АНАЛИЗ МЕТОДОВ И АЛГОРИТМОВ ПОИСКА ПЛАГИАТА В ИСХОДНОМ КОДЕ

Иванов И. П.
Кафедра информационных технологий автоматизированных систем, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: com.ilya.ivanov@gmail.by

В тезисах рассматриваются классификации методов поиска плагиата исходного кода, анализируются преимущества и недостатки некоторых современных алгоритмов поиска плагиата в исходном коде.

Введение

Вопрос плагиата очень стар и до сих пор актуален. Сама задача поиска плагиата довольно широка и сильно зависит от предметной области. Важная разница между плагиатом исходного кода и свободным текстом является то, что методы, используемые для их обнаружения различаются. Обнаружение копирования исходного кода является хорошо изученной областью и считается проще, чем обнаружение плагиата в свободном тексте, поскольку в исходном коде набор слов, который можно использовать, ограничен и строго регламентирован.

I. Классификация методов

Принято выделять два метода в оценке уровня заимствования в исходном коде [1]:

- аттрибутные методы;
- структурные методы.

Первыми в данной области появились аттрибутные методы. Сути их заключается в представлении исходного текста в виде точки в n-мерном пространстве, где каждая координата является некоторой количественной характеристикой программы. Примером такой характеристики может служить количество строк кода, количество операторов ветвления и циклов, количество переменных и т.д.

Две программы могут считаться похожими, если соответствующие точки в пространстве признаков близки или совпадают.

Недостатками метода является:

- высокий уровень ложных срабатываний;
- поверхностное изменение кода (без изменения логики программы) может скрыть факт заимствования;
- метод практически не работает, если задачей стоит поиск заимствования частей программы.

Другой, более современный метод состоит в сравнении программ с учетом их структуры. Эта процедура более сложная, чем сравнение отдельных аттрибутов программы. Очевидно, что это такие алгоритмы имеют высокую сложность и требуют больше ресурсов. Классическим примером структурного подхода является построение абстрактного дерева программы с последующим сравнением деревьев для разных программ. Сложность такого метода кубическая, что на практике не даёт возможности эффективно применять его для большого числа длинных программ. Сложность реализации структурных алгоритмов является платой за их точность.

Алгоритмы, относящиеся к структурному методу, мы и рассмотрим далее.

II. Структурные алгоритмы

Здесь мы рассмотрим несколько современных структурных алгоритмов для поиска плагиата с исходном коде, имеющие под собой готовые имплементации.

Прежде чем начать рассмотрение, давайте познакомимся с основной моделью представления программы в подобных алгоритмах, а именно: токенов.

Пусть у нас есть две строки кода $for(int\, i = n; i >= 0; i--)$ и $for(int\, k = n; k > -1; k--)$. Для многих очевидно, что обе строки кода эквивалентны, однако это не так очевидно для ЭВМ. Чтобы бороться с такими попытками сокрытия плагиата было придумана модель *токенизированного представления кода*. Суть заключается в игнорировании несущественных (легко модифицируемых) деталей программного кода, таких как названия переменных, функций и т.д. Процедуру токенизации можно описать следующим образом:

- каждому оператору языка приписывается заранее известный код;
- порядок следования кодов соответствует порядку следования операторов в исходном коде.

Жадное строковое замощение

Рассмотрим подробнее эвристический алгоритм получения жадного строкового замощения (Greedy String Tiling [2]). На вход алгоритма поступает две строки символов над фиксированным алфавитом (токенизированное представление кода), а на выходе выдает набор общих для двух строк непересекающихся подстрок, близкий к оптимальному. В алгоритме задействованы две эвристики:

– примем, что более длинные последовательные пересечения лучше, чем множество меньших и непоследовательных, в том числе, если в сумме длина последних больше;
– алгоритм игнорирует пересечения, длины которых меньше заданного порога.

Обе эвристики способствуют потере оптимального замощения, но результаты работы алгоритма, как правило, достаточно близки к нему, чтобы свидетельствовать о факте плагиата. Также вторая эвристика способствует фильтрации шумов, то есть маленьких участков кода, которые случайно совпали в текстах исходных кодов и не являются плагиатом. Сложность худшего случая для этого алгоритма является $O(n^3)$, но на практике она ниже $O(n^2)$, где n – длина токенизированного представления исходного кода программы. Также можно отметить высокую устойчивость алгоритма ко вставке операторов. Этот алгоритм успешно используется в программном средстве JPlag [3].

Метод идентификационных меток

Данный метод предоставляет возможность находить копии и частичные копии в текстовой базе большого объема. Для этого каждому файлы сопоставляется его более краткое представление – идентификационные метки (fingerprints).

Алгоритм локальной дактилоскопии можно представить так:

1. На вход алгоритма подается токенизированное представление исходного кода;
2. Текст разбивается на токены по k-gram алгоритму [4];
3. От каждого такого отрезка берется хэш-функция. На практике часто используется хэш функция из алгоритма Рабина-Карпа [5];
4. Последовательность значений, полученная на предыдущем шаге, разбивается на последовательность пересекающихся окон (отрезков) длинной w. Значение w получается как $w = t-k+1$, где t – порог гарантии (если у двух программ есть общая подстрока длиной как минимум t, то она будет найдена), k – порог шума (минимальная длина рассматриваемой подстроки), так чтобы $k \leq t$. Оба значения выбираются пользователем.
5. Происходит выборка значений хэш-функции для каждого окна методом просеивания [6]. В каждом окне выбирается минимальное значение хэша. Если таких значений несколько, то выбирается самое правое.

Полученный набор значений называется отпечатком документа. Если у двух документов совпадают значения отпечатков, то, с большой долей вероятности, в них есть совпадающие подстроки, что может свидетельствовать о плагиате.

К достоинствам алгоритма можно отнести:
– по итогам все отпечатки можно занести в базу данных, которая ускорит проверку один-против-всех;
– общие подстроки длиной меньше, чем порог шума, игнорируются;
– общие подстроки длиной больше, чем порог гарантии, гарантированно будут выявлены;
– алгоритм не чувствителен к перестановкам больших фрагментов кода;
– разбиение участка совпадения вставкой уникального блока, перестановка небольшого количества операторов, а также переименование переменных и функций слабо изменят значение функции схожести.

Недостатками же можно отметить:
– невысоких шанс совпадения отпечатков программ, но отсутствие совпадения в исходных кодах;
– для нахождения непосредственно совпадающих участков требуется применение других алгоритмов (например, алгоритм суффиксного дерева [7]);
– замена оператора на схожий (например, while на for) в середине совпадающего блока, каждый длиной меньше k может привести к полному игнорированию совпадения.

Данный алгоритм успешно применяется в программном средстве MOSS [8].

Список литературы

1. T. Lancaster, F. Culwin, “Classifications of Plagiarism Detection Engines”, Italics, Vol. 4 (2), 2005.
2. Wise M.J. String similarity via greedy string tiling and running Karp-Rabin matching. // Dept. of CS, University of Sydney. December 1993.
3. Institute for Program Structures and Data Organization [Электронный ресурс]. – Электронные данные. – Режим доступа : `https://jplag.ipd.kit.edu` – Дата доступа: 21.09.2019.
4. N-gram Language Models [Электронный ресурс] : Article / Stanford Universoty. – Электронные данные. – Режим доступа : `https://web.stanford.edu/~jurafsky/slp3/3.pdf` – Дата доступа: 20.09.2019.
5. R. Rarp, M. Rabin, Pattern-matching algorithms. IBM Journal of Research and Development, 21(2): 249-260, 1987.
6. Aiken A., Schleimer S., Wikerson D. Winnowing: local algorithms for document fingerprinting. // Proceedings of ACMSIGMOD Int. Conference on Management of Data. San Diego. 2003. P. 76–85. ACM Press. New York, USA. 2003.
7. Document Overlap Detection System for Distributed Digital Libraries [Электронный ресурс]. – Электронные данные. – Режим доступа : `https://www.researchgate.net/publication/221347616_Document_overlap_detection_system_for_distributed_digital_libraries` – Дата доступа: 21.09.2019.
8. Plagiarism Detection [Электронный ресурс]. – Электронные данные. – Режим доступа : `https://theory.stanford.edu/~aiken/moss` – Дата доступа: 21.09.2019.