Creating a Database of Assessment Instruments

Daniel Savoie, Garrison Benson, Lidiya Ilcheva, Dr. Ryan McFall

Abstract

This project is a component of a larger project whose goal is to build an online database of assessments and records of student performance on these assessments. Our portion of this task is to process an assessment (e.g. a homework assignment, quiz or examination) to identify the questions and load them into this database. To encourage use of the system, we need this process to be quick and easy for the user. Copying and pasting the questions from the assessments into the database would be easy, but not quick—hence the need to shift some of the work away from users.

Our method takes an assessment instrument in PDF format as input. It extracts information from this input file, including text and images, and attempts to separate and categorize the questions, using white space and keywords to guide the process. To handle any questions our method extracted incorrectly or missed, we developed an application that provides the capability to edit the questions. We made it a high priority to design this interface to be easy to use, consistent with our goal in creating a quick and easy method from the user's perspective.

Our algorithm has been tested on a sample of 50 assessments from different educators and institutions. On this sample, we achieved a success rate of over 86% in correct question extraction.

Introduction

The database stores questions in XML format in accordance with the QTI 1.2 specification [3]. Question types include true or false, multiple choice, fill in the blank, image hot spot, short answer, and many others, although those five are primarily what we support. After a question is extracted and categorized, we generate XML to represent the question.

For our purposes, the PDF format itself is a hurdle; it is a format for viewing, not editing. Yet we chose PDF in the interest of supporting the widest possible range of student assessment documents. Because of the capability of Adobe's PDF format to be generated from numerous applications running on various operating systems, it is essentially a universal format, meeting our goal. And because PDF is nearly universal, we were able to find open source code for helping with the uncommon task of extracting information from PDF files. This wide range of supported documents, however, is a problem in and of itself.

Writers of student assessments have vastly differing ways of laying out their documents. Multiple choice Biology exams from the same University but written by different professors, for example, are very different in some of the cases we looked at. Some have two columns, some have images, and some have no white space between questions. This is only some of the variability when question type, subject and institution are held constant, and it follows that this variability greatly increases when less closely related assessments are compared. We indeed found this to be the case upon examining assessments outside our initial batch from one University.

Below are two different questions from different assessments. The questions have a similar appearance but require very different responses.

A short answer question with five sub-questions:

12. (20 Pts) A portion of the output from the program STATMECH is attached. Assuming that $\epsilon = 0.03$ eV, and using these data from STATMECH, find:

    (a) the average energy per particle for the particles in the A partition when the two partitions have reached equilibrium.
    (b) the temperature of the A partition at equilibrium using the fact that an Einstein oscillator has six degrees of freedom.
    (c) the temperature of the A partition at equilibrium as determined from the entropy. (Compare this answer to that of part (b).
    (d) how does the temperature of the A partition compare to that of the reservoir.
    (e) the probability that the A partition will remain in a state with an average energy of $0.0805U$ instead of moving towards equilibrium.

A multiple choice question:

20. Choosing the right data structure

    (a) depends only on the operations one wishes to perform on the data.
    (b) depends on the operations one wishes to perform on the data, and how often each is performed, but not on whether the data is static or dynamic.
    (c) depends on the operations one wishes to perform and how often, and whether the data is static or dynamic.
    (d) depends only on whether the data is static or dynamic.

Our collection of documents reveals several recurring complications. One of these is how to recognize the start of a question when a question can begin with integers, letters, roman numerals, or none of those. Another is the use of whitespace information. White space can have many meanings in a document, and changing the allowed threshold that determines when a question ends can lead to an endless cycle of breaking and fixing. In addition, images extracted from the PDF document need to be associated with their corresponding questions. Deciding what should and should not be considered an indication of question type (e.g., multiple underscores in a row could indicate a fill in the blank question, but the question might actually be multiple choice) is also a problem. And of the five question types we are working with, short answer has the most complications; short answer questions with multiple sub-questions are fairly common and can be represented an endless number of ways.

The user interface used to edit the output of our recognition algorithm is designed with potential errors resulting from these complications and other common errors in mind. Question type can be changed with a drop-down box, images can be dragged to and from questions, and the text can be edited. Choices can be added or removed from multiple choice questions, and whole questions can be deleted. This is not to understate the importance of high accuracy, however, as high accuracy reduces editing time.

Background

We performed an extensive search for related work. Aside from general information on extracting text from documents, and on natural language processing, we did not find any past projects which attempted to solve our problem specifically. However, Dr Chao and Dr Fan's work on extracting layout and content of PDF documents [1] was a stepping stone in forming our ideas on a method for extracting the text and images of the assessment documents. Their paper describes a general way of extracting the content of a PDF file (text and images) by using coordinate information. Their algorithm

separates text, images, and vector graphics and processes them independently. We process them simultaneously, and do not keep information about the layout of the document, such as indentation and the position of images relative to the question that uses them, because it is irrelevant for our purposes.

To extract the content of the assessment instruments we use a Java library called Multivalent [2]. It reads PDF, among other file formats, and provides various tools to manipulate PDF files; in our work we use Multivalent to extract images and text from the PDF files. Multivalent provides classes for building a document tree of the PDF file, which contains all of the text and images of the file in its leaves. The rest of the nodes combine the leaves in a hierarchy, in which the whole document is the root of the tree. The Multivalent library also includes a PDF viewer, which is embedded in our user interface as a Swing component.

Algorithm

We first attempted a method that was more dependent on understanding the text than on white space. The method began by finding the start of a question. It would then examine subsequent text until it could determine the question type. Once done, a new question object was created and a polymorphic parse() method was invoked. The parse() method had the document passed as a parameter and would begin processing the question at the location at which the new question was found. Prompt (the part of every question that asks the question--for some question types there is no other text) and image fields needed to be filled in, along with choice fields for multiple choice questions. In every iteration the method would be checking for the start of another question, indicating that it was time to return to the main method.

This worked well for multiple choice, true or false, and fill in the blank questions; ultimately we had to abandon this method due to its insufficiency in handling short answer questions and less traditionally formatted assessments. Short answer questions can be formatted a limitless number of ways, so trying to understand them conflicted with the understanding of other questions.

The approach we are using first separates all the questions and then attempts to infer the question type from their content.

Our work can be separated into two stages: extracting the content of a PDF document, and recognizing and classifying the questions in it. To extract the content of the PDF files we use a Java library called Multivalent [2] and its ability to construct a document tree of the contents of the PDF file. A document tree of a PDF file consists of leaf nodes, which contain the content of the document, and parent nodes, which group the content in the leaf nodes in a hierarchy. There are three types of leaves: text, image and graph. In our algorithm we only extract information from text and image leaves and ignore the graph leaves, save for their coordinate information. Each graph leaf contains a single vector graphics object, which can be a path, a line, or a point. Multivalent constructs a separate document tree for each page of the PDF document. From the tree one can retrieve information about the position of each separate text element on the page. We use this coordinate information to combine the extracted text into words and lines. As image leaves do not contain coordinate information, we insert HTML image tags in the text to help us identify the position of each image in the flow of the document. Every time we encounter an image leaf, we create a new text leaf containing an HTML image tag with the name of the file in which the image would be stored, and insert it in the extracted text. The text nodes which contain the image tags are assigned coordinate information based on the coordinate information of the text, which is immediately before the image. This information helps us associate images with questions in the next stage of the algorithm. At this stage we also combine images which are split horizontally into several pieces (ranging from 2 to 12 or even 15 pieces), with each piece being in a separate leaf of the document tree. Figure 1 shows such an image. This image, which is from a biology exam, is extracted from the file into 16 separate pieces (shown in Figure 2). Our algorithm combines it back to the state it is seen in the PDF. As we do not possess coordinate information for images, we combine them based on proximity and common width. All split images are combined in our test set with only one error, where two separate sequential occurrences of the same image are combined by mistake. Figure 3 shows an image from a biology assessment, in which the students are asked to predict the results of three experiments on a batch of radish seeds under different conditions (light and water; light, no water; water, no light). The seeds will not change in the

presence of light and no water. The instructor illustrates with fact by using the same image twice - first to show the state of the seeds before the experiment, and then to show the state of the seeds after the experiment. As a result out algorithm combines the two separate sequential occurrences of the same image. The correct placement of the two images in the file is shown in Figure 4.

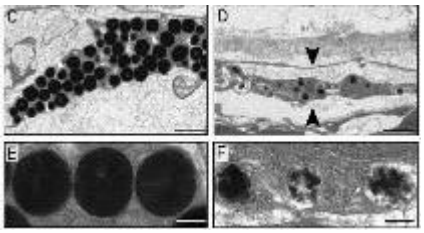


Figure 1. *This image is originally extracted and broken horizontally into 16 images. Our algorithm combines it accurately into one image.*

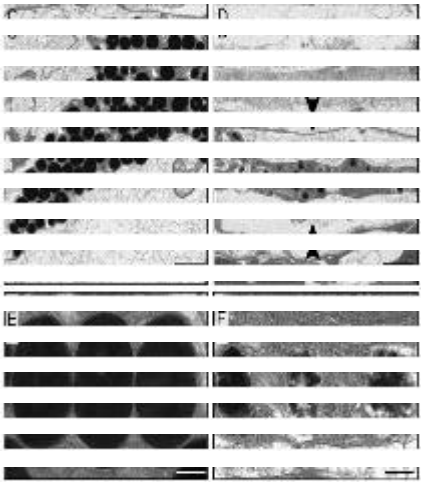


Figure 2. *Shows the 16 separate pieces of the image in Figure 1 as extracted originally from the PDF document.*

Figure 3. *This image is combined by mistake; comprising it are two separate but identical images that happened to occur sequentially.*

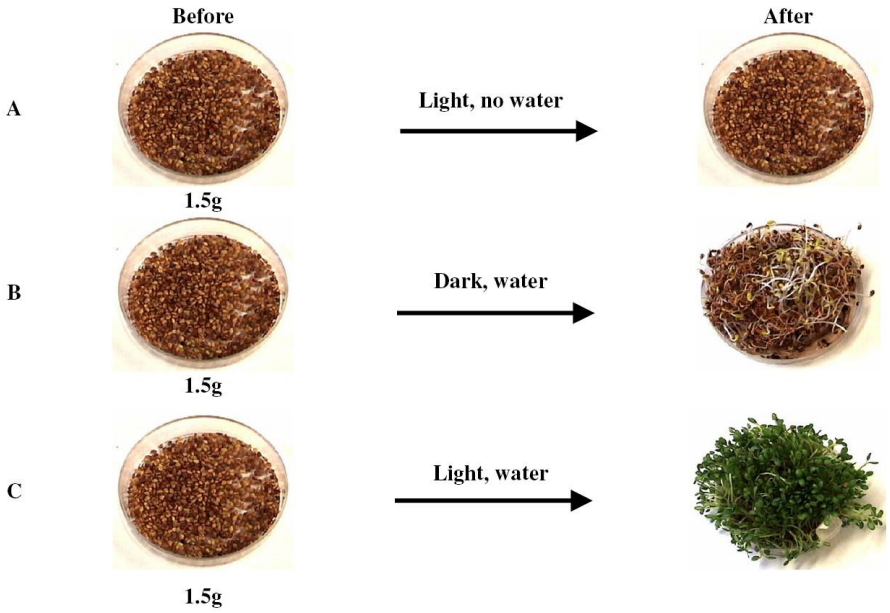


Figure 4. *This image shows the radish seeds as they are originally in the PDF document. The algorithm incorrectly combines the two images in the first row.*

Much of our effort in this stage of the project goes into combining the text fragments and images, which we extract from the leaves of the document tree, into meaningful blocks. This includes combining split words, combining words into lines, and identifying the place of each image in the flow of the document by inserting an image tag in the text at the place where the image appears.

One of the problems that we run into during this stage is losing emphasis information (e.g. bold and italics). Emphasis is lost because of the many different ways it is stored in the document tree, which Multivalent constructs. It is usually found in a comment attached to the text leaf. However, it can be represented as a different font, or the same font with different weight, or often, the name of the font is blank, which causes a loss of font information. Another problem is that special symbols such as Greek characters and arrows (in chemical equations) appear as the question mark character in the extracted text. Special symbols are lost because text is extracted in ASCII. In addition, superscripts and subscripts appear as normal text.

After the algorithm has extracted the text and organized it into words and lines, we use the list of extracted text lines to identify the text of each question. To do this we go through a four-step process.

1. We do a rough combining of text lines into blocks, each of which contains the text of one question. In this step we keep image tags separate from the text of questions, because an image can be a part of multiple questions.

2. We associate the text blocks that contain image tags with the text blocks containing a question related to the image.

3. We identify common errors, and attempt to fix them.

4. We identify each block of text as a separate question and classify it as one of 5 common question types – multiple choice, true/false, fill-in-the-blank, short answer, or image hot spot (a question which asks the student to identify something in an image).

The first thing to do once the text has been extracted is to organize it into blocks that contain one question each. To identify question content we use several logical cues that indicate the start of a new question. Our algorithm considers that a new question starts based on these events: a new page starts; a new question label is found (Question labels are in the form of numbers, followed by some punctuation mark like a period or a right parenthesis, or a space); or some pre-determined amount of white space is found in between two lines of text. Also in this step of the algorithm we identify any answer keys that might be present in the file and exclude them from the list of questions. In this step we also identify an image tag as starting a new question. Each image tag normally ends up in a separate text block, and only occasionally has some text appended to it, such as a label or an

explanation (e.g. “Figure 6.1”). Leaving image tags in separate blocks from the rest of the text allows us to associate a single image with more than one question in the next step of the algorithm. At the end of this step, the algorithm has reorganized the list of text lines into a list of text blocks, each of which roughly contains either an image tag, or the text of one question.

Once the text is organized into blocks we attempt to link each image with the questions that use it. We observed common features of a set of 64 real assessments, and found it typical for images to precede the question(s) to which they were related. So in this second step of the algorithm, we link images with the question(s) immediately following them. In our algorithm we keep a list of images available for association. Every time we encounter a question which indicates that it is related to an image, we associate the images in the list (usually only one image) with the question. Words such as “figure” and “picture” in the text of the question, as well as sentences starting with the word “which”, are normally found to be related to an image, and these are the questions to which we attach an image from the list. Every time the algorithm encounters a question that does not contain any image-associating cues, it clears the list, based on the assumption that an instructor asks questions about an image immediately after showing the image, and once one unrelated question is asked, no more questions about that image will follow.

After linking images with the questions that use them, we're almost ready to identify the type of each question, which is the final goal of the extraction algorithm. But in testing the algorithm we noticed the occurrence of several common errors. One major error was separating a question’s text into two text blocks, either because of an image or because of some amount of white space which occurred in the middle of the question's content. This was common with multiple choice questions in which an image or white space appeared between the question and its answer choices. Thus, after linking images with questions, our algorithm identifies these errors and fixes them. It identifies split questions and combines the two parts of the question into a single block. In addition, in some assessments a set of answer choices is used for more than one question. We attempt to associate these choices with each question. We observed that instructors tend to format a group of related questions in the following manner:

For questions 14-16 use the following options to indicate the correct answer:

A. (text of choice A)
B. (text of choice B)
C. (text of choice C)
D. (text of choice D)
E. (text of choice E)

14. (Text of question 14)
15. (Text of question 15)
16. (Text of question 16)

Our algorithm identifies these occurrences and makes the proper associations. The output of the algorithm for these three questions will be:

14. (Text of question 14)
A. (text of choice A)
B. (text of choice B)
C. (text of choice C)
D. (text of choice D)
E. (text of choice E)

15. (Text of question 15)
A. (text of choice A)
B. (text of choice B)
C. (text of choice C)
D. (text of choice D)
E. (text of choice E)

16. (Text of question 16)
A. (text of choice A)
B. (text of choice B)
C. (text of choice C)
D. (text of choice D)
E. (text of choice E)

After going through the extracted text three times and organizing it into text blocks which contain one question each, the final step of the algorithm looks at each text block's content and attempts to classify it as one of 5 basic question types. Each question type has common elements which appear consistently in the majority of the questions of that type. For example, multiple choice questions contain answer choices which are normally labeled with letters (e.g. “A. B. C. D.”). Fill in the blank questions contain underscores for representing the blank space. True or false questions contain some form or abbreviation of the words true and false (e.g. “T/F”, “True or false”, “T F “, “T or F”, etc.). We use these simple cues to identify the type of each question.

Results

We chose 50 documents from our collection of 64 by randomly excluding 14 documents. For each document we recorded six counts: the number of real questions (manually counted and sometimes ambiguous), the number of extracted questions, the number of questions the algorithm got correct, incorrect, and missed, and the number of extra questions. A question was considered correctly extracted if its type was correct and its text was mostly retained Multiple choice questions with some of the choices truncated could be correct, along with questions missing directions. Questions meeting the above conditions but with extra text or incorrectly associated images (missing or extra) were also considered correct. Missed questions were real questions that were not extracted. Extra questions were not real questions in the original document, but were nonetheless extracted by the algorithm in error. The sum of correct, incorrect, and missed questions equals the number of real questions; the sum of correct, incorrect, and extra questions equals the number of extracted questions. Using these six numbers, we calculated four percentages for each exam: percent correct, extracted, missed and extra. These four percentages were averaged among the 50 documents, meaning that each document was weighted equally, regardless of its number of questions.

We reached a success rate of 86% in correctly extracting questions. The success rate has been improved from 75% from the original version of this method (the abandoned method had a success rate of 60%). In addition, 91% of the questions were at least extracted. The two other averages are the percentages of missed and extra questions, at 9% and 100%. Because of one assessment that contained only one question in a table format and for which sixteen extra questions were identified, the extra questions figure is inflated. Many of the other extra questions come from portions of the assessment which contain instructions and/or formulas. The number of extra questions identified could be significantly reduced by asking the user to exclude those pages containing no questions. With the assessments weighted by number of questions rather than equally, the percentage of extra questions identified was 50%. (Is this ok, because of the changed paragraph above?)

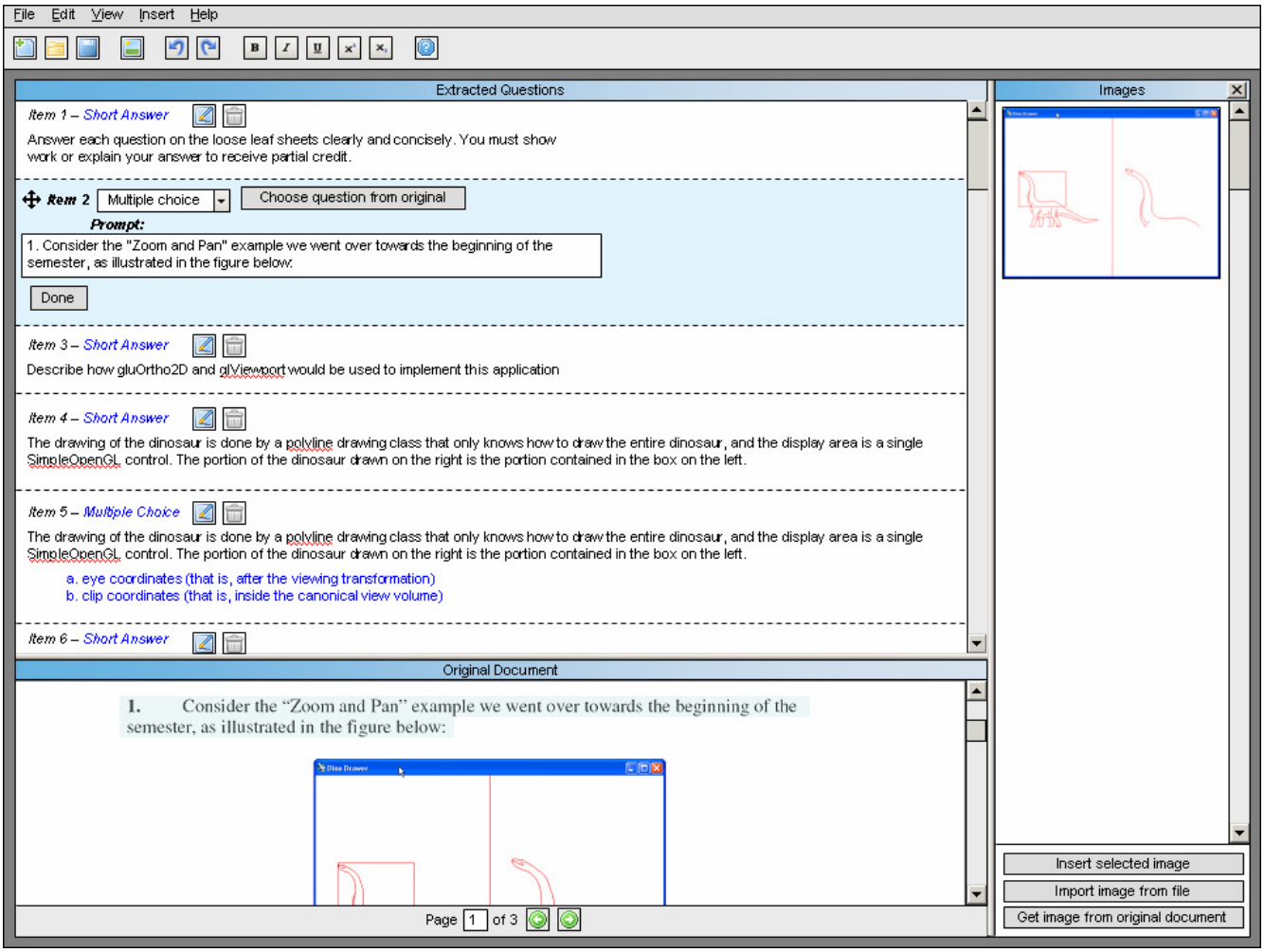
Analysis of the results

There are variations in the sample because of the great variations in the formatting of each exam. In addition, some exams contain questions that we could not classify or extract reliably, such as “Fill in the table”. There are a few problems likely responsible for these results, the major one being vector graphics not being accounted for. The problem with vector graphics objects such as tables, and graphs, is that the text which they contain appears in separate leaves in the document tree. This makes restoring the vector graph a difficult task, one that often produces inaccurate results. The algorithm we wrote to handle these often stores the text of a nearby question in the vector graphics objects. The other questions, which are not extracted, came largely from cases we could not handle, such as tables. Questions for which it was difficult even for a human to properly decipher (e.g. a short answer question with several vaguely related or unrelated parts) are also commonly missed. For well-formatted consistent exams, the algorithm is able to extract all of the questions; it extracts all questions for 50% of the files in the sample. Another unresolved issue is the extraction of instructions as questions, which results in many extra questions for some exams; the percentage of extra questions closely resembles the ratio of instructions to questions in the assessment instrument. This problem could be alleviated by allowing the user to exclude certain pages of the document from the extraction process (instructions tend to appear on the first and last pages of an exam). What would further increase accuracy would be to research more in depth the use of white space as a means for separating questions. By changing the amount of white space, which is a criterion for separating questions, one can vary the outcome of the algorithm; but in general increasing the threshold white space dramatically decreases the number of extra questions, while it decreases the accuracy of extraction of true questions. The first version of the algorithm recognized 75% of the questions. Its accuracy was gradually improved mainly by attempting to link images to questions (which contributed to recognizing the question type correctly) as well as attempting to merge parts of the same question, or split a text block in two in the cases when it contained more than one question (which contributed to extracting more questions accurately).

User interface

Because of the algorithm's inaccuracy, it was necessary to create a tool through which an ordinary, everyday user could correct mistakes made in question extraction. In order to ensure user-friendliness, we designed it according to the Principle of Least Astonishment, which states that the best option is that which is least surprising to the user. In other words, it is best to cater to the user's expectations as much as possible. Some design problems were solved by simply asking users how they would expect the program to behave. So that a new user wouldn't have to learn the program from scratch, we borrowed many interface elements from existing applications. For instance, the toolbar and menu bar are purposefully designed to be very similar to those in Microsoft Office. This way, the user's prior experience can provide a great deal of the necessary knowledge.

In order to catch design flaws early on, the interface was designed in the form of paper mock-ups and tested with actual users before implementation began. Participants were given a series of tasks to perform and were asked what they would do (i.e. where they would click) in each situation. Eight users were tested in total – four were novice or average computer users, and four were advanced computer users. In most cases, the advanced computer users had very little trouble understanding the design, while the less experienced users stumbled on some of the more ambiguous elements. After testing, the design was critiqued and revised as necessary, based on user responses. This process helped us identify and solve a number of design problems.



The final mockup for the interface design, shown above, is broken down into three panels: one for the extracted questions, one for the imported PDF document, and one for any images that were found in the PDF. All three are surrounded by a dark gray background (intended to mimic Microsoft Office products such as Word or PowerPoint), which subtly suggests to the user where the interface ends and the editable data begins.

The extracted questions pane, which takes up the majority of the window, displays the questions that the parser has identified. It is designed to present the user with all the information they need to see a mistake, so they can quickly scan the results without having to click anything. When a user does identify a mistake the parser has made, he/she can click the edit button (pencil icon) to put the question in "edit mode". In the above mockup, item 2 is in edit mode while items 1, 3, and 4 are in view mode.

The original document pane displays the imported PDF file. When a user selects a question in the extracted questions pane, the corresponding text is highlighted in the original document panel. This helps the user understand where many of the parser's mistakes come from – incorrectly separating the questions. The user can fix this mistake by clicking "Choose question from original" (next to a question in edit mode) and selecting the correct text.

The image pane (only visible after the user has clicked the image button in the toolbar) displays all of the images that were extracted from the imported PDF. Users can drag and drop images to the appropriate questions or choices. In addition, if an image was not extracted correctly (a common problem for tables and graphs), the user can drag a box around a section of the PDF and add its contents to the image pane.

Conclusion and Future Work

In this paper we have described a method for importing questions from assessment instruments into a database. Our algorithm achieved a success rate of over 86% with correct question extraction. Drawbacks to our method include not extracting special symbols, extracting superscripts and subscripts as normal text, and the high number of extra questions which come from instructions. Future work on this project should be focused on reducing the number of extra questions. This can be achieved by allowing the user to exclude certain pages from the extraction process as well as by experimenting with the white space threshold criteria for separating questions. The accuracy of the extracted text could be improved by developing a method for extracting vector graphics objects, such as graphs and diagrams, as well as by extracting special characters. Another are in which the method can be improved is recognizing composite question types and extracting more uncommon or difficult question types, such as fill-in-the-table and matching. In terms of improving the accuracy of the extracted text, work can be done to extract superscripts and subscripts correctly, as well as to extract special symbols.

Many features of the user interface have yet to be implemented, including text formatting options, special characters, question and choice re-ordering, options to designate correct answers, and the tools for interacting with the parser. In addition, it requires polishing. In particular, the visuals need to be improved to match or exceed the aesthetic appeal of the mockups. When these features are implemented, further user testing should occur. It is likely that many changes will need to be made to how the user interacts with the parser and the PDF document, to engineer a more efficient work flow.

References:

[1] Chao, H., Fan, J., "Layout and Content Extraction for PDF Documents". *In proceeding of IAPR Int. workshop on Document Analysis Systems*, 2004

[2] Phelps, T. A. and Wilensky, R. 2001. "The multivalent browser: a platform for new ideas". *In Proceedings of the 2001 ACM Symposium on Document Engineering* (Atlanta, Georgia, USA, November 09 - 10, 2001). DocEng '01. ACM Press, New York, NY, 58-67

[3] *IMS Question & Test Interoperability: ASI Best Practice & Implementation Guide Date.* (2002). Retrieved August 2, 2007 from IMS Global Learning Consortium Web site: http://www.imsglobal.org/question/qtiv1p2/imsqti_asi_bestv1p2.html