ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

УДК 004.891

МЕТОДЫ И АЛГОРИТМЫ ИДЕНТИФИКАЦИИ ФРАГМЕНТОВ ПРОГРАММНЫХ СИСТЕМ ДЛЯ ФОРМИРОВАНИЯ РЕКОМЕНДАЦИЙ ПО ПОВЫШЕНИЮ ИХ БЫСТРОДЕЙСТВИЯ

А. В. Горчаков, ассистент кафедры корпоративных информационных систем института информационных технологий РТУ МИРЭА, Москва, Россия;
orcid.org/0000-0003-1977-8165, e-mail: gorchakov@mirea.ru

Инновации в архитектуре серверов позволили создавать гетерогенные вычислительные платформы для решения специализированных задач. Существует потребность в ускорении программных систем на основе возможностей, предоставляемых гетерогенной вычислительной платформой, на которой производятся развёртывание и запуск программной системы, с целью достижения наилучшей производительности программ при выполнении специализированных вычислений. ***Целью данного исследования*** *является разработка методов и алгоритмов идентификации фрагментов программ в процессе интеллектуального статического анализа для вынесения рекомендаций, следование которым позволит повысить быстродействие фрагментов программных систем. Результатами исследования являются: новый способ преобразования программ в векторные представления на основе цепей Маркова, построение которых производится для деревьев абстрактного синтаксиса и графов «определение-использование»; алгоритм поиска фрагментов в деревьях абстрактного синтаксиса по программе-примеру на основе метода ближайшего соседа и меры Йенсена – Шеннона; методика идентификации фрагментов программных систем для вынесения рекомендаций по повышению быстродействия программных систем, основанная на формировании базы программ-примеров и вариантов повышения их быстродействия при помощи ускорителей, доступных на гетерогенной вычислительной платформе, с последующим применением разработанного алгоритма поиска фрагментов в деревьях абстрактного синтаксиса.*

Ключевые слова: *статический анализ, анализ программного кода, алгоритм классификации, цепи Маркова, деревья абстрактного синтаксиса.*

DOI: 10.21667/1995-4565-2023-86-96-109

Введение

Компьютеризация и цифровизация всех сфер общества приводит к расширению круга задач, стоящих перед разработчиками программных систем (ПС). Для ускорения и упрощения разработки ПС широко применяются и активно развиваются инструменты статического анализа программ. В результате последних разработок в этой области были предложены методы и алгоритмы интеллектуального статического анализа для задач поиска ошибок [1], уязвимостей [2], нарушений стандартов программирования [3], а также для задач восстановления названий идентификаторов при обратной разработке ПС [4, 5], определения функциональных возможностей [6] и авторства [7] по коду ПС.

Инновации в архитектуре серверов позволили создавать гетерогенные вычислительные платформы для решения специализированных задач [8]. В гетерогенных платформах центральный процессор общего назначения дополняется специализированными ускорителями,

такими как графические процессоры (англ. Graphics Processing Unit, GPU) [9], интегральные схемы специального назначения (англ. Application-Specific Integrated Circuit, ASIC), программируемые логические интегральные схемы (англ. Field-Programmable Gate Array, FPGA) [10]. Существует потребность в ускорении ПС на основе возможностей, предоставляемых гетерогенной вычислительной платформой, на которой производятся развёртывание и запуск ПС, с целью достижения наилучшей производительности ПС при выполнении специализированных вычислений [11, 12]. В этой связи актуальна разработка методов и алгоритмов идентификации фрагментов ПС в процессе их статического анализа для вынесения рекомендаций по переработке фрагментов ПС с целью повышения их быстродействия. Следование таким рекомендациям позволит повысить производительность фрагментов и, как следствие, приведёт к повышению быстродействия ПС.

Ряд инструментов интеллектуального статического анализа текстов программ [1-4, 7] основан на применении алгоритмов кластеризации и классификации, причём предварительно выполняется преобразование текстов программ в представления, содержащие только значимые для решения конкретной прикладной задачи признаки. Для обеспечения возможности обучения и применения искусственных нейронных сетей, метода опорных векторов и иных алгоритмов, принимающих на вход и обрабатывающих векторы, программы предварительно преобразуются в векторные представления, компоненты которых кодируют признаки программ. В результате исследований в области интеллектуального статического анализа был предложен ряд подходов к преобразованию программ в векторные представления, включая word2vec [13], code2vec [4], гистограммы кодов операций [14]. В [15] приведены результаты экспериментального исследования word2vec, code2vec, гистограмм кодов операций и иных подходов к извлечению признаков из программ и преобразованию программ в векторные представления. Согласно полученным в [15] результатам представления программ на основе цепей Маркова, построение которых производится для деревьев абстрактного синтаксиса (англ. Abstract Syntax Tree, AST), позволяют достичь наилучшего качества классификации в задаче определения реализованного алгоритма.

В данной статье рассматривается задача идентификации фрагментов текстов программ для вынесения рекомендаций по повышению быстродействия ПС. Для решения этой задачи предложена методика, основанная на предварительном формировании базы программ-примеров и вариантов повышения их быстродействия при помощи ускорителей, доступных на гетерогенной вычислительной платформе, с последующим применением алгоритма поиска фрагментов в деревьях абстрактного синтаксиса по программе-примеру. Для поиска фрагментов, обладающих потенциалом ускорения, предложен алгоритм, основанный на обходе дерева абстрактного синтаксиса с применением метода ближайшего соседа, расстояния Йенсена – Шеннона, а также нового способа преобразования программ в векторные представления на основе цепей Маркова, отличающегося от описанного в [15] способа использованием не только AST, но и графа «определение-использование» (англ. Definition-Use Graph, DU Graph), с целью сохранения информации о зависимостях по данным в векторных представлениях программ.

Методика идентификации фрагментов программ для формирования рекомендаций по повышению быстродействия ПС и задача поиска фрагментов программ

Поиск фрагментов ПС по программе-примеру (англ. Code-to-code Search) [16] может выполняться как для выявления дублирующего уже реализованную в ПС функциональность кода с целью его переработки для снижения размера артефактов сборки ПС, так и для выявления фрагментов ПС, обладающих теми же свойствами, что и искомая программа-пример.

Задача идентификации фрагментов программ для вынесения рекомендаций по повышению быстродействия ПС, функционирующих на специализированной гетерогенной вычислительной платформе, может быть сведена к задаче формирования базы программ-примеров

и вариантов повышения их быстродействия при помощи ускорителей, доступных на вычислительной платформе, а также к задаче поиска фрагментов по программе-примеру.

Предлагаемая методика идентификации фрагментов программ для вынесения рекомендаций по повышению быстродействия ПС включает следующие шаги.

1. Формирование базы реализаций алгоритмов на языке программирования ПС.

2. Трансляция реализаций алгоритмов в представления, пригодные для запуска на каждом из ускорителей, доступных на специализированной гетерогенной вычислительной платформе, где развёртываются и эксплуатируются ПС.

3. Оценка времени работы реализаций алгоритмов на каждом из ускорителей, доступных на рассматриваемой специализированной гетерогенной вычислительной платформе.

4. Дополнение базы реализаций алгоритмов коэффициентами, вычисленными на основе оценок производительности алгоритмов на доступных ускорителях.

5. Выполнение поиска фрагментов по программе-примеру в текстах программ анализируемых ПС для каждой из реализаций алгоритмов в базе алгоритмов.

6. Создание отчёта о перспективах ускорения ПС. При формировании отчёта используются результаты поиска фрагментов по примеру для каждой реализации алгоритма из базы реализаций, сформированной на шаге 1. В отчёт включаются найденные фрагменты ПС, а также прогнозируемые коэффициенты их ускорения, полученные на шагах 2 – 4.

Рассматриваемая задача поиска фрагментов программ по программе-примеру представлена совокупностью (t, q, A), где $t \in T$ – AST для статического анализа, $q \in T$ – AST, представляющее собой поисковый запрос, $A = \{a_1, a_2, \cdots, a_n\}$ – конечное множество из n правильных ответов, то есть таких фрагментов t, которые в действительности соответствуют поисковому запросу q, причём $\forall a_i \in A : a_i \in T$, а T – множество допустимых AST.

При решении данной задачи целью является построение отображения $g : T \times T \to \mathbb{T}$, ставящего в соответствие анализируемому дереву синтаксиса t и запросу q множество из k фрагментов $P = \{p_1, p_2, \cdots, p_k\}$, найденных по запросу q, причём $P \in \mathbb{T}$, где $\mathbb{T}$ – множество всех множеств деревьев абстрактного синтаксиса T, $\forall p_i \in P : p_i \in T$, а для оценки качества k результатов поиска множество P сравнивается с множеством правильных ответов A.

Найденные k результатов поиска $p_1, p_2, \cdots, p_k$ могут быть истинно положительными (англ. True Positive, TP), $\mathrm{TP} = \left|\{p \in P : p \in A\}\right|$, или ложно положительными (англ. False Positive, FP), $\mathrm{FP} = \left|\{p \in P : p \notin A\}\right|$. Ответы, не вошедшие в множество из k результатов поиска P, при этом присутствующие в множестве правильных ответов A, называют ложно отрицательными (англ. False Negative, FN), $\mathrm{FN} = \left|\{a \in A : a \notin P\}\right|$. Для оценки качества результатов поиска P на основе метрик TP, FP, FN используются точность для k результатов (англ. Precision), полнота (англ. Recall), F1 мера (англ. F1 Score) [17]:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{k} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = \frac{\left|\{p \in P : p \in A\}\right|}{\left|\{p \in P : p \in A\}\right| + \left|\{p \in P : p \notin A\}\right|}, \tag{1}$$

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = \frac{\left|\{p \in P : p \in A\}\right|}{\left|\{p \in P : p \in A\}\right| + \left|\{a \in A : a \notin P\}\right|}, \tag{2}$$

$$\mathrm{F1\ Score} = 2\left(\frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}\right). \tag{3}$$

При решении задачи идентификации фрагментов программ для вынесения рекомендаций по повышению быстродействия ПС, функционирующей на специализированной гетероген-

ной вычислительной платформе, поиск фрагментов по примеру выполняется для идентификации кода, ускоренная версия которого повышает быстродействие по сравнению с оригинальной версией во столько же раз, во сколько растёт быстродействие при ускорении программы-примера. Процедура поиска фрагментов ПС по программе-примеру повторяется для каждой программы-примера, включённой в базу программ-примеров и вариантов повышения их быстродействия при помощи ускорителей, доступных на специализированной гетерогенной вычислительной платформе.

Методы и алгоритмы поиска фрагментов ПС по примеру

В рассматриваемой задаче поиска фрагментов в дереве абстрактного синтаксиса t по примеру q множество правильных ответов A, как правило, содержит малое число фрагментов, соответствующих программе-примеру q. Следовательно, задача сводится к разработке алгоритма, устойчивого к несбалансированным данным.

Алгоритм 1 – Построение графа «определение-использование» для AST языка Python
Algorithm 1 – Construction of the Definition-Use Graph for an AST of the Python language

Вход: n – анализируемая вершина дерева абстрактного синтаксиса;
p – вершина дерева абстрактного синтаксиса, контекст анализа;
s – словарь, содержащий значения видимых переменных;
d – флаг, определяющий, находится ли вершина n в левой части присваивания.

1. **Определить** отображение g, ставящее в соответствие вершине AST её тип.
2. $E = \varnothing$.
3. **Если** $g(n) = \text{Name}$, **выполнить**:
4. **Если** $d = 1$, **выполнить** $s[n.\text{id}] \leftarrow p$.
5. **Иначе**, **если** $n.\text{id} \in s$, **выполнить** $E \leftarrow E \bigcup \{(s[n.\text{id}], p)\}$.
6. **Если** $g(n) \in \{\text{For}, \text{AsyncFor}\}$, **выполнить**:
7. $E \leftarrow E \cup$ **Алгоритм 1** ($n.\text{target}, n, s, 1$).
8. **Для каждого** поддерева $a \in \{n.\text{iter}\} \bigcup n.\text{body} \bigcup n.\text{orelse}$ **выполнять**:
9. $E \leftarrow E \cup$ **Алгоритм 1** ($a, n, s, 0$).
10. **Конец цикла**.
11. **Возвратить** E.
12. **Если** $g(n) = \text{Assign}$, **выполнить**:
13. $E \leftarrow E \cup$ **Алгоритм 1** $(n.\text{value}, n, s, 0)$.
14. **Для каждой** цели присваивания $t \in n.\text{targets}$ **выполнять**:
15. $E \leftarrow E \cup$ **Алгоритм 1** $(t, n.\text{value}, s, 1)$.
16. **Конец цикла**.
17. **Возвратить** E.
18. **Если** $g(n) \in \{\text{FunctionDef}, \text{AsyncFunctionDef}\}$, **выполнить**:
19. $s \leftarrow \varnothing$, записать в s пустой словарь.
20. **Для каждого** аргумента $a \in n.\text{args}$ **выполнить** $s[a.\text{arg}] \leftarrow a$.
21. **Для каждого** потомка c вершины n **выполнять**:
22. $E \leftarrow E \cup$ **Алгоритм 1** (c, n, s, d).
23. **Конец цикла**.
24. **Возвратить** E.

Для решения задачи (t, q, A) предлагается предварительно преобразовать программу-пример q к цепи Маркова, построение которой производится для AST и графа «определение-использование». Алгоритм 1 описывает последовательность действий для построения графа

«определение-использование» на основе AST программы на языке Python [18]. Алгоритм построения цепи Маркова для направленного графа, которым является как AST, так и граф «определение-использование», описан в [15, 19].

Алгоритм 1 выполняет обход AST языка программирования Python, построение которого предварительно осуществляется средствами стандартной библиотеки [18]. Квадратные скобки в Алгоритме 1 используются для обозначения записи элемента в словарь по ключу, точка используется для обозначения доступа к полям структур, являющихся частью AST, по их именам. В процессе обхода AST Алгоритмом 1 на основе информации об определении и использовании переменных формируется множество связей между вершинами графа «определение-использование». После этого выполняется построение цепи Маркова для AST (рисунок 1, *б*) и графа «определение-использование» (рисунок 1, *в*) согласно алгоритму, описанному в работах [19, 20].

После преобразования программы-примера в цепь Маркова, построение которой производится и для AST, и для графа «определение-использование» (рисунок 1, *в*), Алгоритмом 2 выполняются извлечение из AST фрагментов и их сравнение с программой-примером. Фрагменты в процессе их извлечения из AST преобразуются в представления на основе цепей Маркова при помощи Алгоритма 2. Сравнение векторных представлений программ на основе цепей Маркова осуществляется на основе меры Йенсена – Шеннона (англ. Jensen – Shannon Divergence, JSD), применявшейся в [19, 20] при кластеризации программ:

$$\mathrm{JSD}\left(\vec{v}_i,\vec{v}_j\right)=\frac{1}{2}\sum_{k=1}^{m}v_{il}\log_2\frac{v_{il}}{\frac{1}{2}\left(v_{il}+v_{jl}\right)}+\frac{1}{2}\sum_{k=1}^{m}v_{jl}\log_2\frac{v_{jl}}{\frac{1}{2}\left(v_{il}+v_{jl}\right)}, \quad (4)$$

где $\vec{v}_i$ и $\vec{v}_j$ – сравниваемые векторы, полученные в результате преобразования матриц смежности цепей Маркова, принадлежащих пространству $\mathbb{R}^{h\times h}$, к векторам, принадлежащим пространству $\mathbb{R}^m$ при $m=h^2$; v_{il} – l-я компонента вектора $\vec{v}_i$; v_{jl} – l-я компонента вектора $\vec{v}_j$.

Как показано на рисунке 1, представление программы на основе цепи Маркова для AST и для графа «определение-использование», используемое в данной работе (рисунок 1, *в*), отличается от представления программы на основе цепи Маркова только для AST (рисунок 1, *б*), исследованного в [15, 19, 20], наличием рёбер, выделенных на рисунке 1,*в* синим пунктиром. Веса этих рёбер обозначают вероятность наличия связи между парами элементов абстрактного синтаксиса в графе «определение-использование» программы, построение которого выполняется при помощи Алгоритма 1.

Для обеспечения возможности настройки влияния весов переходов между вершинами цепей Маркова для AST (такие переходы обозначены чёрными сплошными линиями на рисунке 1, *в*) и весов переходов между вершинами цепей Маркова для графа «определение-использование» (такие переходы обозначены пунктирными линиями на рисунке 1, *в*) на значения, которые принимает мера (4), предлагается модифицировать меру (4) следующим образом:

$$\mathrm{D}\left(\vec{v}_{i1},\vec{v}_{j1},\vec{v}_{i2},\vec{v}_{j2},\omega_1,\omega_2\right)=\omega_1\mathrm{JSD}\left(\vec{v}_{i1},\vec{v}_{j1}\right)+\omega_2\mathrm{JSD}\left(\vec{v}_{i2},\vec{v}_{j2}\right), \quad (5)$$

где $\vec{v}_{i1}\in\mathbb{R}^m\in\mathbb{R}^m$ и $\vec{v}_{j1}\in\mathbb{R}^m$ – сравниваемые векторные представления программ, полученные в результате преобразования *i*-й и *j*-й программы в цепи Маркова для AST (рисунок 1, *б*), с последующим построением матриц смежности цепей Маркова $\mathbf{m}_{i1}\in\mathbb{R}^{h\times h}$ и $\mathbf{m}_{j1}\in\mathbb{R}^{h\times h}$ и вложением матриц смежности в пространство $\mathbb{R}^m$, где $m=h^2$; $\vec{v}_{i2}\in\mathbb{R}^m$ и $\vec{v}_{j1}\in\mathbb{R}^m$ – сравниваемые векторные представления *i*-й и *j*-й программ, полученные в результате преобразования программ в цепи Маркова для графа «определение-использование» (см. переходы, обозначенные пунктирными линиями на рисунок 1, *в*), с последующим построением матриц смежности цепей Маркова $\mathbf{m}_{i2}\in\mathbb{R}^{h\times h}$ и $\mathbf{m}_{j2}\in\mathbb{R}^{h\times h}$ и вложением матриц смежности в век-

торное пространство $\mathbb{R}^m$, где $m = h^2$, ω_1 и ω_2 – коэффициенты для регулировки влияния переходов различного типа в цепях Маркова для AST и графа «определение-использование» на итоговую меру расстояния (5), основанную на JSD (4).

Алгоритм 2, выполняющий поиск фрагментов AST по программе-примеру q, также представленному AST, выполняет сравнение представления программы в виде цепи Маркова для AST и для графа «определение-использование» с комбинациями фрагментов линейного участка AST длиной $z = |q.\text{body}|$, представленными тем же образом, на основе меры (5).

Визуализации, показанные на рисунке 1, были получены при помощи инструмента для визуализации графов graphviz [21].

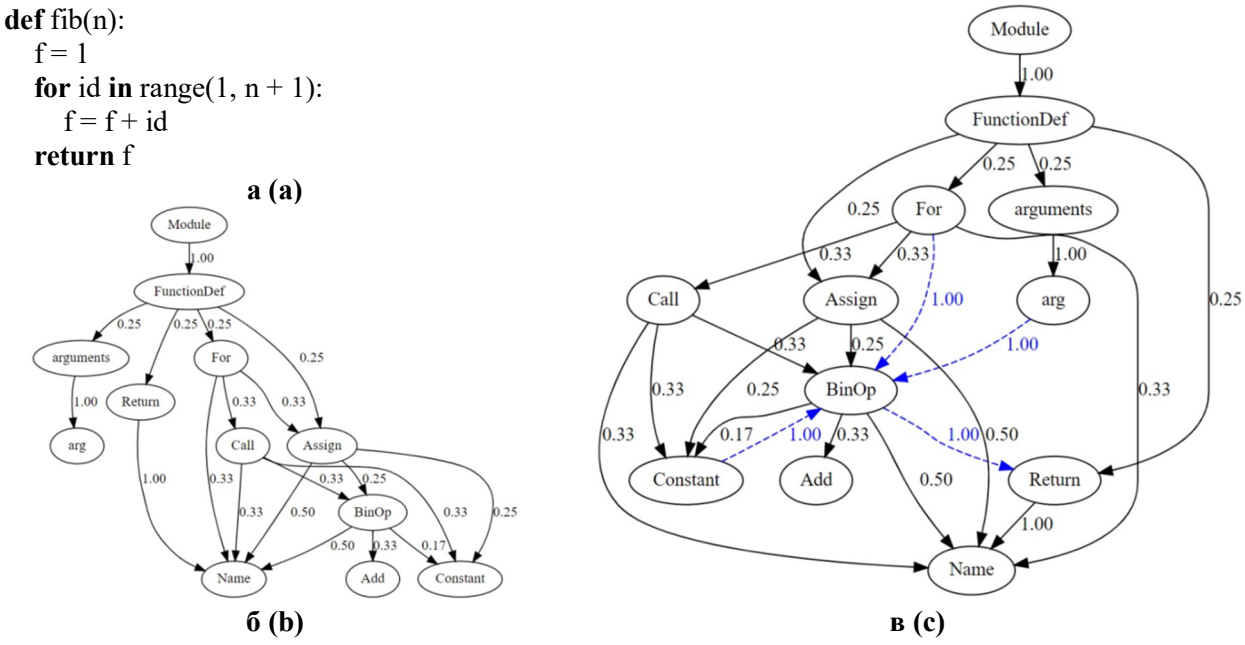


Рисунок 1 – Программа и её представления: а – простая программа на языке Python; б – цепь Маркова для AST этой программы; в – цепь Маркова для AST и для графа «определение-использование», синими пунктирными линиями обозначены рёбра зависимостей по данным между элементами синтаксиса

Figure 1 – Program and its representations: a – a simple program written in Python; b – AST-based Markov chain for this program; c – Markov chain based on an AST and a definition-use graph, dashed edges with arrows mean data dependencies among syntactic elements of an AST

Результатом работы Алгоритма 2 является k найденных фрагментов дерева абстрактного синтаксиса f, наилучшим в смысле (5) образом соответствующих запросу q. Оценка качества работы Алгоритма 2 при наличии правильных ответов A, представленных заранее выделенными фрагментами синтаксического дерева f, может быть дана по формулам (1), (2) и (3). В предложенной методике идентификации фрагментов программ для вынесения рекомендаций по повышению быстродействия ПС, предполагающей составление базы реализаций алгоритмов и коэффициентов их ускорения на различных специализированных ускорителях, Алгоритм 2 используется для поиска фрагментов AST по примерам реализаций алгоритмов из базы реализаций с коэффициентами их ускорения на доступных ускорителях. Результаты работы Алгоритма 2 используются для вынесения рекомендаций по ускорению ПС, в отчёт включаются найденные фрагменты и прогнозируемые выигрыши в производительности при реализации фрагментов на специализированных ускорителях.

Алгоритм 2 – Поиск фрагментов AST Python по программе-примеру
Algorithm 2 – Search for Python AST fragments by a program example

Вход: t – AST, в котором выполняется поиск фрагментов,
q – AST программы-примера,
k – число фрагментов, наиболее похожих в смысле (4) на q,
ω_1 – вес расстояния между цепями Маркова для AST сравниваемых программ,
ω_2 – вес расстояния между цепями Маркова для графа «определение-использование».

1. $z = |q.\text{body}|$ число вершин AST в теле функции, цикла или иного оператора q.
2. **Построить** матрицы смежности цепей Маркова $\mathbf{m}_{q1} \in \mathbb{R}^{h \times h}$ и $\mathbf{m}_{q2} \in \mathbb{R}^{h \times h}$ для q.
3. **Построить** векторы $\vec{v}_{q1} \in \mathbb{R}^m$ и $\vec{v}_{q2} \in \mathbb{R}^m$, где. $m = h^2$, на основе матриц $\mathbf{m}_{q1}$ и $\mathbf{m}_{q2}$.
4. $R = \varnothing$, пустое множество результатов поиска.
5. **Для каждой** вершины n дерева синтаксиса t **выполнять**:
6. $s = n.\text{body}$, множество вершин AST в теле n.
7. **Если** $s \neq \varnothing$, **выполнить**:
8. **Для каждой** комбинации $c \in \binom{s}{z}$ потомков вершины n **выполнять**:
9. $f = \text{Module}\left(\text{body} = \{c\}\right)$, поместить в f оператор AST с телом, включающим c.
10. **Построить** матрицы смежности цепей Маркова $\mathbf{m}_{f1} \in \mathbb{R}^{h \times h}$ и $\mathbf{m}_{f2} \in \mathbb{R}^{h \times h}$ для f.
11. **Построить** векторы $\vec{v}_{f1} \in \mathbb{R}^m$ и $\vec{v}_{f2} \in \mathbb{R}^m$, где $m = h^2$, на основе $\mathbf{m}_{f1}$ и $\mathbf{m}_{f2}$.
12. $d = \text{D}\left(\vec{v}_{q1}, \vec{v}_{f1}, \vec{v}_{q2}, \vec{v}_{f2}, \omega_1, \omega_2\right)$, вычислить расстояние (5) между q и f.
13. $R \leftarrow R \bigcup \{(d, f)\}$.
14. **Конец цикла**.
15. **Конец если**.
16. **Конец цикла**.
17. **Возвратить** подмножество R из k результатов с наименьшими расстояниями d.

Численный эксперимент

Оценка качества векторных представлений программ на основе цепей Маркова для AST и для графа «определение-использование» (англ. Definition-Use Graph, DU-граф) в задаче классификации программ для определения реализованного алгоритма проводилась на наборе [19], включающем 14070 программ, решающих уникальные задачи, автоматически сгенерированные системой Цифровой ассистент преподавателя (ЦАП) [22]. При оценке качества векторных представлений программ использовалась методология, описанная в [15].

Из набора, включавшего 14070 программ, каждой из которых была поставлена в соответствие метка класса $y \in \{1..11\}$, обозначающая тип решаемой программой задачи, 10 раз случайно выбирались 100 программ, которые затем использовались для обучения и оценки качества классификаторов различных типов методом перекрёстной проверки по k блокам при $k = 5$ на основе многоклассовых версий метрик точности (1), полноты (2) и F1-меры (3) с макро-усреднением [15]. Векторные представления программ на основе цепей Маркова для AST (рисунок 1, *б*), а также векторные представления на основе цепей Маркова для AST и DU-графа (рисунок 1, *в* и Алгоритм 1) использовались для обучения и оценки качества классификаторов на основе метода *k* ближайших соседей (англ. k-Nearest Neighbours, KNN), метода опорных векторов (англ. Support Vector Machine, SVM), случайного леса (англ. Random

Forest, RF), искусственной нейронной сети (англ. Multi-Layer Perceptron, MLP). Параметры KNN, SVM, RF, MLP были установлены равными значениям, указанным в [15].

Результаты оценки качества представлены на рисунке 2.

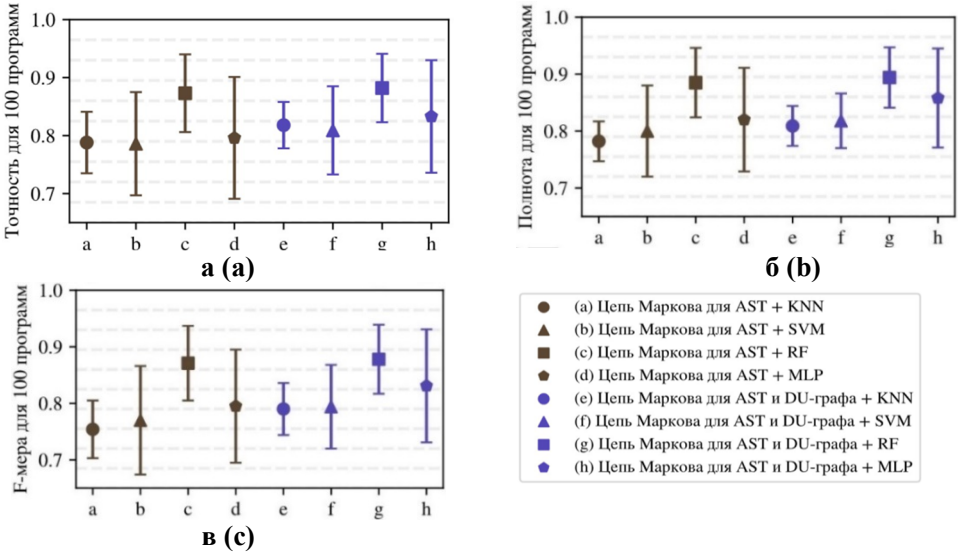


Рисунок 2 – Результаты оценки качества векторных представлений программ на основе цепей Маркова для AST (выделены чёрным цветом) и для графа «определение-использование» (выделены синим цветом) на основе: а – точности; б – полноты; в – F1-меры

Figure 2 – Results of quality assessments of program embeddings based on Markov chains built for AST (highlighted in black) and Definition-Use Graph (highlighted in blue) according to: a – precision, b – recall; c – F1-measure

Вертикальные отрезки на рисунке 2 обозначают интервалы, ограниченные сверху значением $x+\sigma$ и снизу значением $x-\sigma$, где x – среднее значение качества классификатора, а σ – среднеквадратическое отклонение. Маркерами разных форм выделено среднее значение качества классификаторов KNN, SVM, RF, MLP. Согласно рисунку 2, векторные представления программ на основе цепей Маркова, построение которых производится и для AST, и для DU-графов (рисунок 1, *в*), позволяют достичь лучшей точности классификации по сравнению с векторными представлениями на основе цепей Маркова только для AST (рисунок 1, *б*). При этом качество классификаторов на основе KNN и MLP улучшается значительно.

Оценка качества работы Алгоритма 2 проводилась на ряде задач (t, q, A) поиска фрагментов AST t по программе-примеру q, также представленном AST. Сведения о задачах $П_1$, $П_2$, $П_3$, код которых открыт, представлены в таблице 1. Тексты анализируемых программ t программ-примеров q для задач $П_1$, $П_2$, $П_3$ опубликованы на GitHub [23]. Код программы t в задаче $П_3$ был взят из системы ЦАП [22]. Результаты поиска k фрагментов в AST t по примеру q сравнивались с множеством правильных ответов A с использованием (1), (2) и (3). Параметр k Алгоритма 2 был установлен равным $|A|+2$, где $|A|$ – число правильных ответов, зависящее от задачи [23].

Таблица 1 – Сведения о рассмотренных задачах

Table 1 – Information about the considered problems

Задача (t, q, A)	$П_1$	$П_2$	$П_3$
Число строк кода в t	28	45	157
Число типов вершин AST в t	19	23	41
Всего фрагментов в t	146	116	289
Правильных фрагментов	4	3	5
Неправильных фрагментов	142	113	284
Значение k	6	5	7

Качество результатов поиска зависит от значений весовых коэффициентов ω_1 и ω_2 в формуле (5), являющихся параметрами Алгоритма 2. При этом ω_1 обозначает величину влияния весов рёбер цепей Маркова для AST на функцию расстояния (5), а ω_2 обозначает величину влияния весов рёбер цепей Маркова для графа «определение-использование» на (5). В таблице 2 приведены результаты численной оценки качества работы Алгоритма 2 согласно (1), (2) и (3) при различных значениях параметров ω_1 и ω_2.

Таблица 2 – Результаты численной оценки качества Алгоритма 2 на рассмотренных задачах
Table 2 – The results of numerical assessments of Algorithm 2 in the considered problems

Задача	Модель	Параметры	Точность (1)	Полнота (2)	F1-мера (3)
Π_1	AST	$\omega_1 = 1, \omega_2 = 0$	0.17	0.25	0.20
	DU	$\omega_1 = 0, \omega_2 = 1$	0.50	0.75	0.60
	AST+DU	$\omega_1 = 1, \omega_2 = 1$	**0.67**	**1.00**	**0.80**
Π_2	AST	$\omega_1 = 1, \omega_2 = 0$	0.57	0.80	0.67
	DU	$\omega_1 = 0, \omega_2 = 1$	0.00	0.00	0.00
	AST+DU	$\omega_1 = 1, \omega_2 = 1$	**0.71**	**1.00**	**0.83**
Π_3	AST	$\omega_1 = 1, \omega_2 = 0$	0.20	0.33	0.25
	DU	$\omega_1 = 0, \omega_2 = 1$	**0.60**	**1.00**	**0.75**
	AST+DU	$\omega_1 = 1, \omega_2 = 1$	**0.60**	**1.00**	**0.75**

AST в таблице 2 обозначает, что при заданных значениях параметров ω_1 и ω_2 при сравнении программ мерой (5) используется только цепь Маркова для AST [15] (рисунок 1, *б*). DU в таблице 2 обозначает, что при сравнении программ используется цепь Маркова для графа «определение-использование» (англ. Definition-Use Graph, DU-граф). AST+DU в таблице 2 обозначает, что при заданных значениях параметров ω_1 и ω_2 используется цепь Маркова как для AST, так и для графа «определение-использование» (рисунок 1, *в*). Жирным в таблице 2 выделены лучшие результаты оценки качества в конкретной задаче и лучшая модель.

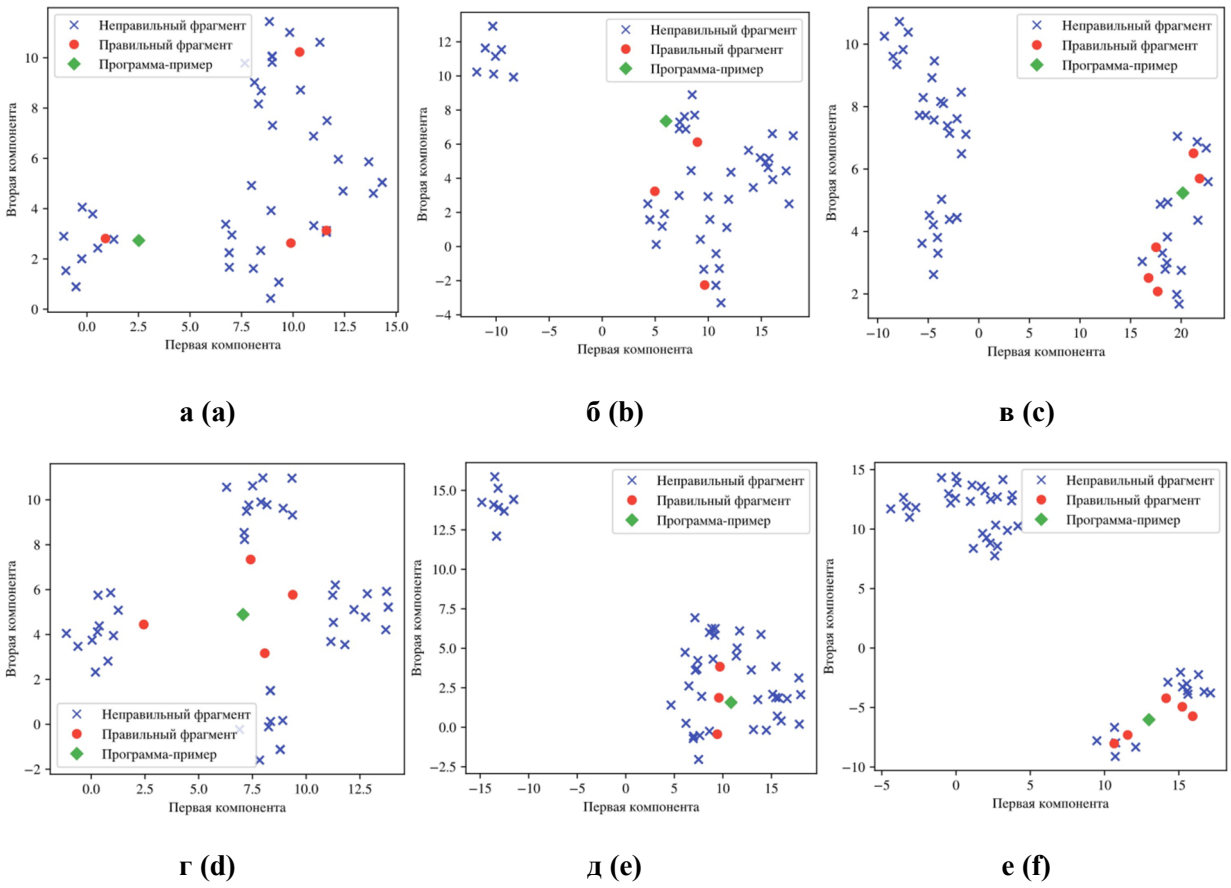
Как показано в таблице 2, представления программ на основе цепей Маркова для AST и для графа «определение-использование» (Алгоритм 1 и рисунок 1, *в*) позволяют достичь лучшей точности (1), полноты (2) и F1-меры (3) в рассмотренных задачах поиска фрагментов в AST программ по сравнению с представлениями программ на основе цепей Маркова только для AST [15] или с представлениями на основе цепей Маркова только для графа «определение-использование». При этом качество поиска фрагментов в смысле (1), (2) и (3) для различных значений параметров ω_1 и ω_2 зависит от рассматриваемой задачи [23].

Для визуальной оценки качества векторных представлений фрагментов программ на основе цепей Маркова для AST [15] (рисунок 1, *б*) и векторных представлений на основе цепей Маркова для AST и графа «определение-использование» (рисунок 1, *в*) было выполнено снижение размерности векторных представлений с целью вложения многомерных векторов в векторное пространство $\mathbb{R}^2$, пригодное для визуализации. Для снижения размерности была использована модификация алгоритма UMAP [24] с методом оптимизации на основе градиентного спуска с адаптивным шагом Adam [25]. Полученные при помощи UMAP визуализации 50 наиболее похожих в смысле (5) на q фрагментов t для задач Π_1, Π_2, Π_3 представлены на рисунке 3.

Алгоритм UMAP выполняет снижение размерности векторов, принадлежащих пространству высокой размерности, на основе построения графа сходств ближайших соседей каждого вектора [24].

Как показано на рисунке 3, векторные представления фрагментов t, в действительности соответствующие программе-примеру q (рисунок 3 в виде круглых точек), размещены алго-

ритмом UMAP в пространстве низкой размерности ближе к вектору программы-примера q (рисунок 3 в виде ромба) в случае использования цепей Маркова как для AST, так и для графа «определение-использование» (рисунок 3, *г, д, е*), по сравнению с цепями Маркова, построение которых производится только для AST [15] (рисунок 3, *а*, *б*, *в*).



а (a) **б (b)** **в (c)**

г (d) **д (e)** **е (f)**

Рисунок 3 – Визуализации векторных представлений фрагментов t на основе цепей Маркова для AST для: а – $П_1$; б – $П_2$ в –, $П_3$ визуализации векторных представлений фрагментов t на основе цепей Маркова для AST и графа «определение-использование» для: г – $П_1$; д – $П_2$; е – $П_3$

Figure 3 – Visualizations of vector representations of fragments of t based on Markov chains constructed for ASTs for: a – $П_1$; b – $П_2$; c – $П_3$ visualizations of vector representations of fragments of t based on Markov chains for AST and definition-use graph for: d – $П_1$; e – $П_2$; – $П_3$

На основе Алгоритма 1, Алгоритма 2, а также алгоритма построения цепи Маркова для направленного графа, описанного в [15] (рисунок 1), было разработано расширение для редактора Visual Studio Code, позволяющее выполнять поиск фрагментов AST по программе-примеру, выделять найденные фрагменты с показом коэффициентов потенциального прироста производительности, формируя таким образом рекомендации по ускорению (рисунок 4). Коэффициенты прироста производительности в статическом анализаторе, примеры работы которого показаны на рисунке 4, вычисляются после трансляции программ-примеров в представления, пригодные для запуска на специализированных ускорителях, в результате выполнения замеров производительности программ согласно описанной в данной статье методике идентификации фрагментов ПС для вынесения рекомендаций по повышению быстродействия ПС.

```
38  def predict_slow(theta, x):
39      prediction_outputs = []
40      for example in x:
            NumPy ускорит код в 1.25 раз, PyTorch ускорит код в 3.25 раз
41          prediction_output = 0
42          for i in range(len(example)):
43              prediction_output = prediction_output + example[i] * theta[i]
44          prediction_outputs.append(prediction_output)
        NumPy ускорит код в 3.26 раз, PyTorch ускорит код в 10.26 раз
45      prediction_activations = []
46      for i in range(len(prediction_outputs)):
47          prediction_activations.append(1 / (1 + math.exp(1 - prediction_outputs[i])))
48      return prediction_activations
```

Рисунок 4 – Визуализация результатов статического анализа
Figure 4 – Visualizations of the results of static analysis

Заключение

В результате проведенного исследования была предложена методика идентификации фрагментов программ для вынесения рекомендаций по повышению быстродействия ПС на специализированной гетерогенной вычислительной платформе, сводящаяся к формированию базы реализаций алгоритмов на языке программирования, использующемся в ПС, замерам производительности оригинальных реализаций алгоритмов и реализаций, ускоренных при помощи доступных на гетерогенной вычислительной платформе ускорителей, выполнению поиска фрагментов AST по примеру, созданию отчёта по перспективам ускорения ПС с использованием полученных численных оценок.

Для выполнения поиска фрагментов AST по примеру с учётом специфики предметной области был предложен Алгоритм 1 построения графа «определение-использование» для программ на языке Python, а также Алгоритм 2 поиска фрагментов в AST. Впервые описан способ преобразования программ в векторные представления на основе цепей Маркова, построение которых производится для AST и графа «определение-использование» (рисунок 1, *в*), позволяющий достичь лучшей точности (1), полноты (2) и F1-меры (3) в рассмотренных тестовых задачах по сравнению с цепями Маркова для AST (рисунок 1, *б*), описанными в [15]. Для подтверждения эффективности нового способа преобразования программ в векторные представления были получены численные оценки качества классификаторов (рисунок 2), а также при помощи алгоритма UMAP были получены визуализации, показанные на рисунке 3. На основе описанных в статье методов и алгоритмов было разработано средство статического анализа программ для редактора Visual Studio Code (рисунок 4).

Тем не менее, рассмотренные в статье методы и алгоритмы поддерживают только подмножество всего синтаксиса языка программирования Python. Так, например, Алгоритм 1 при построении графа «определение-использование» не учитывает изменения данных при помощи расширений оператора присваивания (AugAssign), таких как +=, *=, **= и других, изменения полей классов, содержимого словарей, списков. Дальнейшие исследования могут быть направлены на доработку Алгоритма 1 для поддержки всего синтаксиса языка Python, на разработку алгоритмов построения графа «определение-использование» для других языков, широко используемых в индустрии при разработке ПС, или на адаптацию Алгоритма 1 и Алгоритма 2 для работы с AST, не зависящими от языка программирования.

Библиографический список

1. **Shi K., Lu Y., Chang J., Wei Z.** PathPair2Vec: An AST path pair-based code representation method for defect prediction. Journal of Computer Languages. 2020, vol. 59, p. 100979.

2. **Li X., Wang L., Yang Y., Chen Y.** Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning. Applied Sciences. 2020, vol. 10, p. 1692.

3. **Loriot B., Madeiral F., Monperrus M.** Styler: learning formatting conventions to repair Checkstyle violations. Empirical Software Engineering. 2022, vol. 27, no. 6, p. 149.

4. **Alon U., Zilberstein M., Levy O., Yahav E.** code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages. 2019, vol. 3, p. 40.

5. **Kovalenko V., Bogomolov E., Bryksin T., Baccheli A.** PathMiner: A Library for Mining of Path-Based Representations of Code. Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp. 13-17.

6. **Marcus A., Maletic J.I.** Identification of high-level concept clones in source code. Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE 2001), pp. 107-114.

7. **Kurtukova A., Romanov A., Shelupanov A.** Source Code Authorship Identification Using Deep Neural Networks. Symmetry. 2020, vol. 12, no. 12, p. 2044.

8. Глинских В.Н., Дудаев А.Р., Нечаев О.В. Высокопроизводительные гетерогенные вычисления CPU-GPU в задаче электрического каротажа нефтегазовых скважин. Вычислительные технологии. 2017, Т. 22, №. 3, с. 16-31.

9. **Мирзоян Д.** Основные аспекты применения GPGPU систем. Современные информационные технологии и ИТ-образование. 2011, №. 7, с. 988-994.

10. **Andrianova E., Sovietov P., Tarasov I.** Hardware acceleration of statistical data processing based on fpgas in corporate information systems. Proceedings of the 2020 2nd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA), pp. 669-671.

11. **Zhou X., Canady R., Bao S., Gokhale A.** Cost-effective hardware accelerator recommendation for edge computing. Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20).

12. **Mokri P., Hempstead M.** Early-stage automated accelerator identification tool for embedded systems with limited area. Proceedings of the 39th International Conference on Computer-Aided Design (ICAAD 2020), pp. 1-8.

13. **Qiao Y., Zhang W., Du X., Guizani M.** Malware Classification Based on Multilayer Perception and Word2Vec for IoT Security. ACM Transactions on Internet Technology. 2021, vol. 22, p. 10.

14. Damásio T., Canesche N., Pacheco V., Botacin M., da Silva A.F., Pereira F.M.Q. A Game-Based Framework to Compare Program Classifiers and Evaders. Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization, 2023, pp. 108-121.

15. **Gorchakov A.V., Demidova L.A., Sovietov P.N.** Analysis of Program Representations Based on Abstract Syntax Trees and Higher-Order Markov Chains for Source Code Classification Task. Future Internet. 2023, vol. 15, no. 9, p. 314.

16. **Yu Y., Huang Z., Shen G., Li W., Shao Y.** ASTENS-BWA: Searching partial syntactic similar regions between source code fragments via AST-based encoded sequence alignment. Science of Computer Programming. 2022, vol. 222, p. 102839.

17. **Silveira T., Zhang M., Lin X., Liu Y., Ma S**. How good your recommender system is? A survey on evaluations in recommendation. International Journal of Machine Learning and Cybernetics. 2019, vol. 10, pp. 813-831.

18. AST – Abstract Syntax Trees [Электронный ресурс]. Python 3.10.5 Documentation. URL: https://docs.python.org/3/library/ast.html (дата обращения: 23.06.2022)

19. **Demidova L.A., Andrianova E.G., Sovietov P.N., Gorchakov A.V.** Dataset of Program Source Codes Solving Unique Programming Exercises Generated by Digital Teaching Assistant. Data, 2023, vol. 8, no. 6, p. 109.

20. **Демидова Л.А., Советов П.Н., Горчаков А.В.** Кластеризация представлений текстов программ на основе цепей Маркова // Вестник Рязанского государственного радиотехнического университета. 2022. №. 81. С. 51-64. DOI: 10.21667/1995-4565-2022-81-51-64.

21. **Gansner E.R., North S.C.** An open graph visualization system and its applications to software engineering // Software: practice and experience. 2000. vol. 30, no. 11. pp. 1203-1233.

22. **Sovietov P.N., Gorchakov A.V.** Digital Teaching Assistant for the Python Programming Course. Proceedings of the 2022 2nd International Conference on Technology Enhanced Learning in Higher Education (TELE). pp. 272-276.

23. **Gorchakov A.V.** Sample programs for debugging intelligent static analysis tool which uses program representations based on Markov chains constructed from AST [Электронный ресурс]. GitHub Gist. URL: https://gist.github.com/worldbeater/9c36c43b64a41d6dfee965e24152c1b5 (дата обращения: 02.10.2023).

24. **Demidova L.A., Gorchakov A.V.** Fuzzy Information Discrimination Measures and Their Application to Low Dimensional Embedding Construction in the UMAP Algorithm. Journal of Imaging. 2022, vol. 8, no. 4, p. 113.

25. **Kingma D.P., Ba J.** Adam: A method for stochastic optimization. arXiv. 2014, arXiv:1412.6980.

UDC 004.891

METHODS AND ALGORITHMS FOR IDENTIFYING PROGRAM FRAGMENTS FOR MAKING RECOMMENDATIONS WITH THE AIM TO INCREASE THE SPEED OF SOFTWARE SYSTEMS

A. V. Gorchakov, assistant of the Department of Corporate Information Systems, Institute of Information Technologies, RTU MIREA, Moscow, Russia;
orcid.org/0000-0003-1977-8165, e-mail: gorchakov@mirea.ru

Innovations in server architecture made it possible to create heterogeneous computing platforms for solving specialized problems. There is a need to accelerate software systems based on the capabilities provided by the heterogeneous computing platform on which the software system is deployed and run, in order to achieve the best software performance when performing specialized calculations. ***The aim of this research*** *is the development of methods and algorithms for identifying program fragments during the process of intelligent static analysis in order to make recommendations for reworking a software system in order to increase its performance. The results of the research are: a new method for converting programs into vector representations based on Markov chains constructed for abstract syntax trees and definition-use graphs; algorithm for searching fragments in abstract syntax trees by a program example based on the k-nearest neighbors method and the Jensen-Shannon distance function; a technique for identifying program fragments to make recommendations for improving the performance of software systems, based on the collection of a database of example programs and options for increasing their performance using accelerators available on a heterogeneous computing platform, followed by the use of the developed algorithm for fragments search in abstract syntax trees.*

Keywords: *static analysis, source code analysis, classification algorithm, Markov chains, abstract syntax trees.*

DOI: 10.21667/1995-4565-2023-86-96-109

References

1. **Shi K., Lu Y., Chang J., Wei Z.** PathPair2Vec: An AST path pair-based code representation method for defect prediction. *Journal of Computer Languages.* 2020, vol. 59, p. 100979.

2. **Li X., Wang L., Yang Y., Chen Y.** Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning. *Applied Sciences.* 2020, vol. 10, p. 1692.

3. **Loriot B., Madeiral F., Monperrus M.** Styler: learning formatting conventions to repair Checkstyle violations. *Empirical Software Engineering*. 2022, vol. 27, no. 6, p. 149.

4. **Alon U., Zilberstein M., Levy O., Yahav E.** code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*. 2019, vol. 3, p. 40.

5. **Kovalenko V., Bogomolov E., Bryksin T., Baccheli A.** PathMiner: A Library for Mining of Path-Based Representations of Code. *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 13-17.

6. **Marcus A., Maletic J.I.** Identification of high-level concept clones in source code. *Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pp. 107-114.

7. **Kurtukova A., Romanov A., Shelupanov A.** Source Code Authorship Identification Using Deep Neural Networks. *Symmetry*. 2020, vol. 12, no. 12, p. 2044.

8. **Glinskikh V.N., Dudaev A.R., Nechaev O.V.** High-performance CPU – GPU heterogeneous computing in resistivity logging of oil and gas wells. *Journal of Computational Technologies*. 2017, vol. 22, no. 3, pp. 16-31.

9. **Mirzoyan D.** Osnovnye aspekty primeneniya GPGPU sistem. *Modern Information Technologies and IT-Education*. 2011, no. 7, pp. 988-994 (in Russian).

10. **Andrianova E., Sovietov P., Tarasov I.** Hardware acceleration of statistical data processing based on fpgas in corporate information systems. *Proceedings of the 2020 2nd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*, pp. 669-671.

11. **Zhou X., Canady R., Bao S., Gokhale A.** Cost-effective hardware accelerator recommendation for edge computing. *Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*.

12. **Mokri P., Hempstead M.** Early-stage automated accelerator identification tool for embedded systems with limited area. *Proceedings of the 39th International Conference on Computer-Aided Design (ICAAD 2020)*, pp. 1-8.

13. **Qiao Y., Zhang W., Du X., Guizani M.** Malware Classification Based on Multilayer Perception and Word2Vec for IoT Security. *ACM Transactions on Internet Technology*. 2021, vol. 22, p. 10.

14. **Damásio T., Canesche N., Pacheco V., Botacin M., da Silva A.F., Pereira F.M.Q.** A Game-Based Framework to Compare Program Classifiers and Evaders. *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization.* 2023, pp. 108-121.

15. **Gorchakov A.V., Demidova L.A., Sovietov P.N.** Analysis of Program Representations Based on Abstract Syntax Trees and Higher-Order Markov Chains for Source Code Classification Task. *Future Internet*. 2023, vol. 15, no. 9, p. 314.

16. **Yu Y., Huang Z., Shen G., Li W., Shao Y.** ASTENS-BWA: Searching partial syntactic similar regions between source code fragments via AST-based encoded sequence alignment. *Science of Computer Programming*. 2022, vol. 222, p. 102839.

17. **Silveira T., Zhang M., Lin X., Liu Y., Ma S**. How good your recommender system is? A survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*. 2019, vol. 10, pp. 813-831.

18. *AST – Abstract Syntax Trees [Digital Resource]. Python 3.10.5 Documentation*. URL: https://docs.python.org/3/library/ast.html (accessed at: 23.06.2022).

19. **Demidova L.A., Andrianova E.G., Sovietov P.N., Gorchakov A.V.** Dataset of Program Source Codes Solving Unique Programming Exercises Generated by Digital Teaching Assistant. *Data*, 2023, vol. 8, no. 6, p. 109.

20. **Demidova L.A., Sovietov P.N., Gorchakov A.V.** Clustering of Program Source Text Representations Based on Markov Chains. *Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta*. 2022, no. 81, pp. 51-64. (in Russian). DOI 10.21667/1995-4565-2022-81-51-64

21. **Gansner E.R., North S.C.** An open graph visualization system and its applications to software engineering. *Software: practice and experience*. 2000. vol. 30, no. 11, pp. 1203-1233.

22. **Sovietov P.N., Gorchakov A.V.** Digital Teaching Assistant for the Python Programming Course. *Proceedings of the 2022 2nd International Conference on Technology Enhanced Learning in Higher Education (TELE)*. pp. 272-276.

23. **Gorchakov A.V.** *Sample programs for debugging intelligent static analysis tool which uses program representations based on Markov chains constructed from AST* [Digital Resource]. GitHub Gist. URL: https://gist.github.com/worldbeater/9c36c43b64a41d6dfee965e24152c1b5 (accessed at: 02.10.2023).

24. **Demidova L.A., Gorchakov A.V.** Fuzzy Information Discrimination Measures and Their Application to Low Dimensional Embedding Construction in the UMAP Algorithm. *Journal of Imaging*. 2022, vol. 8, no. 4, p. 113.

25. **Kingma D.P., Ba J.** Adam: A method for stochastic optimization. *arXiv*. 2014, arXiv:1412.6980.