Lessons on Using ITS Data to Answer Educational Research Questions

Cecily Heiner, Joseph Beck, Jack Mostow

RI-NSH 5000 Forbes Avenue
Pittsburgh, Pennsylvania 15213 USA
{cecily, joseph.beck, Mostow} @ cs.cmu.edu

Abstract Some tutoring system projects have completed empirical studies of student-tutor interaction by manually collecting data while observing fewer than a hundred students. Analyzing larger, automatically collected data sets requires new methods to address new problems. We share lessons on design, analysis, presentation, and iteration. Our lessons are based on our experience analyzing data from Project LISTEN's Reading Tutor, which automatically collected tutorial data from hundreds of students. We hope that these lessons will help guide analysis of similar datasets from other intelligent tutoring systems.

Keywords: intelligent tutoring systems, log mining, guided oral reading,

1 Introduction

In this paper, we give a short history of Project LISTEN data mining and discuss lessons on four steps of converting data to ideas: design, analysis, presentation, and iteration. We illustrate our lessons with anecdotes from our experience writing a paper for InSTIL/ICALL (Heiner *et al.*, 2004), a symposium on Computer Aided Language Learning. The ICALL paper presents a study to improve the help selection policy in the Project LISTEN Reading Tutor. The study compares various help types (e.g. saying the word, giving a letter to sound rule, or rereading part of the sentence). The goal of the study is to identify effective help types based on the students' performance after help is given.

Project LISTEN's Reading Tutor has also collected data, with and without specific planned experiments in mind, and stored it in a database. The Reading Tutor uses automatic speech recognition to listen to children read aloud (Mostow & Aist, 2001). The 2002-2003 version of the Reading Tutor was used by hundreds of students who read millions of words in nine schools. The Reading Tutor automatically logged information about each tutoring session in a database as described in (Mostow *et al.*, 2002). In addition to the automatically collected data, the database contains some manually entered data such as test scores.

There has been prior work in broad data collection without a specific experiment in mind. For example, the AnimalWatch tutor for mathematics (Woolf *et al.*, 2001) recorded student interactions at a fine-grain size which led directly to research on estimating student performance at solving math problems (Beck & Woolf, 1998, 2000; Beck *et al.*, 2000). The logging was also generic enough to support future, and at the time unthought of, research on computing the student's zone of proximal development (Murray & Arroyo, 2002) and performance of the pedagogical module (Arroyo *et al.*, 2003). Other examples of this idea include the REDEEM authoring tool (Major *et al.*, 1997), where how much time students spent on a page was collected in order to provide for reports to teachers. These timing data were eventually used to compare how REDEEM's teaching differed from traditional CBT (Ainsworth & Grimshaw, 2002).

2 Project LISTEN Data Mining History

Project LISTEN encountered a number of bottlenecks in mining data from the Reading Tutor, and explored various ways to overcome them.

One set of bottlenecks involved how to collect, harvest, transport, and pool data from the Reading Tutor. Solutions to these problems varied from one school year to the next as Reading Tutor deployment grew.

- 1996-97: Eight third graders used the Reading Tutor on a single computer, which logged all data directly to a 1 gigabyte (GB) Jaz ™ disk. Field staff replaced the disks monthly.
- 1997-98: Ten classrooms at one school used the Reading Tutor, one computer per classroom. Each computer had a CD burner, which field staff used to archive Reading Tutor data on-site. About half the CD burners failed over the course of the year in the dusty classroom environment.
- 1999-2000: 15 computers at two schools ran the Reading Tutor. Field staff copied log data to SCSI disk drives every few weeks. The Reading Tutor used a database to record events.
- 2000-2001: 33 computers at three schools ran the Reading Tutor – 20 in two labs, 12 in classrooms. Newer computers had disks large enough to store a full year's data (roughly 10 GB). Each Reading Tutor sent back a nightly summary over the Internet via FTP, because there was too much data to send all of it back by this route without exhausting network bandwidth. However, they did send back selected utterances for one automated experiment.

Another set of bottlenecks involved analyzing the voluminous Reading Tutor logs. The logs were intended largely for debugging purposes. They included output of diverse form and content from hundreds of different points in the application code, rendering them both ill-structured and long, with hundreds or thousands of lines per minute of tutorial interaction. Each launch of the Reading Tutor generated a separate log file, typically megabytes in length. Consequently, data about an individual student were spread over many different log files, making it difficult to trace the consequences of a tutorial action. Such detailed logs sometimes provided helpful evidence for debugging. However, they were too long for humans to read in quantity, and their non-uniform structure made them tedious to parse for purposes of automated analysis. Such analyses used Perl scripts to parse log files, scan for specified patterns of events, and aggregate across multiple log files. The scripts were long, slow, and complex enough to put their correctness in doubt.

3 Design

Now all of our data is stored in a database. The database structures our data, which makes analysis quicker and more accurate. Our data is designed as a set of twenty two databases. A few of the databases contain lexical resources used by the Reading Tutor such as a dictionary. More than half of the databases contain data derived by individual researchers. Each researcher has a database to store tables related to their individual research. These researcher databases are dynamic; they change with shifts in research focus, contain abstruse variable names, and abstract away data not relevant to the current investigation. If an individual develops a new and exceptionally useful way of organizing and grouping the data, then the databases containing the raw data are meticulously updated to reflect the new insight. However, separating individual research data from raw data allows us to protect the raw data from many accidents while preserving individual creativity.

The remaining databases contain the raw data (data logged directly by the Reading Tutor) with one database for each academic year. For example, data from the 2002-2003 academic year is stored in one database, and the 2003-2004 data is stored in a different database. We release a new version of the Reading Tutor with major changes each summer, and we release smaller bug patches during the academic year, but the bug patches do not change the functionality of the tutor. Grouping data by academic year insures that all data in a database are from the same general version of the Reading Tutor.

Some of our tables in the raw data databases, e.g. the Aligned Word table, Utterance table, and Sentence Encounter table contain information about student sessions at increasingly larger grain sizes, starting with a student's attempt to read a word and increasing to include all of a student's attempts to read a sentence. As the grain size of the data increases, the number of rows in the table decreases. Other tables contain information

about different kinds of interactions such as multiple choice questions, help given, and tutorial steps students take. Some data, such as the original audio, is stored outside of the database, but linked to the database. Some of the common variables in the tables are:

Machine Name: a string identifying the computer
User ID: a string identifying the student
Start Time: a timestamp indicating when the specified event (e.g. utterance) began
End Time: a timestamp indicating when the specified event ended
Target Word: word student should have read
Aligned Word: word recognized by automatic speech recognition
Word Index: position of the word in the sentence
Latency: time between previous word and this word
Type: help type given (e.g. SayWord)

Some people may argue that our database is poorly designed because we have redundant data in several places, e.g, storing the User ID in the Sentence Encounter table, Utterance table, and Aligned Word table. However, this redundant data helps us in two ways. First, some of the tables, such as Aligned Word, have extra fields for efficiency. Theoretically, we could obtain the User ID through a join with the Utterance table. Such a join is trivial on a small table, but very time-consuming for a table containing more than five million rows of data. Consequently, we have several extra fields in the Aligned Word table. Second, some redundant fields aid in recovery from error. For example, some of our tables contain extra timestamp fields to allow for improperly terminated sessions. Redundant information about machine and the user in many of the tables occasionally allows us to detect login errors, un-terminated session errors, and time-out errors.

Experiments can be designed before or after data collection. Pre-defined experiments have pre-defined outcome variables. They may also have controls and/or randomized experimental manipulation. Pre-defined experiments may have independent trials in which the outcome of one trial does not depend on the outcome of another trial. Unfortunately, pre-designing an experiment properly may require a code change in the intelligent tutoring system as well as additional constraints. Code changes are expensive because they require development time. Because pre-defined experiments are expensive, we have limited opportunities to run them. An alternative is to perform post-hoc experiments on data after we have collected it. These experiments may leverage the randomized experimental manipulation when making tutorial decisions. However, because the experiments were not pre-defined these analyses can be plagued with confounds (individual outcomes affected by several variables) and masks (elements outside of the experiment which affect the outcome variable). Many of our analyses have characteristics of both kinds of experiments.

One example of a design with both kinds of characteristics is our ICALL analysis. We had a pre-defined, randomized experimental manipulation. The Reading Tutor used a specified probability distribution to randomly choose a help type. We had independent trials because the Reading Tutor randomly selected a help type each time the student clicked for help. However, we chose our outcome variable after the data had been collected. Specifically, when a student receives help on a word, the automatic speech recognition acceptance at the next encounter of the same word is the outcome variable. Furthermore, when the student clicked for help several (n) times on a single word, then later help events (2..n) may have masked the effect of the first help event. Although our experiment design was not perfect, it was sufficient to begin analysis of the data.

4 Analysis

Identifying inaccurate or incomplete data is the first step of analysis. Common causes of inaccuracies are bugs and crashes. Three common kinds of inaccuracies in our data are outliers, missing data, and inconsistent data. Outliers are data that lie outside the expected distribution. Some human judgment is necessary when examining the expected distribution and identifying outliers. Outliers contaminate the mean, so in addition to examining the mean, we check the values of the max and the min. Our colleagues have also suggested examining the first and third quartiles or the median; these measures are also more resistant to outliers. Sometimes the database contains unexpected values, such as undefined or null, or strings containing unexpected characters. For example, to denote a second pronunciation in our database, we put the word followed by a set of parenthesis with a number to indicate alternate pronunciations. When we do not handle these values properly,

they can easily lead to "missing data." For example, we may accidentally collapse all pronunciations for a word into one unit instead of one unit per pronunciation.

We cannot inspect each individual row of data in a large dataset to insure that we are handling it properly. However, we can quickly examine the size of a dataset (the number of rows). If there is missing data, then the dataset will be smaller than expected. There may be times when we don't know exactly how many items should be in a dataset. When we can't calculate the exact size of a derived data set, we can consider the order of magnitude of the dataset and the consistency with other datasets. If we are expecting a dataset that contains hundreds of items, and the result set contains thousands of items, there may be a problem. Additionally, if the appearance of a dataset is dramatically different from a similar dataset without reason, then there is a conflict between the two datasets. Conflicting datasets may not be accurate and should not be used for analysis.

We encountered each kind of inaccuracy in our ICALL analysis. For example, we did a quick analysis to show the average duration of each help type. To expedite the analysis, we did not consider fractions of a second. We initially found that the average length of the SayWord help type, which plays the audio recording for a single word, was more than eight seconds. Dubious, we queried the database for the minimum and maximum duration in seconds. The minimum was 0 seconds, which is believable for short one syllable words which took less than half a second to say. However, the maximum duration to say a single word was 4.99 days. Part of the tutoring system (probably part of the logging subsystem) had failed, creating an outlier in our data. In another part of the analysis, we had to determine which words were accepted or rejected by the speech recognizer. Our initial attempts at this task involved string manipulation and matching. At one point, we added up the counts for each category of words we were including in our analysis. We found we were 5,637 words short of the number we were expecting. We examined a small sample of the data, and we realized that we were not including homographs. Homographs are words with the same spelling but different pronunciations. In our database, we represent homographs with an extra set of parentheses after the word and a number indicating the number of alternative pronunciations, e.g. READ(1) and READ(2). We had failed to allow for the parentheses, but by examining the counts, we were able to detect a problem in our queries. In another exploratory analysis for the ICALL paper, we wanted to consider the position in a sentence of a word that a student clicked on in order to receive help, and analyze how it relates to help type. We believed that there would be more words in the earlier sentence positions because sentence length varies, but we didn't know exactly how much sentence length varied. We were concerned about a steep drop in number of words occurring after the first position in the sentence. To confirm our suspicion, we plotted the number of words in each sentence position for two different years of data. Figure 1 shows our graph and the much steeper drop for the 2002-2003 data. Until we can find a change in the Reading Tutor that justifies such a dramatic change in word position frequencies, we cannot use word position in our analyses because we are not sure the data are accurate.

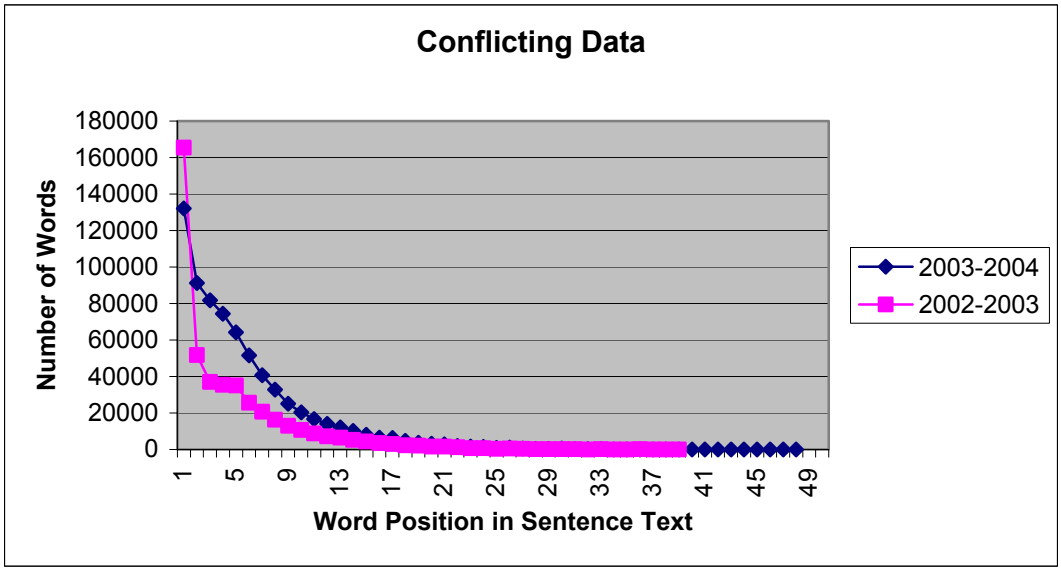

Figure 1: Inconsistent Data

Much analysis can be completed simply with Structured Query Language (SQL), a database programming language. Compared to traditional programming, queries can be quicker to write and easier to

debug. Queries are powerful because they abstract away data structures, allowing the programmer to focus on algorithms. The programmer doesn't have to write a data structure to parse the data, store the data, structure the data, display the data, or perform rudimentary data manipulations; data structure and display are provided in the tables and SQL. When the programmer is debugging queries, viewing the data in the tables is relatively simple. In Figure 2, we show a sample query that lists the help types and their efficacies according to student ability and word difficulty.

```
create table Cecily.abstract_pivot
select student_ability, word_difficulty, Type, AVG(correct) as mean,
STDDEV(correct) as SD, COUNT(*) as N
from Cecily.abstract_prepivot
group by student_ability, word_difficulty, Type;
```

Figure 2 SQL for ICALL analysis

Sometimes, simple SQL is sufficient, but other times it is not adequate. Some experiments require us to write query-generating programs. These programs may belong to one of two major categories, syntactic tweaks and simulation. One common example of syntactic tweaks is cross-validation code. In a cross validation, the data is divided into two sets, a model is trained on one set of data, and the model is tested on the remaining data. This process is repeated for each fold of the cross validation. For example, for a three fold cross-validation, the data would be divided into three sets, and each set would be used as the test set once.

Simulations model a process or how things change over time. For example, a learning curve measures student performance over time. Another kind of simulation is knowledge tracing (Corbett & Anderson, 1995). Knowledge tracing models the acquisition of procedural knowledge. When writing query-generating programs, reducing developer time should remain a high priority. Some procedures are still faster to implement in SQL. For example, calculating averages or doing a two fold cross-validation is faster in SQL because a two-fold cross validation can be done with four queries (two for the training set and two for the test set), and it is faster to write four queries than to write code to generate two kinds of queries (one for the training set and one for the test set). When query-generating code is required, we have generally found that writing generic code for special-purpose tables is better than writing special code for generic tables. For example, separating data into folds for a cross validation is easier and faster in SQL than in the code for a query-generating program. In SQL, this change requires one query "update 'table_name' set 'fold'= 'new foldnumber.'" To do the same task in a query-generating program, at least two lines of code would be needed; one to construct the query, and one to execute the query. Similarly, storing information in variables within the program is faster than querying the database, writing the values back to the database, and querying the database again.

Part of our ICALL analysis required a cross-validation. We initially tried a 270 fold cross validation with one fold for each student. We wrote code that built two tables, one for the training sets and another for the test sets. This code was slow because several queries went back and forth from the database, and we eventually added so many queries that the program ran out of memory before it completed execution. At one point, we were running the code in installments with a counter built into the program to skip the first several students whose information had already been added to the tables. We spent most of a day running this code once! Unfortunately, with so many folds, many of the folds contained incomplete or poorly estimated data. Thus, we had to change to a cross validation with fewer folds, where each fold contained more students. When we changed our approach, we also changed our code. This time we reduced the number of queries in the code and stored as much information as we reasonably could in program variables. We also changed the tables we accepted as input. Before, each acceptance or rejection of a word was an individual row in the database; now we aggregated the rows so that each row contained a count of accepted and rejected words for each student. We also added a field in the table for the fold (or set number) of the cross validation, so we could easily change how many folds we were using in our cross validation. We wrote the new code in less time than the old code, and the new code was more readable and easier to debug. The new code was also shorter with 120 lines instead of 180 lines. As an added bonus, the program completed execution in about fifteen minutes. Later, when we wanted to change the dimensions of our analysis, we could alter the tables using SQL instead of changing the query-generating code. For example, we could switch from a two fold cross-validation to a ten fold cross-validation by changing the fold field in the input table.

5 Presentation

Simplicity, clarity, and reproducibility are important in presenting research. Although statistics are not easy to understand, they can convey simply a complex idea such as the probability that results are reproducible. We can often compute simple statistics using the same software package that we used to run our experiment. For example, WEKA (Witten & Frank, 1999) will output a confidence matrix. We can calculate the mean, standard deviation, sample size, and mode with relative ease directly in SQL. However, sometimes we need to run more complex statistical tests or procedures such as a regression analysis. When we need to compute complex statistics, we usually use SPSS (SPSS, 2000) or Excel (Excel, 2000). We use SPSS for especially complex statistics and statistics that we can't do in Excel. SPSS facilitates complex statistics because the program allows the user to save a syntax file that can be loaded later, and used to rerun the exact same procedure on a dataset. We also use SPSS for analysis of large datasets because Excel has a limit of approximately 65,000 rows, and some of our data sets are larger than that. We use Excel when the statistics are easier to compute in Excel. Applying formulas to sets of numbers (e.g. filling column C with the sum of the values of the cells in rows A and B) is generally faster in Excel than SPSS. Excel is also useful for generating charts and graphs such as the graph in Figure 1.

Excel and SPSS can simplify work, but they can also make work more complicated. For example, we use an older version of SPSS that restricts variable names to eight characters or fewer. If we want to be able to refer to a variable using the same name in SQL, SPSS, and Excel, we are restricted to the shortest name length, eight characters. Consequently, we may use shorter names than necessary in SQL, so we can use the same name in SPSS. Another example involving names is our attempt to replicate our preliminary ICALL results. We initially used SPSS for the analysis. Later, we realized that we could have completed the same analysis more quickly and easily with SQL and Excel. One critical aspect of the analysis involved rounding the student levels and word levels to the nearest whole number. SPSS rounded in the traditional sense, rounding everything .5 and greater up and everything less than .5 down. When we tried to repeat the analysis in SQL using the round function, we were appalled to discover that .5 rounds down in our version of SQL, and the exact implementation depends upon C libraries, which vary from machine to machine (2004). This is one example of different semantics for the same name in different programs.

Finally, results must be reproducible. One tool to facilitate reproducing results is a repository. We use a Visual Source Safe (VisualSourceSafe, 1998) repository to store our papers and our programs. We also store our queries, syntaxes, important data sets, and anything else that might be necessary to reproduce our results. Storing is especially important for empirical studies involving data, especially when that data comes from students, may involve outliers, and the nature of the experiment may involve many changes in scope of the problem. Individual students may be included or excluded from a study because there is insufficient data about that individual. Outliers may be excluded from the data because they result from system crashes. Other details may mean that other parts of the dataset must be excluded. Remembering all of the reasons that different pieces of data were included or excluded in an experiment without a recorded, organized, stored explanation is impossible. Because researchers using a database independently may not need to synchronize their code changes with each other, repository software may initially appear less necessary than it is for a project where participants are actively writing code that needs to be synchronized. However, over time, the dataset, the queries, and the code used in an experiment will change significantly and reproducibility will suffer without a repository because the researchers will not be able to remember exactly which pieces of data were used and were not used.

Before beginning the investigation for the ICALL paper, we reproduced the results from a similar study reported in (Aist & Mostow, 1998) . Reproducing an experiment that we were familiar with helped researchers new to the project learn rudimentary research skills. A familiar problem simplified advising for more senior members of the team. Because we could verify the results of our experiments against the previous results, we did not need to worry about verifying the accuracy of the results, and we were free to focus on the methods.

6 Iteration

The product of research is ideas, and these ideas suggest new experiments, which produce new data and new ideas. Our research cycle follows the academic year because we release major changes to the functionality of the Reading Tutor during the summer and collect data during the academic year.

In our most recent cycle, we have published many papers based on different subsets of the data from the 2002-2003 school year. We now discuss three sets of those papers here. First, we explain how each paper is a refinement of an idea from an earlier paper and show how changing an aspect of the intelligent tutoring system or collecting more data altered the experimental design; then we compare the datasets and methodologies. First, (Mostow *et al.*, 2003) used automated emotional scaffolding as a refinement of the study (Aist *et al.*, 2002) which examined the effect of emotional scaffolding from a human wizard. The earlier study compared a few dozen trials from fourteen students where each trial consisted of reading a list of words or limericks. The later study compared 5, 695 trials where each trial is a series of up to five spelling tasks. The later experiment had an order of magnitude more data than the earlier experiment because all of the data could be automatically collected without human intervention. The later study has fewer trials than some of the other automated experiments because the experiment required a code-change and an activity that interrupted the primary activity of the Reading Tutor, which is reading stories aloud. An example of an experiment with data collected while students were reading aloud is (Heiner et al., 2004). This experiment analyzed 191,497 help events compared to 62,645 help events studied in (Aist & Mostow, 1998). In the later study, we collected enough data to disaggregate the data into smaller subsets based on student and word level, a disaggregation that was not possible with less data. In addition to supporting refinements of older studies, automatic logging makes possible new kinds of experiments that are not practical to run by hand. For example, (Beck *et al.*, 2003) examined 2.4 million word encounters and used them to predict student proficiency. This study considered individually read words, the finest grain-level available in Reading Tutor data. Manually examining or evaluating this much data would not be practical.

7 Conclusion and Future Work

We have shared a series of lessons and anecdotes and discussed some difficulties and benefits of a large dataset collected automatically by an intelligent tutoring system. Specifically, we have shown that designing experiments with automatic logging may require code changes. However, automatic logging allows collection of larger datasets than manual logging. Additional experiments may be designed after the data has been collected; however, these experiments will often require refinements of the dataset. We compared and contrasted SQL and query generating programs and suggested situations in which each is appropriate. We also discussed various tools for presenting and storing research results. We have shown that studying large empirical datasets can result in publications and answers to educational research questions that are not possible with smaller, manually collected datasets.

However, there are many unanswered questions about empirical methods for intelligent tutoring systems. First, it would be nice to have a generic database schema that can be applied to many or all intelligent tutoring systems, thus simplifying the process of replicating results in various domains and on different datasets. More work on designing experiments could shed light on how to produce cleaner data sets and better data logs with less researcher supervision and analysis. Additional work is also needed to identify and answer research questions that require large datasets collected over an entire year. Possibly, year-long datasets with hundreds of hours of tutorial interactions per student will enable researchers to answer larger grained educational research questions about learning tools and curricula; however, the cost of developing and supporting systems with integrated, year-long curricula and automatic data logging remains prohibitive, and these systems are still somewhat scarce. Analytical methods (e.g. knowledge tracing) for intelligent tutoring system data are few in number. New ways to model what the student has learned at finer grain sizes than the traditional pre- and post-test would be useful for measuring the efficacy of individual tutorial actions. Understanding the effect of individual tutorial actions may in turn lead to a greater understanding of how tutorial actions interact to affect students' learning.

Acknowledgements

This work was supported by the National Science Foundation, under ITR/IERI Grant No. REC-0326153. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the official policies, either expressed or implied, of the sponsors or of the United States Government.

We thank other members of Project LISTEN who contributed to this work and the students and educators at the schools where Reading Tutors recorded data. We also thank Karen Wong, Ryan Baker, and Jonathan Brown for discussions about this paper.

References

Ainsworth, S., & Grimshaw, S. (2002). Are ITSs Created with the REDEEM Authoring Tool More Effective than "Dumb" Courseware? *Sixth International Conference on Intelligent Tutoring Systems*, 883-892.

Aist, G., Kort, B., Reilly, R., Mostow, J., & Picard, R. (2002, June 4). Experimentally Augmenting an Intelligent Tutoring System with Human-Supplied Capabilities: Adding Human-Provided Emotional Scaffolding to an Automated Reading Tutor that Listens. *Proceedings of the ITS 2002 Workshop on Empirical Methods for Tutorial Dialogue Systems*, San Sebastian, Spain, 16-28.

Aist, G., & Mostow, J. (1998, March). Estimating the effectiveness of conversational behaviors in a reading tutor that listens. *Working Notes of the AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, Stanford, CA.

Arroyo, I., Murray, T., Beck, J. E., Woolf, B. P., & Beal, C. (2003). A formative evaluation of AnimalWatch. *Eleventh International Conference on Artificial Intelligence in Education*, 371-373.

Beck, J. E., Jia, P., & Mostow, J. (2003, June 22-26). Assessing student proficiency in a Reading Tutor that listens. *Proceedings of the 9th International Conference on User Modeling*, Johnstown, PA, 323-327.

Beck, J. E., & Woolf, B. P. (1998). Using a Learning Agent with a Student Model. *Fourth International Conference on Intelligent Tutoring Systems*, 6-15.

---. (2000). High-level Student Modeling with Machine Learning. *Fifth International Conference on Intelligent Tutoring Systems*, 584-593.

Beck, J. E., Woolf, B. P., & Beal, C. R. (2000). ADVISOR: A machine learning architecture for intelligent tutor construction. *Seventeenth National Conference on Artificial Intelligence*, 552-557.

Corbett, A., & Anderson, J. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction, 4*, 253-278.

Excel. (2000). Excel (Version 2000). Redmond Washington: Microsoft.

Heiner, C., Beck, J. E., & Mostow, J. (2004, June 17-19). Improving the Help Selection Policy in a Reading Tutor that Listens. *Proceedings of the InSTIL/ICALL Symposium on NLP and Speech Technologies in Advanced Language Learning Systems*, Venice, Italy.

Major, N., Ainsworth, S., & Wood, D. J. (1997). REDEEM: Exploting Symbiosis Between Psychology and Authoring Environments. *International Journal of Artificial Intelligence in Education, 8*(3/4), 317-340.

Mostow, J., & Aist, G. (2001). Evaluating tutors that listen: An overview of Project LISTEN. In P. Feltovich (Ed.), *Smart Machines in Education* (pp. 169-234). Menlo Park, CA: MIT/AAAI Press.

Mostow, J., Beck, J., Chalasani, R., Cuneo, A., & Jia, P. (2002, October 14-16). Viewing and Analyzing Multimodal Human-computer Tutorial Dialogue: A Database Approach. *Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces (ICMI 2002)*, Pittsburgh, PA, 129-134.

Mostow, J., Beck, J. E., & Valeri, J. (2003, June 22). Can Automated Emotional Scaffolding Affect Student Persistence? A Baseline Experiment. *Proceedings of the Workshop on "Assessing and Adapting to User Attitudes and Affect: Why, When and How?" at the 9th International Conference on User Modeling (UM'03)*, Johnstown, PA, 61-64.

Murray, T., & Arroyo, I. (2002). Towards Measuring and Maintaining the Zone of Proximal Development in Adaptive Instructional Systems. *Sixth International Conference on Intelligent Tutoring Systems*, 749-758.

MySQL. (2004). *Online MySQL Documentation*, from http://dev.mysql.com/doc/mysql/en/Mathematical_functions.html

SPSS. (2000). SPSS for Windows (Version 10.1.0). Chicago, IL: SPSS Inc.

VisualSourceSafe. (1998). Visual Source Safe (Version 6.0). Redmond, Washington: Microsoft.

Witten, I. H., & Frank, E. (1999). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.

Woolf, B. P., Beck, J. E., Eliot, C., & Stern, M. K. (2001). Growth and Maturity of Intelligent Tutoring Systems: A Status Report. In P. Feltovich (Ed.), *Smart Machines in Education: The coming revolution in educational technology* (pp. 99-144): AAAI Press.