УДК 004.4

СИСТЕМА АВТОМАТИЗАЦИИ ЧИСЛЕННОЙ ОЦЕНКИ СХОДСТВА ANDROID-ПРИЛОЖЕНИЙ

В. В. Петров [0009-0004-4213-7328]

Институт информационных технологий и интеллектуальных систем, Казанский (Приволжский) федеральный университет, ул. Кремлёвская, 35, г. Казань, 420008

valeryvpetrov.itis@gmail.com

Аннотация

Работа посвящена проектированию и разработке системы автоматизации численной оценки сходства Android-приложений. Задача оценки сходства приложений сведена к оценке сходства множеств графов потока управления, построенных на основе кода из classes.dex файлов приложений. Значение сходства вычислено на основе матрицы сходства. Для сравнения графов потока управления использованы алгоритмы редактирования графов и расстояние Левенштейна. Сформулированы критерии сходства приложений и исследованы формы их представления. Представлены виды моделей Android-приложений и методы их построения. Разработан прототип системы автоматизации численной оценки сходства Android-приложений. С помощью инструментов параллельного программирования выполнена оптимизация программного решения. Проведены эксперименты и сделан вывод о способности разработанной системы выявлять сходства между Android-приложениями.

Ключевые слова*: сходство Android-приложений, сходство программ, матрица сходства, расстояние редактирования графов потока управления, визуализация матрицы сходства, граф потока управления.*

ВВЕДЕНИЕ

Сходство программ оценивается во многих задачах: обнаружения дублированного или клонированного кода, проверки авторских прав или патентов, повы-

шения утилизации кодовой базы и др. Для решения каждой из приведенных задач прежде всего необходимо определить, что означает «сходство программ», а также предложить способы его измерения.

Опытный программист способен оценить сходство программ по ключевым критериям: наименованию классов, функций, переменных, потока управления, использованию литералов и др. В таком случае часть критериев может быть определена и оценена субъективно и отличаться от сравнения к сравнению не только одним программистом, но и несколькими. В таких условиях возникает необходимость как в формализации критериев, по которым определяется сходство программ, так и в разработке методов вычисления оценки их сходства.

Для оценки сходства между множеством программ необходимо выполнить $\frac{n!}{2(n-2)!}$ попарных сравнений, где n – количество программ. Для задачи нахождения плагиата возможен поиск среди миллионов программ. При этом программы могут состоять из тысяч файлов, каждый из которых может иметь тысячи строк кода. Выполнение сравнения вручную может потребовать годы, в то время как размер программ и их количество растут каждый день. Все эти условия формируют потребность в автоматизации сравнения сходства программ.

Решение проблемы автоматизации численной оценки сходства программ имеет практическую значимость Эта проблема актуальна для индустрии приложений для ОС Android. Согласно исследованию [1], от 5 до 13% Android-приложений, распространяемых на шести Android-площадках, являются клонами. В исследовании [2] показано, что как минимум 141 приложение было клонировано. Также известны случаи плагиата мобильных приложений [3, 4].

Приложения для ОС Android распространяются в виде .apk и .aab артефактов [5]. Артефакты могут быть скачаны с различных площадок, таких как Google Play Store, Huawei App Gallery и др. При этом площадки могут быть не только официальными. Так как перед установкой приложения необходимо скачать на устройство артефакт, возникает ситуация, когда злоумышленник получает прямой доступ к артефакту и способен выполнить MATE-атаку [6]. После этого злоумышленник может опубликовать измененное приложение в интернете под видом исходного и украсть пользовательские данные, ухудшить имидж компании и др.

В рамках настоящей работы спроектирована и разработана система автоматизации численной оценки сходства Android-приложений. Для моделирования Android-приложения использована матрица графов потока управления функций приложения. Сходство графов потока управления оценена при помощи вычисления расстояния редактирования графов. Вычисления итогового значения сходства выполнено с использованием технологий параллельных вычислений. Разработаны инструменты для автоматической и ручной оценок сходства, визуализации матрицы сходства и графов потока управления.

Статья имеет следующую структуру. В разделе 1 представлен обзор существующих решений и проведён сравнительный анализ этих решений. В разделе 2 дана постановка задачи сравнения Android-приложений и введены необходимые обозначения. В разделе 3 представлен сравнительный анализ форм представления Android-приложений и дано обоснование выбора формы представления приложений, используемой в настоящей работе. В разделе 4 проведён сравнительный анализ видов моделей приложений и дано обоснование выбора конкретной модели. В разделе 5 представлены методы построения моделей приложений, проведено их сравнение с целью выбора подходящего метода. В разделе 6 предложен метод оценки сходства моделей приложений на основе расстояния редактирования графов приложений. В разделе 7 описан алгоритм численной оценки сходства Android-приложений. В разделе 8 представлены вспомогательные инструменты для выполнения экспертной оценки сходства приложений. В разделе 9 описаны эксперименты и приведен анализ результатов.

1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

В последнее время появилось множество исследований, направленных на выявление клонированных или перепакованных приложений в среде Android (см., например, [7–11]). Эти исследования, как правило, завершаются простым решением «да/нет» о том, является ли приложение копией, но в них отсутствует детальное обоснование этого решения и областей сходства. Тем не менее, в сфере исследований, разработок и даже среди пользователей существует очевидная потребность в понимании различий между разными версиями приложений. Например, разработчикам маркетплейсов и их пользователям часто требуется определить изменения в последнем обновлении приложения, чтобы убедиться, что они

соответствуют рекламируемым «новым» функциям. Разработчики приложений могут анализировать изменения в собственных приложениях, чтобы оценить их влияние на рейтинг в маркетплейсе. Тем временем исследователи могут разрабатывать системы рекомендаций по изменениям, анализируя версии приложений, и методы обнаружения вредоносного содержимого в переупакованных вредоносных программах.

Задача обнаружения переупакованных приложений представляет собой серьезную проблему. В последнее время научное сообщество, учитывая постоянно растущее количество приложений, занимается созданием быстрых методов оценки сходства на основе ресурсов приложений или машинного обучения. Однако результаты применения этих методов все еще требуют тщательной проверки путем детального сравнения потенциальных пар переупакованных приложений.

К сожалению, ведущие методы обнаружения клонов или переупакованных приложений основаны на сложных внутренних эвристиках, которые трудно воспроизвести, а разработанные прототипы инструментов часто недоступны для развития исследований в этой области [10]. Многие исследования по обнаружению переупакованных приложений [8, 11] не предоставляют инструменты, которые могло бы повторно использовать исследовательское сообщество. Насколько известно, такие общедоступные инструменты, как Androguard [12], FSquaDRA [13] и SimiDroid [14], являются основными ресурсами для изучения сходства приложений. Androguard анализирует сходства на уровне кода, FSquaDRA – на основе ресурсов. Однако ни Androguard, ни FSquaDRA не описывают различия между приложениями, что ограничивает их возможности для глубокого анализа сходств. SimiDroid призван восполнить этот пробел в исследованиях, создав фреймворк с открытым исходным кодом, который легко масштабируется и способен объяснять сходства и различия приложений. SimiDroid оснащен тремя разработанными плагинами для сравнения сходства на уровне кода, Android-компонентов и ресурсов приложения. В статье [14] SimiDroid проведен сравнительный анализ с Androguard и FSquaDRA. Результаты анализа показали, что методы, основанные на сравнении кода, дают более точные оценки сходства, чем подходы, основанные на сравнении ресурсов. SimiDroid был протестирован на реальных приложениях в несколь-

ких тематических исследованиях, продемонстрировав свою способность объяснять сходства в различных контекстах. Однако SimiDroid не позволяет исследователям изменять формулу, используемую для расчета баллов сходства.

Отличительной особенностью настоящего исследования являются сравнение скомпилированных Android-приложений в формате .apk с использованием графа потока управления в качестве модели программы, а также применение матрицы сходства для сравнения моделей.

2. ПОСТАНОВКА ЗАДАЧИ

Даны два Android-приложения P_1 и P_2. Необходимо вычислить $sim(P_1, P_2)$ – степень сходства приложений P_1 и P_2. При этом:

- P_1 и P_2 могут иметь одну из форм представления: *исходный код, скомпилированный код, скомпилированный артефакт*;
- $sim(P_1, P_2) \in [0; 1]$;
- $sim(P_1, P_2) = 0$ означает, что программы не имеют никакого сходства;
- $sim(P_1, P_2) = 1$ означает, что программы абсолютно идентичны;
- $sim(P_1, P_2) = sim(P_2, P_1)$.

Для построения модели программы используется функция $m: P \to M$, которая выделяет характеристики программ и объединяет их в структуру данных, которая называется моделью программы, причем:

- Характеристики программы выделяются при помощи *статического* или *динамического анализа* кода программы.
- $m(P_1) = M_1, m(P_2) = M_2$ являются моделями программ P_1 и P_2 соответственно.
- Значение $m(P)$ может быть получено только из P.
- Программа P_2 называется *производной* от программы P_1, если над ней были выполнены преобразования, не нарушающие смысла программы P_1. К таким преобразованиям относятся: обфускация (запутывание кода программы), изменение графа потока управления, шифрование констант и др.
- Если программа P_2 является производной от P_1, то $M_1 = M_2$.

3. ВЫБОР ФОРМЫ ПРЕДСТАВЛЕНИЯ ANDROID-ПРИЛОЖЕНИЯ

Android-приложения могут быть представлены в трех формах:

- Исходный код – в репозитории с приложением или на компьютере разработчика.
- Скомпилированный код – при промежуточной компиляции исходного кода в байт-код JVM [15] или байт-код ART [16].
- Скомпилированный артефакт – при компиляции приложения. В результате компиляции создаются файлы в форматах .apk или .aab, которые могут быть установлены на смартфон.

С точки зрения преобразования формы представления, исходный код содержит в себе наибольшее количество авторской информации, интеллектуальной собственности, так как исходный код создается человеком, его разработчиком. Скомпилированный код и артефакт получаются при помощи стандартных инструментов: компиляторов и др., которые преобразуют форму представления по известным алгоритмам. При этом компилятор способен удалить часть информации из исходного кода, например, комментарии, которые будет невозможно восстановить при декомпиляции кода. Поэтому количество полезной информации в исходном коде больше, чем в скомпилированном артефакте. С другой стороны, во время работы приложения выполняются скомпилированный, оптимизированный или обфусцированный коды, а не исходный.

С точки зрения доступности формы представления, скомпилированный артефакт имеет наибольшую доступность, так как может быть скачан из официального магазина приложений; любого интернет-ресурса; устройства, на котором установлено приложение, в отличие от исходного кода, доступ к которому, как правило, ограничен владельцем репозитория или компьютера.

В рамках данной работы в качестве сравниваемых программ выступают Android-приложения в форме скомпилированного артефакта – .apk файла. Выбор такой формы представления обоснован ее наибольшей доступностью и фактическим использованием при работе программы.

Архив .apk содержит следующие файлы (см., например, [17]):

- скомпилированный Dalvik / ART байт-кода – classes.dex. Таких файлов может быть несколько;
- манифеста приложения – AndroidManifest.xml;
- метаинформации – META-INF;
- ресурсов приложения – assets, res;

- скомпилированных нативных библиотек – lib;
- скомпилированных ресурсов – resources.arsc.

Каждый из этих файлов содержит информацию о работе приложения и может быть использован при оценке сходства; resources.arsc, META-INF, AndroidManifest.xml содержат типовую информацию о приложении и не несут особой ценности при копировании. Более того, содержимое этих файлов кратно меньше, чем у classes.dex, assets, res, lib. Директории assets и res содержат макеты приложения, файлы анимации, статические ресурсы и др. При реверс-инжиниринге этот код может быть использования для плагиата пользовательского интерфейса приложения; classes.dex и lib содержат скомпилированный код, наибольшее количество полезной информации о работе приложения и занимают основную часть .apk файла. В lib находится скомпилированный C/C++ код. Использование нативного кода при разработке Android-приложений является узконаправленным решением для специфичных задач – отрисовки сложной графики, ресурсоемких вычислений и др. – и на практике встречается редко.

В рамках данной работы предложено рассматривать файлы classes.dex в качестве источника информации об Android-приложении, так как они содержат скомпилированный исходный код и занимают значительную часть .apk файла.

4. ВЫБОР МОДЕЛИ ANDROID-ПРИЛОЖЕНИЯ

Исходно classes.dex представляет собой бинарный код для среды выполнения Android: Dalvik или ART. Для построения моделей на основе бинарного кода, инструкций, блоков инструкций, функций достаточно декомпилировать файл .apk и выполнить поверхностный анализ содержимого. Такой подход позволяет добиться высокой скорости построения в сравнении с другими моделями. Однако ни одна из них не описывает структуру приложения и является чувствительной к обфускации.

Следующие модели, помимо декомпиляции .apk, требуют дополнительных преобразований для построения, что снижает общее время работы алгоритма. Абстрактное синтаксическое дерево (abstract syntax tree, AST) описывает структуру программы и выполняемые инструкции, что является ее преимуществом (см., например, [19]). Основной недостаток AST заключается в том, что дерево строится на основе исходного кода программы, и в этом случае файл classes.dex

необходимо предварительно декомпилировать до .class файлов и потом снова декомпилировать до .java, .kt файлов. Дважды выполненная декомпиляция приводит к потерям информации, что критично для модели, использующей AST.

Модель на основе иерархии классов может быть построена на основе Android-приложения, так как при его разработке используются объектно-ориентированные языки. Модель описывает различные зависимости между классами: наследование, композицию и др. (см., например, [20]). Недостатком модели на основе иерархии классов является то, что она не описывает фактическую реализацию класса, то есть модель не в полной мере отражает содержимое программы.

Граф зависимостей программы [21] описывает поток управления функции и поток данных, что является преимуществом среди прочих моделей. Единственным недостатком является то, что граф не содержит информации о других вызываемых функциях.

Граф вызовов [22] также может быть использован в качестве модели Android-приложения: он описывает его структуру и не зависит от используемого языка программирования. Однако современные обфускаторы способы изменять поток управления, что влияет на чувствительность модели. Более того, граф вызовов не описывает содержимое функций.

Граф потока управления [23] в отличие от двух предыдущих моделей описывает выполняемые блоки инструкций. Также граф описывает структуру программы и имеет размерность меньше, чем у AST.

В рамках данной работы предложено в качестве модели Android-приложения использовать граф потока управления, построенный на основе скомпилированного кода файлов classes.dex. Несмотря на то, что граф потока управления также чувствителен к обфускации потока управления и требует дополнительных преобразований для построения, он является наиболее подходящим среди прочих при моделировании программ. Этот вывод подтверждается в систематическом обзоре литературы по теме сходства программ [18].

5. ВЫБОР МЕТОДА ПОСТРОЕНИЯ МОДЕЛИ

Для построения графа потока управления Android-приложения могут быть

использованы как статический, так и динамический анализ приложения. При статическом подходе анализ Android-приложения никогда не выполняется, а модель строится на основе кода из файлов classes.dex. При динамическом анализе Android-приложения он запускается на виртуальном или реальном устройстве, и исследуется его поведение во время выполнения.

Статический анализ приложения эффективен, потому что может исследовать набор всех возможных путей выполнения путем аппроксимации поведения программы. Это важно, потому что некоторые сценарии работы Android-приложений трудно вызвать динамически: выполнение задач по расписанию, работы приложения в фоне и др.

Основное преимущество динамического анализа заключается в том, что он отображает фактическое выполнение Android-приложения, а запутывания кода, применяемые к classes.dex, оказывают меньшее влияние на поведение программы.

В рамках настоящей работы предложено строить граф потока управления Android-приложения при помощи статического анализа. Выбор такого метода преимущественно обусловлен непоследовательной (асинхронной) природой выполнения Android-приложений. Взаимодействие с базовыми компонентами Android осуществляется по системе обратных вызовов: изменение интерфейса, обращение в сеть, работа в фоновом режиме и др., что значительно усложняет покрытие всех сценариев использовании приложения в случае использования динамического анализа.

6. ВЫБОР МЕТОДА ОЦЕНКИ СХОДСТВА МОДЕЛЕЙ

Метод сравнения на основе максимального общего подграфа используется для оценки сходства графов (см., например, [24]). Большим преимуществом этого метода является возможность контролировать размерность общего подграфа, что ускоряет его построение. Однако данный метод имеет один значительный недостаток: наличие нескольких общих подграфов. Эта проблема особенно актуальна при построении максимального общего подграфа для графов малого размера. Такой вывод подтвержден в исследовании [25].

Для оценки сходства графов потока управления в рамках настоящей работы

использован оптимизированный алгоритм вычисления расстояния редактирования [26]. Этот алгоритм требует меньше памяти и времени, чем вариации алгоритма A* (A star), и способен учитывать содержимое блоков инструкций графа потока управления.

В рамках настоящей работы расстояние редактирования графа вычисляется по формуле [26]

$$GED(g_1, g_2) = \min_{e_1,\dots,e_k \in \gamma(g_1,g_2)} \sum_{i=1}^{k} c(e_i), \quad (1)$$

где GED – расстояние между графами g_1 и g_2; $GED(g_1, g_2) \geq 0$;

e_i – одна из операций редактирования: вставка, удаление, замена;

$e_1, e_2, \dots, e_k \in \gamma(g_1, g_2)$ – путь редактирования длины k;

$\gamma(g_1, g_2)$ – множество путей редактирования g_1 в g_2;

$c(e_i)$ – функция стоимости операции редактирования, $c(e_i) = 1$.

Для приведения значения расстояния редактирования графа (1) в интервал [0; 1] предлагается использовать следующую формулу:

$$\widehat{GED}(g_1, g_2) = \frac{GED(g_1, g_2)}{\max(size(g_1), size(g_2))}, \quad (2)$$

где $\widehat{GED}(g_1, g_2)$ – расстояние редактирования между графами g_1, g_2;

$\widehat{GED}(g_1, g_2) \in [0; 1]$; $size(g)$ – размер графа g, $size(g) = |V| + |E|$.

Принятие решения о редактировании вершины графа потока управления выполняется по формуле

$$decision = sim(S_1, S_2) < threshold, \quad (3)$$

где $decision = \{\text{Ложь}, \text{Истина}\}$ — решение о редактировании;

$threshold$ – порог принятия решения, $threshold \in [0; 1]$;

S_1, S_2 – сравниваемые последовательности;

$M = len(S_1), N = len(S_2)$ – длины последовательностей;

$sim(S_1, S_2) = \max(M, N) - D(S_1, S_2)$.

$$D(S_1, S_2) = D(M, N),$$

где D – расстояние Левенштейна [27], $D \in [0; \max(M, N)]$.

Если значение *sim* меньше порогового, то вершина редактируется.

7. ОПИСАНИЕ АЛГОРИТМА

Алгоритм автоматизации численной оценки сходства Android-приложений можно разделить на основные этапы:

1. Подготовка к построению моделей сравниваемых приложений.
2. Построение моделей (графов потока управления) первого и второго Android-приложения.
3. Построение матрицы сходства на основе расстояния редактирования графов потока управления.
4. Нахождение пар наиболее схожих элементов.
5. Вычисление итогового значения сходства.

Для ускорения работы алгоритма построения матрицы сходства разработанная программа имеет механизмы параллелизации вычисления как на уровне строк матрицы, так и на уровне ее ячеек. Для этого создается конечное множество процессов, между которыми распределяется задача вычисления строк матрицы сходства. Внутри каждого процесса создается конечное множество потоков, каждый из которых вычисляет сходство между двумя графами потока управления.

Помимо параллелизации при вычислении расстояния редактирования используется параметр, устанавливающий верхнюю границу времени вычисления. При достижении этой границы в качестве результата алгоритм возвращает значение расстояния редактирования, актуальное на момент остановки.

8. ВСПОМОГАТЕЛЬНЫЕ ИНСТРУМЕНТЫ

В этом разделе приведено краткое описание программных инструментов,

разработанных для получения детальной информации о ходе выполнения и результатах оценки сходства. Функция calculate_apks_similarity.py в режиме оценки сходства Android-приложений выводит в консоль информацию («логи») о выполнении алгоритма: списки генерируемых графов потока управления, результаты вычисления сходства и др. Сгенерированные файлы содержат полезную для исследователя информацию, представленную в текстовом виде. Однако логи не визуализируют матрицу сходства и пары наиболее схожих графов потока управления. Для повышения скорости, простоты и качества экспертного анализа результатов работы calculate_apks_similarity.py была разработана дополнительная программа.

Интерфейс программы построен в виде окна, на котором изображается вычисленная calculate_apks_similarity.py матрица сходства. Отображаемая матрица сходства обрабатывает нажатия на ячейки. При нажатии на ячейку программа генерирует .pdf файла в котором находятся визуализированные графы потока управления. Граф на первой странице соответствует строке матрицы, на второй – столбцу. Сгенерированный .pdf файл имеет имя dots_<номер строки>_<номер столбца>_<значение сходства>.pdf (например, dots_1_23_1.0.pdf) и находится в папке visualize_m_comp. Папка visualize_m_comp находится в папке переданной аргументом --output_dir calculate_apks_similarity.py.

9. ЭКСПЕРИМЕНТЫ

На данный момент не существует стандартизированных критериев оценки сходства Android-приложений. Существует решение суда № А40-20593/2017 в котором присутствует формулировка «Схожесть наименований двух программ для ЭВМ и их основное назначение не могут служить основанием для вывода об их тождественности, так как для подобного вывода необходимы специальные знания (заключение эксперта)» [28]. Что касается научных исследований, то в ряде статей сравнивают собственные разработки с аналогами (например, в [29] k-gramm сравнивается с моделью предложенной Харуаки Тамада). В других выполняется оценка существующих программ без сравнения аналогов (например, в [30] код изменяется вручную и затем вычисляется сходство между исходной и измененной программой).

Наиболее простыми сценариями проверки сходства двух Android-приложений являются крайние значения интервала сходства: 0 – одно приложение пустое (нет кода) и любое Android-приложение (есть код); 1 – сравнение приложения с самим собой. Так как модель программы должна является её инвариантом (не изменяться при трансформациях с сохранением семантики), то в качестве ещё одного эксперимента можно использовать сравнение Android-приложения до обфускации и после (обфускация не должна влиять на значение сходства, либо влиять незначительно). Ожидается, что чем больше техник обфускации будет применено к Android-приложению, тем меньшим сходством оно будет обладать при сравнении его с исходным вариантом. Однако в систематическом обзоре литературы [17] подчеркивается, что на данный момент отсутствует модель, которая является устойчивой к любым трансформациям с сохранением семантики.

В рамках данной работы программное решения оценивается по следующим сценариям:

1. Android-приложение сравнивается с пустым .apk файлом.
2. Android-приложение сравнивается с самим собой.
3. Android-приложение сравнивается с вариантом этого приложения, в котором были изменены имена классов, пакетов при помощи ProGuard.
4. Android-приложение сравнивается с вариантом этого приложения, оптимизированным при помощи ProGuard вариант (опция -optimizationpasses 5).
5. Производится сравнение двух разных Android-приложений.

Сгенерированные файлы, информация о выполнении программ в рамках каждого эксперимента размещаются в GitHub репозитории работы в папке с соответствующим номером в папке «exp» (например, для первого эксперимента эта папка exp/1).

Описанные ниже эксперименты выполнялись на компьютере со следующей конфигурацией:

- наименование: MacBook Pro;
- операционная система: macOS Monterey (версия 12.4);
- процессор: Apple M1 Pro с 10 ядрами;
- объем оперативной памяти: 16 ГБ.

Краткое описание результатов экспериментов представлено в Таблице 1.

Значение колонки «Верно сопоставленные пары сходства» вычислено как отношение корректно найденных пар к общему числу выбранных пар сходств и найдено экспертно на основе анализа графов потока управления.

Таблица 1. Описание результатов экспериментов.

№	$sim(P_1, P_2$	Время вычисления, сек	Верно сопоставленные пары сходства	ins_block_sim_threshold	ged_timeout_sec, сек	processes_count	threads_count
1	0.0	2.3	0%	0.95	60	10	45
2	0.99	5.3	100%	0.95	60	10	45
3	0.9	3.6	100%	0.95	60	10	45
4	0.2	3.4	80%	0.95	60	10	45
5	0.19	64.4	23%	0.95	60	10	45

В первом эксперименте результат полностью соответствует ожиданиям. Так как второй .apk файл не содержал classes.dex, то построенная модель состояла из пустого множества графов потока управления. Поэтому dots_2.csv не содержит ни одной записи. Матрица сходства имеет размерность 17:0.

Во втором эксперименте матрица сходства симметрична по диагонали, что говорит о сравнении двух идентичных .apk файлов (см. рис. 1). Большинство ячеек матрицы имеют черный цвет и соответствуют конструкторам класса R (класс с идентификаторами ресурсов в Android).

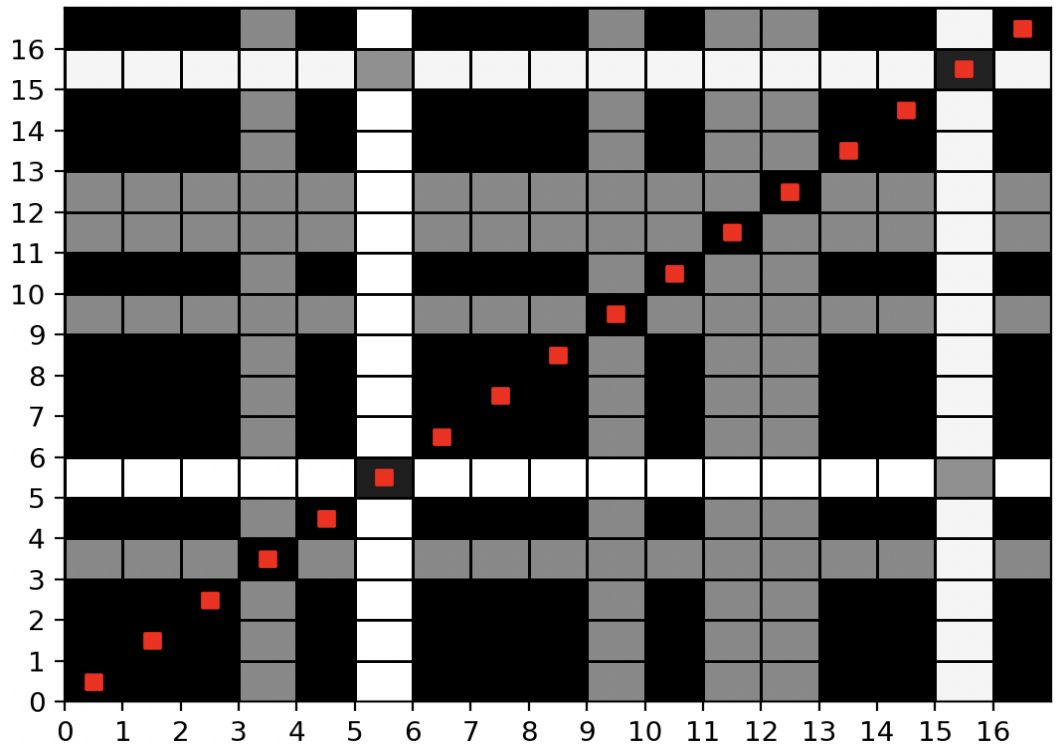


Рис. 1. Визуализация матрицы сходства из эксперимента 2

Примеры графов потока управления для ячеек 2:4, 10:16 представлены на рис. 2 слева и справа соответственно.

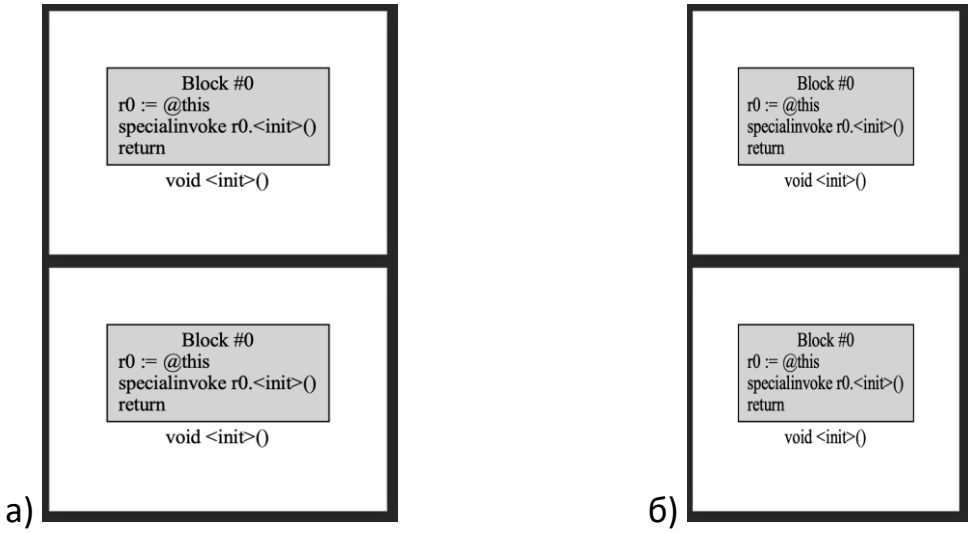


Рис. 2. Визуализация графов потока управления из эксперимента 2: R.layout void <init>().dot и R.xml void <init>().dot (а), R.style void <init>().dot и Unused void <init>().dot (б)

Строки и столбцы с серыми ячейками как правило имеют заметные структурные различия и соответствуют более сложным функциям программы. На рис. 3 представлен графы потока управления для ячейки 4:3, на рис. 4 для 15:5.

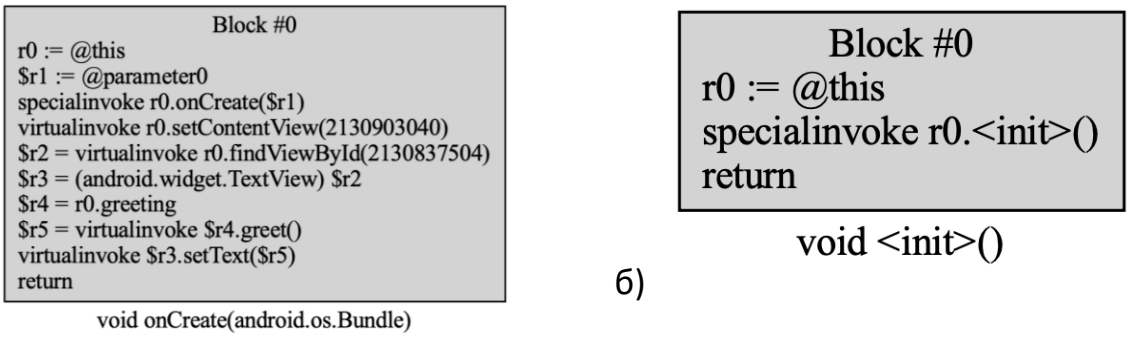


Рис. 3. Визуализация графов потока управления из эксперимента 2: MainActivity void onCreate(android.os.Bundle).dot (а) и R.xml void <init>().dot (б)

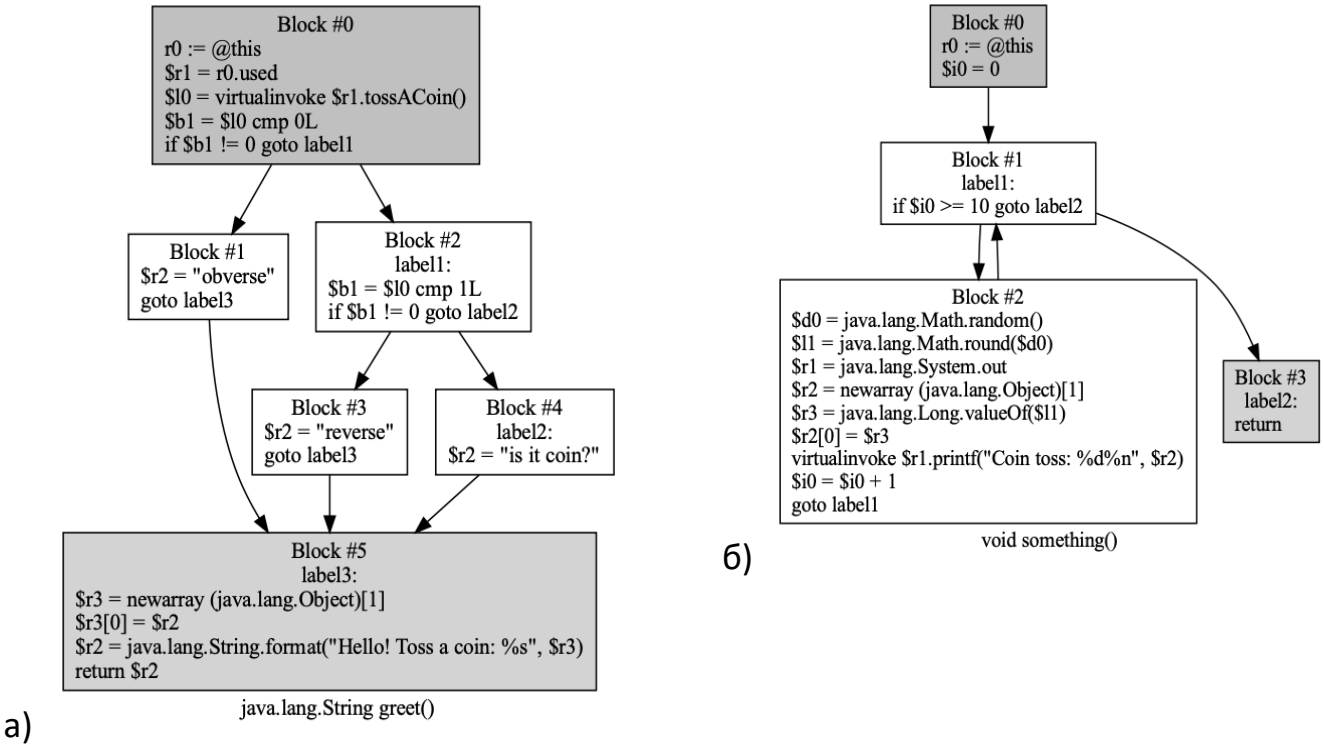


Рис. 4. Визуализация графов потока управления из эксперимента 2: Greeting java.lang.String greet().dot (а) и Unused void something().dot (б)

Значение сходства 0.99 приближено к 1.0. Программа сформировала корректный список пар соответствия графов потока управления. Однако

допустила ошибки при вычислении сходства для графов 5:5 (0.89), 15:15 (0.88).

В отличие от матрицы сходства из второго эксперимента, в данном случае пары графов не образуют диагональ (см. рис. 5).

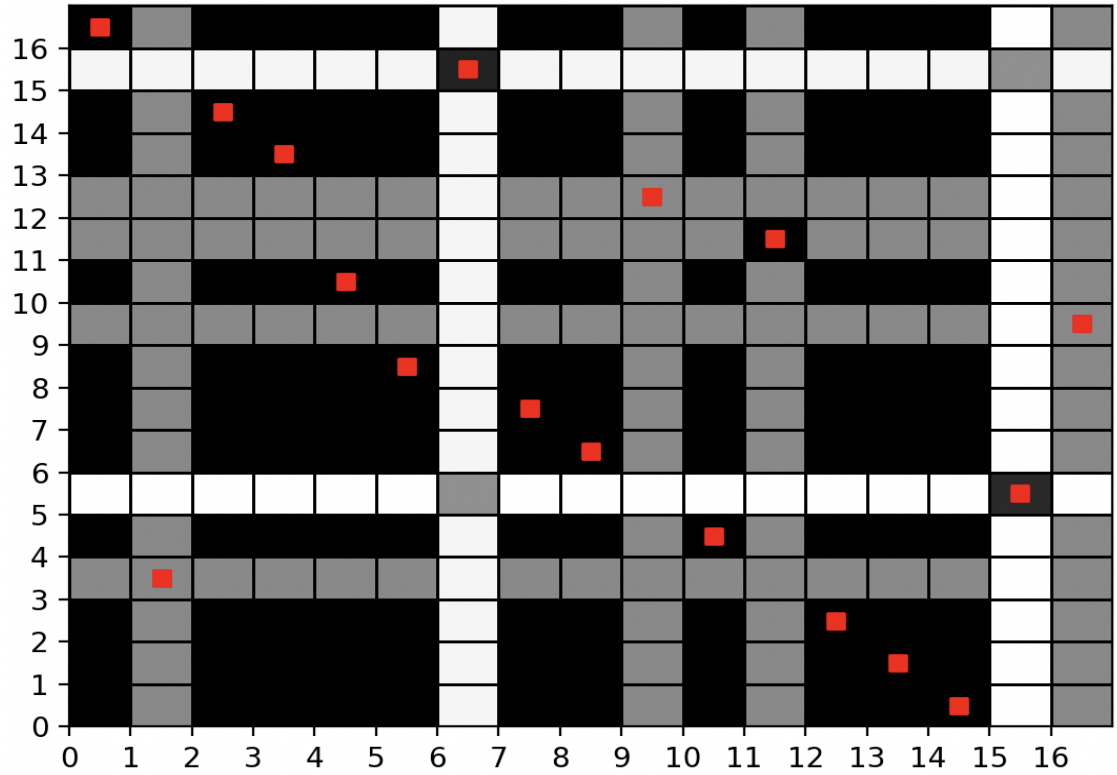


Рис. 5. Визуализация матрицы сходства из эксперимента 3

Это связано с тем, что имена методов при обфускации .apk изменяются, как и порядок их обхода при формировании списка графов. На рис. 6 видно, что структурно графы идентичны, но в блоках инструкции на третьей строке видно незначительное различие в наименовании классов «com.example.simpleapplication.something.Used» и «a.b».

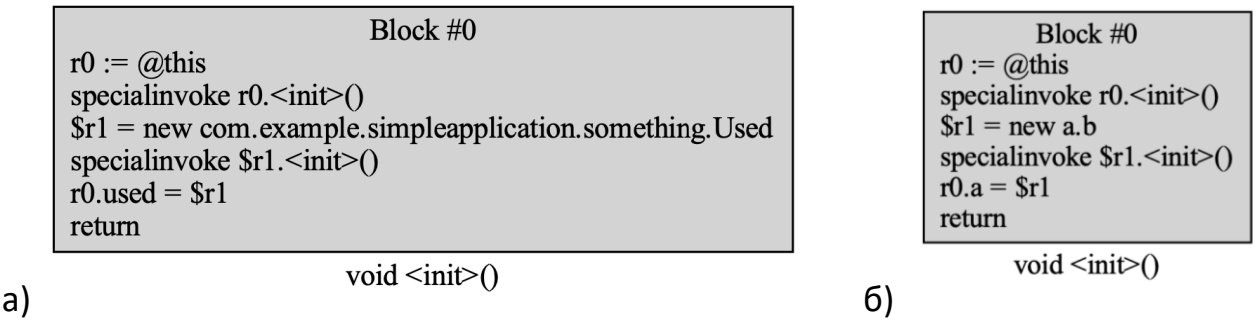


Рис. 6. Визуализация графов потока управления из эксперимента 3: Greeting void <init>().dot (а) и b void <init>().dot (б)

В строке 5 также видна разница в наименовании свойства класса «used» и «a». Именно эти различия в именах приводят к уменьшению значения сходства. Значение сходства 0.9 приближено к 1.0. Программа сформировала корректный список пар соответствия графов потока управления. Однако допустила ошибки при вычислении сходства блоков инструкций 3:1 (0.5), 5:15 (0.85) и др.

В четвертом эксперименте после выполнения оптимизации .apk файла количество классов сократилось с 17 до 5 (см. рис. 7).

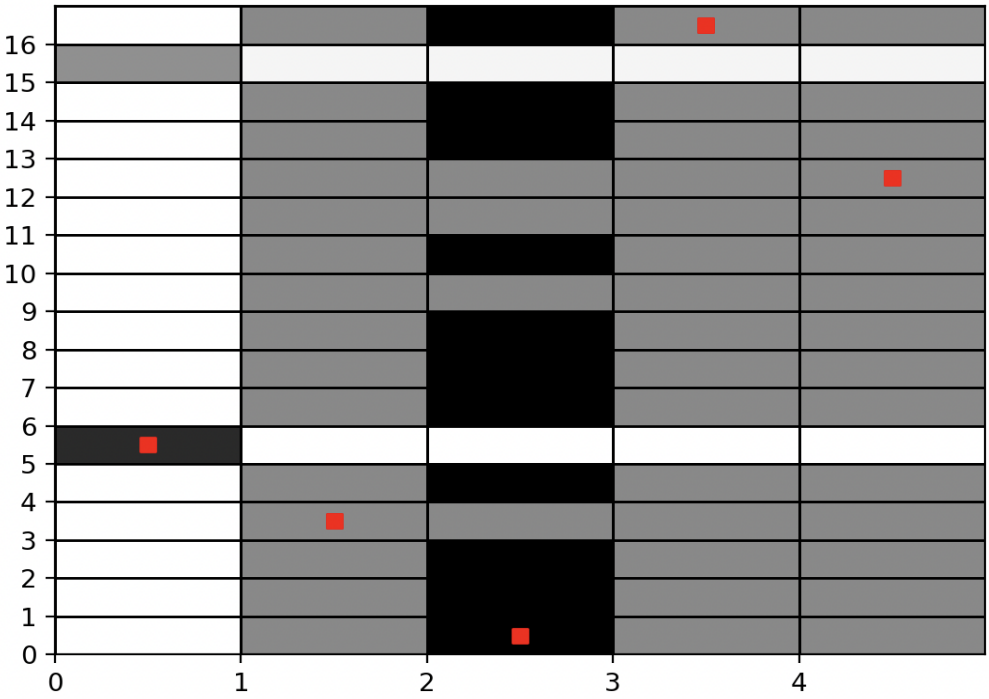


Рис. 7. Визуализация матрицы сходства из эксперимента 4

Помимо случаев, уже рассмотренных в третьем эксперименте, когда переименование снижает сходство, в этом примере есть один новый сценарий. На рис. 8 графы потока управления имеют структурное сходство.

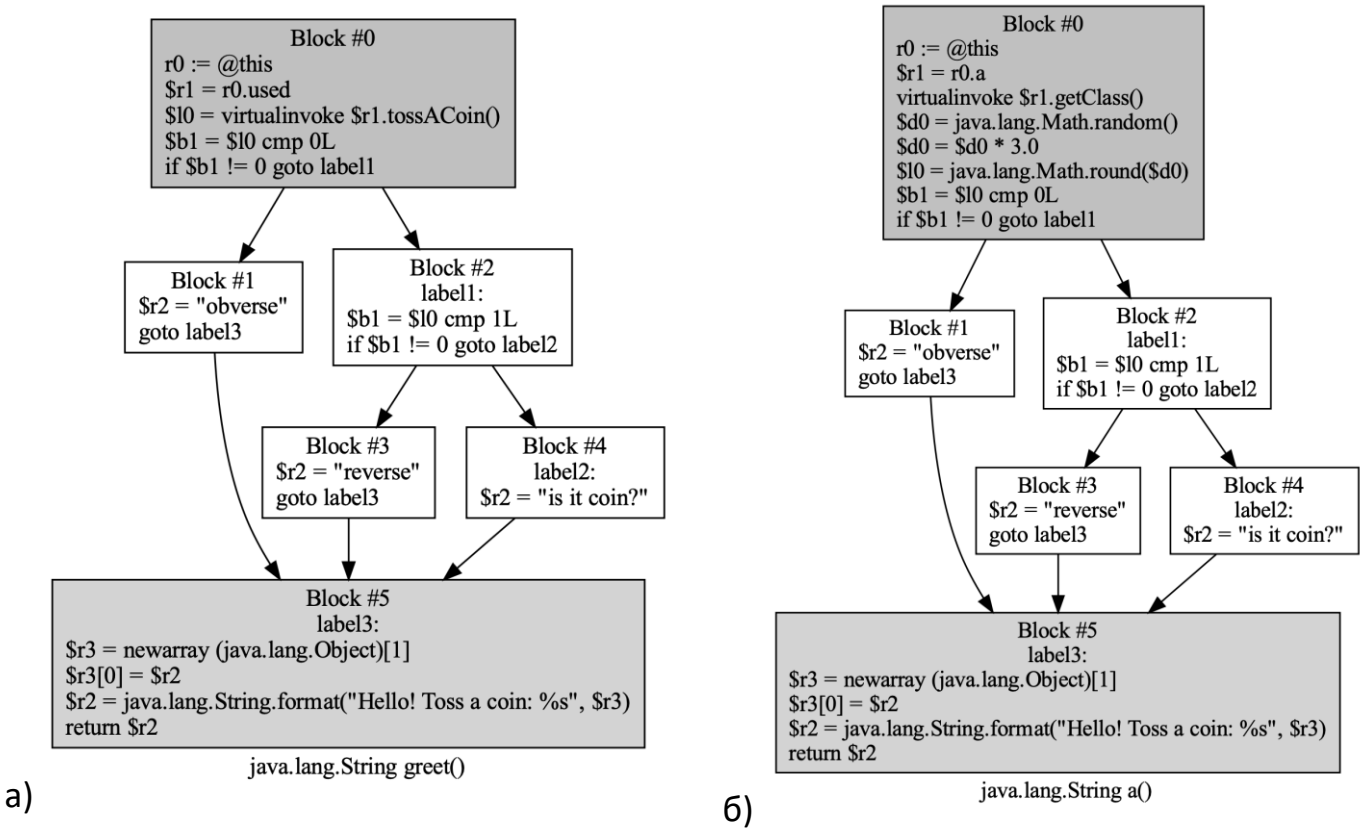


Рис. 8. Визуализация графов потока управления из эксперимента 4: Greeting java.lang.String greet().dot (а) и a java.lang.String a().dot (б)

Однако в «Block #0» графа б) был перенесен код из класса «com.example.simpleapplication.something.Used», при этом сам класс был удален из сборки. Значение сходства 0.2 показывает, что модификации графа потока управления оказывают сильное влияние на итоговое значение сходства. Несмотря на низкое значение сходства, программа верно сопоставила 4 из 5 графов потока управления. При использовании опции --ins_block_sim_threshold 0.5 сходство повышается до 0.29 (+0.09). Однако понижение значения порога эквивалентности блоков инструкций повышает количество схожих между собой графов потока управления, что может привести к формированию некорректных пар сходства. Данный вывод проиллюстрирован на рис. 9.

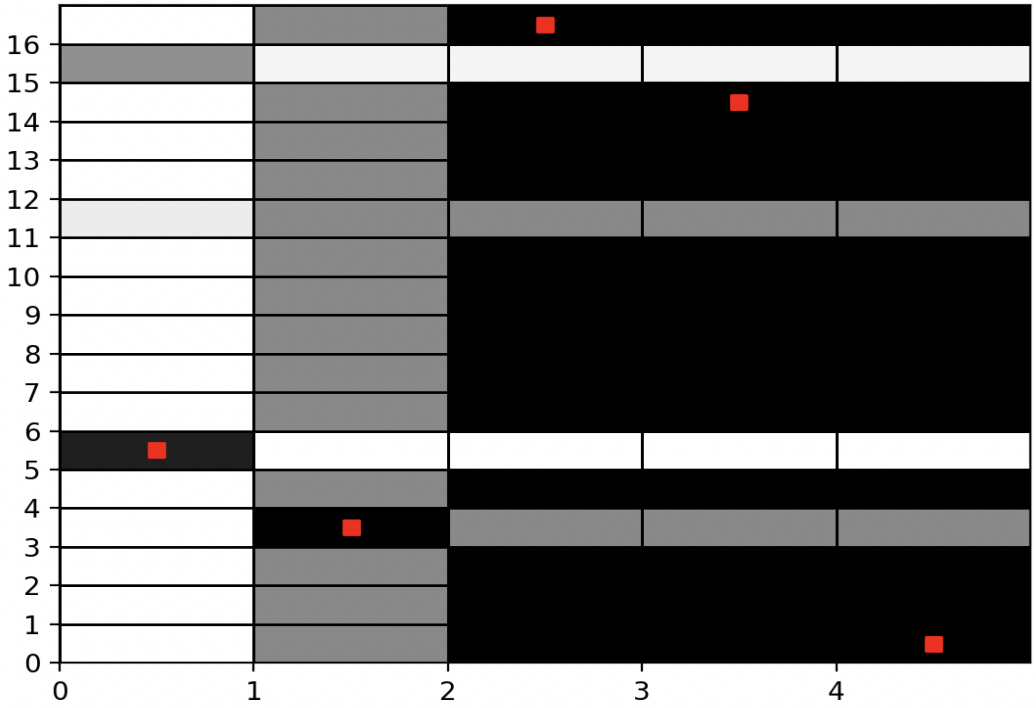


Рис. 9. Визуализация матрицы сходства из эксперимента 4

В пятом эксперименте значение сходства 0.19 показывает, что два принципиально разных Android-приложения имеют незначительное сходство (см. рис. 10). Пары сходства, сформированные программой, преимущественно не имеют схожей структуры. Пары со сходством, приближенным к 1, представляют собой графы потоков простых конструкторов классов и не несут полезной информации о сходстве всего приложения.

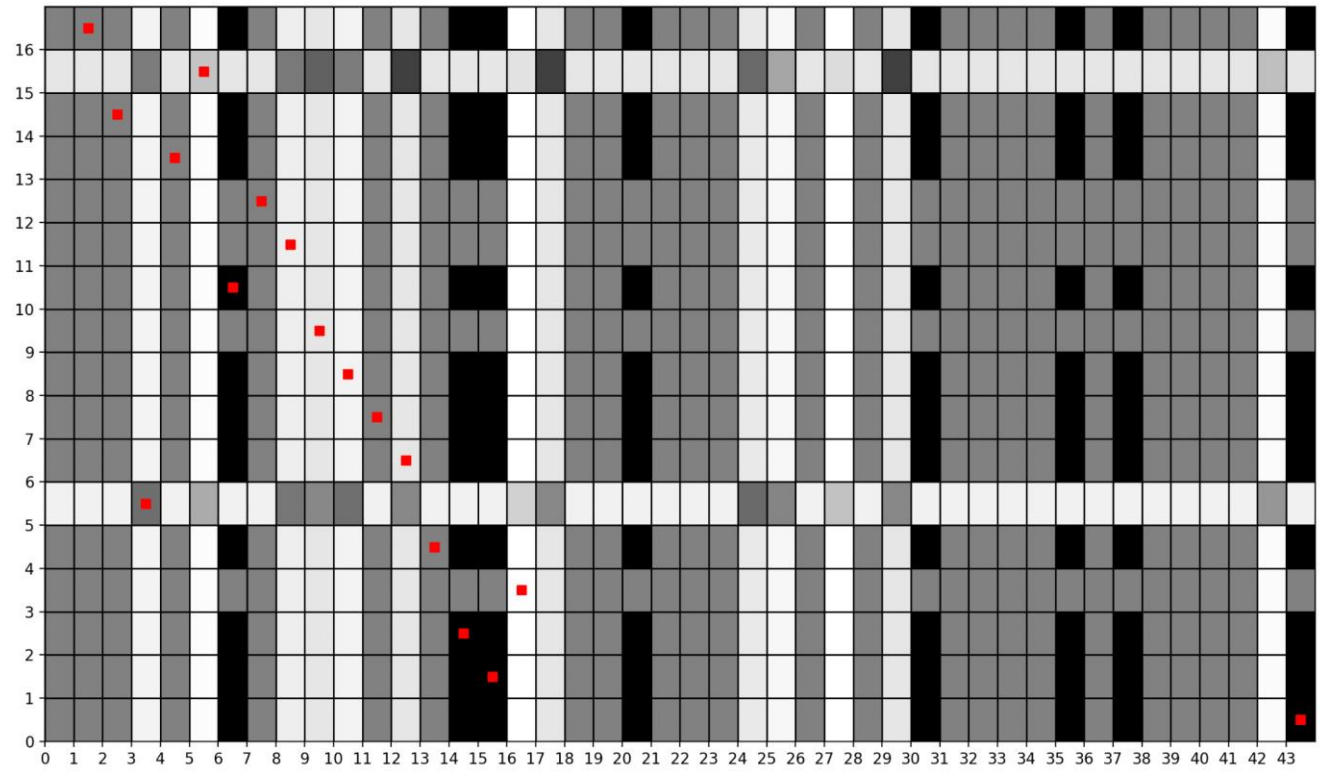


Рис. 10. Визуализация матрицы сходства из эксперимента 5

ЗАКЛЮЧЕНИЕ

Работа посвящена проектированию и разработке системы автоматизации численной оценки сходства Android-приложений. Были выполнены следующие задачи.

- Проведен анализ форм представления программ, видов моделей программ, методов построения моделей программ, видов сходства программ.
- Предложена модель представления Android-приложений, методы ее построения и алгоритм оценки сходства моделей.
- Спроектирован и разработан программный модуль для построения модели программы и численной оценки сходства.

Задача численной оценки сходства Android-приложений имеет ряд практических применений. Например, Google Play Store, RuStore, Huawei AppGallery и др. могут оценить сходство публикуемых приложений для поиска плагиатов. Функционал поиска плагинов позволит площадкам оповещать разработчиков-авторов о краже их интеллектуальной собственности, оповещать пользователей площадок

о подозрительности приложений с высокой степенью сходства, запрещать публиковать такие приложения и др. Также численная оценка сходства Android-приложений может быть использована для оценки работы обфускаторов (например, ProGuard, R8), при анализе кражи интеллектуальной собственности в судебных делах и прочих сценариях.

Разработанная система является прототипом решения задачи численной оценка сходства Android-приложений. Выполненные в работе эксперименты показали состоятельность разработанного решения. Среди преимуществ отметим: высокую скорость работы, репрезентативность модели с точки зрения структуры программы, инструменты для экспертного анализа. Среди недостатков укажем: чувствительность к обфускации. Для более точной оценки эффективности работы системы потребуются дополнительные эксперименты.

Дальнейшие развитие исследования предполагает повышение точности вычисления расстояния редактирования графов. Несмотря на то, что программа корректно формирует пары схожих графов потока управления, значения их сходства не равно 1.0. Также необходимо решить вопрос учёта оценки графов малой мощности.

Прототип разработанной системы прошел государственную регистрацию в виде программы для ЭВМ в Федеральной службе по интеллектуальной собственности (Роспатент) [31].

Ряд результатов, полученных в проведенном исследовании, представлен и обсуждён на конференции «Научный сервис в сети Интернет 2023» [32].

Текущая версия приложения размещена в открытом доступе в репозитории GitHub и доступна по ссылке: https://github.com/valeryvpetrov-dev/android-apps-similarity.

СПИСОК ЛИТЕРАТУРЫ

1. *Zhou W., Zhou Y., Jiang X., Ning P.* Detecting repackaged smartphone applications in third-party android marketplaces // Second ACM conference on Data and Application Security and Privacy. 2012. P. 317–326. https://doi.org/10.1145/2133601.2133640

2. *Crussell J., Gibler C., Chen H.* Attack of the clones: Detecting cloned applications on android markets // European Symposium on Research in Computer Security.

2012. P. 37–54. https://doi.org/10.1007/978-3-642-33167-1_3

3. Market Shocker! Iron Soldiers XDA Beta Published by Alleged Thief // Android Headline. URL: https://www.androidheadlines.com/2011/01/market-shocker-iron-soldiers-xda-beta-published-by-alleged-thief.html.

4. Fake Mobile Apps Steal Facebook Credentials, Cryptocurrency-Related Keys // TREND MICRO. URL: https://www.trendmicro.com/en_us/research/22/e/fake-mobile-apps-steal-facebook-credentials--crypto-related-keys.html.

5. Android App Bundle frequently asked questions // Android developers. URL: https://developer.android.com/guide/app-bundle/faq

6. *Akhunzada A., Sookhak M., Anuar N.B., Gani A., Ahmed E., Shiraz M., Furnell S., Hayat A., Khan M.K.* Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directions // Journal of Network and Computer Applications. 2015. No. 48. P. 44–57. https://doi.org/10.1016/j.jnca.2014.10.009

7. *Chen J., Alalfi M.H., Dean T.R., Zou Y.* Detecting android malware using clone detection // Journal of Computer Science and Technology. 2015. No. 30. P. 942–956. https://doi.org/10.1007/s11390-015-1573-7

8. *Wang H., Guo Y., Ma Z., Chen X.* Wukong: A scalable and accurate two-phase approach to android app clone detection // Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015. P. 71–82. https://doi.org/10.1145/2771783.2771795

9. *Chen K., Liu P., Zhang Y.* Achieving accuracy and scalability simultaneously in detecting application clones on android markets // Proceedings of the 36th International Conference on Software Engineering. 2014. P. 175–186. https://doi.org/10.1145/2568225.2568286

10. *Li L., Bissyandé TF., Papadakis M., Rasthofer S., Bartel A., Octeau D., Klein J., Traon L*. Static analysis of android apps: A systematic literature review // Information and Software Technology. 2017. No. 88. P. 67–95. https://doi.org/10.1016/j.infsof.2017.04.001

11. *Guan Q., Huang H., Luo W., Zhu S*. Semantics-based repackaging detection for mobile apps // Engineering Secure Software and Systems: 8th International Symposium. 2016. No. 8. P. 89–105. https://doi.org/10.1007/978-3-319-30806-7_6

12. *Desnos A*. Android: Static analysis using similarity distance / Desnos A. // 2012 45th Hawaii international conference on system sciences. 2012. P. 5394–5403.

https://doi.org/10.1109/HICSS.2012.114

13. *Zhauniarovich Y., Gadyatskaya O., Crispo B., La Spina F., Moser E*. FSquaDRA: Fast detection of repackaged applications // Data and Applications Security and Privacy XXVIII: 28th Annual IFIP WG 11.3 Working Conference. 2014. No. 28. P. 130–145. https://doi.org/10.1007/978-3-662-43936-4_9

14. *Li L., Bissyandé TF., Klein J*. Simidroid: Identifying and explaining similarities in android apps // 2017 IEEE Trustcom/BigDataSE/ICESS. 2017. P. 136–143. https://doi.org/10.1007/s11390-019-1918-8

15. The Java® Virtual Machine Specification // Oracle.
URL: https://docs.oracle.com/javase/specs/jvms/se7/html/

16. Android Runtime (ART) and Dalvik // Android Open Source Project.
URL: https://source.android.com/docs/core/runtime/.

17. *Ratazzi E.P*. Understanding and improving security of the Android operating system // PhD dissertation; Syracuse University, 2016.
URL: https://surface.syr.edu/etd/592/

18. *Cesare S., Xiang Y*. Software similarity and classification – 1. Springer London, 2012. 88 p. https://doi.org/10.1007/978-1-4471-2909-7

19. *Jones J*. Abstract Syntax Tree Implementation Idioms // Proceedings of the 10th conference on pattern languages of programs (plop2003). 2003. P. 26.
URL: https://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf

20. *Heck A.J.P.* OOP: Class Hierarchy // Persoonlijke pagina's van FNWI-medewerkers Personal pages of Science staff.
URL: https://staff.fnwi.uva.nl/a.j.p.heck/Courses/JAVAcourse/ch3/s1.html

21. *Ferrante J., Ottenstein K.J., Warren J.D*. The program dependence graph and its use in optimization // ACM Transactions on Programming Languages and Systems (TOPLAS). 1987. No. 9 (3). P. 319–349. https://doi.org/10.1145/24039.24041

22. *Callahan D., Carle A., Hall M.W., Kennedy K*. Constructing the procedure call multigraph // IEEE Transactions on Software Engineering. 1990. No. 16(4). P. 483–487. https://doi.org/ 10.1109/32.54302

23. *Allen F.E*. Control flow analysis // ACM Sigplan Notices. 1970. No. 5(7). P. 1–19. https://doi.org/10.1145/800028.808479

24. *Kruegel C., Kirda E., Mutz D., Robertson W., Vigna G*. Polymorphic worm detection using structural information of executables // Recent Advances in Intrusion

Detection: 8th International Symposium. 2006. No. 8. P. 207–226. https://doi.org/10.1007/11663812

25. *Marcelli A., Quer S., Squillero G*. The maximum common subgraph problem: A portfolio approach // arXiv:1908.06418 preprint. 2019. URL: https://www.researchgate.net/publication/335258488_The_Maximum_Common_Subgraph_Problem_A_Portfolio_Approach

26. *Abu-Aisheh Z., Raveaux R., Ramel J.Y., Martineau P.* An exact graph edit distance algorithm for solving pattern recognition problems // 4th International Conference on Pattern Recognition Applications and Methods. 2015. No. 1. https://doi.org/10.5220/0005209202710278

27. *Левенштейн В.И.* Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии наук СССР. 1965. № 163.4. С. 845–848.

28. Критерии сходства программ // ООО «АйТи-Лекс». URL: http://www.it-lex.ru/legal-cases/skhodstvo-programm/

29. *Myles G., Collberg C*. K-gram based software birthmarks // Proceedings of the 2005 ACM symposium on Applied computing. 2005. P. 314–318. https://doi.org/10.1145/1066677.1066753

30. *Liu C., Chen C., Han J., Yu P.S*. GPLAG: detection of software plagiarism by program dependence graph analysis // Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. 2006. . 872–881. https://doi.org/10.1145/1150402.1150522

31. Свидетельство о государственной регистрации программы для ЭВМ № 2023665834 Российская Федерация. Система автоматизации численной оценки сходства Android-приложений: № 2023664873: заявл. 16.07.2023: опубл. 20.07.2023 / В.В. Петров. – EDN NELKIS.

32. *Петров В.В.* Система автоматизации численной оценки сходства Android-приложений // Научный сервис в сети Интернет: труды XXV Всероссийской научной конференции (18–21 сентября 2023 г., онлайн). М.: ИПМ им. М.В. Келдыша, 2023. С. 283–297. https://doi.org/10.20948/abrau-2023-33

AUTOMATED SYSTEM FOR NUMERICAL SIMILARITY EVALUATION OF ANDROID APPLICATIONS

V. V. Petrov [0009-0004-4213-7328]

Institute of Information Technology and Intelligent Systems, Kazan (Volga region) Federal University, Kremlyovskaya ul., 35, Kazan, 420008

valeryvpetrov.itis@gmail.com

Abstract

This paper is devoted to the design and development of a system for automating numerical similarity assessment of Android applications. The task of application similarity evaluation is reduced to the similarity evaluation of sets of control flow graphs constructed based on code from classes.dex files of applications. The similarity value was calculated based on the similarity matrix. The algorithms of graph editing and Levenshtein distance were used to compare control flow graphs. Application similarity criteria were formulated and their representation forms were investigated. Types of Android application models and methods of their construction are presented. A prototype of the system for automating the numerical evaluation of Android-applications similarity is developed. Optimization of the software solution is performed with the help of parallel programming tools. Experiments are carried out and the conclusion is made about the ability of the developed system to detect similarities between Android applications.

Keywords*: Android application similarity, program similarity, similarity matrix, control flow graph edit distance, similarity matrix visualisation, control flow graph.*

REFERENCES

1. *Zhou W., Zhou Y., Jiang X., Ning P.* Detecting repackaged smartphone applications in third-party android marketplaces // Second ACM conference on Data and Application Security and Privacy. 2012. P. 317–326. https://doi.org/10.1145/2133601.2133640

2. *Crussell J., Gibler C., Chen H.* Attack of the clones: Detecting cloned applications on android markets // European Symposium on Research in Computer Security. 2012. P. 37–54. https://doi.org/10.1007/978-3-642-33167-1_3

3. Market Shocker! Iron Soldiers XDA Beta Published by Alleged Thief // Android Headline. URL: https://www.androidheadlines.com/2011/01/market-shocker-iron-soldiers-xda-beta-published-by-alleged-thief.html.

4. Fake Mobile Apps Steal Facebook Credentials, Cryptocurrency-Related Keys // TREND MICRO. URL: https://www.trendmicro.com/en_us/research/22/e/fake-mobile-apps-steal-facebook-credentials--crypto-related-keys.html.

5. Android App Bundle frequently asked questions // Android developers. URL: https://developer.android.com/guide/app-bundle/faq

6. *Akhunzada A., Sookhak M., Anuar N.B., Gani A., Ahmed E., Shiraz M., Furnell S., Hayat A., Khan M.K.* Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directions // Journal of Network and Computer Applications. 2015. No. 48. P. 44–57. https://doi.org/10.1016/j.jnca.2014.10.009

7. *Chen J., Alalfi M.H., Dean T.R., Zou Y.* Detecting android malware using clone detection // Journal of Computer Science and Technology. 2015. No. 30. P. 942–956. https://doi.org/10.1007/s11390-015-1573-7

8. *Wang H., Guo Y., Ma Z., Chen X.* Wukong: A scalable and accurate two-phase approach to android app clone detection // Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015. P. 71–82. https://doi.org/10.1145/2771783.2771795

9. *Chen K., Liu P., Zhang Y.* Achieving accuracy and scalability simultaneously in detecting application clones on android markets // Proceedings of the 36th International Conference on Software Engineering. 2014. P. 175–186. https://doi.org/10.1145/2568225.2568286

10. *Li L., Bissyandé TF., Papadakis M., Rasthofer S., Bartel A., Octeau D., Klein J., Traon L*. Static analysis of android apps: A systematic literature review // Information and Software Technology. 2017. No. 88. P. 67–95. https://doi.org/10.1016/j.infsof.2017.04.001

11. *Guan Q., Huang H., Luo W., Zhu S*. Semantics-based repackaging detection for mobile apps // Engineering Secure Software and Systems: 8th International Symposium. 2016. No. 8. P. 89–105. https://doi.org/10.1007/978-3-319-30806-7_6

12. *Desnos A*. Android: Static analysis using similarity distance / Desnos A. // 2012 45th Hawaii international conference on system sciences. 2012. P. 5394–5403. https://doi.org/10.1109/HICSS.2012.114

13. *Zhauniarovich Y., Gadyatskaya O., Crispo B., La Spina F., Moser E*. FSquaDRA: Fast detection of repackaged applications // Data and Applications Security and Privacy XXVIII: 28th Annual IFIP WG 11.3 Working Conference. 2014. No. 28. P. 130–145. https://doi.org/10.1007/978-3-662-43936-4_9

14. *Li L., Bissyandé TF., Klein J*. Simidroid: Identifying and explaining similarities in android apps // 2017 IEEE Trustcom/BigDataSE/ICESS. 2017. P. 136–143. https://doi.org/10.1007/s11390-019-1918-8

15. The Java® Virtual Machine Specification // Oracle. URL: https://docs.oracle.com/javase/specs/jvms/se7/html/

16. Android Runtime (ART) and Dalvik // Android Open Source Project. URL: https://source.android.com/docs/core/runtime/.

17. *Ratazzi E.P*. Understanding and improving security of the Android operating system // PhD dissertation; Syracuse University, 2016. URL: https://surface.syr.edu/etd/592/

18. *Cesare S., Xiang Y*. Software similarity and classification – 1. Springer London, 2012. 88 p. https://doi.org/10.1007/978-1-4471-2909-7

19. *Jones J*. Abstract Syntax Tree Implementation Idioms // Proceedings of the 10th conference on pattern languages of programs (plop2003). 2003. P. 26. URL: https://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf

20. *Heck A.J.P.* OOP: Class Hierarchy // Persoonlijke pagina's van FNWI-medewerkers Personal pages of Science staff. URL: https://staff.fnwi.uva.nl/a.j.p.heck/Courses/JAVAcourse/ch3/s1.html

21. *Ferrante J., Ottenstein K.J., Warren J.D*. The program dependence graph and its use in optimization // ACM Transactions on Programming Languages and Systems (TOPLAS). 1987. No. 9 (3). P. 319–349. https://doi.org/10.1145/24039.24041

22. *Callahan D., Carle A., Hall M.W., Kennedy K*. Constructing the procedure call multigraph // IEEE Transactions on Software Engineering. 1990. No. 16(4). P. 483–487. https://doi.org/ 10.1109/32.54302

23. *Allen F.E*. Control flow analysis // ACM Sigplan Notices. 1970. No. 5(7). P. 1–19. https://doi.org/10.1145/800028.808479

24. *Kruegel C., Kirda E., Mutz D., Robertson W., Vigna G*. Polymorphic worm detection using structural information of executables // Recent Advances in Intrusion Detection: 8th International Symposium. 2006. No. 8. P. 207–226.

https://doi.org/10.1007/11663812

25. *Marcelli A., Quer S., Squillero G*. The maximum common subgraph problem: A portfolio approach // arXiv:1908.06418 preprint. 2019.
URL: https://www.researchgate.net/publication/335258488_The_Maximum_Common_Subgraph_Problem_A_Portfolio_Approach

26. *Abu-Aisheh Z., Raveaux R., Ramel J.Y., Martineau P.* An exact graph edit distance algorithm for solving pattern recognition problems // 4th International Conference on Pattern Recognition Applications and Methods. 2015. No. 1.
https://doi.org/10.5220/0005209202710278

27. *Levenshtein V.I*. Binary codes with correction of deletions, insertions and substitutions of symbols // Reports of the USSR Academies of Sciences. 1965. No. 163.4. P. 845–848. URL:
https://www.mathnet.ru/links/ebbbb75259f2fb388db92a54ec642b7d/dan31411.pdf

28. Criteria for similarity of programs // IT-Lex LLC.
URL: http://www.it-lex.ru/legal-cases/skhodstvo-programm/.

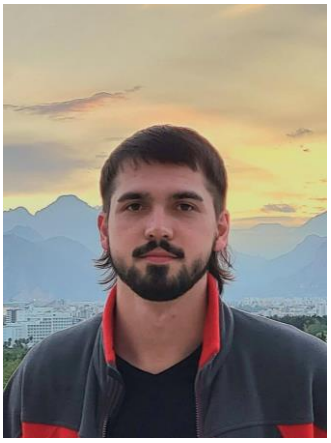
29. *Myles G., Collberg C*. K-gram based software birthmarks // Proceedings of the 2005 ACM symposium on Applied computing. 2005. P. 314–318.
https://doi.org/10.1145/1066677.1066753

30. *Liu C., Chen C., Han J., Yu P.S*. GPLAG: detection of software plagiarism by program dependence graph analysis // Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. 2006. P. 872–881.
https://doi.org/10.1145/1150402.1150522

31. Certificate of state registration of computer program No. 2023665834 Russian Federation. System of automation of numerical estimation of Android-applications similarity: No 2023664873: applied. 16.07.2023: published 20.07.2023 / V.V. Petrov. – EDN NELKIS.

32. *Petrov V.V*. Automation system for numerical assessment of the similarity of Android applications // Scientific service on the Internet: proceedings of the XXV All-Russian Scientific Conference (September 18–21, 2023, online). M.: IPM im. M.V. Keldysh, 2023. P. 283–297. https://doi.org/10.20948/abrau-2023-33

СВЕДЕНИЯ ОБ АВТОРЕ



ПЕТРОВ Валерий Владимирович – магистр программной инженерии, аспирант Института информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Valery PETROV – Magister of Software Engineering, Institute of Information Technology and Intelligent Systems, Kazan (Volga Region) Federal University.

Current scientific interests: Android application similarity, program similarity, static analysis, software modeling.

email: valeryvpetrov.itis@gmail.com;

ORCID: 0009-0004-4213-7328

Материал поступил в редакцию 24 апреля 2024 года

Переработанная версия – 16 мая 2024 года