ИТЕРАЦИОННЫЙ МЕТОД ПОИСКА КЛОНИРОВАННОГО КОДА, ОСНОВАННЫЙ НА ВЫЧИСЛЕНИИ РЕДАКЦИОННОГО РАССТОЯНИЯ

Кононов Д.С.[1]

В статье рассмотрены текстовые методы поиска клонированного кода в исходных кодах программ. Замечено, что подавляющее большинство таких методов основывается на аппроксимирующих алгоритмах вычисления редакционного расстояния вследствие высокой временной сложности точных алгоритмов локального выравнивания строк. Предложен метод поиска клонированного кода, заключающийся в последовательном применении алгоритмов локального выравнивания строк к промежуточным представлениям входных данных, полученных путём использования алгоритмов сжатия данных с потерей информации. После обработки сжатого промежуточного представления для следующей итерации выбираются только участки данных, соответствующие выравниваниям, вес которых больше заданного порога. Доказаны требования к используемым в предлагаемом методе алгоритмам сжатия данных с потерей информации. Выполнена оценка выигрыша в вычислительных затратах от коэффициента сжатия данных. Показано, что наибольший эффект даёт коэффициент сжатия на первой итерации, а в случае высокой схожести строк сжатого промежуточного представления следует сразу переходить к обработке неизменённых исходных данных. При условии редкости события появления клонированного кода предлагаемый метод позволяет достичь квадратичного выигрыша в зависимости от максимального коэффициента сжатия исходных данных.

***Ключевые слова**: программная безопасность, некорректности программ, повторно используемый код, заимствованные программные компоненты, программная избыточность, локальное выравнивание строк, сжатие с потерей информации, временная сложность, динамическое программирование.*

DOI: 10.21581/2311-3456-2017-1-16-21

При разработке программного обеспечения (ПО) зачастую применяется способ программирования известный под названием «скопировал - вставил» [1]. Этот способ приводит к появлению большого числа ошибок в ПО, которые, в свою очередь, приводят к уязвимостям информационных систем. Появившийся дубликат участка программного кода называют клонированным кодом. Отметим, что поиск клонированного кода может быть полезен и в задачах сертификации. Например, при проведении сертификационных испытаний новой версии ПО можно значительно сократить области исследования за счёт исключения найденных совпадений в программном коде новой версии ПО с версией, которая уже прошла сертификационные испытания. В этом случае процесс исключения уже изученных участков программного кода будет неотличим от поиска клонированного кода. Кроме того, проводя подобные исследования программного кода в рамках сертификации, можно будет обнаруживать недекларируемые возможности ПО [2], а также выявлять кражу интеллектуальной собственности [3, 4].

Важность и актуальность рассматриваемой задачи доказывает наличие большого числа исследований, посвящённых выявлению клонированного кода [5-7]. В то же время, в настоящий момент не существует универсального метода выявления клонированного кода [6, 8]. Данный факт является следствием противоречия между избирательностью выявления клонированного кода и вычислительной сложностью метода. Под избирательностью в контексте статьи будем понимать качественную оценку вероятности принятия правильного решения, или же, что эквивалентно, вероятность не совершения ошибок первого и второго рода.

Одним из основных классов методов поиска клонированного кода являются текстовые методы (рис. 1) [6]. Этот класс характеризуется использованием математического аппарата вычисления редакционного расстояния между строками. Однако существующие методы вычисления редакционного расстояния имеют квадратичную временную сложность в зависимости от количества символов в строках (за исключение метода «четырёх русских», который не используется из-за значительной пространственной сложности) [9]. Поэтому, на практике, применяются алгоритмы, аппроксимирующие редакционное расстояние.

1 Кононов Дмитрий Сергеевич, кандидат технических наук, ФГУП «18 ЦНИИ» МО РФ, Москва, sdk516@yandex.ru

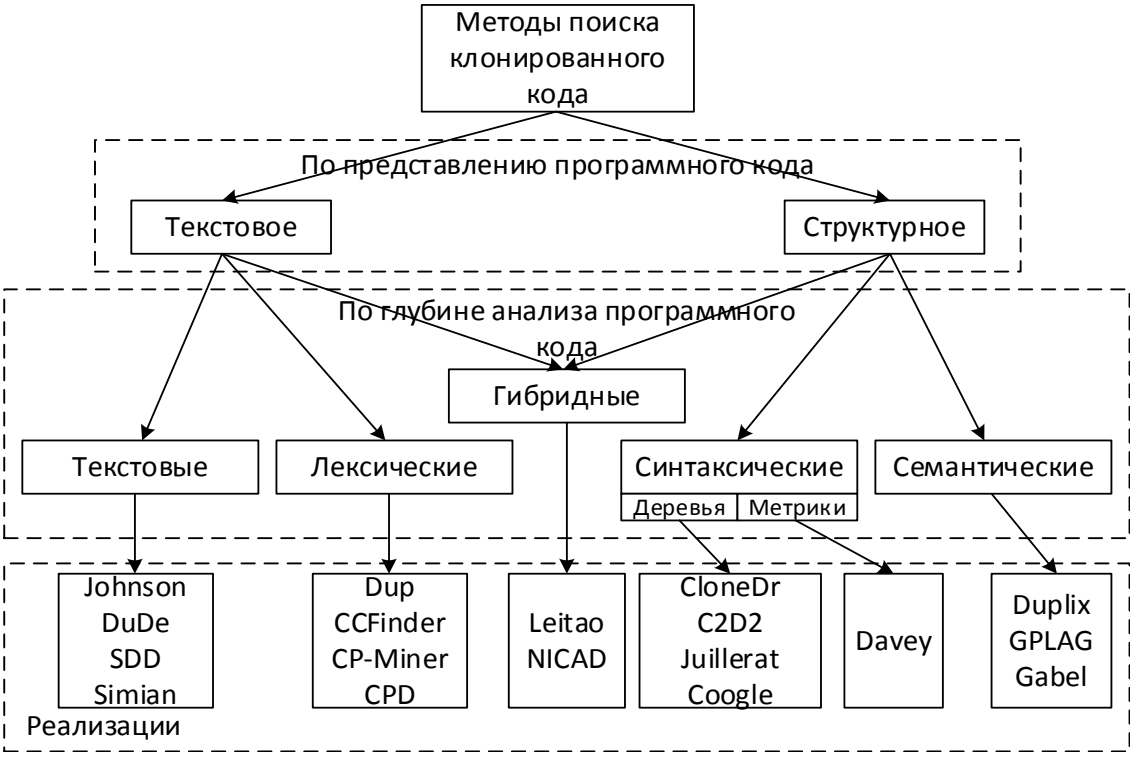

Рис. 1. *Классификация методов поиска клонированного кода*

Таблица 1.
Классификация клонированного кода

Тип клонированного кода	Описание
1	Идентичные участки программного кода с изменениями в форматировании, пробелах и комментариях.
2	Синтаксически идентичные участки программного кода с изменениями идентификаторов, литералов, типов. Включает изменения характерные для типа 1.
3	Скопированные участки программного кода с изменениями, добавлением или удалением команд. Включает изменения характерные для типа 2.

Среди них можно выделить различные алгоритмы хеширования строк [10, 11], нормализации формата [6], визуального сопоставления [12-14], построения суффиксного дерева [14], частотного поиска подпоследовательностей [5, 6]. Уменьшение временной сложности используемых алгоритмов сказывается на избирательности методов, что выражается в практически полной несостоятельности методов из данного класса при поиске клонированного кода типа 3 (табл. 1) [6, 8].

В статье предлагается метод, позволяющий преодолеть отмеченный недостаток текстовых методов, за счёт ускорения работы алгоритма локального выравнивания. Этот алгоритм вычисляет точное значение локального выравнивания и заключается в обобщении редакционного расстояния между строками на случай поиска отдельных схожих участков строк с использованием настраиваемых параметров поиска: матриц замен и штрафов за делецию[2] [9].

Языки высокого уровня отличаются наличием большой избыточности в тексте программ, что обусловлено удобством работы для человека. Если сократить избыточность в тексте программ, то длина текста программ сократится, соответственно, по квадратичной зависимости сократится и время работы алгоритмов локального выравнивания. Если развить рассмотренный подход, то можно сокращать длину текста программ за счёт использования алгоритмов сжатия данных с потерей информации. В этом случае предлагается использовать итеративный

2 Делеция — операция удаление символа.

метод поиска заимствований программного кода, представленный на рис. 2.

Предлагаемый метод заключается в дополнительной обработке программного кода перед использованием алгоритмов вычисления редакционного расстояния с целью сокращения объёма обрабатываемых данных за счёт применения сжатия с потерей информации. Он состоит из следующих шагов:

1) формируется сжатое текстовое представление программного кода с потерей информации;

2) производится поиск клонированного кода по сформированному сжатому тексту программы;

3) выбираются те участки текста программы, в которых был найден клонированный код, остальные участки кода отбрасываются;

4) оставшиеся участки кода передаются в виде входных данных на первый шаг, затем итерация повторяется.

В ходе применения рассматриваемого метода на первой итерации находятся участки программного кода, в которых может содержаться клонированный код. Однако из-за применённого сжатия исходных данных точная величина редакционного расстояния для них не известна. На каждой последующей итерации отклонение получаемых величин от истинной величины уменьшается за счёт снижения доли утрачиваемой информации, вплоть до обработки исходных данных. Это свойство можно интерпретировать как увеличение точности обнаружения клонированного кода с каждой итерацией.

Важно отметить, что предлагаемый метод будет работоспособен только в случае невозможности для каждого из применяемых сжатых представлений пропуска клонированного кода, т.е. на каждой итерации должна быть дана оценка сверху для истинной величины редакционного расстояния. Другими словами, в ходе обработки соответствующих сжатых текстовых представлений не должны допускаться ошибки второго рода при нулевой гипотезе, заключающейся в отсутствии клонированного кода. Докажем, что в этом случае применение рассматриваемого метода позволит найти все участки клонированного кода, которые найдёт и алгоритм вычисления редакционного расстояния, применённый к несжатым исходным текстам программ.

Лемма. Даны две строки s_1 и s_2, из символов алфавита Σ. Рассмотрим функцию $f(s)$, которая переводит строку s в строку s' из символов алфавита Σ', при этом $L = \frac{d(s)}{d(s')} > 1$, где $d(s)$ – количе-

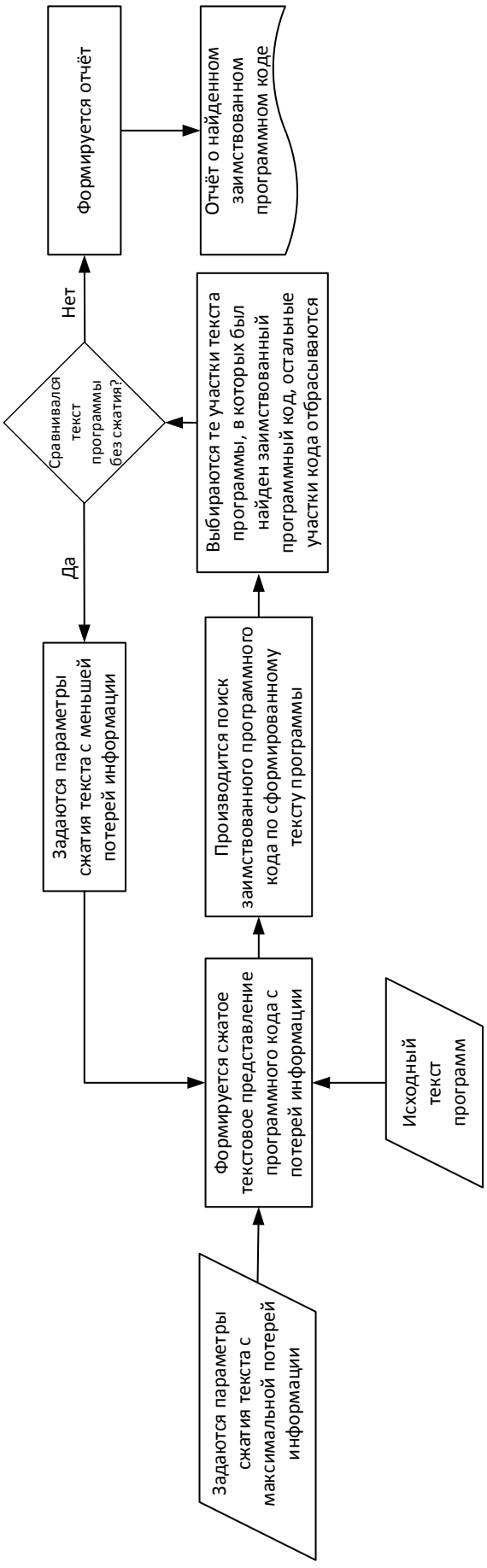

Рис. 2. *Блок-схема итерационного метода поиска клонированного программного кода*

ство символов в строке. Обозначим через $W(i,j)$ вес локального выравнивания, начинающегося соответственно с позиций i и j в исходных строках, причём $f(s[i]) = s'[i']$. Рассмотрим множество пар Q координат начал локальных выравнивания для строк s_1 и s_2, и Q' — для s_1' и s_2' соответственно. Обозначим подмножество Q_T, такое что $Q_T \subseteq Q: (i,j) \in Q_T \Longrightarrow W(i,j) \geq T$, где T — заданное пороговое значение. Аналогично Q_T' — для строк s_1' и s_2'. Пусть

$$\forall (i,j) \in Q\ \exists (i',j') \in Q': W'(i',j') \geq W(i,j). \quad (1)$$

Тогда существует инъекция между множествами Q_T и Q_T', переводящая $(i,j) \longrightarrow (i',j')$.

Доказательство. Рассмотрим множество Q_T. Пусть $Q_T = \emptyset$, тогда $Q_T \subseteq Q_T'$ по определению.

Пусть $Q_T \neq \emptyset$. Множества Q и Q' — конечные, так как сравниваются конечные строки. Соответственно, и их подмножества Q_T и Q_T' также конечные. Так как $Q_T \neq \emptyset$ и конечно, то можно выбрать элемент $(i,j) \in Q_T$. По условию леммы $\exists (i',j') \in Q': W'(i',j') \geq W(i,j)$. В силу $(i,j) \in Q_T$, то $W(i,j) \geq T$. Отсюда, $\exists (i',j') \in Q': W'(i',j') \geq T \Longrightarrow (i',j') \in Q_T'$ по определению. Таким образом, мы построили требуемую инъекцию, ч.т.д.

Теорема. Если все используемые в итеративном методе поиска клонированного кода сжатия с потерей информации функции f_k соответствуют требованиям леммы, где k — номер итерации. Тогда в результате работы метода будет получено Q_T^N тождественное Q_T, где N — количество итераций.

Доказательство. По индукции с использованием свойства функций f_k — $\forall (i,j) \in Q\ \exists (i',j') \in Q': W'(i',j') \geq W(i,j)$, получим

$$\forall (i,j) \in Q_T\ \exists (i',j') \in Q_T^1: W'(i',j') \geq W(i,j) \geq T.$$

Также, применяя лемму по индукции, придём к следующему

$$(i,j) \in Q_T^N \Longrightarrow W(x,y) \geq T. \quad (3)$$

Кроме того,

$\forall (i,j) \in Q_T\ \exists (i',j') \in Q_T^N: W'(i',j') \geq W(i,j)$. Для Q_T^N функция f_N является тождественной, то есть (i,j) — есть координаты начал локальных выравниваний в исходных строках. Соответственно, $|Q_T^N| \geq |Q_T|$. Но в Q_T входят все выравнивания $W(i,j) \geq T$, содержащиеся в исходных строках. Таким образом, $|Q_T^N| = |Q_T|$, иначе формируется противоречие. С учётом вышеизложенного $\forall (i,j) \in Q_T^N \Longleftrightarrow (i,j) \in Q_T$, соответственно Q_T^N тождественно Q_T по определению, ч.т.д.

Таким образом, для работы предлагаемого метода необходимо разработать сжатые представления программного кода с потерей данных, которые не допускают ошибок второго рода. С этих же позиций можно рассматривать и нормализацию программного кода перед поиском клонированного кода [6]. В качестве примера представлений, обладающих требуемыми свойствами, можно предложить замену конкретных элементов текста программ (переменных, функций, операций и других) на односимвольное обозначение класса этих элементов.

Рассмотрим теперь выигрыш в сокращении вычислительных затрат от применения описанного метода. Как отмечено выше, алгоритмы локального выравнивания строк имеют квадратичную временную сложность. Введём меру сжатия исходных текстов программ, используемых представлений программного кода, как относительное сокращение длины исходной строки, которую обозначим как L_i, где индекс i показывает номер итерации. Согласно описанию метода $L_N = 1$. Тогда отношение вычислительных затрат при выполнении алгоритмов выравнивания строк на исходных и на сжатых данных для первой итерации равно L_1^2. Теперь предположим, что на следующую итерацию была передана часть кода длиной $1/M_i$ от изначального. Согласно описанию метода $M_1 = 1$. Тогда отношение вычислительных затрат на следующей итерации будет равно $(L_i M_i)^2$, где $L_i M_i$ – относительное сжатие объёма обрабатываемых данных на i-итерации с учётом исключённых участков кода на предыдущих итерациях. Можно записать временную сложность предлагаемого метода относительно сложности алгоритма локального выравнивания (обозначенного как C)

$$\sum_{i=1}^{n} \frac{1}{(L_i M_i)^2} C = C \sum_{i=1}^{n} \frac{1}{(L_i M_i)^2}. \quad (4)$$

Рассмотрим худший случай для этого алгоритма, когда сравниваются два идентичных программных кода. Тогда $M_i = 1$, для всех i, а временная сложность метода будет определяться выражением

$$C \sum_{i=1}^{n} \frac{1}{(L_i)^2} = C \left(1 + \sum_{i=1}^{n-1} \left(\frac{1}{L_i} \right)^2 \right). \quad (5)$$

Как видно, в этом случае после обнаружения на первом этапе значительного совпадения программных кодов следует сразу переходить на обработку исходных текстов программ без сжатия

для снижения вычислительных затрат.

В случае если $M_i \gg 1$, то всеми L_i, кроме L_1 можно пренебречь, тогда

$$C\left(\left(\frac{1}{L_1}\right)^2 + \sum_{i=2}^{n}\left(\frac{1}{M_i}\right)^2\right). \quad (6)$$

Согласно определению M_i может быть вычислено по рекуррентной формуле $M_{i+1} = M_i * q_i$, где q_i – часть кода, которая была удалена из рассмотрения при переходе от i-итерации к следующей. Пусть $q_i = q = const$, тогда M_i представляет собой геометрическую последовательность. Соответственно, можно записать

$$C\left(\left(\frac{1}{L_1}\right)^2 + \sum_{i=2}^{n}\left(\frac{1}{q^i}\right)^2\right) =$$

$$= C\left(\left(\frac{1}{L_1}\right)^2 + \sum_{i=2}^{n}\left(\frac{1}{q^2}\right)^i\right) =$$

$$= C\left(\left(\frac{1}{L_1}\right)^2 + \frac{1-\left(\frac{1}{q}\right)^{2n}}{1-\left(\frac{1}{q}\right)^2} - 1\right) =$$

$$= C\left(\left(\frac{1}{L_1}\right)^2 + \frac{q^2-\left(\frac{1}{q^2}\right)^n}{q^2-1} - 1\right). \quad (7)$$

При $q \geq \sqrt{2}$ дополнительные вычислительные затраты метода не превысят L_1^2. Если предположить, что наличие клонированного программного кода является редким событием, то в этом случае $q \gg 1$ и выигрыш от использования метода в среднем асимптотически приближается к L_1^2. Поэтому основное внимание необходимо уделять на максимальное сжатие на первой итерации.

Предлагаемый метод при разработке соответствующих сжатых представлений программного кода позволит достичь компромисса между избирательностью и временной сложностью текстовых методов определения клонированного кода. При этом, учитывая опыт смежных научных дисциплин, где применяются алгоритмы локального выравнивания, можно ожидать значительного повышения избирательности в отношении наиболее сложных для обнаружения типов клонированного кода. Это в свою очередь приведёт к повышению качества как разработки программного кода, так и результатов сертификации.

***Рецензент**: Голов Игорь Юрьевич, кандидат технических наук, ФГУП «18 ЦНИИ» МО РФ, golovnew2@yandex.ru*

Литература

1. Demeyer S., Ducasse S., Nierstrasz O. Object-oriented reengineering patterns. Square Bracket associates, 2009. 360 p.
2. Markov A.S., Fadin A.A., Tsirlov V.L. Multilevel Metamodel for Heuristic Search of Vulnerabilities in the Software Source Code // International Journal of Control Theory and Applications, 2016, Volume 9, No. 30, pp. 313-320.
3. Barabanov A.V., Markov A.S., Tsirlov V.L. Methodological framework for analysis and synthesis of a set of secure software development controls // Journal of Theoretical and Applied Information Technology. 2016. Т. 88. № 1. С. 77-88.
4. Кононов Д.С. Об улучшении избирательности одного алгоритма определения авторства вредоносного кода // Вопросы кибербезопасности. 2016. № 2 (15). С. 29-35.
5. Zhenmin L., Shan L., Suvda M., Yuanyuan Z. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code // IEEE Trans. on SE, v. 32, No. 3.
6. Roy C.K., Cordya J.R., Koschke R. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach // Science of Computer Programming, v. 74, No 7, May 2009, pp. 470–495.
7. Svajlenko J., Chanchal K.R. Evaluating Modern Clone Detection Tools // Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME). 2014. С. 10.
8. Roy C.K., Cordy J.R. An Empirical Study of Function Clones in Open Source Software // WCRE ‹08 Proceedings of the 2008 15th Working Conference on Reverse Engineering. 2008. С. 81-90.
9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Санкт-Петербург: Невский Диалект, БХВ-Петербург, 2003. 654 с.
10. Toomey W. Ctcompare: Code Clone Detection Using Hashed Token Sequences. Zurich. // Proceedings of the 6th International Workshop on Software Clones (IWSC ‹12). 2012. С. 92-93.
11. Uddin M.S., Roy C.K., Schneider K.A., Hindle A. On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems // Proceedings of the 2011 18th Working Conference on Reverse Engineering (WCRE ‹11). 2011. С. 13-22.
12. Ducasse S., Rieger M., Demeyer S. A Language Independent Approach for Detecting Duplicated Code // Proceedings of the IEEE International Conference on Software Maintenance (ICSM99). 1999. С. 109-119.
13. Wettel R., Marinescu R. Archeology of Code Duplication: Recovering Duplication Chains from Small Duplication Fragments // Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC ‹05). 2005. С. 63-71.

14. Lee S., Jeong I. SDD: High Performance Code Clone Detection System for Large Scale Source Code // Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA ‹05) San Diego. 2005. C. 140-141.
15. CCFinder: A Multilinguistic Token-based Code Clone Detection System for Large Scale Source Code // IEEE Transactions on Software Engineering, T. 28, № 7, июль 2002. C. 654-670.

ITERATIVE METHOD FOR CLONE CODE DETECTION BASED ON COMPUTING THE EDIT DISTANCE

***D. Kononov*[3]**

Abstract. *In the article the lexical methods for code clone detection in source codes of programs are examined. Noticed, that the vast majority of such methods are based on approximation algorithms for computing the edit distance because of the high time complexity of exact algorithms, which perform local sequence alignment. A new method for clone code detection is suggested. This method consists in sequential applying of local sequence alignment algorithms on the intermediate representations of the input data obtained through the use of algorithms for data compression with loss of information. After processing the compressed intermediate representation for the next iteration only portions of the data corresponding alignments, which have weight greater than a predetermined threshold, are selected. The requirements for suitable algorithms for data compression with loss of information are proved. The estimation of gain in time depending on the data compression ratio is performed. It is shown that the greatest effect provides the compression ratio in the first iteration. In the case of high similarity of compressed intermediate representation strings one should go directly to the processing of unmodified input data. Considering the rareness of the occurrence of cloned code the proposed method allows to reach a quadratic payoff depending on the maximum compression ratio of input data.*

Keywords: *clone code, edit distance, local sequence alignment, compression with loss of information, time complexity, dynamic programming.*

References

1. Demeyer S., Ducasse S., Nierstrasz O. Object-oriented reengineering patterns. Square Bracket associates, 2009. p. 360.
2. Markov A.S., Fadin A.A., Tsirlov V.L. Multilevel Metamodel for Heuristic Search of Vulnerabilities in the Software Source Code. International Journal of Control Theory and Applications, 2016, Volume 9, Issue No. 30, pp. 313-320.
3. Barabanov A.V., Markov A.S., Tsirlov V.L. Methodological framework for analysis and synthesis of a set of secure software development controls. Journal of Theoretical and Applied Information Technology, 2016, Volume 88, No 1, pp. 77-88.
4. Kononov D.S. Ob uluchshenii izbiratel'nosti odnogo algoritma opredeleniya avtorstva vredonosnogo koda, Voprosy kiberbezopasnosti, 2016, No 2 (15), pp. 29-35.
5. Zhenmin L., Shan L., Suvda M., Yuanyuan Z. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code, IEEE Trans. on SE, v. 32, No. 3.
6. Roy C.K., Cordya J.R., Koschke R. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, Science of Computer Programming, Vol. 74, No 7, 2009, pp. 470–495.
7. Svajlenko J., Chanchal K.R. Evaluating Modern Clone Detection Tools. In Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME), 2014, p. 10.
8. Roy C.K., Cordy J.R. An Empirical Study of Function Clones in Open Source Software. WCRE ‹08 Proceedings of the 2008 15th Working Conference on Reverse Engineering, 2008, pp. 81-90.
9. Gasfild D. Stroki, derev'ja i posledovatel'nosti v algoritmah. Informatika i vychislitel'naja biologija. St. Petersburg: Nevskij Dialekt, BHV-Peterburg, 2003, p. 654.
10. Toomey W. Ctcompare: Code Clone Detection Using Hashed Token Sequences. In Proceedings of the 6th International Workshop on Software Clones (IWSC ‹12), Zurich. 2012. pp. 92-93.
11. Uddin M.S., Roy C.K., Schneider K.A., Hindle A. On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems. In Proceedings of the 2011 18th Working Conference on Reverse Engineering (WCRE ‹11), 2011, pp. 13-22.
12. Ducasse S., Rieger M., Demeyer S. A Language Independent Approach for Detecting Duplicated Code. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM99), 1999, pp. 109-119.
13. Wettel R., Marinescu R. Archeology of Code Duplication: Recovering Duplication Chains from Small Duplication Fragments. In Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC ‹05), 2005, pp. 63-71.
14. Lee S., Jeong I. SDD: High Performance Code Clone Detection System for Large Scale Source Code. Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA ‹05), San Diego. 2005, pp. 140-141.
15. CCFinder: A Multilinguistic Token-based Code Clone Detection System for Large Scale Source Code, IEEE Transactions on Software Engineering, Vol. 28, No. 7, 2002, pp. 654-670.

3 Dmitrii Kononov, Ph.D., FGUP «18 CNII» MO RF, Moscow, sdk516@yandex.ru