Intelligent Functional Dependency Tutoring Tool

Jessica Enright, Ben Chu and Osmar R. Zaïane
Department of Computing Science, University of Alberta, Canada
{enright, bechu, zaiane}@cs.ualberta.ca

Abstract: Computer tools for education, particularly those that allow a learner to work through many examples of a problematic type of exercise, can be helpful to that learner's understanding. Functional dependencies and their use in finding candidate keys are an area with which learners often have difficulty in undergraduate database courses. In addition to helping the students, a learning tool could help the instructor if it collected information about learner attempts of the exercises. This project makes an effort to develop such a tool, helpful to both learners and educators.

Introduction

One of the fundamental concepts to teach in a database design course is the concept of relation decomposition, which consists of dividing relations (or data tables) into smaller tables in order to reduce redundancy, eliminate wasted storage and more importantly reduce anomalies or inconsistencies due to data updates. The central tool in producing these decompositions and refinement of the database in what is called normal forms, is the theory of functional dependencies, often called normalization theory [6]. A functional dependency $X \rightarrow Y$ (read X determines Y) is a constraint on some attribute sets X and Y indicating for every pair of records r_1 and r_2 in the database, if r_1 and r_2 agree on all attributes in X, then r_1 and r_2 agree on all attributes in Y. Given some functional dependencies and using Armstrong's axioms [3] one can derive new functional dependencies. This process is important in database design as it helps identifying candidate keys and is the basis for database normalization. Armstrong's axioms are these three fundamental properties: *reflexivity* stating that if Y is included in X then $X \rightarrow Y$; *augmentation* stating that $X \rightarrow Y$ entails $XZ \rightarrow YZ$ if X, Y and Z are in the same relation; and *transitivity* stating that if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$. These axioms lead to other commonly used rules such as *union*, *decomposition* and *pseudo-transitivity* [6].

Learners often find functional dependencies among the most challenging topics presented in undergraduate database courses. As with many other logical problems, repeated practical exercises seem to help learners master the material. This motivation is the major thrust in developing FDTutor, a web-based tutoring system for functional dependencies with self-evaluation and group assessment.

Intelligent tutoring systems (ITS) can assist both learners and educators, and can fit into the overall learning experience in a number of ways. An ITS can provide both exercises to the learner, and assessment information to the educator. Merceron and Yacef [7] developed an ITS for logic exercises, with data mining functions available to the educator. Their system, Logic-ITA, allows students to practice doing formal proofs with propositional logic. It can determine when a student has made a mistake, and logs that and other information. The logged information is entered into a database. That database can be queried for information

using SQL, or can be mined by finding associations between mistakes and particular learner behaviours.

El-Khoury *et al.* [5] created the TURING system to aid learners and educators in learning and teaching mathematics skills. In this system the ITS is complementary to the efforts of human efforts. It allows the learner to practice mathematical exercises, and gives helpful messages when the learner needs extra help. It attempts to simulate collaboration between learners when providing these messages. It also assists the educator in developing problem and strategy descriptions. El-Khoury *et al.* plan to test learner response to TURING in a real-world classroom situation.

ITS can model the behaviour and knowledge of students to better provide feedback and suggestions. In [8] Mitrovic introduced the NORMIT system, a web-based ITS for teaching database normalization, including the process of finding candidate keys from functional dependencies. NORMIT uses constraint-based modeling to model learners. Using this model, NORMIT can give intelligent suggestions to learners who are having difficulty with an exercise. However, NORMIT does not include data mining utilities to assist the educator in the assessment of the learning activities.

In the remainder of this paper, we describe a design for our ITS, FDTutor, used for learning about functional dependencies that is helpful to both learners and educators. We also describe how data mining, such as association rule mining and sequential pattern mining, can provide useful patterns to help assess the learning process.

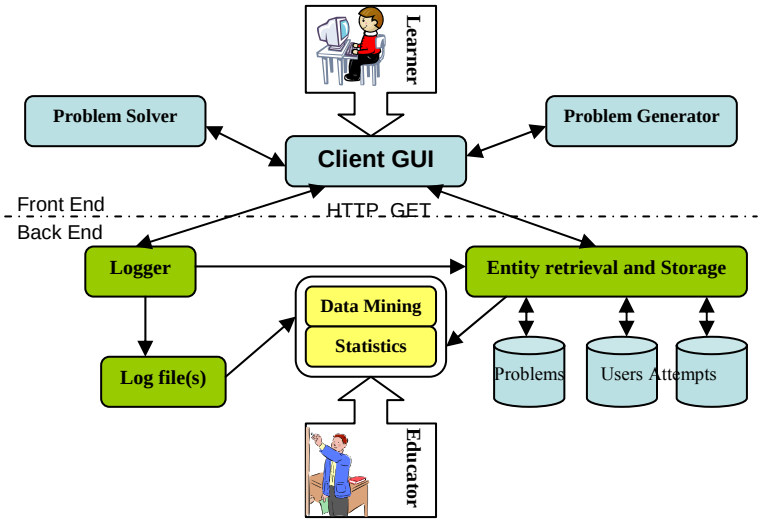


Figure 1. An overview of the prototype tutoring system architecture.

1 The FDTutor System

An ITS for teaching functional dependencies should be helpful to both learners and educators. We have designed our system with this in mind.

The learner can use our system to both practice exercises, and to evaluate his or her relative strength in the class. A simple front-end interface allows to a learner to log in, and

select either a problem from a list of previously-completed problems, or a newly generated problem. Upon selecting a problem, one can select which of Armstrong's axioms to apply, and to which functional dependencies. The result of the application is shown on the screen. If the learner is struggling with the exercise, he or she can request a hint. The system provides an automatically-generated suggestion for the next step of the derivation. When the system detects that the learner has completed the exercise, it congratulates the learner.

FDTutor also provides analytical tools to the learner. He or she might want to know any number of statistics for the purpose of self-evaluation. The system can display the number of exercises completed by the learner compared to the mean number of exercises completed across all learners. It can display the mean number of hints requested by the learner per exercise, or the mean number of steps taken to complete an exercise of some difficulty as compared to these statistics for the group of all learners. The learner can even request data on the mean number of steps taken to complete a particular problem across all learners, to compare this number to his or her own performance. Optionally, the time spent solving a problem can also be compared to the mean time across learners that successfully attempted the same exercise.

The educator can use the system's analytical tools to tailor his or her presentation of material. The system provides association pattern mining and sequential pattern mining. This gives the educator the opportunity to detect repeated patterns of actions on the part of the learners. If learners are repeatedly making the same types of mistakes, an educator can observe this and then modify classroom lesson plans accordingly. The educator can also access a particular learner's record of activity in the system, as well as various simple statistics for individual problems and sets of problems.

The idea of defining a difficulty for a given problem is an important one. It makes the analytical tools the system provides to learners and educators more potent. It allows learners and educators to intelligently assess which problems should be attempted at each stage of learning. There are two main sources of information on the difficulty of a problem in the system. The first is the system, the second the learners. The system automatically generates problems, and so must also solve them. The difficulty of a problem might be defined as the number of steps in the solution to that problem found by the system. Of course, any solving algorithm implemented is unlikely to be optimal. As the learners complete problems, the length of the solution found and the number of hints requested are logged, building a new measure of difficulty. The system might self-assess the difficulty of its problems and the efficacy of its solver based on these learner solutions.

Now that we have outlined the functionality and goals of our system design, we speak specifically to the structure of the design itself – the interactions between components of the system. These interactions are illustrated in Figure 1.

Roughly, the system can be divided into the front and back ends. The front end runs on the learner's local machine, and the back end on a server. The front end comprises the learner's graphical user interface (GUI), a problem generator, and a problem solver. The problem solver can both completely solve problems, and generate the results of a single axiom application. Note that even if the learner is temporarily offline, he or she can still solve new problems, as they are generated locally.

The back end comprises logging utilities, information storage and retrieval, and utilities for performing and viewing data mining analytics.

The information interface between the front end and back end is mediated by the HTTP protocol, and basically consists of the learner's client sending logs of the learner's activities to the logger, and getting problems from the entity retrieval and storage unit. As a means of demonstrating the value of the system we designed, we have implemented a prototype version of the system.

2 The Front End

The front end of this project is the only point of learner interaction, and so must ideally be intuitive, portable, and reasonably robust. We chose to implement the front end in Java, and used Swing for GUI building.

As mentioned in the design, a Solver runs local to the learner. The Solver works with functional dependencies represented by sets of elements in their antecedents and consequents. The Solver is invoked when a learner selects an axiom to apply. It can solve exercises itself as well - it does this by exhaustive enumeration of closures.

A learner can load problems, try to solve them by applying axioms to functional dependencies, or generate problems. The complexity of the problems generated locally is contingent upon past performance of the learner and these problems are later uploaded to the server for others to solve as well. Problems are loaded from a Problem menu. This menu is subdivided into several sub-menus - Solved Problems, Attempted Problems, and New Problems (Figure 2). The solved problems are problems the learner has at some point solved. A learner can re-solve a problem from scratch if they choose then compare to observe improvements. Attempted problems are those that were at least viewed by the learner, but have not been solved. New problems have not been viewed by the learner, though they may have been previously available.

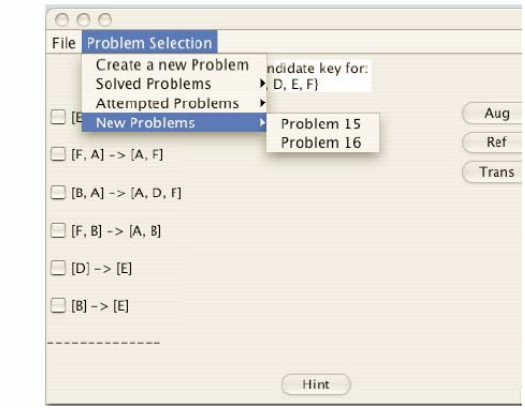
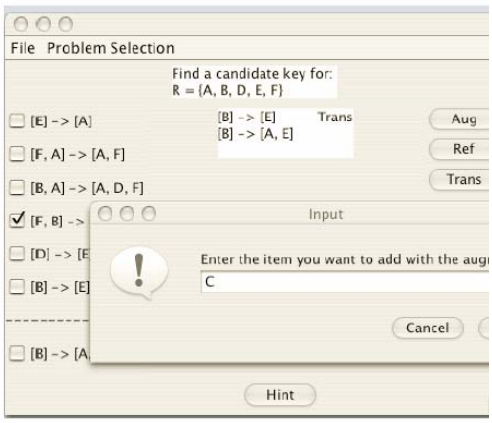


Figure 2: A screenshot of the nested problem selection menu is shown.

A learner solves a problem by applying the three Armstrong's axioms to functional dependencies that are specified with the problem, or derived by the learner (Figure 3). If a learner tries to apply an axiom with an incorrect number or type of arguments, the GUI displays an error message. A learner can ask for a hint at any time. The hint given is an

identified candidate key (Figure 4) or any intermediary statement to prove in the path to the solution, the level of which depends upon the past performance of the learner solving similar problems.

A: 

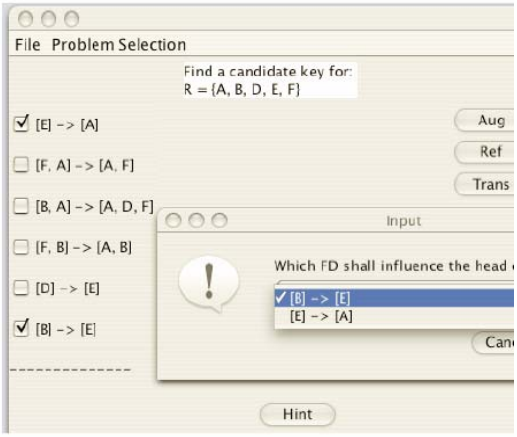
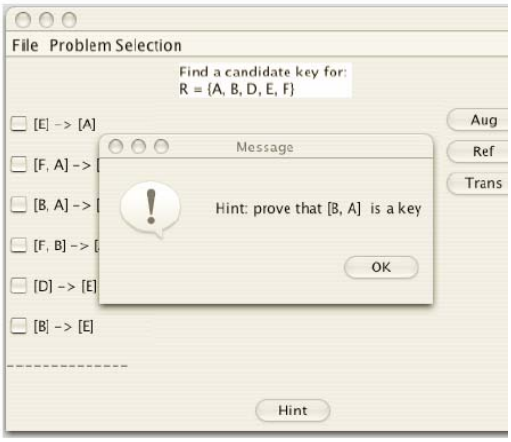
B: 

Figure 3: Screenshots of a learner applying the augmentation (A) and transitivity (B) axioms.

A: 

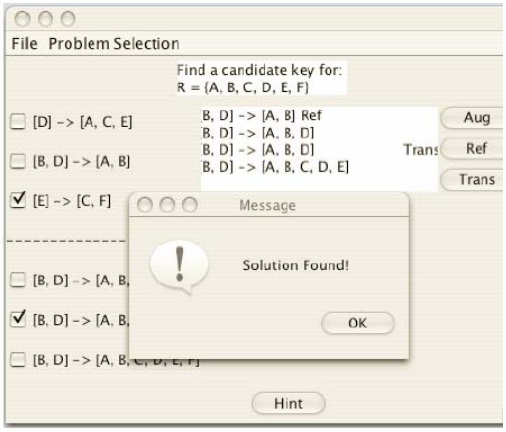
B: 

Figure 4: (A) A learner requesting a hint. (B) Detection that the learner found a solution.

Logged information is sent to the server not at each click, or each exercise, but instead when the learner exits the program. If an exception occurs when trying to send the log to the server (likely due to lack of an internet connection) then the log is not cleared, and will be sent with the next log. What is logged is very important to the success of the later data mining steps. Pertinent information is logged, including learner identification, time information, problem information, and various learner actions such as errors, requested hints, actions used, etc.

3 The Back End

The back end consists of four main components, two of which are actively involved in communication with the front end. Most of the implementation of the back end, including the

interface exposed to the front end, is written using the PHP[1] scripting language. The back end also serves an external interface to the administrator of the tutoring system, usually the educator. This interface is web-based, allowing statistical and data mining analysis to take place within a web browser environment.

The database component of the back end serves as the storage and retrieval system for three entities: learners, problems, and problem attempts.

Two different mining algorithms were incorporated into the ITS implementation: *association pattern mining* and *sequential pattern mining*. Association patterns, also known as association rules, are regularities in transactional databases hinting frequent associations between co-occurring items. In the context of FDTutor, an item could be an action by the learner, such as selection of an Armstrong axiom, or an event such as an error or the abandonment of a problem. There are well researched methods for discovering interesting relations or association rules and different measures of interestingness exist [9]. We chose the most common *support* and *confidence* which measure frequency and conditional probability respectively. But other measures of interestingness [10], such as *lift*, *cosine*, *conviction*, etc., are also possible and are available to the analysis. The most popular algorithm for discovering association rules is Apriori [1] for which many implementations exist and we chose Borgelt's[2] for its known efficiency for datasets of small sizes as we are dealing with..

Sequential patterns are simply frequent sequences of items in transactional data. In our context they are frequent sequences of events or actions. The problem was introduced in [2] but many more approaches were suggested in the literature. Sequential pattern mining for the tutoring system finds patterns in the logs that have above a specified support. We implemented this mining of frequent learner actions using Zaki's SPADE algorithm [11]. This gives the educator an idea of the most frequent learner action sequences logged as the learners attempt to solve problems. To narrow the scope of the mining, the implementation also allows mining only on learner problem attempts that lead to a solution. This can give the educator an idea of the frequent sequences that build up to a solution.

Association pattern mining also takes information from the logs and produces patterns with a specified minimum support and confidence. We implemented this mining of frequent errors, association rules between errors, and association of actions with problem solutions using Borgelt's Apriori implementation [4]. The mined frequent patterns demonstrate which errors occur most frequently and how these errors are associated with other errors. Additionally, rules mined can indicate which learner actions occurring during a problem attempt are associated with finding a problem's solution.

3.1.1 Mining Results

We ran the mining algorithms on a simulated data set where we classified particular learners beforehand to see how our mining picked up patterns. The simulated data was collected by simply asking students to test the system and being pragmatic or less pragmatic solving the problems in order to simulate good and less effective learners. We created three classes of

[1] http://www.php.net

[2] http://fuzzy.cs.uni-magdeburg.de/ borgelt/apriori.html

learners: random, average, and pragmatic. A random learner's actions make little sense, as if the learner is choosing actions and dependencies without reason. Therefore, the log entries for the learner contain many mistakes and the learner will not have very many solved problems. An average learner's actions would show a pattern of reasonable actions with the occasional mistake and a number of solved problems. A pragmatic learner will not make any mistakes and will solve every problem attempted. The majority of learners in our sample data set fell into the average category. The rest were either random or pragmatic.

By running the sequential mining on the sample learner actions with a support of 30% (see Figure 5) we observe that 4 out of 9 frequent x-sequences (where $x \geq 2$) contain a solution to the problem. Also, just over a third of the sequential transactions contain an incorrect use of the transitivity axiom. Out of all sequences, approximately half manage to find a solution to the exercise. This matches the proportional assignment of learner classes in our sample data.

Database Statistics

Number of Transactions	52
Max. item in database	23
Average sequence length	4.92308

Sequence	Support (min support=0.3)
Found a solution	28
Ask for hint	18
Created empty FD	16
Made a non-increasing FD	19
Applied Reflexivity axiom	32
Applied Transitivity axiom	44
1. Ask for hint 2. Found a solution	16
1. Ask for hint 2. Applied Transitivity axiom	18
1. Made a non-increasing FD 2. Applied Transitivity axiom	19
1. Applied Reflexivity axiom 2. Found a solution	16
1. Applied Reflexivity axiom 2. Applied Transitivity axiom	20
1. Applied Transitivity axiom 2. Found a solution	28
1. Applied Transitivity axiom 2. Applied Reflexivity axiom	20
1. Applied Transitivity axiom 2. Applied Transitivity axiom	26
1. Ask for hint 2. Applied Transitivity axiom 3. Found a solution	16

Figure 5: The results obtained from sequential mining a sample data set with min support = 30%.

Running the sequential mining against only learner actions that lead to a problem solution, we more clearly saw the steps that led to a solution. Particularly it seemed that asking hints and using the transitivity axiom most frequently led to a solution. The former property indicates that the hints given by the program tend to be helpful in guiding a learner to a solution. The latter property seemingly indicates that our problem set happens to consist of those that are solvable mainly through applying transitivity to the functional dependencies.

The running of association rule mining yields some interesting information. The most frequently committed error in our dataset was that of using transitivity incorrectly. This follows from our earlier observation that transitivity is used to solve our problems more than any other axiom. However, the frequent association of errors shows that incorrectly using

reflexivity and augmentation occur more often together than the incorrect usage of transitivity with anything else.

Using association mining, only against attempts that result in solution, yields similar results to sequential mining in this manner. The observation is that transitivity usage and the asking of hints are associated with solutions.

The statistics component of the back end allows the educator to gain an aggregate view of the current state of the tutoring system. The educator can view individual and aggregate information on the number of learners, problems, problem attempts, and number of steps taken by learners in solving problems. An example of statistics for individual learners is shown in Figure 7.

The number of attempts per problem is available, along with the success rates of each problem. The former gives an indication of which problems are tried most often, while the latter shows the success learners are having in solving each problem. This can help identify ”difficult” problems. Another indicator of problem difficulty is the average length of all learner’s actions to reach a solution per given problem. This is given in bar chart form on the statistics page. An example of problem attempt success rates is shown in Figure 6.

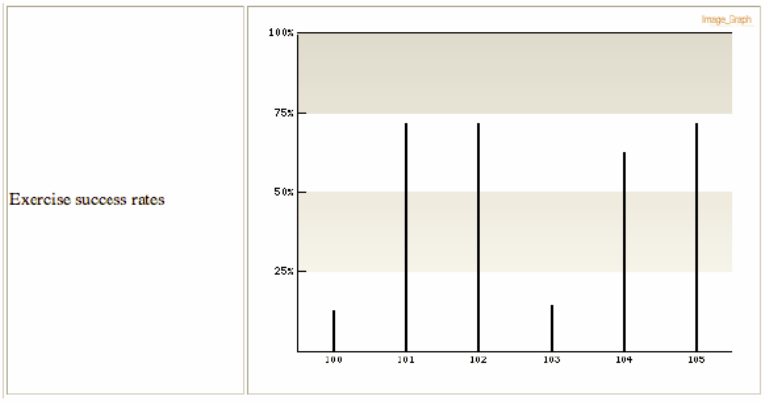


Figure 6: A sample bar chart from the statistics page with problem ids on the x-axis and success rates on the y-axis.

Success rate of student	User Id: 1 Name: Brown, Joe Number of attempted exercises: 6 Number of solved exercises: 6 Success rate: 100.00% Average attempts per exercise: 1.00	User id: 1 - Brown, Joe
Student Ranking (Top 3)	1. Brown, Joe[1] => 100.00% 2. Diller, Phyllis[2] => 66.67% 3. Drogs, Bo[3] => 66.67%	Show All Students
Student Ranking (Bottom 3)	1. Gerald, Cooper[6] => 0.00% 2. Tsunada, Hyandi[7] => 0.00% 3. Doe, John[8] => 0.00%	

Figure 7: A sample of aggregate and per-learner student statistics available in the tutoring system.

4 Discussions and Conclusions

The results presented here show that a full implementation and learner testing of an ITS for functional dependency exercises is valuable for both learners and educators. Data mining allows an educator to better understand any difficulty the learners are encountering, and enrich the learning process as required. The ability to complete an effectively limitless number of exercises, and use statistics for self-evaluation relative to the rest of the learner group would be invaluable to learners.

In further implementation, we plan to improve the Solver (and therefore our understanding of problem difficulty), increase the breadth of exercise types, and increase the data mining options available to the educator. Particularly, we are integrating clustering techniques and contrasting techniques to separate pragmatic learners from learners that deserve particular attention. We are also working on visualization tools for both the learners as well as the educators for self and group assessment.

FDTutor will be used on a cohort of undergraduate students taking a database course in the Fall of 2008. The approach would be at a first stage to provide FDTutor to a group of students while another group would be taught traditionally, then at a later stage provide the tool to both groups, and compare the performance of both groups on either stages. The preliminary tests for the tool evaluation were positive indicating approval by potential users.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases, 20th International Conference on Very Large Data Bases, VLDB, pp 487-499, 1994.

[2] R. Agrawal amd R. Srikant, Mining Sequential Patterns, 11th International Conference on Data Engineering (ICDE), 1995

[3] W. W. Armstrong. Dependency Structures of Data Base Relationships. *Information Processing* 74, North Holland, 1974

[4] C. Borgelt. Efficient Implementations of Apriori and Eclat, *Workshop of Frequent Item Set Mining Implementations (FIMI)*, Melbourne, FL, USA, 2003

[5] S. El-Khoury, E. Aïmeur, P.R. Richard and J.M. Fortuny, Development of an Intelligent Tutorial System to Enhance Students' Mathematical Competence in Problem Solving, *E-Learn World Conference on E-Learning in Corporate, Government, Healthcare, & Higher Education*, Vancouver, Octobre 2005.

[6] M. Kifer, A. Bernstein and P. Lewis, *Database Systems, An application-oriented approach*, 2nd edition, Addison Wesley, 2006

[7] A. Merceron and K. Yacef. A web-based tutoring tool with mining facilities to improve learning and teaching. In Proceedings of *11th International Conference on Artificial Intelligence in Education*, pages 201–208. IOS Press, 2003.

[8] A. Mitrovic: NORMIT: A Web-Enabled Tutor for Database Normalization. *International Conference on Computer Education*: 1276-1280, 2002

[9] G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules. Knowledge Discovery in Databases, pp 229-248, AAAI/MIT Press, 1991

[10] P-N. Tan, V. Kumar, J. Srivastava, Selecting the Right Interestingness Measure for Association Patterns, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002

[11] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.