Problem Generation: Evaluation of two domains in Operating Systems

Amruth N. Kumar
Ramapo College of New Jersey
505, Ramapo Valley Road
Mahwah, NJ 07430-1680
amruth@ramapo.edu

Abstract

Our work on courseware involves generating problems to test a user's learning. We propose to generate these problems dynamically in order to present an endless supply of problems to the user. In order to facilitate the user's learning, we want to vary the "challenge" of the dynamically generated problems. Our current task is to identify a measure of "challenge" in problems on a topic, so that we can systematize and automate variations in the challenge of dynamically generated problems.

We propose to use a measure of "challenge" of problems based on analysis of the knowledge/ability necessary to solve problems in a given domain. By limiting such knowledge/ability to the minimum necessary to solve a problem, we attempt to avoid counting as evidence of learning, any serendipitous solving of problems.

In this paper, we analyze two different problem domains in Operating Systems in an attempt to derive measures of "challenge" for problems in those domains: storage placement, and processor scheduling. For these domains, we identify taxonomies of problems based on the minimum knowledge/ability required to solve them. We tentatively generalize the taxonomies to derive a measure of "challenge" for a *class* of scheduling problems dealing with discrete quantities.

Introduction

Problem-solving is an integral part of learning in science. Any intelligent tutoring system for science must not only present problems for the learner to solve, but also be able to solve problems on-line in order to provide feedback to the learner.

Problems may be produced in a tutoring system by either selecting them out of a canned repository or by dynamically generating them. Dynamic generation has several advantages over selecting from a canned repository:

- the supply of problems is infinite;
- it is less likely that a learner will see the same problem twice, which is more conducive to increasing the confidence of the learner in the tutoring system.

When a repository of canned problems is provided, the challenge of the problems in the repository may be rated by a human being. When problems are generated dynamically, this rating would have to be automated. Hence, our interest in defining the notion of "challenge" of problems, and quantifying it so that it can be automatically measured and used.

In this paper, we consider two problem domains in Operating Systems: storage placement and processor scheduling. In each domain, we first catalog the knowledge/skills necessary to solve problems. Then, we derive a taxonomy of problems for the domain based on the level of "challenge" of the problems. We tentatively generalize the taxonomy to domains with discrete values and scheduling tasks: domains typical of Operating Systems topics. Finally, we report on our implementation of problem generators for the two domains, which are tuned to vary the challenge of generated problems while appearing to be random and unpredictable in nature.

"Challenging" Problems

The "challenge" of a problem is its difficulty as perceived by the learner. We define the "challenge" of a problem as a measure of how little a learner may know and still manage to correctly solve a problem. We are allowing for not only the case where a learner consciously solves a problem by applying the relevant knowledge, but also the case where the learner "stumbles upon" the correct answer by either improperly applying the relevant knowledge, or applying some irrelevant knowledge.

In order to quantify the amount of knowledge/ability needed to solve a problem, we first list the "quantums" of knowledge applicable to a domain. Here, we label them a $\rightarrow$ d. These quantums are arranged in order of increasing complexity, where each quantum (e.g., c) presupposes all earlier ones (a,b). We then characterize each problem in a domain in terms of the minimum quantum level necessary to correctly solve it.

In problem solving scenarios, learners often bring to bear default assumptions that significantly affect their ability to solve a problem. To completely assess the challenge of a problem, it is necessary to make these

assumptions explicit, especially when they are not related to the problem domain but to other factors such as common sense/cultural background of the learner. One such assumption is that a learner will scan the solution space left to right, and top to bottom. We will indicate how this assumption affects our analysis of the challenge of problems.

Storage Placement

The Domain: An incoming job must be placed in memory. Memory will have jobs which arrived earlier, as well as holes where an incoming job may be placed. Storage placement deals with the task of selecting a hole in which to place an incoming job. We consider three algorithms: First fit (select the first accommodating hole), Best fit (select the largest accommodating hole) and Worst fit (select the smallest accommodating hole) (Silberschatz and Galvin 1994).
The **quantum units** in this domain are:

- find holes (ability 'a' to identify holes)
- find holes that fit (ability 'b' to size holes)
- find the correct hole that fits (ability 'c' to apply an algorithm)
- find the correct hole that fits, even when it is distinct from the correct hole according to a competing algorithm (ability 'd' to distinguish between algorithms)

Following is a taxonomy of problems in the domain of storage placement. We rate the "challenge" of a problem according to the level of ability required to solve it (a/b/c/d):

How many holes exist in memory?

- zero holes - required level: a, to identify that no holes exist in memory.
- one hole exists in memory - Is it large enough to hold an incoming job?
  - yes - required level: a. Learner does not have to know that the hole fits. The learner can just drop a job in the hole, and still get the benefit of the doubt!
  - no - required level: b. Learner must know that the hole does not fit in order *not* to schedule an arriving job in it!
- many holes exist in memory - How many holes are large enough to accommodate an arriving job?
  - zero/none - required level: b.
  - one hole - Is it the first hole among all the holes?
    - yes - required level: a. Once again, a learner can get the benefit of the doubt by just finding the first hole in memory, and dropping an arriving job into it without assessing whether the hole is large enough to fit the job.
    - no - required level: b. A learner must know that an arriving job does not fit in the first hole in order not to schedule the job in it.
  - many holes are large enough to accommodate an incoming job - Is the correct answer among these holes:
    - the first among all the holes? - required level: a.
    - the first among the fitting holes, but not the first among all the holes? - required level: b.
    - neither the first among all the holes nor the first among the fitting holes? - required level: c.
    - neither the first among all the holes nor the first among the fitting holes, nor even the correct hole according to a competing algorithm? - required level: d.

Our default assumption affects first fit placement algorithm: a learner's answer to a problem may be correct according to first fit placement even when the learner may not have understood first fit algorithm.

Our objective is to promote the generation of problems which require at least the abilities a and b. For first fit placement, a and b are the only required abilities. Ability b distinguishes apart first fit placement from the default behavior of a learner, which is to perform left-to-right, top-to-bottom scanning of the solution space. Abilities a, b and c are required for best fit and worst fit algorithms. Ability c distinguishes apart best and worst fit placements from first fit placement. Ability d distinguishes apart best fit from worst fit placement.

Processor Scheduling

The Domain: In a multiprogrammed machine, several jobs may be waiting for the processor. Processor scheduling is the task of selecting the next job for the processor to run. We consider three algorithms: First-in First-Out (select the first job), Shortest-Job-First (select the shortest job) and Highest Response Ratio Next (select the job with the best ratio of waiting time to service time) (Silberschatz and Galvin 1994).
The **quantum units** in this domain are:

- find the outstanding jobs (ability a to identify waiting jobs)
- find the correct job to be scheduled next (ability c to apply an algorithm)
- find the correct job to be scheduled next, even when it is distinct from the correct job according to a competing algorithm (ability d to distinguish between algorithms)

Note that ability b is missing: the processor can technically pick *any* job in the queue. We do not consider the issue of blocked processes, and job queue contains only those jobs that have already arrived.

Following is a taxonomy of problems in the domain of processor scheduling. We rate the "challenge" of a problem according to the level of ability required to solve it (a/c/d):

How many jobs await in the queue?

- zero jobs - required level: a, to identify that no job is in the queue.
- one job - required level: a, to identify that one job is in the queue.
- many jobs - Is the job that must be scheduled next:
  - the first job in the queue - required level: a. A learner can get the benefit of the doubt by just finding the first job in the queue, and scheduling it next without any knowledge of the scheduling algorithms.
  - a job other than the first one in the queue - required level: c. A learner must be able to apply the correct scheduling algorithm.
  - neither the first job in the queue nor the correct job according to a competing algorithm? - required level: d.

Our default assumption affects First-In-First-Out (FIFO) processor scheduling algorithm: a learner's answer to a problem may be correct according to FIFO even when the learner may not have understood the algorithm.

Our objective is to promote the generation of problems which require at least the abilities a and c. For FIFO scheduling, only ability a is required. Abilities a and c are required for Shortest-Job-First (SJF) and Highest Response Ratio Next (HRN) algorithms. Ability c distinguishes apart SJF and HRN from FIFO, while ability d distinguishes apart SJF from HRN scheduling.

Generalizing "Challenge"

We will now attempt to generalize the notion of "challenge" from the above two case studies. (In the future, we plan to include the problem domain of page replacement for this generalization.) Our generalization will apply to problem domains with the following characteristics, which are shared by both the domains above:

- The task involved is scheduling.
- The value-space of the domain is discrete, not continuous.
- The solution strategy is greedy: i.e., resources/partial steps are scheduled/committed as soon as possible, rather than preserved/conserved. In search parlance of Artificial Intelligence, these problems have absolute rather than relative solutions (Rich and Knight 1991).

We begin by distinguishing between answers and responses:

- an answer is a correct solution to a problem;
- a response is what a learner might consider to be (and hence produce as) an answer.

All answers are responses, whereas all responses need not be answers.

A preliminary generalization of our problem taxonomy is as follows:

	First Fit		Best Fit		Worst Fit	
Version	v4	v12	v4	v12	v4	v12
level b	50	126	29	69	40	26
level c	NA	NA	16	72	55	146

Table 1: Comparison of early and late versions of a Problem Generator for Storage Placement

How many responses does a problem have?

- zero - required level: b.
- one - required level: a. The solution is straightforward.
- many - Is the answer:
  - the first possible response? - required level: a. This discounts the default assumption that the learner scans the solution space top-to-bottom, left-to-right.
  - not the first possible response - required level: c.
    - not even the answer for any competing algorithm - required level: d.

Implementation

We implemented problem generators for storage placement and processor scheduling. Our goals were to:

- make the generators random and unpredictable. In storage placement, the generator uses random numbers to decide between arrivals and departures of jobs, the size of an arriving job (4% to 20% of total memory), and which job should depart. In processor scheduling, the generator uses random numbers to decide whether jobs arrive at a given time instant, and if so, how many jobs, as well as the length of each job (1 to 10 units of time).
- tune the generator to generate challenging problems at least 50% of the time. We used our measure of challenge in two ways:
  - We used it to measure the quality of the problems generated by the generator.
  - We incorporated it in the form of heuristics to tune the generator, so that the quality of the generated problems improved with successive versions of the generator: the problems were more evenly distributed over all the levels: a, b, c and d.

Table 1 indicates the improvement in the challenge of the generated problems from version 4 to version 12 of the generator for storage placement (Kumar 1997a, Kumar 1997b). The figures were obtained by running the problem generator through 500 combined arrivals and departures.

Similar figures for processor scheduling are shown in Table 2. The figures were obtained by running the problem generator for 500 units of time. We did not include FIFO algorithm in the table because it cannot be tested

	SJF		HRN	
Version	v2	v7	v2	v7
level c	9	23	17	42
level d	36	87	7	12
Jobs Scheduled	74	121	72	110

Table 2: Comparison of early and late versions of a Problem Generator for Processor Scheduling

beyond level a. The last row indicates the number of jobs that were scheduled in the 500 time units. This number, which is a count of the problems generated during each run, improved steadily from version 2 to version 7 due to heuristics we used to tune the problem generator. Level c in the table refers to problems that cannot also be categorized as level d, i.e., challenging problems for which, the answers from both SJF and HRN are the same.

Currently, we are experimentally validating our measure of challenge for problems in processor scheduling. We are polling advanced Computer Science students who have already taken the Operating Systems course as well as non-Computer Science majors who are taking the Computer Literacy course to try a set of problems, and rank them by difficulty.

Our work can be incorporated into an Intelligent Tutoring System in the following ways:

- **Metering:** Our approach can be used to discretize and quantify levels of difficulty in a problem domain. Therefore, an ITS can use it to meter and compare problems.
- **Testing:** Since the levels of challenge are arranged on an increasing scale (a $\rightarrow$ d), an ITS can vary the difficulty of generated problems along this scale in response to the learner's performance.
- **Feedback:** Since the levels of difficulty are based on qualitative analysis of the problem domain, an ITS can use the bases of these levels to hypothesize why a learner is erring on a certain problem (i.e., extrapolate the current mental model of the student), and generate a corrective course of action. E.g., if a student makes a mistake at level c in processor scheduling, the student does not understand the priority of the algorithm used. On the other hand, if the student makes mistakes at level a, the student does not understand the concept of processor scheduling.

Discussion

We proposed a measure of "challenge" for problems in a domain based on the knowledge/ability necessary to solve them. This measure may be used to assess the quality of dynamically generated problems, and automate the task of varying their level of challenge.

We examined two problem domains in Operating Systems and have included the results of incorporating our measure of challenge in problem generators for these domains. We have also tentatively generalized our characterization of "challenge" to a class of scheduling problems which uses discrete value-space.

Future work includes:

- analyzing page replacement problem domain to further generalize the problem taxonomy;
- using constraints to implement the generalized taxonomy of problems.

References

Kumar, A.N. 1997a. Generating challenging problems for first-fit storage placement algorithm. In Proceedings of the Florida Artificial Intelligence Research Symposium (FLAIRS '97) (Special Track on Intelligent Tutoring Systems), 57-61. Daytona Beach, FL.

Kumar, A.N. 1997b. Generating Challenging Problems in Intelligent Tutoring Systems: A Case Study of Storage Placement Algorithms. In Proceedings of the Eighth International PEG Conference, 128-134. Sozopol, Bulgaria.

Rich, E. and Knight K., 1991. *Artificial Intelligence.* New York,:McGraw Hill.

Silberschatz, A. and Galvin, P.B., 1994. *Operating System Concepts.* Reading, Mass:Addison Wesley.