DOI 10.37882/2223-2966.2023.10.36

УСОВЕРШЕНСТВОВАННЫЙ АЛГОРИТМ ПРЕОБРАЗОВАНИЯ ПРОГРАММНОГО КОДА В СЕМАНТИЧЕСКУЮ СЕТЬ ДЛЯ ОЦЕНКИ ЗНАНИЙ ШКОЛЬНИКОВ В ПРОЦЕССЕ ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ

AN IMPROVED ALGORITHM FOR CONVERTING PROGRAM CODE INTO A SEMANTIC NETWORK FOR ASSESSING STUDENTS' KNOWLEDGE IN THE PROCESS OF LEARNING PROGRAMMING

A. Fedorov
E. Avksentieva

Summary. The article presents an improved algorithm for converting a program year into a semantic network for its further machine analysis and automation of the process of assessing students' knowledge in the process of learning programming. Variants of the representation of the resulting semantic network are considered.

Keywords: semantic network, e-learning, automation, matrix representation.

Федоров Александр Сергеевич
Аспирант, Университет ИТМО
comrade_1997@mail.ru
Авксентьева Елена Юрьевна
к.п.н., доцент, Университет ИТМО
avksentievaelena@rambler.ru

Аннотация. В статье представлен улучшенный алгоритм преобразования программного года в семантическую сеть для его дальнейшего машинного анализа и автоматизации процесса оценки знаний школьников в процессе обучения программированию. Рассмотрены варианты представления получаемой семантической сети.

Ключевые слова: семантическая сеть, электронное обучение, автоматизация, матричное представление.

Введение

Инновационные процессы в современной общеобразовательной школе актуализируют проблему эффективного личностного развития не только в рамках совершенствования методики обучения отдельным дисциплинам учебного плана, но и развития общеобразовательных учреждений в целом. В результате возникли объективные предпосылки для выбора обучающимися индивидуально-образовательных траекторий, которые бы наиболее полно отвечали их личностным потребностям и устремлениям. Однако обучающиеся испытывают существенные трудности в выборе образовательной траектории и далеко не всегда ощущают себя ответственными за сделанный выбор, за свой личностный рост. В качестве одного из путей решения поставленных задач, можно рассматривать автоматизированное проектирование индивидуальной-образовательной траектории на основе знаний обучающихся [1, 2].

В контексте обучения программированию данная проблема усложняется тем, что ученику необходимо оценить качество своих программ. Данные программы могут выводить верные значения на тестовых данных, но при этом содержать критические структурные изъяны. Таким образом возникает задача автоматического оценивания знаний обучающихся и на этой основе построения их индивидуальных образовательных траекторий [1, 2]. Данная тема рассматривалась в работе «Метод преобразования семантической сети для автоматизации оценивания решения задач по программированию в процессе электронного обучения» [3]. Статья рассматривает возможность представления программного кода в виде семантической сети для последующего машинного анализа. Предлагаемый алгоритм построения семантической сети можно условно поделить на две составляющие, а именно само построение сети и преобразование сети в массив чисел для последующей обработки.

Во время работы алгоритма исследуемая программа разбивается на логические блоки и далее на слова из заранее заданного тезауруса языка программирования. Полученные слова становятся вершинами семантической сети, а отношения, в которые они вступают друг с другом становятся рёбрами создаваемого графа. В качестве множества допустимых отношений используются как традиционные понятия из области семантических сетей (ISA, AKO, has_part и др.), так и чисто языковые отношения (return, has_type, assignment и т.п.).

Далее полученный граф преобразуется в множество точек N-мерного пространства. Для этого каждой вер-

шине ставится в соответствие массив пар <отношение (ребро)>-<ближайшее понятие (вершина)>.

Поставим в соответствие всем узлам графа некоторое численное значение, характеризующее их близость к другим понятиям тезауруса. Например, все действия с переменными имеют координаты в окрестности числа 100, а все вызовы функций — координаты в окрестности числа 1 000. Тогда полученный массив пар можно представить как словарь, где индексом будет являться тип отношения, а значением ячейки — численная характеристика ближайшей вершины, до которой можно добраться, используя данное отношение. Если при этом каждый раз получать указанные массивы для всех допустимых отношений в графе, то можно говорить о множестве точек N-мерного пространства т.к. все вершины будут иметь одинаковое количество координат. Однако, отметим недостатки предложенного способа построения семантической сети и её представления:

- Семантическая сеть использует как понятия из области языка программирования, так и понятия из области естественных языков: используемые понятия не являются четко определенными, а также обоснованными.
- В статье не формализован сам алгоритм построения семантической сети.
- Численные значения ставятся в соответствие узлам графа без должного обоснования.
- Получаемое представление графа может терять в своей точности, например, при наличии нескольких равноудаленных от текущей вершины узлов с одинаковыми рёбрами

Учитывая указанные проблемы, настоящая работа ставит своей целью усовершенствование алгоритма преобразования программного кода в семантическую сеть и методов её представления для последующей машинной обработки, реализованной на языке C++.

Представление программного кода в виде семантической сети

На основе работ [4–5] можно выделить три этапа построения семантической сети:

1. Определить абстрактные объекты и понятия предметной области, необходимые для решения поставленной задачи. Оформить их в виде вершин.
2. Задать свойства для выделенных вершин, оформив их в виде вершин, связанных с исходными вершинами атрибутивными отношениями.
3. Задать связи между вершинами, используя функциональные, пространственные, количественные, логические, временные, атрибутивные отношения.

Выделим основные принципы построения семантической сети на основе программного кода:

1. Предварительно производится унификация программного кода: циклы и условия приводятся к единой форме записи, исключаются такие незначащие символы как пробелы, табуляции и переходы на новую строку, а также комментарии.
2. Порядок выполнения операторов не имеет значения. Это вызвано тем, что обычно для части команд в программе допустима вариативность в последовательности их выполнения, то есть их перестановка не повлияет на результат программы. Например, если программа выполняет вычисления над координатами точки двумерного Евклидового пространства, то нет разницы в том, какую координату точки изменить первой, *x* или *y*. При этом из-за невозможности внедрения какой-либо иерархии на основе последовательности операторов, вводится иерархия вложенности: операторы могут связываться друг с другом отношением вложенности, например, выражение находится в цикле, цикл находится в функции.
3. Все циклы преобразуются к циклу while, действия же, выполняемые в его блоках (объявление/определение переменной, проверка условия, изменение счетчика), становятся операциями внутри цикла. Это вызвано тем, что в итоговой сети не должно существовать разницы между действиями внутри блоков задания цикла и действиями в теле цикла. Например, вечный цикл с условием выхода внутри его тела и просто цикл с условием выхода равнозначны. Также как равнозначен инкремент счётчика внутри тела цикла или внутри блока задания цикла.

   while (true) if (n > 10) break; <-> while(n > 10)
   for(;; ++i) <-> for(;;) ++i;

4. Условия не различаются по своему типу (if, else if, else). При этом условия могут связываться друг с другом в последовательную цепь. Это вызвано тем, что блок else if концептуально использует предыдущий блок if при определении истинности выражения: если условие в блоке if истинно, то условие в else/else_if будет всегда ложно. Несколько вложенных условий заменяются одним условием. В самом деле нет разницы между несколькими условиями и одним условием, записанным через логические операторы. Естественно, данная замена происходит при условии, что внутри тела условия имеется только операторы условия.

   if (a > b) if (a > c) <-> if (a > b && a > c)

5. В выражениях учитывается только использование переменных/функций, численные значения не учитываются. Также не учитываются математические отношения связывающие переменные

в выражения. Это вызвано потенциально бесконечным количеством возможных записей для любой формулы. Иными словами выражения только отражают отношение некоторой абстрактной зависимости между переменными.

a /= 15 * b * c <-> a = a/ (3 * b * c * 5) <-> a = a,b,c

Основываясь на приведённых выше принципах, опишем общий вид получаемой в результате семантической сети. Итоговая сеть формируется из узлов, представляющих собой операторы языка, либо его концепции и рёбер, которые соединяют данные узлы отношениями, в которые вступают операторы языка. В написанной реализации алгоритма использовались следующие узлы:

— концепты (типы данных, а также функция, цикл и условие как некоторое абстракции);
— имена функций;
— имена переменных;
— реализации условия (по блокам, блок представляет собой один оператор if, else if, else);
— реализации цикла;
— конкретные выражения.

Имена функций и переменных не используются в дальнейшем и требуются только на этапе построения семантической сети для связывания участников выражения с конкретными переменными и команд с функциями, в которых они расположены. Ребра являются направленными, однако для упрощения обхода сети алгоритмами поиска, узлы связываются одновременно двунаправленными рёбрами. Каждая такая пара рёбер отражает отношение как его прямую и обратную часть, например, «ребёнок» и «родитель», или «включает» и «входит в состав». Всего используется восемь отношений (приводятся отношение и обратное ему отношение):

— input/get_from — для отображения ввода;
— output/put_to — для отображения вывода;
— use/participate — для отображения участия переменных в выражении или условии;
— sets/assigned — для отображения задания значения переменных в выражении;
— is/child — для отнесения узла к какому-либо языковому концепту;
— contain/located — для отображения вложенности одних операторов в другие;
— return/associated — для отображения типа, возвращаемого функцией;
— next/previous — для отображения связанности древовидных условий (else и else if).

Таким образом, на этапе задания допустимых узлов и рёбер можно определить, насколько точно семантическая сеть будет передавать текст программы, либо насколько сильно она будет его обобщать. Например, можно объединить отношения contain и use, или input и sets. При этом данный выбор влияет только на чувствительность сети, асимптотика алгоритма же остаётся неизменной.

Тогда алгоритм преобразования программного кода в семантическую сеть выглядит следующим образом:

1. Препроцессинг программного кода (унификация синтаксиса).
2. Выделение в программе отдельных операторов.
3. Последовательно для каждого оператора производится установление ему в соответствие некоторого узла в семантической сети и отношений этого узла с другими вершинами. Обработку команд программы необходимо выполнять последовательно для того, чтобы узлы, соответствующие используемым в выражениях переменным, уже существовали. То есть объявление переменной создаёт для неё вершину, а использование переменный лишь порождает новое отношение.
4. После определения типа узла, которое соответствует текущему оператору проверяется наличие начала вложенного блока, например, тела цикла, условия, функции. Если это имеет место быть, то алгоритм вызывает себя рекурсивно с места начала вложенного блока. Полученные в результате работы рекурсии имена всех узлов внутри блока связываются с текущим узлом отношением вложенности.
5. Имя текущего узла добавляется в список узлов, созданных алгоритмом при данном вызове
6. На данном этапе, если алгоритм не встречает символ завершения вложенного блока (например окончание тела цикла) и, при этом операторы программы не иссякли, то алгоритм возвращается на шаг 3.
7. Алгоритм возвращает список имён, созданных им узлов вызывающему окружению.

Способы представления семантической сети

Была написана программа, которая по описанному выше алгоритму строит семантическую сеть программного кода. Данная программа доступна по адресу https://github.com/rcomrad/KusOntology.

Построим семантическую сеть на основе программы, реализующей поиск всех чётных чисел на заданном отрезке, написанную на языке C++ и имеющую следующий вид:

```
main()
{
int min, max;
cin >> min >> max;

for (int i = min; i < max + 1; i++)
{
```

```
if (i % 2 == 0) cout << i << " ";
}

return 0;
}
```

Первым возможным способом представления семантической сети может являться перечисление узлов и его соседей [6], полученный результат представлен в Таблице 1.

Представление имеет вид таблицы исключительно с целью экономии места. В нём каждый узел отображается пронумерованным пунктом вместе со своим именем. Далее следует словарь из отношения и массива имён узлов-соседей, с которыми стартовый узел связан текущем отношением. Данное представление является наиболее удобочитаемым для человека форматом и детально демонстрирует работу алгоритма. Естественным минусом является же то, что список соседей занимает достаточный объем как при печати, так и в оперативной памяти компьютера.

Следующим логичным шагом может является присвоение числовых характеристик вместо строчных значений для рёбер и вершин, что позволяет представить и хранить семантическую сеть в более компактном виде. Данное представление не приводится вследствии его тривиальности (слова в Таблице 1 заменяются уникальными числами). Однако данное изменение позволяет создать весьма компактное представление семантической сети. Для этого перемножим числовые характеристики соседних рёбер и связанных с ними вершин. После такой операции вершина будет характеризоваться некоторым одномерным числовым массивом. При этом упорядочим произведения в порядке номеров вершин, характеристики которых входят в произведения. Если до какой-либо вершины ребро отсутствует (она не является соседом), будем считать её произведение равным нулю.

Таблица 1.

Представление семантической сети списком соседей

1) cicle child cicle_0 2) condition child cond_block_0 cond_block_1 3) function is main 4) main child function return int contain cicle_0 expr_block_000 expr_block_001 max min 5) int associated main child i max min	6) i is int assigned expr_block_002 expr_block_004 participate expr_block_003 expr_block_005 located cicle_0 put_to expr_block_006 7) max is int get_from expr_block_001 participate expr_block_003 located main 8) min is int get_from expr_block_000 participate expr_block_002 located main	9) cond_block_0 is condition use expr_block_003 located cicle_0 10) cond_block_1 is condition use expr_block_005 contain expr_block_006 located cicle_0 11) cicle_0 is cicle contain cond_block_0 cond_block_1 expr_block_002 expr_block_004 i located main 12) expr_block_000 input min located main	13) expr_block_001 input max located main 14) expr_block_002 sets i use min located cicle_0 15) expr_block_003 use i max participate cond_block_0 16) expr_block_004 sets i located cicle_0 17) expr_block_005 use i participate cond_block_1 18) expr_block_006 output i located cond_block_1

	2	41	37	37	2	43	43	43	43	43	43	43	2	31	5	3	31	31
2	0	97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	89	0	101	101	0	0	0	101	0	101	0	0	0	101	0	103	0	0
37	0	103	0	0	89	0	0	0	71	0	0	0	0	0	0	0	0	0
37	0	103	0	0	89	0	0	0	0	0	71	101	0	0	0	0	0	0
2	0	0	97	97	0	0	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	0	53
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	53	0
43	0	103	0	0	0	0	0	0	0	0	0	0	0	79	0	0	0	71
43	0	0	73	0	0	0	0	0	0	0	0	0	0	71	0	0	71	0
43	0	103	0	0	0	0	0	0	0	0	0	0	0	79	0	0	0	0
43	0	0	0	73	0	0	0	0	0	0	0	0	0	71	0	0	0	0
43	0	0	0	103	0	0	0	0	0	0	0	0	0	61	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89	0	0
31	0	103	0	0	0	0	0	83	73	83	73	67	0	0	89	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	109	97	97
3	0	101	0	0	0	101	101	0	0	0	0	0	97	0	107	0	101	101
31	0	0	0	0	0	0	59	0	73	0	0	0	0	0	89	103	0	0
31	0	0	0	0	0	59	0	73	0	0	0	0	0	0	89	103	0	0

Рис. 1. Матрица семантической сети

Тогда семантическую сеть можно представить в виде следующей матрицы, представленной на рисунке 1

Стоит отметить, что в данном представлении каждому ребру/вершине ставилось в соответствие характеристика в виде простого числа с целью получения уникальных произведений характеристик в каждом конкретном случае. Также матричная форма позволяет обрабатывать семантическую сеть существующими алгоритмами определения изоморфности графов. Например, при оценке качества программного кода путём сравнения его с эталонным решением [3] становится возможным использовать такие алгоритмы изоморфности графов как ISD [7].

Можно заметить, что данная матрица является частично симметричной относительно главной диагонали. Под этим будем подразумевать то, что все нулевые значения симметричны относительно главной диагонали. Однако ненулевые числа при отражении относительно главной диагонали заменяются на некоторое другое значение. При этом симметрично «отражённое» значение является постоянным для «отражаемого» числа и не меняется в зависимости от положения ячейки в матрице. Это вызвано тем, что каждое отношение имеет для себя противоположное отношение. Что и проявляется в домножении значений ячеек на некоторое отношение N/M относительно главной диагонали, где N — числовая характеристика отношения, а M — числовая характеристика обратного отношения.

Заключение

В статье был представлен улучшенный алгоритм построения семантической сети из программного кода, написанного обучающимися средней школы в процессе обучения программированию. Полученная семантическая сеть может применяться для оценки знаний обучающихся по программированию и на её основе построения индивидуальных образовательных траекторий и, как следствие, повышения качества образовательного процесса. В статье также обозначены различные способы представления построенной семантической сети.

ЛИТЕРАТУРА

1. Е.И. Огородникова, «Проектирование образовательных траекторий личности», Образование через всю жизнь: непрерывное образование в интересах устойчивого развития., 2012, № 1 — С. 219–221
2. Е.И. Огородникова, «Выстраивание индивидуальных образовательных траекторий», Образование через всю жизнь: непрерывное образование в интересах устойчивого развития., 2012, № 1 — С. 219–221
3. А.С. Федоров, А.Н. Шиков, «Метод преобразования семантической сети для автоматизации оценивания решения задач по программированию в процессе электронного обучения», Вестн. Астрахан. гос. техн. ун-та. Сер. управление, вычисл. техн. информ., 2020, № 4, 7–17
4. Родзин С.И., Родзина О.Н. Модели представления знаний. Практикум по курсу «Системы искусственного интеллекта»: учебн. пособие. Таганрог: Изд-во ЮФУ, 2014. 151
5. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. СПб: Питер, 2000. 384 с.
6. Иванова Г.С. «Способы представления структурных моделей» Программные продукты и системы, 2011, № 4 — С. 91–96
7. Погребной В.К. Решение задачи определения изоморфизма графов, представленных атрибутными матрицами // Известия Томского политехнического университета. — 2012. — № 5. — С. 52–56.

Журнал «Современная наука: актуальные проблемы теории и практики»