.ObjectModel;
    using System.ComponentModel;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The IClusterGroup file.
    /// </summary>
    /// <typeparam name="T">The type of object that is being clustered.</typeparam>
    /// <typeparam name="L">The label that is applied to the group.</typeparam>
    public interface IClusterGroup<T, L> : INotifyPropertyChanged
    {
        /// <summary>
        /// Gets a value indicating whether this instance is deletable.
        /// </summary>
        /// <value><c>true</c> if this instance is deletable; otherwise, <c>false</c>.</value>
        bool IsDeletable
        {
            get;
        }

        /// <summary>
        /// Gets the unique id.
        /// </summary>
        /// <value>The unique id.</value>
        int Uid
```

```
        {
            get;
        }

        /// <summary>
        /// Gets or sets the grouplabel.
        /// </summary>
        /// <value>The group label.</value>
        L Label
        {
            get;
            set;
        }

        /// <summary>
        /// Gets the read only list of nodes.
        /// </summary>
        /// <value>The list fo nodesnodes.</value>
        ReadOnlyObservableCollection<IClusterNode<T>> Nodes
        {
            get;
        }

        /// <summary>
        /// Label value was updated.
        /// </summary>
        void UpdateLabelValue();
    }
}



// <copyright file="IClusterNode.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The IClusterNode.cs file.</summary>
namespace ClusterLibraryCore
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The IClusterNode file.
    /// </summary>
    /// <typeparam name="T">The type of object that is being clustered.</typeparam>
    public interface IClusterNode<T>
    {
        /// <summary>
        /// Gets the value.
        /// </summary>
        /// <value>The value.</value>
        T Value
        {
            get;
        }
    }
}



// <copyright file="ClusterException.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The ClusterException.cs file.</summary>
namespace ClusterLibraryCore.Exceptions
```

```
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The base exception for the ClusterLibrary.
    /// </summary>
    public class ClusterException : Exception
    {
    }
}
```

```
// <copyright file="InvalidClusterGroupException.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The InvalidClusterGroupException.cs file.</summary>
namespace ClusterLibraryCore.Exceptions
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The execption thrown with an invalid ClusterGroup is specified.
    /// </summary>
    public class InvalidClusterGroupException : ClusterException
    {
    }
}
```

```
// <copyright file="Grade.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The Grade.cs file.</summary>
namespace GradeLibrary
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The different state for a grade.
    /// </summary>
    public enum Grade
    {
        /// <summary>
        /// The box is not valid.
        /// </summary>
        INVALID,

        /// <summary>
        /// The answer has not been graded.
        /// </summary>
        NOTSET,

        /// <summary>
        /// The answer is correct.
        /// </summary>
        CORRECT,
```

```
        /// <summary>
        /// The answer is incorrect.
        /// </summary>
        INCORRECT
    }
}
```

```
// <copyright file="GroupData.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The GroupData.cs file.</summary>
namespace GradeLibrary
{
    using System;
    using System.Collections.ObjectModel;
    using System.ComponentModel;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The meta data that is part of a group of grades.
    /// </summary>
    public class GroupData : INotifyPropertyChanged, ICloneable
    {
        /// <summary>
        /// The grade that is assigned to the group.
        /// </summary>
        private Grade grade;

        /// <summary>
        /// The list of terms that is included in the group.
        /// </summary>
        private Collection<string> includeList;

        /// <summary>
        /// The list of terms that are excluded from the group.
        /// </summary>
        private Collection<string> excludeList;

        /// <summary>
        /// Initializes a new instance of the <see cref="GroupData"/> class.
        /// </summary>
        public GroupData()
        {
            this.grade = Grade.NOTSET;
            this.includeList = new Collection<string>();
            this.excludeList = new Collection<string>();
        }

        /// <summary>
        /// Event for updating properties.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        /// <summary>
        /// Gets or sets the grade.
        /// </summary>
        /// <value>The grade.</value>
        public Grade Grade
        {
            get
            {
                return this.grade;
            }
```

```
            set
            {
                this.grade = value;
                this.NotifyPropertyChanged("Grade");
            }
        }

        /// <summary>
        /// Gets the included term list.
        /// </summary>
        /// <value>The included term list.</value>
        public Collection<string> IncludeList
        {
            get { return this.includeList; }
        }

        /// <summary>
        /// Gets the excluded term list.
        /// </summary>
        /// <value>The excluded term list.</value>
        public Collection<string> ExcludeList
        {
            get { return this.excludeList; }
        }

        /// <summary>
        /// Creates a new object that is a copy of the current instance.
        /// </summary>
        /// <returns>
        /// A new object that is a copy of this instance.
        /// </returns>
        public object Clone()
        {
            GroupData gd = new GroupData();
            gd.grade = this.grade;
            gd.includeList = new Collection<string>();
            gd.excludeList = new Collection<string>();
            for (int i = 0; i < this.includeList.Count; i++)
            {
                gd.includeList.Add(this.includeList[i]);
            }

            for (int i = 0; i < this.excludeList.Count; i++)
            {
                gd.excludeList.Add(this.excludeList[i]);
            }

            return gd;
        }

        /// <summary>
        /// Signals that a property of this object has changed.
        /// </summary>
        /// <param name="info">The property that is being affected.</param>
        private void NotifyPropertyChanged(string info)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(info));
            }
        }
    }
}



// <copyright file="IAnswer.cs" company="Jared Hatfield">
// All Rights Reserved 2011
```

```
// </copyright>
// <summary>The IAnswer.cs file.</summary>
namespace GradeLibrary
{
    using System;
    using System.Collections.ObjectModel;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The interface used to access the information about an answer.
    /// </summary>
    public interface IAnswer : ICloneable
    {
        /// <summary>
        /// Gets the answer.
        /// </summary>
        /// <value>The answer.</value>
        string Answer
        {
            get;
        }

        /// <summary>
        /// Gets the alternates.
        /// </summary>
        /// <value>The alternates.</value>
        ReadOnlyCollection<string> Alternates
        {
            get;
        }
    }
}
```

```
// <copyright file="IClusterAlgorithm.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The IClusterAlgorithm.cs file.</summary>
namespace GradeLibrary
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using ClusterLibraryCore;

    /// <summary>
    /// An algorithm for processing
    /// </summary>
    public interface IClusterAlgorithm
    {
        /// <summary>
        /// Gets or sets the cluster.
        /// </summary>
        /// <value>The cluster.</value>
        ICluster<IAnswer, GroupData> Cluster
        {
            get;
            set;
        }

        /// <summary>
        /// Processes this instance.
        /// </summary>
        void Process();
    }
```

```
}
```

```
// <copyright file="CacheManager.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The CacheManager.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.Generic;
    using System.Collections.ObjectModel;
    using System.Diagnostics;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The cache manager that is responsible for speeding up string comparison computations.
    /// </summary>
    internal class CacheManager
    {
        /// <summary>
        /// The defult number of items to store in the cache.
        /// </summary>
        private const int DefaultCacheSize = 100;

        /// <summary>
        /// The maximum value for the cache.
        /// </summary>
        private const int MaximumCacheSize = 10000;

        /// <summary>
        /// The padding used as part of the caching process.
        /// </summary>
        private const string Padding = "++-&|^|&-++";

        /// <summary>
        /// The total number of hits.
        /// </summary>
        private static int totalHits = 0;

        /// <summary>
        /// The total number of misses.
        /// </summary>
        private static int totalMisses = 0;

        /// <summary>
        /// The lock used to synchronize access to the cache.
        /// </summary>
        private object cacheLock;

        /// <summary>
        /// The number of itmes to store in the cache.
        /// </summary>
        private int cacheSize;

        /// <summary>
        /// The counter that keeps track of the number of hits on the cache.
        /// </summary>
        private int hits;

        /// <summary>
        /// The counter that keeps track of the number of misses on the cache.
        /// </summary>
        private int misses;

        /// <summary>
```

```
/// The cache that is responsible for speeding up computations.
/// </summary>
private Dictionary<string, int> cache;

/// <summary>
/// The queue used to remove items from the cache.
/// </summary>
private Queue<string> queue;

/// <summary>
/// Initializes a new instance of the <see cref="CacheManager"/> class.
/// </summary>
internal CacheManager()
{
    this.cacheLock = new object();
    this.cacheSize = CacheManager.DefaultCacheSize;
    this.hits = 0;
    this.misses = 0;
    this.cache = new Dictionary<string, int>();
    this.queue = new Queue<string>();
}

/// <summary>
/// Gets the statistics for the cache.
/// </summary>
/// <returns>The description of the statistics.</returns>
public static string Statistics()
{
    return "Hits = " + CacheManager.totalHits + "\tMisses = " + CacheManager.totalMisses;
}

/// <summary>
/// Resets the size of the cache back to the default value.
/// </summary>
internal void ResetCache()
{
    lock (this.cacheLock)
    {
        // Reset the hits and misses
        this.hits = 0;
        this.misses = 0;

        // Clear the cache
        this.cache.Clear();
        this.queue.Clear();

        // Set the size of the cache back to the default.
        this.cacheSize = CacheManager.DefaultCacheSize;
    }
}

/// <summary>
/// Sets the size of the cache.
/// </summary>
/// <param name="size">The size of the cache.</param>
internal void SetCacheSize(int size)
{
    lock (this.cacheLock)
    {
        // Set the maximum size for the cache.
        if (size > CacheManager.MaximumCacheSize)
        {
            size = CacheManager.MaximumCacheSize;
        }

        // Update the cache size
        if (size > this.cacheSize)
        {
            // We are increasing the cache size so just make the change
```

```
            this.cacheSize = size;
        }
        else
        {
            // We are decreasing the cache size so first prune the cache
            while (this.queue.Count > size)
            {
                // The cache opperates in a FIFO manner with the assistance of a queue.
                string remove = this.queue.Dequeue();
                this.cache.Remove(remove);
            }

            // Set the size of the specified value.
            this.cacheSize = size;
        }
    }
}

/// <summary>
/// Gets the specified key value.
/// </summary>
/// <param name="s">The first string.</param>
/// <param name="t">The second string.</param>
/// <returns>
/// The requested value if it was located, ortherwise null if it is not found.
/// </returns>
internal Nullable<int> Get(string s, string t)
{
    lock (this.cacheLock)
    {
        string key1 = CacheManager.GetKey(s, t);
        string key2 = CacheManager.GetKey(t, s);
        if (this.cache.Keys.Contains(key1))
        {
            int i = this.cache[key1];
            this.hits++;
            CacheManager.totalHits++;
            return i;
        }
        else if (this.cache.Keys.Contains(key2))
        {
            int i = this.cache[key2];
            this.hits++;
            CacheManager.totalHits++;
            return i;
        }
        else
        {
            this.misses++;
            CacheManager.totalMisses++;
            return null;
        }
    }
}

/// <summary>
/// Puts the specified s.
/// </summary>
/// <param name="s">The first string.</param>
/// <param name="t">The second string.</param>
/// <param name="n">The number.</param>
internal void Put(string s, string t, int n)
{
    lock (this.cacheLock)
    {
        string key = CacheManager.GetKey(s, t);
        this.cache.Add(key, n);
        this.queue.Enqueue(key);
        if (this.queue.Count > this.cacheSize)
```

```
                {
                    // The cache opperates in a FIFO manner with the assistance of a queue.
                    string remove = this.queue.Dequeue();
                    this.cache.Remove(remove);
                }
            }
        }

        /// <summary>
        /// Gets the key.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The key used for lookup.</returns>
        private static string GetKey(string s, string t)
        {
            return s + CacheManager.Padding + t;
        }
    }
}



// <copyright file="DamerauLevenshteinDistance.cs" company="Jared Hatfield">
// All Rights Reserved 2010
// </copyright>
// <summary>The DamerauLevenshteinDistance.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The Damerau Levenshtein Distance.
    /// </summary>
    internal class DamerauLevenshteinDistance : IDistance
    {
        /// <summary>
        /// The cache used for speeding up the comparisons.
        /// </summary>
        private static CacheManager cache = new CacheManager();

        /// <summary>
        /// Compute the distance between two strings.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The levenshtein distance.</returns>
        internal static int Distance(string s, string t)
        {
            // Attempt to retreive value from cache
            Nullable<int> cached = DamerauLevenshteinDistance.cache.Get(s, t);
            if (cached != null && cached.HasValue)
            {
                return cached.Value;
            }

            // Calculate the value
            int n = s.Length;
            int m = t.Length;
            int[,] d = new int[n + 1, m + 1];

            // The first string was empty so the distance is the length of the second string.
            if (n == 0)
            {
                return m;
```

```
    }

    // The second string was empty so the distance is the length of the first string.
    if (m == 0)
    {
        return n;
    }

    // Check to see if the strings are actually equal to each other
    if (s.Equals(t))
    {
        return 0;
    }

    // Fill the matrix with the starting values.
    for (int i = 0; i <= n; d[i, 0] = i++)
    {
    }

    for (int j = 0; j <= m; d[0, j] = j++)
    {
    }

    // Run the algorithm...
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            int cost = 0;
            if (i >= n && j >= m)
            {
                cost = 1;
            }
            else if (i >= n || j >= m || s[i] == t[j])
            {
                cost = 0;
            }
            else
            {
                cost = 1;
            }

            d[i, j] = Math.Min(
                Math.Min(
                    d[i - 1, j] + 1,         // deletion
                    d[i, j - 1] + 1),        // insertion
                d[i - 1, j - 1] + cost);     // substitution

            if (i > 1 && j > 1 && i < n && j < m && s[i] == t[j - 1] && s[i - 1] == t[j])
            {
                d[i, j] = Math.Min(
                    d[i, j],
                    d[i - 2, j - 2] + cost); // transposition
            }
        }
    }

    int result = d[n, m];

    // Cache the value
    DamerauLevenshteinDistance.cache.Put(s, t, result);

    return result;
}

/// <summary>
/// Computes the string distance.
/// </summary>
/// <param name="s">The first string.</param>
```

```
        /// <param name="t">The second string.</param>
        /// <returns>The string distance.</returns>
        internal override int GetDistance(string s, string t)
        {
            return DamerauLevenshteinDistance.Distance(s, t);
        }
    }
}



// <copyright file="GroupAnalysis.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The GroupAnalysis.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using ClusterLibraryCore;
    using GradeLibrary;

    /// <summary>
    /// The algorithm for analyzing groups.
    /// </summary>
    internal class GroupAnalysis
    {
        /// <summary>
        /// The analysis label;
        /// </summary>
        private string label;

        /// <summary>
        /// The token distance.
        /// </summary>
        private double tokenDistance;

        /// <summary>
        /// The damerau levenshtine distance.
        /// </summary>
        private double damerauLevenshtineDistance;

        /// <summary>
        /// The longest substring distance.
        /// </summary>
        private double longestSubstringDistance;

        /// <summary>
        /// The average length.
        /// </summary>
        private double averageLength;

        /// <summary>
        /// The size of the set.
        /// </summary>
        private int size;

        /// <summary>
        /// The calculated value.
        /// </summary>
        private double calculatedValue;

        /// <summary>
        /// Initializes a new instance of the <see cref="GroupAnalysis"/> class.
        /// </summary>
        /// <param name="label">The label.</param>
```

```
        /// <param name="g1">The first group.</param>
        /// <param name="g2">The second group.</param>
        internal GroupAnalysis(string label, IClusterGroup<IAnswer, GroupData> g1, IClusterGroup<IAnswer,
GroupData> g2)
        {
            this.label = label;
            this.tokenDistance = GroupVariance.Compute(g1, g2, new TokenizedStringDistance());
            this.damerauLevenshtineDistance        =        GroupVariance.Compute(g1,        g2,        new
DamerauLevenshteinDistance());
            this.longestSubstringDistance = GroupVariance.Compute(g1, g2, new LongestSubstring());
            this.averageLength = this.CombinedAverageLength(g1, g2);
            this.size = g1.Nodes.Count + g2.Nodes.Count;
            this.Calculate();
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="GroupAnalysis"/> class.
        /// </summary>
        /// <param name="label">The label.</param>
        /// <param name="group">The group.</param>
        internal GroupAnalysis(string label, IClusterGroup<IAnswer, GroupData> group)
        {
            this.label = label;
            this.tokenDistance = GroupVariance.Compute(group, new TokenizedStringDistance());
            this.tokenDistance = GroupVariance.Compute(group, new TokenizedStringDistance());
            this.damerauLevenshtineDistance          =          GroupVariance.Compute(group,          new
DamerauLevenshteinDistance());
            this.longestSubstringDistance = GroupVariance.Compute(group, new LongestSubstring());
            if (group.Nodes.Count == 1)
            {
                this.tokenDistance = 1;
                this.longestSubstringDistance = group.Nodes[0].Value.Answer.Length;
            }

            this.averageLength = this.CombinedAverageLength(group);
            this.size = group.Nodes.Count;
            this.Calculate();
        }

        /// <summary>
        /// Gets the lable.
        /// </summary>
        /// <value>The lable.</value>
        internal string Lable
        {
            get { return this.label; }
        }

        /// <summary>
        /// Gets the token distance.
        /// </summary>
        /// <value>The token distance.</value>
        internal double TokenDistance
        {
            get { return this.tokenDistance; }
        }

        /// <summary>
        /// Gets the damerau levenshtein distance.
        /// </summary>
        /// <value>The damerau levenshtein distance.</value>
        internal double DamerauLevenshteinDistance
        {
            get { return this.damerauLevenshtineDistance; }
        }

        /// <summary>
        /// Gets the longest substring distance.
        /// </summary>
```

```
        /// <value>The longest substring distance.</value>
        internal double LongestSubstringDistance
        {
            get { return this.longestSubstringDistance; }
        }

        /// <summary>
        /// Gets the average length.
        /// </summary>
        /// <value>The average length.</value>
        internal double AverageLength
        {
            get { return this.averageLength; }
        }

        /// <summary>
        /// Gets the size of the group.
        /// </summary>
        /// <value>The size of the group.</value>
        internal int Size
        {
            get { return this.size; }
        }

        /// <summary>
        /// Gets the calculated value.
        /// </summary>
        /// <value>The calculated value.</value>
        internal double CalculatedValue
        {
            get { return this.calculatedValue; }
        }

        /// <summary>
        /// Gets the report rows.
        /// </summary>
        /// <returns>The string containing the HTML report rows.</returns>
        internal string GetReportRows()
        {
            int round = 5;
            StringBuilder sb = new StringBuilder();
            sb.Append("<td>" + Math.Round(this.TokenDistance, round) + "</td>");
            sb.Append("<td>" + Math.Round(this.DamerauLevenshteinDistance, round) + "</td>");
            sb.Append("<td>" + Math.Round(this.LongestSubstringDistance, round) + "</td>");
            sb.Append("<td>" + Math.Round(this.AverageLength, round) + "</td>");
            sb.Append("<td>" + this.Size + "</td>");
            sb.Append("<td>" + Math.Round(this.CalculatedValue, round) + "</td>");

            return sb.ToString();
        }

        /// <summary>
        /// Calculates the value.
        /// </summary>
        private void Calculate()
        {
            // Calculate the value used for comparisons
            //    this.calculatedValue    =    this.tokenDistance    +    this.damerauLevenshtineDistance    -
this.longestSubstringDistance;
            this.calculatedValue    =    (this.tokenDistance    +    this.damerauLevenshtineDistance    -
this.longestSubstringDistance + this.averageLength) * this.size;
        }

        /// <summary>
        /// Computes the combined average length.
        /// </summary>
        /// <param name="g1">The group 1.</param>
        /// <param name="g2">The group 2.</param>
        /// <returns>The average length of a word.</returns>
```

```
        private double CombinedAverageLength(IClusterGroup<IAnswer, GroupData> g1, IClusterGroup<IAnswer,
GroupData> g2)
        {
            double total = 0;
            for (int i = 0; i < g1.Nodes.Count; i++)
            {
                total += g1.Nodes[i].Value.Answer.Length;
            }

            for (int i = 0; i < g2.Nodes.Count; i++)
            {
                total += g2.Nodes[i].Value.Answer.Length;
            }

            return total / (double)(g1.Nodes.Count + g2.Nodes.Count);
        }

        /// <summary>
        /// Computes the combined average length.
        /// </summary>
        /// <param name="group">The group.</param>
        /// <returns>The average length of a word.</returns>
        private double CombinedAverageLength(IClusterGroup<IAnswer, GroupData> group)
        {
            double total = 0;
            for (int i = 0; i < group.Nodes.Count; i++)
            {
                total += group.Nodes[i].Value.Answer.Length;
            }

            return total / (double)group.Nodes.Count;
        }
    }
}



// <copyright file="GroupVariance.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The GroupVariance.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.ObjectModel;
    using System.Linq;
    using System.Text;
    using ClusterLibraryCore;
    using GradeLibrary;

    /// <summary>
    /// Calculats the group variance for a list of answers.
    /// </summary>
    internal class GroupVariance
    {
        /// <summary>
        /// Computes the variance for two groups assuming they were merged together.
        /// </summary>
        /// <param name="a">The cluster group a.</param>
        /// <param name="b">The cluster group b.</param>
        /// <param name="distanceAlgorithm">The distance algorithm.</param>
        /// <returns>The computed variance.</returns>
        internal static double Compute(IClusterGroup<IAnswer, GroupData> a, IClusterGroup<IAnswer,
GroupData> b, IDistance distanceAlgorithm)
        {
            Collection<IAnswer> list = new Collection<IAnswer>();
            for (int i = 0; i < a.Nodes.Count; i++)
            {
```

```
                list.Add(a.Nodes[i].Value);
            }

            for (int i = 0; i < b.Nodes.Count; i++)
            {
                list.Add(b.Nodes[i].Value);
            }

            return GroupVariance.Compute(list, distanceAlgorithm);
        }

        /// <summary>
        /// Computes the variance for the specified group.
        /// </summary>
        /// <param name="group">The cluster group.</param>
        /// <param name="distanceAlgorithm">The distance algorithm.</param>
        /// <returns>The computed variance.</returns>
        internal    static    double    Compute(IClusterGroup<IAnswer,    GroupData>    group,    IDistance
distanceAlgorithm)
        {
            Collection<IAnswer> list = new Collection<IAnswer>();
            for (int i = 0; i < group.Nodes.Count; i++)
            {
                list.Add(group.Nodes[i].Value);
            }

            return GroupVariance.Compute(list, distanceAlgorithm);
        }

        /// <summary>
        /// Computes the variance for the specified list of answers.
        /// </summary>
        /// <param name="list">The list of answers..</param>
        /// <param name="distanceAlgorithm">The distance algorithm.</param>
        /// <returns>The computed variance.</returns>
        internal static double Compute(Collection<IAnswer> list, IDistance distanceAlgorithm)
        {
            int total = 0;
            for (int i = 0; i < list.Count; i++)
            {
                for (int j = 0; j < list.Count; j++)
                {
                    if (i > j)
                    {
                        total += distanceAlgorithm.GetDistance(list[i].Answer, list[j].Answer);
                    }
                }
            }

            double normalize = ((list.Count * list.Count) - list.Count) / 2.0;
            if (normalize == 0)
            {
                return normalize;
            }

            return total / normalize;
        }
    }
}



// <copyright file="HatfieldCluster.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The algorithm for clustering the answers.</summary>
namespace HatfieldCluster
{
```

```
using System;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Linq;
using System.Text;
using ClusterLibraryCore;
using GradeLibrary;

/// <summary>
/// The algorithm for clustering the answers.
/// </summary>
public class HatfieldCluster : IClusterAlgorithm
{
    /// <summary>
    /// The instance of the cluster.
    /// </summary>
    private ICluster<IAnswer, GroupData> cluster;

    /// <summary>
    /// The list of seeds.
    /// </summary>
    private ObservableCollection<string> seeds;

    /// <summary>
    /// The flag indicating that a report should be generated.
    /// </summary>
    private bool reportGeneration;

    /// <summary>
    /// The report that was generated.
    /// </summary>
    private StringBuilder report;

    /// <summary>
    /// The number of iterations the algorithm has executed.
    /// </summary>
    private int iterations;

    /// <summary>
    /// Initializes a new instance of the <see cref="HatfieldCluster"/> class.
    /// </summary>
    public HatfieldCluster()
    {
        this.cluster = null;
        this.seeds = new ObservableCollection<string>();
        this.reportGeneration = false;
        this.report = new StringBuilder();
        this.iterations = 0;
    }

    /// <summary>
    /// Initializes a new instance of the <see cref="HatfieldCluster"/> class.
    /// </summary>
    /// <param name="cluster">The cluster.</param>
    public HatfieldCluster(ICluster<IAnswer, GroupData> cluster)
    {
        this.cluster = cluster;
        this.seeds = new ObservableCollection<string>();
        this.reportGeneration = false;
        this.report = new StringBuilder();
        this.iterations = 0;
    }

    /// <summary>
    /// Gets or sets the cluster.
    /// </summary>
    /// <value>The cluster.</value>
    public ICluster<IAnswer, GroupData> Cluster
    {
```

```
    get { return this.cluster; }
    set { this.cluster = value; }
}

/// <summary>
/// Gets the seeds.
/// </summary>
/// <value>The seeds.</value>
public ObservableCollection<string> Seeds
{
    get { return this.seeds; }
}

/// <summary>
/// Gets the report.
/// </summary>
/// <value>The report.</value>
public string Report
{
    get
    {
        if (this.reportGeneration)
        {
            return this.report.ToString();
        }
        else
        {
            return "<h1>No Report Generated.</h1>";
        }
    }
}

/// <summary>
/// Gets or sets a value indicating whether [report generation].
/// </summary>
/// <value><c>true</c> if [report generation]; otherwise, <c>false</c>.</value>
public bool ReportGeneration
{
    get
    {
        return this.reportGeneration;
    }

    set
    {
        this.reportGeneration = value;
        this.report = new StringBuilder();
    }
}

/// <summary>
/// Gets the iterations.
/// </summary>
/// <value>The iterations.</value>
public int Iterations
{
    get { return this.iterations; }
}

/// <summary>
/// Processes this instance.
/// </summary>
public void Process()
{
    StringBuilder graph = new StringBuilder();
    if (this.reportGeneration)
    {
        this.report = new StringBuilder();
        this.report.AppendLine("<html><head>");
```

```
                this.report.AppendLine("<style    type=\"text/css\">td{width:120px;}tr.headrow{background-
color: #C1CDCD; font-weight: bold;}tr.highlightrow{background-color: #FDFCDC;}</style>");
                this.report.AppendLine("</head><body><h1>Report</h1>");
                graph.AppendLine("digraph {");
                graph.AppendLine("\tsubgraph cluster_0 {");
                for (int i = 1; i < this.cluster.Groups.Count; i++)
                {
                    graph.AppendLine("\t\t" + this.GraphVizNode(this.cluster.Groups[i], true) + ";");
                }

                graph.AppendLine("\t}");
            }

            int counter = 1;
            int candidateCount = 2;
            while (this.cluster.Groups.Count > 2 && candidateCount > 1)
            {
                if (this.reportGeneration)
                {
                    this.report.AppendLine("<h2>Pass " + counter + "</h2>");
                    this.DebugAnalysis();
                }

                this.iterations++;
                double total = 0;
                double num = 0;
                double min = int.MaxValue;
                int groupOne = -1;
                int groupTwo = -1;
                Collection<double> calculatedValues = new Collection<double>();
                Collection<GroupAnalysis> calculatedGroups = new Collection<GroupAnalysis>();

                // Perform all of the necessary comparisons
                for (int i = 1; i < this.cluster.Groups.Count; i++)
                {
                    for (int j = 1; j < this.cluster.Groups.Count; j++)
                    {
                        if (i > j)
                        {
                            string label = "Group " + j + ",Group " + i;
                            GroupAnalysis    ga    =    new    GroupAnalysis(label,    this.cluster.Groups[i],
this.cluster.Groups[j]);
                            calculatedGroups.Add(ga);
                            double dist = ga.CalculatedValue;
                            calculatedValues.Add(dist);
                            total += dist;
                            num++;
                            if (min > dist)
                            {
                                min = dist;
                                groupOne = i;
                                groupTwo = j;
                            }
                        }
                    }
                }

                // Calculate the threshold
                double average = total / num;
                double stddev = StandardDeviation.Calculate(calculatedValues);
                double threshold = average - stddev;

                // Count the number of comparisons that fall under the threshold
                candidateCount = 0;
                for (int i = 0; i < calculatedGroups.Count; i++)
                {
                    if (threshold > calculatedGroups[i].CalculatedValue)
                    {
                        candidateCount++;
```

```
                }
            }

            // Display the results table
            if (this.reportGeneration)
            {
                this.report.AppendLine("<h3>Comparisons</h3><table border='1' padding='4'>");
                this.report.AppendLine("<tr  class=\"headrow\"><td  colspan=2>Comparison</td><td>Token
Distance</td><td>Damerau    Levenshiein    Distance</td><td>Longest    Common    Substring</td><td>Average
Length</td><td>Size</td><td>Calculated Variance</td></tr>");
                for (int i = 0; i < calculatedGroups.Count; i++)
                {
                    GroupAnalysis ga = calculatedGroups[i];
                    string[] label = ga.Lable.Split(',');
                    if (ga.CalculatedValue < threshold)
                    {
                        this.report.AppendLine("<tr class=\"highlightrow\">");
                    }
                    else
                    {
                        this.report.AppendLine("<tr>");
                    }

                    this.report.AppendLine("<td>" + label[0] + "</td><td>" + label[1] + "</td>");
                    this.report.AppendLine(ga.GetReportRows());
                    this.report.AppendLine("</tr>");
                }

                this.report.AppendLine("</table>");

                // Display the value distribution graph
                string imgurl = this.ValueGraphURL(calculatedGroups, threshold, average, min);
                if (imgurl.Length < 2000)
                {
                    this.report.AppendLine("<br /><img src=\"" + imgurl + "\" /><br />");
                }
                else
                {
                    this.report.AppendLine("<br />Image could not be displayed.<br />");
                }

                // Display the report summary
                this.report.AppendLine("<h3>Analysis</h3>");
                this.report.AppendLine("Average Cost = " + average + "<br />");
                this.report.AppendLine("Standard Deviation of Cost = " + stddev + "<br />");
                this.report.AppendLine("Threshold = " + threshold + "<br />");
                this.report.AppendLine("Number of Candidates = " + candidateCount + "<br /><br />");
            }

            if (threshold > min)
            {
                if (this.reportGeneration)
                {
                    this.report.AppendLine("<i>Merging Group " + groupTwo + " and Group " + groupOne +
"</i><br />");
                }

                string a = this.cluster.Groups[groupOne].Uid.ToString();
                string b = this.cluster.Groups[groupTwo].Uid.ToString();
                this.cluster.Merge(this.cluster.Groups[groupOne], this.cluster.Groups[groupTwo]);
                string c = this.cluster.Groups[groupTwo].Uid.ToString();
                if (this.reportGeneration)
                {
                    graph.AppendLine("\t" + a + "->" + c + "[label=\"" + counter + "\"];");
                    graph.AppendLine("\t" + b + "->" + c + "[label=\"" + counter + "\"];");
                    graph.AppendLine("\t" + this.GraphVizNode(this.cluster.Groups[groupTwo], false) +
";");
                    if (candidateCount == 1)
                    {
```

```
                        this.report.AppendLine("<i>Algorithm terminating because only one candidate
found.</i><br />");
                    }

                    this.report.AppendLine("<hr />");
                }
            }
            else
            {
                if (this.reportGeneration)
                {
                    this.report.AppendLine("<i>No groups left to merge.</i><br />");
                    this.report.AppendLine("<hr />");
                }

                break;
            }

            counter++;
        }

        if (this.reportGeneration)
        {
            this.report.AppendLine("<h2>Final State</h2>");
            this.DebugAnalysis();
            this.report.AppendLine("<hr />");
            graph.AppendLine("\tsubgraph cluster_1 {");
            for (int i = 1; i < this.cluster.Groups.Count; i++)
            {
                graph.AppendLine("\t\t"     +     this.cluster.Groups[i].Uid     +     "[style=filled,
fillcolor=lightblue];");
            }

            graph.AppendLine("\t}");
            graph.AppendLine("}");
            string graphUrl = "https://chart.googleapis.com/chart?cht=gv&chl=" + graph.ToString();
            graphUrl  =  graphUrl.Replace("\t",  string.Empty).Replace("\n",  string.Empty).Replace("  ",
"+");
            this.report.AppendLine("<h1>Graph Report</h1>");
            this.report.AppendLine("<img src='" + graphUrl + "' />");
            this.report.AppendLine("<h2>GraphViz Source Code</h2>");
            this.report.AppendLine("<pre  style=\"font-family:  Andale  Mono,  Lucida  Console,  Monaco,
fixed, monospace; color: #000000; background-color: #EEE; font-size: 10px; line-height: 4px; \"><code>");
            this.report.AppendLine(graph.ToString().Replace("\n", "<br />\n"));
            this.report.AppendLine("</code></pre>");
            this.report.AppendLine("</body></html>");
        }
    }

    /// <summary>
    /// Generate the label for a group representing a GraphViz node.
    /// </summary>
    /// <param name="g">The group.</param>
    /// <param name="original">if set to <c>true</c> [original].</param>
    /// <returns>The node with the label.</returns>
    private string GraphVizNode(IClusterGroup<IAnswer, GroupData> g, bool original)
    {
        StringBuilder sb = new StringBuilder();
        sb.Append(g.Uid.ToString());
        sb.Append("[label=\"");
        for (int i = 0; i < g.Nodes.Count; i++)
        {
            sb.Append(g.Nodes[i].Value.Answer);
            if (i + 1 < g.Nodes.Count)
            {
                sb.Append(", ");
            }
        }
```

```
            sb.Append("\"");
            if (original)
            {
                sb.Append(", style=filled, fillcolor=lightyellow");
            }

            sb.Append("]");
            return sb.ToString();
        }

        /// <summary>
        /// Generates the graph for the calculated values.
        /// </summary>
        /// <param name="calculatedGroups">The calculated groups.</param>
        /// <param name="threshold">The threshold.</param>
        /// <param name="avg">The average.</param>
        /// <param name="min">The minimum.</param>
        /// <returns>The URL to a Google charts scatter plot.</returns>
        private string ValueGraphURL(Collection<GroupAnalysis> calculatedGroups, double threshold, double
avg, double min)
        {
            int number = 1;
            StringBuilder sb = new StringBuilder();
            sb.Append("https://chart.googleapis.com/chart?cht=s&chs=600x300&chd=t:");
            for (int i = 0; i < calculatedGroups.Count; i++)
            {
                int index = (int)Math.Round(100.0 * (double)i / (double)calculatedGroups.Count);
                for (int j = 0; j < number; j++)
                {
                    sb.Append(index);
                    if (j + 1 < number)
                    {
                        sb.Append(",");
                    }
                }

                if (i + 1 < calculatedGroups.Count)
                {
                    sb.Append(",");
                }
            }

            sb.Append(",0,100,0,100");

            double scale = 0;
            for (int i = 0; i < calculatedGroups.Count; i++)
            {
                if (calculatedGroups[i].CalculatedValue > scale)
                {
                    scale = calculatedGroups[i].CalculatedValue;
                }
            }

            sb.Append("|");
            for (int i = 0; i < calculatedGroups.Count; i++)
            {
                sb.Append(Math.Round(100.0 * calculatedGroups[i].CalculatedValue / scale, 2));
                if (i + 1 < calculatedGroups.Count)
                {
                    sb.Append(",");
                }
            }

            // Add the points for the horizontal lines
            sb.Append(",");
            sb.Append(Math.Round(100.0 * threshold / scale, 2));
            sb.Append(",");
            sb.Append(Math.Round(100.0 * threshold / scale, 2));
            sb.Append(",");
```

```
            sb.Append(Math.Round(100.0 * avg / scale, 2));
            sb.Append(",");
            sb.Append(Math.Round(100.0 * avg / scale, 2));

            // Add all of the special things to the end of the URL
            sb.Append("&chxt=x,y&chco=FF0000&chdl=Value");
            int n = calculatedGroups.Count;
            sb.Append("&chm=o,0000FF,0,-1,0|o,FF0000,0,0:" + (n - 1) + ":,10");
            sb.Append("|D,FFBA00,1," + n + ":" + (n + 1) + ",2,-1|D,000000,1," + (n + 2) + ":" + (n + 3) +
",2,-1");
            for (int i = 0; i < calculatedGroups.Count; i++)
            {
                if (calculatedGroups[i].CalculatedValue <= min)
                {
                    sb.Append("|o,00FF00,0," + i + ",10");
                    break;
                }
            }

            sb.Append("&chxr=0,1," + (n + 1) + "," + Math.Ceiling(n / 15.0) + "|1,0," + scale);
            return sb.ToString();
        }

        /// <summary>
        /// Prints out useful information to the debug window.
        /// </summary>
        private void DebugAnalysis()
        {
            double total = 0;
            this.report.AppendLine("<h3>Groups</h3><table border=1 padding=4>");
            this.report.AppendLine("<tr             class=\"headrow\"><td>Group</td><td>Nodes</td><td>Token
Distance</td><td>Damerau    Levenshiein    Distance</td><td>Longest    Common    Substring</td><td>Average
Length</td><td>Size</td><td>Calculated Variance</td></tr>");

            for (int i = 1; i < this.cluster.Groups.Count; i++)
            {
                this.report.Append("<tr><td>Group " + i + "</td><td>");

                for (int j = 0; j < this.cluster.Groups[i].Nodes.Count; j++)
                {
                    this.report.Append(this.cluster.Groups[i].Nodes[j].Value.Answer + "<br />");
                }

                this.report.AppendLine("</td>");
                GroupAnalysis ga = new GroupAnalysis(string.Empty, this.cluster.Groups[i]);
                if (this.reportGeneration)
                {
                    this.report.AppendLine(ga.GetReportRows());
                }

                total += ga.CalculatedValue;
                this.report.AppendLine("</tr>");
            }

            this.report.AppendLine("</table>");
            this.report.AppendLine("<br />Total Variance = " + total);
        }
    }
}



// <copyright file="IDistance.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The IDistance.cs file.</summary>
namespace HatfieldCluster
{
```

```
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// The abstract class for computing various types of string distances.
    /// </summary>
    internal abstract class IDistance
    {
        /// <summary>
        /// Computes the string distance.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The string distance.</returns>
        internal abstract int GetDistance(string s, string t);
    }
}



// <copyright file="LongestSubstring.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The LongestSubstring.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// Computes the length of the longest substring.
    /// </summary>
    internal class LongestSubstring : IDistance
    {
        /// <summary>
        /// The cache used for speeding up the comparisons.
        /// </summary>
        private static CacheManager cache = new CacheManager();

        /// <summary>
        /// Computes the length of the longest substring.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The length of the longest substring.</returns>
        internal static int Distance(string s, string t)
        {
            // Attempt to retreive value from cache
            Nullable<int> cached = LongestSubstring.cache.Get(s, t);
            if (cached != null && cached.HasValue)
            {
                return cached.Value;
            }

            // Calculate the value
            if (String.IsNullOrEmpty(s) || String.IsNullOrEmpty(t))
            {
                return 0;
            }

            int[,] mat = new int[s.Length, t.Length];
            int len = 0;

            for (int i = 0; i < s.Length; i++)
```

```
            {
                for (int j = 0; j < t.Length; j++)
                {
                    if (s[i] != t[j])
                    {
                        mat[i, j] = 0;
                    }
                    else
                    {
                        if ((i == 0) || (j == 0))
                        {
                            mat[i, j] = 1;
                        }
                        else
                        {
                            mat[i, j] = 1 + mat[i - 1, j - 1];
                        }

                        if (mat[i, j] > len)
                        {
                            len = mat[i, j];
                        }
                    }
                }
            }

            // Cache the value
            LongestSubstring.cache.Put(s, t, len);

            return len;
        }

        /// <summary>
        /// Computes the string distance.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The string distance.</returns>
        internal override int GetDistance(string s, string t)
        {
            return LongestSubstring.Distance(s, t);
        }
    }
}



// <copyright file="StandardDeviation.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The StandardDeviation.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.ObjectModel;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// Calculates the standard deviation.
    /// </summary>
    internal class StandardDeviation
    {
        /// <summary>
        /// Calculates the standard deviation for the list.
        /// </summary>
        /// <param name="list">The list of values.</param>
        /// <returns>The standard deviation.</returns>
```

```
        internal static double Calculate(Collection<double> list)
        {
            double sumOfValuesSquared = 0;
            double sumOfValues = 0;

            foreach (double item in list)
            {
                sumOfValues += item;
            }

            foreach (double item in list)
            {
                sumOfValuesSquared += Math.Pow(item, 2);
            }

            return Math.Sqrt((sumOfValuesSquared - (Math.Pow(sumOfValues, 2) / list.Count)) / list.Count);
        }
    }
}
```

```
// <copyright file="TokenizedStringDistance.cs" company="Jared Hatfield">
// All Rights Reserved 2011
// </copyright>
// <summary>The TokenizedStringDistance.cs file.</summary>
namespace HatfieldCluster
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    /// <summary>
    /// Computes the tokenized string distance.
    /// </summary>
    internal class TokenizedStringDistance : IDistance
    {
        /// <summary>
        /// The cache used for speeding up the comparisons.
        /// </summary>
        private static CacheManager cache = new CacheManager();

        /// <summary>
        /// Computes the tokenized string distance.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The tokenized string distance.</returns>
        internal static int Distance(string s, string t)
        {
            // Attempt to retreive value from cache
            Nullable<int> cached = TokenizedStringDistance.cache.Get(s, t);
            if (cached != null && cached.HasValue)
            {
                return cached.Value;
            }

            // Calculate the value
            string[] stoken = s.Split(' ');
            string[] ttoken = t.Split(' ');

            int count = 0;
            for (int i = 0; i < stoken.Length; i++)
            {
                if (!ttoken.Contains(stoken[i]))
                {
                    count++;
```

```
                }
            }

            for (int i = 0; i < ttoken.Length; i++)
            {
                if (!stoken.Contains(ttoken[i]))
                {
                    count++;
                }
            }

            // Cache the value
            TokenizedStringDistance.cache.Put(s, t, count);

            return count;
        }

        /// <summary>
        /// Computes the string distance.
        /// </summary>
        /// <param name="s">The first string.</param>
        /// <param name="t">The second string.</param>
        /// <returns>The tokenized string distance.</returns>
        internal override int GetDistance(string s, string t)
        {
            return TokenizedStringDistance.Distance(s, t);
        }
    }
}
```

APPENDIX B

This appendix contains an output report from the clustering algorithm that details the calculations performed by the algorithm.

Report

Pass 1

Groups

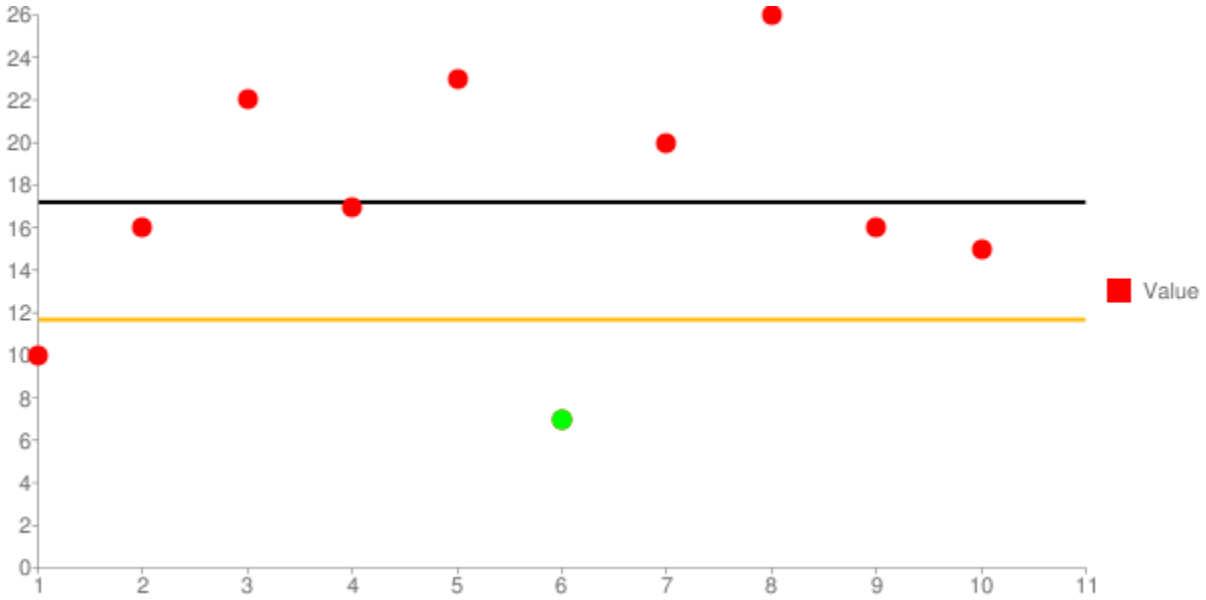
Group	Nodes	Token Distance	Damerau Levenshiein Distance	Longest Common Substring	Average Length	Size	Calculated Variance
Group 1	cat	1	0	3	3	1	1
Group 2	a cat	1	0	5	5	1	1
Group 3	dog	1	0	3	3	1	1
Group 4	dogs	1	0	4	4	1	1
Group 5	mouse	1	0	5	5	1	1

Total Variance = 5

Comparisons

Comparison		Token Distance	Damerau Levenshiein Distance	Longest Common Substring	Average Length	Size	Calculated Variance
Group 1	Group 2	1	3	3	4	2	10
Group 1	Group 3	2	3	0	3	2	16
Group 2	Group 3	3	4	0	4	2	22
Group 1	Group 4	2	3	0	3.5	2	17
Group 2	Group 4	3	4	0	4.5	2	23
Group 3	Group 4	2	1	3	3.5	2	7
Group 1	Group 5	2	4	0	4	2	20
Group 2	Group 5	3	5	0	5	2	26
Group 3	Group 5	2	3	1	4	2	16

Group 4	Group 5	2	2	1	4.5	2	15



Analysis

Average Cost = 17.2
Standard Deviation of Cost = 5.5281099844341
Threshold = 11.6718900155659
Number of Candidates = 2

Merging Group 3 and Group 4

Pass 2

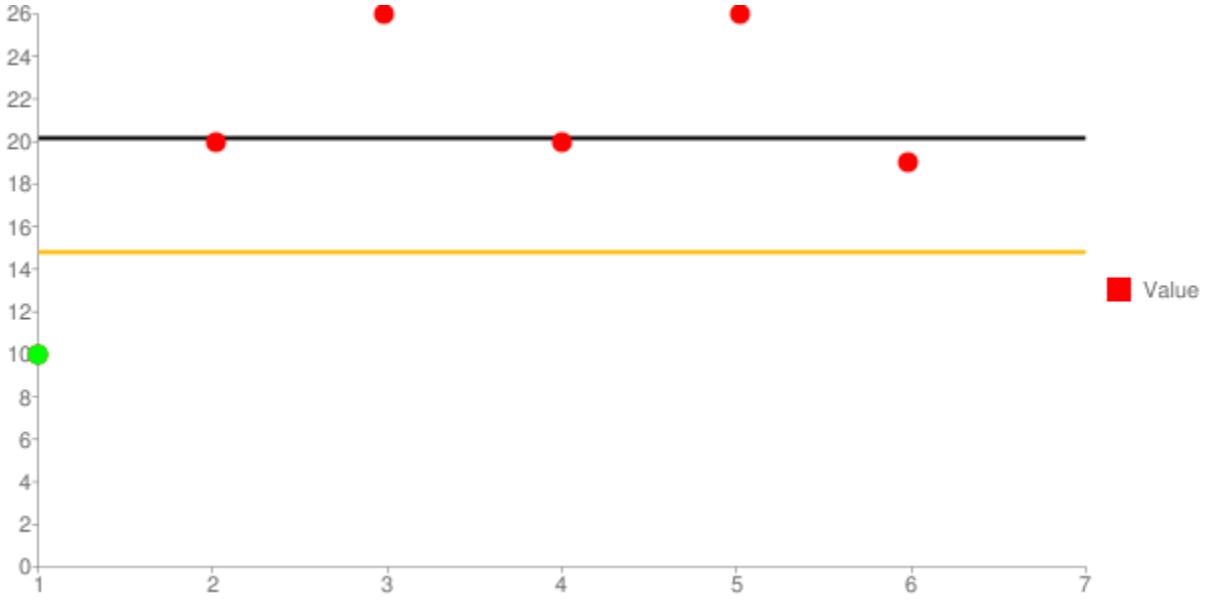
Groups

Group	Nodes	Token Distance	Damerau Levenshiein Distance	Longest Common Substring	Average Length	Size	Calculated Variance
Group 1	cat	1	0	3	3	1	1
Group 2	a cat	1	0	5	5	1	1
Group 3	dog dogs	2	1	3	3.5	2	7
Group 4	mouse	1	0	5	5	1	1

Total Variance = 10

Comparisons

Comparison		Token Distance	Damerau Levenshiein Distance	Longest Common Substring	Average Length	Size	Calculated Variance
Group 1	Group 2	1	3	3	4	2	10
Group 1	Group 3	2	2.33333	1	3.33333	3	20
Group 2	Group 3	2.66667	3	1	4	3	26
Group 1	Group 4	2	4	0	4	2	20
Group 2	Group 4	3	5	0	5	2	26
Group 3	Group 4	2	2	1.66667	4	3	19



Analysis

Average Cost = 20.1666666666667
Standard Deviation of Cost = 5.36708072936821
Threshold = 14.7995859372985
Number of Candidates = 1

Merging Group 1 and Group 2
Algorithm terminating because only one candidate found.

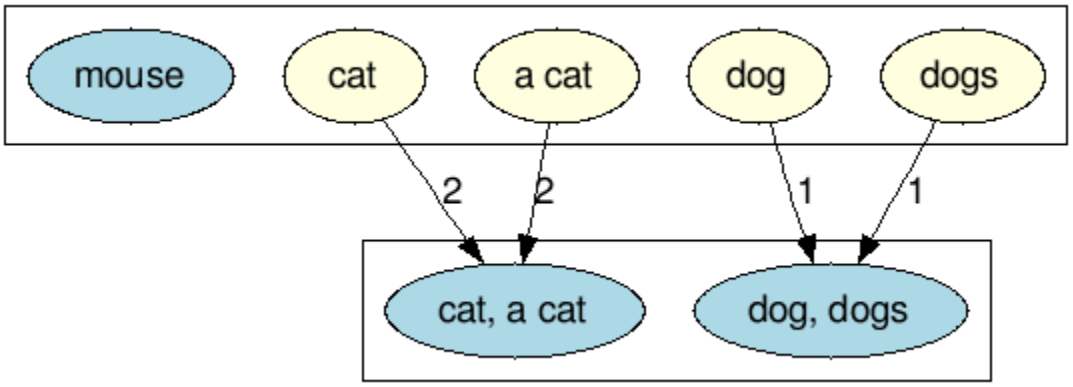
Final State

Groups

Group	Nodes	Token Distance	Damerau Levenshiein Distance	Longest Common Substring	Average Length	Size	Calculated Variance
Group 1	cat a cat	1	3	3	4	2	10
Group 2	dog dogs	2	1	3	3.5	2	7
Group 3	mouse	1	0	5	5	1	1

Total Variance = 18

Graph Report



GraphViz Source Code

```
digraph {
	subgraph cluster_0 {
		1[label="cat", style=filled, fillcolor=lightyellow];
		2[label="a cat", style=filled, fillcolor=lightyellow];
		3[label="dog", style=filled, fillcolor=lightyellow];
		4[label="dogs", style=filled, fillcolor=lightyellow];
		5[label="mouse", style=filled, fillcolor=lightyellow];
	}

	4->6[label="1"];
	3->6[label="1"];
	6[label="dog, dogs"];
	2->7[label="2"];
```

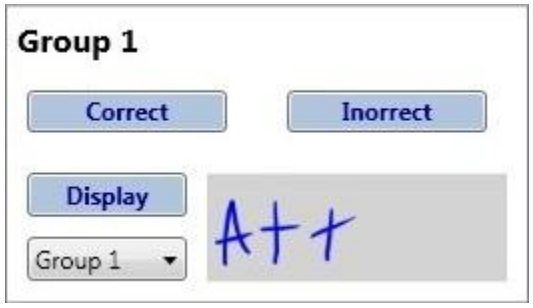
```
	1->7[label="2"];
	7[label="cat, a cat"];

	subgraph cluster_1 {
		7[style=filled, fillcolor=lightblue];
		6[style=filled, fillcolor=lightblue];
		5[style=filled, fillcolor=lightblue];
	}
}
```

APPENDIX C

This appendix contains the clustering results from an analysis performed by DPX Answers on sample questions answered by students in a classroom setting.

Question 1

Your Cell Phone Provider:

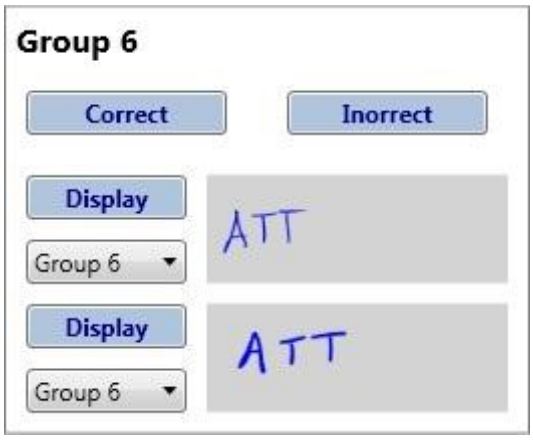



Group 2
Correct
Inorrect
Display
Group 2
Verizon
Display
Group 2
Verizon
Display
Group 2
Verizon
Display
Group 2
Verizon
Display
Group 2
Verizon
Display
Group 2
Verizon
Display
Group 2
Verizon
Display
Group 2
Verizon


Group 3
Correct
Inorrect
Display
Group 3
T-Mobile

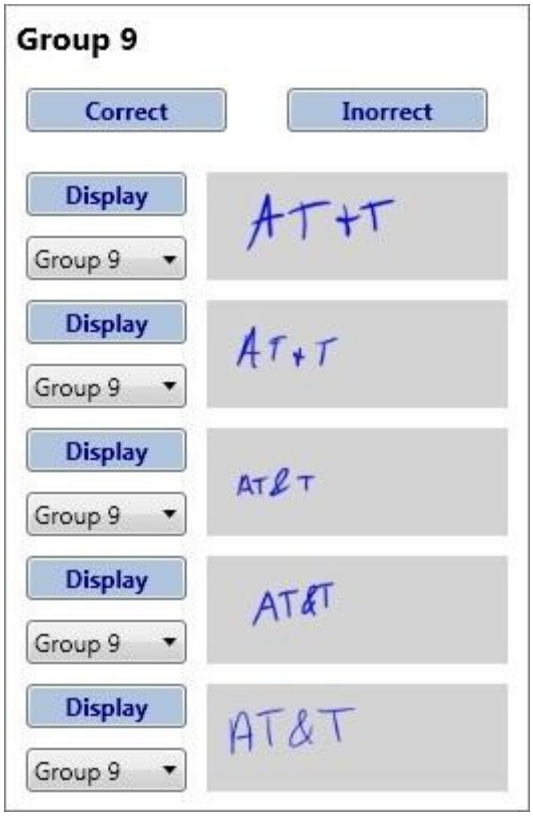

Group 4
Correct
Inorrect
Display
Group 4
tmobile


Group 5
Correct
Inorrect
Display
Group 5
verison

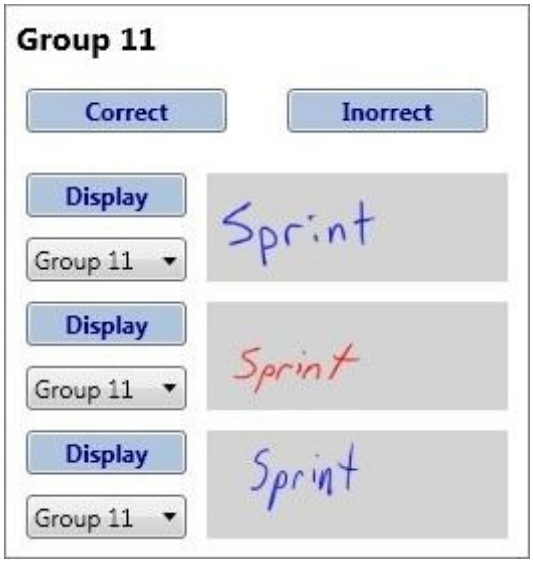

Group 6
Correct
Inorrect
Display
Group 6
ATT
Display
Group 6
ATT


Group 7
Correct
Inorrect
Display
Group 7
AT&T


Group 8
Correct
Inorrect
Display
Group 8
at &t

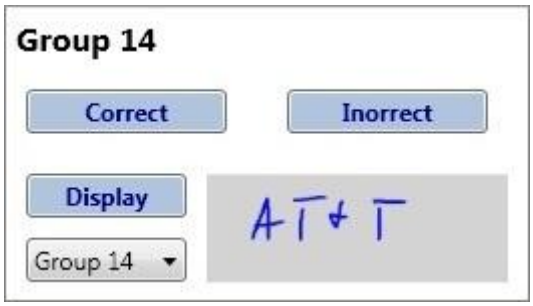

Group 9
Correct
Inorrect
Display
Group 9
AT+T
Display
Group 9
AT+T
Display
Group 9
AT&T
Display
Group 9
AT&T
Display
Group 9
AT&T


Group 10
Correct
Inorrect
Display
Group 10
Cincinnati Bell

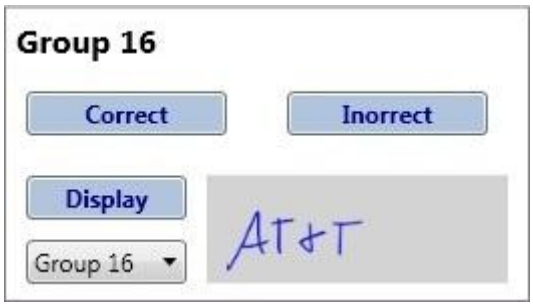

Group 11
Correct
Inorrect
Display
Group 11
Sprint
Display
Group 11
Sprint
Display
Group 11
Sprint

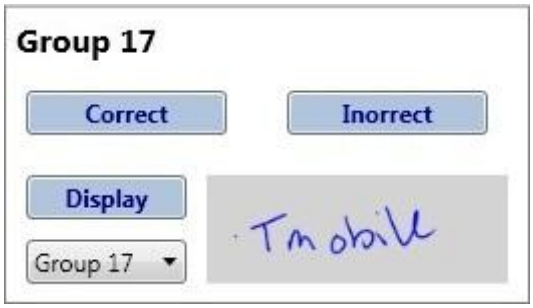

Group 12
Correct
Inorrect
Display
Group 12
AT&T
Display
Group 12
AT &T


Group 13
Correct
Inorrect
Display
Group 13
VERIZON


Group 14
Correct
Inorrect
Display
Group 14
AT&T

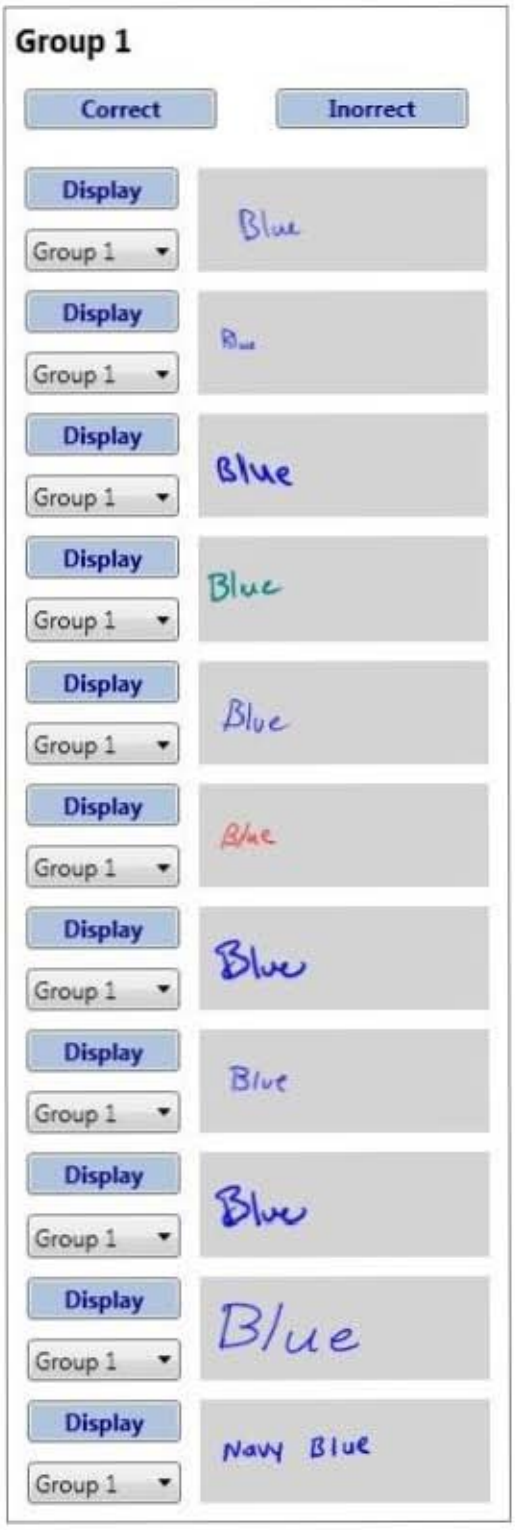

Group 15
Correct
Inorrect
Display
Group 15
Virgin Mobile


Group 16
Correct
Inorrect
Display
Group 16
AT&T

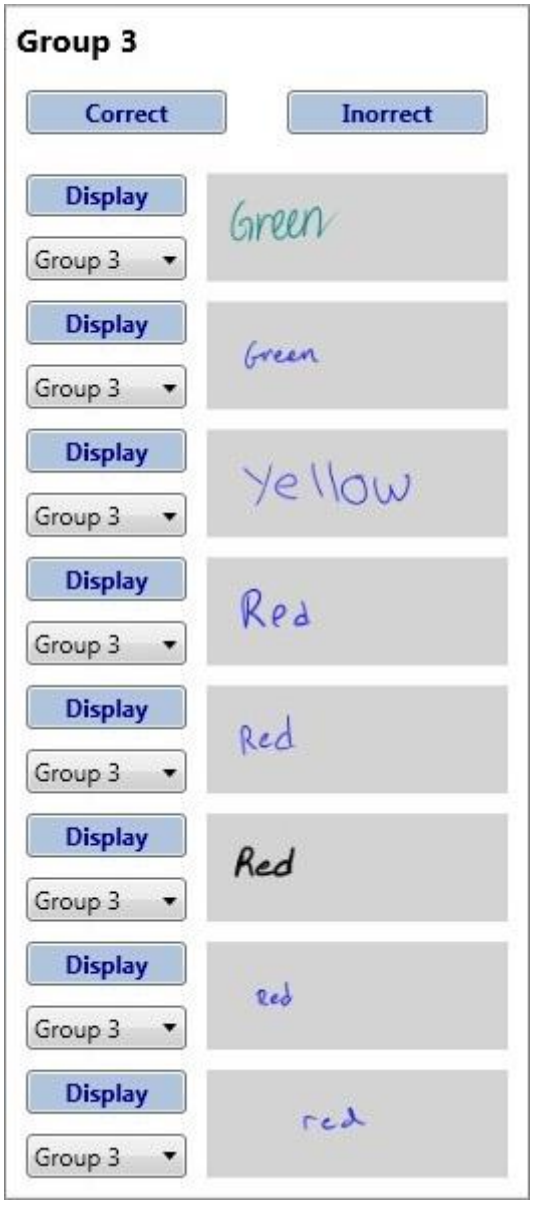

Group 17
Correct
Inorrect
Display
Group 17
Tmobile

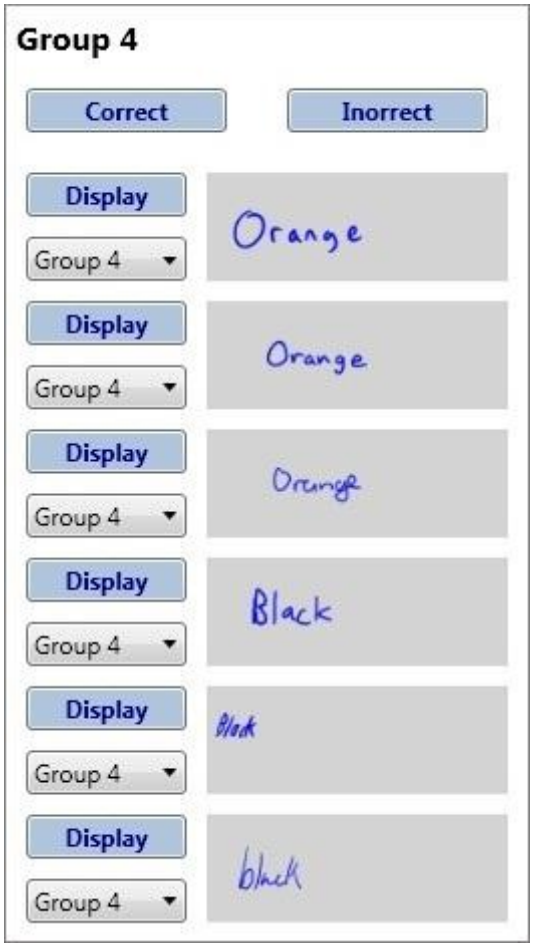
Question 2

Your Favorite Color:



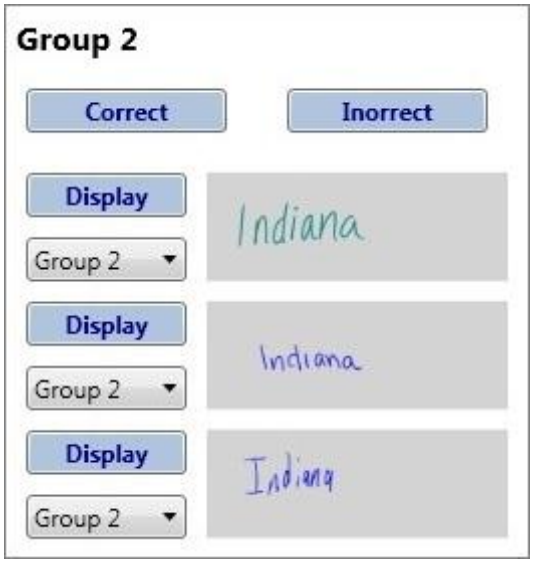

Group 2
Correct
Inorrect
Display
Group 2
TEAL
Display
Group 2
N/A
Display
Group 2
pink
Display
Group 2
PNK
Display
Group 2
Purple
Display
Group 2
Purple
Display
Group 2
Purple

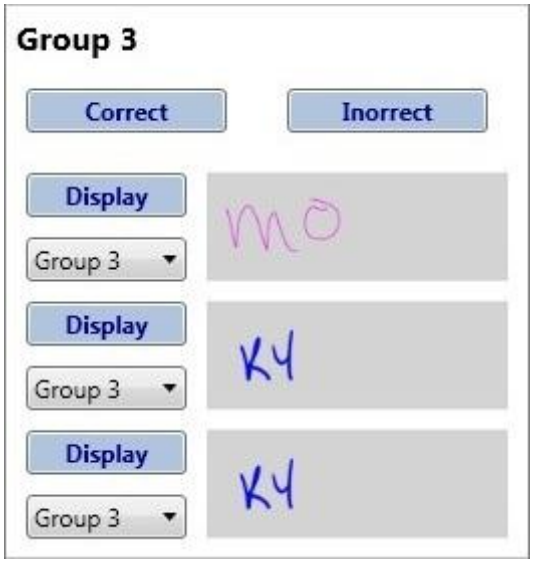

Group 3
Correct
Inorrect
Display
Group 3
Green
Display
Group 3
Green
Display
Group 3
Yellow
Display
Group 3
Red
Display
Group 3
Red
Display
Group 3
Red
Display
Group 3
red
Display
Group 3
red

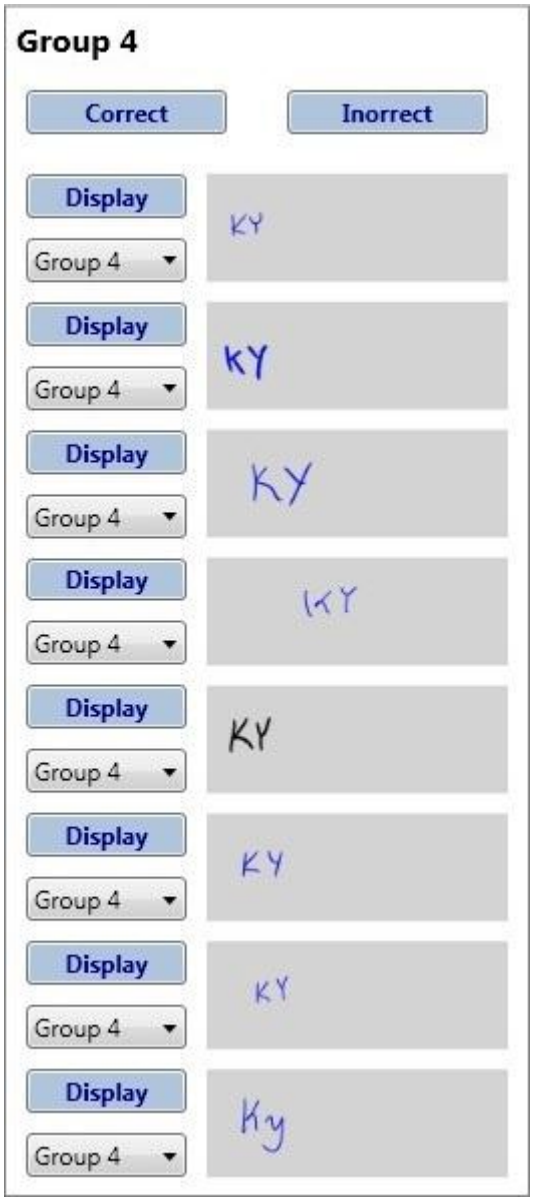

Group 4
Correct
Inorrect
Display
Group 4
Orange
Display
Group 4
Orange
Display
Group 4
Orange
Display
Group 4
Black
Display
Group 4
Black
Display
Group 4
black

Question 3

Your Home State:


Group 2
Correct
Inorrect
Display
Group 2
Indiana
Display
Group 2
Indiana
Display
Group 2
Indiana


Group 3
Correct
Inorrect
Display
Group 3
MO
Display
Group 3
KY
Display
Group 3
KY


Group 4
Correct
Inorrect
Display
Group 4
KY
Display
Group 4
KY
Display
Group 4
KY
Display
Group 4
KY
Display
Group 4
KY
Display
Group 4
KY
Display
Group 4
KY
Display
Group 4
Ky


Group 5
Correct
Inorrect
Display
Group 5
Michigan
Display
Group 5
Michigan


Group 6
Correct
Inorrect
Display
Group 6
Kentacky

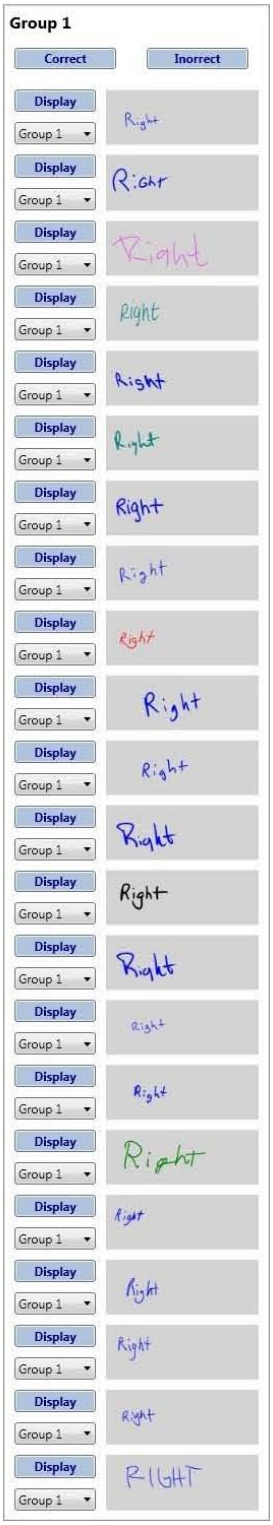

Group 7
Correct
Inorrect
Display
Group 7
Califorria

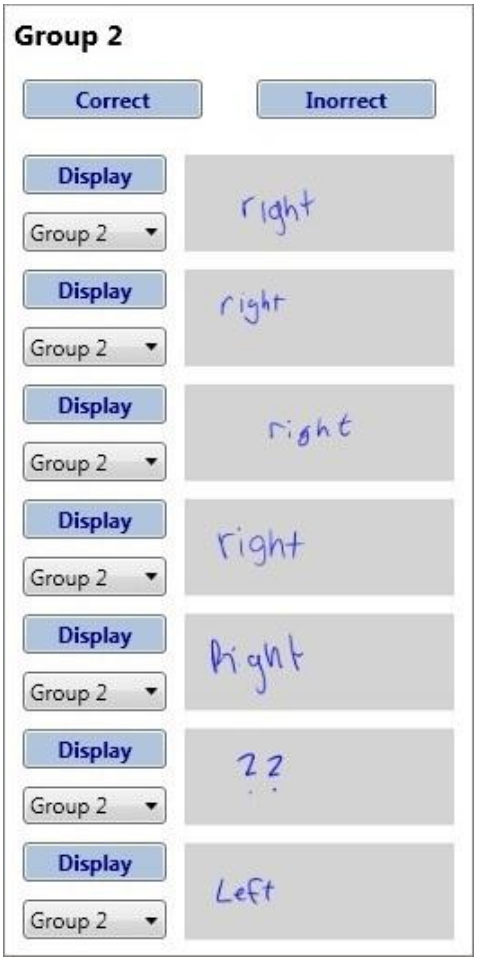

Group 8
Correct
Inorrect
Display
Group 8
KY

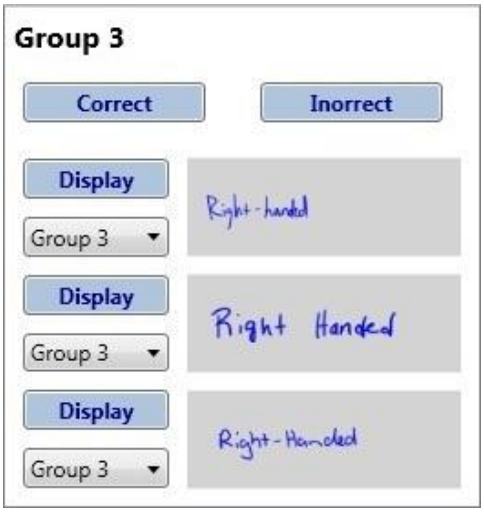

Group 9
Correct
Inorrect
Display
Group 9
Tennessee

Question 4

Your Handedness:




Group 2
Correct
Inorrect
Display
Group 2
right
Display
Group 2
right
Display
Group 2
right
Display
Group 2
right
Display
Group 2
Right
Display
Group 2
22
Display
Group 2
Left

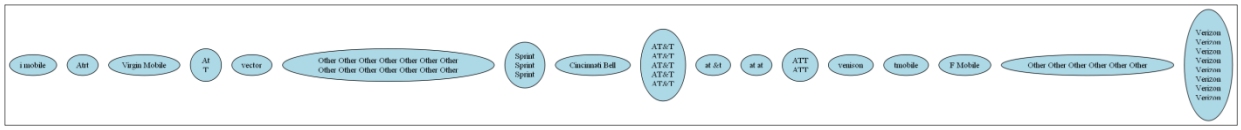

Group 3
Correct
Inorrect
Display
Group 3
Right-handed
Display
Group 3
Right Handed
Display
Group 3
Right-Handed

APPENDIX D

This appendix contains the graphical visualization of the clustering results performed by DPX Answers on sample questions answered by students in a classroom setting. The visualization contains the original clusters and the results of the algorithm are shown as indicated through the merging of clusters in the order indicated by the number on the edges of the graph.

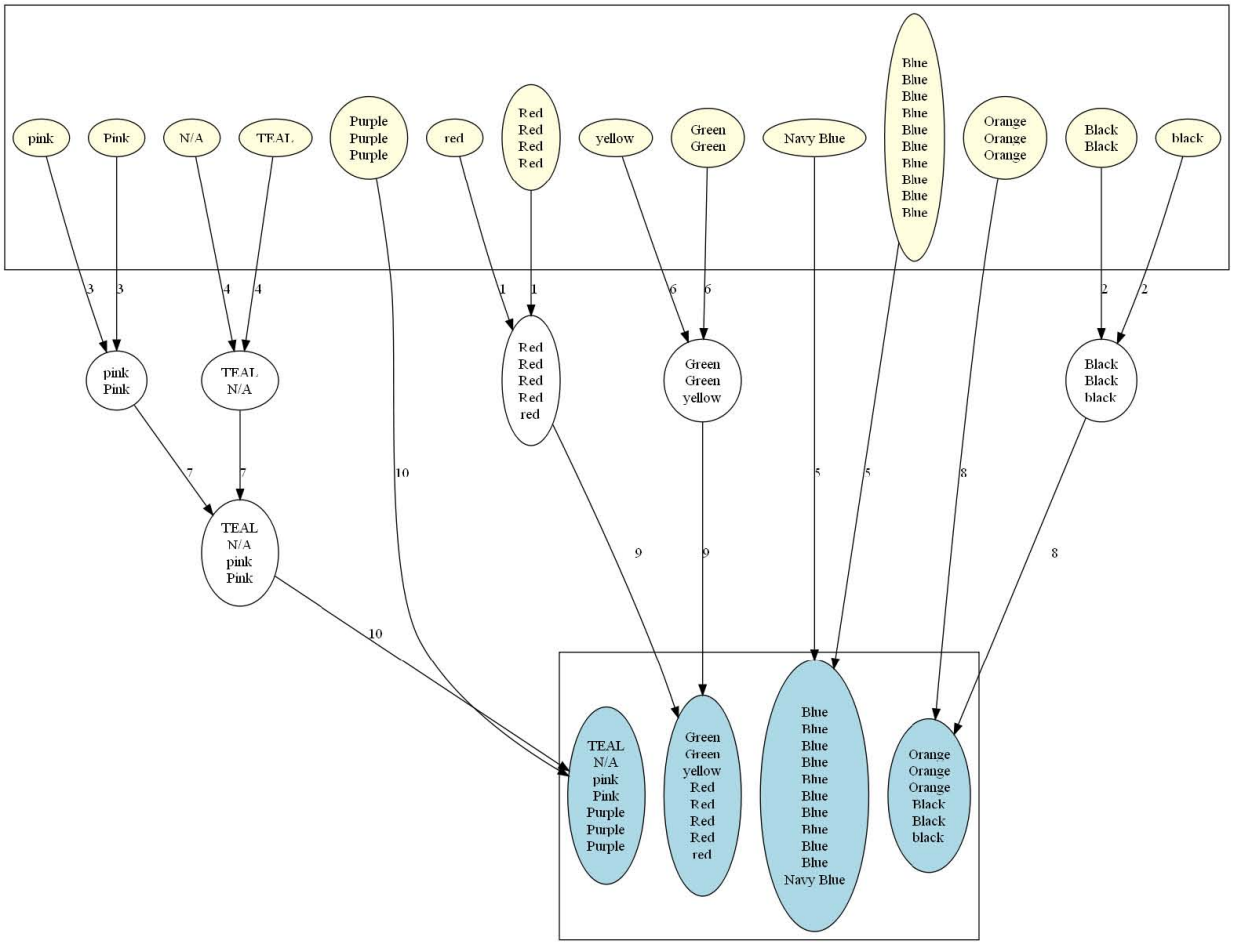
Question 1

Your Cell Phone Provider:



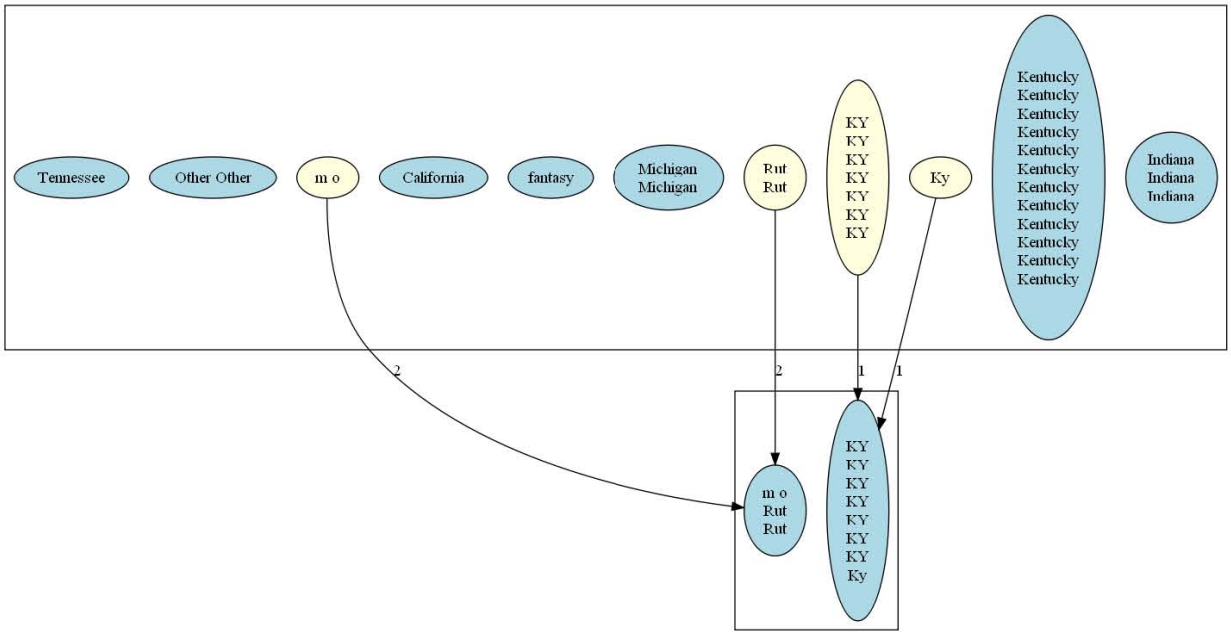
Question 2

Your Favorite Color:



Question 3

Your Home State:



Question 4

Your Handedness: