Question Difficulty Assessment in Intelligent Tutor Systems for Computer Architecture

Tao Li[1] and Sam Sambasivam[2]
Computer Science Department, Azusa Pacific University
Azusa, CA 91702, USA

Abstract

Assessing the difficulty of an exercise question is a subjective process and is usually done by the instructor based on experience. An accurate assessment of the difficulty of exercise and exam questions is important and will help to better allocate credits to assignments and exams. Our contribution is in defining a relatively objective approach to assessing question difficulties. Our approach applies to courses in many disciplines and can be automated with computer software.

Keywords: intelligent tutor, automatic question generation, difficulty assessment, Java, guided problem solving

1. INTRODUCTION

In this paper, we present a system for tutoring computer architecture and a novel and practical approach for assessing the difficulty of exercise/exam questions. Although there has been much research and development effort for computer assisted learning and intelligent tutoring, for example [1,2,3], the difficulty assessment problem has not been systematically studied. Our approach to assessment can be automated and be embedded in an intelligent distance learning system. This approach has been used in our computer architecture learning assistant system. We are also considering applying the same approach to learning systems for operating system and computer networking. It is clear to us that our approach is also applicable to basic physics courses, electronics, etc.

Specifically, our approach is designed for course subjects that are based on well-formulated concept hierarchies. We allow a combination of fixed object composition and dynamic multiple instantiations of objects. This approach covers a wide range of courses. However, it is not readily applicable to subjects like algorithm design and data structures that require the development of good algorithms.

This paper is organized as follows:

- We first give a detailed presentation of the difficulty assessment problem and our assessment methods.
- An introduction to our computer architecture tutoring system and its underlying knowledge structures is next presented.
- Algorithms used in guided learning and automatic question generation are discussed. The authors feel that this discussion is important since the difficult assessment method is closely attached to the algorithms.
- A guided problem solving approach used in our system is also discussed.
- A summary about our system is given in the end.

[1]Email: tli@apu.edu, [2] SSambasivam@apu.edu

The system employs a concept graph and an associated equation hierarchy. Unlike most educational software that uses a limited set of assessment questions generated by the authors or instructors, the key component of our system is the automatic question generator that also constructs a data structure to help the students with guided problem solving.

2. DIFFICULTY ASSESSMENT

The types of knowledge structures needed in solving most problems in science and engineering fall into two categories: static structure and dynamic structure. These two types of structures are discussed below. We propose a method for difficulty assessment when static structures are used. This approach can be regarded as objective when other methods are absent in current tutoring systems.

Problem Solving with Static Knowledge Structure

Most problem solving in primary and middle school mathematics can be characterized as using static knowledge structure since they typically involve only a few steps using the basic arithmetic operations. The concepts involved in physics and chemistry problem solving are more than those in primary and middle schools, but the number of steps are also limited.

Some courses in computer science are similar to physics and chemistry, for example, computer architecture and data networking. The quantitative side of these courses require only static structures. Although arrays may be used and the structures are typically constructed at run-time, these structures are derived from a moderate size concept graph that is known. This type of structure is common in many courses and is the emphasis of our paper.

Problem Solving with Dynamic Knowledge Structure

This type of structures is often encountered in courses such as geometry, calculus, and engineering courses. Many steps may be involved in solving a problem, and may formulae and postulates may be required the process.

In this type of problem solving, a student is typically given the following resources:

1) A set of operators that can be applied to perform transforms in problem solving state space. Each operator has a condition and a transformation. When the condition is satisfied in the current state, the transformation may be applied to generate a new state.
2) A set of postulates or axioms that form the basis of the state space. The inference starts from these.
3) Some heuristic rules or algorithms to guide the problem solving process. These rules guide the selection of operators in each step.

A student with good problem solving skills typically possesses good heuristics and follows logical inference.

There is no known static structure defined for this type of problem solving. This is partly due to the high complexity of the state space and the large number of postulates and operators. Efficient heuristic search is typically used in computer solution of such problems. Searching for a sequence of steps that lead to a solution of the Rubic's Cube exemplifies this situation. Readers may refer to an AI text such as (Russell, S. and Peter Norvig, 2003) for a discussion on heuristic search.

Static Knowledge Structure in Computer Architecture

We use the functional concept graph in our teaching system. This is an extension of the Conceptual Graph structures [4]. A functional concept graph **G** consists of a set of nodes **V** and a set of directed edges **E**. The edges connect the nodes of **V**. In this paper, **G** is always a Directed Acyclic Graph (DAG).

The functional concept graph is a hierarchical data structure: each node n_i of the DAG is associated with a level l_i. Node n_i may have a set of incoming edges $\{e_{i,1}, e_{i,2}, \ldots e_{i,k}\}$, connecting lower level nodes. A node without any incoming edge is a *source* node. A node is also associated with a value. The value v_i of a node n_i is computed by a function $v_i = \psi_i(v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, where $v_{i,j}$ is the value of the input node n_j connected by edge $e_{i,j}$. The set of input nodes form the set of partners for node n_i. Associated with each

node n_i is also an interval (l_i,u_i), which is empirically determined by the author of the tutoring system.

An inverse function ϕ_{ij} is defined, when appropriate, for an input edge $e_{i,j}$ such that $v_{i,j} = \phi_{ij} (e_{i,1}, \ldots, e_{i,j-1}, n_i, e_{i,j+1}, \ldots, e_{i,n})$. This value can be assigned to the corresponding partner node n_j. The existence of inverse functions may bring an extra degree of flexibility and allow the system to generate more sophisticated questions. An example of a functional concept graph is shown in Figure 1. A larger concept graph for computer architecture is given in Figure 2.

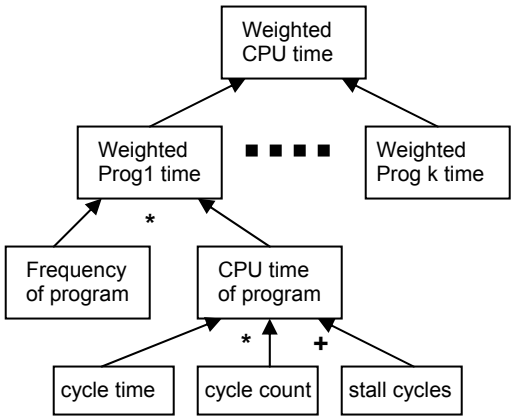

Figure 1. An example of concept graph.

In the example of Figure 1, the weighted CPU time is defined as the sum of the weighted program time of the k programs. The weighted program time γ_i of program i is the CPU time of executing the program times the frequency of execution F_i of the program. The CPU time T_i is defined as the product of cycle time T_c and (cycle count + stall cycles). We denote cycle count by N_c and stall cycles by N_s. For programs executed on the same CPU, the cycle time is the same.

We extend our knowledge structure to allow the grouping of input nodes. Specifically, we have two types of nodes, AND nodes and OR nodes. To compute the output of an AND node, all input values must be present. The equation associated with an AND node represents a function of all the inputs. On the other hand, the output is computed with any one input value at an OR node.

There is a value interval for each node which needs be specified by the author. A random number is generated within the interval of a node if the question generation algorithm decides to terminate at the node. An AND group may have a variable number of input nodes that are created at run-time.

Difficulty Assessment with Static Knowledge Structure

In this paper, we address the issue of difficulty assessment in problem solving with static knowledge structures.

Practically, the measuring of the degree of difficulty is a subjective matter. However, it is beneficial to make the measuring procedure as objective as possible. Without a proper knowledge model, it is difficult to realize this goal.

We find, from the experience of using exercise questions from some textbooks, that students tend to find a question more difficulty when more concepts are involved and more equations are used in solving a problem. This is intuitively easy to understand since it takes more time organize thoughts when more items are involves and more steps are needed in solving a problem. Of course, the complexity of each equation also adds to the degree of difficulty. But this has less impact on the overall difficulty.

Hence, we define the degree of difficulty as $D = w_1N + w_2P + w_3M$, where N is the number of conditions given in the questions (that is, the number of terminal nodes), P is the number of downward edges in the paths traversed during question generation, and M is the number of upward edges traversed during question generation. We use three weight factors w_1, w_2 and w_3 to balance between the path length and the number of conditions. Typically, $w_1<w_2<w_3$ because upward edges traversed represent more difficult concept association and downward edges, and the total number of edges, which corresponds to the number of problem solving steps involved, carries more information about the effort needed in problem solving.

This definition of degree of difficulty serves only as a rough guidance during automatic question generation. It is by no means a perfect objective measure. However, it does help in assessing the student's understanding of a certain subject. If a student is able

to solve a problem involving most concepts and equations about a subject, he or she should have a reasonably good understanding of the subject. We end this section with the following example.

Example. Consider the weighted CPU time in Figure 1. Assume that we have three programs, P1, P2, and P3, under consideration. The total Weighted CPU Time Φ is given.

For program P1, P2 and P3, the student is given N_{s1}, N_{c1}, N_{s2}, N_{c2}, N_{s3}, and N_{c3}. In addition, F_1 and F_2 are also given. The student is asked to find F_3. This question has 9 given values (conditions).

To compute F_3, one must know γ_3 and T_3. Since $\gamma_3=F_3*T_3$, $F_3=\gamma_3/T_3$. This requires one upward move to the node for γ_3. Now, γ_3 can be computed from Φ, γ_1 and γ_2 as $\gamma_3=\Phi-\gamma_1-\gamma_2$. This requires another upward move to the node for Φ. A total of two upward moves are needed.

To compute each of T_1, T_2 and T_3, one must traverse three downward edges. To compute each of γ_1 and γ_2, two downward edges are traversed. From Φ to γ_1 and γ_2, two downward edges are also traversed. This results in a total of $(3\times3+2\times2+2)=15$ downward edges.

We assign $w_1=1$, $w_2=5$ and $w_3=25$. The total complexity of the problem is thus $1\times9 + 5\times15 + 2\times25 = 134$.

3. AUTOMATIC QUESTION GENERATION

Our system contains an automatic question generator which is able to generate questions based on the augmented concept graph. Although not exhaustive, the questions to be generated in our computer architecture course typically fall into the following two types:

- direct evaluation of a result from given conditions. The result can be compared against a value or a range. This requires the construction of one concept structure and the evaluation of a single result.
- comparison of two or more results evaluated from different conditions. Typically, this requires the building of two or more concept graphs and the evaluation of several values. The values are then compared. This differs from the previous type only in the number of concept graphs.

A third type of question requires to find the relationship between a certain node n_c and some other nodes in the graph and to plot the values of the node n_c with respect to the values of those other nodes, in a given range. We do not include this type of questions in our automatic question generation algorithm, nor do we include design questions.

Since the direct evaluation type and the comparison type differ only in the number of concept structures, we will concentrate on question generation with one concept graph. The discussion of our automatic question generation algorithm is presented below. The algorithm is presented for ease of understanding, but it is not quite complete.

An integer named ***credit*** is what we use to represent complexity. The larger is this number, the more complex the question is. We use a queue OPEN to store those nodes that are not yet expanded. A node is expanded if its descendants are generated. We assume that each node N_i stores an estimated maximum difficulty D_i that is obtained by downward expanding the concept graph rooted at this node to the maximum. In addition, we assume that we have two basic types of nodes in a concept graph: the *array* nodes and the *regular* nodes. An array node may have several descendants of the same class and a result is computed from the descendent values. A regular node has descendants of different classes.

Algorithm *QuestionGeneration*(credit)
1. Select a node N_r from the entire concept graph and mark it as "closed".
2. If D_r < credit, then
   Traverse upward;
   credit = credit–w_3;
   Push the parent of N_r to OPEN;
   else
   Randomly decide traverse up or down to node N;
   Push the descendants of N to OPEN;
   credit = credit–w_2* num_descendants;

3. Dequeue a node N from OPEN and mark it as closed.
4. If D >= credit, then
   Select a random value in the interval specified in the node for each node in OPEN queue;
   Finish question generation;
   else
   Randomly decide traverse up or down to node N_d;
   Push the descendants of N_d to OPEN;
   credit = credit-w_2*num_descendants;
5. Go go step 3.

Remark: the credit value in fact denotes the desired question difficulty. The difficulty of the automatically generated question may exceed the desired difficulty by a small percentage.

A good question generator should possess these three characteristics: domain consistency, correctness, and completeness of coverage. Domain consistent algorithms generate questions that produce results that are mostly contained in the specified empirical value intervals. If this is the case for every node, we say the value domains are consistent. In other words, if we randomly choose, for those nodes in the input conditions, the values in the corresponding interval, we expect that the result value to be computed also lies in the specified interval of the destination node. Domain consistency depends on the experience of the system authors.

A correct question generation algorithm must not generate questions which are not solvable based on the given concept graph. This is important because students often cannot determine whether or not a question is solvable and they may waste much time trying to solve an impossible problem.

When the algorithm is able to generate questions that exercise every function of the DAG and relate every concept in the hierarchy (in many questions), we say it has a complete coverage.

Our automatic question generator will not generate any unsolvable question. In addition, it provides domain consistency and completeness. This is a very important point and the users will not waste time to tackle an unsolvable problem and will not suffer from the frustration of not being able to solve a problem after devoting a significant amount of time. In addition, it also guarantees that the question generator will provide a wide variety of questions for the users.

The screen shot of an automatically generated question is shown in Figure 3.

4. GUIDED PROBLEM SOLVING

One of the major hurdles in building a successful tutoring system is the diagnosis of mistakes made by students. It is extremely difficult to build a comprehensive knowledge base that can model unpredictable student behavior. A typical student model does not cover many unexpected behavior.

In our opinion, it is too difficult or nearly impossible to build a student model that covers all the possible student behavior and relevant knowledge in problem solving. It is more convenient to provide the students with a set of tools (such as definitions and equations) that they can use in solving problems of a particular subject. We call this system guided problem solving and diagnosis. Although this may occasionally limit the student's creativity, the pay-off is significant — the system can diagnose more mistakes a student makes during problem solving and provide more sensible advice to the student.

Our guided problem solving algorithm utilizes the data structure generated by the automatic question generator. It is essentially the reverse of the question generation process. The guided learning process is closely coupled with the graphics interface.

After a question is automatically generated, we keep the data structure and the OPEN queue. If the user chooses to enter guided learning mode, the system initializes the user interface so that the equations for the nodes on the OPEN queue are visible and other equations are not visible. This helps to narrow down the search range.

The user may select one equation and put an answer in there. If the result of calculation is correct, the system will fill the intermediate result in a table. This helps the user to remeber intermediate values. The node is then eliminated from the OPEN queue and

the corresponding equation, if not used by any other node in OPEN, becomes invisible. When the user enters an incorrect intermediate result, the system will inform the user about the mistake.

This process continues until all the nodes are removed from the OPEN queue. At this time, a correct final result is arrived at. A screen shot of a step in guided problem is shown in Figure 4.

5. SUMMARY AND CONCLUSION

An objective method for assessing the difficulty of automatically generated questions is presented here. This method has been applied to implement a novel intelligent tutoring system for computer architecture learning. Our system is able to automatically generate a large variety of questions from the knowledge base. This approach is applicable to tutoring many subjects in science and engineering.

We designed robust algorithms for automatic question generation in statically structured systems. We also designed a system guided learning approach that is closely attached to the automatic question generator. Part of the knowledge structure has been implemented. The system with a partial knowledge is working and it demonstrates the feasibility of our approach.

We are formulating an approach to assess the difficulty of questions that require dynamical application of many operators, and we will apply the approach to solving difficult questions in geometry.

6. ACKNOWLEDGMENT

The authors wish to acknowledge the effort of our students for implementing the computer architecture tutoring system.

7. REFERENCES

Frasson, C., G. Gauthier and A. Lesgold (eds.) (1996). *Intelligent Tutoring Systems*, LNCS-1086, 3rd International Conference on Intelligent Tutoring Systems (ITS'96), Montreal, Canada, June.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann.

Larkin, J., R. Chabay and C Sheftic (eds.) (1990). *Computer Assisted Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration*, Erlbaum.

Sowa, J. (1984). *Conceptual structures: Information Processing in Mind and Machines*, Addison-Wesley, Reading, MA.

Russell, S. and Peter Norvig (2003). Artificial Intelligence: A Modern Approach, Prentice Hall, Upper Saddle River, NJ.

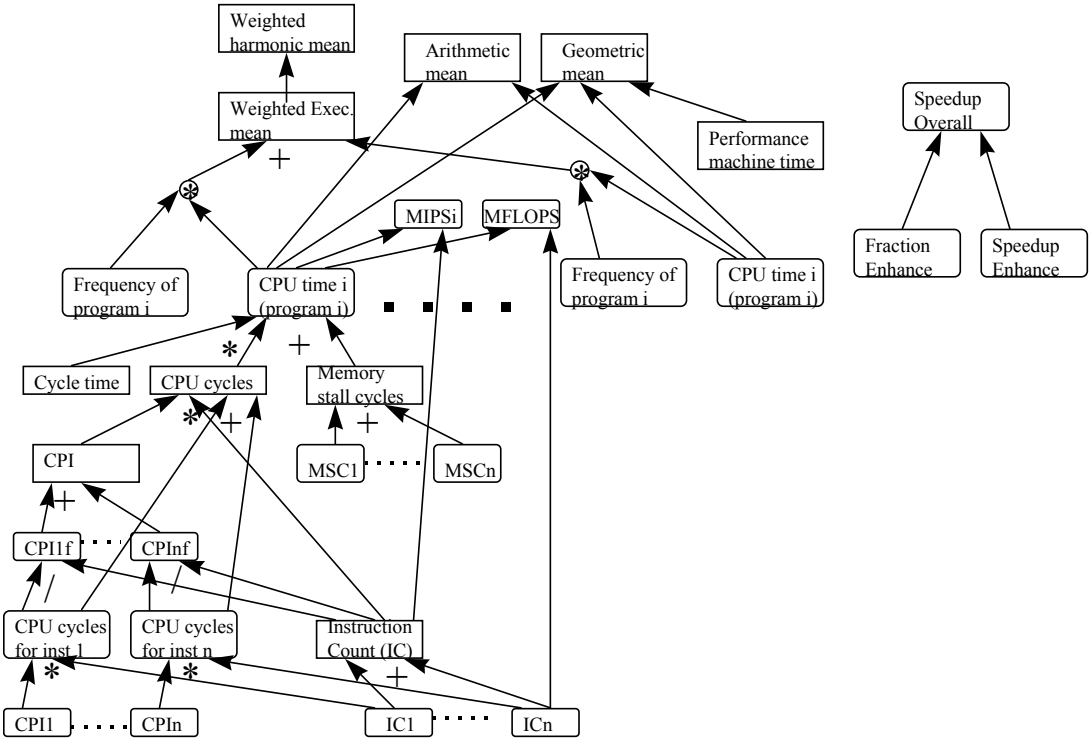

Figure 2. A larger concept graph.

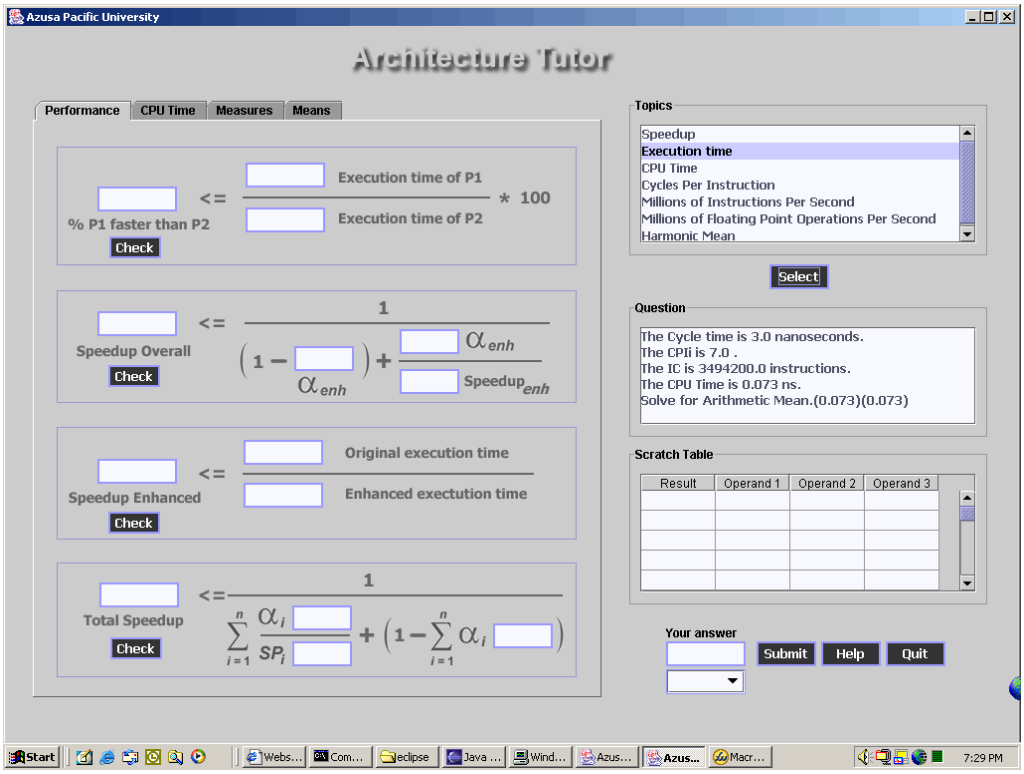

Figure 3. An Example of Automatically Generated Question

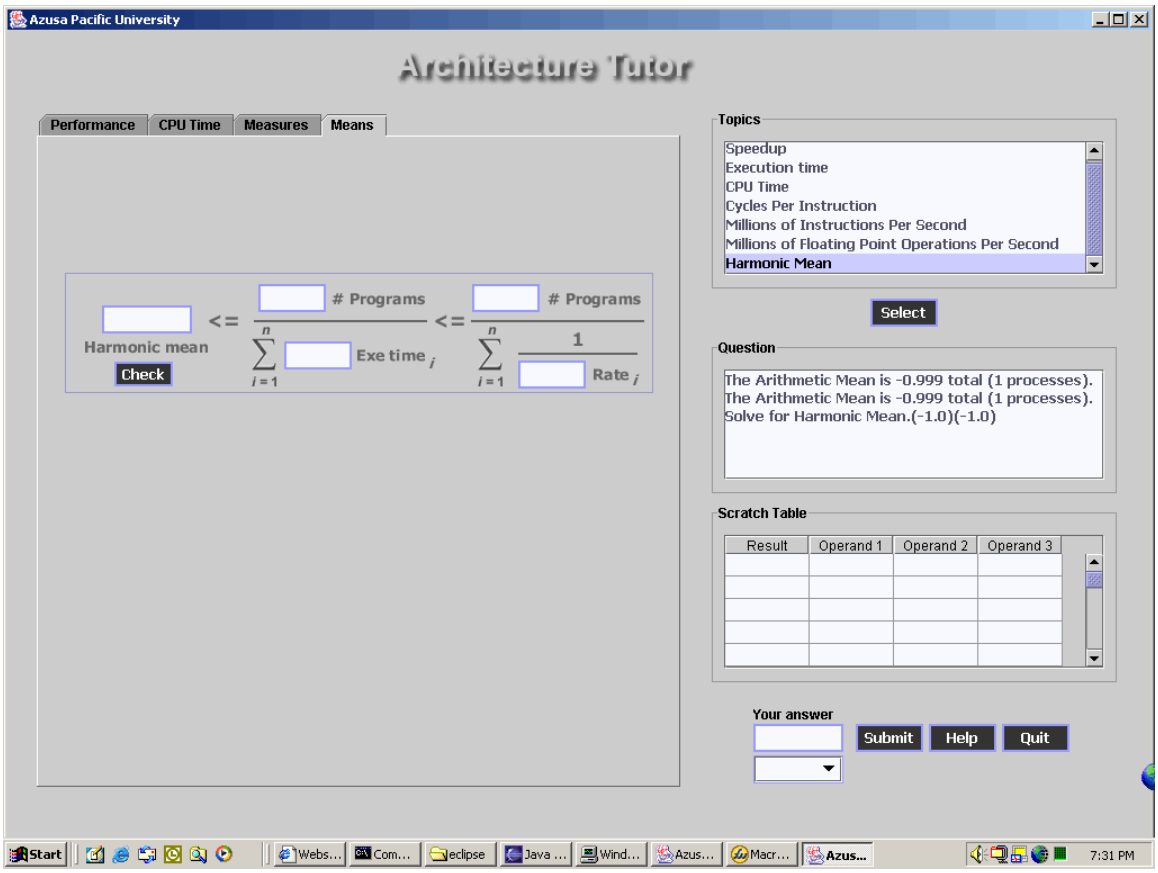

Figure 4. An Screen Shot of Guided Problem Solving.