УДК 004.5 + 004.588
ББК 32.973.26-018.2

К АВТОМАТИЗИРОВАННОЙ ПРОВЕРКЕ РЕШЕНИЙ ОДНОГО КЛАССА ЗАДАЧ В СЛЕДЯЩИХ ИНТЕЛЛЕКТУАЛЬНЫХ ОБУЧАЮЩИХ СИСТЕМАХ

Смирнова Н. В.[1]
(*ФГБУН Институт проблем управления РАН, Москва*)

Рассматриваются обучающие программы, обеспечивающие интерактивную поддержку процесса решений задач предметов естественнонаучного цикла и их автоматизированную оценку. Описывается алгоритм выявления зависимостей между формулами, посредством которых представляются сведения об известных решениях учебной задачи, и формулой, введенной обучаемым, позволяющий в ряде случаев определить, является ли введенная обучаемым формула верным шагом в решении задачи или его имитацией. Данный алгоритм является частью алгоритма измерения прогресса в решении задачи.

Ключевые слова: следящая интеллектуальная обучающая система, автоматизированная обучающая система, автоматическая проверка решений, дерево и/или, символьные вычисления

1. Введение

Следящими интеллектуальными обучающими системами (следящими ИОС, англ. model-tracing intelligent tutoring systems) называются программы, предназначенные для обучения предметам естественнонаучного цикла, обеспечивающие интерактивную поддержку процесса решения задач обучаемым и автоматизированную оценку его решений. Название «следящие

[1] *Наталия Викторовна Смирнова, научный сотрудник* (*smirnovanatalia2008@gmail.com*).

ИОС» обусловлено тем, что для проверки хода решения обучаемого на завершенность и правильность необходимо сверять шаги решения обучаемого с шагами имеющихся в ИОС решений. На рис. 1 в качестве примера следящей ИОС представлен скриншот интерфейса прототипа обучающей системы, разработанного в лаборатории 17 ИПУ РАН (в подсистеме проверки решений разработанного прототипа используется «наивный» способ измерения прогресса в решении, подробнее о нем в следующем разделе статьи).

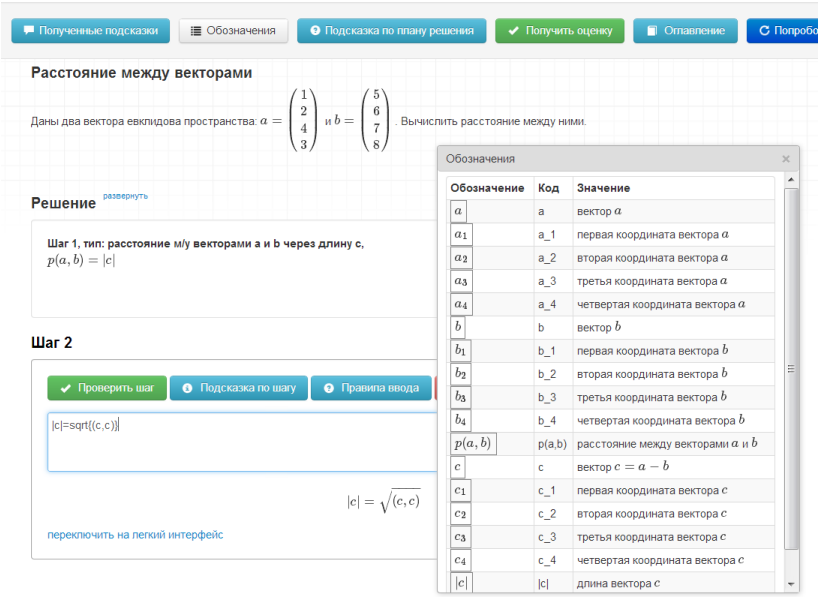

Рис. 1. Скриншот интерфейса решения задачи следящей ИОС «Волга»

К одному из наиболее распространенных типов следящих ИОС относятся программы, в которых в качестве решения требуется ввести одну формулу или заполнить текстовые поля, соответствующие недостающим частям некоторой формулы. Примером такой системы является STACK [11]: подсистема, разработанная для автоматического оценивания заданий по

алгебре и началам анализа [12], которая может быть интегрирована в другие обучающие среды, например, в Moodle [10]. Также в качестве примера подобных систем можно упомянуть ActiveMath [9] и ряд электронных учебников по школьной математике, созданных на основе работ М.А. Левинской [3].

По-видимому, наиболее развитой программой, в которой реализована проверка решений, вводимых в достаточно свободной форме, является Andes Physics Tutor [13, 17]. В следующих разделах статьи будет подробно описана подсистема проверки решений Andes Physics Tutor. В этой подсистеме важную роль играет алгоритм определения того, от каких формул из формул, посредством которых представляются сведения об известных решениях учебной задачи, зависит формула, введенная обучаемым. В данной работе предлагается альтернативный алгоритм выявления зависимостей между формулами, который, в отличие от алгоритма, реализованного в Andes Physics Tutor, способен обрабатывать формулы, содержащие векторы и матрицы, а также потенциально является менее уязвимым к имитации правильно введенных шагов решения.

2. Постановка задачи

Далее рассматриваются определенные формальные задачи на нахождение искомых объектов (см. классификацию учебных задач в [7]). При этом предполагается, что:

1) решение обучаемого состоит из ряда формул и не содержит естественно-языковых вставок;

2) при вводе формул студент может использовать только те обозначения, которые внесены в программу автором курса для данной задачи и не может вводить свои обозначения (см. рис. 1);

3) известные решения рассматриваемых задач состоят только из таких формул, которые могут быть проверены без привлечения сведений о других формулах, ранее введенных обучаемым (к примеру, этому условию не удовлетворяют задачи, для решения которых необходимо решать систему уравнений методом Гаусса).

Разработчики Andes Physics Tutor предложили осуществлять проверку шага решения студента в два этапа (шагу решения, как правило, соответствует одна формула, введенная с использованием LaTeX-подобного синтаксиса). На первом этапе осуществляется верификация шага, т.е. первичная проверка шага. Если шаг успешно верифицируется, то далее запускается алгоритм измерения прогресса в решении.

Верификация шага решения осуществима с помощью простого эвристического алгоритма. Например, в Andes Physics Tutor используется прием «color by numbers», заключающийся в следующем. В формулу, введенную обучаемым, подставляются значения содержащихся в ней переменных. Далее выполняется проверка на равенство обеих частей получившегося выражения. Известно, что использовании такого способа проверки некорректность подвыражений в формуле, введенной обучаемым, может быть не выявлена, если они умножаются на выражение, которое равно 0. В Andes Physics Tutor предусмотрены специальные эвристики для выявления подобных случаев. В процессе проверки на равенство двух выражений осуществляется упрощение выражений. Для упрощения выражений используются библиотеки символьных вычислений (примером такой библиотеки является SymPy [8]).

Проблема измерения прогресса в решении представляет больший интерес. Наиболее простой способ ее решения заключается в представлении каждого известного решения задачи в виде списка формул и организации сравнения формулы, введенной обучаемым, с формулами из этих списков. Этот способ плох тем, что даже для самой простой задачи требуется ввести слишком много формул и решений, многие из которых незначительно отличаются друг от друга. Так, например, для задачи «Решение квадратного уравнения» автором в прототип обучающей системы, в котором использовался «наивный» способ измерения прогресса в решении, было введено 16 решений, при том, что были учтены не все подходы к решению этой задачи. Иными словами, при использовании этого способа возникает проблема множественности эквивалентных комбинаций формул (МЭКФ), генерируемых обучаемыми при решении задач.

Для иллюстрации работы описываемых в статье алгоритмов будет использоваться следующая задача.

Задача.

Дано: векторы евклидова пространства

$$a = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, b = \begin{pmatrix} 3 \\ 4 \end{pmatrix}.$$

Вычислить расстояние между a и b.

Часть решений данной задачи может быть сгенерирована на основе следующих формул:

(1) $p(a,b) = |c|$;

(2) $|c| = \sqrt{(c,c)}$;

(3) $(c,c) = c_1^2 + c_2^2$;

(4) $c = a - b$;

(5) $c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$;

(6) $a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$;

(7) $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$;

(8) $a_1 = 1$;

(9) $a_2 = 2$;

(10) $b_1 = 3$;

(11) $b_2 = 4$.

Поясним проблему МЭКФ на примере этой задачи. Комбинация формул сводится к замене величин в одной формуле алгебраическими выражениями, полученными из других формул, участвующих в комбинации. Например, комбинация формул (1) и (2) имеет вид:

$$p(a,b) = \sqrt{(c,c)},$$

а комбинация формул (1)–(3) такова:

$$p(a,b) = \sqrt{c_1^2 + c_2^2}.$$

Можно представить себе много возможных решений задачи: решение, содержащее формулы (1), (2) и (3); решение, содержащее формулу (1) и комбинацию формул (2) и (3); решение, содержащее комбинацию формул (1)–(3) и т.д. Более эффективной оказывается такая архитектура обучающей системы, при использовании которой автор курса для каждой задачи вносит

- формулы, на основе которых могут быть сгенерированы все известные ему решения и подсказки к ним (будем для краткости называть их «базовыми»);
- известные величины и их значения («дано…»);
- искомые величины («найти…»),

а известные ему решения генерируются системой автоматически. Измерение прогресса в решении может быть реализовано примерно следующим образом. При проверке шага решения обучаемого

1) выявляется, на основе каких «базовых» формул могла бы быть получена введенная обучаемым формула (т.е. какие «базовые» формулы обучаемый скомбинировал для ее получения, другими словами, от каких «базовых» формул зависит формула, введенная обучаемым);

2) соответствующие «базовые» формулы ищутся в сгенерированных решениях и помечаются как использованные обучаемым при решении задачи;

3) для каждого i-го сгенерированного решения задачи вычисляется степень прогресса PR_i, равная отношению количества использованных обучаемым формул к количеству всех формул, присутствующих в решении;

4) в качестве текущего значения прогресса обучаемого выбирается максимальное значение PR_i.

При рассмотрении данной архитектуры подсистемы проверки решений ИОС может возникнуть несколько вопросов. Во-первых, вопрос о том, каким образом и кем контролируется полнота списка «базовых» формул. В качестве ответа на этот вопрос процитируем [5]: «одним из наиболее надежных считается метод постепенного пополнения и изменения базы знаний, причем поводом к изменению является каждый случай, когда

система не смогла решить задачу или решила ее неправильно». Так, например, можно в интерфейс ввода решения задачи добавить кнопку «Пожаловаться». Обучаемый нажимает на эту кнопку, если уверен в том, что система несправедливо не засчитала его шаг решения как правильный. Тогда шаг решения обучаемого временно засчитывается, а также формируется специальный отчет для преподавателя, на основе которого преподаватель может не только изменить оценку за задачу, но и принять решение о необходимости пополнения соответствующего списка «базовых» формул.

Во-вторых, может возникнуть вопрос о том, каким образом выявляется подмножество «базовых» формул, от которых зависит формула, введенная обучаемым. Этому вопросу и посвящена данная статья. В разделе 3 описывается известный алгоритм выявления зависимостей между «базовыми» формулами и формулой, введенной обучаемым, реализованный в Andes Physics Tutor, а в разделе 4 – алгоритм выявления зависимостей между формулами, предлагаемый автором.

Список «базовых» формул, представляющий известные решения приведенной выше задачи, не является полным. Это связано с тем, что при его составлении были учтены не все возможные решения этой задачи. При наличии более полного списка «базовых» формул результаты работы обоих алгоритмов выявления зависимостей будут содержать зависимости, выявленные при использовании менее полного списка. Поэтому в данной работе можно ограничиться имеющимся списком «базовых» формул.

Описываемые в статье алгоритмы имеют программную реализацию (см. [14–15]). Отметим, что для алгоритма Andes Physics Tutor необходима библиотека SymPy версии 0.7.2, а для предлагаемого алгоритма – SymPy версии 0.7.3. Стабильная версия предлагаемого алгоритма находится в ветке master, а самые последние доработки – в ветке develop.

3. Алгоритм выявления зависимостей между «базовыми» формулами и формулой, введенной обучаемым, реализованный в Andes Physics Tutor

В Andes Physics Tutor шаги решения, вводимые обучаемыми, рассматриваются с точки зрения уравнений и зависимостей между ними. Если уравнение, введенное обучаемым, является линейной комбинацией предыдущих введенных им уравнений, значит, оно является лишь повторением уже введенных шагов решения. Знание о линейной зависимости уравнения, введенного обучаемым, и некоторых уравнений из списка «базовых» для этой задачи уравнений свидетельствует о том, что обучаемый при вводе шага решения мог использовать именно эти уравнения (в общем случае может быть выявлено несколько вариантов зависимостей между формулой, введенной обучаемым, и различными подмножествами «базовых» формул).

Зависимость уравнений, в случае, когда все уравнения линейны, имеет место, если нормаль к гиперплоскости, задаваемой уравнением, введенным обучаемым, является линейной комбинацией нормалей гиперплоскостей, задаваемых остальными уравнениями. В нелинейном случае вместо коэффициентов каждого уравнения используются координаты его градиента, в котором все переменные заменены на числовые значения.

Рассмотрим этот алгоритм на примере задачи вычисления расстояния между векторами (см. раздел 2). Поскольку данным алгоритмом не предусматривается обработка формул, содержащих векторы и матрицы, то в качестве списка «базовых» уравнений возьмем формулы (1)–(3), формулы (8)–(11) и формулы

(12) $c_1 = a_1 - b_1$,

(13) $c_2 = a_2 - b_2$.

Пусть обучаемый ввел такую формулу:

(14) $p(a,b) = \sqrt{(c,c)}$.

Для того чтобы определить, какие формулы использовал обучаемый для вывода формулы (14), составим таблицу 1. Строки таблицы 1 соответствуют градиентам всех рассматрива-

емых формул в точке, координаты которой равны числовым значениям переменных из условий задачи.

Таблица 1. Градиенты «базовых» уравнений и формулы, введенной обучаемым, в точке, соответствующей решению задачи

Формула f	$grad\ f\Big\vert_{\substack{p(a,b)=\sqrt{8},(c,c)=8,\vert c\vert=\sqrt{8},c_1=-2,\\ a_1=1,b_1=3,c_2=-2,a_2=2,b_2=4}}$
$p(a,b)-\vert c\vert$	A = [1.0, 0, –1.0, 0, 0, 0, 0, 0, 0]
$c_1-a_1+b_1$	B = [0, 0, 0, 1.0, –1.0, 1.0, 0, 0, 0]
$c_2-a_2+b_2$	C = [0, 0, 0, 0, 0, 0, 1.0, –1.0, 1.0]
$\vert c\vert-\sqrt{(c,c)}$	D = [0, –0.18, 1.0, 0, 0, 0, 0, 0, 0]
$(c,c)-c_1^2-c_2^2$	E = [0, 1.0, 0, 4.0, 0, 0, 4.0, 0, 0]
a_1-1	F = [0, 0, 0, 0, 1.0, 0, 0, 0, 0]
a_2-2	G = [0, 0, 0, 0, 0, 0, 0, 1.0, 0]
b_1-3	H = [0, 0, 0, 0, 0, 1.0, 0, 0, 0]
b_2-4	I = [0, 0, 0, 0, 0, 0, 0, 0, 1.0]
$p(a,b)-\sqrt{(c,c)}$	J = [1.0, –0.18, 0, 0, 0, 0, 0, 0, 0]

Далее решается система уравнений

(15) $\alpha A+\beta B+\chi C+\delta D+\varepsilon E+\phi F+\varphi G+\gamma H+\eta Y=J$.

Решением данной системы уравнений является вектор с 9 координатами, равными 0, кроме первой и четвертой координаты (они равны 1). Это означает, что для вывода формулы (14) обучаемый использовал формулы из 1 и 4 строки таблицы 1, т.е. формулы (1) и (2).

В случае, когда при поиске зависимостей получается переопределенная система, далее путем комбинаторного перебора подмножеств уравнений этой системы генерируются квадратные системы уравнений. В случае успешного решения каждой полученной таким образом системы запоминается полученный вариант зависимости между формулами.

Если в результате работы алгоритма формируется несколько вариантов зависимостей между «базовыми» формулами и формулой, введенной обучаемым, то для выбора варианта, наиболее соответствующего мысленному пути, проделанному обучаемым, далее используются дополнительные эвристики. К сожалению, разработчики Andes Physics Tutor лишь упоминают о существовании таких эвристик и не приводят их подробного описания.

Формула (14) соответствует шагу обучаемого в наиболее благоприятной для выявления зависимостей между формулами форме, поскольку в ней ни одна переменная из условия задачи не была заменена на соответствующее числовое значение. Очевидно, что чем больше переменных заменено на числовые значения и чем более вычислительно свернутой является формула, введенная обучаемым, тем большая неопределенность имеет место при выявлении зависимостей между формулами. Посмотрим, к каким результатам приводит использование вышеописанного алгоритма в случаях, когда в формуле, введенной обучаемым, часть переменных заменена на числовые значения.

ПРИМЕР 3.1.

Пусть обучаемый ввел формулу

(16) $(c,c) = (a_1 - b_1)^2 + 4$.

Алгоритм выдаст правильный результат: формулы (3), (9), (11), (12), (13).

ПРИМЕР 3.2.

Пусть обучаемый узнал, что $(c, c) = 8$ (подобная ситуация может возникнуть в компьютерном классе после разговора между двумя сидящими рядом студентами) и ввел формулу

(17) $(c,c) = 16/2$.

Формула (17) корректна, но не может быть засчитана как шаг решения задачи. Она лишь является его имитацией. Между тем, алгоритм покажет, что при выводе формулы (17) обучаемый использовал формулы (3), (8)–(13).

ПРИМЕР 3.3.

Пусть обучаемый узнал, что $(c, c) = 8$ и (c, c) некоторым образом зависит от c_1, c_2 и ввел формулу

(18) $(c,c) = c_1 + c_2 + 12$.

Поскольку $c_1 = -2$, $c_2 = -2$, то формула корректна, хотя и не может быть засчитана как шаг решения задачи. Между тем, алгоритм покажет, что обучаемый использовал формулу (3).

Как показывают примеры 3.2–3.3, в случаях, когда в формуле, введенной обучаемым, часть переменных заменена на числовые значения, алгоритм измерения прогресса в решении, реализованный в подсистеме проверки решений Andes Physics Tutor, становится *уязвимым* для имитации правильно введенных шагов решения.

4. Предлагаемый алгоритм выявления зависимостей между «базовыми» формулами и формулой, введенной обучаемым

Осознанное решение любых учебных задач включает в себя анализ ситуации и выполнение некоторой последовательности вычислительных действий, в ходе выполнения которых обучаемый находит значения ряда величин, устанавливающих связь между искомыми и известными величинами. Будем называть *целевыми*

- величины, которые связывают искомые и известные величины;
- величины, соответствующие искомым объектам.

Так, для рассматриваемой задачи искомой величиной является $p(a,b)$, известными величинами – a_1, a_2, b_1, b_2, целевыми величинами – $p(a, b)$, $|c|$, (c, c), c, c_1, c_2.

Будем считать, что предлагаемый алгоритм работает на множестве тех формул, которые для данной задачи указал автор курса, а не на множестве всех формул в базе знаний обучающей системы. Кроме того, будем считать, что каждая «базовая» формула соответствует одному из следующих случаев:

- вычисление некоторой целевой величины (вида x = <символьное выражение>);
- формула, связывающая обозначение вектора с обозначениями его координат (вида $x = (x_1 \ldots x_n)$);
- определение известной величины (вида x = <число>).

Для рассматриваемой задачи (см. раздел 2 статьи) допустимыми «базовыми» формулами являются формулы (1)–(11). Эти ограничения позволяют снизить вычислительную сложность предлагаемого алгоритма.

В программе каждая «базовая» формула представляется посредством такой структуры данных, как «соотношение вычислимости». Этот термин, а также его содержательный смысл были задействованы из раздела работы [1], посвященного применению логических методов к интеллектуализации обучающих систем. Например, «базовая» формула (3) представляется соотношением вычислимости со следующими значениями свойств:

- известные величины (обозн. *known_variables*): c_1, c_2;
- целевая величина (обозн. *goal_variable*): (c, c);
- формула (обозн. *formula_text*): (c, c)=$c_1^2 + c_2^2$.

В разработанной программе соотношения вычислимости представляются классами со свойствами следующих типов: строка, список, элементами которого являются строки[1]. На рис. 2 представлен фрагмент кода разработанной программы, из которого можно получить информацию о том, как вносятся сведения, необходимые для автоматизированной проверки учебной задачи в ИОС.

Закрепим за каждым соотношением вычислимости идентификационный номер (в статье считается, что идентификационный номер соотношения вычислимости соответствует порядковому номеру «базовой» формулы). Для удобства записи будем обозначать *calc*(x) соотношение вычислимости с идентификационным номером x.

---

[1] *Список – абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза.*

```
def testName(self):

    #enter info about "basic" formulas for solutions generation and checking students' input
    calc_relations = []
    calc_relations.append(calc.CalcRelation('|c|',['p(a,b)'], 'p(a,b)=|c|'))
    calc_relations.append(calc.CalcRelation('p(a,b)', ['|c|'], 'p(a,b)=|c|'))
    calc_relations.append(calc.CalcRelation('|c|', ['(c,c)'], '|c|=sqrt{(c,c)}'))
    calc_relations.append(calc.CalcRelation('(c,c)', ['|c|'], '|c|=sqrt{(c,c)}'))
    calc_relations.append(calc.CalcRelation('(c,c)', ['c_1','c_2'], '(c,c)=c_1^2+c_2^2'))
    calc_relations.append(calc.CalcRelation('c', ['a','b'], 'c=a-b'))
    calc_relations.append(calc.CalcRelation('c', ['c_1', 'c_2'], 'c=[c_1;c_2]'))
    calc_relations.append(calc.CalcRelation('a', ['a_1', 'a_2'], 'a=[a_1;a_2]'))
    calc_relations.append(calc.CalcRelation('b', ['b_1', 'b_2'], 'b=[b_1;b_2]'))
    notations = ['a', 'a_1', 'a_2', 'a_3', 'a_4', 'b', 'b_1', 'b_2', 'b_3', 'b_4', 'p(a,b)', 'c', 'c_1',
    solution_point = {'a_1': 1, 'b_1': 3, 'a_2': 2, 'b_2': 4, 'c_1': -2, 'c_2': -2, '(c,c)': 8}
    vectors = ['c', 'a', 'b']
    sought_variable = 'p(a,b)'
    known_variables = ['a_1', 'a_2', 'b_1', 'b_2']
```

Рис. 2. Пример того, как вносятся сведения, необходимые для автоматизированной проверки учебной задачи в ИОС

На основе соотношений вычислимости автоматически строятся деревья И/ИЛИ (подробнее о дереве И/ИЛИ см. [4]). Для рассматриваемой задачи строится два дерева И/ИЛИ. В данном разделе опишем их внешнее представление и использование, их внутреннее представление и алгоритм построения по имеющимся соотношениям вычислимости см. в следующем разделе статьи. Деревья И/ИЛИ, построенные для рассматриваемой задачи, можно представить следующим образом (см. рис. 3). В обоих деревьях рассматриваемой учебной задачи присутствуют только И-узлы.

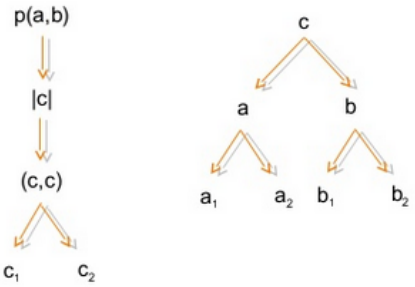

Рис. 3. Деревья И/ИЛИ, посредством которых представляется информация об известных решениях рассматриваемой учебной задачи

Узлам каждого дерева соответствуют целевые величины, листьям – известные величины или величины, являющиеся координатами некоторой целевой величины (например, c_1 и c_2 являются координатами целевой величины c). Каждое ребро дерева соединяет одну целевую величину с одной или несколькими целевыми величинами и/или известными величинами. Каждому ребру дерева соответствует одно (И-узел) или несколько соотношений вычислимости (ИЛИ-узел).

На основе И/ИЛИ деревьев генерируются все известные решения задачи. Эти решения могут состоять из нескольких частей. Для каждой части решения запоминаются не только номера соотношений вычислимости, но и целевые величины, которые вычисляются по ходу этой части решения. Для рассматриваемой задачи формируется одно решение, состоящее из двух частей:

Часть 1.

- соотношения вычислимости: 1, 2, 3;
- целевые величины: $p(a,b)$, $|c|$, (c,c).

Часть 2.

- соотношения вычислимости: 4, 6, 7;
- целевые величины: c, a, b.

Посредством F_{st} обозначается формула, которую ввел обучаемый, посредством N_r – величины в правой части этой формулы. Будем рассматривать случай, когда в F_{st} имеется только один знак «=» и в левой ее части (считаем, что F_{st} делится знаком «=» на левую и правую части) находится только одна величина. Если шаг решения обучаемого не удовлетворяет данному условию, то можно привести его к требуемой форме с помощью эвристик, задействующих библиотеку символьных вычислений.

Предлагаемый алгоритм не только выявляет зависимости между F_{st} и «базовыми» формулами, но и в некоторых случаях обнаруживает факты имитации правильно введенных шагов решения. Идея алгоритма заключается в следующем.

Шаг 1. *На основе каждого решения, имеющего часть, в списке целевых величин которой присутствует величина из левой части F_{st}, генерируется выражение. В правой части*

полученного выражения все величины, кроме тех, что присутствуют в правой части F_{st}, заменяются на числовые значения (*т.е. воспроизводится мысленный путь обучаемого – вывод введенной им формулы*).

Шаг 2. *Выражения, полученные на шаге* 1, *сравниваются с F_{st}. Если найдется хоть одно выражение e, эквивалентное F_{st}, то это означает, что обучаемый ввел правильный шаг и при этом мог использовать соотношения вычислимости из соответствующих частей того решения, на основе которого было сгенерировано e. Если не найдется ни одного такого выражения, то это означает, что обучаемый мог сымитировать шаг решения или что в базе знаний обучающей системы отсутствуют некоторые* «*базовые*» *формулы.*

Генерация выражения начинается с индекса I_0, соответствующего индексу целевой величины, совпавшей с величиной из левой части F_{st}. Во время генерации выполняются следующие операции: получение соотношений вычислимости по идентификационному номеру; обращение к свойствам соотношений вычислимости; поиск и замена строковых величин. В терминах алгоритмов поиска в пространстве состояний (англ. state space search) текущее состояние решаемой задачи генерации выражения задается с помощью следующих переменных: E^i – выражение; N^i – величины в правой части E^i; U^i – соотношения вычислимости, использованные для генерации выражений $E^1, \ldots, E^i$. Условие остановки генерации выражения таково: достигнут конец решения или $N^i \supseteq N_r$

Иногда для генерации одного выражения приходится генерировать несколько выражений, унифицировать их и, комбинируя некоторые выражения, формировать результирующее выражение (см. пример 4.1).

Перед началом работы алгоритма все обозначения задачи сортируются по длине и последовательно заменяются в F_{st}, соотношениях вычислимости и списках целевых величин частей решений на обозначения вида *x*1*y*, …, *xny*. Это позволяет не только избежать нежелательных коллизий при выполнении операций поиска и замены строк (например, может возникнуть

коллизия у обозначений $p(a, b)$, a, b), но и эффективно вставлять забытые обучаемым знаки умножения.

Напомним, что предлагаемый алгоритм запускается только в том случае, если шаг решения обучаемого прошел верификацию. Если на шаге 2 алгоритма не найдется выражения, эквивалентного F_{st}, то, если исключить версию о недостающих «базовых» формулах в базе знаний, весьма маловероятно, что обучаемый ввел без злого умысла формулу, которая все же смогла пройти верификацию, поэтому версия о неправильно введенном шаге решения не рассматривается.

Что касается известных проблем алгоритмической неразрешимости массовой задачи эквивалентности термов (равенства слов), то еще в 1970-х гг. (В.М. Глушков и др.) отмечалась полезность понятия практической разрешимости, учитывающей эвристики и вычислительную сложность задач, в сравнении с зачастую практической неразрешимостью ряда теоретически разрешимых задач. Кроме того, практический опыт разработчиков следящих ИОС, упомянутых во введении данной статьи, а также опыт автора свидетельствуют о том, что в подавляющем большинстве случаев проверок решений не слишком сложных учебных задач упрощение разности выражений с использованием достаточно развитых инструментов символьных вычислений будет успешно осуществлено.

Проиллюстрируем работу предлагаемого алгоритма на примерах.

ПРИМЕР 4.1.

Пусть обучаемый ввел формулу

(19) $(c,c) = (a_1 - b_1)^2 + (a_2 - b_2)^2$.

Алгоритмом генерируется выражение, эквивалентное формуле (19), поэтому делается вывод о том, что обучаемый ввел правильный шаг, использовав «базовые» формулы соотношений вычислимости 3, 4, 6, 7 из Части 1 и Части 2 Решения 1. Эквивалентность выражений в разработанной программе проверяется с задействованием модуля sympy.simplify библиотеки SymPy.

В ходе работы алгоритма в данном случае в том числе выполняется следующая последовательность шагов.

Шаг 1. Выясняется, что можно вычислить (c, c) с помощью Части 1 Решения 1. Генерируется выражение

$$(c,c) = c_1^2 + c_2^2 .$$

Поскольку невозможно «продвинуть» состояние решаемой задачи генерации выражения дальше, а условие остановки не выполнено, то проверяется, не являются ли c_1 и c_2 координатами некоторого вектора (напомним, что для каждой учебной задачи автором курса также задается список векторов, см. рис. 2).

Шаг 2. Поскольку c_1 и c_2 являются координатами вектора c, происходит поиск части решения, на основе которой можно вычислить c (это Часть 2). Генерируется выражение для вектора c, которое имеет вид

$$c = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad .$$

Шаг 3. С помощью модуля sympy.unify происходит сопоставление выражения, сгенерированного для нахождения вектора c и определения вектора c через его координаты, т.е. сопоставляются выражения

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \text{ и } \begin{pmatrix} \mathrm{c}_1 \\ \mathrm{c}_2 \end{pmatrix}.$$

Выясняется, что

$$c_1 = a_1 - b_1, c_2 = a_2 - b_2 .$$

Полученные замены подставляются в выражение, полученное на шаге 1. К списку номеров соотношений вычислимости, найденных на шаге 1, добавляются номера, найденные на шаге 3. Полученный таким образом список номеров соотношений вычислимости возвращается алгоритмом в качестве найденной зависимости между «базовыми» формулами и формулой, введенной обучаемым.

ПРИМЕР 4.2.

Пусть обучаемый ввел формулу

$$(c,c) = c_1 + c_2 + 12 .$$

В результате работы алгоритма будет сгенерировано выражение

$(c,c) = c_1^2 + c_2^2$.

Сгенерированное выражение окажется неэквивалентным формуле, введенной обучаемым, и шаг решения обучаемого (в отличие от алгоритма Andes Physics Tutor) будет помечен алгоритмом как подозрительный.

ПРИМЕР 4.2.

Пусть обучаемый ввел формулу

$(c,c) = 16/2$.

Будет сгенерировано выражение

$(c,c) = (1-3)^2 + (2-4)^2$.

эквивалентное формуле, введенной обучаемым – т.е. имитация правильно введенного шага будет не выявлена.

5. Алгоритм построения дерева И/ИЛИ по имеющимся соотношениям вычислимости

В разработанной программе деревья И/ИЛИ представляются посредством словарей, элементами которых являются списки списков[1]. В рассматриваемой задаче деревья И/ИЛИ имеют следующий вид:

Дерево 1.

{0: [[[[1]]]], 1: [[[[2]]]], 2: [[[[3]]]]}.

Дерево 2.

{0: [[[[4]]]], 4: [[[[6, 7]]]]}.

Ключ 0 обозначает корень дерева, остальные ключи соответствуют идентификационным номерам некоторых соотношений вычислимости. В общем случае элементы дерева выглядят

[1] *Словарь – структура данных, в которой, в отличие от массива, доступ к элементам производится по ключу. Словарь – неупорядоченное множество пар вида ключ: значение с требованием уникальности ключей в пределах одного словаря.*

следующим образом: x: [[e_1], …, [e_n]], где e_i имеет вид [[z_{11}, …, z_{1m}], …, [z_{p1}, …, z_{pr}]]. Обозначения z_{11}, …, z_{1m}, …, z_{p1}, …, z_{pr} соответствуют идентификационным номерам некоторых соотношений вычислимости.

Каждый элемент дерева отражает такую информацию: для нахождения *i*-й переменной из *calc*(*x*).*known_variables* требуется применить соотношения вычислимости z_{11}, …, z_{1m}, или …, или соотношения z_{p1}, …, z_{pr}.

Процесс построения дерева И/ИЛИ включает два основных этапа: генерация веток и формирование поддеревьев. На заключительном этапе из поддеревьев собирается требуемое дерево (путем их объединения). В разработанной программе ветки дерева И/ИЛИ представляются посредством списков, элементами которых являются целые числа. То есть ветка дерева И/ИЛИ имеет вид [*x*, *y*], где *x* и *y* – идентификационные номера некоторых соотношений вычислимости. Ветка [*x*, *y*] отражает информацию о том, что для вычисления одной из *calc*(*x*).*known_variables* нужно использовать соотношение вычислимости *calc*(*y*).

Генерация веток осуществляется с помощью структур данных *Path* со следующими свойствами:

- S^i – текущее состояние (представляет собой некоторое подмножество обозначений задачи);
- *Index* – идентификационный номер использованного соотношения вычислимости.

В начале работы алгоритма ищутся соотношения вычислимости *calc*(*x*) такие, что *calc*(*x*).*goal_variable* совпадает с искомой величиной задачи (если искомых величин несколько, то описываемый ниже алгоритм построения дерева повторяется для каждой из этих величин). На основе каждого найденного соотношения *calc*(*x*) формируется *currPaths* – набор объектов *Path*, каждому из которых в свойство S^i помещаются *calc*(*x*).*known_variables*, а в свойство *Indexes* – идентификационный номер *x*. Далее для формирования веток дерева И/ИЛИ повторяется описываемая ниже процедура.

Шаг 1. Перебираются обозначения из S^i каждого объекта *Path* из *currPaths*, для каждого такого обозначения ищутся

соотношения вычислимости *calc*(*x*), у которых *calc*(*x*).*goal_variable* совпадает с этим обозначением. Если такое соотношение находится, то

- формируется объект *newPath* с S^i := *calc*(*x*).*known_variables*, *Index* := *x*;
- формируется ветка дерева И/ИЛИ [*Path.Index*, *x*].

Шаг 2. *currPaths* полагаем равными набору объектов *newPaths*, сформированных на предыдущем шаге алгоритма на основе всех объектов *Path* из *currPaths*. Если *newPaths* не пустое множество, переходим к Шагу 1.

В ходе выполнения вышеописанной процедуры может возникнуть такая ситуация, когда объект *newPath* не сформируется (т.е. не найдется соотношение вычислимости, отвечающее требуемому условию). В этом случае, если обозначения из S^i объекта *Path* являются координатами некоторого вектора (напомним, что для каждой учебной задачи автор курса указывает список векторов и формулы, связывающие обозначение вектора с обозначениями его координат), то далее алгоритмом генерируется сообщение о том, что необходимо построить дерево И/ИЛИ для этого вектора.

Формирование поддерева И/ИЛИ осуществляется следующим образом.

1) На вход подается соотношение вычислимости *calc*(*x*).

2) Из множества веток извлекается множество идентификационных номеров соотношений вычислимости, включающее последние элементы веток, у которых первый элемент совпадает с номером рассматриваемого отношения вычислимости. Обозначим это множество *Ids*.

3) Для каждой *j*-й переменной *var* из *calc*(*x*).*known_variables* выполняется следующая процедура:

- ищутся y_k из множества *Ids* такие, что *calc*(y_k).*goal_variable* совпадает с *var*, на их основе формируются элементы $z^k = [z_1, \ldots, z_m]$, где $z_1, \ldots, z_m$ берутся из *calc*(y_k).*known_variables*;
- на основе элементов z^k формируется элемент $e_j = [[z^1], \ldots, [z^r]]$.

4) Формируется поддерево x: $[[e_1], \ldots, [e_n]]$.

6. Вычислительная сложность предлагаемого алгоритма

На данном этапе полностью оценить вычислительную сложность предлагаемого алгоритма не представляется возможным, так как этот алгоритм задействует такие сторонние алгоритмы и методы, как аппарат регулярных выражений (задействуется в некоторых случаях во время операций поиска и замены строк), алгоритм упрощения выражений, алгоритм унификации выражений.

Поскольку предполагается, что деревья И/ИЛИ, посредством которых представляется информация об известных решениях учебных задач, будут генерироваться в режиме оффлайн, т.е. на этапе настройки системы, то наиболее критическим местом алгоритма является генерация решений и поиск по ним. Соответственно, вычислительную сложность алгоритма характеризует количество решений, которые могут быть сгенерированы для учебной задачи на основе деревьев И/ИЛИ.

Вообще деревья И/ИЛИ являются удобным инструментом перечисления разнообразных информационных объектов, имеющих достаточно сложную структуру. В [2] деревья И/ИЛИ применялись для генерации вопросов и тестовых заданий в компьютерном тестировании. В [2] предлагается алгоритм подсчета вариантов в дереве И/ИЛИ в виде следующей рекурсивной функции:

$$\omega(z) = \begin{cases} \sum_{i=1}^{n} \omega(s_i^z) & \text{для ИЛИ - узла} \\ \prod_{i=1}^{n} \omega(s_i^z) & \text{, для И - узла} \\ 1 & \text{, для листа} \end{cases}$$

где z – рассматриваемый узел дерева; $\{s_i^z\}$ – множество сыновей узла z; n – количество сыновей; $\omega(z)$ – количество вариантов для узла z.

Пусть степень любого узла (т.е. количество сыновей узла) дерева И/ИЛИ не превышает m. Назовем глубиной дерева И/ИЛИ количество узлов, встречающихся по ходу наиболее длинного маршрута от корня дерева к листу (включая узел, соответствующий корню дерева). Рассмотрим деревья И/ИЛИ глубины 2.

Наиболее вычислительно сложным окажется дерево T^2_{max}, представляющее собой И-узел, сыновья которого – ИЛИ-узлы с сыновьями-листьями. У такого дерева

$\omega(z) \le m^m$.

Рассмотрим деревья И/ИЛИ глубины 3. Наиболее вычислительно сложным окажется дерево, представляющее собой И-узел, каждый сын которого – дерево вида T^2_{max}. У такого дерева

$\omega(z) \le (m^m)^m$, т.е. $\omega(z) \le m^{m^2}$.

Продолжая аналогичные рассуждения, получим, что для дерева И/ИЛИ глубины p и максимальной степенью узла m

$\omega(z) \le m^{m^{p-1}}$.

В такой постановке предлагаемый алгоритм имеет высокую вычислительную сложность, поэтому необходимо его оптимизировать. Оптимизация может заключаться в том, чтобы генерировать выражения, которые сравниваются с формулой, введенной обучаемым и оценивать прогресс в решении на основе деревьев И/ИЛИ, а не на основе решений (т.е. исключить решения, генерируемые на основе деревьев И/ИЛИ как промежуточное звено).

7. Заключение

В данной статье на примере задачи из курса линейной алгебры рассмотрен ряд случаев, возникающих при проверке решений одного класса учебных задач. Предложен алгоритм выявления зависимостей между «базовыми» формулами, посредством которых представляются сведения об известных решениях учебной задачи, и формулой, введенной обучаемым. Отметим, что идея и реализация предлагаемого алгоритма измерения прогресса в решении похожа на некоторые

идеи и техники из теории систем переписывания термов [16]. Так, например, отношения вычислимости похожи на *редукции* – бинарные отношения, с помощью которых кодируются правила преобразования термов; в ходе переписывания термов так же, как и в предлагаемом алгоритме, осуществляется преобразование (переписывание) совокупности одних величин (термов) в другие. К сожалению, напрямую аналогию провести нельзя, поскольку отличаются цели и критерии остановки преобразования.

Предлагаемый алгоритм, в отличие от известного алгоритма, реализованного в подсистеме проверки решений Andes Physics Tutor, в некоторых случаях обнаруживает факты имитации правильно введенных шагов решения, а также способен обрабатывать формулы, содержащие векторы и матрицы. Предлагаемый алгоритм имеет свои ограничения (см. раздел 2) и обладает значительно большей вычислительной сложностью, чем известный алгоритм.

Дальнейшее развитие исследований предполагает:

1) оптимизацию предлагаемого алгоритма;

2) разработку дополнительных эвристик для обработки часто встречающихся на практике отклоняющихся от рекомендуемых правил ввода шагов решений, таких как

$(1-3)^2 + (2-4)^2 = 8$,

$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1-2 \\ 3-4 \end{pmatrix}$,

и т.д.;

3) уточнение класса задач, к которым применим предложенный алгоритм (с помощью [6]).

Как показывает ряд приведенных выше примеров, автоматизированная проверка решений, введенных в достаточно свободной форме, возможна, но имеет свои ограничения. Чем меньше переменных и больше числовых значений в формуле, введенной обучаемым, тем большая неопределенность возникает при ее анализе. Предложенные эвристики помогают в ряде случаев, но не во всех, поэтому при проектировании следящей ИОС желательно предусматривать возможность для обучаемого

временно засчитать нераспознанный системой шаг решения и вывести его в специальном отчете для преподавателя.

Литература

1. ВАСИЛЬЕВ С.Н. САБИТОВ Ш.Р., СМИРНОВА Н.В. и др. *Методическое обеспечение логических и оптимизационных задач в интеллектуальной обучающей системе «Волга»* // Аналитическая механика, устойчивость и управление: Труды X Международной Четаевской конференции. – Казань: изд-во Казан. гос. техн. ун-та, 2012. – Т. 4. – С. 324.
2. КРУЧИНИН В.В. *Алгоритмы и перечислительные свойства деревьев И-ИЛИ* [Электронный ресурс]. – URL: http://fdo.tusur.ru/articles/docs/25fc9f27315b76d32d7594de7811a1a7.pdf (дата обращения: 14.03.2014).
3. ЛЕВИНСКАЯ М.А. *Продукционная модель интерактивной компоненты обучающей системы* // Сб. науч. тр. «Математика. Компьютер. Образование». – Ижевск, Научно-издательский центр «Регулярная и хаотическая динамика», 2003. – Т. 1. – С. 81–92.
4. НЕЛЬСОН Н. *Искусственный интеллект.* – М.: Мир, 1973. – С. 91–128.
5. НОВИКОВ Ф.А. *Искусственный интеллект: представление знаний и методы поиска решений: Учеб. пособие*. – СПб.: Изд-во Политехн. ун-та, 2010. – 240 с.
6. ПОДКОЛЗИН А.С. *Компьютерное моделирование процессов решения математических задач*. *Т. 1–2*. – М.: Изд-во ЦПИ при мех.-мат. ф-те МГУ, 2001. – 235 с.
7. ФРИДМАН Л.М. *Логико-педагогический анализ школьных учебных задач*. – М.: Педагогика, 1977. – 208 с.
8. ČERTIK O., MEURER A. *SymPy – a Python library for Symbolic Mathematics* [Электронный ресурс]. – URL: http://sympy.org/ru/index.html (дата обращения: 14.03.2014).
9. MELIS E., SIEKMANN J. *ActiveMath: an Intelligent Tutoring System for Mathematics* // Artificial Intelligence and Soft Computing – ISAISC 2004. – Springer, Berlin – Heidelberg, 2004. – Vol. 3070. – P. 91–101.

10. *Moodle Mathematics* [Электронный ресурс]. – URL: http://docs.moodle.org/24/en/Mathematics (дата обращения: 14.03.2014).
11. SANGWIN C.J. *Assessing Elementary Algebra with STACK* // International Journal of Mathematical Education in Science and Technology. – 2008. – Vol. 38, №8. – P. 987–1002.
12. SANGWIN C.J. *Automating the marking of core calculus and algebra: eight years on*. [Электронный ресурс]. – URL: http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/2009-12-mmg.pdf (дата обращения: 14.03.2014).
13. SHAPIRO J.A. *An Algebra Subsystem for Diagnosing Students' Input in a Physics Tutoring System* // International Journal of Artificial Intelligence in Education. –2005. – №15. – P. 205–228.
14. SMIRNOVA N.V. *A checking students' input subsystem for model-tracing intelligent tutoring systems* – [Электронный ресурс]. – URL: https://github.com/indra-uolles/solution_tracer (дата обращения: 14.03.2014).
15. SMIRNOVA N.V. *Andes dependencies checking algorithm* [Электронный ресурс]. – URL: https://github.com/indra-uolles/Andes_dependencies_algo (дата обращения: 14.03.2014).
16. TERESE *Term Rewriting Systems*. – *Cambridge Tracts in Theoretical Computer Science*. – Cambridge University Press, 2003. – 908 p.
17. VANLEHN K., LYNCH C., SCHULZE K. AND OTHER. *The Andes Physics Tutoring System: Lessons Learned* // International Journal of Artificial Intelligence in Education. –2005. – Vol. 15, №3. – P. 147–204.

TOWARDS DIAGNOSING STUDENTS' INPUT IN MATHEMATICAL TASKS OF PARTICULAR TYPE FOR USE IN MODEL-TRACING INTELLIGENT TUTORING SYSTEMS

Natalia Smirnova, Institute of Control Sciences of RAS, Moscow, researcher (smirnovanatalia2008@gmail.com).

Abstract: *The paper is dedicated to computer-aided assessment of multi-step problems provided by interactive model-tracing intelligent tutoring systems. An approach is suggested for determining which formulas could student's formula be derived from. In some cases, the suggested approach can discover whether student's step is an imitation. The suggested approach is a part of the algorithm which calculates student's solution progress.*

Keywords: model-tracing intelligent tutoring systems, tutoring systems, computer-aided assessment, and/or graph, symbolic calculations

Статья представлена к публикации членом редакционной коллегии М.В. Губко

Поступила в редакцию 27.08.2013.
Опубликована 31.03.2014.