УДК 004.4:378.147 https://doi.org/10.26160/2541-8637-2026-17-294-298

АЛГОРИТМ ЛЕГКОВЕСНОГО ОБНАРУЖЕНИЯ СХОЖЕСТИ ПРОГРАММНОГО КОДА В СИСТЕМАХ АВТОМАТИЧЕСКОЙ ПРОВЕРКИ ЗАДАНИЙ

Лященко Б.С.
Мытищинский филиал Московского государственного технического университета имени Н.Э. Баумана, Мытищи

Ключевые слова: автоматическая проверка программ, антиплагиат, схожесть кода, токенизация, *n*-граммы, коэффициент Жаккара, Docker, микросервис.
Аннотация. Предлагается легковесный алгоритм выявления схожих программных решений для локальной системы автоматической проверки лабораторных работ. Алгоритм основан на нормализации исходного кода, токенизации, построении *n*-грамм и вычислении коэффициента Жаккара. Описана микросервисная архитектура на Python FastAPI и Docker, обеспечивающая безопасный запуск программ в изолированных контейнерах и интеграцию модуля антиплагиата в общий конвейер проверки. Рассмотрена схема экспериментальной оценки, включающая подбор порога, анализ распределений схожести и исследование ложных срабатываний, вызванных шаблонным кодом.

LIGHTWEIGHT ALGORITHM FOR DETECTING SOURCE CODE SIMILARITY IN AUTOMATED ASSIGNMENT ASSESSMENT SYSTEMS

Lyashchenko B.S.
Mytischi Branch of Bauman Moscow State Technical University, Mytischi

Keywords: automated assessment, anti-plagiarism, code similarity, tokenization, *n*-grams, Jaccard similarity, Docker, microservice.
Abstract. The paper proposes a lightweight source-code similarity detection algorithm for a local automated assessment system used in programming laboratories. The method relies on normalization, tokenization, *n*-gram construction and Jaccard similarity. A microservice architecture based on Python FastAPI and Docker is described, with emphasis on isolated student programs execution and a anti-plagiarism local module integration into the grading workflow. An evaluation protocol including threshold selection and false-positive analysis is outlined.

Автоматическая проверка программных заданий позволяет преподавателю получать единообразные результаты тестирования, быстро выдавать обратную связь и поддерживать большие учебные потоки. Однако проверка только функциональной корректности не исключает академического плагиата: студент может отправить модифицированную копию чужой работы, которая успешно пройдет тесты. В результате возникает необходимость в дополнительном модуле, выявляющем аномально похожие решения без ручного просмотра всего массива отправок.

Для локальной инфраструктуры кафедры важны ограничения по вычислительным ресурсам, конфиденциальности и сопровождению. Поэтому оправдан подход, в котором модуль антиплагиата работает на собственном сервере, не использует крупные модели машинного обучения и не передает исходные тексты во внешние сервисы. В статье рассматривается именно такой вариант: легковесный алгоритм предварительного скрининга, который не заменяет

экспертного решения преподавателя, а формирует ранжированный список подозрительных пар решений.

Наиболее известными средствами поиска заимствований в программных работах являются MOSS и JPlag [1-3]. MOSS предоставляет сервис определения схожести программ и подчеркивает, что высокий показатель сходства сам по себе не является доказательством плагиата [1]. JPlag, напротив, ориентирован на локальный запуск и попарное сравнение множества программ, что особенно важно для университетских установок, где недопустим вывод студенческих работ за пределы внутренней сети [2].

Научная база подобных систем опирается на исследования клонов программ и методы сравнения шинглов документов. В обзорных работах по клонам выделяются типы I-III, соответствующие различиям в форматировании, переименовании идентификаторов и локальных вставках либо удалениях [4]. Для учебного антиплагиата именно эти типы наиболее показательны. Фундаментальные работы по resemblance и winnowing [5] показывают, что сравнение множеств n-грамм после нормализации остается практичным компромиссом между устойчивостью к поверхностным изменениям и вычислительной экономичностью.

Следовательно, для кафедральной системы рационально использовать локальный алгоритм класса «нормализация – токены – n-граммы – множественная метрика». Такой подход уступает сложным семантическим методам в глубине анализа, но выигрывает в прозрачности, простоте внедрения и контролируемости вычислительных затрат.

Система автоматической проверки строится по микросервисному принципу и включает API-шлюз на базе FastAPI, очередь фоновых задач, модуль Docker-runner, хранилище решений и результатов, а также модуль локального антиплагиата. После загрузки работы API выполняет валидацию данных и создает задачу в очереди. Рабочий процесс запускает программу в отдельном контейнере, собирает результаты тестирования и передает исходный код в сервис статического анализа схожести.

Контейнеризация играет ключевую роль в обеспечении безопасности. Согласно документации Docker, контейнеры обладают изолированными файловой системой, сетевым пространством и деревом процессов и могут ограничиваться по памяти и вычислительной квоте. Поэтому каждая отправка исполняется в отдельной контролируемой среде. Модуль антиплагиата не исполняет пользовательский код и работает только с текстом программы, что позволяет отделить статический анализ от среды выполнения пользовательских программ.

Алгоритм состоит из пяти этапов. Сначала выполняется нормализация исходного кода: удаляются комментарии, повторяющиеся пробелы и различия форматирования; при необходимости литералы и пользовательские идентификаторы заменяются маркерами LIT и ID. Затем нормализованный текст преобразуется в последовательность токенов $T = (t_1, t_2, ..., t_m)$. На следующем шаге строится множество n-грамм $G_n(T) = \{(t_i, ..., t_{i+n-1})\}$, где n выбирается из диапазона 4-7; в работе используется $n = 5$ как практический компромисс.

Чтобы уменьшить влияние шаблонного кода, вводится фильтрация часто встречающихся n-грамм. Если $\frac{DF(g)}{M} > \theta$, где $DF(g)$ – число решений с n-граммой g, а M – число работ по заданию, такая n-грамма исключается из сравнения. После фильтрации для двух работ A и B вычисляется коэффициент Жаккара:

$$J(A,B) = |G_A \cap G_B| / |G_A \cup G_B|.$$

Если $J(A, B) \geq \tau$, пара включается в отчет как потенциально заимствованная. Порог τ подбирается экспериментально; для задач первичного скрининга практически разумным оказывается диапазон 0,5-0,6.

С вычислительной точки зрения алгоритм остается легковесным. Нормализация, токенизация и построение n-грамм выполняются линейно по длине программы. Попарное сравнение N работ одного задания имеет квадратичную сложность, но при кафедральных объемах данных (десятки и сотни отправок) остается приемлемым. При необходимости возможно ускорение за счет индексации и пересчета только для новых решений. Сравнительные свойства рассматриваемых подходов приведены в таблице 1.

Табл. 1. Сопоставление существующих и предлагаемого подходов

Подход	Режим	Требования	Особенности
MOSS	внешний сервис	доступ в Интернет	высокая практическая точность, но внешняя публикация результатов по URL
JPlag	локальный запуск	Java и отдельный инструмент	развитый специализированный анализ попарных сходств
Предлагаемый алгоритм	локальный модуль	Python-стек кафедры	простая интеграция, прозрачность и низкая вычислительная стоимость

Для внедрения алгоритма предлагается двухэтапная схема оценки. Сначала выполняется модельный эксперимент, позволяющий подобрать рабочие значения n и τ и исследовать влияние фильтрации общих n-грамм. Затем проводится проверка на реальном архиве студенческих работ с экспертной разметкой преподавателя. Для каждой пары решений фиксируются коэффициент Жаккара, время расчета, число общих n-грамм и принадлежность пары к одному из классов: обычная, спорная или явно заимствованная. Основными метриками являются precision, recall и $F1$. На рисунке 1 показано примерное распределение попарной схожести в модельном эксперименте.

Подозрительные пары смещены в область высоких значений J, однако существует зона перекрытия с обычными решениями. Это перекрытие объясняется наличием шаблонного кода, канонических способов решения простых задач и малым объемом отдельных программ. На рисунке 2 приведена precision-recall кривая для выбора порога.

При малых τ полнота близка к единице, но точность снижается из-за множества ложных срабатываний. При росте τ отчет становится компактнее, но

увеличивается риск пропуска модифицированных копий. Для учебного процесса обычно предпочтителен режим повышенной точности, поскольку преподавателю важно получить короткий и интерпретируемый список пар для ручной проверки.

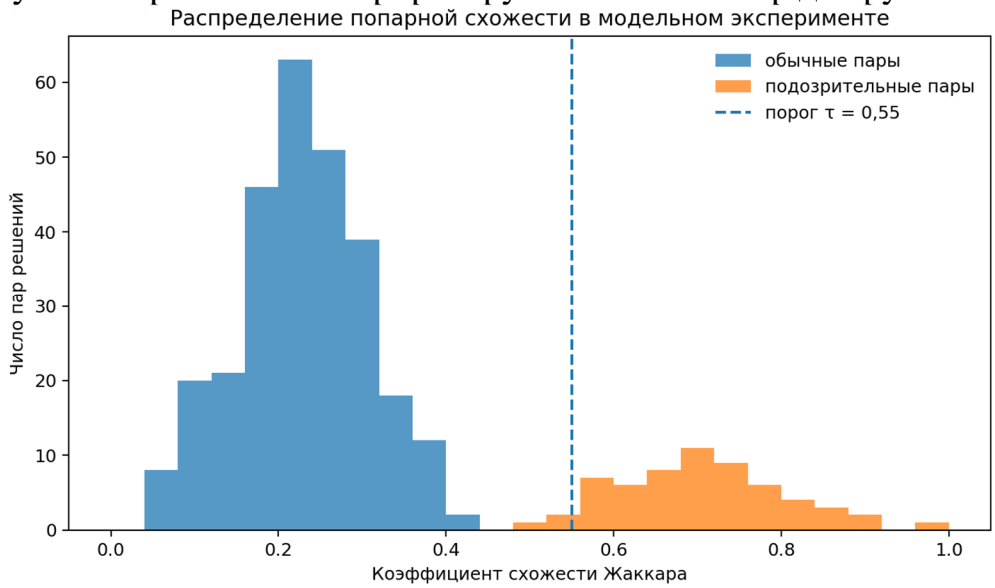

Рис. 1. Пример распределения коэффициента Жаккара для обычных и подозрительных пар

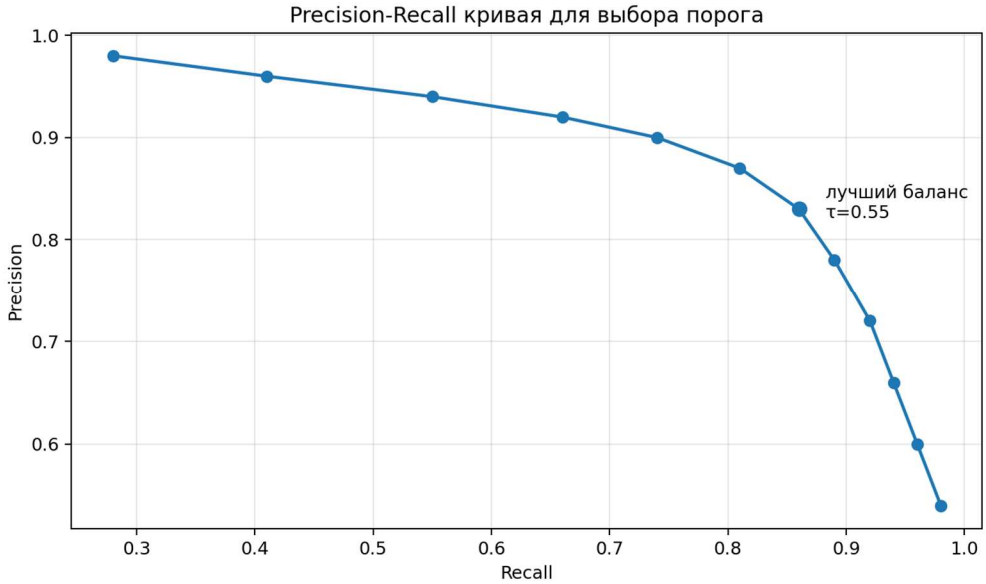

Рис. 2. Precision-Recall кривая для выбора порога детектирования

Наиболее типичные ложные срабатывания возникают в четырех случаях: при наличии общего каркасного кода, при канонических решениях простых задач, при очень коротких программах и при сравнении повторных отправок одного и того же студента. Для первых двух случаев эффективна фильтрация частых *n*-грамм; для третьего вводится минимальная длина программы; для четвертого сравнения между личными версиями исключаются из анализа. Такой набор правил позволяет сохранить легковесность алгоритма и одновременно повысить его практическую полезность.

Заключение

Предложен легковесный алгоритм обнаружения схожести программного кода, предназначенный для локальной системы автоматической проверки заданий.

Алгоритм основан на нормализации, токенизации, построении *n*-грамм и вычислении коэффициента Жаккара, а его результаты интерпретируются как материал для экспертной проверки, а не как автоматическое доказательство плагиата. Показано, что такой подход хорошо сочетается с микросервисной архитектурой на Python FastAPI и Docker, где статический анализ текста программ естественным образом отделен от безопасного контейнерного исполнения пользовательского кода.

Практическая ценность подхода состоит в возможности развертывания на локальном сервере кафедры без применения крупных моделей машинного обучения и без обращения к внешним сервисам. Уже в базовом варианте предложенный алгоритм обеспечивает разумный компромисс между точностью, прозрачностью и вычислительной экономичностью.

Список литературы

1. Aiken A. Moss: A System for Detecting Software [Электронный ресурс]. – Режим доступа: https://theory.stanford.edu/~aiken/moss.
2. JPlag: Detecting Source Code Plagiarism [Электронный ресурс]. – Режим доступа: https://github.com/jplag/JPlag.
3. Prechelt L., Malpohl G., Philippsen M. Finding plagiarisms among a set of programs with JPlag // Journal of Universal Computer Science. 2002, vol. 8, no. 11, pp. 1016-1038.
4. Roy C.K., Cordy J.R. A Survey on Software Clone Detection Research // Technical Report 2007-541. Kingston: Queen’s University, 2007.
5. Schleimer S., Wilkerson D. S., Aiken A. Winnowing: local algorithms for document fingerprinting // Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. 2003, pp. 76-85.

Сведения об авторе:
Лященко Борис Сергеевич – студент.