Constraint-based Problem Generation for a Self-Assessment System

Nguyen-Thinh Le, Wolfgang Menzel
Department of Informatics, University of Hamburg, Germany

Abstract: In this paper, we describe a component of a self-assessment system which is capable to generate different exercise instances for different users. The system is used to give prospective students of Informatics a better feeling what kind of thinking skills this discipline requires. The system offers diversified problem tasks and by observing the performance of a student over a series of different problem instances we are able to assess the development of his/her problem solving abilities. We apply the constraint-based approach both for the problem generation and skill assessment. A first version of the system has been initially tested by thirteen high school students from Hamburg.

Keywords: Self-assessment, constraint-based, problem generation

Introduction

Increasing demand of using information technology in industry, economy and administration results in brilliant career chances for Computer Scientists. Many prospective students ask themselves whether they should decide for a study with a major in Informatics. However, not many of them have a clear idea, which kind of skills are required for this discipline. No doubt, many people associate this kind of study with the ability of using computers, programming or creating spreadsheets. We developed a component INCOM-Self for a web-based system which helps prospective students to assess themselves whether their qualifications meet the requirement for a study of Informatics, namely working with formal descriptions, dealing with possibly conflicting requirements, as well as comparing alternative solutions with respect to their degree of goodness.

Current surveyed self-assessment systems provide problem tasks for which usually a unique correct answer is expected and a solution has to be chosen from a list of alternatives. On the contrary, our self-assessment component poses questions which correspond more closely to real-life problems, namely problems with many possible solutions, or even no perfect solution at all. Then, the task of the student is to find an optimal solution for the given problem.

We apply a constraint-based approach, both to generate problems and to evaluate solutions submitted by students. A constraint represents a requirement of a problem task. Each constraint divides the solution space into two parts: one part of correct solutions and another part of incorrect ones. All constraints taken together, describe the boundary for the space of correct solutions. Problem tasks can be generated based on a set of specified constraints at an almost arbitrary number and the student can be encouraged to practice the same type of exercise over and over again. Therefore, the system not only allows us to evaluate the problem solving capabilities properly, but also a student’s perseverance and learning progress.

In the next section, we survey several self-assessment systems for prospective students of Informatics in Germany and review some techniques for problem generation. In the second section, we describe our self-assessment component and the constraint-based approach it applies. In the last section, we summarize the effectiveness of the constraint-based approach for the generation of self-assessment problems and describe our future work.

1. State of Art

1.1 Self-assessment systems for the study of Informatics

At present, there are three self-assessment systems which are used widely in German universities to convey prospective students an overall picture of Informatics. Technical University of Chemnitz [1] provides an off-line test with seven questions, each of them testing a specific skill: ability to reason, ability to abstract a problem and to model a problem, optimization problem, ability to capture a problem situation completely, comprehension of recursive representation, ability to specify an algorithm for a given problem. Freie Universität Berlin [2] offers a self-assessment system with multiple choice tests. Users have to choose possible correct answers for each question. After nineteen questions have been answered, users get a report with information about correct answers and achieved marks. The University of Munich [3] provides a more extensive self-assessment which contains exercises in logic, algorithmic thinking, ability to abstract, analytical thinking, Mathematics, English and German. The off-line test [1] and two other self-assessment systems have three characteristics in common:

1. For each problem task, a number of correct solutions is expected.
2. Every student will be given the same set of problems.
3. There is no feedback from the system immediately after a solution is submitted.

Our self-assessment component is capable to generate problem tasks randomly, for each problem there may be many correct solutions and users can learn from the system feedback in order to improve their solutions.

1.2 Technology of problem generation

In order to avoid students getting bored when doing a self-assessment task repeatedly, we are interested in techniques for problem generation in Intelligent Tutoring Systems. The authoring tool [4] facilitates automatic problem generation for a simple domain: the system merely selects random numbers from predefined ranges. Belmont and co-workers [5] use templates to generate problems such as true/false choice, or fill-in-the-blanks. Kumar's programming tutoring system also uses templates to generate problems of a more complex class: debugging problems, problems on predicting the output of programs and problems on evaluating expressions [6]. For each learning objective, there is a number of templates. Each template contains some meta-variables which are instantiated during problem generation. Systems that use some sort of templates for defining the structure of a problem may result in the same risk as systems with manually authored problems: if the problem (or template) set is too small, students might receive the same problem repeatedly. Animalwatch [7], a system for teaching mathematics via word problems, uses templates to generate new exercises, where the system simply instantiates different numbers to create a new problem. Although Animalwatch contains 6000 templates, students still complained of receiving the same problem twice but with different numbers. The system of Kojima and Miwa [8] generates word problems using problem generation episodes. Each problem generation episode is comprised of a base example problem and a new analogical instance which was generated from an example problem. With this system a small amount of problems can be expanded to a variety of problems. However, after generation the system requires users to correct generated problems adequately.

Martin [9] applied the constraint-based modeling technique to generate new problems that fit the student's current model, and concludes that this is superior to selecting one from a pre-defined list. The approach of problem generation proposed in [9] needs to construct a solution based on target constraints first and then converts it into natural language for presentation to the student. Martin admits that this approach has one major drawback: it requires that both the solution and problem generation algorithms be fail-safe. Unfortunately, testing the constraint set to ensure that only correct solutions will be generated might be not possible. In problem generation, generating plausible natural language queries that do not make the solution obvious is also difficult. Moreover, in solution generation, the most common problem is that the algorithm fails to terminate.

We also want to adopt the constraint-based modeling technique to generate problems, but without the necessity to create a solution in advance.

2. Our Constraint-based Self-Assessment System

2.1 Characteristics of assessment exercises

The study of Informatics requires the ability of analyzing and solving problems in daily life scenarios. Typical characteristics of such problems are: 1) For each problem task, several solutions might exist; 2) The requirements of a problem task might be conflicting and sometimes, no perfect solution can be expected at all, but they can be distinguished by their degree of goodness. Currently, our system provides four problem tasks of this class (Appendix A). The following exercise is used to illustrate how our system works.

<u>Family party:</u> *You should arrange family members around a table so that a family party can take place harmoniously. Please, choose places for members of this family so that the following conditions can be satisfied (if possible) where the importance of each condition is listed in descending order:*

1. *It is indispensable to separate people who are at odds with each other;*
2. *It is important that people with contrary interests should not sit next to each other;*
3. *It is desirable that immediate neighbors should be of different gender.*

Name	Age	Gender	Interests	Aversions	At odds with
Marie	42	F	health, children	politics, stock market	Uncle Karl, Lieschen
Anna	10	F	games, prominent	war, religion	
Uncle Karl	61	M	choir, religion	health, politics	Marie, Uncle Westerwelle
...	...	...	...	...	...

Our goal is that the system is capable to generate problem tasks dynamically so that each student can work with different variants of the same problem type. For this purpose, we apply the constraint-based modeling technique.

2.2 The constraint-based approach

Each problem task contains several requirements which can be modeled as constraints. A constraint consists of a relevance and a satisfaction condition [10]. The former one specifies which problem-solving state is relevant, and the latter one describes the state which fulfills the task requirements. For example, the first requirement of the problem task "*Family party*" can be expressed as follows:

Relevance: IF A and B are immediate neighbors

Satisfaction: THEN A and B should not be at odds with each other

A constraint is violated if an appropriate problem-solving state is relevant and the satisfaction part of the constraint is evaluated to false. We enrich the constraint formula above with an explanation and a penalty which is incurred in case that constraint is violated. The explanation can be used to explain why a problem-solving state does not satisfy a requirement. The penalty value expresses the severity of that requirement. For the requirements in the problem task "*Family party*" we specify three constraints as follows:

Constraint 1:
Relevance: IF A and B are immediate neighbors
Satisfaction: THEN A and B should not be at odds with each other
Explanation: A is at odds with B, better avoid to place them beside each other.
Penalty: 15

Constraint 2:
Relevance: IF A and B are immediate neighbors
Satisfaction: THEN A and B should not have contrary interests
Explanation: A and B have conflicting interests, better avoid to place them together.
Penalty: 10

Constraint 3:
Relevance: IF A and B are immediate neighbors
Satisfaction: THEN A and B should not be of the same gender
Explanation: A and B are of the same gender, better do not place them beside each other.
Penalty: 5

Constraint 1, 2 and 3 have different penalty values because the first requirement is indispensable and the second one is more important than the third one. We refer to Constraint 1 as a hard constraint and Constraint 2 and 3 as soft constraints.

A constraint divides a space of all solutions into two parts: the first one contains positive solutions which satisfy that constraint, and the other one contains negative solutions. For a hard constraint the negative ones are unacceptable whereas the negative solutions for a soft constraint may still be acceptable but to a lesser degree than the positive ones. The space of perfect solutions is the intersection of positive solutions for all constraints, i.e. no constraint is violated (see Figure 1). Note, that the set of perfect solutions can be empty, e.g. Constraint 3 above can never be fulfilled for an odd number of participants. Even under such circumstances, the system can determine the optimal solution and provides a critical assessment as long as there is still room for improvement.

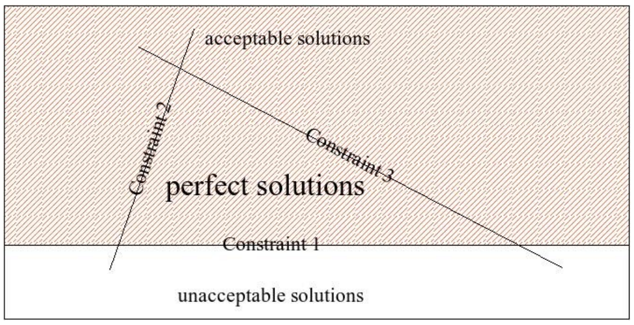


Figure 1: Constraints delimit the space of perfect solutions

2.2 Solution evaluation

Whenever a solution for the problem task "*Family party*" is submitted for evaluation, constraints 1-3 will be examined to determine whether the solution fulfills the task requirements. To examine each constraint, the part of the solution, which is relevant to the constraint, is selected, for example, to examine Constraint 1 we select from the submitted solution "two family members A and B who are immediate neighbors". After that, the satisfaction part of the constraint is examined. If it evaluates to false, i.e. the constraint is violated, its corresponding explanation is returned to the user.

2.3 Problem generation

A problem task consists of a problem text and problem data. The problem text describes the task and problem data will be used to solve the task. To generate an instance of the problem "*Family party*", we need to generate problem data according to the following procedure:

1. Setup a database of persons from which a family can be specified. Each person record has the following format: *person(Name, Age, Gender, Interest, Aversion, People at odds).*
2. Select a required number of different records from the database of persons. (Currently, for this problem type, we specify a family size of seven persons.)
3. Find the optimal solution for the generated problem.
4. In case no solution can be determined, which satisfies hard constraints, (in our case, Constraint 1), go to step 2.

As Martin mentions in [9] that his constraint-based problem generation algorithm has a termination problem, two questions arise immediately: 1) Will our algorithm above always generate a problem task? 2) Can our problem generation approach work for problem domains with larger constraint sets? If all seven family members are odds at each other, then we cannot place them to a table so that the first requirement of the problem task can be satisfied. Since we implement our system in Prolog whose computation is based on backtracking, the algorithm above terminates if a correct solution can be found or after all records of the database of persons have been tried unsuccessfully. In the latter case, the computation returns a false value which means no solution can be found which satisfies all "hard" requirements of the problem task. For this reason, it is the responsibility of the exercise author to validate whether his data are consistent and reasonable, that means at least a problem task does exist. With respect to the number of constraints the second question is also not dramatic. The procedure of problem generation can be expressed as a `generate-and-test` procedure where the `test` sub-procedure can be defined as a sequence of checking a set of constraints, irrespective of the size of the constraint set.

The only problem of our algorithm lies mainly at the second step: select records from a database. For some problems, problem data are not specified in a database by an exercise author, but can be generated randomly. Thus, we need to embed restrictions into this process. For instance, to generate a task for the scheduling problem (see Appendix A, problem task 4) we need to create a list of tasks t1, t2,... and their precedence relations. The generation of precedence relations is not trivial because we have to avoid cycling relations. This restriction increases the complexity and thus, slows down the process of problem generation. For example, to create a precedence relationship for N tasks, we will have S = 1+2+...+(N-1)= N*(N-1)/2 possible relations. In our example, for each possible precedence relation, we need a restriction check, if we had X restrictions which have to be considered within the data generation process and between two elements t1, t2 there are Y types of relationships, then S*X*Y restriction checks have to be executed. This is resource intensive.

Our problem generation technique can be extended to generate problems of different size (i.e. the number of family members for the "*Family party*" problem), and a different degree of difficulty (i.e. the number of possible solutions). In the first version of our INCOM-Self system, however, this feature has not yet been implemented.

2.4 Feedback generation and test result

When a user submits an incorrect solution, the system returns an appropriate explanation and also she/he is offered the possibility to improve her/his solution. Thereby, we can evaluate the perseverance of the student. For each solution, the student receives a corresponding score. From the score development, we reason about the ability to learn from system feedback in addition to his/her ability to solve the problem. Even if the student has solved the task correctly, he is offered the possibility to repeat the same exercise type with a new problem instance (Figure 2). This enables us to evaluate the skills and personal characteristics of a user according to several orthogonal dimensions:

1. Problem solving ability as the average score obtained in a series of solution attempts.
2. Perseverance as a combination of several factors: time needed to solve the task, continuity, number of solution attempts, and maximal score achieved.
3. Ability of learning from feedback as the slope of scores over a sequence of attempts.
4. Improvement of problem solving ability.
5. Systematicity of problem solution to distinguish between random guessing and systematic improvement.

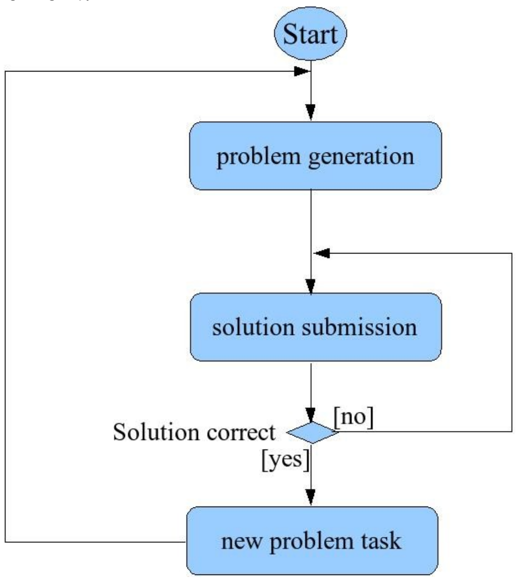


Figure 2: The problem generation cycle

3. Conclusion and Future Work

We applied the constraint-based modeling technique to develop a self-assessment component capable of generating problem tasks automatically. The approach is particularly suited for problem tasks which have the following features:

- Solutions for each problem task are sufficiently rich in information, because only then the constraint-based diagnosis can derive helpful feedback information [12]. Therefore, we have favored ordering problems over simple numerical computations.
- The solution space for each problem task is characterized by static conditions. For example, the solution space for the problem "*Family party*" is defined by three weighted constraints which represent task requirements of different importance (Figure 1). Our constraint-based technique does not examine each step of the student's input as the model-tracing technique which is applied in cognitive tutors would do [11].
- Furthermore, our system hosts only problem tasks for which intuitively obvious user

interfaces can be created so that the student does not need any additional training using the system before.

Our self-assessment component differs from others by following features:

- The range of problem tasks is not only limited to the one which have an unique solution, but rather include problems with multiple correct solutions. The system provides problem tasks which require to compare and evaluate solutions.
- The system is able to generate problem tasks dynamically so that each user can work with different instances of the same problem type. This allows us to observe the student's performance over a series of attempts for similar problems.
- The system returns feedback to possible deficiencies of a submitted solution and the user can improve her/his solution. Thus, the ability of problem solving, the perseverance and the ability of learning from feedback can be observed.

We have started the evaluation of our self-assessment component with thirteen students of the high school Gymnasium Allee in Hamburg-Altona. All of them are students of the thirteenth class and have Informatics as a major course. One of them is female. At a first glance, we noticed that most of the participants were able to use the system feedback for improving their solutions successively. We intend to conduct an empirical analysis with a larger amount of high school students.

Acknowledgments

This research was funded by the E-Learning Consortium Hamburg, grant number A0035.

References

[1] http://www.tu-chemnitz.de/fsrif/selbsttest01 *Freiwilliger Selbsttest Informatik* (Unsolicited self-assessment for the study of Informatics), Technical University of Cheminitz, Germany.

[2] http://www.inf.fu-berlin.de/inst/ag-tech/eignungstest.html *Selbsttest für alle an der Informatik Interessierte* (Self-assessment for those who are interested in the study of Informatics), Freie Universität Berlin, Germany.

[3] http://www.pms.ifi.lmu.de/eignungstest *Selbsttest zur Prüfung der Eignung zum Studium der Informatik* (Self-assessment the eligibility of the study of Informatics), Ludwig-Maximilians-University of Munich.

[4] Blessing (1997). *A Programming by demonstration authoring tool for model-tracing tutors*. International Journal of Artificial Intelligence in Education 8, pp. 233-261.

[5] Belmont, M.V. et al. (2002). *Automatic generation of problems in web-based tutors.* In Jain, L.C., Howlettt, R.J., Ichalkaranje, N.S. and Tonfoni, G. (Eds.), *Virtual Environments for Teaching & Learning*, World Scienctific.

[6] Kumar, Amruth (2005). *Rule-based adaptive problem generation in programming tutors and its evaluation.* In *Proceedings of Workshop on Adaptive Systems for Web-Based Education: Tools and Reusability*, 12th International Conference on Artificial Intelligence in Education, Amstedam.

[7] Arroyo, I., Beck, J., Beal, C. and Woolf, B.P. (2000). *Macro adapting Animalwatch to gender and cognitive differences with respect to hint interactivity and symbolism.* In Gautier, G., Frasson, C. and VanLehn, K. (Eds.), *Proceedings of the 5th International Conference on Intelligent Tutoring Systems*, Montreal, Springer, pp. 574-583.

[8] Kojima, K. & Miwa, K. (2005) *A system that generates word problems using problem generation episodes.* In C.-K. Looi et al, *Towards sustainable and scalable educational innovations informed by the learning sciences*. IOS Press, pp. 195-202.

[9] Martin, B. (2001). *Intelligent Tutoring Systems: The practical Implementation of Constraint-based Modelling.* PhD thesis, University of Canterbury.

[10] Ohlsson, S. (1994). *Constraint-based student modelling.* In J. E. Greer, G.I. McCalla, *Student Modelling: The Key to Indivi-dualized Knowledge-based Instruction,* 167-189. Berlin.

[11] Anderson, J.R. & Corbett, A.T. (1995). *Cognitive tutors: Lessons learned.* In The Journal of The Learning Sciences, 4(2), 167-207.

[12] Kondaganallur, V., Weitz, R., & Rosenthal, D. (2005) *A comparison of model-tracing and constraint-based intelligent tutoring paradigms.* International Journal of AI in Education, 15(2), 117-144.

Appendix A: Sample problem tasks

Problem task 1: card sets sorting

You have a card deck of 52 cards. Please sort the following card sets in ascending order of their cardinality:
Set A contains cards which are hearts and not figures.
Set B contains cards which are red and queens.
Set C contains cards which are not spades and not small numbers.
Set D contains cards which are diamonds and numbers.
Set E contains cards which are not red but big numbers.

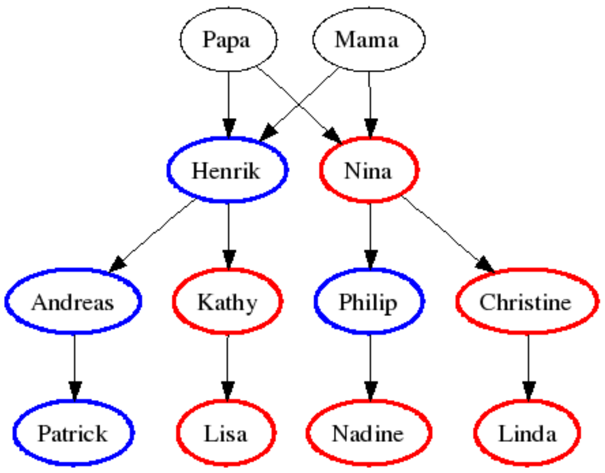
Please, consider the following definition:
Colors: red, black.
Figures: jack, queen, king.
Small numbers: two, three, four, five, six.
Big numbers: seven, eight, nine, ten.
Even numbers: two, four, six, eight, ten.
Odd numbers: three, five, seven, nine.

Problem task 2: family festival (see section 2.1)

Problem task 3: marriage allowance

There is a family living alone on an isolated island. Please, determine from the family tree, which pairs in this family are allowed to get married according to the following rules:

1. Homosexual pairs are not allowed in this family.
2. Cousins who are related within two generations are not allowed.
3. Only pairs of the same generation can get married.



Problem task 4: Scheduling

“You are given a collection of tasks t1, t2,... with their execution times D1, D2,... respectively. The tasks are to be executed on a set of three identical processors. Any task can be executed on many processors, but each processor can only execute one task at a time. There is a precedence relation between tasks which tells what tasks, if any, have to be completed before some other task can be started. The scheduling problem is to assign tasks to processors so that the precedence relation is not violated and the all the tasks together are processed in the shortest possible time. The time that the last task in a schedule is completed is called the finishing time of the schedule. We want to minimize the finishing time over all permissible schedules.”