КОМПЬЮТЕРНОЕ ОБЕСПЕЧЕНИЕ И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

DOI: 10.24143/2072-9502-2020-4-7-17
УДК 004.9

МЕТОД ПРЕОБРАЗОВАНИЯ СЕМАНТИЧЕСКОЙ СЕТИ ДЛЯ АВТОМАТИЗАЦИИ ОЦЕНИВАНИЯ РЕШЕНИЯ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ В ПРОЦЕССЕ ЭЛЕКТРОННОГО ОБУЧЕНИЯ

А. С. Федоров, А. Н. Шиков

Национальный исследовательский университет ИТМО,
Санкт-Петербург, Российская Федерация

Представлен метод преобразования семантической сети понятий, получаемой для программного кода в *N*-мерный вектор. Предложенный метод позволит автоматизировать определение используемых алгоритмов при решении задач по программированию в процессе электронного обучения. Метод включает оригинальные алгоритмы построения и преобразования семантической сети. Для определения алгоритма в программном коде находится шаблон этого алгоритма, представленный в виде подграфа абстрактных понятий языка в семантической сети, построенной на основе данного кода. Поиск алгоритма с помощью сравнения подграфа сети с шаблонной сетью позволили выявить алгоритм BFS с заданной точностью: порог отсечения для выходов перцептрона равен 0,85, из расчета точности однослойного перцептрона при классификации базы MNIST, равной 88 %, что подтверждает эффективность разработанного метода и требует дальнейших исследований с применением методов машинного обучения для поиска оптимального значения координат узлов семантической сети и шаблонов алгоритмов.

Ключевые слова: автоматизация, алгоритмы машинного обучения, семантическая сеть, программный код, алгоритм BFS, числовой вектор.

Для цитирования: *Федоров А. С., Шиков А. Н.* Метод преобразования семантической сети для автоматизации оценивания решения задач по программированию в процессе электронного обучения // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. 2020. № 4. С. 7–17. DOI: 10.24143/2072-9502-2020-4-7-17.

Введение

При обучении языкам программирования возникает задача оценки качества обучения. Во-первых, необходимо оценивать качество решения поставленных задач при условии, что различные решения выдают одинаковый результат. Во-вторых, требуется оценивать степень усвоения материала обучаемыми как вероятность применения полученных знаний для решения задач, похожих на учебные, но не идентичных им. Это достаточно актуально, когда обучение осуществляется с применением дистанционных технологий и электронного обучения с большим числом обучаемых, т. к. в этом случае процесс проверки учебных заданий достаточно трудоемок и занимает много времени. Поэтому автоматизация оценки качества решения задач по программированию является актуальной задачей, требующей реализации.

Одной из основных задач при автоматизации процесса проверки программного кода для некоторой задачи по программированию является определение участков кода, представляющих собой алгоритм, необходимый для решения задачи. После определения алгоритма в программном коде можно замерить различные метрики данного решения либо оценить количество «излишних» объектов и действий, а также оценить связность участка кода с остальным кодом.

Методы машинного обучения обеспечивают наилучшим образом решение поставленной задачи исследований. «При решении задачи машинного обучения в большинстве случаев используется индивидуальный подход. Человек, решающий эту задачу, на основе своего опыта выбирает определенный алгоритм, настраивает его параметры» [1, с. 20].

Основные трудности заключаются в невозможности прямой обработки кода данными алгоритмами. Большинство алгоритмов машинного обучения работают с данными, представляемыми в *N*-мерных векторах евклидова пространства. Таким образом, необходимо построить отображение множества взаимосвязанных команд программы во множество точек *N*-мерного пространства. При этом должны выполняться следующие условия:

– близкие по смыслу действия и объекты языка (например, несколько условий или несколько функций) ставятся в соответствие близким точкам в *N*-мерном пространстве;

– размерность пространства, вмещающего итоговое преобразование кода, не должна зависеть от размера кода, иначе будет невозможно сравнивать коды со значительной разницей в размере;

– мера занимаемого пространства для различных решений должна стремиться к значениям одного порядка, иначе при нормировании входных данных возможна потеря информации о близости понятий, представленной в виде расстояния между точками.

Преобразование программы во множество *N*-мерных точек обеспечивается семантической сетью, которая представляет собой ориентированный граф. «Проблема поиска решения в базе знаний сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, отражающей поставленный запрос к базе» [2, с. 23]. По сети можно осуществить поиск, используя знания о смысле отношений и операции: сопоставление с образцом, поиск, замена, взятие копии.

Алгоритм построения семантической сети

В работе «Модели представления знаний» [3] рассматриваются 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.

Абстрактными объектами предметной области могут выступать обобщенно понимаемые конструкции языка программирования С++: функции, типы данных, переменные и их виды, а также условия и циклы. Например, на рис. 1 представлен вариант подграфа базовых понятий предметной области языка С++.

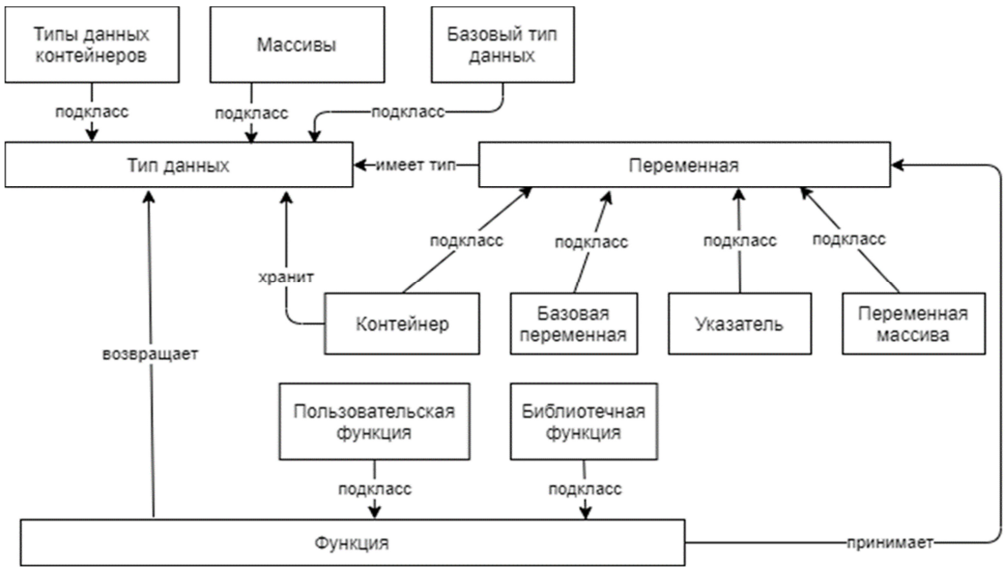

Рис. 1. Подграф сети абстрактных объектов предметной области

Свойства выделенных понятий можно определить как конкретные реализации команд в языке программирования: функции, переменных, условий, типов данных и циклов. Используя приведенный подграф, можно описать объявление переменной и контейнера в функции main() на языке C++ (рис. 2).

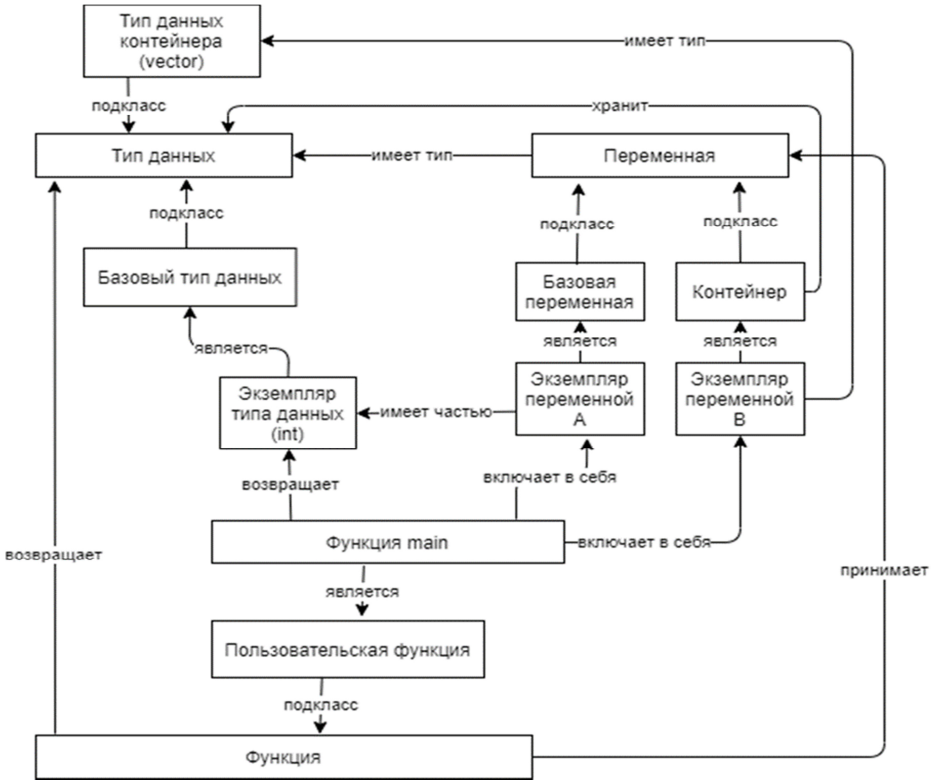

Рис. 2. Структура семантической сети

В качестве связей между вершинами предлагается использовать отношения реализации абстрактных понятий – IsA (отношение принадлежности к классу), для элементов, входящих в программный блок (например, блок функции main) использовать отношение часть–целое, а также отношения вызова или использования объектов (например, вызов функции, использование переменной как аргумента).

Таким образом, для преобразования программного кода в семантическую сеть разобьем код на действия построчно (строку определим как набор команд, заканчивающийся специализированным символом: точкой с запятой, фигурной скобой, круглой скобкой для операторов цикла и условия). Узлы данной сети отображают команды программного кода (например, объявление переменной или проверка условия). Узел определяется используемыми для этого действия сопутствующими элементами программы (например, используемыми в условии переменными) и вложенными действиями, как, например, функция определяется командами, выполняемыми при вызове функции (телом функции). Данный способ был реализован на практике с использованием языка Java, что позволяет получать семантическую сеть по коду. Приведем пример для кода, представленного ниже:

```
int main ( )
{
int a;
a = 0;
double b;
if (a > 5) a = 6;
else b = 6;
}
```

Получена сеть по коду, приведенному выше, представленная на рис. 3.

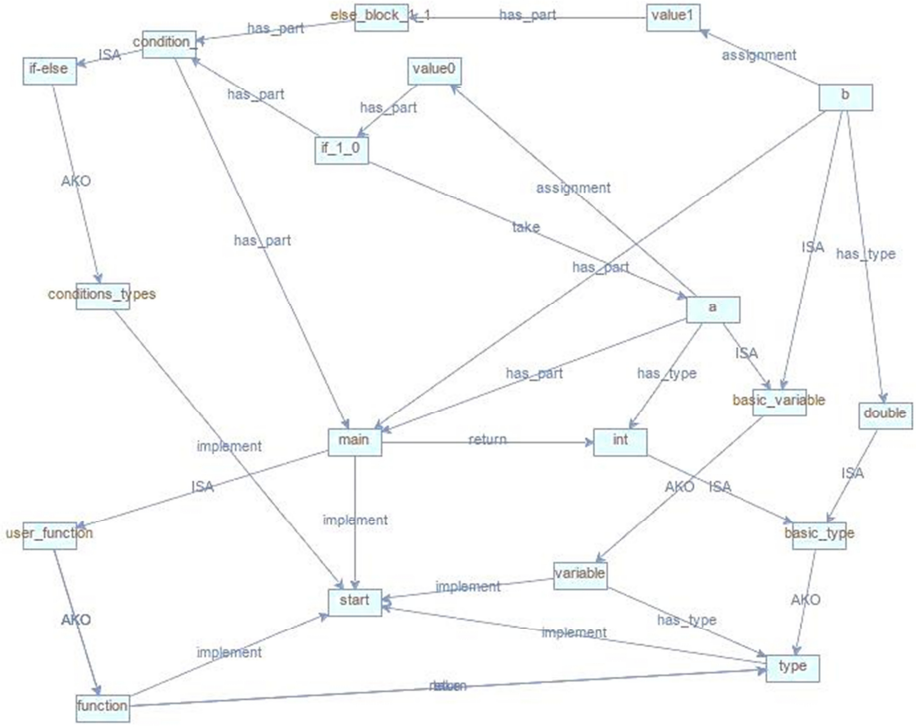

Рис. 3. Структура полученной семантической сети

Названия узлов в данной сети включают стандартные типы данных – int, double; названия переменных – a, b; названия функций – main; условная точка входа – start; принимаемые значения – value0, value1; блоки условия: condition_1 – ветвление номер 1, if_1_0 – блок номер 0 типа «если условие истинно» оператора ветвления номер 1, else_block_1_1 – блок номер 1 типа «иначе» оператора ветвления номер 1; абстрактные понятия: variable – переменная, type – тип, basic_variable – базовая переменная (переменные, не являющиеся указателями, массивами, контейнерами, структурами или объектами иного составного типа данных), function – функция, user_function – функция, написанная пользователем, condition_types – виды операторов ветвления, if-else – оператор ветвления если–иначе; ребра могут представлять связи: ISA – является, AKO (A Kind Of) – отношение подкласс/разновидность, has_part – является частью, assigment – принимает значение, take – имеет аргументом, has_type – имеет тип, return – возвращает значение, impliment – включает (например, все действия в теле функции по отношению к функции являются включенными в нее).

Для отображения принадлежности таких понятий, как массив и переменные, к одному базовому классу (массив и одиночная переменная имеют большее сходство, чем массив и функция), в данную сеть добавляются узлы-понятия предметной области.

Построение семантической сети для алгоритма

Хорошим примером для задачи определения алгоритма являются задачи по олимпиадному программированию. Такие задачи могут быть решены с применением различных подходов, так же как и сама реализация одного и того же алгоритма может различаться в соответствии со стилистикой программиста. Естественно предположить, что каждый алгоритм имеет свои уникальные конструкции. Это может быть использование определенных контейнеров, методов обхода, функций и др., которые можно обобщить, абстрагировать и создать некий «шаблон» алгоритма в виде семантической сети. Данная сеть строится по аналогичному принципу, описанному выше, но не содержит конкретных понятий (например, конкретных переменных). При наличии шаблонов для возможных алгоритмов среди них выбирается тот, который в наибольшей степени соответствует подсети исследуемого кода. При этом программный код не может состо-

ять только из алгоритма, например, также необходимы ввод-вывод данных. Для решения проблемы наличия иных действий в задаче, помимо самого алгоритма решения, возможно разделить задачу на несколько алгоритмов (например, ввод, поиск ответа, вывод) и оценивать каждый из них в отдельности.

Способ создания шаблона может варьироваться. Одним из вариантов является автоматическая генерация шаблона на основе большого количества тестовых данных – примеров программ/алгоритмов. Однако такой подход достаточно трудозатратен из-за необходимости разметки данных (примеров программ), при этом далеко не всегда имеется обширная база данных решений задачи. Представим метод ручного создания шаблона для алгоритма BFS.

Очень часто в задачах на алгоритм поиска в ширину BFS требуется найти наиболее эффективное решение, например минимальный путь. Далее рассматривается модифицированный алгоритм BFS, где вместо окраски узлов используется информация о том, посещался ли данный узел. Хранится данная информация с помощью высчитываемого расстояния от стартового узла до узла, рассматриваемого на данной итерации алгоритма. Таким образом, вместе с обходом графа одновременно вычисляется минимальное расстояние до узла. Без предварительного создания необходимых массивов иерархию шагов, которая отражает вложенность действий в программе, можно представить следующим образом [4]:

1. Инициализация буфера FIFO начальным значением.
2. Запуск цикла, завершающего свою работу только после того, как буфер FIFO будет пуст.
2.1. Взятие очередного узла из буфера FIFO.
2.2. Удаление очередного узла из буфера FIFO.
2.3. Запуск цикла по обходу соседей полученного из буфера узла.
2.3.1. Сравнение расстояния в целевом узле (соседе полученного из буфера узла) и расстояния в полученном узле плюс расстояние от текущего узла до целевого узла.
2.3.1.1. При срабатывании условия в буфер FIFO добавляется целевой узел [4].

По данному алгоритму разработана сеть для определения алгоритма BFS, представленная на рис. 4.

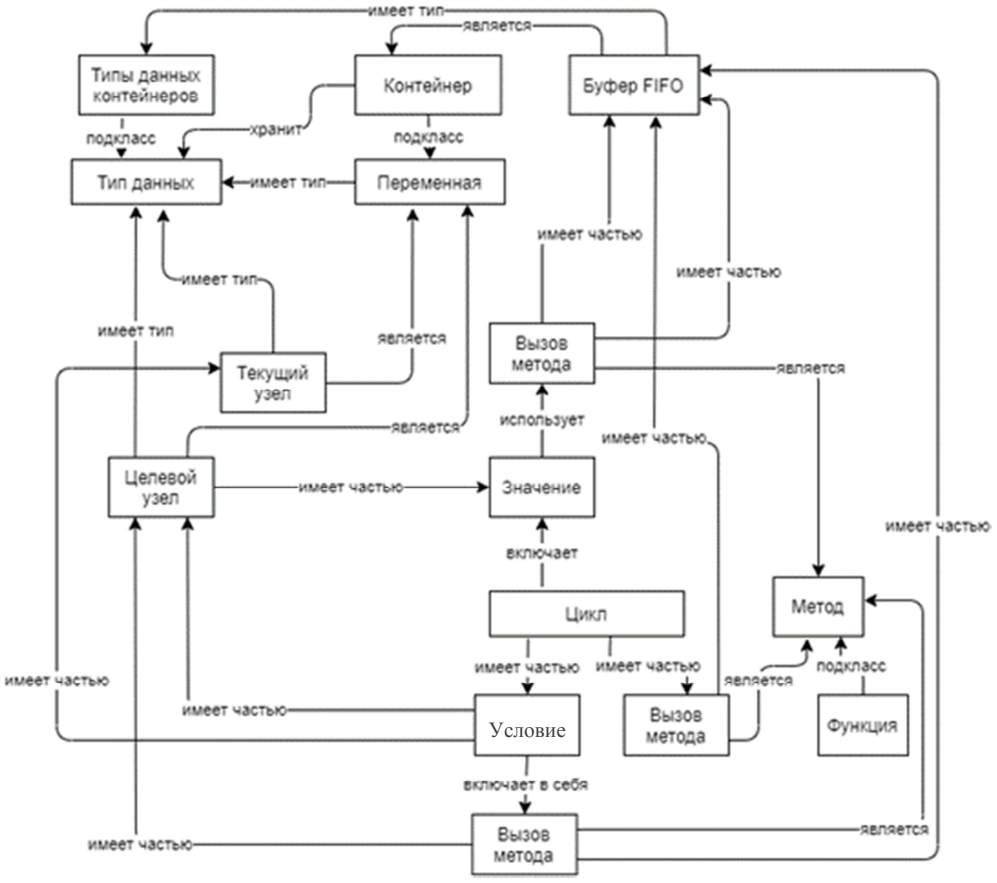

Рис. 4. Сеть алгоритма BFS

В данной сети отсутствуют некоторые связи. Например, «Целевой узел» не имеет отношения «включен» с каким-либо узлом. Данное отношение подразумевает иерархическую подчиненность в коде. Отсутствие иерархических связей создает устойчивость относительно места объявления переменной. Присутствуют и обобщения, например, не указаны однозначно методы, используемые контейнеры и цикл. Узел «Текущий узел» имеет обобщенный тип данных и является обобщенной переменной. Все это позволяет обеспечить устойчивость к различным реализациям алгоритма. Полученная сеть устойчива к порядку выполнения команд (например, к различному порядку объявления переменных), поскольку в ней учитывается только иерархическая связь между командами.

Преобразование семантической сети в численный вектор

Для обработки семантической сети с помощью алгоритмов машинного обучения необходимо поставить в соответствие данной сети некоторый *N*-мерный вектор. Для этого поставим в соответствие каждому узлу в сети некоторый вектор. Данный вектор описывает узел через его отношения с другими узлами семантической сети, в соответствии с ребрами, которыми они соединены друг с другом. Тогда вся сеть будет описываться набором векторов – таблицей. Данная таблица является двумерным массивом, а задача преобразования двумерного массива в одномерный достаточно тривиальна.

Сама семантическая сеть является промежуточным результатом разбиения кода на блоки и необходима для построения отношений между различными объектами программы (например, вложенность или использование в качестве аргумента переменных), с помощью которых и будет происходить разбиение кода на блоки. Данные отношения необходимы не только для иерархического упорядочивания команд (функция main(), ее блоки и блоки блоков можно представить как иерархически подчиненные друг другу команды), но и для сохранения связей с объектами вне программного блока (например, переменная, используемая в цикле, может быть объявлена вне него). Если разделять код исключительно по указанному иерархическому принципу, значимые связи между объектами программы могут быть утрачены. Сама сеть представляется в виде списков смежных вершин.

Еще одной сложностью использования семантической сети программного кода напрямую является громоздкость ее графического представления. Даже для незначительного участка кода, приведенного выше (код для рис. 3), получаемая сеть весьма объемна (см. рис. 3). При этом в общем случае сеть будет иметь множество пересечений ребер и узлов при отрисовке в реализованном программном обеспечении, что требует ручного смещения узлов в графическом отображении сети. Таким образом, таблица также выполняет роль представления сети в удобочитаемом формате.

Для преобразования семантической сети в вышеописанную таблицу можно воспользоваться модифицированным способом перекрестного поиска (поиска в ширину). Поиск будет запускаться из центрального узла для данного модуля программы/подсети (например, из узла функции main ()). Получим для каждого узла список пар «отношение-узел», отражающих путь до ближайшего узла с заданным свойством.

Формально алгоритм получения таблицы из семантической сети можно представить следующим образом:

1. Выбрать стартовый узел сети для поиска в ширину.

2. Определить для каждого узла его соседей в соответствии с его связями: поиск в ширину распространяется по направленным ребрам, передавая в качестве аргумента предыдущие узлы, затронутые поиском, и их ребра.

3. Для каждого узла на основании его соседей строится двумерный массив, где количество элементов равно количеству видов ребер; сами элементы представляют собой набор узлов, являющихся соседними с данным узлом.

4. Полученные вектора для узлов сети разбиваются по типу (контейнеры, циклы, условия и т. п.).

5. Массивы соседей записываются друг под другом, образуя таблицу, блоками, полученными на предыдущем этапе.

Ниже в качестве примера в таблицу преобразован простейший пример алгоритма BFS:

```
vector<vectorr<int>>graph(nodeCount, vectorr<int>(0));

vector<int>arr(nodeCount, 1e6);
arr[startPoint] = 0;

queue<int>q;
q.push(startPoint);

while (!q.empty())
{
   int current = q.front();
   q.pop();

   for (int i = 0; i < graph[current].size(); i++)
   {
        int next = graph[current][i];

        if (arr[next] > arr[current] + 1)
        {
             arr[next] = arr[current] + 1;
             q.push(next);
        }
   }
}
```

Полученная семантическая сеть кода не приводится по причине ее объемности. Полученная таблица (без абстрактных понятий) представлена ниже (табл.).

Таблица, получаемая из семантической сети

Типы действий (сегмент)	Имя	ISA	Является_частью	Включает
Объявление контейнеры	graph	контейнер vector	vector<int>	–
	arr	контейнер vector	int	–
	q	контейнер queue	int	–
Циклы	while_1	цикл пока	очередь_1	–
	for_1	цикл for	graph	while_1
Условия	if_1	ординарное условие	arr, next, current	while_1
Действия с переменными	current	присваивание	front_1	while_1
	next	присваивание	graph, current	while_1
	arr[]	присваивание	arr, next, current	while_1
Вызов функций	pop_1	метод контейнера queue	–	while_1
	push_1	метод контейнера queue	next	while_1
	front_1	метод контейнера queue	–	while_1

Для сохранения связности сети некоторые операторы, не входящие в модуль (подсеть), все равно присутствуют в таблице. Для упрощения вычислений отношения имеют тип, присваивание и аргумент. Однако это может уменьшить качество преобразования сети в таблицу (уменьшается количество передаваемой информации).

Для перехода к числовому вектору каждому узлу ставится в соответствие число на координатной оси. При этом близкие по смыслу понятия (например, две функции или две переменные) группируются, т. е. должны иметь координаты в окрестности некоторой общей точки. Например, все действия с переменными имеют координаты в окрестности точки 100, а все вызовы функций – координаты в окрестности точки 1 000. Переход к координатам необходим, поскольку в общем случае структура сети неизвестна, а это означает, что неочевидна возможность покраса графа (нумерование узлов графа) таким образом, чтобы сгруппировать узлы по смыслу понятий, которые они отображают. Общий алгоритм для получения координат не выработан, координаты подбирались эмпирически. Итоговый числовой вектор является последовательным набором всех значений таблицы.

Полученные значения координат для узлов сети подставляются в таблицу в те ячейки, в которых записаны имена узлов, поставленных в соответствие данным координатам. Естественно, что в ячейке может содержаться несколько узлов (например, для условия if_1 имеется 3 узла, связанных отношением «является частью» – if(arr[next] > arr[current] + 1)), в таком случае значение ячейки равняется среднему арифметическому подставляемых координат (один из возможных способов объединить получаемые значения). В итоге будет получена таблица соответствия узлов сети некоторому числовому вектору. Данная таблица определяет узел через его соседей. Для шаблонов алгоритмов можно построить схожие сети, за исключением того, что в таблице будут присутствовать только абстракции и обобщения.

Для унификации порядка строк в таблице она разбивается на сегменты по типу узла (выполняемого действия (см. табл., колонка «Типы действий (сегмент)»). Чтобы решить задачу различного количества строк в двух сравниваемых таблицах (шаблонной и исследуемой), таблицы-шаблоны для алгоритмов можно дополнить пустыми строчками в каждом сегменте (ограниченного количества) – таким образом определяется размер входного вектора N для алгоритма машинного обучения. Например, на каждый сегмент таблицы выделяется пять строк, три из которых заполнены, а остальные пустые, и если в алгоритме присутствуют дополнительные операторы циклов, условий или тому подобное, шаблон все равно может быть сопоставим с тестируемым образцом – лишние операторы образца займут места пустых строчек в таблице. Это позволяет отсечь модули со слишком большим количеством операторов (модули, состоящие из нескольких алгоритмов), поскольку в данных таблицах будет сбиваться порядок сегментов при превышении требуемого размера таблицей исследуемого кода: таблица урезается без учета порядка сегментов.

Получаемая таблица (шаблон) уникальна для каждой задачи. Ее структура формируется как преподавателем напрямую при выделении алгоритмов в задаче (например, ввод, получение ответа, вывод), так и косвенно путем «обучения» нейронной сети на идеальном решении: на идеальном решении строятся семантическая сеть и таблица, которые и используются для оценки решений обучающихся. При этом преподаватель может признавать корректным решением несколько алгоритмов и создать для каждого из них отдельный перцептрон.

Существует проблема определения порядка строк в сегменте таблицы, выбора сопоставляемых строк при неравном количестве пустых и заполненных строк в таблице. Используемым решением в данном исследовании является заполнение пустых строк имеющимися строчками до совпадения количества заполненных строк.

Метод определения алгоритма

Чтобы выявить алгоритм в исследуемом программном коде, необходимо выявить участок кода, соответствующий шаблону искомого алгоритма. Поскольку шаблон алгоритма является идеальной моделью, можно воспринимать алгоритм в коде как зашумленный образ. Поскольку вышеописанная таблица рассматривается как шаблон, идеальный образ, а исследуемые программы как зашумленный образец, для определения алгоритма использовался однослойный перцептрон. Сравнение различных видов нейронных сетей не проводилось.

Процесс обучения для однослойного перцептрона представляет собой присваивание коэффициентам перцептрона W значений числового вектора полученного шаблона для искомого алгоритма – обучение в традиционном смысле не проводится, коэффициенты каждого перцептрона равны значениям числового вектора таблицы шаблона, который отображает данный перцептрон. В качестве функции активации используется сигмоида вида

$$f$$

$$(x) = \frac{1}{1+e^{-x}} f.$$

Схема используемого однослойного перцептрона приведена на рис. 5.

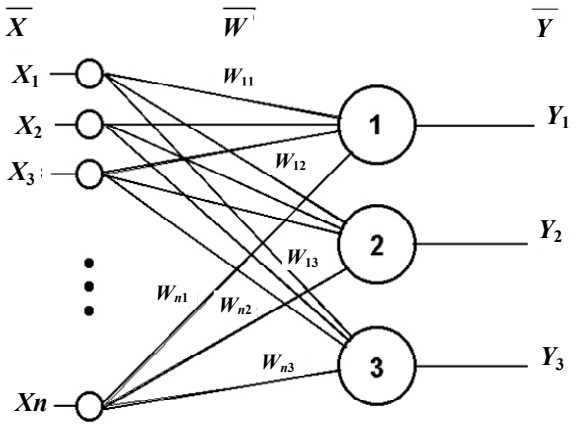

Рис. 5. Схема перцептрона

Шаблон алгоритма запоминается через настройку весов, где j – номер перцептрона; $i \in [0, N]$. Исследуемый программный код подается на входы x_i в виде N-мерного вектора. Каждый перцептрон формирует выходное значение y_k, где k – количество перцептронов в слое. Выходное значение y_k характеризует степени схожести поступившего на вход программного кода с занесенным в перцептрон шаблоном [5].

Для решения вопроса поиска конкретного участка кода, соответствующего шаблону, исследуемый код рекурсивно разбивается на составные блоки, каждый из которых сверяется с шаблоном. Например, сначала на вход однослойного перцептрона подается весь код основной функции main (семантическая сеть в виде N-мерного вектора), затем из кода исключается объявление основной функции (содержание функции, т. е. набор команд, определяющий ее, остается). Полученный код разбивается на блоки по принципу вложенности (цикл и его тело, условие и его тело и т. п., одиночные операторы могут пропасть, как это описано выше). Далее, полученные блоки по отдельности сравниваются с шаблоном с помощью однослойного перцептрона. Затем каждый из блоков разбивается на новые блоки путем удаления связывающих их команд, и весь процесс повторяется, пока код не будет разбит на одиночные команды. При достижении определенного порогового значения выходным значением однослойного перцептрона можно говорить о наличии алгоритма в участке кода, который был подан на вход однослойного перцептрона.

В понятиях сети данный подход можно представить как разбиение сети на подсети. Узел, представляющий функцию main(), а также все последующие узлы, образующие программный блок (например, циклы или условия), можно определить как «центральные» узлы. Потомки узла определяются как узлы, соединенные с данным узлом отношением включения (например, узел определения main соединен отношениями включения с узлами команд, содержащихся в функции main). Тогда на каждой итерации рекурсии исключается очередной центральный узел сети, а его потомки становятся новыми центральными узлами и образуют со своими потомками подсети для очередной итерации рекурсии.

В итоге после обработки исследуемого кода, создания на его основе семантической сети и обработки данной сети однослойным перцептроном с рекурсивным разбиением сети на подсети выходные значения однослойного перцептрона образуют массив соответствий. В полученном массиве каждой подсети семантической сети исходного программного кода (блоку программного кода) соответствует некоторый алгоритм с некоторой степенью схожести, равной выдаваемому на выходе перцептрона значению.

Следует отметить, что шум может нести информацию о неэффективном использовании переменных, лишних действиях и прочих ошибках логического характера. Одним из направлений модификации алгоритма является выделение, обработка и распознавание шума в получаемых сетях.

Заключение

Поиск алгоритма с помощью сравнения подграфа сети с шаблонной сетью позволил выявить алгоритм BFS с заданной точностью: порог отсечения для выходов перцептрона равен 0,85 из расчета точности однослойного перцептрона при классификации базы MNIST равной 88 % [6]. В данном случае предполагается схожесть задач – распознавание зашумленных образов по заданному шаблону.

Однако база данных, используемая при тестировании созданного на основе изложенного алгоритма программного обеспечения, не является полной. Вероятна неустойчивость полученного алгоритма к различным вариантам решения. Способом решения данной проблемы может быть автоматическая настройка значений координат и автоматическое создание шаблонов алгоритмов. Для реализации данной задачи необходимо создать значительную базу решений.

В рамках проведенного исследования предложен метод по преобразованию сети понятий, получаемой для программного кода, в *N*-мерный вектор. Предложенный метод позволит автоматизировать определение используемых алгоритмов при решении задач по программированию. Возможным направлением будущих исследований является применение методов машинного обучения для поиска оптимального (дающего наилучшие результаты определения алгоритма) значения координат узлов семантической сети и шаблонов алгоритмов.

СПИСОК ЛИТЕРАТУРЫ

1. *Невоструев К. Н.* Обзор литературы по методам машинного обучения (machine learning) // КИО. 2014. № 4. С. 19–26.

2. *Гаврилова Т. А., Хорошевский В. Ф.* Базы знаний интеллектуальных систем. СПб.: Питер, 2001. 394 с.

3. *Родзин С. И., Родзина О. Н.* Модели представления знаний. Практикум по курсу «Системы искусственного интеллекта»: учебн. пособие. Таганрог: Изд-во ЮФУ, 2014. 151 с.

4. *Обход* в ширину // Сайт вики-конспектов Университета ИТМО. URL: https://neerc.ifmo.ru/wiki/index.php?title=Обход_в_ширину (дата обращения: 20.03.2020).

5. *Осовский С.* Нейронные сети для обработки информации / пер. с пол. И. Д. Рудинского. М.: Финансы и статистика, 2002. 344 с.

6. *MNIST* (База данных). URL: https://ru.wikipedia.org/wiki/MNIST_(%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)#cite_note-Multideep-8 (дата обращения: 14.02.2020).

Статья поступила в редакцию 25.03.2020

ИНФОРМАЦИЯ ОБ АВТОРАХ

Федоров Александр Сергеевич – Россия, 197101, Санкт-Петербург; Национальный исследовательский университет ИТМО; магистрант кафедры программной инженерии и компьютерной техники, направление подготовки « Информатика и вычислительная техника»; comrade_1997@mail.ru.

Шиков Алексей Николаевич – Россия, 197101, Санкт-Петербург; Национальный исследовательский университет ИТМО; канд. техн. наук, доцент; доцент факультета программной инженерии и компьютерной техники; shik-off@mail.ru.

SEMANTIC NETWORK TRANSFORMATION METHOD FOR AUTOMATION OF PROGRAMMING PROBLEMS SOLUTIONS EVALUATION IN E-LEARNING

A. S. Fedorov, A. N. Shikov

ITMO University,
Saint-Petersburg, Russian Federation

Abstract. The article presents a semantic network transformation method for a programcode into an N-dimensional vector. The proposed method allows automating the quality assessment of solving programming problems in the process of e-learning. The method includes the authentic algorithms of building and converting the network. In order to determine the algorithm in the program code there is a template of this algorithm, presented in the form of a subgraph of abstract con-

cepts of the language in the semantic network, built on the basis of this code. The search for the algorithm by comparing the subgraph of the network with the template network helped to identify the BFS algorithm with a given accuracy: the cutoff threshold for the perceptron outputs is 0.85, which is based on the calculation of accuracy of the single-layer perceptron in the classification of the MNIST base equal to 88%, which confirms the effectiveness of the developed method and requires further research using machine learning methods to find the optimal value of the coordinates of the nodes of the semantic network and templates of algorithms.

Key words*:* automation, machine learning algorithms, semantic network, program code, BFS algorithm, numeric vector.

For citation: Fedorov A. S., Shikov A. N. Semantic network transformation method for automation of programming problems solutions evaluation in e-learning. *Vestnik of Astrakhan State Technical University. Series: Management, Computer Science and Informatics*. 2020;4:7-17. (In Russ.) DOI: 10.24143/2072-9502-2020-4-7-17.

REFERENCES

1. Nevostruev K. N. Obzor literatury po metodam mashinnogo obucheniia (machine learning) [Literature review on machine learning methods]. *KIO*, 2014, no. 4, pp. 19-26.

2. Gavrilova T. A., Khoroshevskii V. F. *Bazy znanii intellektual'nykh sistem* [Knowledge bases of intelligent systems]. Saint-Petersburg, Piter Publ., 2001. 394 p.

3. Rodzin S. I., Rodzina O. N. *Modeli predstavleniia znanii. Praktikum po kursu «Sistemy iskusstvennogo intellekta»: uchebnoe posobie* [Knowledge representation models. Workshop on course of artificial intelligence systems: tutorial]. Taganrog, Izd-vo IuFU, 2014. 151 p.

4. Obkhod v shirinu [Breadthways round]. *Sait viki-konspektov Universiteta ITMO*. Available at: https://neerc.ifmo.ru/wiki/index.php?title=Obkhod_v_shirinu (accessed: 20.03.2020).

5. Osowski S. *Sieci neuronowe do przetwarzania informacji* [Neural networks for information processing]. Warszawa: Oficyna wydawnicza politechniki warszawskiej, 2000. 419 p. (Rus. ed.: Osovskii S. Neironnye seti dlia obrabotki informatsii / per. s pol. I. D. Rudinskogo. M.: Finansy i statistika, 2002. 344 s.).

6. *MNIST (Baza dannykh)* [MNIST (database)]. Available at: https://ru.wikipedia.org/wiki/MNIST_(%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)#cite_note-Multideep-8 (accessed: 14.02.2020).

The article submitted to the editors 25.03.2020

INFORMATION ABOUT THE AUTHORS

Fedorov Alexander Sergeevich – Russia, 197101, Saint-Petersburg; ITMO University; Master´s Course Student of the Department of Software Engineering and Computer Technology, direction of training "Computer Science and Engineering"; comrade_1997@mail.ru.

Shikov Alexey Nikolaevich – Russia, 197101, Saint-Petersburg; ITMO University; Candidate of Technical Sciences, Assistant Professor; Assistant Professor of the Department of Software Engineering and Computer Engineering; shik-off@mail.ru.