Applications of Data Mining in Constraint-Based Intelligent Tutoring Systems

Karthik Nilakant
Supervisor: Dr A. Mitrovic

November 14, 2004

Abstract

This report describes an investigation into the use of data mining processes, with respect to student interaction with Intelligent Tutoring Systems (ITSs). In particular, a framework for the analysis of constraint-based tutors is developed. The framework, which involves three phases (collection, transformation and analysis), is implemented to extract patterns in student interaction with an ITS called SQL-Tutor. This investigation identifies alternative techniques in each of the three phases, and discusses the advantages and disadvantages of each approach. The report also highlights a number of key knowledge areas in which the mining process can be used to find rules, relationships and patterns. A range of typical findings from an existing tutoring system are used to illustrate each of these knowledge areas. It is envisaged that the knowledge that is extracted using these techniques will be used to improve the tutoring systems.

Contents

Chapter 1

Introduction

The Intelligent Tutoring System (ITS) is a technology used for computer-based education, which has been utilised across several domains with success. Prior research into ITS technology has been largely speculative: a new concept, feature or methodology would be implemented, and the results would be evaluated in a controlled trial. Such experiments are not always the most desirable way to evaluate these systems. As the field has matured, ITS software has started to become more common in educational institutions around the world. As a result, changes to these systems are becoming harder to implement, because software stability and robustness is of more importance than new features. Rather than run a trial to evaluate the effectiveness of a new feature, it may be preferable to evaluate a current system based on the way in which students interact with it.

Software engineers often use metrics to improve their understanding of software in order to manage it effectively. Such metrics may be based entirely on the design and source code of the system, or they may be related to the system's performance and productivity. The main performance criterion for ITS software is the effectiveness of the software at educating students. Unfortunately, such a metric is too difficult to obtain directly in practice. Instead, indirect measures, such as student performance on tests before and after using the ITS, are often used. While such measures can be used to gauge the effectiveness of one tutoring technique over another, and to validate the tutor's effectiveness overall, these metrics cannot be used to identify specific problems with the tutor. If students' performance increase on post-tests is found to be negligible, the ITS designers have no indication of what is wrong with the system, and are often forced to completely redesign it.

This problem is compounded in commercial ITS software, which may soon be in widespread use around the world. While these systems have normally been validated in experimental trials, it may be that some students are not experiencing the same benefit as others. A technique is required, which will identify the specific problems with the ITS that are affecting such students. Another problem with commercial systems is that changes to the system (such as changes to the problem set) need to be evaluated in some unobtrusive manner. It is common, in the software community, for "beta testing" to occur – the software will be released to a limited consumer base, which will report defects and provide comments on the software's performance. It would be beneficial if a similar field-testing technique were available for evaluating ITS performance.

One possible approach to dealing with these problems would be to engineer the system such that it reported all interaction between the ITS and the student back to the designers, and to use this information to determine the causes of problems in the system. For example, an ITS designer could read through one such report, and find that an incorrect feedback statement caused a student to fail a problem repeatedly. The designer could then change the feedback statement, or change the conditions that caused that statement to appear. However, such an approach simply transforms the problem into one of information management. Many systems, particularly commercial-grade systems, will produce far too much information to be analysed by a single person, or even a group of people.

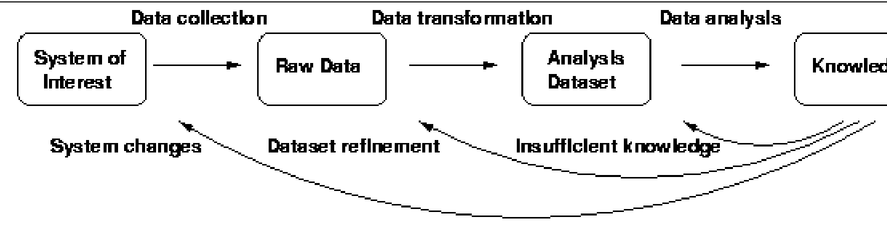


Figure 1.1: A diagram illustrating the information flows in a typical data mining process.

This report outlines research that was conducted into the use of data mining techniques to deal with the information management problem described above. In particular, the report discusses techniques that have been applied to the tutoring systems built by the Intelligent Computer Tutoring Group (ICTG) at the University of Canterbury. In this chapter, the main aims of this research are discussed. An overview of research related to this investigation is then given. Related research can be categorised into ITS research, research into the evaluation of tutoring systems, data mining research in general, and other studies into mining ITS data logs.

1.1 Aims

Data mining is a term that was coined to encompass all facets of information management. It describes a process that takes raw information and extracts "knowledge". This knowledge could take the form of assertions about the data in general, statements about constrained subsets of the data, or relationships between variables in the data.

This report investigates how data mining techniques can be used in the development and evolution of ITS software. In particular, there are three specific questions that this research attempts to resolve:

1. What are the typical processes involved in data mining when applied to intelligent tutoring systems, especially constraint-based systems?
2. What difficulties are associated with mining ITS data, and what techniques could be employed to overcome these difficulties?
3. What are some of the key knowledge areas in student-tutor interaction with these systems, and how can data mining techniques be used to extract knowledge in these areas?

To answer the first question, a framework for data mining is required. The data mining process can be classified into tasks within this framework. These tasks can then be carried out in an investigation of intelligent tutoring systems. The framework that has been adopted in this research separates the data mining process into three inter-related stages, as illustrated in Figure 1.1. The first stage, concerned with *data collection*, governs tasks that must be carried out to produce the raw data for mining. The second stage, *data transformation*, is concerned with converting the raw data into a format that can be understood by other data mining tools, and applying filtration and aggregation techniques to the raw data. The final stage, *data analysis*, is concerned with extracting interesting knowledge from the transformed data, using techniques such as statistical analysis, machine learning and information visualisation. Changes to the system, and progressive refinement of the datasets that are analysed mean that the process is iterative.

The bulk of this report deals with the various tasks associated with each of the three data mining stages outlined above. Each of these stages is expanded upon in detail in the body of this report. The data mining process itself was carried out on various systems built by the ICTG. This resulted in a set of concrete outcomes and findings for each stage, which will be used to illustrate the process.

The second question deals with issues that may arise from putting the process into practice. Once again, the data mining that was conducted on real-world ITS software is used to discover these issues, and to find a successful resolution where possible.

Finally, the third aim is concerned with using the framework to extract useful and interesting knowledge. A data mining analysis of student interaction with ITSs should produce knowledge that is helpful to the ITS designers. The investigation identified some of the core areas of ITS interaction that can be analysed using various data mining techniques.

1.2 Intelligent Tutoring Systems

Modern education has become a problem of economics. Schools, universities and governments have to make decisions about how to allocate scarce resources, in the form of teachers and support staff in order to meet the ever-increasing needs of a large student population. In primary and secondary education, approximately 30 students are allocated to each teacher. In tertiary education, a lecturer may provide instruction to 150 or more students at a time. Although this ensures that many more people receive a formal education, the individual needs of each student are not always met. This means that the standard of education could conceivably be much higher, if sufficient resources were available.

Technology has played an important role in the development of education. Early advances in technology, such as the printing press, focused on making specialised knowledge available to a wider audience. In more recent times, the focus has shifted to improving the quality of education for each individual student. ITS technology provides one approach to simulating a highly customised approach to education, previously only found in a one-on-one human tutoring environment.

The name "intelligent tutor" stems from the fact that ITS software models student behaviour, and changes the way it interacts with each student based on their individual model. Early research into intelligent tutoring focused primarily on the student model (Beck, Stern & Haugsjaa 1996), and how to use it to change the teaching strategy (known as the pedagogical module). Most of the early research into ITSs was conducted at Carnegie Mellon University, and this has resulted in a number of different systems that are known in the field as cognitive tutors (Anderson, Corbett & Koedinger 1995). These tutors are built on a theory known as ACT-R (Anderson & Lebiere 1998), which models the cognitive architecture of each student through a set of production rules (which represent the student's procedural memory as actions that are to be performed given a goal and a situation), and "chunks" of declarative data (statements and assertions about objects in the problem domain).

The tutoring systems developed by the ICTG at Canterbury use a different theoretical basis for student modelling. Known as constraint-based modelling, this approach models solution states in the problem domain as those that conform to a set of constraints (Ohlsson 1994). Each of these constraints is made up of two parts: a relevance condition and a satisfaction condition. The relevance condition specifies the circumstances that cause the constraint to become relevant. For example, a constraint may be relevant whenever the student uses the word "FROM" in their answer. The satisfaction condition specifies a further set of conditions that must be met by solutions, in order to constitute a correct solution. For example, it may be that if the student uses the word "FROM" in their solution, this should be followed by a list of names in some database.

A direct correspondence exists between every constraint and a concept to be learnt about the problem domain. This means that an analysis of constraint violations and satisfactions should produce a very accurate picture of how students learn. For example, a constraint that is often violated may be associated with a concept in the problem domain that students find difficult to understand.

Over the past nine years, a number of constraint-based tutors have been developed by the ICTG; an overview of this research is presented by Mitrovic, Mayo, Suraweera & Martin (2001). The first of these tutors is a system known as SQL-Tutor, which guides students through a set of problems that are designed to teach the Structured Query Language (SQL) used by most relational database management systems. In the years subsequent to the creation of SQL-Tutor, experimental evaluations of different versions of the system have been carried out (Mitrovic & Martin 2002, Mayo & Mitrovic 2000). Typically, these evaluations would compare two groups of students, each of which are provided with different versions of the tutor, by measuring the change in their learning through a pre-test/post-test analysis.

The ICTG has also produced other tutors related to the field of databases, including an entity-relationship modelling tutor (Suraweera & Mitrovic 2002) and a tutor for database normalisation (Mitrovic 2002). All of the tutors have been converted for use on the worldwide web. A tool to aid in the development of these constraint-based tutors, known as WETAS (Web-Enabled Tutor Authoring System) was also developed (Martin & Mitrovic 2002).

The WETAS development environment allows designers of a constraint-based tutor to focus on the creation of a domain model (in the form of constraints and an associated ontology), and on a problem set for students to work through. WETAS itself provides a "shell", which manages the components that are common to all constraint-based tutors, such as the pedagogical module, student model and a basic user interface. This report will describe how the component-based architecture of WETAS could be augmented with a data mining framework, allowing in-depth analysis of any tutor developed using WETAS.

1.3 Data Mining

The terms "data mining" and "knowledge discovery" have tended to be used synonymously is recent years. A more specific definition would be to state that the process of data mining is applied to the problem of discovering knowledge within very large sets of homogeneous data. In practice, this is the type of information stored by banking software, accountancy systems, booking systems, market research firms and so on. Data mining allows patterns and relationships to be quickly discovered from these large datasets, so that it may be used, for example, to boost sales revenue, or to manage resources better.

A common practical example of data mining is the analysis of items sold by a supermarket, or departmental store, to each customer. Known in the industry as "market-basket" analysis, the aim is to find groups of seemingly unrelated products that are commonly purchased together. Supermarkets may then choose to place these products within close proximity, or change their pricing scheme to take advantage of these relationships.

Although the majority of research into data mining is concerned with the machine learning algorithms that can be used to process the data to extract knowledge, there are a number of other tasks that must occur in the data mining process. This includes implementing mechanisms to generate, measure or retrieve data; designing systems to store raw and transformed data in a structured manner (known as *data warehousing*); pre-processing the raw data to remove abnormalities such as missing or erroneous values (noise); and conducting preliminary analyses to determine which data analysis approach would be most beneficial. A thorough review of data preparation techniques is presented by Pyle (1999).

Machine learning technology forms the core of most data mining applications. Machine learning is a general term associated with algorithms that build a model of a given data set, in order make predictions about future data, or to identify patterns in the training data. The majority of machine learning algorithms are classification algorithms – that is, they are used to predict the value of one variable (known as the "class"), given the values of a range of other variables. A commonly-used example of this involves a set of data from patients of an optometrist, which may provide information on each patient's age, eye characteristics, and the type of contact lenses that were prescribed for each patient. A classification algorithm would be able to "train" on this data, and then use the model it has built to predict the prescription for any new patient.

Research into machine learning has produced a vast array of different approaches to classification. Each algorithm has associated advantages and disadvantages; one of the tasks of any data mining process is to identify a suitable algorithm (or combination of algorithms) to use. Some classification algorithms, such as ID3 (Quinlan 1986) and the closely-related C4.5, use "decision trees" to model the dataset, which can be used to produce disjunctive rules about the training data. Another class of algorithms, known as Instance-Based Learning (IBL) algorithms, classify data by determining each example's "proximity" to areas of Euclidian space that correspond to each class. Nearest-neighbour (Aha, Kibler & Albert 1990) and NNGE (Martin 1995) are examples of IBL algorithms. Other types of classification algorithms, such as neural networks, have different characteristics and will work well in certain situations. PRISM (Cendrowska 1988) is an algorithm that generates a model as a set of rules, which can be easily interpreted by a human analyst. Mitchell (1997) provides an excellent analysis of the various machine learning algorithms that are available, and the benefits and limitations of each approach.

Algorithms for mining relationships known as "association rules" also exist. Rather than attempting to predict the value of a specific variable in the data, association rules define implications that exist between conjunctions of any group of attribute-values in the dataset, and any other attribute-value. Essentially, this means that classification is carried out using each attribute as the class attribute. For example, assume a certain dataset exists with three attributes: A, B and C. It may be that in all cases where A equals 2 and B equals 3, C is equal to 4. This could be represented by an association rule of the following form:

$$(A = 2) \wedge (B = 3)[20\%] \quad \rightarrow \quad (C = 4)[100\%]$$

The left hand side of the rule is called the rule's *premise*, while the right hand side is called the *consequence*. The percentage in brackets next to the premise represents the "support" for this association rule – it means that 20% of all the examples in the dataset obey the constraint on the left side of the rule. The percentage next to the consequence represents the "confidence" for this rule. The confidence figure specifies the proportion of examples that satisfy the premise that also satisfy the consequence (in this case, all of the relevant examples do so, giving 100% confidence).

Association rule-mining algorithms are often more beneficial than classification algorithms for data mining purposes, especially in cases where a general model of a large dataset is needed. Algorithms such as the Apriori algorithm (Agrawal & Srikant 1994), which has emerged as the primary algorithm used to mine these rules, can be used to explore multiple unknown relationships between attributes in a dataset.

Although the data mining process is well-defined, no single integrated software application exists to execute the entire process. In particular, the mechanisms used to generate data vary greatly from one data mining application to another. Data miners usually adopt a 'pipeline' architecture, where data 'flows' from one stage to the next. A variety of different tools exist to facilitate the transformations and processing associated with each stage, such as database management systems, statistical analysis packages, machine learning implementations, and custom-built tools for data filtration. In this research, a workbench called WEKA (Witten & Frank 2000) was used as part of the data transformation and analysis phases of the process. WEKA (the Waikato Environment for Knowledge Acquisition) integrates a large number of machine learning algorithms, including the classification and association algorithms mentioned previously, together with tools for analysing and experimenting with very large datasets.

1.4 Data Mining Applied to ITS Student Interaction

Data mining analysis of ITS log files has been used in similar investigations of different tutoring system architectures. The Adaptive Hypermedia Architecture, or AHA! (Romero 2003) is a system that uses student modelling to customise a hypermedia document (through the dynamic removal or addition of links and content). Data mining was used to reveal patterns in usage, similar to the analysis carried out by website statistics packages. Both the Apriori algorithm and genetic algorithms were used to mine the data. The study found that genetic algorithms were more likely to produce rules that were useful, but tended to be slower in general.

Logic-ITA (Merceron & Yacef 2003) is another system that is an existing example of research into ITS data mining. Logic-ITA is a production-rule based intelligent tutor, which trains students in propositional logic. Statistics such as user error, completed exercises and level of learning for each student were gathered by the tutor, and used as the basis for a data mining analysis. The evaluation was able to identify common errors made by the student population, in addition to identifying the kinds of students who were likely to experience difficulty with the system.

Automated log analysis is currently a highly active topic in the field of ITS research, because it plays a very important role in system evaluation. A number of tutoring systems have been developed to be “data aware” – that is, they are designed to record as much information as possible for future experimental analysis using data mining. One example is Project LISTEN, a tutor that employs speech recognition to teach reading skills. The tutor records all interactions with its students to a very fine level of granularity, so that the data may be mined at a later stage. Heiner, Beck & Mostow (2004) present a paper that describes the process of designing a data mining architecture for the LISTEN tutor. Mostow (2004) also presents a review of the LISTEN system, and highlights some important design aspects for data mining that could be used in other systems.

Strategies for making use of the knowledge gained by data mining, and for extracting useful knowledge have also been researched. Arroyo, Murray & Woolf (2004) provide an outline of how data mining can be used to identify students experiencing difficulty with a cognitive tutor, and how to alter the system to alleviate the problem. Koedinger & Mathan (2004) describe how data mining can be used to infer different patterns in student learning by using learning curves.

Finally, it is important to note that although a significant amount of research is currently being carried out in this field, few of these techniques have previously been applied to constraint-based tutors. The research presented in this report identifies unique issues associated with mining the logs of ITS software built using constraint-based modelling.

Chapter 2

Data Collection

Data mining always begins with collecting data; one cannot mine something that one does not have. The data collection phase is a crucial part of the data mining process. Without a rich raw dataset to analyse, very little knowledge can be extracted. Furthermore, the collection phase is not well-supported by data mining tools such as WEKA – it is up to the individual to plan and implement an appropriate data collection strategy.

In a standard experimental evaluation, a hypothesis is normally formed before conducting the trial. The trial itself validates (or invalidates) the hypothesis. In a data mining analysis, the process is inverted – data is collected from relevant sources, and this data is then mined for hypotheses (which take the form of rules and models) about the dataset in general. The type of knowledge that can be extracted depends entirely on the attributes of the dataset. It is therefore beneficial to ensure as many variables as possible are accounted for, when collecting data.

This chapter will describe a model of student interaction, and show that data mining analysis is essentially the same as traditional evaluation, except that it is performed at a finer level of granularity. The model will then be used to describe how interaction data can be extracted and stored for a standard constraint-based tutoring system.

2.1 Modelling Student Interaction

For this investigation, the type of knowledge that is required consists of rules, relationships and patterns in student interaction with constraint-based tutors. To identify the different variables that can be recorded, a model of student interaction was developed. Figure 2.1 shows that interaction can be analysed at a number of different layers. A typical evaluation of a tutoring system will focus only on the outermost layer; a pre-test is conducted before the beginning of a course of instruction, and a post-test is conducted after the course. This gives an indication of the change in performance experienced by each student, albeit with a coarse level of granularity. A more detailed analysis would explore learning outcomes at the inner layers of the model.

For example, during a course of instruction, each student is likely to use the tutoring system in multiple sessions. Each session could begin and end with tests to evaluate changes in performance on a per-session basis. During each session, the student attempts a number of problems. Information about whether the student was able to solve each problem could be recorded. The student may attempt each problem multiple times before arriving at a solution. Data related to individual attempts on each problem could be collected.

At the finest level of granularity, each attempt violates and satisfies a number of constraints. Information about every constraint violation and satisfaction could be collected after each attempt, resulting in a representation of the cognitive state of the student, otherwise known as the student model. The difference between the standard student model (used internally by the tutoring system) and this model is that, for the purposes of data mining, this information is augmented with contextual data. That is, information about the outer layers of the model is recorded

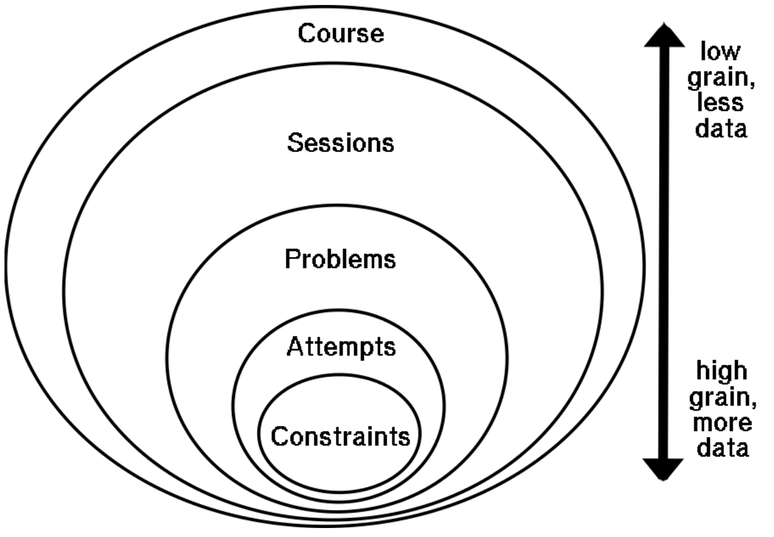


Figure 2.1: A model of student-tutor interaction showing the various layers of granularity.

in conjunction with each constraint violation or satisfaction. For each constraint violation that is recorded, information about the student that caused the violation, the course that the student participated in, the problem that was being attempted, and information about the attempt that caused the violation must also be stored, so that the analysis phase can highlight relationships between data from similar contexts.

One of the first tasks in the data mining process is to choose an appropriate level of granularity, and to identify all of the variables that can be recorded at that level of granularity. In this investigation, the two innermost layers (attempts and constraints) of the model were chosen; it was found that per-attempt granularity was sufficient for most purposes, while per-constraint granularity resulted in a very large dataset that was only used for one specialised study.

If each attempt is regarded as the basic unit of interaction with the tutor, the primitive features of interaction can be found by examining the tasks that students go through in each attempt. A student will initially be presented with a problem statement, and possibly some additional information (known as scaffolding) to help them solve the problem. The student then constructs their solution to the problem, typically by reading the problem and thinking about it for some time. The student then must input their solution into the tutoring system, usually by typing it in directly, or by using some graphical user interface to assemble a solution. When the student is happy with their solution, or if they feel that they need further help, they can submit their solution to the tutor. The tutor analyses the student's solution, and determines which constraints have been violated. If no constraint violations have occurred, then the solution is correct, and the student may move on to a new problem. If the solution is incorrect, the tutor generates one or more feedback statements to help the student to improve their solution. The level of feedback that the tutor produces can be controlled by the student: the tutor can produce short statements about the number of "mistakes" (constraint violations) that the student made, or more detailed feedback relating to one or more of the constraint violations.[1] This feedback is then used by the student on subsequent attempts on the same problem. Table 2.1 summarises the data that could be recorded for each attempt made by a student.

[1]A default feedback path is also defined for the tutoring systems in this investigation. If the student does not ever set the feedback level directly, then the default feedback on the first attempt tells the student whether their solution is correct or incorrect. After the second attempt, the student is told how many errors exist. After the third and subsequent attempts, the tutor gives the student advice related to one of the constraints that was violated. The student must explicitly ask for a partial or full solution to the problem, if they want it.

pre-attempt Information used by the student prior to making an attempt:

- available feedback
- problem number
- attempt number
- problem complexity
- problem context

attempt Information about the attempt itself:

- time spent on attempt

post-attempt Information related to the outcome of an attempt:

- constraints
  - relevant
  - satisfied
  - violated
- requested feedback level
- whether the solution is requested

Table 2.1: A taxonomy of variables related to a student's attempt at a problem.

Choosing the attempt as the primitive episode of interaction with the tutor allows the categorisation of data into attributes associated with student actions before, during and after making an attempt. Mostow (2004) defines these three stages, respectively, as the decision stage, context stage and outcome stage of interaction with the tutor. An interesting feature of Table 2.1 is the lack of information associated with the context stage – if it were possible to record exactly what the student did (or thought) while making an attempt, an analysis of the data may yield more specific knowledge. However, the constraint-based tutors under study in this investigation employ an "on-demand" feedback approach to tutoring. Students are free to formulate a solution to a problem in any manner, and are only guided when they choose to submit their solution. This contrasts with most cognitive tutors (such as the Carnegie Learning Algebra Tutor), which will stop a student whenever a deviation from the correct solution path is detected.

For constraint-based tutors, the bulk of useful data that can be collected is associated with the outcome, or post-attempt stage of interaction. A CBM tutor assesses the student's solution and produces a list of relevant, satisfied and violated constraints for that attempt. Constraint violations that occur in the post-attempt stage are essentially equivalent to the triggering of "buggy rules" in a cognitive tutor, which would be recorded at the context stage. This means that although student modelling information is recorded at different stages of the interaction process in cognitive tutors and constraint-based tutors, equivalent data can be gathered from both kinds of systems.

To explore the data related to the attempt variables, which will be used to augment and enrich the dataset, an analysis of the various data stores available within an ITS is required. Figure 2.2 shows a typical architecture adopted by constraint-based tutoring systems such as SQL-Tutor. The diagram shows the main processing units and the main information stores in a typical constraint-based ITS. The domain model contains a set of constraints relevant to the tutor's domain, which may be structured into an ontology (a hierarchical representation that groups together related constraints into higher-level knowledge units). In some tutors, each constraint specifically contains information based on some domain-level concept. For example, in SQL-

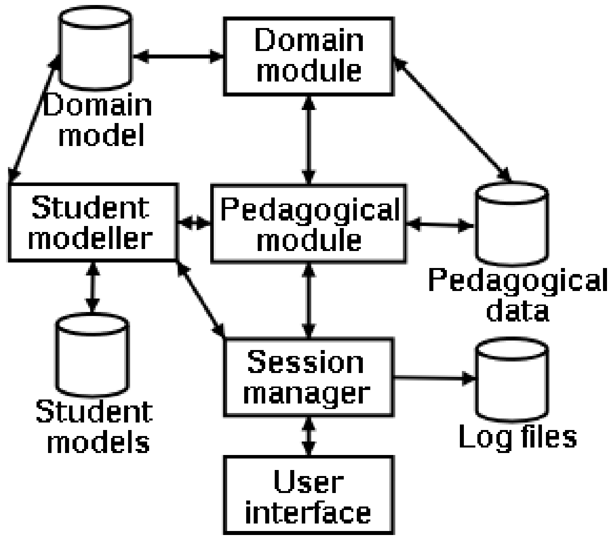


Figure 2.2: A diagram of the standard architecture of a constraint-based tutor.

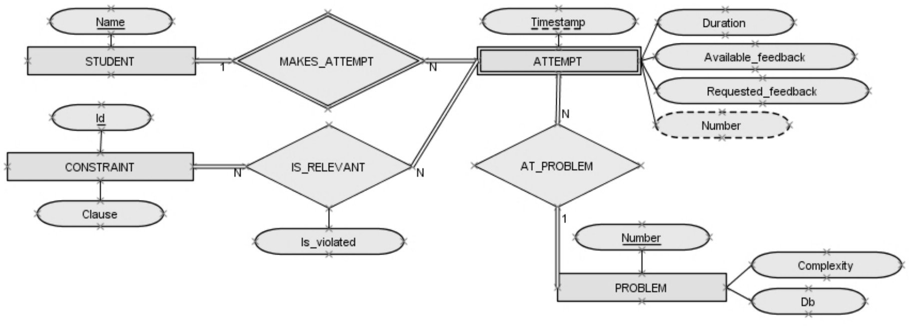


Figure 2.3: An entity-relationship schema of the interaction data that can be extracted from SQL-Tutor.

Tutor, each constraint is associated with a clause of the SQL SELECT statement. This allows the tutor to give the student a rough indication of their current learning status. The constraint base is used by the domain and student modelling modules.

Another information store is the pedagogical dataset, which contains a set of problems and their answers. Some tutors also store complexity information for each problem, which the pedagogical module could use to select appropriate problems based on data from the student model. Ideal solutions to each problem take the form of plain-text strings (which may be be encoded in XML), so that they may be acted upon by the domain module's pattern matching routines.

Finally, the student model and log files store interaction information for each student. The student model lists all of the constraints that have been relevant at some time for a particular student, and records satisfactions and violations of each of these. The log file for each student also contains this information, in a more verbose format, designed for debugging purposes. The log files also contain data related to other elements of student interaction, such as feedback, timing, attempted problems and so on. The log files were the main source of data in this investigation – the next section will outline the various parts of a typical log file that were used for data extraction.

After the variables to be recorded have been established, a structured model of the data is needed, so that the data can be extracted and stored in an organised manner. Figure 2.3 is an entity-relationship (ER) diagram of the data that can be collected about student interaction with a single tutor. The diagram shows the main interaction entities (in rectangles), as well as the attributes of these entities (in ovals) and the relationships between them (in diamonds).

The main entity in this diagram, as discussed previously, is the *Attempt* entity. Each attempt at a problem is modelled as a weak entity; this is because a single attempt can only be uniquely identified with respect to a particular student. The *Attempt* entity has some of the attributes that were listed in Table 2.1, and the other attributes are available through relationships with other entities, such as constraints and problems.

This particular schema refers to SQL-Tutor, which groups together problems associated with the same database description. The "Db" attribute of the "PROBLEM" entity holds this information. Each constraint in the domain model also has an associated clause from the SQL SELECT statement that it attempts to teach.

2.2 Extracting Interaction Data

This section will describe how data was recorded and extracted from SQL-Tutor. The framework described in the previous section is used to show how data related to each interaction entity is gathered.

2.2.1 Attempt Information

When SQL-Tutor was first developed, one of the system's primary functions was to record student interaction in human-readable log files, for manual analysis. The interaction logs also assisted the designers in performing debugging tasks. For this investigation, the information in the log files needed to be "distilled" into a machine-readable format, in order to allow data mining tools to operate on this data at a later stage.

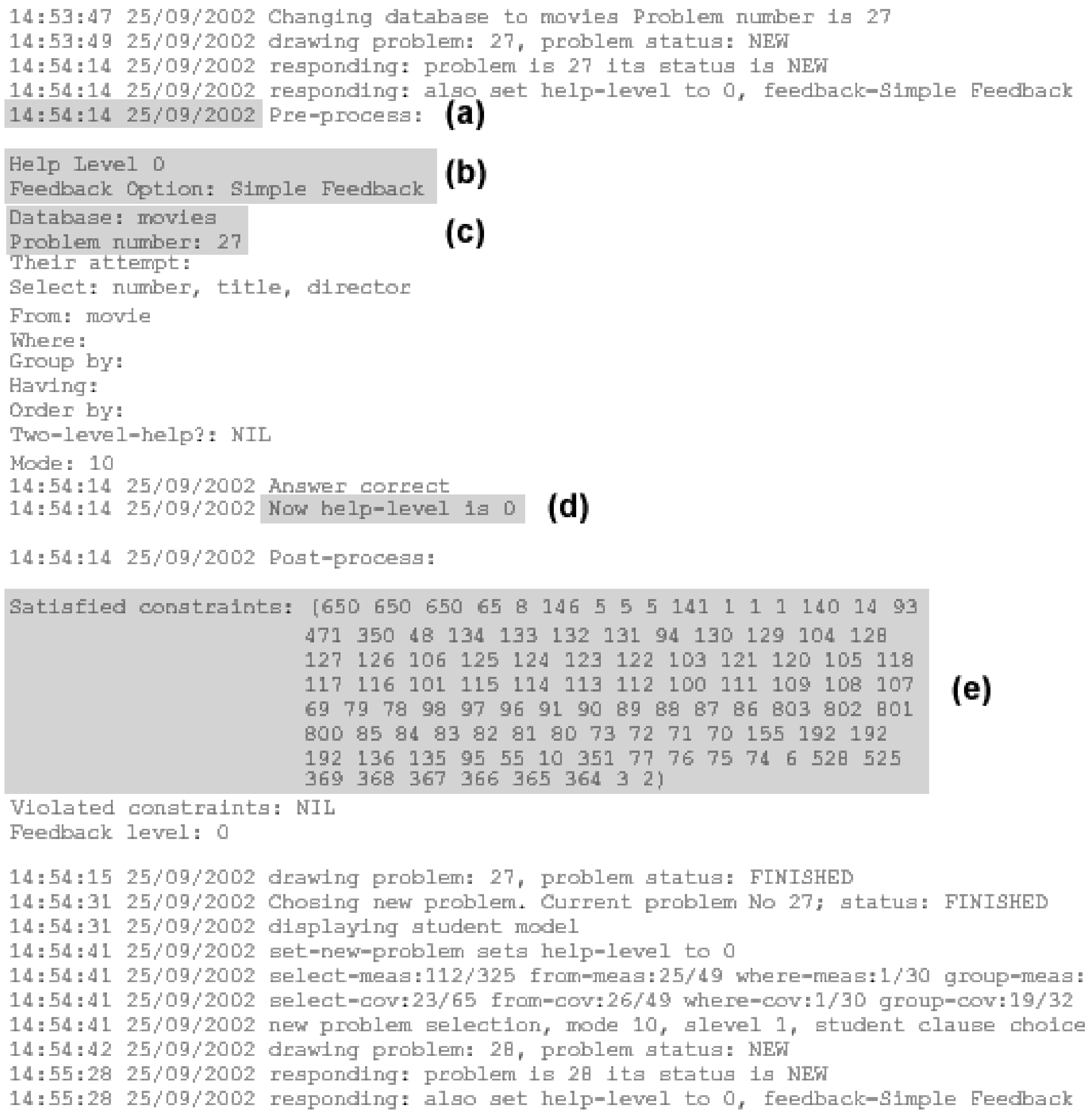
Figure 2.4 shows an annotated extract from the log file of a randomly selected student that used SQL-Tutor. Parts (a) through (e) of Figure 2.4 highlight some key areas of the extract, which correspond to some of the variables that were discussed in the previous section. Part (a) denotes the time stamp at the start of an attempt that was made by the student. In part (b), the available feedback for that attempt is recorded. Part (c) of the extract contains information related to the problem being attempted. Part (d) contains the requested feedback level (which has not changed in this case, because the student's attempt was correct). Part (e) is the most important part of the extract, as it contains all of the information about the constraints that were relevant, violated or satisfied in this particular attempt.

A tool was developed to locate the key elements within the log files, process some of this data in order to derive other attributes, and to prepare the data for storage. The implementation of this tool is discussed later in this chapter. The main function of the tool was to strip off the extraneous matter in the log files, and to retain only the important data. Table 2.2 outlines the interaction attributes that were extracted or derived by this tool, which make up the core of the dataset that was analysed.

2.2.2 Collection of Related Information

Additional data about problems, constraints and students were used to enrich the core dataset. In SQL-TUTOR, problems have a complexity rating ranging from 1 to 9 (least to most difficult). Each problem is also grouped according to the database description to which it refers. This data was extracted directly from SQL-TUTOR's problem-set source file. Similarly, each constraint's associated SQL clause was recorded by analysing the constraint source files.

Each student's ID was recorded, in addition to a "mode" value, which indicates the type of experiment each student was involved in. SQL-TUTOR organises the interaction logs into



```
14:53:47 25/09/2002 Changing database to movies Problem number is 27
14:53:49 25/09/2002 drawing problem: 27, problem status: NEW
14:54:14 25/09/2002 responding: problem is 27 its status is NEW
14:54:14 25/09/2002 responding: also set help-level to 0, feedback-Simple Feedback
14:54:14 25/09/2002 Pre-process:  (a)

Help Level 0                        (b)
Feedback Option: Simple Feedback
Database: movies                    (c)
Problem number: 27
Their attempt:
Select: number, title, director
From: movie
Where:
Group by:
Having:
Order by:
Two-level-help?: NIL
Mode: 10
14:54:14 25/09/2002 Answer correct
14:54:14 25/09/2002 Now help-level is 0   (d)

14:54:14 25/09/2002 Post-process:

Satisfied constraints: (650 650 650 65 8 146 5 5 5 141 1 1 1 140 14 93
                       471 350 48 134 133 132 131 94 130 129 104 128
                       127 126 106 125 124 123 122 103 121 120 105 118
                       117 116 101 115 114 113 112 100 111 109 108 107   (e)
                       69 79 78 98 97 96 91 90 89 88 87 86 803 802 801
                       800 85 84 83 82 81 80 73 72 71 70 155 192 192
                       192 136 135 95 55 10 351 77 76 75 74 6 528 525
                       369 368 367 366 365 364 3 2)
Violated constraints: NIL
Feedback level: 0

14:54:15 25/09/2002 drawing problem: 27, problem status: FINISHED
14:54:31 25/09/2002 Chosing new problem. Current problem No 27; status: FINISHED
14:54:31 25/09/2002 displaying student model
14:54:41 25/09/2002 set-new-problem sets help-level to 0
14:54:41 25/09/2002 select-meas:112/325 from-meas:25/49 where-meas:1/30 group-meas:
14:54:41 25/09/2002 select-cov:23/65 from-cov:26/49 where-cov:1/30 group-cov:19/32
14:54:41 25/09/2002 new problem selection, mode 10, slevel 1, student clause choice
14:54:42 25/09/2002 drawing problem: 28, problem status: NEW
14:55:28 25/09/2002 responding: problem is 28 its status is NEW
14:55:28 25/09/2002 responding: also set help-level to 0, feedback-Simple Feedback
```

Figure 2.4: An extract from a log file for a student that used SQL-Tutor.

Attribute	*Method of extraction*
Problem number	Parsed directly from log.
Attempt number	Derived from problem number and attempt history.
Available feedback	Parsed directly from log.
Satisfied constraint IDs	Parsed directly from log.
Violated constraint IDs	Parsed directly from log.
Requested feedback	Parsed directly from log.
Attempt duration	Derived from attempt time stamps.

Table 2.2: Variables related to attempts, and the methods of their extraction.

separate files for each student. The session manager produces two files for each student, which follow the naming convention *[student ID]* and *[student ID].LOG*. The first file stores the student's constraint-based model, while the .LOG file contains the student's individual interaction log. The data extraction tool traverses these files to collate data for an entire group of students. The file names are used to identify the current user.

2.3 Storing Interaction Data

Once the relevant data have been extracted from the log files (and other related files), some structured way of storing the data is needed. Since the extracted data conforms to an entity-relationship model (see Figure 2.3), a relational database would be a good candidate for storing the collected data. Relational databases have several advantages over "flat file" storage, including the following:

- Data can be separated into groups of related attributes (tables). This allows data to be extracted separately, and also allows the schema to be changed at a later stage without affecting the data that already exists (for example, extra tables could be added). This is particularly important in data mining applications, since more information about any entity may become available in future – if a flat file were used for storage, it is likely that the entire file would need to be regenerated.

- Customised datasets from across the entire database can be retrieved with ease using SQL statements. SQL also allows data to be filtered, grouped, sorted and/or aggregated, providing data analysts with a variety of different "views" of the raw data.

- Relational database management systems provide superior performance, security and data integrity features over flat file storage.

Table 2.3 shows the relational schema that was used. Tables are related to each other using "foreign key" attributes. For example, the *Attempt* table is made up of a number of rows (or tuples) that each correspond to a single attempt made by a student on some problem. The student ID and problem number are used to identify the student and problem associated with that attempt. A separate table is used to store the mappings between attempts and the relevant constraints for that attempt, since multiple constraints can be relevant for multiple attempts (a many-to-many relationship exists).

Some deviations from the schema illustrated in Figure 2.3 are present in the relational schema of Table 2.3. The main difference is that rather than using a combination of the attempt timestamp and student ID for the primary key of the **attempt** table, an automatically incremented ID attribute was used. A similar attribute was created for the **has_relevant** table. This allows for easier formulation of queries that relate the *Attempt* and *Constraint* tables (the join condition is simplified).

Attempt: attempt_id, attempt_num, avail_feedback, select_feedback, time_diff, problem_num (FK), student_id (FK)

Student: student_id, mode

Problem: problem_num, database, complexity

Has_relevant: relevant_id, attempt_id (FK), constraint_id (FK), is_violated

Constraint: constraint_id, clause

Table 2.3: Relational database schema for the collected interaction data.

2.4 Implementation Details

The data collection tool was written as a standard Java application. This meant that the tool could take advantage of J2SE's stream manipulation features, its rich data structure library, and its DBMS connectivity capabilities. The utility itself was lightweight, at approximately 300 lines of code. A partial source code listing for the tool has been included in the appendix of this report, which shows how information was extracted from various parts of each log file.

MySQL server version 4.1 was used as the relational database management system. The relational schema was created using the server's administration tools, and the data collection utility was used to load the interaction data into the various tables. Some tables, such as the problem and constraint tables, were populated directly from flat files containing the relevant data.

The tool was used to create a minable dataset from the 2002 SQL-Tutor evaluation, which involved approximately 70 students. In plain-text format, the combined log files were originally 9.3 megabytes in length; the tool produced a raw dataset of approximately 150 kilobytes, containing over 6000 logged attempts.

The SQL-Tutor student logs that were processed using this tool were similar in nature to the logs that are produced by other tutoring systems such as ER-Tutor and NORMIT. Furthermore, the WETAS authoring tool produces log data in a similar format. This means that the same data extraction tool could be used to collect information from a variety of different systems.

Chapter 3

Data Transformation

In the previous chapter, a process was put in place to collect a large amount of data, in terms of both attributes (such as problem complexity and attempt number) and instances (such as each attempt and each problem), to maximise the potential for extracting interesting and meaningful knowledge from the system. At this stage, the dataset is in a state that can be analysed by the tools that will be described in the next chapter (assuming the relevant format conversions are carried out). However, with such a large amount of information, it is unlikely that an analysis of the entire raw dataset will produce any useful relationships or patterns.

Performing the analysis described above is akin to asking the question "How do students interact with this tutoring system?" If a general question such as this were put to an ITS expert, it is likely that they would only be able to give a similarly vague (or obvious) answer. To garner specific knowledge about the system, a more specific question is needed. Take, for example, the following conversation between a fictitious analyst and an ITS expert:

> **Q:** How do students interact with the tutor?
>
> **A:** They attempt problems, and sometimes they violate constraints.
>
> **Q:** When a student violates a constraint, what happens?
>
> **A:** They might try to change their solution, or they might give up.
>
> **Q:** If a student re-attempts a problem, but does not violate fewer constraints, what happens then?
>
> . . .

In a similar manner, a data mining analysis will normally follow a refinement approach that incrementally constrains the analysis dataset until specific, interesting, reliable knowledge is extracted.

At this stage, the meaning of the term "analysis dataset" should be made clear. At the completion of the collection phase, the raw data is structured into a number of relations. Most data analysis tools operate on a single table of values, which can be constructed by performing a "join" between each of the relations in the database. This results in a structure known as an analysis dataset. Table 3.1 shows how the analysis dataset can be categorised vertically and horizontally into *attributes* and *instances*. Each of the attribute-values $x_{i,j}$ can vary depending on its attribute *type*. Each attribute has a data type, which may be numeric, textual or draw from some other domain. If the domain of values that an attribute could take is finite, it is said to be a *nominal* attribute.

In this chapter, a number of issues related to transforming data to constrain and refine the analysis dataset are discussed. An analysis dataset can be transformed in a number of ways, to produce new, constrained analysis datasets. Such transformations facilitate the progressive refinement technique outlined above. Transformations may take place on the instances of the dataset, which normally results in the removal of one or more instances through filtration. The

	$attribute1$	$attribute2$	$\ldots$	$attribute_m$
$instance_1$	$x_{1,1}$	$x_{1,2}$	$\ldots$	$x_{1,m}$
$instance_2$	$x_{2,1}$	$x_{2,2}$	$\ldots$	$x_{2,m}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$instance_n$	$x_{n,1}$	$x_{n,2}$	$\ldots$	$x_{n,m}$

Table 3.1: The format of the analysis dataset.

attributes of a dataset may also be transformed, possibly also through filtration, or through conversion of the attribute type.

3.1 Dataset Filtration

Refinement of the dataset typically involves reducing the dataset by filtering out any information that is of no interest. Normally, in the analysis stage, some knowledge will be gathered that necessitates some data filtration, in order to allow a certain aspect of the data to be analysed in greater detail. For example, an analysis of the raw data may find that problems from a certain database (in SQL-Tutor) tend to violate a certain constraint more often. To explore the reasons for this, the analysis could be restricted to only those problems. Filtering out needless information allows analysis tools to produce more reliable results.

Data is filtered by defining some condition on one or more attributes (for example, "attempt_num = 1 AND db = movies"), and checking whether each instance satisfies that condition. The instances that violate the condition are removed from the resultant dataset.

It is important to note that filtration should be a distinct stage of the data mining process, which occurs strictly after data collection. The data collection phase should gather as much information as possible about the system of interest, so that all of this information is usable at a later stage. Filters can be used to act on the data to produce new analysis datasets, but the raw data store should remain unchanged.

Two different tools for data filtration will be discussed: filtering through database queries, and using the WEKA workbench for filtration.

3.1.1 Using SQL for Filtration

Having the raw data archived within a relational database system allows analysis datasets to be extracted using the Structured Query Language (SQL). SQL is sufficiently powerful to allow filters to be placed on the extracted datasets. An SQL SELECT statement can be used to extract only those attributes of the raw dataset that are of interest, to join relations together to form a single dataset, and to place conditions on the instances that may be returned. For example, the following statement could be used to find all of the information about each attempt that was collected, together with the associated problem information.

```
SELECT * FROM attempt, problem WHERE attempt.problem_id =
problem.problem_id;
```

An analysis of this dataset may warrant a filter that restricts the data to only first attempts at a problem. The following statement could be used to achieve this.

```
SELECT * FROM attempt, problem WHERE attempt.problem_id =
problem.problem_id AND attempt.attempt_num = 1;
```

Another advantage of SQL is that the filters for commonly used analysis data can be stored in the database as "views". For example:

```
CREATE VIEW attempt_problem AS SELECT * FROM attempt, problem WHERE
attempt.problem_id = problem.problem_id;

SELECT * FROM attempt_problem WHERE attempt.attempt_num = 1;
```

It is also possible to form slightly more complicated conditions on these filters. For example, it is possible to count the number of constraint violations and only including those attempts below a certain threshold. However, for more complicated filters that correspond more closely to analysis processes, WEKA could be used.

3.1.2 Using WEKA for Filtration

The WEKA machine learning suite provides also some filtration capabilities. Like SQL, WEKA can also be used to filter out instances containing certain attribute-values. The WEKA instance filters also provide the following functionality:

- The "RemovePercentage" and "Resample" filters may be used to randomly remove instances from the analysis dataset. This is useful in situations where the dataset is so large that an analysis tool (for example, a neural network) would spend an excessive amount of time to process the entire dataset.
- The "RemoveFolds" and "RemoveMisclassified" filters are special filters that run a classification algorithm over the data, and use the results of the analysis to filter the dataset directly.

One of the drawbacks with using the filters provided by WEKA is that there is no standard way to store and restore commonly-used filters – only the resulting datasets can be saved.

3.2 Attribute Conversion

The second class of transformations operates on the values of attributes in every instance of an analysis dataset. The main issue in attribute conversion is deciding how to convert continuous attributes into nominal attributes. Attribute transformations can also be used to perform other operations on the domains of attributes.

3.2.1 Discretisation

Many of the machine learning data analysis tools (which will be discussed in the next chapter) cannot operate on attributes that have continuous domains. Continuous attributes in the analysis dataset must be converted to nominal attributes before the machine learning algorithms can operate on the data.

There are two main techniques for discretising continuous attributes. The first technique involves scanning the entire dataset for all of the possible attribute-values that a continuous attribute takes, and using these values as the domain of a nominal attribute. This is essentially constraining the domain of an attribute to only those attribute-values that exist in the dataset. If this technique is used, none of the attribute-values are actually changed. Instead, the domain of that attribute becomes a closed set of possible values. However, this technique often leads to the creation of very sparse datasets, from which knowledge can be hard to extract.

The other technique, often referred to as 'binning', defines a set of nominal 'classes' for each attribute, and assigns each attribute-value to one of these classes. For example, if a numeric attribute has values that range continuously from 0 to 100, the domain could be categorised into four bins (0 to 25, 25 to 50, 50 to 75 and 75 to 100). Each attribute-value is converted into a nominal attribute that corresponds to one of these bins.

At least three different strategies exist for defining the intervals, each of which depend on the number of bins (n) that are desired:

Equal-width: Divides the total range of the attribute into n equal-width intervals.

Equal-frequency: Calculates the interval of each class so that each class is allocated an approximately equal amount of instances from the dataset.

Customised: The intervals for each of n classes are user-defined.

Binning does affect the attribute-values in a dataset, leading to some information loss (also known as quantisation loss). However, the use of binning allows patterns that occur across instances with attributes within the same range to be found. The WEKA attribute filters provide support for both equal-width and equal-frequency binning. Customised binning is not directly supported by WEKA – the new attribute must be derived using the techniques outlined below.

3.2.2 Attribute Derivation

The final kind of transformation is closely related to the data analysis process discussed in the next chapter. In many analyses, a new attribute needs to be derived from one (or more) of the attributes in a dataset. The new attribute may result from a mathematical transformation of another attribute. For example, the time difference attribute could be converted to minutes instead of seconds. The most common kind of derivation performs some kind of aggregation on another attribute (or related dataset). For example, the customised binning technique described above is a method of grouping together the values of an attribute. Another kind of aggregation occurs when the constraints related to each attempt are grouped into an attribute of the attempt dataset. In many analyses, a "violated count" attribute was included as an attribute of the analysis dataset. This attribute identified the total number of violated constraints in each attempt.

Attribute derivation is usually performed using the tools described in the next chapter. The main difference is that in this case, those tools are not being used to extract knowledge directly, but to augment the analysis dataset in some way (to allow useful knowledge to be extracted).

3.3 Implementation Details

All of the collected data from the SQL-Tutor logs, including numeric information, was stored in the database in either text (VARCHAR) or numeric (INTEGER) format. A "StringToNominal" filter was normally used to convert dense text attributes (such as problem complexity) to a nominal data type. Some attributes, such as attempt duration, were stored as numeric data, which meant binning was required.

The attempt duration initially could have been pre-filtered to remove all attributes over a threshold (for filtering out attempts that were uncharacteristically long). However, it was decided to define an initial custom binning scheme, which grouped all attempt duration values over a certain threshold (30 minutes) into one class, and use equal-frequency binning for the remaining instances.

In addition to using the analysis tools that will be described in the next chapter, attributes may also be derived using procedural code. For example, when producing an analysis dataset that contains lists of violated constraints, it may be necessary to keep a "history" of constraint violations. Procedural code was written to maintain a record of previous constraint violations for each student. The exploratory analysis of Section 4.4.5 uses this data for an association-rule analysis of constraint violations.

Chapter 4

Data Analysis

The final phase of the data mining process involves processing an analysis dataset for relationships and patterns that give the analyst some knowledge about the system of interest. This is the area of data mining that is most widely supported by tools, and the range of tools that are available allow knowledge of varying levels of obscurity to be extracted.

Three main techniques for analysing data will be discussed in this chapter. The first technique, straightforward statistical analysis, is suitable for finding specific answers to analysis questions, such as ″How many attempts violated constraints?″ and ″What was the average attempt duration?″. The second technique is machine learning, which is used to extract unknown information about a specific area of interest amongst the data. Machine learning could be used to find unknown relationships between the attempt duration and number of violated constraints, for example. Finally, information visualisation tools could be used to change the representation of multivariate data, so that it is easier for a human analyst to observe directly. Graphs and 3D representations of data are examples of information visualisations, where the viewer may be able to see patterns in the visual layout.

In a typical analysis, each of the various techniques is used in conjunction with the progressive refinement methodology described in the previous chapter. For example, information visualisation could be used to highlight an important subset of an analysis dataset. Data transformations could be applied, and the refined dataset could be analysed using machine learning tools to extract specific relationships.

This chapter will detail the various approaches to data analysis that are available, and the tools and techniques that are applicable to each approach. These techniques are then applied to a number of different analyses on the SQL-Tutor interaction data that has been used as a running example in the previous chapters, and the results of these investigations are reported.

4.1 Rudimentary Statistical Analysis

Many studies overlook the importance of basic measurement operations, such as counting instances, calculating averages and calculating variance, in analysing large datasets. These basic operations can provide an analyst with specific, reliable knowledge about a dataset. The statistical analysis tools that will be discussed in this section are suited to investigative procedures where specific knowledge is being searched for. Typical questions that can be answered by simple statistical analysis include ″What is the average number of constraint violations?″, ″What is the average problem complexity?″ and ″What is the total time spent in attempts?″. Questions such as these can be applied to multiple datasets to highlight their differences and similarities.

4.1.1 Analysis Using SQL

SQL provides functionality for a number of statistical operations, such as those outlined above. This allows some statistical analysis to be performed directly on the relational database, provided that SQL can be used to extract an appropriate analysis dataset (an SQL query could not be used to analyse data that needs to be filtered by WEKA, for instance).

The following example shows how the average constraint violation count could be extracted from the entire raw dataset:

```
SELECT AVG(cnt) FROM (SELECT COUNT(*) AS cnt FROM attempt,
has_relevant WHERE has_relevant.attempt_id = attempt.attempt_id AND
has_relevant.is_violated = TRUE GROUP BY has_relevant.relevant_id)
AS t1
```

The query above uses a nested subquery to count all of the constraint violations for each attempt, and then averages the resulting table in the outer query.

4.1.2 Analysis Using Spreadsheet Software

Spreadsheet software such as Microsoft Excel and OpenOffice Calc can also be used for performing simple analyses of a dataset. A format translation is necessary to export an appropriately filtered analysis dataset into a spreadsheet tool – most spreadsheets will import the CSV file format. A spreadsheet allows an analyst to perform data transformation and analysis in the same environment, which allows for easier experimentation with an obscure analysis dataset.

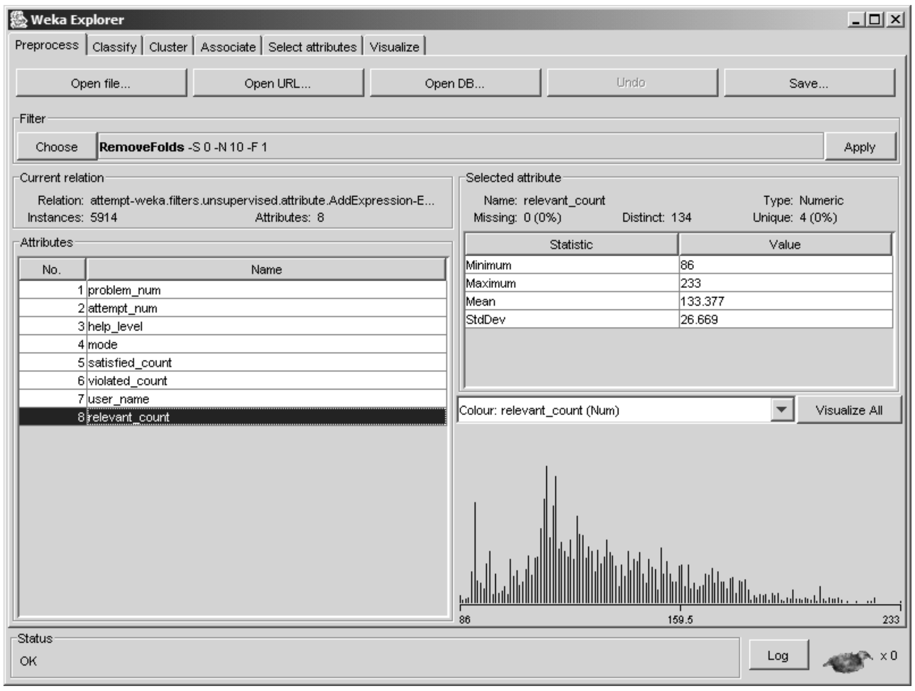
Data mining analysis using a spreadsheet will normally involve importing the analysis dataset into a range of cells (one column per attribute), defining one or more derived attributes in adjacent columns, using other cells to store the results of some aggregate functions on the dataset and linking this data to an information visualisation (such as a graph). By combining these phases into a single tool and providing the analyst with an intuitive user interface, the knowledge-extraction process is made less tedious. This is useful in situations where an analyst wants to gather simple statistics from a variety of different subsets of the data. Instead of cycling through different tools for transformation, statistics gathering and information visualisation, the spreadsheet software allows the analyst to quickly gather the necessary knowledge in a unified environment.

Spreadsheets offer a number of advantages over standard SQL queries, especially when extended analysis of a dataset is required. The main advantage is that attributes (and their derivatives) in a spreadsheet can be easily linked to an information visualisation, allowing the analyst to visualise changes to the dataset as new derived attributes are added, and also as the derivation functions are changed. Spreadsheet utilities also provide much more statistical functionality than standard SQL statements.

4.2 Information Visualisation

One of the oldest techniques for analysing large amounts of information is to represent the data in some visual display. Tufte (1983) describes a range of different techniques for graphically representing information, together with a number of famous examples. Techniques such as graphing have been used to good effect in the past, not only to reveal trends, but also to emphasise key features of a dataset.

Information visualisations make up the least automated class of analysis tools. Essentially, they are another kind of data transformation, since the analysis dataset is simply reformatted using graphing techniques. The actual analysis is left to the viewer of the visualisation, who will ideally be able to recognise interesting features in graphs and 3D representations of data.



Filter: **RemoveFolds** -S 0 -N 10 -F 1

Current relation
Relation: attempt-weka.filters.unsupervised.attribute.AddExpression-E...
Instances: 5914 Attributes: 8

Attributes

No.	Name
1	problem_num
2	attempt_num
3	help_level
4	mode
5	satisfied_count
6	violated_count
7	user_name
8	relevant_count

Selected attribute
Name: relevant_count Type: Numeric
Missing: 0 (0%) Distinct: 134 Unique: 4 (0%)

Statistic	Value
Minimum	86
Maximum	233
Mean	133.377
StdDev	26.669

Colour: relevant_count (Num)

Status
OK

Figure 4.1: A screen image taken from WEKA explorer, showing the Preprocess pane.

4.2.1 Spreadsheet Visualisations

Modern spreadsheet applications allow many different kinds of graphs and charts to be produced from data that has been entered into the spreadsheet. When plotting data using a spreadsheet, the visualisation is linked dynamically to the data content. This means that if the data changes (for instance, if a filtration condition is changed), the chart is updated automatically.

Normally, raw instance data will not be plotted directly on a graph in a spreadsheet. Instead, the data will usually be pre-processed (using the techniques described in the previous section) to aggregate features of the data, and the resulting table of values will be visualised.

For example, one visualisation that was constructed early in the investigation involved plotting the number of attempts against the number of relevant constraints in each attempt. To be able to do this, the number of relevant constraints had to be calculated for each attempt. After this, attempts with the same constraint count were grouped together and counted. This produced a dataset that contained two attributes: the first attribute stored possible values for relevant constraint counts, and the second attribute recorded the number of attempts for which that many constraints were relevant.

Once again, the primary advantage of spreadsheets is that the complete transform-analyse-visualise cycle is facilitated within a single integrated environment.

4.2.2 Visualisation with WEKA

The WEKA workbench also provides visualisation capabilities, through its Explorer interface. The 'Preprocess' pane of WEKA Explorer gives the analyst a range of vital statistics about an analysis dataset. Figure 4.1 shows an example of a typical information display in the Preprocess pane.

In the image, a graph showing the number of attempts at problems with differing levels of relevant constraints is shown. A table summarising the relevant constraint count attribute is also displayed, showing statistics such as the mean and standard deviation for that attribute.

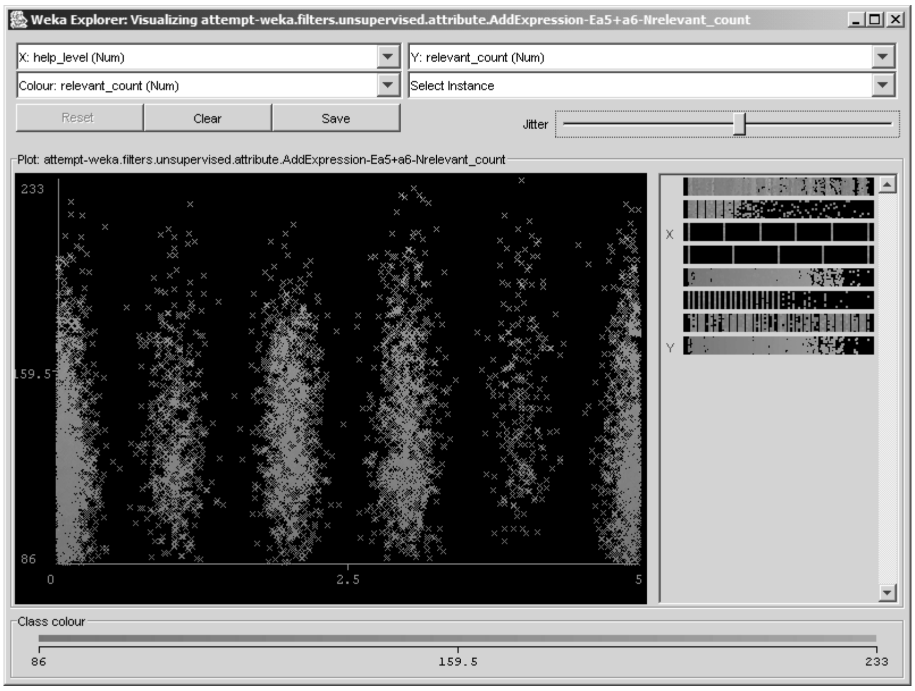

Weka Explorer: Visualizing attempt-weka.filters.unsupervised.attribute.AddExpression-Ea5+a6-Nrelevant_count
X: help_level (Num)
Y: relevant_count (Num)
Colour: relevant_count (Num)
Select Instance
Reset
Clear
Save
Jitter
Plot: attempt-weka.filters.unsupervised.attribute.AddExpression-Ea5+a6-Nrelevant_count
233
159.5
86
0
2.5
5
X
Y
Class colour
86
159.5
233

Figure 4.2: A scatterplot visualisation of feedback levels versus relevant constraints.

The WEKA Explorer interface also provides other visualisations that are able to represent large quantities of raw instance data, unlike most spreadsheet charts. In Figure 4.2, a scatterplot has been used to visualise the varying feedback levels in SQL-Tutor, against the relevant count attribute for each attempt.

In the scatterplot, the "jitter" feature has been used to visually enhance the density of attempts using each feedback level. Jittering offsets each plotted point by some random distance. This allows points that would normally be place on top of each other to be separated. Jittering the graph allows the sparseness or density of instances to be easily evaluated. For example, in Figure 4.2, it is evident that the level 4 feedback state is seldom used by students, whereas levels 0 (no feedback) and 5 (complete solution) are among the most commonly used levels. Furthermore, it is clear that the distribution of attempts within each feedback level is uniform across the range of relevant constraint counts.

4.3 Machine Learning

Machine learning tools are employed in most data mining investigations. A variety of different tools exist, and all are suited to an exploratory analysis of a dataset. The WEKA workbench provides direct support for various machine learning analyses.

4.3.1 Mining Association Rules

Association rules are normally the most useful form of knowledge that can be produced by specialised machine learning algorithms. Such rules associate one or more attributes of a dataset with another attribute, producing "if-then" statements concerning attribute-values. The rule-mining algorithm also assigns quantities (support and confidence) to each rule that it creates, to give an indication of the reliability of each rule.

For example, the following association rule was mined from the entire raw dataset of SQL-Tutor interaction data:

$$(help\_level = 1)[10\%] \quad \rightarrow \quad (attempt\_num = 2)[74\%]$$

This rule has a confidence rating of 74%, and a support rating of 10%. This means that the rule-mining algorithm found that of the 10% of attempts where the previous feedback level was 1, 74% of those instances were on the second attempt at a problem.

The Apriori algorithm was used for all association rule mining analyses in this investigation. The Apriori algorithm works on a set that contains multiple lists of items. The analysis datasets that have been discussed previously are converted into this representation by converting each instance into a list, where the items in the list are attribute-values. This results in a set of lists where each list has a fixed number of items; variations of the algorithm can also be applied to variable-length lists (such as the lists of constraints violated in each attempt).

The algorithm works by identifying *frequent itemsets* in the data. These are groups of items (of a given length) that commonly occur within the lists of items in the analysis dataset. For example, to mine the rule above, the algorithm would have found frequent itemsets of size 1 and 2 that meet a specified support threshold. $\{help\_level = 1\}$ is one of the frequent itemsets that was found, with a support rating of 10% (601 instances). After searching for frequent itemsets of size 2, the set $\{help\_level = 1, attempt\_num = 2\}$ was found, with a total of 443 matching instances. Since the first itemset above is a subset of this, and the confidence level (which is measured by dividing the frequency of the larger itemset by that of the smaller itemset) exceeds a specified threshold, the corresponding association rule is generated.

The thresholds for confidence and support must be set carefully, if sufficient knowledge is to be gained from the dataset. The WEKA implementation sets a default minimum support threshold of 10% of the entire dataset. With a raw dataset of around six thousand instances, itemsets occurring in at least 600 different instances need to be found. It was found that reducing the support threshold to 5% or less often resulted in rules that could prove interesting and useful.

A side effect of the Apriori algorithm is that the frequently used itemsets that are found may also be used directly, to highlight important patterns in the analysis dataset. This particular technique was used to analyse patterns in constraint violations. In that investigation, it was found that WEKA's Apriori implementation was unable to process the large dataset that resulted from using the fine granularity of per-constraint data. Instead, another open-source standalone implementation of the Apriori algorithm, written by Borgelt & Kruse (2002), was used.

4.3.2 Applications of Classification Algorithms

Classification algorithms, such as decision-tree and instance-based learners, can also help to provide insight into some datasets. Each of these algorithms will build up a model of the input data, which is roughly equivalent to a list of the association rules that were described previously.

The main difference between mining association rules and classification rules is that classification algorithms only construct a model for one attribute (the class attribute). Association rules can contain any attribute-value on the right hand side (as the consequent), while the consequents of a classification algorithm's model are always attribute-values of the class attribute. This fact could be used to narrow down an exploratory analysis, which may have started with mining association rules, to a single attribute of interest.

The WEKA workbench provides strong support for data mining using classification algorithms. The workbench provides access to implementations of multiple algorithms, with a variety of different output representations. The model that a classifier produces can take the form of a mathematical function, a set of rules, a decision tree, or some other representation.

For example, the following rules were part of the model generated by the NNge algorithm, after mining the raw dataset, using the attempt number attribute as the class:

```
. . . class 1 IF : problem_num in {58,119,79,80,82} ^ help_level in
```

```
{0} ^ violated_count in {6,0,7,2,5,3,9,12,13}  (19)

class 1 IF : problem_num in {13,69,23,62,118,120,175} ^ help_level
in {0} ^ violated_count in {1,9,12}  (16)

class 2 IF : problem_num in {35,83,94,87,152,155,90,86,157,102,175}
^ help_level in {5} ^ violated_count in {6,1}  (11)

class 3 IF : problem_num in {33,45,48,34,53} ^ help_level in {0} ^
violated_count in {0}  (5) . . .
```

The numbers in brackets after each rule indicate how many instances satisfy that rule. This particular algorithm has created a number of conjunctive rules that could be used for classification, such as "If the problem being attempted is x, y, z and the available help level was 5, then the student violated n constraints."

4.4 Applied Analysis of SQL-Tutor

This chapter has described a number of techniques that can be used to analyse data. It is the role of the analyst to select appropriate techniques and to act on the results if necessary. The human analyst has two key abilities that are crucial in guiding the data analysis phase of data mining:

1. A knowledge of the domain that is being processed, which gives the analyst the ability to decide which knowledge is useless and which knowledge is interesting.
2. Deductive and inductive reasoning skills, which enable the analyst to go through the process of refining the dataset, using different analysis methods, to reach a specified knowledge goal.

The range and depth of knowledge that can be extracted depends on the human analyst's ability to apply these skills, and the richness of the collected dataset.

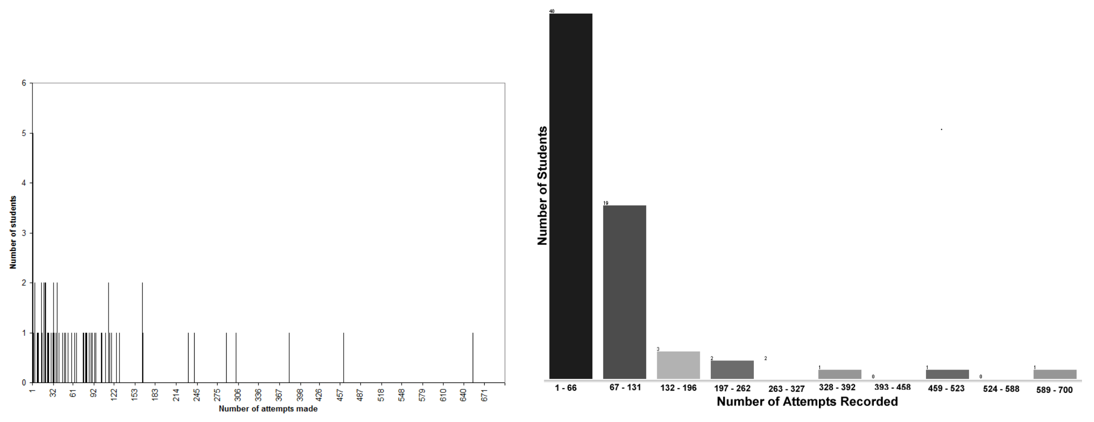
In this section, the previously described analysis techniques are applied in combination, to extract various knowledge from key areas of interest in student interaction with SQL-Tutor. Rather than provide an exhaustive analysis of student interaction with SQL-Tutor, the following examples serve to illustrate a range of methods that can be employed to analyse interaction data. In particular, this section gives examples from a number of different "knowledge areas" within the student interaction data. These could be used as the starting point for a number of extended investigations.

4.4.1 System Usage

One of the first topics that will often interest ITS designers is the evaluation of the popularity of their tutors. The level of usage of a tutor gives a rough indication of student satisfaction, and can lead onto more involved analyses. Data mining at fine granularity allows standard usage statistics to be decomposed, to allow analysis of different parts of a tutoring system.

Basic statistics about the dataset that was used throughout this investigation can be found using SQL queries. In the dataset, a total of 5914 attempts were recorded on the system. The interaction logs of 69 different students were used. This gives an arithmetic mean of 84.5 attempts per student. The distribution of numbers of attempts made by each student was collated using the following query:

```
SELECT count, Count(count) as freq FROM (SELECT Count(problem_num)
AS count FROM attempt GROUP BY student_id) GROUP BY count ORDER BY
count;
```



(a) Non-discretised column graph (b) The same data, discretised using equal-distance binning.

Figure 4.3: A graph that shows system usage, based on number of attempts made.

The resulting analysis dataset was imported into a spreadsheet utility, and converted to a bar chart for visualisation. This chart is shown in Figure 4.3. The chart shows that very few students make more than around 120 attempts. However, a number of outliers exist, with one student having approximately 650 different attempts recorded. The results of this analysis could be used to filter future datasets. For example, some subsets of the raw dataset could be made, to categorise the interaction data based on student type (students with a large number of attempts recorded versus low-usage students, for example). These datasets could then be used to highlight differences between these students.

Using the attempt count to indicate system usage may not always be desirable. Some students may tend to request help more often than others, which will increase their attempt count. The analyst may not consider this to be a true indicator of usage. Other ways of measuring usage include examining the number of problems that each student attempts, and looking at the time spent in performing attempts. Usage could also be analysed from a different perspective, such as 'per-problem' instead of 'per-student'.

Figure 4.4 illustrates the distribution of attempts on different problems in the system (this was produced in a similar manner to the per-student distribution above). The x-axis is ordered by increasing problem number. An immediately striking feature of this graph is that the attempts seem to be concentrated on the set of problems at the beginning, and then drop away dramatically after approximately the 70th problem. It was found that this set of problems corresponded to the first two databases in SQL-Tutor (which were called 'movies' and 'company'). These two databases were used as examples in lectures for the course that the tutor was being used with. Since students should have been familiar with the schemas for these databases, it may be that they tended to focus on the problems associated with those schemas. Figure 4.5 seems to validate this hypothesis to a certain extent. In the graphs, the distribution of attempts at each problem is binned according to the associated problem set (database). Figure 4.5(a) shows the distribution for the 2002 (coursework-related) data. The graph of Figure 4.5(b) was produced from data collected from an online, web-based version of SQL-Tutor, that people from around the world are able to use. The distribution of the first graph clearly shows that the first two databases seem to be preferred over the remaining problem sets. Although the first two categories are also popular in the second distribution, the "cd-collection" database is more popular than the "movies" problem set. A possible reason for the "company" problem set being popular in both cases could be hat it is selected by default.

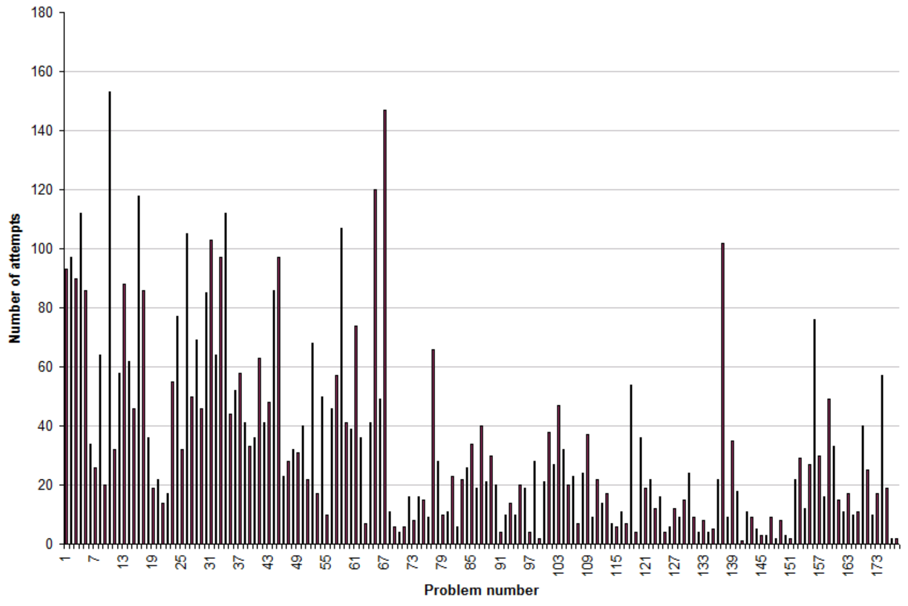
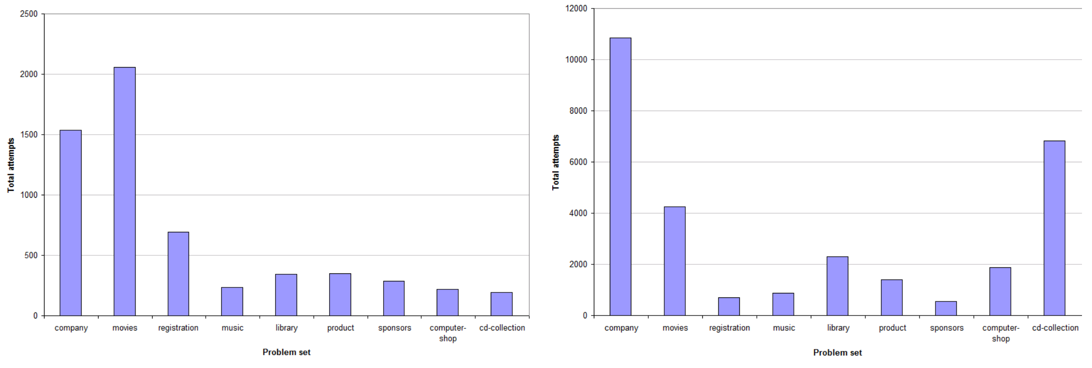


Figure 4.4: A graph showing the number of attempts made by all students at each problem.



(a) Distribution of 2002 SQL-Tutor experiment.

(b) Distribution of attempts gathered from Internet-based SQL-Tutor system.

Figure 4.5: Graphs of total attempts made by students in two different systems, categorised into problem sets.

Rank	Problem Number	Average attempts per student	Number of first attempts
1	67	24.5	6
2	174	14.3	4
3	57	11.4	5
4	137	10.2	10
5	65	10.0	12

Table 4.1: Problem difficulty, based on average attempts per problem

4.4.2 Student Performance

A more direct evaluation of the effectiveness of the tutor (and the traditional technique that is used to evaluate these systems) is to analyse student performance on different problems. Learning ability is normally measured by analysing changes in performance. Patterns mined from the interaction data can reveal groups of problems that students tend to find difficult.

In the previous section, the distribution of attempts at problems was found. These statistics can be used as the basis for an analysis of the difficulty of each problem. One possible measure of problem difficulty is the average number of attempts that a student spends on a problem (on the premise that a more difficult problem is likely to require more attempts, on average). To be able to calculate this, the total number of students that attempted each problem is needed. This can be obtained by filtering out all instances with an attempt number greater than one, and counting the number of instances that exist for each problem number. The difficulty measure can then be computed by dividing the total number of attempts at each problem by the total number of first attempts at that problem. Table 4.1 ranks the top five problems using this measure.

It was noted previously that the attempt count may be affected by the student's attitude to requesting help. An analysis of problem difficulty could take into account the outcome of attempts at each problem. For example, another measure for problem complexity could be the proportion of students that managed to solve the problem (out of all the students that attempted the problem).

4.4.3 Constraints

At a finer level of granularity, the analysis of constraint violations could help to identify specific areas of the course content that students find difficult to learn. Data mining analysis of constraints essentially produces a combined student model for all of the students (or a restricted student base).

As discussed previously, the first phase of the Apriori algorithm can be used to find frequent groups of items in lists. This technique could be used on the variable-length lists of constraints that are violated in each attempt. An analysis dataset consisting of lists of violated constraints for each attempt was used for this purpose. Table 4.2 shows frequent itemsets of four or five constraints that were mined from the raw dataset. A minimum support rating of 1% was used. The table also shows the top ten most violated constraints.

Some interesting results can be found in the output of these analyses. Constraint number 20, which was violated in 5.1% of all attempts in the raw data (ranked 11th), and constraint 142, which was ranked 15th, both appear in a number of highly frequent itemsets. The hint given to the student after a violation of constraint 20 is "When you compare the value of an attribute to a constant in WHERE, they must be of the same type." Other often-violated constraints, such as 7, 142 and 147, are related to the referencing of non-existent attributes in a database. Constraint 239, which also occurs in the frequent itemsets, is violated when the student "need[s] another search condition, using a string constant." This seems to indicate that students often make the mistake of using non-existent attributes in their WHERE clauses. Unfortunately, it also seems that instead of generating a single feedback statement to alert students to this fact, the system generates a range of feedback messages that do not describe the situation accurately (by themselves). For example, Most frequent itemsets with five items:

- 20 372 239 7 147 (64 instances)
- 142 372 239 7 147 (72)
- 142 20 239 7 147 (101)

Most frequent itemsets with four items:

- 142 239 7 147 (197)
- 20 239 7 147 (133)

Top 10 most violated constraints

1. 147 (violated 636 times)
2. 239 (624)
3. 7 (572)
4. 192 (571)
5. 10 (494)
6. 55 (470)
7. 65 (416)
8. 650 (387)
9. 370 (349)
10. 5 (335)

Table 4.2: Groups of constraints (identified by ID number) from SQL-Tutor.

the feedback for constraint 20 may not be appropriate when the attribute in question does not exist.

This analysis could be used as the starting point for the redesign of one or more constraints. For instance, groups of constraints that commonly occur together may be candidates for aggregation into a single, higher-level constraint. On the other hand, single constraints that tend to be violated more frequently than others may be candidates for decomposition into two or more lower-level constraints.

The constraint dataset is quite large and may require a variety of different approaches to be mined effectively. In some cases, it may be prudent to reduce the granularity of the dataset by using an aggregate measure of constraints per attempt. The total number of relevant, violated or satisfied constraints in an attempt could be used for exploratory analysis.

4.4.4 Feedback

The effectiveness and suitability of feedback generated by the system could also be used as a topic for analysis. Feedback is the key mechanism that is used by the tutor to train students. Among the topics of interest in this area are the distribution of different levels of feedback; deviations from the default feedback path; feedback patterns in groups of problems with varying complexity; and improvements (in constraint violations) resulting from feedback. Two examples of feedback

Attempt #	Instances	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
1	1544	88%	0%	2%	3%	0%	7 %
2	992	9%	45%	14%	13%	4%	15 %
3	821	6%	4%	44%	24%	5%	17 %
4	645	5%	3%	37%	31%	4%	19 %
5	458	7%	3%	38%	36%	4%	22 %
> 5	1454	10%	6%	21%	27%	7%	29 %
Total	5914	29%	10%	20%	19%	4%	18%

Table 4.3: Distribution of feedback levels, at different stages of problem-solving

analysis are given in this section.

The Apriori algorithm can be performed on an analysis dataset that contains a feedback attribute, in order to produce some knowledge that relates performance on attempts to the level of help received. The following association rule was mined from the raw interaction dataset.

$$(attempt\_num = 0)[25\%] \quad \rightarrow \quad (feedback\_level = 0)[88\%]$$

This rule states that on first attempts at problems (which make up 25% of the entire raw dataset), the likelihood (or level of confidence) of the student receiving no feedback for that attempt is 88%. This means that only 12% of students explicitly change from the default feedback path on the first attempt. They may ask for the ideal solution on their first attempt (level 5), or they may choose some other form of feedback explicitly.

An investigation related to the knowledge gained above could proceed to analyse the distribution of requested feedback levels, with respect to the attempt number. For example, it is known that 88% of first attempts at a problem receive no feedback. It is unlikely that this would be the case on subsequent attempts. Table 4.3 decomposes the feedback distribution.

If the student follows the default feedback path, they would receive simple feedback (level 0) after their first attempt, an error flag (level 1) after their second attempt, and a hint (level 2) after their third attempt. To proceed to the higher levels of feedback, the student must manually make a selection. Table 4.3 shows that the default feedback path is most popular in the first three attempts, with 45% of students receiving level 1 in their second attempt, and 44% receiving level 2 in their third attempt. An interesting feature of the distribution is that by the 4th attempt, students begin to ask for the ideal solution (level 5 feedback) more than on average. In the distribution of the second attempt, level 5 is the second most popular choice. Generally, if a student asks for the ideal solution, it means that they have given up trying to solve the problem. It seems that around a fifth of all students give up after three attempts at a problem – the ITS designer may consider not allowing the student to select this level of feedback until they complete five or more attempts.

4.4.5 Exploratory Analysis

The knowledge areas that have been described to this point, such as usage, performance and feedback, lead the analyst to a targeted investigation of the data. In some cases, the analyst may (initially) have no constraints on the type of knowledge that is to be found. In an exploratory analysis, the analyst starts by searching for general relationships in a large analysis dataset (normally using a machine learning approach), and refining the investigative procedure. An example of a simple exploratory investigation is given in this section.

The PART machine learning algorithm (Frank & Witten 1998) uses a decision-tree based procedure (based on the C4.5 algorithm) to generate rules for a dataset. It is a classification algorithm, which means the consequence attribute in each rule is fixed. An investigation of problem complexity was carried out by using PART to classify the complexity of the problem in each attempt, based on the attempt number, the constraint violation count and the feedback level.

PART produced a total of 57 rules (with a "pruning" threshold of 1%). The 3-part conjunctive rules that were produced are listed below (in the listing, absolute values for support and confidence are given instead of percentages).

$$\begin{aligned}
(viol\_count = 11) \wedge (attempt = 1) \wedge (feedback = 0)[19] &\rightarrow (complexity = 2)[9] \\
(viol\_count = 1) \wedge (attempt = 1) \wedge (feedback = 0)[147] &\rightarrow (complexity = 2)[104] \\
(viol\_count = 9) \wedge (attempt = 1) \wedge (feedback = 0)[23] &\rightarrow (complexity = 2)[17] \\
(viol\_count = 9) \wedge (attempt = 1) \wedge (feedback = 5)[18] &\rightarrow (complexity = 4)[11] \\
(viol\_count = 10) \wedge (attempt = 6) \wedge (feedback = 3)[3] &\rightarrow (complexity = 2)[1] \\
(viol\_count = 1) \wedge (attempt = 4) \wedge (feedback = 2)[33] &\rightarrow (complexity = 4)[22]
\end{aligned}$$

The second rule in the listing is the strongest, with 104 different instances satisfying both the premise and consequence. The rule states that students often violate a single constraint on their first attempt at problems with a complexity rating of 2.

To investigate this further, the analyst could restrict the dataset to attempts at problems with level 2 complexity. The dataset above was filtered to remove problems without a complexity rating of 2, resulting in a new analysis dataset with 1234 instances (covering 26 different problems). The data was reanalysed using the Apriori algorithm. With a confidence threshold of 75%, the following rules were produced.

$$\begin{aligned}
(attempt = 1) \wedge (viol - count = 0)[13\%] &\rightarrow (feedback = 0)[99\%] \\
(attempt = 1)[33\%] &\rightarrow (feedback = 0)[92\%] \\
(feedback = 0) \wedge (viol - count = 0)[16\%] &\rightarrow (attempt = 1)[82\%] \\
(feedback = 0)[37\%] &\rightarrow (attempt = 1)[80\%] \\
(feedback = 1)[12\%] &\rightarrow (attempt = 2)[80\%]
\end{aligned}$$

This list of rules has two interesting features. The previous analysis showed that first attempts, which only violated one constraint, were common on problems with level 2 complexity. In this analysis, the main finding is that correct first attempts at these problems commonly use the lowest level of feedback. The second interesting point is that the last two rules show that the default feedback path is frequently used for the first two attempts. The previously described study into feedback (Section 4.4.4) found that, overall, only 45% of second attempts at problems used the default (level 1) feedback mechanism. However, the last rule in this analysis shows that 80% of second attempts at problems with level 2 complexity used the default level of feedback.

A conclusion that could be drawn from these findings is that students are more likely to follow the default feedback path (that is, let the tutor determine the correct level of feedback for each attempt) if few constraints are violated. The combined rules from these analyses seem to show that in this set of problems, first attempts which violate either zero or one constraints often occur. After being told that their solution contains "only one mistake", a student could be more likely to make a minor change, and resubmit the solution without explicitly setting the feedback level. If the student is told that their solution is still wrong after their second attempt, they may be more inclined to deviate from the default feedback path. To investigate this further, the dataset used above was filtered so that only third attempts at problems remained. Analysis of the feedback distribution showed that level 2 feedback was only used in 38% of attempts, which is lower than the overall average for third attempts at problems. This adds strength to the claim that the student is likely to ask for higher levels of feedback after finding that a minor change to their solution was not sufficient.

Chapter 5

Conclusion

This report has described a framework for data mining, which can be applied to the analysis of ITS interaction logs. The aims of this investigation were to engineer a process for mining constraint-based interaction logs; to identify alternative techniques at each stage of the process (and to describe the benefits and limitations of each technique); and to identify a range of areas of knowledge discovery for which the mining process can be used.

The investigation identified three key processes that are involved in a data mining experiment: data collection, data transformation and data analysis. It was found that separating the process into these three distinct phases allowed a human analyst to gather knowledge through a process of progressive refinement.

In the data collection phase of the investigation, a model of student interaction was created. This meant that a range of variables relevant to student interaction could be identified and recorded. The goal of the collection phase is to collate as much information as possible. The likelihood of extracting interesting knowledge is increased when more information is available to be analysed.

The transformation phase is concerned with narrowing down the search space to a specific analysis dataset. Data filtration and conversion allows a subset of the raw data to be extracted, so that a particular area of interest can be analysed. Restricting analysis to a particular area of interest allows more reliable rules and relationships to be extracted from the data.

The analysis phase, which is concerned with searching for knowledge in a transformed analysis dataset, can be carried out using a range of different techniques. The type of technique that should be used in an analysis depends on the generality of the search. For instance, if a specific statistic about a specific attribute is being searched for, a statistical analysis tool is warranted. If an exploratory analysis of the interaction data is being conducted, where any knowledge about any of the attributes in a dataset is being searched for, then a machine learning approach should be used.

The process was able to extract knowledge about a number of different features of the SQL-Tutor dataset on which it was applied. The main advantage of data mining analysis over traditional evaluation techniques is that information at a much finer level of granularity is available. Since this also means that more data are available for analysis, a more thorough investigation into student interaction is possible. This report has shown that diverse knowledge about system usage, problem difficulty, constraint violations and system feedback can be extracted from datasets based on student interaction logs.

The process described in this report could be used to investigate a number of tutoring systems. Analysis could be based on previously recorded data from historic tutoring systems, or from the logs of a running system, or even on future systems (that are built using the same ITS framework described in this report). A range of opportunities exist for research that will build on the framework presented in this report.

5.1 Further Research

The collection, transformation and analysis processes described in this report could be used to extract knowledge about many current (and future) intelligent tutoring systems. One of the main areas for further research that will stem from this investigation is to put the framework into practice, in order to find patterns in interaction in a range of constraint-based ITSs (such as SQL-Tutor, EER-Tutor, NORMIT, LBITS and others). Extended analysis, using the techniques that have been described, could produce knowledge that leads to improvement of a system (or systems). Furthermore, with data available from a range of different systems, the scope for comparative analysis is increased. Data mining could be applied to extract knowledge from interaction records from different tutors, from different student groups that use the same tutor, or from sets of records for different versions of the same tutor.

This report has not discussed any findings related to the models generated by the ITSs for each student. A data mining analysis incorporating constraint violation (or satisfaction) histories could produce interesting results. Data mining could also be used to investigate patterns in learning, such as learning rate curves for each constraint. Patterns in the way that students "learn" constraints may prove more useful than traditional performance measurements.

The data collection phase of this investigation processed student log files to extract relevant data. In future systems, this process could be streamlined by building data logging support into the tutoring system, such that interaction information is recorded directly into a relational database (instead of a log file). This may prove useful in tutors that have a large student base, where the system is used on a continuous basis. If student log processing were used in these cases, the entire database would need to be updated at every snapshot. With direct data logging, the interaction data are always kept up to date.

This report has described a number of techniques and tools to carry out data mining on constraint-based ITSs. The analyst's job could be made easier if these tools were integrated into a single application that allowed the entire transformation and analysis process to be carried out. Such a tool would probably require a database query engine to extract appropriate analysis datasets; support for rudimentary statistical analysis (counts, averages, totals and so on); support for mining association rules and frequent itemsets in datasets; and some information visualisation support. Further research into the design and implementation of such a tool could be carried out in future.

5.2 Acknowledgments

The author would like to thank his supervisor, Dr Tanja Mitrovic, for her support and assistance throughout the course of this research. The assistance of fellow post-graduate computer science students also proved invaluable, especially that which was offered by other members of the ICTG. The following people deserve special mention: Lara Rennie for proof-reading assistance; Oliver Hunt and Ronald Highet for help with typesetting; and Konstantin Zakharov for helping the author to understand the ITS software in this study. The terms 'Microsoft', 'Excel', 'OpenOffice' and 'MySQL' are the property of their respective trademark owners.

Bibliography

Agrawal, R. & Srikant, R. (1994), 'Fast algorithms for mining association rules', *Proc. 20th International Conference on Very Large Databases* .

Aha, D., Kibler, D. & Albert, M. (1990), 'Instance-based learning algorithms', *Machine Learning* .

Anderson, J., Corbett, A. & Koedinger, K. (1995), 'Cognitive tutors', *Journal of the Learning Sciences* **4**(2), 167–207.

Anderson, J. R. & Lebiere, C. (1998), *The atomic components of thought*, Erlbaum.

Arroyo, I., Murray, T. & Woolf, B. (2004), Inferring unobservable learning variables from students help seeking behavior, *in* J. Mostow & P. Tedesco, eds, 'Proc. ITS 2004 Log Analysis Workshop', pp. 29–38.

Beck, J., Stern, M. & Haugsjaa, E. (1996), 'Applications of AI in education', *ACM Crossroads* **3**(1), 11–16.

Borgelt, C. & Kruse, R. (2002), Induction of association rules: Apriori implementation, *in* 'COMPSTAT2002'.

Cendrowska, J. (1988), 'Prism: an algorithm for inducing modular rules', *Int. J. of Man-Machine Studies* **27**, 349–370.

Frank, E. & Witten, I. (1998), Generating accurate rule sets without global optimisation, *in* J. Shavlik, ed., 'Machine Learning: Proc 15th Int. Conference', Morgan Kauffman, San Francisco, California.

Heiner, C., Beck, J. & Mostow, J. (2004), Lessons on using ITS data to answer educational research questions, *in* J. Mostow & P. Tedesco, eds, 'Proc. ITS 2004 Log Analysis Workshop', pp. 20–28.

Koedinger, K. & Mathan, S. (2004), Distinguishing qualitatively different kinds of learning using log files and learning curves, *in* J. Mostow & P. Tedesco, eds, 'Proc. ITS 2004 Log Analysis Workshop', pp. 20–28.

Martin, B. (1995), 'Instance-based learning: Nearest neighbour with generalisation', *Working Paper Series, Computer Science, University of Waikato* **95**(18), 90.

Martin, B. & Mitrovic, A. (2002), 'Authoring web-based tutoring systems with WETAS', *Proc. ICCE 2002, Auckland* pp. 183–187.

Mayo, M. & Mitrovic, A. (2000), 'Using a probabilistic student model to control problem difficulty.', *Proc. ITS'2000* pp. 524–533.

Merceron, A. & Yacef, K. (2003), 'A web-based tutoring tool with mining facilities to improve learning and teaching', *AIED 2003* pp. 201–208.

Mitchell, T. (1997), *Machine Learning*, McGraw-Hill.

Mitrovic, A. (2002), 'NORMIT, a web-enabled tutor for database normalization', *Proc. ICCE 2002* .

Mitrovic, A. & Martin, B. (2002), 'Evaluating the effect of open student models on learning', *Proc. 2nd Int. Conf. on Adaptive Hypermedia and Adaptive Web-based Systems* .

Mitrovic, A., Mayo, M., Suraweera, P. & Martin, B. (2001), 'Constraint-based tutors: A success story', *Proc. 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* pp. 931–940.

Mostow, J. (2004), Some useful design tactics for mining ITS data, *in* J. Mostow & P. Tedesco, eds, 'Proc. ITS 2004 Log Analysis Workshop', pp. 20–28.

Ohlsson, S. (1994), 'Constraint-based student modeling', *Student Modeling: the Key to Individualized Knowledge-based Instruction* pp. 167–189.

Pyle, D. (1999), *Data Preparation for Data Mining*, Morgan-Kaufmann.

Quinlan, J. (1986), 'Induction of decision trees', *Machine Learning* **1**, 81–106.

Romero, R. (2003), 'Discovering prediction rules in AHA! courses', *UM 2003* pp. 25–34.

Suraweera, P. & Mitrovic, A. (2002), 'Kermit: A constraint-based tutor for database modeling', *Proc. 6th International Conference on Intelligent Tutoring Systems* .

Tufte, E. (1983), *The visual display of quantitative information*, Graphics Press.

Witten, I. & Frank, E. (2000), *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann.

Appendix A

Source Code

This source code listing made up part of the Java tool that was used to extract data from the SQL-Tutor log files. The methods that were used to process each file are given.

```
. . .
public void parse(String filename) {
    try {
        File thisFile = new File(filename);
        String userName = thisFile.getName().substring(0,
            thisFile.getName().indexOf('.'));
        BufferedReader r = new BufferedReader(new FileReader(filename));
        String nextLine;

        String attemptString = null;
        Map problemMap = new HashMap();

        int problemNum, modeNum, helpLevel, satCount, violCount,
            newTime = 0, oldTime = 0;
        String db = null;
        Date time = null;

        while ((nextLine = r.readLine()) != null) {
            if (attemptString != null)
                attemptString = attemptString.concat(nextLine + "\n");

            if (nextLine.indexOf("Pre-process:") != -1) {
                DateFormat df = new SimpleDateFormat(
                    "hh:mm:ss dd/MM/yyyy");

                time = df.parse(nextLine);
                attemptString = "";
            }

            if (nextLine.indexOf("Feedback level:") != -1) {

                if (time != null)
                    newTime = ((int) time.getTime() / 1000);

                problemNum = numberAfterString("Problem number: ",
                    attemptString);
                if (problemMap.containsKey(new Integer(problemNum))) {
                    problemMap.put(new Integer(problemNum),
                        new Integer(((Integer) problemMap.get(
                            new Integer(problemNum))).intValue() + 1));
                } else {
                    problemMap.put(new Integer(problemNum), new Integer(1));
                }


                db = stringAfterString("Database: ", attemptString);
                if (difficultyMap.containsKey(db)) {
                    if (((Integer) difficultyMap.get(db)).compareTo(
                            new Integer(problemNum)) > 0)
                        difficultyMap.put(db, new Integer(problemNum));
                } else {
                    difficultyMap.put(db, new Integer(problemNum));
                }

                modeNum = numberAfterString("Mode: ", attemptString);
                helpLevel = numberAfterString("Help Level", attemptString);

                satCount = countInsideBrackets("Satisfied constraints: (",
                    attemptString);
                violCount = countInsideBrackets("Violated constraints: (",
                    attemptString);

                attemptString = null;
                int timeDiff = newTime - oldTime;
                if (timeDiff < 0) timeDiff = 0;

                /*
                 * Code here to output record to database (using
                 * JDBC) or to flat file
                 */


                 oldTime = newTime;
            }
        }

    } catch (IOException ex) {

    } catch (ParseException ex) {
      ex.printStackTrace();
    }
}

. . .
```