мых показателей, т. е. номенклатуры измеряемых показателей, проводится через процедуру расширения в дополнительной области аккредитации.

При этом не учитывается следующее обстоятельство: как правило, при введении или замене новой нормативной документации не требуется изменение материальной базы, технического оснащения испытательной лаборатории и номенклатуры измеряемых показателей.

Учитывая, что в течение даже одного календарного года происходит введение новых ГОСТ и аналогичных документов, а также происходят изменения законодательства (в части охраны труда), испытательные лаборатории вынуждены ежегодно проходить процедуру расширения областей аккредитации, срок проведения которой составляет 90 рабочих дней.

В соответствии со ст. 13 Федерального закона № 426-ФЗ определен перечень факторов воздействия, который ИЛ обязана иметь в области аккредитации. Однако практика замены процедуры актуализации процедурой расширения области аккредитации создает ситуации, когда ИЛ в составе организаций, занимающихся СОУТ, большую часть времени имеет неполную область аккредитации.

При этом речь идет об организациях, которые находятся в реестре Росаккредитации и хотя бы раз подтвердили свою компетентность через документарную и выездные проверки Росаккредитации.

Для разрешения этой ситуации необходимо разработать единые рекомендации по актуализации областей аккредитации испытательных лабораторий, связанных с изменением статуса нормативных документов.

Получено 01.04.2016

Предложенный ряд мер представляет собой алгоритм, с помощью которого возможна оптимизация процедуры подтверждения компетенции испытательных лабораторий.

Библиографические ссылки

1. О специальной оценке условий труда : Федеральный закон Российской Федерации от 28 декабря 2013 г. № 426-ФЗ / [Принят Государственной Думой 23 декабря 2013 г.].

2. Руководство Р 2.2.2006-05. Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда = Guide on Hygienic Assessment of Factors of Working Environment and Work Load. Criteria and Classification of Working Conditions / [Утв. Главным государственным санитарным врачом России 29.07.05].

3. *Евтюгин Г. А.*, *Муслинкина Л. А.*, *Будников Г. К.*, *Казакова Э. Х.* // Аналитическая химия. – 1999. – Т. 54, № 4.

4. *Васильев В. П.* Аналитическая химия. – Ч. 2. Физико-химические методы анализа. – М., 1989.

5. Разъяснение Минтруда России по наиболее часто встречающимся вопросам о специальной оценке условий труда // Официальный сайт Министерства труда и социальной защиты Российской Федерации. – URL: www.rosmintrud.ru

6. Официальный сайт Федеральной службы по аккредитации (Росаккредитации). – URL: http://fsa.gov.ru/

7. Там же.

8. *Васильев В. П.* Указ. соч.

9. ГОСТ Р 51000.4–2011. Общие требования к аккредитации испытательных лабораторий / [Утв. и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 29 июня 2012 г. № 143-ст.].

УДК 004.421

П. К. Вытовтов, аспирант, ИжГТУ имени М. Т. Калашникова
Е. М. Марков, кандидат технических наук, доцент, ИжГТУ имени М. Т. Калашникова
В. А. Куликов, доктор технических наук, профессор, ИжГТУ имени М. Т. Калашникова

РАЗРАБОТКА МЕТОДА ОПРЕДЕЛЕНИЯ ВРЕМЕННОЙ СЛОЖНОСТИ АЛГОРИТМОВ С ПРИМЕНЕНИЕМ НЕЙРОННЫХ СЕТЕЙ *

Введение

Для определения сложности исполняемого кода используются различные методы: цикломатическая сложность [1], метрики Холстеда [2] и др. Одним из таких показателей является временная сложность, которая представляется в виде функции от объема входных данных и выражается с использованием нотации «О большое» [3]. Выделяют постоянное время ($O(1)$), линейное время ($O(n)$), логарифмическое время ($O(\log n)$) и др.

Данное представление сложности алгоритма и (или) программной реализации помогает оценить примерную зависимость между временем выполнения функции и объемом входных данных. Очевидно, что чем выше показатель временной сложности, тем менее пригодным для реальных задач является алгоритм и (или) его программная реализация.

Таким образом, вопрос автоматизации определения временной сложности является актуальным в рамках задачи оценки качества кода программного обеспечения [4].

Постановка и обзор задачи

В рамках работы над определением комплексного показателя качества кода программного обеспече-

* Работа выполнена в рамках проекта № 3.751.2014/к госзадания Минобрнауки на 2014–2016 гг.

ния [5] нами была поставлена задача – разработать метод автоматизированного определения временной сложности отдельной функции. В качестве входных данных используются некоторые характеристики анализируемого отрывка программного кода, а результатом является представление временной сложности с использованием нотации «О большое».

Ранее исследования в данной области проводились как в России [6], так и за рубежом [7]. Однако круг практического применения разработанных методов является достаточно ограниченным.

Следовательно, необходимо разработать универсальный метод оценки временной сложности кода программного обеспечения, который позволит автоматизированно классифицировать программные реализации алгоритмов.

Практическая реализация

Рассмотрим графы потока управления функций наиболее распространенных классов временной сложности (рис. 1).

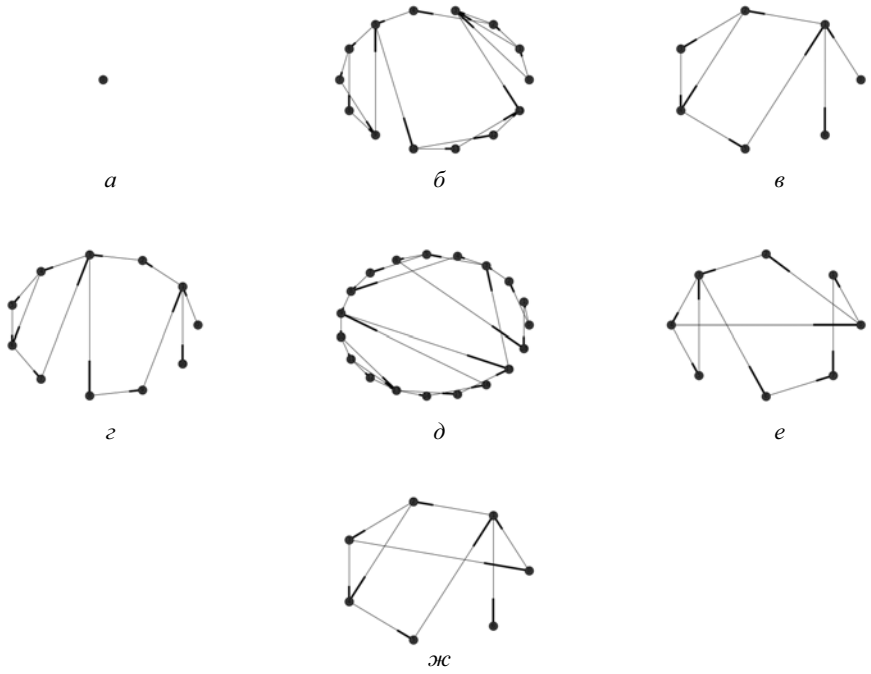

Рис. 1. Графы потока управления для типовых алгоритмов классов временной сложности: *а* – $O(1)$ – определение четности числа; *б* – $O(\log n)$ – двоичный поиск; *в* – $O(n)$ – поиск максимального элемента в массиве; *г* – $O(n^k)$ – пузырьковая сортировка; *д* – $O(n \log n)$ – пирамидальная сортировка; *е* – $O(2^n)$ – определение порядка перемножения матриц полным перебором; *ж* – $O(n!)$ – решение задачи коммивояжера полным перебором

Из рисунка видно, что графы для показателей временной сложности $O(n)$ и $O(\log n)$ схожи (рис. 1, *б*, *в*). Вторую группу схожих по структуре графов образуют функции с временной сложностью $O(n\log n)$, $O(n^k)$, $O(2^n)$ и $O(n!)$ (рис. 1, *г–ж*). Отдельно можно выделить класс сложности $O(1)$ (рис. 1, *а*), где кольца в графе отсутствуют полностью. Таким образом, первым критерием разделения программных функций на классы временной сложности является количество и вложенность колец в графе потока управления.

Рассмотрим условия завершения циклов в задаче двоичного поиска (рис. 2) и в задаче поиска максимального значения в массиве (рис. 3). Видно, что в случае линейного времени используется только операция инкрементации, а в случае логарифмического – добавляется операция деления. Очевидно, что для сложности $O(n)$ может быть добавлена операция вычитания, а для $O(\log n)$ – операции умножения и определения остатка от деления. Таким же образом во второй группе можно отделить показатель $O(n\log n)$ от остальных. Следовательно, вторым критерием разделения функций по классам временной сложности является наличие (или отсутствие) операций умножения и деления.

```
1: while (first < last) {
2:     int mid = first + (last – first) / 2;
3:     if (value <= array[mid]) last = mid;
4:     else first = mid + 1;
5: }
```

Рис. 2. Основной цикл процедуры двоичного поиска

```
1: for (index = 1; index < length; ++index) {
       ...
6: }
```

Рис. 3. Основной цикл процедуры поиска максимального значения в массиве

Сравним варианты циклов (в данном случае рекурсивный вызов функции мы также рассматриваем как цикл) функций классов сложности $O(n^k)$ (рис. 4) и $O(n!)$ (рис. 5). Видно, что ключевой особенностью

алгоритмов со сложностью $O(n^k)$ является ряд вложенных циклов, количество итераций которых может быть приведено к одинаковому значению. В свою очередь, у алгоритмов со сложностью $O(n!)$ количество итераций в каждом вложенном цикле уменьшается на единицу, пока не достигнет одной итерации в цикле.

```
1: for (i = 0; i < length-1; ++i) {
2:     for (j = 0; j < length-i-1; ++j) {
           ...
8:     }
9: }
```

Рис. 4. Циклы пузырьковой сортировки

```
1: void salesman_problem(int towns) {
2:     int index;
3:     for (index=0; index < towns; ++index) {
4:         if (towns-1 > 1) salesman_problem(towns-1);
5:     }
6: }
```

Рис. 5. Функция решения задачи коммивояжера полным перебором

Оставшиеся программные реализации, которые не удовлетворяют перечисленным выше условиям, будем относить к классу сложности $O(2^n)$.

Таким образом, можно построить дерево принятия решений для выполнения классификации функций (рис. 6).

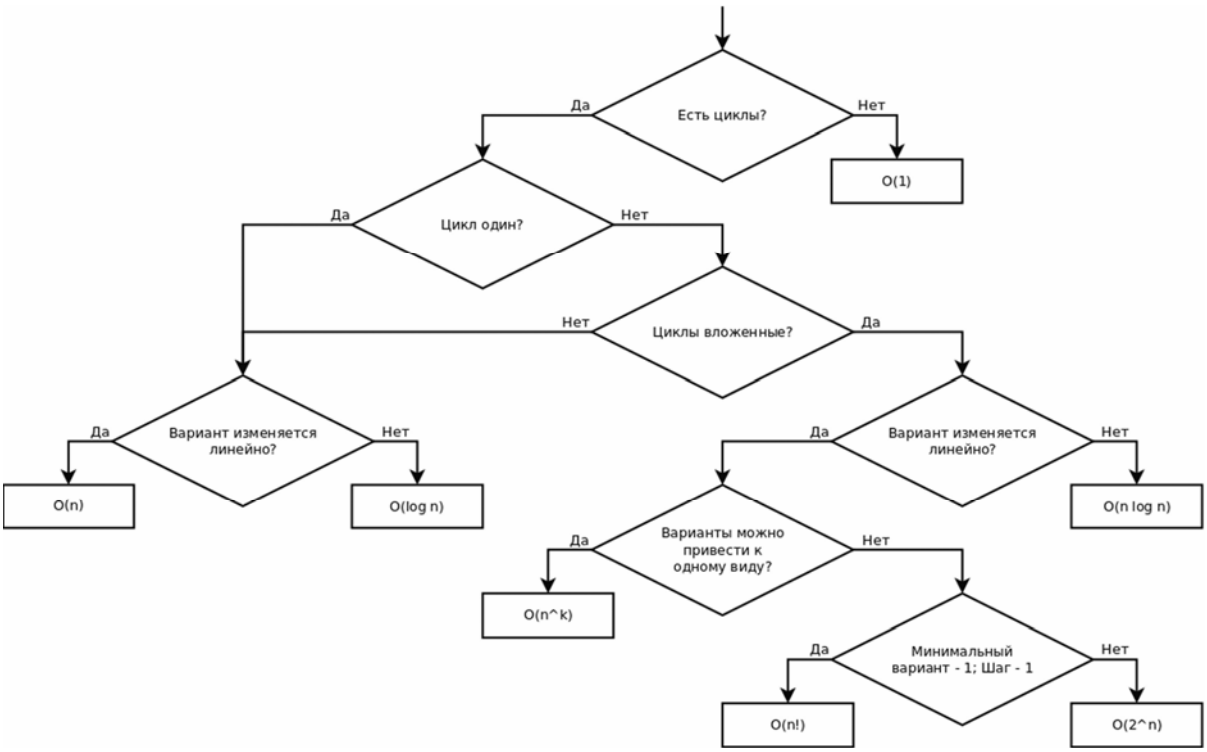

Рис. 6. Дерево принятия решений классификации функции

Из-за сложности вычисления условий «Варианты можно привести к одному виду?» и «Минимальный вариант – 1; шаг – 1?» было принято решение применить искусственную нейронную сеть для выполнения задачи классификации.

Входной слой нейронной сети состоит из восьми нейронов и принимает следующие параметры: 1) количество колец в графе потока управления; 2) количество условий в графе потока управления; 3) количество вложенных колец в графе потока управления; 4) количество рекурсивных вызовов; 5) количество операций сложения и вычитания в условных операторах; 6) общее количество математических операций в условных операторах; 7) количество констант в условных операторах; 8) количество переменных в условных операторах. Данные параметры были выбраны исходя из указанных ранее критериев отличия показателей временной сложности алгоритмов.

Выходной слой нейронной сети состоит из семи нейронов и выполняет классификацию функций на следующие группы: $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^k)$, $O(2^n)$ и $O(n!)$.

Скрытый слой нейронной сети выполняет непосредственную классификацию программных функций на основе логистической регрессии.

Программа классификации была реализована на языке программирования *Python* с использованием библиотеки *TensorFlow* [8]. Исходный код реализации свободно распространяется по лицензии *GNU GPLv*3 [9].

Экспериментальные данные

Обучение нейронной сети проводилось на 168 различных функциях [10, 11]. Дальнейшая проверка на 85 тестовых функциях показала, что среднее значение точности обучения нейронной сети составляет 0,84 ± 0,03. Данный показатель был полу-

чен на основе ста независимых обучений нейронной сети с последующим усреднением точности предсказаний класса функций и расчетом среднеквадратического отклонения. При этом разделение всего множества функций на обучающую и тестовую выборки на каждой итерации проводилось случайным образом.

Выведем результаты предсказания для трех случайно выбранных функций из обучающей и тестовой выборок (см. табл.). Из таблицы видно, что коэффициент предсказываемого класса временной сложности значительно отличается от остальных коэффициентов, что подтверждает корректность выбранных входных параметров.

Результаты работы нейронной сети

Название функции	Класс	Коэффициент принадлежности к классу						
		$O(1)$	$O(\log n)$	$O(n)$	$O(n\log n)$	$O(n^k)$	$O(2^n)$	$O(n!)$
hash_table_free_entry	O(1)	0,28	0,12	0,16	0,11	0,11	0,11	0,11
array_deletion	O(n)	0,01	0,04	0,90	0,01	0,05	0,003	0,003
print_matrix	O(n^k)	0,04	0,17	0,28	0,03	0,40	0,04	0,04

Заключение

Предложенный в статье метод оценки временной сложности программных реализаций алгоритмов, использующий искусственную нейронную сеть, может быть применен в различных задачах автоматизации анализа кода программного обеспечения.

На основе данного метода была составлена программа на языке программирования *Python*, выполняющая оценку временной сложности программных реализаций согласно ее семи основным классам. Текущее значение точности оценки составляет $0{,}84 \pm 0{,}03$.

Дальнейшая работа будет направлена на улучшение формализованного представления программного кода и на увеличение точности предсказания класса временной сложности программной реализации алгоритма.

Библиографические ссылки

1. *Thomas J. McCabe.* A Complexity Measure // IEEE Transactions On Software Engineering. – Vol. SE-2, No. 4. – December 1976.

2. *Холстед М. Х.* Начала науки о программах. – М. : Финансы и статистика, 1981.

3. *Michael Sipser.* Introduction to the Theory of Computation // Course Technology Inc, 2006.

4. *Vytovtov P. K.*, *Markov E. M.*, *M. Aiman Al Akkad.* Analysis of Software Code Metrics for Defining Their Priority for Cocol's Metric // Приборостроение в XXI веке – 2014. Интеграция науки, образования и производства : сб. материалов X Всерос. науч.-техн. конф. с междунар. участием (Ижевск, 12–14 нояб. 2014 г.). – Ижевск : Изд-во ИжГТУ имени М. Т. Калашникова, 2015. – 615 с.

5. Там же.

6. *Курмангалеев Ш. Ф.* Методы оптимизации Си/Си++-приложений, распространяемых в биткоде LLVM с учетом специфики оборудования // Труды Института системного программирования РАН. – 2013. – Т. 24. – С. 127–144.

7. *Ravichandhran Madhavan*, *Viktor Kuncak.* Symbolic Resource Bound Inference for Functional Programs // Computer Aided Verification (CAV). – 2014.

8. TensorFlow – an Open Source Software Library for Machine Intelligence. – URL: https://www.tensorflow.org/ (дата обращения: 17 марта 2016 г.).

9. Osanwe/time-complexity. – URL: https://github.com/osanwe/time-complexity (дата обращения: 17 марта 2016 г.).

10. Fragglet/c-algorithms: A library of common data structures and algorithms written in C. – URL: https://github.com/fragglet/c-algorithms (дата обращения: 17 марта 2016 г.)

11. Big-O Algorithm Complexity Cheat Sheet. – URL: http://bigocheatsheet.com/ (дата обращения: 17 марта 2016 г.)

Получено 04.04.2016