Adding Weights to Constraints in Intelligent Tutoring Systems: Does It Improve the Error Diagnosis?

Nguyen-Thinh Le and Niels Pinkwart

Clausthal University of Technology,
Germany
{nguyen-thinh.le,niels.pinkwart}@tu-clausthal.de

Abstract. The constraint-based modeling (CBM) approach for developing intelligent tutoring systems has shown useful in several domains. However, when applying this approach to an exploratory environment where students are allowed to explore a large solution space for problems to be solved, this approach encounters its limitation: It is not well suited to determine the solution variant the student intended. As a consequence, system's corrective feedback might be not in accordance with the student's intention. To address this problem, this paper proposes to adopt a soft computing approach for solving constraint satisfaction problems. The goal of this paper is two-fold. First, we will show that classical CBM is not well-suited for building a tutoring system for tasks which have a large solution space. Second, we introduce a weighted constraint-based model for intelligent tutoring systems. An evaluation study shows that a coaching system for logic programming based on the weighted constraint-based model is able to determine the student's intention correctly in 90.3% of 221 student solutions, while a corresponding tutoring system using classical CBM can only hypothesize the student's intention correctly in 35.5% of the same corpus.

Keywords: intelligent tutoring systems, constraint satisfaction problems, weighted constraint-based model, cognitive diagnosis, evaluation.

1 Introduction

An intelligent tutoring system (ITS) is used to support students individually to solve problems. Research on ITSs has made considerable progress and some systems have been integrated into regular classrooms (e.g., [6]). Constraint-based modelling (CBM) [15] is one of the promising approaches for building ITSs. This technique assumes that a cognitive skill can be acquired in form of declarative knowledge. This approach has been applied successfully to the domains of natural language [12], SQL [13], and for database design [14]. The strength of this approach is that the space of correct solutions is modelled by using constraints to represent a number of domain principles and properties of correct solutions instead of enumerating every correct solution. Furthermore, it is not necessary to

C. Delgado Kloos et al. (Eds.): EC-TEL 2011, LNCS 6964, pp. 233–247, 2011.

anticipate errors possibly made by students. The goals of this paper are twofold. First, we argue that the pure CBM approach is limited in providing appropriate feedback to solutions for problems whose solution space is relatively large, e.g., in programming. Second, we propose and evaluate a weighted constraint-based model which adopts a soft computing approach for solving constraint satisfaction problems: each constraint is enriched with a weight value indicating the importance of the constraint. The weighted constraint-based model is superior to the pure CBM approach with respect to the capability of error diagnosis.

2 A Limitation of a CBM Tutoring System

We illustrate a limitation of the CBM approach in the example domain of programming. According to [9], a programming problem might have several solution strategies and each strategy can be implemented in different solution variants. The sample task *Investment*: *"Calculate the return after investing an amount of money at a constant yearly interest rate"* can be solved by applying different solution strategies, including the following two:

1. *Analytic strategy*: The profit of investing a sum of money with a yearly interest rate is calculated based on mathematical geometric series.
2. *Tail recursive strategy*: A variable can be used to accumulate the sum of investing money and its interest after each year.

In logic programming, the analytic strategy and tail recursive strategy can be implemented as follows:

Analytic strategy:

```
invest(Money,Rate,Period,Return):-Return is Money*(Rate+1)^Period.
```

Tail recursive strategy:

```
inv(M,_,P,Ret):- P=0,M=Ret.
inv(M,R,P,Ret):-P>0, NM is M*R+M, NP is P-1, inv(NM,R,NP,Ret).
```

Each solution strategy can be implemented in many solution variants. For instance, the tail recursive strategy can be implemented in many ways by varying the order of the two clauses or the order of the second and third subgoal in the second clause, by choosing one of two unification techniques (implicit and explicit), or by using the commutative and distributive laws in mathematics to transform arithmetic expressions. As a result, there exist several thousands of correct implementations for the problem *Investment* in total. How can constraints be used to model such a large solution space?

Ohlsson [15] proposed using only constraints to model specific properties of correct solutions. If a solution violates a constraint, the solution does not satisfy a semantic requirement of correct solutions. If the analytic solution strategy above should be modelled using constraints, we have to identify all possible pro-perties required to implement this strategy. For instance, the following constraint represents a property which requires the correct implementation of the exponent term of the analytic formula. Note, for simplicity, this constraint assumes that the order of the argument positions in the clause head is fix.

IF a calculation subgoal S exists **AND** one multiplication term T_m exists on the right hand-side of S **AND** T_m consists of two product factors **AND** one factor is a variable unified with the 1^{st} position of the clause head **AND** the 2^{nd} factor is an exponential term T_e

THEN the exponent of T_e is unified with the 3^{rd} position of the clause head **AND** the exponent basis is a sum of the value 1 and a variable unified with the 2^{nd} position of the clause head

This constraint is specified with five propositions in the relevance part (IF-clause). Such a constraint, whose relevance part contains many conditions, tends to fail in erroneous situations, because the relevance part is not robust against minor deviations from the specified situation. A complex constraint with a conjunction of conditions in the relevance part becomes irrelevant for a student solution if a single one of the conjuncts fails. For example, the following student solution, which implements the analytic strategy, would not match the relevance part of the constraint above, because in the clause body of the student solution the multiplication term consists of one product factor (namely `(Rate+1)^Period`), while the relevance part of the constraint requires two.

A sample erroneous student solution:

```
invest(Money,Rate,Period,Return):-Return is (Rate+1)^Period.
```

Thus, the constraint above can be satisfied even though this undesired result might have been caused by another error elsewhere in the student solution (e.g., one product factor is missing). The potential that complex constraints might become useless is obvious when specifying constraints for the domain of programming. Because the relevance part of a constraint should describe a problem state, it makes no sense to make the relevance part simpler. (If the satisfaction part of a constraint is highly specific, then the constraint can be broken into several simpler constraints according to the rule: $A \rightarrow B \wedge C \equiv (A \rightarrow B) \wedge (A \rightarrow C)$). The approach of using constraints as the only means to model correct solutions produces not only complex but also task-specific constraints. Every time new tasks need to be integrated into a CBM system, it is necessary to specify new task-specific constraints. This is not an easy undertaking task for authors who are not familiar with the constraint representation.

Instead of specifying task-specific requirements in constraints, Ohlsson and Mitrovic [16] suggested using an *ideal solution* to capture the characteristics of correct solutions and defining constraints to compare the necessary components of the ideal solution with a student solution. The approach of using an ideal solution to encapsulate the semantic requirements of solutions has the advantage that manually created complex constraints can be avoided to a certain extent. Furthermore, it is not necessary to specify new constraints for a new task because task-specific requirements are contained in the ideal solution, assuming that the existing set of constraints can cover the learning domain sufficiently. However, choosing an ideal solution among many alternatives for a programming problem is not an easy task. Assuming that an ideal solution for the problem *Investment* is available, similar to the way of defining constraints using an ideal solution, we might define constraints to cover different solution variants in logic

programming. For instance, to solve the sample task *Investment*, the tail recursive strategy requires implementing a guard which checks whether the investment period is positive before it can be decremented recursively. The guard can be implemented in a variety of ways: e.g., `P>0`, `P>=1`, `0<P`, `1=<P`. If we choose the implementation of the tail recursive strategy as an ideal solution, the following constraint checks the correctness of an arithmetic expression and its variants, where $\triangleleft, \triangleleft_s \in \{=<, >=, >, <\}$ and r is a function which finds a reverse operator for an arithmetic comparator according to the following rules: $r(>)\rightarrow<$, $r(<)\rightarrow>$, $r(=<)\rightarrow>=$, $r(>=)\rightarrow=<$.

IF In the ideal solution, there exists an arithmetic test $X \triangleleft Y$ **AND** $SX \triangleleft_s SY$ is a corresponding subgoal in the student solution
THEN $\triangleleft_s$ is identical to $\triangleleft$, and (SX, SY) correspond to (X, Y) **OR** $\triangleleft_s = r(\triangleleft)$, and (SX, SY) correspond to (Y, X)

This constraint is useful for capturing different solution variants of a guard which is required by the tail recursive strategy. However, this constraint is meaningless if student solutions implementing the analytic strategy are diagnosed, because this constraint is intended to check the guard (specified in the IF-clause) and the analytic strategy does not require one. The problem would even be more serious if a constraint is specified to require the existence of a guard like the following.

IF In the ideal solution, there exists an arithmetic test $X \triangleleft Y$
THEN a corresponding $SX \triangleleft_s SY$ exists in the student solution
HINT A guard is missing.

If this constraint is used to evaluate a student solution which implements the analytic strategy, then it would be violated and the error diagnosis results in a feedback "A guard is missing" which is obviously misleading. This is because the solution strategy the student intended to implement is not the same as the one the constraints are based on (tail recursive strategy). This problem has also been identified in the domain of SQL in [11, p. 43] and [7, p. 321], as is discussed in [18, p. 85]. This problem raises the need to hypothesize the solution strategy underlying a solution during the process of diagnosing errors. Once the solution strategy of the student has been identified, it makes sense to evaluate constraints in the context of that specific solution strategy only.

To determine the most plausible hypothesis about the student's solution variant in the context of language learning, Menzel [12] proposed to count the number of constraint violations for each hypothesis. The hypothesis which causes least constraint violations is assumed to be plausible. Yet, the author indicated that this kind of measure is too gross, because constraints which represent grammar rules might not have the same level of importance. For example, if the sentence "These fish stinks" is diagnosed, two hypotheses can be generated: 1) a constraint (C1) requiring the agreement of determiner-noun is violated; 2) a constraint (C2) representing the agreement of noun-verb cannot be satisfied. If only the number of violated constraints is used to evaluate each hypothesis, this might result in inaccurate diagnoses, because researchers in language learning might consider C1 as more or less severe than C2.

As a result, we conclude that the classical CBM approach is limited in providing appropriate feedback to solutions of problems whose solution space is relatively large, e.g., in programming. In the next section, we introduce a weighted constraint-based model which adopts the idea of a *probabilistic* approach for solving constraint satisfaction problems to improve the capability of error diagnosis of CBM tutoring systems.

3 A Weighted Constraint-Based Model for ITS

In order to coach the student, an ITS needs to identify shortcomings in the student solution and to provide appropriate feedback according to the student's intention. Hence, the student solution needs to be analyzed correctly. In the approach proposed in this paper, a weighted constraint-based model (WCBM) serves this purpose. The model includes four modeling components: a *semantic table*, a set of *constraints*, constraint weights, and a set of *transformation rules*. The semantic table represents solution strategies for a given problem. Each solution strategy is described by a set of semantic components. Constraints are used to establish a mapping between the student solution and requirements in the semantic table and to check general well-formedness conditions. Domain-specific transformation rules can be exploited to extend the coverage of different solution variants. A constraint weight represents the importance of each constraint. The error diagnosis process is controlled using these four components.

3.1 Semantic Table

Instead of using a single ideal solution to capture task-specific requirements, the WCBM approach presented in this paper is built on a so-called *semantic table* which serves two proposes: 1) modeling several solution strategies, and 2) representing model solutions in a relational form. The first characteristic serves to hypothesize the most plausible strategy underlying a student solution. The second one has the advantage that solution variants (e.g., created by changing the order of solution components) can easily be covered. Table 1 illustrates a partial semantic table for the problem *Investment*, covering two alternative solution strategies (more could easily be added), where the first row represents the generalized description of the analytic solution strategy and the remaining ones are the generalized description of the tail recursive solution strategy.

3.2 Constraints

In this framework, constraints are employed to model the solution space for each problem and to identify errors based on this model. We distinguish between two types of constraints: *general* and *semantic* constraints.

General Domain Principles: A domain is characterized by certain principles, which every solution variant of any problem must adhere to and which are independent of task-specific requirements. For instance, in logic programming, it

Table 1. An partial semantic table for the example problem *Investment*

Strategy	CI	Head	SI	Subgoal	Description
Analytic	1	p(S,R,P,Ret)	1	Ret is S*(R+1)^P	Using a formula
Tail recursive	1	p(M,_,P,Ret)	1	P=0	Recursion stops
Tail recursive	1	p(M,_,P,Ret)	2	Ret=M	Recursion stops
Tail recursive	2	p(M,R,P,Ret)	1	P>0	Check period
Tail recursive	2	p(M,R,P,Ret)	2	NM is M*R+M	Calculate new money
Tail recursive	2	p(M,R,P,Ret)	3	NP is P-1	Update period
Tail recursive	2	p(M,R,P,Ret)	4	p(NM,R,NP,Ret)	Recur with new period

CI: clause index; SI: subgoal index

is required that all variables on the right hand-side of an arithmetic calculation subgoal (e.g., `X is A+B`) must be instantiated. Otherwise, the evaluation of this arithmetic expression will fail. Such domain principles can be modeled by means of *general constraints*. These constraints can be instantiated by constraint schemas of type (1), where the problem situation X and the condition Y can be composed of many elementary propositions using conjunction or disjunction operators. A problem situation X describes a static state of solution components required to solve an arbitrary task in the domain. If a student solution violates one of the general constraints, the student did not consider the corresponding principle of the domain.

Type **(1)**: **IF** problem situation X is relevant **THEN** condition Y must be satisfied

Semantic Correctness: Using task-specific information specified in the semantic table, constraints can be specified to check the semantic correctness of a student solution (STS). We refer to this kind of constraints as *semantic constraints* which have the following schemas. Constraint schemas (2.1) and (2.2) check for missing or superfluous components in the student solution, respectively.

Type **(2.1)**: **IF** in the semantic table, a component X exists
THEN in the STS, a component corresponding to X exists

Type **(2.2)**: **IF** in the STS, a component Y exists
THEN in the semantic table, a component corresponding to Y exists

If the goal is to check a required value of a component, the following schema can be applied:

Type **(2.3)**: **IF** in the semantic table, a component Z exists and has value A
THEN in the STS, a component corresponding to Z has value A

To model different solution variants for a concept (e.g., an arithmetic comparison), constraint schema of type (3) can be applied to cover two possible variants using the OR-operator.

Type **(3)**: **IF** in the semantic table, a component X exists
THEN in the STS, a component corresponding to X **OR** a variant of X exists

3.3 Transformation Rules

To extend the coverage of the solution space for a problem, various transformation rules which are domain-specific can be exploited, for instance: program transformation in the domain of programming, model transformation in the domain of computational modeling, or mathematical transformation rules.

3.4 Constraint Weights

As discussed in Section 2, classical CBM tutoring systems might provide misleading diagnostic results and are not well suited to decide on the most plausible hypothesis about the student's solution variant, because constraints are solely based on a binary logic (constraint is either violated or not). We need a means to enhance the error diagnosis capability of classical CBM tutoring systems. To serve this purpose, we adopt approaches to softening constraints in a constraint satisfaction problem (CSP), because constraint-based error diagnosis is a CSP whose goal is to identify inconsistencies between an erroneous solution and a constraint system.

The most popular approaches to softening constraints include fuzzy CSP [1], partial CSP [4], cost-minimizing CSP[1] [17], and probabilistic CSP frameworks [2]. The probabilistic CSP approach is most appropriate for constraint-based error diagnosis, because it does not evaluate a constraint system partially (like the partial CSP framework), nor is it necessary to specify constraints with possible instantiations of constraint variables in advance (like fuzzy and cost-minimizing CSP frameworks). This approach has been applied successfully to enhance the quality of error diagnosis, e.g., in the context of a natural language parser [3]. In the approach proposed in this paper, we adopt the probabilistic approach to enhance the diagnosis capability of classical CBM tutoring systems by attaching a *weight*, indicating the measure of importance, to each constraint.

Searching the most plausible hypothesis about the student's solution variant, we need to evaluate the plausibility of all possible hypotheses. For this purpose, adopting the probabilistic approach, we apply a multiplicative model: constraint weights are taken from the interval $[0; 1]$. The value 1 represents the weight for least important constraints and 0 indicates the weight for constraints which model the most important requirements. Constraints of the latter type can be considered *hard constraints*. The importance of a constraint is determined based on the role of the components being investigated. Constraints checking a component which contributes more information to the overall correctness of the solution should receive a weight close to the value 0. The determination of the importance level for constraints resembles the assessment of written examinations by a human tutor: if a solution contains more important components, then it receives a better mark. Constraint weight values need to be adjusted manually to yield acceptable diagnostic results.

[1] This term is also referred to as *weighted* CSP in the literature.

3.5 Error Diagnosis

Given a student solution, the error diagnosis process generates hypotheses about it by matching the student solution against each of the solution strategies specified in the semantic table (left part of Fig. 1). The matching process initializes *global mappings* representing hypotheses about the strategy underlying the student solution. Then, the hypothesis generation process continues to generate hypotheses about the student's solution variant by matching components of the student solution against the ones representing the selected solution strategy. That is, highest-level components of the student solution are matched against components of the same level specified for each solution strategy in the semantic table, similarly for middle-level and lowest-level components. The matching process results in *local mappings* which represent hypotheses about the student's solution variant. They are used to complete the global mappings. Depending on the structural hierarchy of a solution, local mappings are generated according to the levels of that hierarchy.

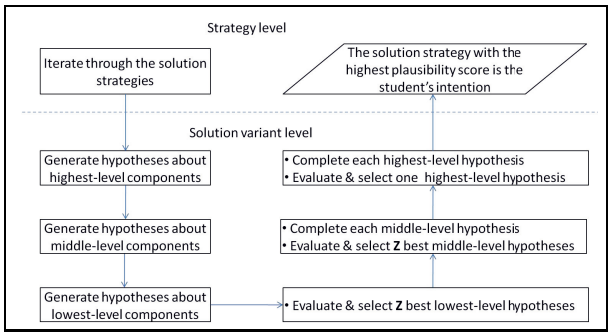

Fig. 1. The process of error diagnosis

After hypotheses on the lowest component level have been generated, they are evaluated with respect to the plausibility, and the best ones are selected using the Beam criterion Z (the right part of Fig. 1). The plausibility of each hypothesis is computed by multiplying the weights of constraints violated by that hypothesis according this formula, where W_i is the weight of a violated constraint: $Plausibility_{Prod}(H) = \prod_{i=1}^{N} W_i$.

The plausibility of each generated hypothesis can be used as a Beam criterion to restrict the space of hypotheses to the most plausible ones: Only a fraction of most plausible hypotheses whose plausibility score is higher than Z ($0 \leq Z \leq 1$) are selected for each level. The best selected hypotheses on the lower level are used to multiply the space of hypotheses on the next higher level. This procedure of generating hypotheses, selecting the best ones using the Beam criterion, and multiplying the space of hypotheses of the next level continues up to the highest one of the structural hierarchy. At this level, the process of generating strategy hypotheses is completed. The error diagnosis process is finished by evaluating

the strategy hypotheses with respect to their plausibility and choosing the best hypothesis for the selected solution strategy. Diagnostic information is derived from constraint violations resulting from the plausibility computation of the most plausible strategy hypothesis.

3.6 Example

Following is a sample student solution in logic programming for the problem *Investment*:

Clause	**Type**	**Implementation**
SC1	base	`interest(Amount,_,0,Amount).`

SC2	recursive	`interest(Amount,Rate,Duration,End):-`
		`X is Amount+Amount*Rate/100, Duration>0,`
		`NewDura is Duration-1,interest(X,Rate,NewDura,End).`

According to the algorithm of the error diagnosis, first, the student solution is matched to the generalized solution description of the analytic strategy, then, the tail recursive strategy. This process results in two global mappings, where SP is the solution provided by the student:

$H_{strategy}$={map(Strategy_Analytic, SP)}
$H_{strategy}$={map(Strategy_Tail_Recursive, SP)}

On the solution variant level, the error diagnosis process matches SP against the semantic components of each solution strategy. Since the structural hierarchy of a logic program consists of clause, subgoal, argument/operator, multiplication term, and product factor (where the last two levels exist only if the solution contains mathematical expressions), the matching process is carried out on the corresponding levels of the hierarchy. For instance, the matching between SP and the generalized description of the strategy *tail recursive* (cf. Table 1) on the clause level results in a single mapping H_{clause} which has two entries. The first component (C_i) of each entry represents the expression specified in the generalized description and the second one (SC_i) is provided by the student solution.

H_{clause}={map(C1, SC1), map(C2, SC2)}

On the subgoal level, the subgoals of the student's clause are mapped against the subgoals of the corresponding clause of the generalized description. For example, taking the second element of the mapping H_{clause} above, subgoals of C2 are matched against subgoals of SC2. Considering, for instance, only arithmetic calculation subgoals, matching the two arithmetic calculation subgoals of the student's clause SC2 against two arithmetic calculation subgoals of the generalized description's clause C2 results in two mappings of arithmetic subgoals:

$H_{subgoal}$(cal)= {map(NP is P-1, DecDuration is Duration-1),
map(Ret is NM+R*NM, X is Amount+Amount*Rate/100)}
$H_{subgoal}$(cal)= {map(NP is P-1, X is Amount+Amount*Rate/100),
map(Ret is NM+R*NM, NewDura is Duration-1)}

On the argument/operator level, the arguments of a student's subgoal are matched against the arguments of the corresponding subgoal of the generalized description. E.g., `DecDuration is Duration-1`, an arithmetic subgoal of

the student solution, is matched against the subgoal `NP is P-1` of the generalized description: arguments on the left hand-side and on the right hand-side are matched, respectively:

$H_{argument}$={map(NP, DecDuration), map(P-1, Duration-1)}.

Similarly, the hypothesis generation process continues to the product factor level. After local mappings have been generated on the product factor level, their plausibility is evaluated by invoking the constraints of that level. Based on the plausibility score of each local mapping, a set of best mappings is selected and used to multiply the space of local mappings on the next higher level, namely the multiplication terms. Again, each of the local mappings on the multiplication term level is evaluated with respect to its plausibility. The process of evaluating local mappings, choosing the best ones, and extending the space of mappings on the higher level continues until the clause level is reached. At this level, a space of global mappings for each generalised solution description is established.

The algorithm next evaluates the plausibility of these global mappings and determines the one which has the highest score for each generalised solution description. According to the first column of Table 2, the hypothesis that the student has implemented the strategy *tail recursive* is more plausible because it has the highest plausibility score (0.25). The second column of the table shows diagnoses resulting from the evaluation of each hypothesis. The evaluation of the most plausible hypothesis results in the following diagnostic information:

Hint 1 In the 2nd clause, the denominator **100** is superfluous.
Hint 2 In the 2nd clause, a variable is expected instead of **Betrag/100**.

Table 2. Plausibility of hypotheses about the implemented solution strategy

Str.; Score	**Weight; Hints**
A; 1e-006	0.01; If you intend to define a non-recursion predicate, then it must exist at least a clause of type non-recursive. 0.01; The predicate definition has more base case than required. 0.01; The predicate definition has more recursive case than required.
T; 0.25	0.5; In the 2nd subgoal, the denominator **100** is superfluous. 0.5; In the 2nd clause, a variable is expected instead of **Betrag/100**.

Str. A: Analytic strategy; Str. T: Tail recursive strategy.

4 Evaluation

To test whether the weighted constraint-based model is more capable than the classical CBM approach with respect to its diagnostic capability (of course, a more precise error diagnosis component can be used to give better learning support), we have used the intention analysis capability of a tutoring system for logic programming named INCOM [8]. The purpose of the intention analysis is to determine the rate of student solutions whose solution strategy and components are identified correctly by the system. In literature, this kind of evaluation

technique is also noted as *algorithm analysis* [5,10], because the approach of identifying the solution strategy is based on anticipated algorithms for a programming problem.

To compare the diagnostic capability of the weighted constraint-based model with the classical CBM approach, we used two versions of INCOM. The first one applies the weighted constraint-based model (WCBM-INCOM). A modified version of INCOM uses constraints without weight values (CBM-INCOM), and corresponds to a classical CBM tutoring system. Here, the plausibility of hypotheses about the solution strategy intended by the student is computed by summing up the number of constraint violations caused by each hypothesis: $Plausibility_{Add}(H) = |C|$, where C is the set of all constraint violations caused by each hypothesis H. (Note, this formula does not use the weight value of constraints like the formula $Plausibility_{Prod}$).

4.1 Study Design

For the evaluation study, we collected exercises and solutions from past written examinations. The examination candidates had attended a course in logic programming which was offered as a part of the first semester curriculum in Informatics. The following seven tasks have been collected from the written examinations (here, the tasks are described briefly).

1. Access to specific elements within an embedded list;
2. Querying a data base and applying a linear transformation to the result;
3. Modification of all elements of a list subject to a case distinction;
4. Creation of an n-best list from a data base;
5. Computing the sum of all integer elements of a list;
6. Counting the number of elements in an embedded list;
7. Finding the element of an embedded list which has the maximum value for a certain component.

For these problems, 221 student solutions were selected according to the following criteria:

- Any piece of code which satisfies minimal requirements of interpreting it as a Prolog program is considered a solution
- Syntax errors in the solutions are ignored (because during the written examination the students did not have access to a computer)
- Both correct and incorrect solutions are taken into account.

For conducting the intention analysis, we involved a human expert who inspected every student solution manually. Student solutions which could be understood by the human expert, were classified as "*understandable*", the others as "*not understandable*". Then, all "*understandable*" solutions were given as input to the two systems under comparison (WCBM-INCOM and CBM-INCOM), which resulted in two lists of constraint violations – one for each system. The human

expert examined the two lists of constraint violations and decided whether the systems analyzed the student solution correctly (in terms of determining the solution variant).

4.2 Results

Table 3 summarizes the statistics of the evaluation. The number of available student solutions per task is indicated in the second column. The third column represents the number of solutions classified as "*not understandable*". The fourth and the fifth column show the absolute and relative amount of solutions which have been analyzed correctly by the WCBM-INCOM system. Similarly, the sixth and the seventh column indicate the absolute and relative number of solutions analyzed correctly by CBM-INCOM.

On average, 90.3% (s.d.=11.0%) of 221 collected student solutions could be analyzed correctly by WCBM-INCOM [8], while CBM-INCOM could achieve correct intention analysis in 35.5% (s.d.=19.9%) of the same solution corpus. In all tasks, the statistics show that the intention analysis capability of WCBM-INCOM is better than the one of CBM-INCOM. In particular, we notice that for Tasks 1-3, the intention analysis capability of CBM-INCOM is higher than 50%, whereas for Tasks 4-7 the intention analysis capability of CBM-INCOM is remarkably worse (Task 6 has the worst intention analysis with only 9.9% correct analysis). This can be explained by the fact that the complexity of Tasks 4-7 is much higher than the one of Tasks 1-3 – here, more possible solution strategies can be (and have been) applied and more implementation variants can be created. The statistics also show that the diagnosis accuracy of the WCBM-INCOM does not show this significant difference between easy and more complex tasks: apparently, adding the constraint weights was sufficient for achieving a much better diagnosis performance.

We next want to illustrate this effect with a concrete example from Task 6 (c.f. Section 4.1) which can be solved by applying either a naive recursive strategy or a tail recursive strategy.

Table 3. Evaluation of the intention analysis

			WCBM-INCOM		CBM-INCOM	
Task	**Solutions**	**Not U.**	**C.A.#**	**C.A.%**	**C.A.#**	**C.A.%**
1	10	0	10	100.0	5	50.0
2	11	0	10	90.9	7	63.6
3	6	1	4	66.7	3	50.0
4	17	1	16	94.1	5	29.4
5	58	2	54	93.1	8	13.8
6	81	0	79	97.5	8	9.9
7	38	2	34	89.5	12	31.6
Sum	221	6	207		48	
Avg.(sd.)				90.3 (11.0)		35.5 (19.9)

Not U.: Not Understandable; C.A.: Correct Analysis.

```
countz(N,L):- L=[], N is 0.
countz(N,L):- L=[Head|Rest], countz(N1, Rest), N is N1+1.
```

The results of error diagnosis executed by WCBM-INCOM and CBM-INCOM for the erroneous student solution above are shown in Table 4. Here, CBM-INCOM was not able to decide on the most plausible hypothesis about the student's solution variant, because two hypotheses of CBM-INCOM (naive recursive and tail recursive) have the same plausibility score (each hypothesis produces five constraint violations). On the contrary, WCBM-INCOM decided on the hypothesis that the student implemented the naive recursive strategy because the plausibility score of this solution strategy (0.064) is higher than the one of the tail recursive strategy (1e-010).

Table 4. Intention analysis for the sample student solution

Strategy	WCBM-INCOM $Plausibility_{Prod}$	CBM-INCOM $Plausibility_{Add}$
Naive recursive	0.064	5
Tail recursive	1e-010	5

In addition to this example, where CBM-INCOM was not able to choose the most plausible hypothesis, there were many cases of wrong intention analysis provided by CBM-INCOM. It even favoured a wrong solution strategy if this had more (but less severe) errors. Here, adding the weights to constraints solved the problem and improved the error diagnosis.

5 Conclusion and Outlook

In this paper, we have argued that the constraint based modelling approach, while useful for ITS development in general, may reach its limitation when it is applied for developing tutoring systems for tasks which have a large solution space. An example for this effect from the domain of programming has been discussed. Additionally, we have introduced a weighted constraint-based model which adopts a probabilistic approach for solving constraint satisfaction problems. In a study, we have shown that this approach is more capable than the pure CBM with respect to error diagnosis: A tutoring system for logic programming based on this extended model was able to analyze the student's intention correctly in 90.3% of 221 student solutions, while a corresponding system which is based on the classical CBM approach was able to achieve a correct intention analysis in only 35.5% of the same corpus of student solutions.

While the examples in this paper were all chosen from logic programming, we believe that the results are valid also beyond this field. Specifically, it is possible to easily apply the weighted constraint-based model to functional programming languages because they are also instances of the declarative programming

paradigm. This characteristic, the atemporal nature, of declarative programming languages makes possible to formulate and to check the well-formedness conditions for a program in a static manner. Whether the weighted constraint-based model can also be applied to other declarative domains (e.g., computational modeling) or other programming paradigms (e.g., imperative programming) is a open question which we are currently investigating.

References

1. Dubois, D., Fargier, H., Prade, H.: Possibility Theory in Constraint Satisfaction Problems: Handling priority, preference and uncertainty. J. Applied Intelligence 6, 287–309 (1996)
2. Fargier, H., Lang, J.: Uncertainty in Constraint Satisfaction Problems: a Probabilistic Approach. In: Moral, S., Kruse, R., Clarke, E. (eds.) ECSQARU 1993. LNCS, vol. 747, pp. 97–104. Springer, Heidelberg (1993)
3. Foth, K.: Hybrid Methods of Natural Language Analysis, Shaker, Germany (2007)
4. Freuder, E.C., Wallace, R.J.: Partial Constraint Satisfaction. J. Artificial Intelligence 58, 21–70 (1992)
5. Johnson, W.L.: Understanding and debugging novice programs. J. Artificial Intelligence 42(1), 51–97 (1990)
6. Koedinger, K.R., Corbett, A.: Cognitive Tutors: Technology bringing learning science to the classroom. In: Sawyer, K. (ed.) The Cambridge Handbook of the Learning Sciences, pp. 61–78. Cambridge University Press, Cambridge (2006)
7. Kodaganallur, V., Weitz, R., Rosenthal, D.: An Assessment of Constraint-based Tutors: A Response to Mitrovic and Ohlsson's Critique of "A Comparison of Model-Tracing and Constraint-based Intelligent Tutoring Paradigms". J. of AI in Education 16, 291–321 (2006)
8. Le, N.T., Menzel, W.: Using weighted constraints to diagnose errors in logic programming - The case of an ill-defined domain. J. on Artificial Intelligence in Education 19(4), 382–400 (2009)
9. Le, N.T., Menzel, W., Pinkwart, N.: Considering Ill-Definedness Of Problem Tasks Under The Aspect Of Solution Space. In: 23_{st} Int. Conference of the Florida Artificial Intelligence Research Society (FLAIRS), pp. 534–535. AAAI Press, Menlo Park (2010)
10. Looi, C.K.: Automatic Debugging of Prolog Programs in a Prolog Intelligent Tutoring System. J. of Instructional Science 20, 215–263 (1991)
11. Martin, B.: Intelligent Tutoring Systems: The Practical Implementation Of Constraint-based Modelling. PhD thesis, University of Canterbury (2001)
12. Menzel, W.: Modellbasierte Fehlerdiagnose in Sprachlehrsystemen. Niemeyer, Tübingen (1992)
13. Mitrovic, A., Mayo, M., Suraweera, P., Martin, B.: Constraint-based Tutors: A Success Story. In: 14_{th} International Conference on Industrial and Engineering Applications of AI and Expert Systems, pp. 931–940. Springer, Heidelberg (2001)

14. Mitrovic, A., Suraweera, P., Martin, B., Weerasinghe, A.: DB-Suite: Experiences With Three Intelligent, Web-based Database Tutors. J. of Interactive Learning Research 15(4), 409–432 (2004)
15. Ohlsson, S.: Constraint-based Student Modeling. In: Greer, J.E., McCalla, G.I. (eds.) Student Modelling: The Key to Individualized Knowledge-based Instruction, pp. 167–189. Springer, Berlin (1994)
16. Ohlsson, S., Mitrovic, A.: Constraint-based Knowledge Representation For Individualized Instruction. J. Computer Science and Inf. Systems 3(1), 1–22 (2006)
17. Schiex, T., Cedex, C.T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Joint Conference in AI (1995)
18. Woolf, B.P.: Building Intelligent Interactive Tutors. Morgan Kaufmann, San Francisco (2009)