Автоматизация проверки на плагиат: новый подход к анализу кода в цифровой образовательной платформе Мирера

Д.И. Кадина[1], А.Г. Леонов[2], Н.С. Мартынов[3], К.А. Мащенко[4], Э.А. Орлов[5], А.И. Стрекалова[6]

[1] НИЦ «Курчатовский институт» — НИИСИ, Москва, Россия, kadinadaria@mail.ru;

[2] НИЦ «Курчатовский институт» — НИИСИ, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, МПГУ, Москва, Россия, Государственный университет управления, Москва, Россия, dr.l@vip.niisi.ru;

[3] НИЦ «Курчатовский институт» — НИИСИ, Москва, Россия, nikolai.martynov@math.msu.ru;

[4] НИЦ «Курчатовский институт» — НИИСИ, Москва, Россия, МГУ им. М. В. Ломоносова, Москва, Россия, Государственный университет управления, Москва, Россия, kirill.mashchenko@niisi.ru;

[5] НИЦ «Курчатовский институт» — НИИСИ, Москва, Россия, eric.al.orlov@gmail.com;

[6] НИЦ «Курчатовский институт» — НИИСИ, Москва, Россия, anastasiia.strekalova@math.msu.ru

Аннотация. Задача выявления плагиата в решениях задач по программированию является высокоприоритетной при разработке цифровых образовательных платформ. Это связано с необходимостью оперативно оценивать уровень заимствования в решениях студентов, чтобы динамически формировать финальную оценку выполнения задания. Методы сравнения решений учащихся, основанные только на текстовом соотнесении их частей без привязки к характерным особенностям используемого языка программирования, зачастую не дают точных и достоверных результатов, так же как и статистические методы, основанные на машинном обучении. В данной работе предложен подход, повышающий качество выявления плагиата при блочно-кусочном сравнении студенческих решений, опирающийся на учёт синтаксических особенностей языков программирования.

Ключевые слова: цифровая образовательная платформа, ЦОП Мирера, антиплагиат, антиплагиат-анализ программного кода, блочное сопоставление.

1. Введение

Плагиат исходного кода (Source Code Plagiarism – SCP) решений обучающихся IT-специальностей с использованием цифровых образовательных платформ (ЦОП) является актуальной проблемой [1; 2]. Статистика убедительно показывает, что от 50% до 79% студентов хотя бы раз прибегали к плагиату решений во время обучения [3; 4]. Использования заимствований только увеличилось после ограничений пандемии COVID-19 [5], а также из-за стремительного развития методов искусственного интеллекта, направленных на генерацию кода для решения задач [6].

Методы решения поставленной задачи можно условно разделить на следующие категории: агрегированное сравнение и структурное сравнение.

Первая категория предполагает независимое от порядка элементов сравнение текстов и использует различные агрегатные методы для выявления фундаментальных характеристик анализируемой части. Часто к этой категории относят различные модели машинного обучения. Например, контекстное или стилистическое [6; 7] сравнение решений подразумевает агрегацию кусков решений в некоторое репрезентативное представление с последующим анализом для обнаружения сходства по конкретной исследуемой характеристике. К той же категории относятся методы, использующие векторные представления частей текста, такие как word2vec [8] и doc2vec [9], а также методы, основанные на подсчёте частот в тексте, такие как BOW [10] и TF-ISF [11]. Несмотря на заявленную эффективность приведенных выше методов, к их недостаткам можно отнести отсутствие чувствительности к структурным перестановкам частей программы и высокую склонность к ложноположительным срабатываниям. Также не учитываются синтаксические особенности языков программирования, ведь в коде одинаковые символы используются для совершенно разных выражений языка программирования. К недостаткам методов, основанных на машинном обучении, можно отнести характерные для них недетерминируемость, неинтерпретируемость и слабую предсказуемость результата, что исключает возможность детального описания сравнения, например, для визуализации в решениях, где именно

было совершено заимствование, чтобы преподаватель мог убедиться в достоверности результата работы алгоритма и исключить ложноположительное срабатывание. К тому же, статистические методы легко обходить с помощью добавления фиктивного кода, так как снижается процент значимой части при значительном добавлении «шума» в текст.

Вторая категория предполагает строгую привязку к синтаксису и взаимосвязям внутри анализируемой части текста. Многие современные инструменты обнаружения SCP измеряют сходство между частями текста в целом, но они также не способны учитывать особенности синтаксиса языка программирования и характерную блочную кодовую структуру, где, как правило, каждый блок кода отвечает за некоторую самостоятельную семантическую единицу, которая при списывании переносится с сохранением структурной целостности. Одним из популярных алгоритмов, относящихся ко второй категории, является алгоритм MOSS (Measure Of Software Similarity) [12]. Его работа основана на анализе структурного сходства через токенизацию и хеширование. Исходный код нормализуется: удаляются пробелы, комментарии, переименованные идентификаторы заменяются на стандартные. Затем код разбивается на последовательности токенов (k-граммы), которые хешируются. Используемый в этом методе алгоритм winnowing выбирает подмножество хешей, представляющих текст. Сходство программ определяется по пересечению найденных хешей. Также популярными структурными алгоритмами сравнения являются JPlag [13] и Sherlock [14]. Нужно отметить, что результаты работы таких алгоритмов значительно варьируются в зависимости от методов оценки [15]. Например, JPlag и Sherlock демонстрируют значительные различия в распределении оценок сходства для одних и тех же данных. Недостатком структурных подходов всё ещё является возможность добавления фиктивного кода в уязвимые места, характерные для конкретного алгоритма. Например, MOSSad [16] – автоматизированный фреймворк, использующий генетическое программирование для внедрения семантически нейтральных изменений, вставляет «пустые» операторы (например, объявление неиспользуемых переменных) в «спорные» с точки зрения SCP позиции, нарушая целостность k-грамм. Это приводит к резкому снижению оценок схожести в MOSS. При этом разработчики алгоритма Sherlock в ответ на подобные уязвимости заявляют, что «Если (студенты) могут заниматься плагиатом и избегать обнаружения Sherlock (или JPlag, или MOSS), то они уже могут программировать на хорошем уровне» [14]. Ещё одним подходом из второй категории является построение графов по заданному тексту. Например, был реализован алгоритм построения семантических графов, который можно адаптировать под синтаксис кода с помощью добавления специфических правил [17]. Также существует алгоритм GPLAG, строящий графы и затем оценивающий их покрытия [18].

Отдельно стоит упомянуть комбинирование различных подходов, которое предлагает использование сразу нескольких инструментов обнаружения SCP и дальнейшее формирование взвешенного результата по группе алгоритмов [19].

В настоящей работе предложен автоматический структурный метод обнаружения SCP в обучающих задачах цифровой образовательной платформы, основанный на поиске наибольших совпадающих по заданной метрике строчных блоков кода в различных языках программирования.

2. Реализация алгоритма

Перед процедурой сравнения студенческих решений на предмет наличия SCP необходимо корректно преобразовывать входные данные.

Решения задач по программированию в базе данных цифровой образовательной платформы зачастую хранятся не одним файлом, а сразу архивом из нескольких файлов, которые могут иметь совершенно различный формат, в связи с чем и возникает необходимость в их слиянии в единый текст. Далее из студенческого тексте решения необходимо изъять части шаблона, предоставленного преподавателем, так как этот текст идентичен и фиксирован в рамках одной задачи у всех обучающихся.

Для выбора алгоритмов обработки текста студенческого решения непосредственно перед процедурой сравнения, необходимо рассмотреть возможные модификации текста пользователями для потенциального обхода алгоритма выявления SCP [20]:

1. Изменение комментариев и отступов;
2. Переименование идентификаторов;
3. Изменения в декларациях элементов (например, объявление дополнительных констант, добавление фиктивных строк кода, а также изменение порядка функций и переменных);
4. Изменение операторов программы на семантические эквиваленты (например, for на while, if на switch).

До процедуры сравнения текст решения очищается от пустых строк и комментариев, специ-

фичных для конкретного языка программирования. Увеличение исходного текста за счет «лишних» строк влияет на вероятностную оценку заимствования, так же, как и незначимые пары скобок. Для решения проблемы переименования имен переменных, применяемых студентами с целью обхода алгоритма SCP, подходит метод универсализации идентификаторов в программном коде с помощью замены различных синтаксических единиц на единый заранее зафиксированный стандарт: имён переменных, названий функций и выражений, характерных для конкретного языка программирования. Так, например, для языка C++ имена переменных при обработке приводятся к единой структуре названий имен с единым началом (V) и числовым окончанием, имена функций заменяются на имена с F, имена объектов начинаются с O, а строки (string) в любом формате преобразуются в имена с S. Третья проблема (изменения в декларации элементов) решается переносом строк, в которых происходит эта декларация, в начало кодового блока. Для исключения возможности влияния на процедуру сравнения изменений операторов программы на семантические эквиваленты достаточно зафиксировать один из взаимозаменяемых операторов и преобразовать текст программы, используя только эти операторы.

Следующим этапом обработки текста решений является этап преобразования с использованием регулярных выражений для ряда выделенных случаев в различных языках программирования. Например, в языках C или C++ префиксный оператор ++n, постфиксный оператор n++ и оператор n+=1 хоть и имеют синтаксические различия, зачастую приводят к одному и тому же ожидаемому результату в рамках студенческих задач. Следовательно, подобные конструкции тоже возможно подвергнуть некоторой универсализации.

В предлагаемом алгоритме сравнения текстов решений на наличие SCP используется блочный метод построчных сравнений, основанный на определенном понятии строки в коде программы. Текст программы во многих языках представляет собой последовательность символов и лишь для программиста визуально разделен на строки. Фактически программу можно «склеить» в единую строку, сохранив семантику. Но такой текст для человека сложен для понимания и редактирования, в отличии от представления программы, когда каждая строка несет некий самостоятельный элемент алгоритма. Тогда такую «склеенную» строку необходимо разбить на фрагменты минимальных синтаксических единиц, которые будут представлять минимальные семантические единицы программы. Например, в языке C++ это возможно сделать по концу оператора, символу «;» или по защищённым конструкциям, таким как if или for, что упростит анализ кода и облегчит последующую визуализацию заимствований. Для этого в процессе разбиения необходимо сохранить взаимно однозначное соответствие между индексами разбитых строк и исходными строками для итоговой разметки списанных строк, а также агрегировать результаты анализа в соответствии с заданной вероятностной метрикой плагиата, что дает возможность видеть результат работы алгоритма отдельно для каждой строки.

Дальнейшая работа по выявлению плагиата состоит в сравнении предобработанного студенческого решения (первый файл) с кандидатом на заимствование (второй файл). Таким кандидатом могут быть эталонное решение преподавателя, что говорит о правильности стиля студенческого решения и(или) более ранее (по времени) решение другого студента, чтобы оценить уровень заимствования. При этом для каждой строки из первого сравниваемого файла осуществляется процедура поиска наиболее подходящих кандидатов среди строк второго файла. Учитывая синтаксические особенности текстов решений задач по программированию, связанные с конкретным языком программирования, осуществляется поиск соответствия между последовательными блоками строк. В частности, когда для i-й строки первого файла находится подходящая j-я строка из второго файла, соответствующая заданной метрике плагиата и превышающая установленный порог, учитывающий специфику задач, составленных преподавателем, необходимо проанализировать их ближайшие строки (i+1-ю из первого файла и j+1-ю из второго файла) на предмет совпадения. Это позволит выявить продолжение совпадения и сформировать целый блок совпадений. Процедура сравнения последующих строк продолжается до тех пор, пока не будет нарушено пороговое условие совпадения для очередной пары строк или не завершится обработка текста. В каждом цикле, при выполнении условия для i+k-ой и j+k-ой строк, найденная пара блоков из k строк первого и второго файла добавляется в качестве кандидатов на соответствие с вычисленной усреднённой вероятностной метрикой $((p_1 + \ldots + p_k) / k)$ по строкам блока. В процессе последовательного формирования совпадающего блока сохраняются все промежуточные блоки, что дает возможность разрешать конфликты в процедуре блочного сопоставления.

Сортировка полученных кандидатов осуществляется с учётом блочного и вероятностного приоритета, характерного для задачи выявления SCP. «Жадность» алгоритма поиска совпадений состоит в нахождении наиболее длинных

совпадающих блоков, которые приоритизируются по усреднённой вероятности для блоков одинаковой длины. Итоговая сортировка кандидатов, представляющих собой пары соотнесённых блоков, осуществляется по следующему принципу: приоритетной является длина кандидатов, а для блоков одинаковой длины сортировка производится по убыванию вероятности плагиата, рассчитанной по заданной метрике.

Процедура разметки строк и сопоставления наиболее совпадающих блоков основывается на алгоритме, который гарантирует сходимость, в том числе в процессе разрешения конфликтов, а также выявляет наиболее длинные блоки плагиата.

Сопоставление найденных блоков из первого файла ко второму осуществляется независимо от порядка, но с учётом приоритета, установленного на этапе предварительной сортировки. Кандидаты добавляются в множество сопоставлений по убыванию после сортировки, начиная с длинных блоков, обладающих наибольшей вероятностной метрикой. При обработке блоков длиной k производится сортировка внутри кандидатов по убыванию усредненной вероятностной метрики строк, описанной выше. В случае отсутствия конфликта в виде пересечений с какимилибо уже занятыми строками, кандидат добавляется в результирующее множество сопоставлений плагиата. Если же возникает конфликт с ранее добавленными строками, то кандидат отбрасывается, и вместо него рассматривается более короткий блок, если таковой имеется дальше в отсортированном множестве кандидатов и не пересекается с уже занятыми строками. На Рисунке 1 изображён пример разрешения конфликта между кандидатами при сравнении двух текстовых файлов.

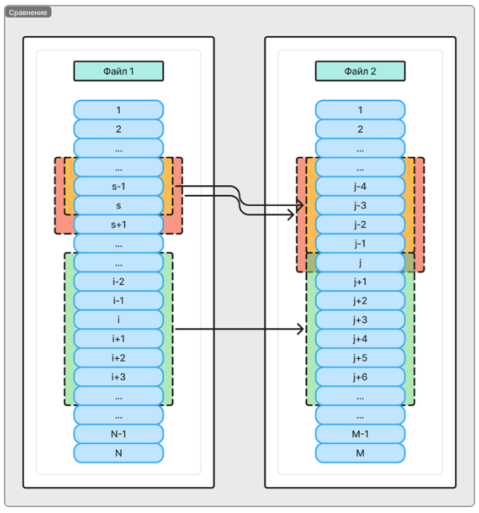

Рис. 1. Пример работы алгоритма разрешения конфликтов между кандидатами

На данном примере в момент, когда очередь доходит до кандидата, выделенного красным цветом, Рисунок 1, в файле 1 соответствующий блок ещё не выбран, однако в файле 2 соответствующие ему строки пересекаются с ранее размеченным зелёным блоком, а именно строка с индексом j содержится сразу в двух этих блоках. В такой ситуации красный блок исключается из рассмотрения, а вместо него рассматривается следующий, меньший по размеру блочный кандидат. Если к моменту, когда очередь дойдёт до распределения жёлтого блока, соответствующие ему строки в файле 2 не будут связаны с кандидатами, находящимися в списке между жёлтым и красным блоком, то жёлтые блоки добавятся к корректному соответствию и зафиксируются в итоговой разметке. Данная итеративная процедура позволяет избегать конфликтов разметки и детерминировать процесс сопоставления, что подчеркивает важность добавления кандидата на каждой итерации построения блока, а не только в момент его формирования с максимальной длиной.

После завершения процедуры блочного сопоставления осуществляется обратный процесс – отображение размеченных строк в исходные, которые, например, изначально могли быть «склеены» в одну, что позволяет агрегировать вероятностные данные для последующего анализа. Данный этап очень важен, чтобы правильно оценить каждый текст и корректно предъявить преподавателю результаты сравнения, поскольку имеет значение вероятность списывания для каждой строки, которая была изначально подана на вход до этапа предобработки. Всё это вместе обеспечивает прозрачность и точность оценки.

Выбор вероятностной метрики, пороги которой существенно влияют на эффективность работы алгоритма, должен осуществляться с учётом специфики выставления баллов за задания и особенностей совпадений между строками в различных языках программирования. Важно корректно определить, какие строки следует считать схожими, а какие не следует. Для большинства популярных языков программирования, использовавшихся в тестировании с реальными пользователями, наилучшей метрикой для оценки и выявления плагиата является расстояние Левенштейна [21]. Эта метрика рассчитывает минимальное количество операций (вставок, удалений и замен), необходимых для преобразования одной строки в другую, что позволяет выявлять не только идентичные фрагменты, но и вариации, возникающие в результате изменений в коде.

3. Заключение

В данной работе рассмотрены существующие подходы и методы по выявлению плагиата исходного кода (SCP) в задачах по программированию. Результатом исследования стало внедрение в цифровую образовательную платформу системы автоматического обнаружения плагиата, которая основана на принципе сопоставления наибольших совпадающих по заданной метрике блоков универсализированного кода, а также учитывающая синтаксические особенности языков программирования и устойчивая к наиболее распространённым способам модификации кода для сокрытия факта списывания.

Разработанный алгоритм был протестирован в рамках эксперимента на реальных учебных данных. Результаты показали высокую степень совпадения оценок вероятности списывания с экспертными ожиданиями преподавателей, а также устойчивость к наиболее распространённым способам модификации кода. Алгоритм продемонстрировал способность выявлять как полные, так и частичные заимствования, при этом сохранял читаемость и интерпретируемость отчётов для преподавателя, что особенно важно для принятия обоснованных решений. Эксперимент подтвердил надёжность и практическую применимость предложенного метода, таким образом, разработанная система представляет собой эффективный инструмент для автоматического выявления плагиата исходного кода в цифровых образовательных платформах.

Работа выполнена в рамках темы государственного задания НИЦ «Курчатовский институт» — НИИСИ по теме № FNEF-2024-0001 (1023032100070-3-1.2.1).

Plagiarism Detection Automation: a New Approach to Code Analysis in the Mirera Digital Educational Platform

D.I. Kadina, A.G. Leonov, N.S. Martynov, K.A. Mashchenko, E.A. Orlov, A.I. Strekalova

Abstract. The task of detecting plagiarism in programming task solutions holds high priority in digital educational platforms due to the necessity of providing accurate and reliable assessment of users' learning progress. Methods that compare submitted solutions solely based on textual similarity, without accounting for the syntactic characteristics of the programming languages in which the solutions are written, often fail to deliver precise and trustworthy results—similarly to statistical approaches based on machine learning. This study proposes a method for block-based comparison of student programming solutions for plagiarism detection, taking into account the syntactic features of programming languages.

Keywords: digital educational platform, DEP Mirera, anti-plagiarism, plagiarism analysis of program code, block-based comparison.

Литература

1. W. Murray. “Cheating in Computer Science”. In: Ubiquity (2010), p. 2. doi: 10.1145/1865907.1865908.

2. G. Cosma and M. Joy. “Towards a Definition of Source-Code Plagiarism”. In: IEEE Transactions on Education (2008), pp. 195–200. doi: 10.1109/te.2007.906776.

3. Curtis, G.J. and Popal, R., 2011. An examination of factors related to plagiarism and a five-year follow-up of plagiarism at an Australian university. International Journal for Educational Integrity, 7(1), pp.30-42.

4. Pierce, J. and Zilles, C., 2017, March. Investigating student plagiarism patterns and correlations to grades. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 471-476).

5. Maryon, Thomas; Dubre, Vandy C. Mrs.; Elliot, Kimberly; Fagan, Mary Helen; Standridge, Emily; and Lieneck, Christian, "COVID-19 Academic Integrity Violations and Trends: A Rapid Review" (2022). Healthcare Policy, Economics and Management Faculty Publications and Presentations. Paper 1.

6. Ambati, S.H., Stakhanova, N. and Branca, E., 2023, October. Learning AI coding style for software

plagiarism detection. In International Conference on Security and Privacy in Communication Systems (pp. 467-489). Cham: Springer Nature Switzerland.

7. Hourrane, O., 2019. Rich style embedding for intrinsic plagiarism detection. International Journal of Advanced Computer Science and Applications, 10(11).

8. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. arXiv:1310.4546 [cs, stat] (Oct 2013)

9. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings 31st International Conference on Machine Learning. vol. 32, pp. 1188– 1196 (2014)

10. Zhang, Y., Jin, R. and Zhou, Z.H., 2010. Understanding bag-of-words model: a statistical framework. International journal of machine learning and cybernetics, 1, pp.43-52.

11. El-Rashidy, M.A., Mohamed, R.G., El-Fishawy, N.A. and Shouman, M.A., 2022. Reliable plagiarism detection system based on deep learning approaches. Neural Computing and Applications, 34(21), pp.18837-18858.

12. Schleimer, S., Wilkerson, D.S. and Aiken, A., 2003, June. Winnowing: local algorithms for document fingerprinting. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (pp. 76-85).

13. Prechelt, L., Malpohl, G. and Philippsen, M., 2002. Finding plagiarisms among a set of programs with JPlag. J. Univers. Comput. Sci., 8(11), p.1016.

14. Joy, M. and Luck, M., 2002. Plagiarism in programming assignments. IEEE Transactions on education, 42(2), pp.129-133.

15. Ahadi, A. and Mathieson, L. (2019). A comparison of three popular source code similarity tools for detecting student plagiarism. In: Proceedings of the Twenty-First Australasian Computing Education Conference, ACE'19, 112–117.

16. Devore-McDonald, B. and Berger, E.D., 2020. Mossad: Defeating software plagiarism detection. Proceedings of the ACM on Programming Languages, 4(OOPSLA), pp.1-28.

17. Леонов А.Г., Мартынов Н.С., Мащенко К.А., Холькина А.А., Шляхов А.В. Автоматизация проверки семантической составляющей текстовых ответов обучающихся в цифровой образовательной платформе // Программные продукты и системы. 2024. Т. 37. № 3. С. 440–452. doi: 10.15827/0236-235X.142.440-452

18. Liu, C., Chen, C., Han, J. and Yu, P.S., 2006, August. GPLAG: detection of software plagiarism by program dependence graph analysis. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 872-881).

19. Hayden Cheers, Yuqing Lin, Weigen Yan, Identifying Plagiarised Programming Assignments with Detection Tool Consensus, Informatics in Education 22(2023), no. 1, 1-19, DOI 10.15388/infedu.2023.05

20. Cheers, H., Lin, Y. and Smith, S.P., 2021. Academic source code plagiarism detection by measuring program behavioral similarity. IEEE Access, 9, pp.50391-50412.

21. Levenshtein, V.I., 1966, February. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady (Vol. 10, No. 8, pp. 707-710).