BAUT: A Bayesian Driven Tutoring System

Song Tan, Kai Qian
CSWE
Southern Polytechnic State University
{stan,kqian}@spsu.edu

Xiang Fu
Computer Science Department
Hofstra University
Xiang.Fu@hofstra.edu

Prabir Bhattacharya
Computer Science Department
University of Cincinnati
bhattapr@ucmail.uc.edu

Abstract—**This paper presents the design of BAUT, a tutoring system that explores statistical approach for providing instant project failure analysis. Driven by a Bayesian Network (BN) inference engine, BAUT analyzes the test cases failed by a student project submission and provides instant diagnosis that guides students in identifying and removing software bugs. Using a parameter learning process, BAUT is able to improve the quality of its analysis. The initial case study with the prototype demonstrates the potential of the system.**

Keywords-Bayesian Network; Automated Tutoring; Verification; Testing; Web Application.

I. INTRODUCTION

Software development and deployment is often a dire process. To hide or to show this complexity is a big question for educators. We advocate the exposure of complexity in computer science classes, as the real-world software development is the best environment for students to sharpen their skills. However, even when a class project is simple, the complexity may get out of the control of an instructor. Just like an old saying, all successful projects are similar, but every unsuccessful one has its own story. The failure of a project can be caused by many factors, e.g., system configuration, network failure, bugs in programs, etc. When students need help with trouble shooting their projects, it is very time consuming for instructors to provide instant help.

We propose the construction of a tool called BAUT (Bayesian Driven Automated Tutoring System) and we outline the design of BAUT in this paper. A research prototype of BAUT, called ProtoBAUT, has been constructed. Our investigation of several case study examples show that it is a promising technique for providing the much needed instant analysis and automated guidance for students when they deal with software failure.

BAUT leverages the use of Bayesian Network. It is developed for web application design classes, but can be extended to other computer science courses. BAUT tackles the following challenge: Given that a collection of test cases have shown that a student project submission has bugs, can guidance/suggestion be automatically generated for helping students in trouble shooting?

For example, if a student's project submission fails all test cases related to operations on a back-end database, the student could be instructed to double check the database connection and configuration (especially the connection string, which is the most probable cause). Such an analysis is feasible because the architecture/technique used by students to tackle a project topic would be highly similar (if not identical). Using tools like Bayesian Network permits training a system. Thus the accuracy of failure analysis can be improved.

BAUT is a continuation of our prior work AWAT [9] and APOGEE [1]. It can be regarded as an enhanced automated project grading tool for web application development classes. Inheriting from AWAT and APOGEE, BAUT takes advantage of Watir [4], an automated testing tool for web applications. Watir drives a Web browser and simulates human tester actions (e.g., clicking a button and entering text into a textbox). Test cases are predefined using excel spread sheet. By running the test cases automatically, student project submission can be graded instantly, automatically, fairly and consistently. Compared with AWAT and APOGEE, BAUT can not only generate the grading report, but also provide accurate failure analysis which guides students to discover and fix software bugs.

Numerous efforts on automated grading have been made during the past decade. Typical examples include Feng and McAllister [3], JavaBat [10], and WebCat [11]. These automated graders do not provide project failure analysis. Our use of Bayesian Network is mainly inspired by the work of C.J. Butz, S. Hua, and R. B. Maguire [2]. In [2], the Bayesian analysis is applied to analyzing the knowledge body and prerequisite relationship among the concepts for a programming class. BAUT applies the Bayesian Network to a different application domain: automated grading and software failure diagnosis.

II. OVERVIEW OF BAUT ARCHITECTURE

This section introduces the architectural design of BAUT, which is under development. A miniature prototype system called ProtoBAUT is available for case study examples.

Figure 1 presents the software architecture of BAUT. It consists of three major components: (1) a repository of Excel specification of test cases, (2) a test engine that relies on Watir and Win32OLE, and (3) a Bayesian Network (BN) inference engine. The BN component itself has its own MySQL database for storing the Bayesian Networks and collecting user responses. The test engine has a collection of web portals (implemented using PHP) which interact with both users and the database of the BN inference engine. Since the BN component relies on MSBNx [5] for assessing and querying a Bayesian Network, the MySQL database serves as the data exchange media.
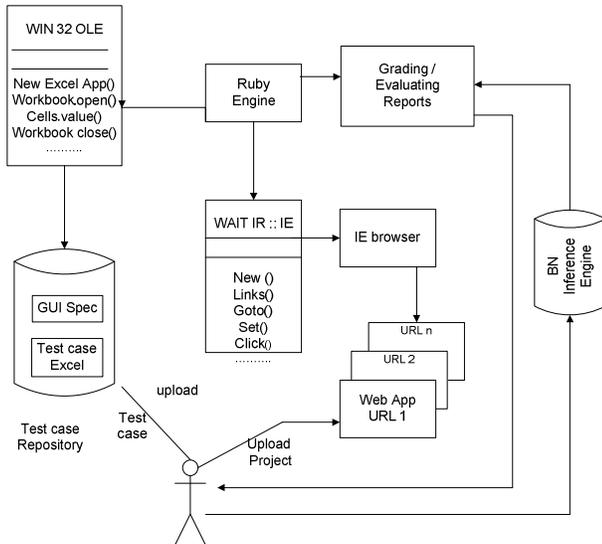

Figure 1. BAUT architecture

As shown in Figure 1, BAUT takes the primary inputs from an Excel document that describes the test cases and URLs of student project submissions. Like AWAT, it drives a web browser and simulates actions of human testers according to the specification given in the test cases. For manipulating Excel spreadsheet specifications, WIN32OLE library is used.

Unlike its predecessor AWAT and APOGEE, BAUT is able to compile a grading report that guides students on the most probable cause of failures. A project analysis report consists of an itemized list of the running results for each test case. In addition, it includes the list of potential causes, ranked by probability.

The web portal also provides a second PHP page (as shown in Figure 2) for evidence collection. Students need to diagnose their software system manually, inspect their code, and fill out the questionnaire and report the real failure causes. It is worthy of note that the guidance information (of

ProtoBAUT) shown in Figure 2 is very brief. It can be extended to provide rich and well-formatted grading report like APOGEE.
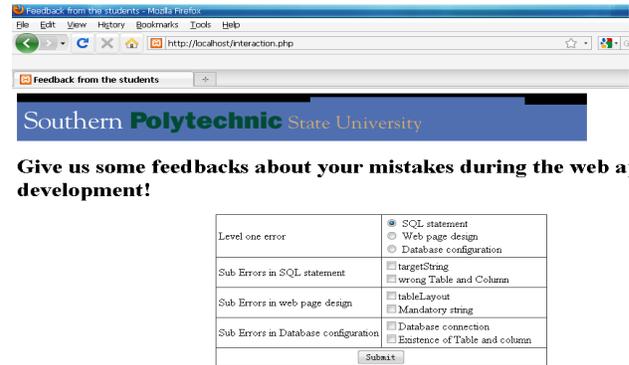

Figure 2. ProtoBAUT Feedback Page

A. Case Study

We briefly describe one sample project as the case study that is used throughout the rest of the paper. Figure 3 shows one example from [9], which evaluates the correctness of a simple bookstore application.

Inputs	Results		
TitleBox1	Submit1		
Start Data			
Java	Java web design	Java basics	**NULL**
C++	Introduction to C++	**NULL**	
Web	Java web design	**NULL**	
End Data			

Figure 3. Test Case Specification

Each test specification starts with a description of test actions (e.g., entering text into TitleBox1 and then clicking button Submit1, as shown in Figure 3). Then each row of the Data section specifies one concrete test case. For example, in Figure 3, the first test case searches for any book with title of "Java" and it expects two listed books.

III. BAYESIAN NETWORK

This section presents the formalism of Bayesian Network, which is used in BAUT for analyzing cause of failures. Each node in a BN represents a random variable that denotes the occurrence of an event. Nodes are connected using directed edges, which represent the probable causal relationship among events. The graph for a BN should be a directed acyclic graph (DAG), i.e., an event cannot be the causal predecessor of itself. When two nodes have no connecting edges, they are regarded as independent from each other. In this paper, each random variable has a Boolean domain.
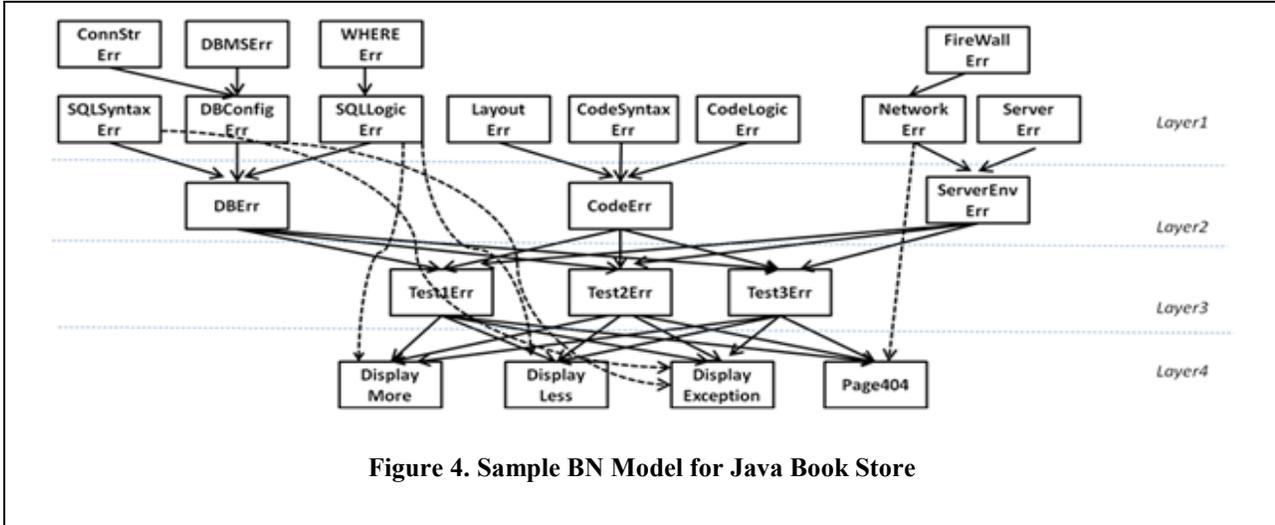
Figure 4. Sample BN Model for Java Book Store

Formally, we use a notation that is slightly different from [8]. A BN is defined as a probability function P over a directed acyclic graph G=(V,E) where V is a set of nodes (random variables), and E a set of directed arcs. Each variable v in V has a Boolean domain {T,F}. Let X be a subset of V, and for each x in V let parent(x) be the set of parent nodes of x in G. P should satisfy the following:

$$P(X) = \prod_{v \in X} P(v \mid \text{parent}(v))$$

Intuitively, the above constraint requires that the probability of a set of events can be computed from the conditional probabilities upon their parents (with causal relations). In practice, the joint probability can be computed from local probability distribution, which is defined as follows. For each node x in V, let U be the set of all functions from parent(x) to {T,F}, the local distribution of x is a function D from U × {T,F} to [0,1] where for each u in U the following is always true:

$$D(u,T) + D(u,F) = 1$$

Intuitively, U represents all the combinations of parents' occurrence. For example, given a 3-node BN like the following, the local distribution of node B can be written as a table in Figure 5, where the sum of each row is always 1.

		T	F
A=T	C=T	0.01	0.99
A=T	C=F	0.02	0.98
A=F	C=T	0.4	0.6
A=F	C=F	0.3	0.7

Figure 5. Local Distribution Table of Node B

In practice, the local conditional distribution is determined by experience. However, parameter learning can be used to fine-tune the causal relation among random variables. We refer readers to [6,7] for advanced probability propagation algorithms. In practice, we used a rather simple BN initialization algorithm, which is similar to [2]. We discuss the details in the next section.

IV. BN INFERENCE ENGINE

In BAUT, Each project should have a separate BN model (dubbed as BAUT BN model). Each BAUT BN model is divided into four layers:

➢ (Layer1) Detailed Cause Layer: This layer includes detailed causes that could lead to software failures. Each node has one and only one child node, which is located in Layer 2 or Layer 1. Exceptions exist for nodes with direct causal arcs to Layer 3 and Layer 4 nodes (explained later).

➢ (Layer2) General Error Category Layer: This is used to classify failure causes.

➢ (Layer3) Test Case Running Results Layer: the pass/failure of each test case is indicated using a node.

➢ (Layer4) Failure Behavior Layer: the details of failure behaviors, e.g., 404 page error, page displaying less than required, page displaying more than required, etc.

Example 4.1 Figure 4 displays the BAUT BN model used for the Java bookstore example in section 2.1. As shown in Figure 4, at Layer 2, there are three error categories: (1) DBErr, which models the failure causes that are related to database, (2) CodeErr, which models the errors that occur in HTML, JSP, and Servlet code, (3) ServerEnvErr which models the error related to the development environment or the server configuration. Each of the test cases mentioned earlier is modeled using a node in Layer 3. The detailed failure behaviors are modeled using four nodes: Page404, DisplayException,

`DisplayLess`, and `DisplayMore`. For example, if the student submission does not display the book title as required, the error is categorized as `DisplayLess`. The classification of detailed error behavior is specified by each individual test case and implemented in test engine using heuristics.

Readers might question the motivation for including "general failure categorization" in Layers 1 and 2. The main benefit is the reduction of complexity. For example, `ConnStrErr` (database connection string error) and `DBMSErr` (database management system not running) can both lead to the `DisplayException` behavior (as database connection could not be established). Classifying them into the same `DBConfigErr` allows lower complexity in defining local probability for `DisplayException`.

Note that Layer 1 nodes may have direct causal arcs (shown as dotted arrows in Figure 4) to Layer 3 or Layer 4 nodes. These links indicate the situation that the occurrence of the parent node (a failure cause) will directly lead to the occurrence of the child node (error), without exception. For example, the `NetworkErr` (e.g., no connection) will always lead to `Page404` error.

A. Initialization of BAUT BN

The construction of BAUT model is a semi-automatic process that consists of 4 steps of work. Steps 1 and 4 are manual, and the rest are automatically completed by a command line tool in BAUT.

1. Model Construction: The instructor, based on his/her past teaching and project tutoring experience, defines the topology (nodes and causal relations) of BAUT BN model. The manual creation process utilizes the BN model editor provided by MSBNx [5]. This results in an XML file as output. The model file could be exposed to students, e.g., to help students get familiar with the possible errors that could cause project failures.
2. Construction of Data Collection Model: This step is automatic. A command-line C# program is used to read from the XML model file and establish a data collection model for initializing and updating the BAUT BN. The database maintains the following information: (1) all nodes in the BAUT BN model, (2) the directed arcs (parent-child relation), (3) layer tag for each node, (4) special causal arcs (represented as dotted arrows in Figure 4), (4) counter for each check box in Figure 2, (5) additional counters for recording false positive rate. This is used for evaluating the effectiveness of BAUT.
3. Automated Update of BAUT BN Model: The program then goes back to re-initialize the local probability distribution table for each node following the rule below: For each special causal arc (dotted arrows) from a node *y* to *x* where *y* in Layer 1: let *U* be the set of functions from *parent(x)* to {*T,F*}, the local distribution information for *x* is a function *D* that satisfies the following: *D(u,T) = 1.0* if *u(y)* is *T*. Intuitively, the rule enforces that any occurrence of *y* leads to the occurrence of *x*.

4. Manual Tuning of Probability Distribution: The last step is manual. The instructor, based on the past experience, revisits each node and re-adjusts the local probability distribution table.

Example 4.2 Figure 6 displays the local probability distribution function of `Page404` in Figure 4. Clearly, the node has 4 parent nodes, including all 3 test case errors and the `NetworkErr` node. Note that in Figure 4, there is a special causal arc from `NetworkErr` to `Page404`. Thus, all entries with `NetworkErr` have the probability of 1 for `Page404` to occur. The other entries are manually reset by instructor. Note that the model even does not have to be sound (e.g., comparing the second and eighth entries) -- imprecise model can still work in practice, as shown by the case study example later. The parameter learning algorithm can also help to improve the precision of the BN analysis.

Parent Node(s)				Page404Err		
Test1Err	Test2Err	Test3Err	NetworkErr	Yes	No	bar charts
Yes	Yes	Yes	Yes	1.0	0.0	
			No	0.155	0.845	
		No	Yes	1.0	0.0	
			No	0.144	0.856	
	No	Yes	Yes	1.0	0.0	
			No	0.134	0.866	
		No	Yes	1.0	0.0	
			No	0.175	0.825	
No	Yes	Yes	Yes	1.0	0.0	
			No	0.124	0.876	
		No	Yes	1.0	0.0	
			No	0.124	0.876	
	No	Yes	Yes	1.0	0.0	
			No	0.134	0.866	
		No	Yes	1.0	0.0	
			No	0.144	0.856	

Figure 6. Local Distribution of Node "Page404Err"

B. BN Parameter Learning

BAUT uses the portal page as shown in Figure 2 for parameter learning. Its purpose is to fine-tune the local probability distribution table for each node so that BAUT BN can provide accurate estimate of failure causes. Note that the hit counter for each event is maintained in the database. This allows us to directly obtain the probability of any combination of events. Thus, BAUT can compute the value for each entry in a local distribution function. The algorithm, described below, uses an idea similar to [2].

Given a node x and let one entry be represented by a function u from parent(x) to {T,F}. Then we denote the set of variables with true value by TS(u) = {v | u(v)=T and v in parent(x)}. (Intuitively, TS means "true set"). For example, for the second entry in Figure 6, TS(u) is {Test1Err, Test2Err, Test3Err}. Let D be the local distribution function. Then D(u,T) is defined using the following formula:

$$D(u,T) = P(x \mid TS(u)) = \frac{P(\{x\} \cup TS(u))}{P(TS(u))}$$

Note that $P(\{x\} \cup TS(u))$ should be interpreted as the probability that x and the events in TS(u) all occur. For

example, let u be the second entry in Figure 6, then D(u,T) is:

$$\frac{P(\text{page404, Test1Err, Test2Err, Test3Err})}{P(\text{Test1Err, Test2Err, Test3Err})}$$

Clearly, the probability of all the four events (`page404`, `Test1Err`, `Test2Err`, `Test3Err`) occurring can be calculated from the data in the database.

C. Case Study of BN Inference

Bayesian inference using BAUT BN can be done in two ways: (1) direct evaluation using MSBNx editor [5], and (2) interaction via Web portal (as shown in Figure 2). We briefly describe the algorithm design of approach 2. Once a student project submission is automatically evaluated using the test engine, for each test case, the following data are collected: (1) whether the test case passes or not, and (2) the *last* error behavior (it has to be one of the `Page404`, `DisplayException`, `DisplayLess`, `DisplayMore`). Error behaviors are captured using heuristics based on test case specification. Its implementation is straightforward using regular expressions. Clearly, the above data can be used for generating the valuation for Layer 3 and Layer 4 nodes. These nodes are then used as the evidence input. The inference process is achieved calling the MSBNx API [5]. In the following, we show several case study examples of the BN inference.

Example 4.3 When a project submission fails all three test cases with 404 error (page unavailable), BN inference identifies `ServerEnvErr` as the most probable cause:

Category	DBErr	CodeErr	ServerEnvErr
Probability	0.9251	0.8341	0.9877

Then by delving into the detailed cause of the `ServerEnvErr`, the following are discovered for each detailed cause:

Cause	NetworkErr	ServerErr	FirewallErr
Probability	0.9356	0.3162	0.1711

The data is commensurate with our past experiences that among the server and environment configuration, the network connection failure is the most probable cause.

Example 4.4 When a student project submission fails all three test cases with `DisplayException` for all three test cases. Inference by BAUT identifies `DbErr` as the most probable cause:

Category	DBErr	CodeErr	ServerEnvErr
Probability	0.9729	0.8425	0.5918

Discussion: Manual modeling of BN nodes may be imperfect. However, in practice, the imperfect model can still produce meaningful results and relatively accurate analysis. For example, many students use string concatenation in servlet for producing SQL queries. Thus,

the model should have included a direct arc from `CodeLogicErr` to `SqlSyntaxErr` and `SqlLogicErr` in Figure 4. However, in most cases, this does not affect the guidance provided by the BN model. The is because as long as test cases fail, the inference (increase of probability) can propagate (backward) through the links originating from `CodeLogicErr`.

On the other hand, in the case study, we did find one significant defect of the model in Figure 6 in analysis. Assume that a project implementation contains the following hard coded query:

```
SELECT * from bookstore
WHERE title like "%Java%"
```

Clearly, the hard coded query did not handle the input parameter about keyword to search. The submission, however, passes `Test1` and fails `Test2` and `Test3`. For `Test2` and `Test3`, both test cases exihit `DisplayLess` error. Intuitively, such a scenario should imply very low probability for `SQLSyntaxErr` but high probability for `SQLLogicErr` and `SQLWhereErr` (`WHERE` clause error). However, the current model in Figure 6 cannot distinguish the difference between the above and the case where all three cases fail at `DisplayLess`. By replicating the sets of behavior nodes for each of the test cases we might solve the problem.

V. CONCLUSION

This paper presents the design of BAUT, an automated tutoring system for providing automated project failure diagnosis to students using Bayesian Network. We show that it is feasible to apply BN inference because the knowledge domain is quite stable. By asking students to provide feedback on the BN model, students are engaged in active learning. They feel excited about providing failure analysis feedback which can be used by their peers.

BAUT is currently under development, with its prototype system ProtoBAUT available for assessing case study examples. The future direction of the project includes completion of the entire BAUT system, improvement of its parameter learning algorithm, and the integration of the project with the APOGEE system [1].

ACKNOWLEDGMENT

REFERENCES

[1] X. Fu, B. Peltsverger, K. Qian, L. Tao, and J. Liu. "APOGEE: automated project grading and instant feedback system for web based computing." In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2008)*, pp. 77-81. March 12-15, 2008.

[2] C.J Butz, S. Hua, R. B. Maguire . "A web-based Bayesian Intelligent Tutoring System for Computer Programming", vol 4, issue 1, pp. 77-97, Web Intelligence and Agent System, 2006.

[3] M. Y. Feng, and A. McAllister, "A Tool for Automated GUI Program Grading", In Proceedings of the 36th ASEE/IEEE Frontiers in Education Conference, October 28-31, 2006.

[4] WatirCraft. "Web Application Testing in Ruby." Available at http://wtr.rubyforge.org/watir_user_guide.html, retrieved June 2009.

[5] Microsoft, Microsoft Bayesian Network Editor, available at http://research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx, retrieved June 2009.

[6] F.V. Jensen, S.L. Lauritzen and K.G. Olesen. "Bayesian updating in causal probabilistic networks by local computations." *Computational Statistics Quaterly*, pp. 269-282, vol. 4, 1990.

[7] G. Shafer and P.P.Shenoy. "Probability Propagating." *Annals of Mathematics and Artificial Intelligence*, vol. 2, 1990.

[8] K. B. Korb, A. E. Nicholson. "Bayesian Artificial Intelligence." ISBN-10: 1584883871, Chapman & Hall/CRC, 2004.

[9] M. Sztipanovits, K. Qian, X. Fu. "The automated web application testing (AWAT) system." In Proceedings of the ACM Southeast Regional Conference 2008, pp. 88-93, March 2008.

[10] N. Parlante, "JavaBat - Java Practice Problems." Available at http://www.javabat.com/.

[11] S. Edwards. "Using software testing to move students from trial-and-error to reflection-in-action." In *Proceedings of the 35th SIGCSE Technical Symp. Computer Science Education*, ACM, 2004, pp. 26-30.