Constraint Authoring System: An Empirical Evaluation

Pramuditha SURAWEERA, Antonija MITROVIC and Brent MARTIN

Intelligent Computer Tutoring Group
Department of Computer Science, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
{pramudi, tanja, brent}@cosc.canterbury.ac.nz

Abstract. Evaluation is an integral part of research that provides a true measure of effectiveness. This paper presents a study conducted to evaluate the effectiveness of CAS, a knowledge acquisition authoring system developed for generating the domain knowledge required for constraint-based tutoring systems with the assistance of a domain expert. The study involved a group of novice ITS authors composing domain models for adding two fractions. The results of the study showed that CAS was capable of generating highly accurate knowledge bases with considerably less effort than the effort required in manual composition.

Introduction

Composing the domain knowledge required for Intelligent Tutoring Systems (ITS) consumes the majority of the total development time [6]. The task requires a multi-faceted set of expertise, including knowledge engineering, AI programming and the domain itself. Our goal is to widen the knowledge acquisition bottleneck by empowering domain experts with little or no programming and knowledge engineering expertise to produce domain models necessary for ITSs.

Researchers have been exploring ways of automating the knowledge acquisition process since the inception of ITSs. Disciple, developed by Tecuci and co-workers[10], is an example of a learning agent shell for developing intelligent educational agents. A domain expert teaches the agent to perform domain-specific tasks by providing examples and explanations (similar to an expert teaching an apprentice), and Disciple uses machine learning techniques to infer the necessary domain knowledge. The Cognitive Tutor Authoring Tools (CTAT) [1] assist in the creation and delivery of model-tracing ITSs. The main goal of these tools is to reduce the amount of artificial intelligence (AI) programming expertise required. CTAT allows authors to create two types of tutors: 'Cognitive tutors' and 'Pseudo tutors'. 'Cognitive tutors' contain a cognitive model that simulates the student's thinking to monitor and provide pedagogical assistance during problem solving. In contrast, 'Pseudo tutors' do not contain a cognitive model: to develop a tutor of this kind, the author needs to record possible student actions and provide corresponding feedback messages.

We have developed an authoring system, named Constraint Authoring System (CAS) that generates a domain model with the assistance of a domain expert. The author is required to describe a domain in terms of an ontology and provide problems

and their solutions. CAS analyses the provided information using machine learning techniques to generate a domain model.

This paper outlines a study conducted with a group of novice authors to evaluate the effectiveness of CAS. They were given the task of composing a domain model for a fractions addition ITS using CAS. The results showed that CAS was capable of generating highly accurate constraint bases even with the assistance of novices. It also showed that CAS reduced the overall effort required to produce domain models.

The remainder of the paper is organised into three sections. The next section gives a brief introduction to CAS. The results and analysis of the experiment are given in section three. The final section presents conclusions and future work.

1. Constraint Authoring System

Constraint Acquisition System is an authoring system that generates the domain model required for constraint-based tutoring systems [5] with the assistance of a domain expert/ teacher. The goal of the system is to significantly reduce the time and effort required for composing constraint bases. We envisage that CAS would enable domain experts with minimal expertise in constraint-based modelling to produce new tutoring systems. The user is only required to model the domain in terms of an ontology and provide example problems and their solutions. Once the required domain-dependant information is provided, CAS generates constraints by analysing the provided information. Although the system is designed to support authors with minimal knowledge engineering expertise, it also offers utilities for experts in composing constraint bases. The system allows experts to modify constraints during the stage of validating the system-generated constraints. Users are also provided with editors for directly adding new constraints to the domain model.

A detailed discussion of CAS is beyond the scope of this paper and has been presented in [7]. Here we only give a short overview of its features. CAS is developed as an extension of WETAS [3], a web-based tutoring shell that facilitates building constraint-based tutors. WETAS provides all the domain-independent components for a text based ITS, including the user interface, pedagogical module and student modeller. It does not provide support for authoring domain models; however, authoring tools may be added to WETAS to provide this need. CAS was used in this manner.

Authoring knowledge using CAS is a semi-automated process with the assistance of a domain expert. The author carries out a number of tasks, including modelling the domain as an ontology, specifying the general structure of solutions and providing example problems and solutions. The constraint generators of CAS use this information to generate both syntax and semantic constraints. The syntax constraints are generated by analysing the ontology and translating each specified restriction into a constraint [9]. The semantic constraints generator uses machine learning techniques to generate constraints by analysing the problems and solutions provided by the domain expert [7]. It generates constraints by comparing and contrasting two alternative solutions to the same problem. Constraints are generated iteratively, and they are generalised or specialised during subsequent analysis of other solutions.

The interface of CAS consists of two main tabs: the “ontology view” and the “domain model editor”. The “ontology view” contains the ontology workspace and other tools necessary for composing the domain-dependant components necessary for each step of the authoring process. Figure 1 illustrates the ontology for the fraction

addition domain, and the relationships defined for the *Improper Fraction* concept. The "ontology view" also contains an interface for adding problems and their solutions that are used for constraint generation. The "domain model editor" tab consists of a set of textual editors for viewing and modifying constraints generated by CAS. Constraints in these editors can be directly loaded into WETAS for testing in an ITS environment. The "domain model editor" tab also contains an editor for modifying and composing problems and solutions in the Lisp representation expected by WETAS.

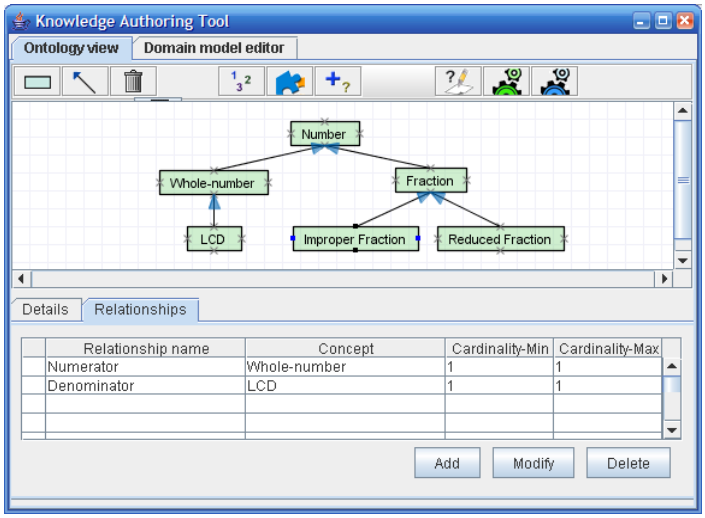

Figure 1: CAS Interface, illustrating the ontology workspace

2. Evaluation Study

In previous work we evaluated CAS by comparing its generated domain models to the domain models developed manually [7, 9]. The domain models were generated with the assistance of expert authors. However, the goal of CAS is to support novice authors to develop ITSs. For that reason, we performed a study with a group of 13 novice authors to evaluate the effectiveness of the system as a whole. The participants were students enrolled in a 2006 graduate course about ITSs at the University of Canterbury. They were assigned the task of producing a complete ITS in WETAS for the domain of adding fractions, using CAS to author the domain model. The goal of the evaluation study was to validate three hypotheses: CAS makes it possible for novices to produce complete constraint bases; the process of authoring constraints using CAS requires less effort than composing constraints manually; the constraint generation algorithm depends on the ontology (i.e. an incomplete ontology will result in an incomplete constraint set).

The participants were required to compose all the domain-dependant components necessary for CAS to generate constraints, including a domain ontology, problems and solutions. After composing the required components, CAS can be used to generate syntax and semantic constraints in a high-level language (pseudo-code). At the time of the study, CAS was not able to convert these into the WETAS constraint language, so the participants were also required to perform this step manually. They were also free to modify/delete constraints.

The participants had attended 13 lectures on ITSs, including five on constraint-based modelling before the study. They were introduced to CAS and WETAS, and were given a task description document that outlined the process of authoring a domain model using CAS and described the WETAS constraint language. The participants were encouraged to follow the suggested authoring process. In addition to the task description, participants were also given access to all the domain model components of LBITS [2], a tutoring system for English language skills. They were also provided with an ontology for database modelling as an example. Finally, the students were instructed to use a particular interface style (one free-form text box per fraction) when building their tutor. The participants were allocated a period of six weeks to complete the task, however, most students only started working on it at the end of week 3.

Twelve out of the 13 participants completed the task satisfactorily, i.e. produced working tutoring systems. One participant failed to complete the final step of converting the pseudo-code constraints to the target constraint language. We only present results of the twelve participants in the rest of the paper.

Analysis of CAS's logs revealed that the participants spent a total of 31.3 (13.4) hours on average interacting with the system. Six and a half hours (4.34) of that was spent interacting with the "ontology view". The majority of the time in the "ontology view" was spent developing ontologies. There was a very high variance in the total interaction time, which can be attributed to each individual's ability.

The participants used the textual editors that were available under the "domain model editor" tab to modify/add domain model components required for WETAS, including problems and their solutions, syntax and semantic constraints and macros. The participants spent a mean total of 24.7 (9.6) hours interacting with the textual editors.

	Constraints		*Coverage*		*Completeness*	
	Syntax	Semantic	Syntax (8)	Semantic (13)	Syntax	Semantic
S1	5	12	5	7	63%	54%
S2	5	13	5	12	63%	92%
S3	4	12	4	12	50%	92%
S4	16	16	5	12	63%	92%
S5	14	18	8	13	100%	100%
S6	15	11	5	12	63%	92%
S7	2	5	3	3	38%	23%
S8	8	13	7	4	88%	31%
S9	5	8	4	4	50%	31%
S10	7	11	5	12	63%	92%
S11	4	18	5	12	63%	92%
S12	9	16	6	1	75%	8%
Mean	7.83	12.75	5.17	8.67	64.58%	66.67%
S.D.	4.73	3.89	1.34	4.5	16.71%	34.61%

Table 1: Total Numbers of Constraints Composed by Participants

In order to evaluate the completeness of participants' constraint bases, we manually compiled an "ideal" set of constraints for the domain, containing eight syntax constraints and 13 semantic constraints. The syntax constraints ensure that each component of a solution (such as the LCD and converted fractions) is in the correct format, and the correct problem-solving procedure is followed. The semantic constraints ensure that each component of a solution is correctly defined.

Table 1 lists the total numbers of constraints (syntax and semantic) composed by the participants under the "Constraints" column. The total numbers of "ideal"

constraints covered by each constraint base are listed under the “Coverage” column. The completeness of each constraint base, calculated as the percentage of constraints accounted for by each constraint base, is given under the “Completeness” column. The participants accounted for five syntax constraints in the “ideal” set (65%) and nine semantic constraints (66%). Only one participant (S5) produced all the necessary constraints. The majority of the others had accounted for over half of the desired constraints. One participant (S12) struggled with composing semantic constraints and only managed to account for one desired constraint.

The “ideal” set of constraints contains five syntax constraints for verifying the syntactic validity of inputs, such as whether the LCD is an integer and whether the entered fractions are syntactically valid. They need to be verified due to the generic nature of the interface the students were told to use, which consisted of a single input box to input a fraction. For example, constraints are required to verify that fractions are of the ``numerator / denominator'' form. However, these constraints are redundant for the solution-composition interface produced by CAS from a complete ontology. It contains two text boxes for inputting a fraction (one for its numerator and the other for its denominator), ensuring that a fraction is of the correct format. Furthermore, these input boxes only accept values of the type specified for the relevant property in the ontology (numerator and denominator would both be defined as integer in this case). As a consequence, constraints such as the ones to verify whether the specified numerator is an integer are also redundant.

	Constraints		Coverage		Completeness	
	Syntax	Semantic	Syntax (3)	Semantic (13)	Syntax	Semantic
S1	7	15	3	12	100%	92%
S2	8	12	3	12	100%	92%
S3	18	0	3	0	100%	
S4	6	23	3	1	100%	8%
S5	9	8	3	12	100%	92%
S6	0	23	0	1		8%
S7	13	26	3	1	100%	8%
S8	6	0	3	0	100%	
S9	11	23	3	1	100%	8%
S10	9	12	3	12	100%	92%
S11	7	12	3	12	100%	92%
S12	17	42	2	1	67%	8%
Mean	9.25	16.33	2.67	5.42	97.00%	49.97%
S.D.	4.97	11.86	0.89	5.82	9.95%	44.30%

Table 2: Total Numbers of Constraints Generated by CAS

The participants were free to make any modifications to the initial set of constraints produced by CAS, including deleting all of them and composing a new set manually. For that reason, we also analysed the constraints produced by CAS automatically, from the domain information supplied by the author. The generated constraint sets were analysed to calculate their completeness (Table 2). CAS generated the three syntax constraints necessary for CAS’s solution interface for 10 participants. There was one situation where just two required constraints were generated (S12), as the result of an incorrectly specified solution structure. The syntax constraints generator failed produce any constraints for participant S6 due to a bug.

CAS only has the ability to generate 12 out of the 13 required semantic constraints, as it is unable to generate constraints that require algebraic functionality. CAS cannot generate a constraint that accounts for common multiples of the two denominators

larger than the lowest common multiple. So the maximum degree of completeness that can be expected is 92%.

CAS generated the maximum possible 12 semantic constraints from domain-dependant components supplied by five participants. However, it was not successful in generating constraints for the remaining participants. Further analysis revealed that there were two main reasons. One of the reasons was that two of the participants (S4 and S9) had unknowingly added empty duplicate solutions for problems, which resulted in constraints that allowed empty solutions. This situation can be avoided easily by restricting the solution interface not to save empty solutions.

Another common cause for not generating useful semantic constraints was an incomplete ontology. Four participants (S4, S6, S7 and S12) modelled the *Fraction* concept with only a single property of type *String*. This results in a set of constraints that compare each component of the student solution against the respective ideal solution component as a whole. These constraints are not of the correct level of granularity. Consequently, the resulting feedback is limited in pedagogical significance. For example, the constraints would have the ability to denote that the student has made an error in the sum, but not able to pinpoint whether the mistake is in the numerator or the denominator. We believe that the decision to model the *Fraction* concept with a single property may have been influenced by the student interface that we required them to use. The participants may have attempted to produce an ontology and solution structure that is consistent with this particular student interface.

The constraint generator failed to produce any semantic constraints for two participants, S3 and S8. It failed to generate constraints for S3 due to a bug in the system. The other participant (S8) did not add any solutions, and therefore semantic constraints could not be generated.

a. *Relevance:* Fraction-1 component of IS has a (?var1, ?*) 'Improper fraction'
   Satisfaction: Fraction-2 component of SS has a (?var1, ?*) 'Improper fraction'

b. *Relevance:* (match IS Fraction-1 ("2." ?var1 ?IS-var2 ?*))
   Satisfaction: (match SS Fraction-1 ("2." ?var1 ?SS-var2 ?*))

Figure 2: An example of translating a pseudo-code constraint into WETAS language

Although we assumed that the generated high-level constraints assisted the participants, there was little evidence in their reports that supported this assumption. Only one participant indicated that the generated constraints assisted him. Since no explanation on the high-level constraint representation was provided to the participants, they may have struggled to understand the notation and to find commonalities between the two representations. For example, Figure 2a shows the pseudo-code representation of a constraint that ensures that the numerator of first fraction specified by the student (SS) is the same as the corresponding value in the ideal solution (IS). The equivalent constraint in the WETAS language is given in Figure 2b.

3. Discussion

Analysing the results from the evaluation study confirmed all our hypotheses. Our first hypothesis about CAS being effective has been confirmed in previous work [7, 9]. The 2006 study revealed that CAS was able to generate all the required syntax constraints for 10 of the 12 participants. Furthermore, CAS generated over 90% of the semantic constraints for half of the participants. Considering that the participants were given

very little training in using the authoring system, the results are very encouraging. Providing the users with more training and improving CAS to be fully integrated with a tutoring server (similar to WETAS) would further increase its effectiveness.

The second hypothesis claims that CAS requires less effort than composing constraints manually. In order to obtain a measure for the effort required for producing constraints using CAS, we calculated the average time required for producing a single constraint. Only the participants whose domain model components resulted in generating near complete constraint bases were used for calculating the average effort, to ensure that incorrectly generated constraints were not accounted. Five participants (S1, S2, S5, S10, S11) spent a total of 24.8 hours composing the required domain-dependant information. They also spent a total of 115.04 hours interacting with the textual editors to produce a total of 107 constraints. Consequently, the participants spent a total of 1.3 hours on average to produce one constraint.

The average time of 1.3 hours per constraint is very close to the 1.1 hours per constraint reported by Mitrovic [4] for composing constraints for SQL-Tutor. The time estimated by Mitrovic can be considered as biased since she is an expert of SQL and knowledge engineering, in addition to being an expert in composing constraints. Therefore, the achievement by novice ITS authors producing constraints in a time similar to the time reported by Mitrovic is significant. Furthermore, the time of 1.3 hours is a significant improvement from the two hours required by a similar study reported in [8]. Although that study involved composing constraints for the domain of adjectives in the English language, the overall complexities of the tasks are similar. Further, after the experiment was completed, it was discovered that the setup of WETAS was not optimal, which led the participants to perform additional effort when writing the final constraints. The figure of 1.3 hours per constraint is therefore probably pessimistic.

Although CAS currently generates constraints in a high-level language, it can be extended to generate constraints directly in the language required for execution. Assuming the generated constraints were produced in the required runnable form, the total of 99 syntax and semantic constraints were produced from 24.8 hours spent on composing the required domain information. Consequently, the participants would only require an average of 15 minutes (0.25 hours) to generate one constraint.

An average of 15 minutes to produce a constraint is a significant improvement from 1.1 hours reported by Mitrovic [4]. The time is more significant as the authoring process was driven by novice ITS authors. However, this does not take into account validating the generated constraints. As the constraint generation algorithm may not produce all the required constraints, the domain author may also be required to modify the generated constraints or add new constraints manually.

Finally, the third hypothesis concentrates on the sensitivity of the constraint generation on the quality of the ontology. Developing a domain ontology is a design task, which depends on the creator's perceptions of the domain. The ontologies developed by two users, especially if the domain is complicated, are very likely to be different. The constraint learning algorithm generated constraints using ontologies developed by the 12 participants. Although the ontologies were different, the syntax constraint generation algorithm managed to produce full constraint sets for almost all participants. However, the semantic constraint generation was more sensitive to the ontology. In particular, it was reliant on defining the *Fraction* concept with enough details. The semantic constraint generator managed to produce 92% complete constraint sets of the correct granularity for ontologies with a correctly defined

Fraction concept. On the contrary, the constraints generated for ontologies with a partially defined *Fraction* concept were too general. They compared each fraction composed by students as a whole against its corresponding fraction in the ideal solution. These constraints result in feedback limited in pedagogical significance.

4. Conclusions

This paper reports on an evaluation study of CAS, an authoring system for constraint-based tutors. The evaluation study conducted with novice authors produced very encouraging results. The syntax constraint generator was extremely effective, producing complete constraint sets for almost every participant. The semantic constraints generator produced constraint sets that were over 90% accurate for half of the participants. Although the constraints produced by the generator for the remaining participants were too general, ITS authors can modify them with little effort to produce constraints of correct granularity. We believe that this would contribute towards the reduction of the author's total workload.

At the time of the evaluation, constraints produced by CAS were not runnable, but the system can be easily extended to produce constraints in the target language. This would dramatically reduce the effort required for composing constraints. The evaluation revealed that the total workload required by a novice to generate a single constraint using CAS would be even less than the time required by experts to write them by hand.

We intend to enhance CAS further to generate constraints directly in the runnable form. CAS should also be fully integrated with WETAS and made consistent with its internal representation. We also plan to conduct further evaluations on the effectiveness of CAS's constraint generation.

References

1. Aleven, V., McLaren, B., Sewall, J. and Koedinger, K., The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. in *ITS 2006*, (Taiwan, 2006), Springer-Verlag, 61-70.
2. Martin, B. and Mitrovic, A., Domain Modeling: Art or Science? in *AIED 2003*, (Sydney, Australia, 2003), IOS Press, 183-190.
3. Martin, B. and Mitrovic, A., WETAS: a Web-Based Authoring System for Constraint-Based ITS. in *AH 2002*, (Malaga, Spain, 2002), LCNS, 543-546.
4. Mitrovic, A., Koedinger, K. and Martin, B., A comparative analysis of cognitive tutoring and constraint-based modeling. in *UM 2003*, (Pittsburgh, USA, 2003), Springer-Verlag, 313-322.
5. Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B., Constraint-based Tutors: a Success Story. in *14th Int. Conf on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, (Budapest, 2001), Springer-Verlag, 931-940.
6. Murray, T. Expanding the Knowledge Acquisition Bottleneck for Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education*, *8*. 222-232.
7. Suraweera, P., Mitrovic, A. and Martin, B., A Knowledge Acquisition System for Constraint-based Intelligent Tutoring Systems. in *AIED 2005*, (Amsterdam, Netherlands, 2005), IOS Press, 638-645.
8. Suraweera, P., Mitrovic, A. and Martin, B., The role of domain ontology in knowledge acquisition for ITSs. in *ITS 2004*, (Maceio, Brazil, 2004), Springer, 207-216.
9. Suraweera, P., Mitrovic, A. and Martin, B., The use of ontologies in ITS domain knowledge authoring. in *SWEL '04, ITS 2004*, (Maceio, Brazil, 2004), 41-49.
10. Tecuci, G. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. Academic press, 1998.