УДК 004.58 doi: 10.26102/2310-6018/2019.24.1.043
ВАК 05.13.18

Т.Е. Есин, И.Н. Глухих

АВТОМАТИЗАЦИЯ ПРЕДОСТАВЛЕНИЯ ПЕРСОНАЛИЗИРОВАННОЙ ОБРАТНОЙ СВЯЗИ НА КУРСАХ ИЗУЧЕНИЯ ПРОГРАММИРОВАНИЯ

ФГАОУ ВО «Тюменский государственный университет», Тюмень, Россия

Использование автоматической обратной связи может существенно повысить успешность начинающих изучать программирование, в особенности на тех, кому приходится учиться внутри большой группы, а время преподавателя ограничено. В статье предложен подход к созданию автоматической обратной связи, основанный на предыдущих решениях. Подход заключается в формировании пространства программных решений – взвешенного графа. Узлами в графе является программный код, вес ребра – количество изменений и действия, которые нужно выполнить, чтобы перейти из одного состояния в другое. Для уменьшения количества уникальных решений проводится нормализация исходного кода с помощью ряда преобразований и построения абстрактного синтаксического дерева. Обратная связь – подсказка следующего шага, которую можно сгенерировать после того, как новое решение будет добавлено в существующий граф и выявлен путь, приводящий к более правильному состоянию. Таким образом, с помощью обратной связи можно достигнуть верного решения. Использование пространства решений также позволяет узнать, какие решения наиболее распространены, какие ошибки возникают и какие пути их исправления предпочитают учащиеся. Поскольку такой подход основан исключительно на данных, от преподавателя не требуется значительного взаимодействия, что делает его масштабируемым и адаптируемым.

Ключевые слова: интеллектуальные образовательные системы, курсы программирования, обратная связь, анализ образовательных данных, learning analytics

Введение. В педагогике обратная связь является одним из самых важных инструментов, поддерживающих обучение. В некоторых случаях бывает достаточно сказать ученику, где он ошибся и что пошло не так, в других – лучше рассказать более детально о том, почему его решение является неверным, направляя подопечного и позволяя ему самому найти путь к исправлению ошибки. Использование второго подхода может быть эффективным в случаях, когда путь к решению является сложным, а в качестве подсказки можно указать на отдельные проблемные части. Этот путь, оказывается, намного более полезным, ведь у ученика остаются шансы получить самостоятельный результат. Стоит отметить, что

использование такого подхода более трудоемко и требует большего времени.

При изучении информатики в школе и на курсах программирования такая обратная связь в некоторой мере уже существует. На базовом уровне учащиеся могут видеть, верен ли их синтаксис на основе полученной обратной связи от компилятора – сообщении об ошибке. Что касается реакции на семантическую составляющую программного решения, то в этом наиболее преуспевают те преподаватели, которым удалось внедрить использование систем тестирования решений. Тестирующие системы также дают информацию о том, насколько успешно программа справляется с поставленной задачей. Тем не менее, такая обратная связь, оказывается, весьма ограниченной: сообщения компилятора вряд ли смогут помочь, если программа синтаксически правильна; автоматическая оценка, как правило, основана на тестовых входных данных, прохождение которых не обеспечивает полной картины успешности решения. К сожалению, время преподавателя ограничено как в аудитории, так и вне академических часов. Когда количество одновременно изучающих программирование учеников увеличивается, становится практически невозможно обеспечить своевременный и полезный отклик сразу всем нуждающимся, уже не говоря о преподавании на массовых открытых онлайн-курсах.

Учитывая вышеописанную ситуацию, разработка метода автоматической генерации обратной связи, которая зависит от содержания программного решения становится необходимостью. Метод может быть востребован в процессе преподавания начальных курсов программирования. Качество автоматической обратной связи будет только увеличиваться от количества изучающих программирование, и, теоретически, такая система может справиться с большей частью ситуаций, в которых оказываются ученики в процессе поиска верного решения. После введения такой системы обратной связи у преподавателя освободится некоторое количество времени, которое он сможет посвятить тем ученикам, которые затрудняются больше всего.

Конечно, такой подход проще предположить, чем реализовать, так как решения задач разнообразны как по стилю написания, так и по используемому алгоритму решения. Системе предоставления обратной связи потребуется определить, насколько далеко учащийся продвинулся в решении задачи, что именно не так с текущим решением, а также как от него перейти к конечному и правильному результату.

В данной статье предложен метод создания автоматической обратной связи, использующий предыдущие ученические решения. Используя эти

данные, можно сказать, какие решения наиболее распространены, какие ошибки возникают и какие пути учащиеся выбирают чаще всего при исправлении этих ошибок. Поскольку такой подход основан исключительно на данных, от преподавателя не требуется значительного взаимодействия с системой, что делает его легко масштабируемым и адаптируемым.

Материалы и методы. Метод предоставления обратной связи опирается на использование пространства программных решений. Пространство решений — это граф, показывающий все возможные пути, по которым может следовать ученик, чтобы пройти путь от считывания исходных данных до вывода правильного решения. Узлы графа являются промежуточными решениями, а ребра – действиями, используемыми для перехода из одного состояния решения в другое.

Такие пространства решений могут быть построены путем извлечения попыток отправки задачи учениками и помещения их в существующий граф в виде цепи – от начала к полному решению задачи. Идентичные верные решения могут быть объединены, они позволят определить те места, где у учащегося есть множественный выбор следующего шага.

Теоретически, пространство решений может стать бесконечно большим, особенно если рассматривать пути, которые не приводят к правильному решению. На практике, есть конечное количество стандартных путей, которые обычно выбирают ученики для решения задачи. Кроме того, в граф должен включаться путь, являющийся авторским решением задачи, путь, предпочитаемый преподавателем, и пути, включающие в себя любые общие заблуждения. Если системе удалость понять, что ученик идет по одному из известных путей (а также момент, когда он перестал ему следовать), тогда можно предоставить наиболее персонализированную обратную связь по его работе.

При детальном рассмотрении прогресса в решении задачи, нужно решить, как часто создавать «контрольные точки» – узлы графа. Уровень детализации при этом может быть как очень высокий – изменение отдельных символов, так и очень низкий – исключительно точки сохранения и компиляции программы. В рамках данной статьи проведена экспериментальная работа с использованием низкого уровня детализации контрольных точек. Благодаря этому, можно точно сказать, что в эти моменты ученик сознательно переходит от одного состояния программы к другому. Каждый раз при запуске программы ученик останавливается в написании кода, уделяя больше внимания тому, как результат зависит от

входных данных. Разумеется, использование такого подхода не может полностью передать тот объем работы, который выполняет учащийся: не учитывается работа вне среды программирования – записи в тетради и обсуждения с однокурсниками или преподавателем.

Создать описанное пространство решений не трудно, но сделать его пригодным для использования – гораздо более сложная задача. Решения отличаются, во-первых, стилистически: названиями переменных, использованием пробелов и отступов; во-вторых, есть множество способов записать один и тот же алгоритм. Как правило, преподаватели и не хотят видеть один и тот же код, отправленный разными учениками – это говорит о копировании решения. Но пространство решений окажется бесполезным, если все пути будут абсолютно разными. По этой причине, размер пространства нужно уменьшать, объединяя семантически эквивалентные состояния.

Существуют различные техники для уменьшения размера пространства программных решений. В работе [1] (Менцель В., 2007) состояния решенной задачи представляются с помощью множества ограничений, в другой работе используются представления графа в виде матрицы смежности для очистки лишних данных [2], также используются нормализующие преобразования исходного кода для упрощения программных решений [3][4]. В рамках данного исследования решения также будут упрощаться при помощи преобразования исходного кода для получения канонического вида. Нормализующие преобразования позволят упростить код, анонимизировать его и упорядочить синтаксис, не затрагивая при этом семантическую составляющую, переводя каждое состояние программы в *нормальный* вид. Во всех таких преобразованиях используются абстрактные синтаксические деревья (далее — АСД), являющиеся отличным инструментом для разбора программы на части. Если две разные программы преобразуются к одной и той же канонической форме, значит такие программы семантически эквивалентны, таким образом, этот метод можно использовать для уменьшения размера пространства решений.

Результаты и их обсуждение.

Пример 1. Рассмотрим простую задачу из начального курса программирования. Программа принимает целое число от 0 до 51, где каждое число соответствует игральной карте. На выходе требуется получить строковое представление данной карты. Четверка бубен соответствует числу 15, поскольку трефы по условию идут раньше, значит

программа должна вернуть представление «4Б», если считать, что значения карт начинаются с двойки. Эта задача проверяет навыки использования операций взятия остатка от деления и целочисленного деления. Одно из неправильных решений приведено на Рисунке 1.

```
def intToCard(value):
        courtVal = value % 13
        court = "234567890ВДКТ"
        suitVal = (value - courtVal) % 4
        suit = "ТБЧП"
        return

   court[courtVal]+suit[suitVal]
```

Рисунок 1 – Попытка учащегося решить задачу про игральные карты

Чтобы нормализировать программу этого ученика, в первую очередь нужно построить АСД из программного кода (Рисунок 2). Все имена переменных при этом анонимизируются. В данном примере переменная *value* станет *v0, courtVal* примет имя *v1* и т.д. Затем запускаются нормализирующие преобразования, чтобы понять, какие из них будут иметь эффект; в данном случае, единственное возможное преобразование – распространение копий (от англ. copy propagation). Это преобразование сокращает список из пяти операторов в один *return*. Часть полученного преобразования можно видеть на Рисунке. Получившееся дерево меньше исходного, но на работе программы это не отражается.

Данный метод был протестирован на наборе данных итоговых решений задач курса «Основы программирования на языке Python». Метод, оказывается, достаточно эффективен: размер пространства решений уменьшается примерно на 50% для задачи средней сложности.

Однако, несмотря на преобразования, в пространстве решений все равно появляется множество уникальных решений. Как правило, это происходит по причине того, что ученики использовали нетривиальные конструкции, нашли необычное решение или сделали неожиданные ошибки. Необычные способы решения добавляют дополнительную сложность в исследование, так как с увеличением количества способов уменьшается вероятность того, что новое решение сможет найти свое место в уже существующем пространстве.

После того, как пространство решений достаточно заполнилось, можно начать генерировать обратную связь. Принятый подход основан на

работе «*Hint Factory*» (Барнес Т., Стэмпер Д., 2008) [5]. Авторы улучшили курс компьютерной логики, в который были внедрены подсказки, наталкивающие на продвижение в решении. В Hint Factory каждый узел пространства – текущее состояние решения, а каждое ребро – следующий логический вентиль, приводящий к итоговому правильному решению. В системе использовался Марковский процесс принятия решений для определения того, как руководить учащимся, находящимся в той или иной ситуации.

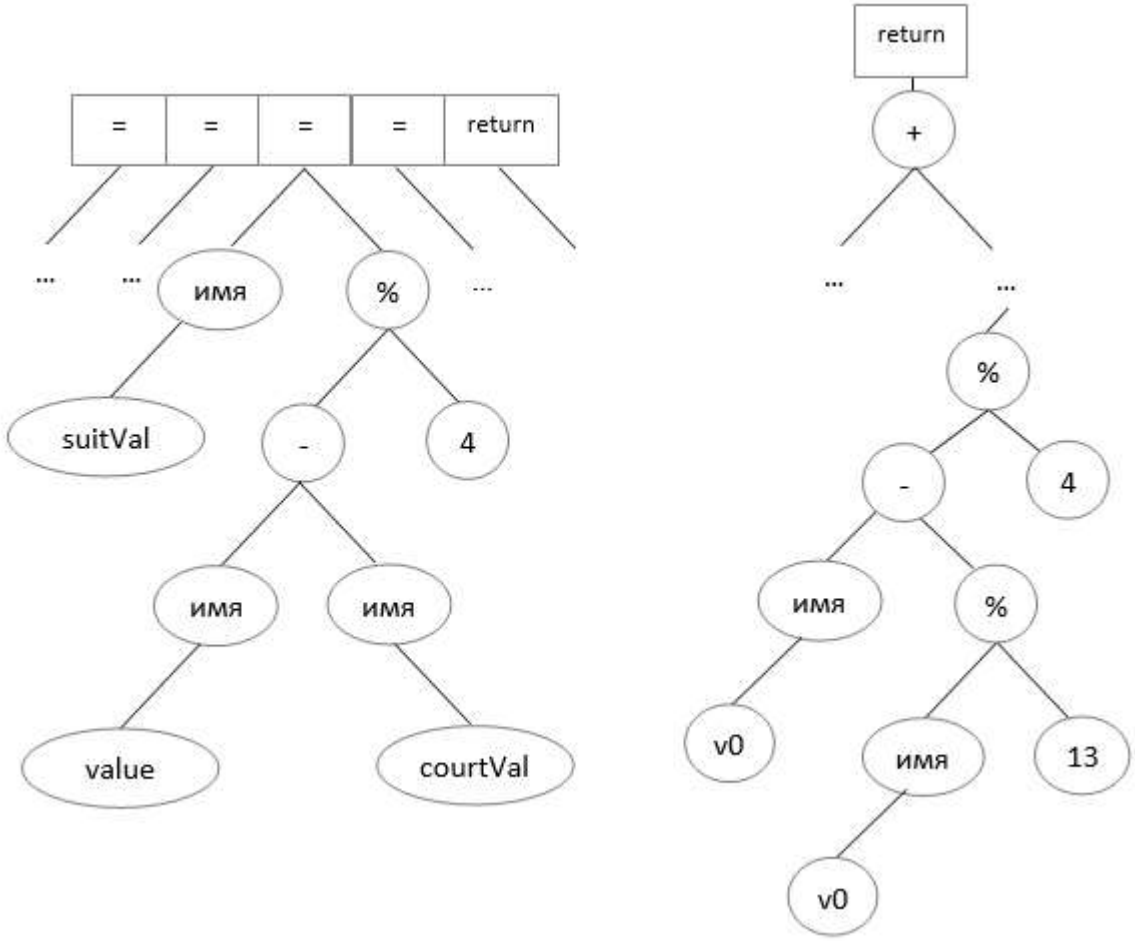

Рисунок 2 – Частичные АСД программы учащегося: до и после нормализации

Хотя текущее исследование во многом заимствует идеи из Hint Factory и других исследований, проблема рассматривается под другим углом: производится попытка найти путь ближайшему промежуточному решению, а не способ попасть из текущего состояния в окончательное решение.

Порядок состояний программного решения

После уменьшения размера пространства решений все еще остается актуальной задача, связанная с процессом составления стратегии движения учащегося по пространству. В пространстве находится множество разных решений, которые близки, но не связаны ранее найденными путями. Требуется определить, из какого из близких решений ученик с большей вероятностью придет к окончательному.

Очевидно, что один из способов выбрать в какое состояние лучше перейти ученику, это посчитать расстояния в дереве для определения, какое количество изменений нужно сделать. Тем не менее, эта метрика не особо эффективна на практике; вес редактирования трудно определить, что делает сравнение нетривиальным. Вместо этого можно использовать расстояние Левенштейна для определения, являются ли две программы близкими друг к другу. Далее, расстояния между состояниями необходимо нормализовать, высчитав процент сходства. Это необходимо, чтобы более короткие программы не имели преимущества перед более длинными, а результаты можно было сравнить.

Алгоритм нахождения пути рекурсивный – лучший путь от A до B будет лучшим элементом из множества путей S, где S – все пути через промежуточные пункты. Пути, которые требуют меньше промежуточных шагов, будут предпочтительными, т.к. требуют от ученика меньше изменений. При этом, также нужно рассмотреть расстояния между этими состояниями.

Тестирующие последовательности программы применяются к каждому состоянию для оценки его корректности. Пути, постепенно увеличивающие количество проходимых тестов, могут считаться более выгодными, чем пути, в которых проходимость тестов не является постоянно возрастающей.

Пример 2. Ранее уже была найдена каноническая форма решения, пометим ее как #22. На Рисунке 3 не отображены промежуточные решения, так как для простоты был использован набор данных из окончательных решений. Для каждой пары из них (которые не обязательно являются правильными) было рассчитано расстояние Левенштейна (показаны состояния, имеющие сходство 90% или выше).

Можно заметить, что состояние #22 – неверное решение – соединяется с тремя возможными состояниями: #4, #34 и #37. Состояние #34 также не является корректным, поэтому можно предложить ученику одно из состояний #37 или #4, т.к. у них одинаковое редакционное расстояние с состоянием #22. Числа на узлах графа обозначают количество учащихся, выбравших такое решение. Становится очевидно, что состояние

#37 более предпочтительно, ученика следует направить с помощью обратной связи к этому состоянию.

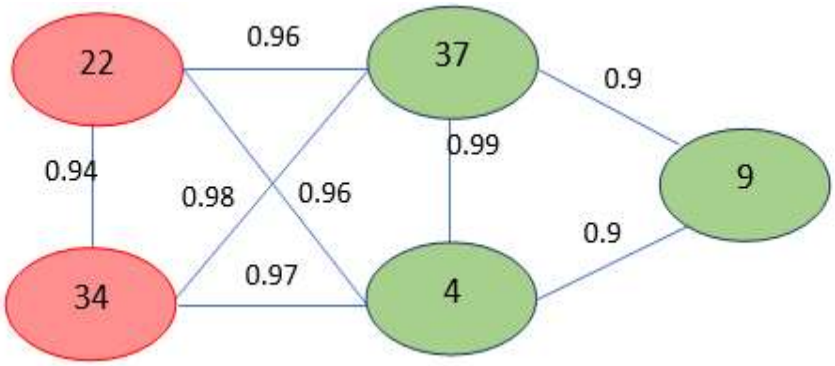

Рисунок 3 – Граф, отражающий соседние состояния от #22

Вычисление расстояния между состояниями программы

Далее, на основе пространства решений нужно сформировать и предоставить обратную связь ученику. Обратная связь представляет собой список действий, необходимых для перехода из одного состояния в другое. Для четко сформулированной задачи, эти действия как правило просты, но их сложность для ученика увеличивается с увеличением количества способов решения задачи.

Ранее в задаче было использована дистанция редактирования строки для определения схожести программ. В более сложных задачах потребуется определять необходимые изменения в дереве, чтобы перейти из одного представления в другое, соответствующее более правильному решению. Задача сравнения обычных деревьев не очень сложна, но для абстрактных синтаксических деревьев этот процесс осложняется тем, что некоторые узлы могут содержат целые списки (например, тело цикла и условного оператора).

После нахождения требуемых изменений, можно использовать информацию об этом для генерации обратной связи. В интеллектуальных обучающих системах обычно используются три уровня подсказок: местоположение ошибки, оператор или операция, являющаяся ошибочной, и, в последнюю очередь, что нужно сделать, чтобы преодолеть ошибку.

Пример 3. Для генерации сообщения для пользователя в примере с колодой карт, найдем изменения в дереве, необходимые для перехода из состояния #22 в состояние #37. Для этого нужно изменить программу в двух частях: сделать одно элементарное изменение и одно более сложное (Рисунок 4). Первое связано с опечаткой в строке с номиналами карт (буква

О вместо 0, обозначающего карту «10»). Сообщение обратной связи может выглядеть следующим образом: в операторе *return* строка «*23456789ОВДКТ*» должна быть заменена на «*234567890ВДКТ*».

Вторая ошибка связана с неправильным представлением того, как найти индекс значения масти. Так как это кусочно-заданная функция**,** ученику необходимо использовать целочисленное деление, чтобы получить верное значение. Хотя код ученика работает правильно (при вычислении остатка от деления на 4 чисел 0, 13, 26 и 39 возвращают нужные значения), это означает, что он не усвоил работу оператора целочисленного деления.

```
def intToCard(v0):
    return "23456789ОВДКТ"[(v0 % 13)] +
           "ТБЧП"[((v0 - (v0 % 13)) % 4)])
```


```
def intToCard(v0):
    return "234567890ВДКТ"[(v0 % 13)] +
           "ТБЧП"[((v0 - (v0 % 13)) // 13)])
```

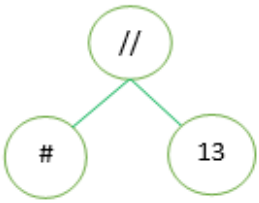

Рисунок 4 – Различие между состояниями в исходном коде и в виде дерева

Генерируя сообщение для учащегося, можно сообщить о том, что левую часть можно оставить без изменений: в правой части оператора *return*, используйте операцию // вместо %. Дальнейшие инструкции могут быть предоставлены по требованию.

Возврат от канонической формы к оригинальному коду

Все преобразования, сделанные для получения состояния #22, привели к тому, что код ученика свернулся в одно выражение возврата значения из функции. Для того, чтобы обратить изменения, необходимо сопоставить переменные, на которые производится замена с тем, как они назывались изначально. Затем нужно изучить строки, где производилось присвоение значений переменным, чтобы найти исходное местоположение, в котором использовалось выражение (Рисунок 5).

```
def intToCard(v0):
    return "23456789ОВДКТ"[(v0 % 13)] +
           "ТБЧП"[((v0 - (v0 % 13)) % 4)])
```

```
def intToCard(value):
    courtVal = value % 13
    court = "23456789ОВДКТ"
    suitVal = (value - courtVal) % 4
    suit = "ТБЧП"
    return court+suit
```

Рисунок 5 – Сравнение канонической и оригинальной программы

После нахождения номеров оригинальных строк, можно говорить о более персонализированную обратную связь. Например, первое сообщение может принять вид «Во 2-й строке выражение «234567890ВДКТ» должно быть записано как «23456789ОВДКТ». Второе так же укажет на строку, изменение в которой необходимо произвести: «В 4-й строке используйте // вместо %».

Заключение. Описанный подход использует концепцию пространств решений для определения того, на какой стадии написания программы находится ученик и предоставления обратной связи, позволяющей приблизиться к правильному решению. Использование пространства программных решений было протестировано в ходе работы над диссертацией, тем не менее, работа над созданием персонализированных и действительно полезных сообщений продолжается. В дальнейшей работе также будет определено, как часто система будет способна предоставить подобную обратную связь, как она влияет на самостоятельность ученика и насколько эффективно использование такой системы в качестве поддержки курса программирования.

ЛИТЕРАТУРА

1. Wolfgang Menzel. «Using constraint-based modelling to describe the solution space of ill-defined problems in logic programming». Advances in Web Based LearningICWL 2007. Springer Berlin Heidelberg, 2008. 367-379.
2. Jin, Wei «Program representation for automatic hint generation for a data-driven novice programming tutor». Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2012.
3. Xu, Songwen, Yam San Chee. «Transformation-based diagnosis of student programs for programming tutoring systems». Software Engineering, 29.4 (2003): 360-384.
4. Jim Reye. «Design of a knowledge base to teach programming» Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2012.
5. Barnes Tiffany and John Stamper. «Toward automatic hint generation for logic proof tutoring using historical student data». Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2008.
6. Алексеев Ю.Е. Автоматизация тестирования студенческих программ / Ю.Е. Алексеев, А.В.Куров // Инженерный журнал: наука и инновации. 2013. вып. 6 (18). URL: http://engjournal.ru/catalog/it/hidden/768.html
7. Веретенников М.В. Автоматизация проверки компьютерных программ в технических дисциплинах: Сб.науч.тр. «Дистанционные

образовательные технологии. Пути реализации». Вып. 1 – Томск: Изд-во ТУСУР, 2004 г. – 130 с., с.38-47
8. Лаптев В.В. Метод оценивания умения и навыков при обучении программированию // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. Научный журнал, № 2 / 2013. – Астрахань: Издательство АГТУ, 2013 г. – 218 с., с.194-201
9. Латыпова В.А. Методики проверки работ со сложным результатом в условиях смешанного и дистанционного автоматизированного обучения // Интернет-журнал «НАУКОВЕДЕНИЕ» Том 7, №3 (2015) http://naukovedenie.ru/PDF/170TVN315.pdf (доступ свободный).
10. Михеев И.В. Программная реализация модуля динамического тестирования учебных программ // Вестник саратовского государственного технического университета. Издательство: Саратовский государственный технический университет имени Гагарина Ю.А. № 1(79), 2015, с.113-117.

T.E.Esin, I.N.Glukhikh

AUTOMATION OF PERSONALIZED FEEDBACK IN THE PROGRAMMING STUDIES COURSES

Tyumen State University, Tyumen, Russia

The use of automatic feedback can significantly increase the success of beginners in programming, especially for those who have to study inside a large group, and the teacher's time is limited. The article proposes an approach to the creation of automatic feedback based on previous solutions. The approach is to form a solution space of programs – weighted graph. The nodes in the graph are the program code; the edge weight is the number of changes and actions that need to be performed in order to go from one state to another. To reduce the number of unique solutions, the source code is normalized using a number of transformations and the construction of an abstract syntax tree. Feedback is a hint of the next step, which can be generated after a new solution is added to an existing graph and the path leading to a more correct state is identified. Thus, with the help of feedback you can reach the right decision. Using the solution space also allows you to find out which solutions are most common, which errors occur and which ways they can be corrected students prefer. Since this approach is based solely on data, the teacher does not need significant interaction with, which makes it scalable and adaptable.

Keywords: intelligent tutoring system, programming courses, automatic feedback, educational data mining, learning analytics

REFERENCES

1. Wolfgang Menzel. «Using constraint-based modelling to describe the solution space of ill-defined problems in logic programming». Advances in Web Based LearningICWL 2007. Springer Berlin Heidelberg, 2008. 367-379.
2. Jin, Wei «Program representation for automatic hint generation for a data-driven novice programming tutor». Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2012.
3. Xu, Songwen, Yam San Chee. «Transformation-based diagnosis of student programs for programming tutoring systems». Software Engineering, 29.4 (2003): 360-384.
4. Jim Reye. «Design of a knowledge base to teach programming» Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2012.
5. Barnes Tiffany and John Stamper. «Toward automatic hint generation for logic proof tutoring using historical student data». Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2008.
6. Alexeev Y.E. Automatation testing student programs / Y.E. Alexeev, A.V. Kurov // Engineering journal: science and innovations. 2013. – No. 6 (18). URL: http://engjournal.ru/catalog/it/hidden/768.html
7. Veretennikov M.V. Automatation testing computer programs in technical disciplines: «Дистанционные educational technologies. Ways of implementation». No. 1 – Tomsk: TUSUR publishing house, 2004 г. – 130 с., с.38-47
8. Laptev V.V. Method of assessing skills and abilities in teaching programming // Bulletin of Astrakhan State Technical University. Series: Management, Computer Engineering, and Computer Science. Scientific Journal, № 2 / 2013. - Astrakhan: ASTU Publishing House, 2013 - 218 p., P. 194-201
9. Latypova V.A. Methods of testing works with a complex result in the conditions of mixed and remote automated training // Internet magazine "SCIENCE" Vol. 7, No. 3 (2015) http://naukovedenie.ru/PDF/170TVN315.pdf (free access).
10. Mikheev I.V. Software implementation of the dynamic testing curriculum module // Bulletin of the Saratov State Technical University. Publisher: Saratov State Technical University named after Gagarin Yu.A. No. 1 (79), 2015, pp.113-117