Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра Системного Программирования

Фомин Сергей Александрович

Использование обучения с подкреплением в задаче автоматического тестирования мобильных приложений

Выпускная квалификационная работа

Научный руководитель:
к.ф.-м.н.
Турдаков Денис Юрьевич
Научный консультант:
Сорокин Константин Сергеевич

Москва, 2021

Аннотация

Использование обучения с подкреплением в задаче автоматического тестирования мобильных приложений

Фомин Сергей Александрович

Мобильные приложения являются основным способом взаимодействия пользователя с мобильным устройством. Этот факт заставляет разработчиков уделять больше внимания качеству получаемого продукта. Любые сбои или зависания приложения во время его использования могут негативно влиять на рейтинг продукта. Различные виды тестирования помогают вовремя найти и исправить ошибки. В данной работе исследуются подходы автоматического тестирования мобильных приложений путем взаимодействия с графическим интерфейсом. Задача этого тестирования состоит в том, чтобы проверить корректность функционирования готового приложения: отсутствие сбоев во время работы. Работа посвящена автоматическому тестированию на основе алгоритмов обучения с подкреплением, так как они показывают большую эффективность в сравнении с другими подходами. В ходе исследования реализованы несколько подходов Q-learning обучения с подкреплением для решения поставленной задачи. Среди них есть как простые алгоритмы, основанные на использовании эвристической функции награды, так и подходы, основанные на глубоких нейронных сетях. По результатам экспериментов выбирается лучший из реализованных алгоритмов и сравнивается с современным инструментом тестирования.

Abstract

A reinforcement learning based approach for automated exploratory testing of mobile applications

Содержание

1 Введение **4**

2 Постановка задачи **7**

3 Обзор существующих решений **9**

4 Исследование и построение решения задачи **13**

4.1 Общие сведения об обучении с подкреплением 13

4.2 Информация об окружении для тестирования 15

4.3 Настройка гиперпараметров 17

4.4 Приложения 18

4.5 Независимые от приложения подходы 19

4.5.1 Абстрактные состояния 19

4.5.2 Нейронная сеть 20

4.6 Зависимые от приложения подходы 23

4.6.1 Награда: частота нажатий 23

4.6.2 Награда: количество интерактивных элементов 25

4.6.3 Награда: расстояние между состояниями 26

4.7 Эпсилон жадная стратегия 28

4.8 Предобучение модели 28

4.9 Анализ результатов 30

4.10 Сравнение с Humanoid 32

5 Описание практической части **34**

6 Заключение **37**

Список литературы **38**

1 Введение

Операционная система Android является одной из самых востребованных на современном рынке мобильных устройств. Исследования показывают [1], что на мобильном устройстве в среднем установлено от 60 до 90 приложений и пользователь тратит на приложения более двух часов в день. Следовательно, проверка надежности и корректности функционирования приложения – важная задача. Как показывает статистика [2], большинство пользователей отказываются от использования приложения в случае его сбоя. Неправильное функционирование графического интерфейса не позволяет пользователям комфортно эксплуатировать весь потенциал приложений, который предоставляют разработчики. Ошибки приложения не только создают неудобства для пользователей, но и отрицательно сказываются на рейтинге продукта. Низкое качество Android-приложений можно объяснить недостаточным вниманием к тестированию в связи с быстрым развитием сферы [3]. Разработчики пренебрегают тестированием, поскольку этот процесс считается трудоемким, дорогостоящим и сопряженным с многократным повторением однотипных действий.

Избежать сбоев в работе мобильных приложений можно еще на этапе разработки, если основательно подходить к тестированию продукта. Тестировать приложения во время их разработки это один из ключевых элементов современного проектирования программного обеспечения. Причем необходимо делать это на всех этапах производства с помощью непрерывной интеграции и непрерывной доставки, иначе можно упустить мелкие ошибки и недочеты, которые в результате превратятся в негативный пользовательский опыт. Одним из основных видов проверки конечного продукта является тестирование приложения через взаимодействие с графическим интерфейсом. Этот вид тестирования проверяет отсутствие сбоев в работе приложения до того, как оно будет выпущено на рынок. Суть такого тестирования заключается в том, чтобы путем взаимодействия с разными интерактивными элементами посещать различные состояния приложения. Чем больше состояний будет покрыто во время тестирования, тем больше уверенность в том, что продукт исправен. Важно, что процесс тестирования происходит без доступа к исходному коду приложения.

Автоматическое тестирование приложений начинается с создания тестовых примеров, которые представляют из себя последовательности действий пользователя при взаимодействии с приложением. Разработка тестовых примеров обычно занимает много

времени, так как производится вручную. Еще чаще тестирование проводится вовсе без написания тестовых примеров: ручное взаимодействие с устройством. Из-за разнообразия приложений на рынке и большого количества способов взаимодействия с ними такое тестирование становится дорогим и неэффективным. На практике можно столкнуться с тем, что тестирование занимает несколько часов, а небольшие изменения в приложении могут привести к новым ошибкам, что требует повторного тестирования [4]. Поэтому компании часто выпускают бета-версии своих продуктов, чтобы тестированием занимались пользователи.

За последние 10 лет было придумано и разработано много альтернативных инструментов, предназначенных для автоматизации данного тестирования [5–11]. Схема автоматического тестирования состоит из множества повторений одних и тех же действий: получение состояния устройства, генерация действия согласно алгоритму, выполнение действия на устройстве. Самым важным этапом является разработка алгоритма генерации нажатия, поскольку от него зависит общая эффективность процесса тестирования. Существующие подходы занимаются исследованием именно этого этапа. Однако оптимального инструмента тестирования все еще не существует [12]. Причиной низкого покрытия тестирования является трудоемкость изучения сложных состояний приложения, достижимых только через конкретные последовательности действий. Исправить ситуацию можно с помощью интеллектуальных систем, основанных на алгоритмах машинного обучения. Одной из таких систем является недавно изобретенная связка тестирующей системы DroidBot [13] и нейросетевого алгоритма генерации действий Humanoid [8].

Для измерения эффективности того или иного подхода используются различные метрики. Из самых важных показателей эффективности [14] можно выделить следующие:

- количество уникальных состояний, открытых во время тестирования
- количество открытых Активностей приложения (соответствует принятому в англоязычном сообществе термину Activity)
- покрытие кода приложения
- количество найденных сбоев и зависаний

Увеличение значений каждой из этих метрик способствует повышению эффективности тестирования приложений. Стоит упомянуть, что первые две метрики опираются на понятие состояния. Активность приложения – некоторое обобщенное понятие состояния, введенное разработчиками Android. Важно понимать, что представление состояния влияет на эффективность тестирования. Используемое в этой работе представление будет описано в главе 4.

В данной работе исследуются подходы, которые генерируют тестовые примеры на основе машинного обучения – обучения с подкреплением [15]. Техника обучения с подкреплением использует метод проб и ошибок для выработки правильной последовательности действий, способной открывать новые состояния. Идея заключается в том, что каждое взаимодействие в каждом состоянии оценивается некоторой функцией награды. Функция награды формируется так, чтобы давать большее вознаграждение за полезные действия и штрафовать за бесполезные. В процессе исследования приложения алгоритм учится предсказывать какое действие приведет к большей суммарной награде и тем самым позволит исследовать как можно больше состояний.

Данная работа имеет следующую структуру. В главе 2 дается формальная постановка задачи. В ней описываются цели работы и ограничения, которым должно удовлетворять решение. В главе 3 делается обзор существующих решений для автоматического тестирования мобильных приложений. Затем рассматриваются работы, в которых применяются подходы обучения с подкреплением для решения похожих задач. В 4-ой главе проводится исследование поставленной задачи и построение алгоритмов ее решения, в том числе проверяются различные эвристики, потенциально способные повысить производительность тестирования. В этой же главе проводятся все необходимые эксперименты для определения лучшего из рассмотренных подходов. Затем лучший подход сравнивается с современным инструментом тестирования Humanoid [8].

2 Постановка задачи

Основной целью данной работы является исследование методов обучения с подкреплением в задаче тестирования мобильных приложения на операционной системе Android через взаимодействие с графическим интерфейсом. Под тестированием в данной работе понимается проверка правильности функционирования готового приложения: отсутствие сбоев во время работы. Путем взаимодействий с графическим интерфейсом нужно обойти как можно больше состояний приложения и убедиться, что эти взаимодействия не вызывают сбоев.

Все рассмотренные подходы обучения с подкреплением должны быть внедрены в систему тестирования DroidBot [13]. Эксперименты, измеряющие эффективность тестирования должны быть выполнены многократно с последующим усреднением результатов. Выводы об эффективности того или иного подхода должны быть сделаны на основе следующих метрик качества:

- Количество уникальных состояний, посещенных во время тестирования;
- Количество открытых Активностей приложения;

Все тесты должны проводиться на заранее выбранных приложениях. Приложения для тестирования должны быть различных категорий и разных степеней сложности.

Рассматриваемые подходы должны удовлетворять следующим требованиям:

- тестирование должно выполняться полностью автоматически, то есть без вмешательства пользователя;
- тестирование не должно использовать исходный код приложения;
- тестирование должно быть максимально приближено к человеческим взаимодействиям;
- тестирование должно быть ограниченно по времени;

Третий пункт требований означает, что в работе не рассматриваются подходы, которые заведомо далеки от человеческих взаимодействий. Например, подходы с частыми

случайными нажатиями. Последний пункт требований означает, что более эффективным подходом к тестированию в этой работе считается тот, который за фиксированное время показал наибольшее значение введенных ранее метрик.

Для чистоты эксперимента тестирование всех подходов должно проводиться на одной и той же ЭВМ.

Общий вывод об эффективности подходов обучения с подкреплением в задаче тестирования мобильных приложений должен быть сделан по результатам сравнения с тестирующей системой глубокого обучения Humanoid [8].

3 Обзор существующих решений

Исследователи уже более десяти лет ведут работы в направлении автоматизации тестирования мобильных приложений на основе взаимодействия с графическим интерфейсом пользователя. В этой части будут описаны существующие подходы к решению похожих задач и выделены их преимущества и недостатки.

Для начала стоит отметить, что все существующие подходы можно разбить на три группы:

- подходы, основанные на случайных нажатиях
- подходы, основанные на моделях взаимодействия
- подходы, основанные на алгоритмах оптимального поиска

Подходы, основанные на случайных нажатиях – одни из самых популярных методов обнаружения сбоев в приложении. Идея такого тестирования проста: с большой частотой генерируются нажатия на экран в случайной позиции. Этот метод может быть пригодным для эффективного стресс-тестирования (тестирование за пределами нормального использования приложения). Например, Android Monkey [7] – инструмент тестирования по стратегии «черный ящик» поставляемый вместе с Android SDK (Software Development Kit). Благодаря своей простоте этот инструмент приобрел значительную популярность в сообществе разработчиков. Помимо простоты, он продемонстрировал хорошую совместимость с множеством Android платформ, что сделало его промышленным стандартом [16]. Однако Android Monkey требуется много времени для генерации сложной последовательности событий, способной посещать необычные состояния. В последовательности взаимодействий, генерируемой этой стратегией, часто встречаются повторяющиеся действия, которые не приводят к исследованию новых состояний. Также недостатком этого подхода являются многочисленные нажатия на неинтерактивные элементы, что вовсе не может изменить текущего состояния приложения. В связи с этим был разработан более совершенный подход, призванный исправить недостатки Monkey – DynoDroid [6]. В нем используются дополнительные эвристики, благодаря которым случайный выбор действия зависит от состояния в котором находится тестируемое приложение. Одна из основных модификаций этого инструмента – возможность

использовать заранее прописанный пользователем сценарий при попадании в определенные состояния. Это дает значительный прирост производительности тестирования, но требует вмешательства пользователя в процесс формирования тестовых нажатий. Общий недостаток случайных стратегий в том, что сбои и ошибки, найденные во время тестирования, трудно воспроизводимы. Что еще более важно, такой подход может генерировать тысячи нажатий в секунду, что нехарактерно для стандартных сценариев использования смартфона.

Тестирование на основе моделей предполагает использование некоторой модели для представления взаимодействия пользователя с графическим интерфейсом приложения. Модель разрабатывается вручную или автоматически путем исследования кода или непосредственного взаимодействия с приложением. Тестирование на основе модели может производиться с использованием разных стратегий: обход в глубину, обход в ширину, гибридная или стохастическая. Важным является то, что этот подход чувствителен к точности построенной модели. В частности, слишком чувствительная модель, построенная с учетом малейших изменений графического интерфейса не сможет эффективно исследовать новые состояния, так как любое нажатие будет приводить к их смене. Наоборот, если модель приложения описывается слишком обобщенными состояниями это может привести к потере эффективности тестирования, так как смена состояния может быть пропущена. Метод A^3E [9] исследует приложения в два этапа: статический анализ кода приложения для построения высокоуровневой модели взаимодействия, и целевое исследование, которое на основе модели приложения позволяет быстро посещать состояния, сложно достижимые при нормальном использовании. Однако, A^3E для представления состояния использует Активность, без учета того, что одна Активность может содержать множество мелких состояний. Это приводит к ухудшению производительности тестирования. Еще один современный подход, основанный на модели взаимодействия - Humanoid [8]. Этот подход использует глубокие нейронные сети для предсказания нажатий в текущем состоянии. Основная идея этого подхода в том, чтобы обучить модель взаимодействовать с приложением так, как это делал бы человек. На основе набора данных пользовательских взаимодействий с приложением, глубокая нейронная сеть пытается выучить основные шаблоны взаимодействий и использовать их для тестирования. Этот подход превосходит многие аналогичные инструменты. Именно с этим подходом будут сравниваться методы обучения с подкреплением.

Основной представитель третьей группы методов – Sapienz [11]. Этот подход основан на многоцелевом тестировании. Sapienz пытается добиться сразу нескольких целей: оптимизации длины тестовых последовательностей, максимальный охват состояний приложения и максимальное количество найденных ошибок. С помощью генетических алгоритмов удается найти оптимальную стратегию тестирования, способную выполнять сразу все поставленные цели.

Современные работы в этой области ведутся над моделе-ориентированными методами, так как они показывают передовые результаты и к ним можно применять алгоритмы машинного обучения. Уже есть работы, которые пытаются внедрить алгоритмы машинного обучения в тестирования мобильных приложений через графический интерфейс [8, 17]. Поставленная задача хорошо подходит под группу алгоритмов машинного обучения – обучение с подкреплением. Есть несколько работ, которые описывают применение подходов обучения с подкреплением для тестирования разного вида программ.

Первая попытка внедрения алгоритма обучения с подкреплением Q-learning для тестирования приложений была сделана в 2018 году [18]. Исследователи предложили подход под названием QBE. Этот подход, на основе заранее построенной модели, способен извлекать сложные последовательности действий, которые приводят к сбоям. Таким образом, перед тестированием необходимо собрать информацию о том, какие жесты пользователя приводят к ошибкам. Функция награды, которая является основным элементом стратегий обучения с подкреплением, награждает алгоритм за достижения состояний с ошибками и тем самым мотивирует чаще выполнять действия, приводящие к таким состояниям. Благодаря примитивному представлению состояния – количество интерактивных элементов в текущий момент времени, этот подход может построить общую Q-таблицу для всех приложений.

С 2018 года область обучения с подкреплением в задаче тестирования мобильных приложений стала активно развиваться. Появились работы, использующие Q-learning алгоритм, целью которых является обучение под единственное приложение [19,20]. Идея таких подходов в том, чтобы объединить этапы обучения и тестирования, то есть формирование Q-таблицы для приложения происходит одновременно с тестированием приложения. Опять же, основной элемент, устанавливающий цель тестирования – функция награды. Так в [19] используется функция награды в виде числа, обратно пропорционального количеству нажатий на данный интерактивный элемент в данном состоянии.

Это позволяет производить тестирование без повторений одних и тех же действия до тех пор, пока не будут выполнены другие действия в этом состоянии. В [20] используют похожий подход, однако функция награды модифицируется. Помимо обратной частоты нажатий к функции добавляется слагаемое, описывающее степень изменения нового состояния: отношение новых интерактивных элементов ко всем интерактивным элементам.

Еще одна современная работа использующая обучение с подкреплением – DRIFT [17]. В DRIFT используется идея моделирования Q-таблицы на основе графовой нейронной сети. Хотя цель этой работы – тестирование программ под операционной системой Windows, идея моделирования Q-таблицы, кажется перспективной и будет использоваться в этой работе.

Несмотря на то, что уже существуют подходы, использующие алгоритмы обучения с подкреплением для тестирования приложений, все они либо примитивны, либо не подходят под ограничения поставленной задачи. Так, в работах [18, 19] рассматриваются слишком простые эвристики. В [17] рассматривается тестирование приложений для другой операционной системы.

4 Исследование и построение решения задачи

Перед тем как рассказывать о подходах к решению поставленной задачи нужно дать формальное описание того, как работают алгоритмы обучения с подкреплением.

4.1 Общие сведения об обучении с подкреплением

Обучение с подкреплением [15] – это одна их больших ветвей машинного обучения. В отличие от других ветвей, таких как обучение с учителем или без учителя, обучение здесь происходит путем учета вознаграждений и наказаний за взаимодействие с окружающей средой. Идея таких подходов основана на концепции поведенческой психологии: методом проб и ошибок достичь максимального результата. В методах обучения с подкреплением основными компонентами являются среда и агент. Агент выступает в качестве объекта, который выполняет действия в среде для достижения определенной цели. В ходе выполнения последовательности действий агент получает информацию о среде. Среда это пространство, в котором агент учится добиваться поставленных целей.

Помимо агента и среды в обучении с подкреплением есть еще три сущности: функция вознаграждения, действия и состояние. Состояние описывает текущую ситуацию в окружающей среде. В каждом состоянии у агента есть определенное количество возможных действий. За каждое действие предусмотрена награда, которая оценивает важность того или иного действия в текущем состоянии. Задача агента в том, чтобы в каждом состоянии научиться выбирать такое действие, которое приводит его к максимальной суммарной награде. Попробовав все действия в текущем состоянии, агент научится определять какое из них с большей вероятностью приведет его к поставленной цели.

Во всех существующих подходах к тестированию программного обеспечения через взаимодействие с графическим интерфейсом, использующих подход обучения с подкреплением, применяется техника Q-learning. Q-learning – это самый распространенный и хорошо изученный подход обучения с подкреплением. В нашем случае эта техника используется для поиска оптимального действия для данного состояния мобильного приложения. Различные стратегии и функции наград устанавливают правила, по которым агент должен отдавать предпочтение определенному действию. После выбора и воспроизведения действия среда изменяет свое состояние на новое, и агент получает на-

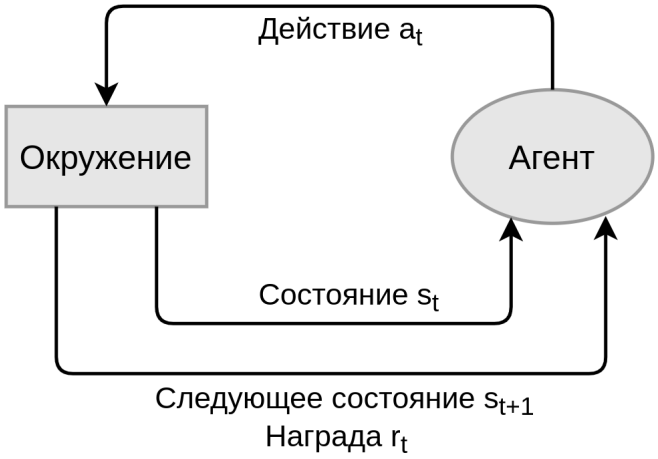


Рис. 1: Один раунд обучения с подкреплением.

граду за выполненное действие. На Рис. 1 показана классическая схема одного раунда взаимодействия агента и среды с помощью техники Q-learning:

1. Агент получает от окружения состояние s_t.
2. Агент выбирает и воспроизводит действие $a_t \in A(s_t)$, где $A(s_t)$ – множество доступных действий в состоянии s_t.
3. Окружение рассчитывает значение функции награды $R(s_t, a_t)$ исходя из состояния s_t и выбранного действия в этом состоянии a_t.
4. Окружение рассчитывает новое состояние s_{t+1} исходя из состояния s_t и выбранного действия в этом состоянии a_t.
5. Агент получает значение награды r_t и переходит в состояние s_{t+1}.

Таким образом, цель агента полностью описывается с помощью функции награды. Поэтому в задачах обучения с подкреплением важно подобрать такую функцию награды, которая будет максимально соответствовать поставленной цели.

За выбор определенного действия в текущем состоянии отвечают так называемые Q-значения. Каждой паре (s_t, a_t) соответствует свое Q-значение, которое описывает потенциальную суммарную награду, которую может получить агент, выбрав действие a_t в состоянии s_t. Совокупность всех таких пар описывает Q-таблицу. Таким образом, выбор

действия агентом, сводится к тому, чтобы найти максимальное Q-значение в текущем состоянии.

Обучение же модели состоит в том, чтобы подобрать такие Q-значения, которые будут близко описывать потенциальную суммарную награду за выполнение определенного действия. Уравнение, которым описывается обновление Q-значений после получения награды выглядит следующим образом

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}), \tag{1}$$

где $R(s_t, a_t)$ – функция награды, γ – фактор дисконтирования, описывающий баланс между немедленной наградой и наградой за будущие действия. Таким образом, за большое число итераций Q-таблица стабилизируется и будет отражать реальную награду за каждое действие.

4.2 Информация об окружении для тестирования

Перед тем, как рассматривать конкретные алгоритмы и их реализации, нужно понимать как происходит взаимодействие с устройством. В качестве устройства во всех экспериментах будет выступать официальный эмулятор операционной системы Android. Удобным инструментом для программного взаимодействия с телефоном является утилита командной строки Android Debug Bridge (adb). Именно ее использует DroidBot [13] для взаимодействия с подключенным устройством. Все реализованные в работе алгоритмы работают на основе генератора тестов DroidBot. Он удобен тем, что может взаимодействовать с приложениями без модификации системы и самого приложения. Это делает DroidBot совместимым со всеми устройствами и приложениями. Также он предоставляет возможность подключать собственные стратегии тестирования, что актуально для этой работы. Еще одна причина, по которой был выбран этот инструмент – подробный отчет о тестировании, из которого можно получить нужные метрики.

Для лучшего понимания процесса тестирования, весь цикл взаимодействия тестового генератора DroidBot и алгоритма генерации нажатий с устройством представлен на Рис. 2. Тестовый генератор способен получать текущее состояние приложения от устройства и воспроизводить нажатия на экране смартфона. Получив состояние от устройства, тестовый генератор обрабатывает его и в удобном виде передает алгорит-

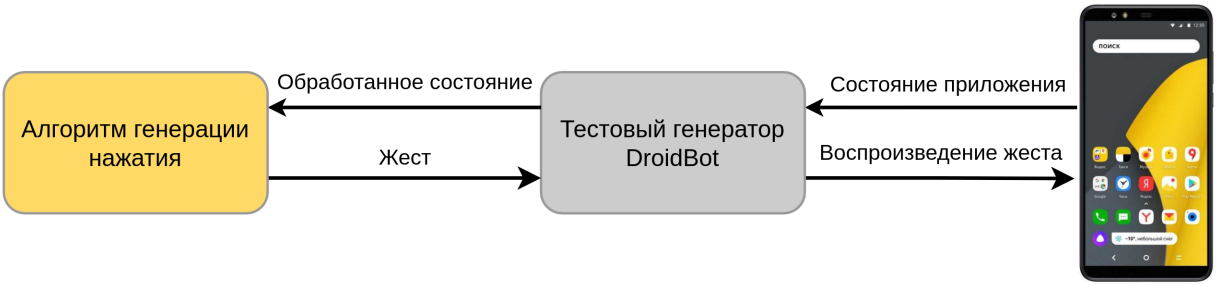


Рис. 2: Цикл взаимодействия устройства, тестового генератора DroidBot и алгоритма генерации нажатий.

му генерации нажатий. Алгоритм генерации нажатий путем взаимодействия с тестовым генератором получает обработанное состояние приложения и генерирует жест для воспроизведения на устройстве. Этот жест воспроизводится путем нажатия в нужную координату экрана, и цикл взаимодействия повторяется.

Как было указано в постановке задачи, в работе используется две метрики из четырех основных, введенных в главе 1. Их числовые значения наиболее информативны и понятны. Они обе сопряжены с понятием состояния приложения. Проблема представления состояния приложения в данной работе решается следующим образом: будем считать, что состояния являются одинаковыми, если список возможных взаимодействий в этих состояниях совпадает. То есть состоянием в этой работе является список возможных взаимодействий в текущий момент времени. Действие в данном состоянии можно описать парой (тип действия, интерактивный элемент). Каждый интерактивный элемент имеет свои типы взаимодействия с ним. Эту информацию предоставляет тестовый генератор DroidBot. Таким образом, в каждый момент времени у нас есть доступ к текущему состоянию и всевозможным взаимодействиям в этом состоянии.

Что касается Активности приложения, то ее тоже можно сопоставлять с некоторым понятием состояния. Отличие состоит в том, что Активность приложения является более обобщенным понятием состояния, которое является единым для всех приложений и описывается классом языка программирования. Несколько состояний в нашем понимании могут находиться в одной Активности. Измерения только Активности приложения может быть недостаточно, так как приложение может содержать только одну Активность и при этом множество состояний. Таким образом, метрика Активности приложения всегда будет ниже метрики состояний, но при этом эти метрики независимы и

хорошо отражают степень покрытия приложения при тестировании. Если говорить про две другие метрики, то они не будут рассматриваться в этой работе. Причина отказа от метрики с покрытием кода состоит в том, что тестирование в данной работе проводится по стратегии «черный ящик»: без доступа к исходному коду. Отказ от метрики с количеством найденных ошибок связан с тем, что эксперименты для проверки эффективности тестирования проводятся на уже готовых приложениях, количество сбоев в которых крайне мало.

Чем больше типов взаимодействия с экраном смартфона будет поддержано во время тестирования, тем потенциально большего покрытия удастся достичь. Поэтому будем использовать все типы жестов, которые предоставляет интерфейс DroidBot: нажатие, двойное нажатие, удержание, вставка текста, горизонтальная и вертикальная прокрутка, кнопки «Меню» и «Назад».

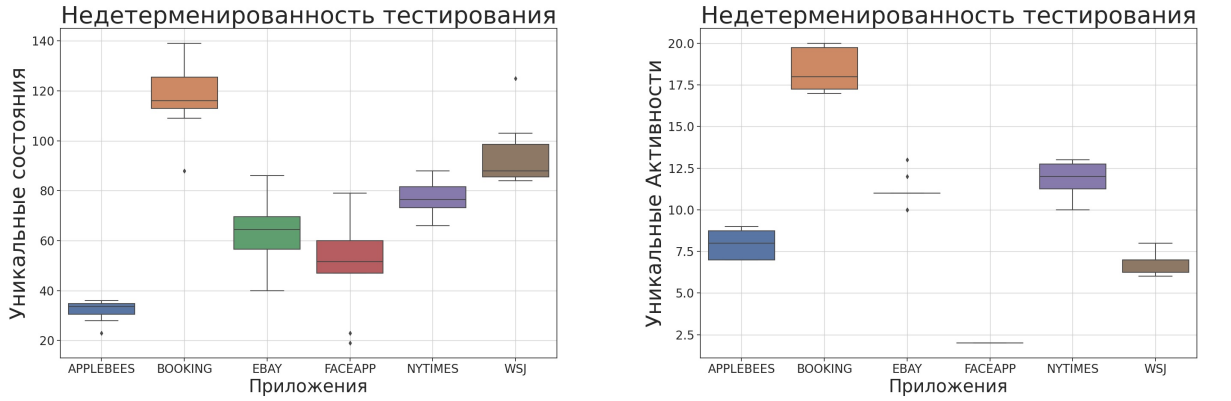
Как и требовалось в постановке, все эксперименты в работе будут проводиться на одной и той же ЭВМ. В нашем случае спецификация ЭВМ следующая: `Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 64 GB RAM, GeForce GTX 1080`.

4.3 Настройка гиперпараметров

В этой части работы будет определено два основных параметра для всех алгоритмов: количество запусков и время тестирования. Для проведения всех экспериментов был реализован базовый алгоритм Q-learning [19], подробное описание которого будет дано позже.

В некоторых алгоритмах, которые будут рассматриваться в дальнейшем, присутствует элемент случайности. Таким образом, тестирование от запуска к запуску может показывать различные значения метрик. Однако это не единственный источник недетерминированности. Так как запуски проводятся на эмуляторе реального устройства, то его работа может сопровождаться некоторыми зависаниями. В добавок ко всему долгая загрузка страниц приложения, наличие анимации и задержки Android Debug Bridge могут привести к различным результатам одного и того же алгоритма. Степень недетерминированности базового алгоритма можно оценить по Рис. 3. На нем представлены результаты тестирования шести приложений в виде диаграмм размаха. Хорошо видна необходимость проводить эксперименты по несколько раз и результат усреднять. Для всех дальнейших алгоритмов измерения будут проводиться по пять раз.

Постановка задачи требовала от алгоритма ограниченности по времени, так как в практических приложениях тестирование должно занимать разумное количество времени. Проведем эксперименты на базовом Q-learning алгоритме для вычисления оптимального времени тестирования. По результатам замеров на шести приложениях (Рис. 4) можно видеть, что графики зависимости метрик от времени выпуклы вверх. Это означает, что с течением времени эффективность поиска новых состояний уменьшается. Для некоторых приложений есть ощутимая разница между 10 и 15 минутами тестирования, но для большинства это несущественный прирост метрики. В качестве компромисса все дальнейшие эксперименты будут проводиться с ограничением времени 12 минут.



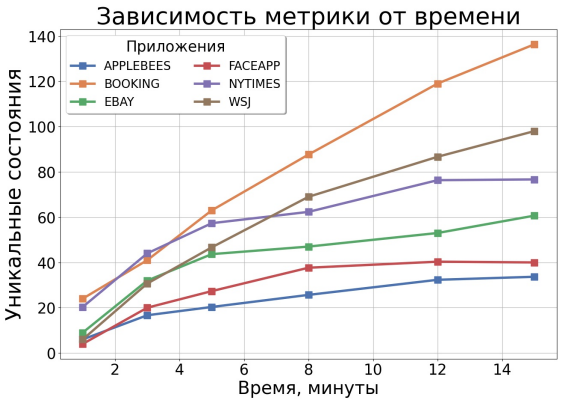
(a) Разброс метрики уникальных состояний.

(b) Разброс метрики уникальных Активностей.

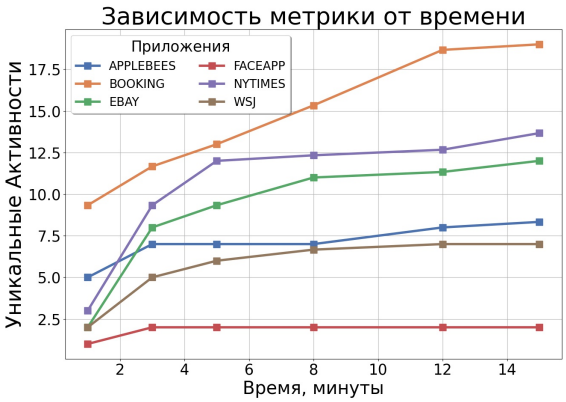
Рис. 3: Недетерминированность тестирования.

4.4 Приложения

Чтобы сделать процесс тестирования наиболее честным, возьмем для тестирования приложения различных категорий и степеней сложности. Всего будет использоваться 10 современных приложений: `Booking`, `EBAY`, `The New York Times`, `Applebee's`, `FaceApp`, `The Wall Street Journal`, `AliExpress`, `Domino's Pizza`, `ColorNote`, `Wikipedia`. В их числе есть как простые приложения с небольшим количеством состояний, так и масштабные приложения с платформы Google Play.



(a) Зависимость метрики состояний от времени.



(b) Зависимость метрики Активности от времени.

Рис. 4: Зависимость метрик от времени.

4.5 Независимые от приложения подходы

Идеальный инструмент тестирования мобильных приложений – это инструмент, который способен производить эффективное тестирование для произвольного приложения. Именно в поисках такого инструмента исследование начинается с алгоритмов обучения с подкреплением, которые обучают обобщенную модель, способную взаимодействовать с произвольным приложением. Такой подход связан с рядом упрощений. Например, введение обобщенного состояния и обобщенного действия, которые подходят под все приложения. Обучение таких моделей происходит онлайн (во время тестирования), так как только методом проб и ошибок можно научить Q-learning алгоритмы производить эффективные нажатия.

4.5.1 Абстрактные состояния

Первая модель, которая была реализована и внедрена в тестирующую систему DroidBot – это модель основанная на алгоритме, описанном в [18]. Это классическая техника Q-learning, в которой состояние описывается количеством интерактивных элементов в текущий момент времени, а действие в текущем стоянии описывается лишь типом жеста. Таким образом, модель в зависимости от количества интерактивных элементов в текущем состоянии должна предсказывать тип взаимодействия, приводящий к потенциально новым состояниям.

В качестве функции награды была выбрана величина, равная количеству интерактивных элементов в состоянии, в которое перешло приложение после воспроизведения действия. Эта величина способствует посещению состояний с потенциально большой вариативностью для исследования. Таким образом, алгоритм учится выбирать те действия, которые приводят его к «богатым» состояниям. Результаты тестирования этого подхода приведены в Таблице 1.

Представление состояния только количеством интерактивных элементов не может детально описать состояние приложения. В следующем пункте будет рассмотрено альтернативное представление состояния.

4.5.2 Нейронная сеть

Для улучшения предыдущего подхода, было решено представлять абстрактное состояние в виде изображения. А именно трехмерного тензора размером $(180, 320, 2)$. Можно заметить, что он представляет из себя две матрицы размера $(180, 320)$. Каждая из двух матриц состоит из нулей и единиц. Единицами на первой матрице выделены интерактивные элементы текущего экрана приложения, на второй – текстовые элементы. Таким образом удалось ввести общее состояние для всех приложений. Для обработки изображений хорошо зарекомендовали себя сверточные нейронные сети. Именно их и будем использовать для предсказания Q-значений.

В этом подходе сверточная нейронная сеть будет моделировать Q-таблицу. В классическом подходе, Q-таблица должна по паре (s_t, a_t) возвращать Q-значение. Для имитации такого возвращения нейронная сеть будет одновременно получать состояние и действие, а возвращать единственное значение – Q-значение. Чтобы на вход сверточной нейронной сети передать одновременно состояние и действие, модифицируем вход: будем добавлять к тензору состояния еще один бинарный слой размера $(180, 320)$. Этот слой будет выделять единицами точку прикосновения пользователя. Архитектура нейронной сети (Рис. 5) была выбрана небольшая, так как данных для обучения будет немного.

Обучение такой сети производится на каждой итерации взаимодействия. Небольшие наборы данных, собранные на всех предыдущих итерациях, постепенно обновляют веса модели и формируют нужную стратегию. Функция награды точно такая же, как и в предыдущем пункте: способствующая более широкому исследованию.

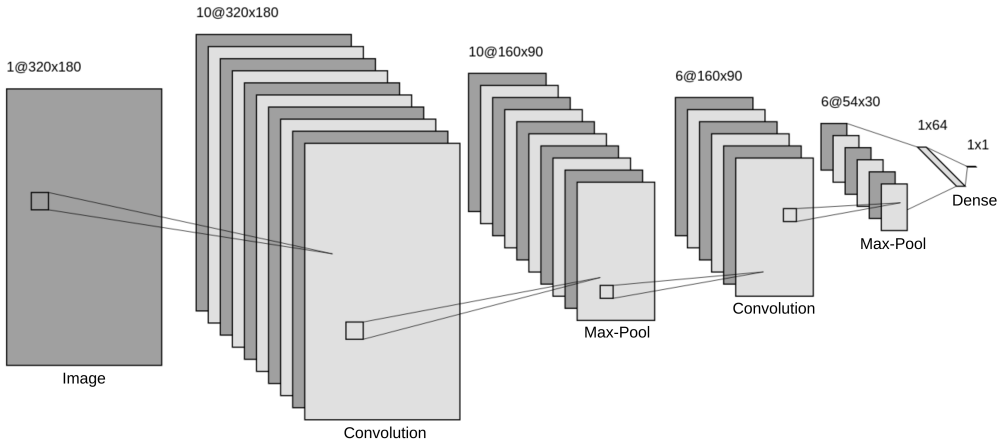


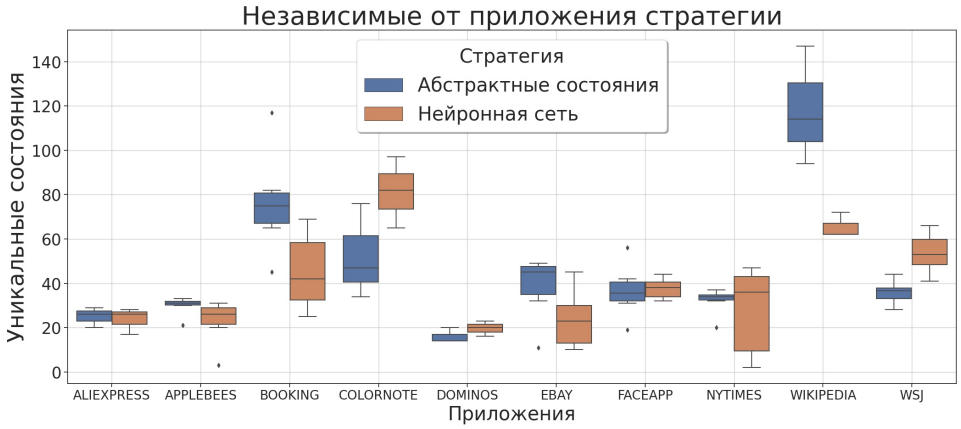
Рис. 5: Архитектура нейронной сети для независимого от приложения подхода.

Результаты тестирования этой стратегии приведены в Таблице 1. Для большей наглядности результаты первых двух подходов представлены на Рис. 6 в виде диаграмм размаха.

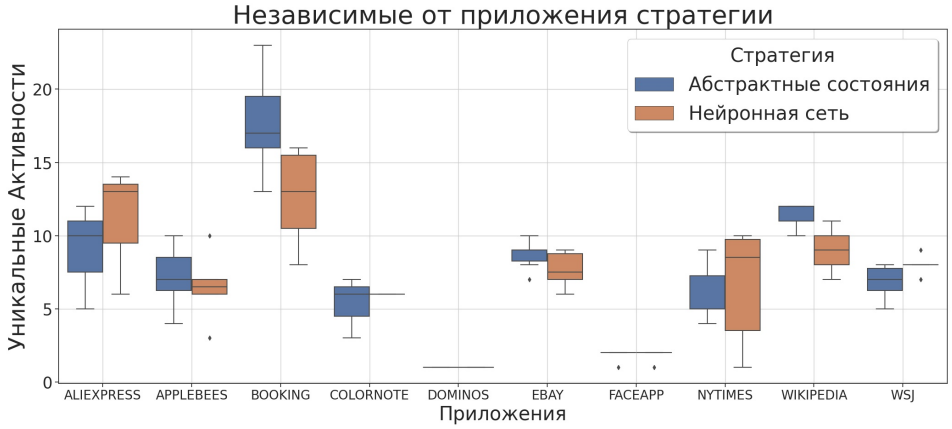
Приложения	Абстрактные состояния		Нейронная сеть	
ALIEXPRESS	**25**	9	24	**12**
APPLEBEES	**30**	**7**	23	**7**
BOOKING	**76**	**18**	45	13
COLORNOTE	52	**6**	**81**	**6**
DOMINOS	16	**1**	**20**	**1**
EBAY	**41**	**8**	24	6
FACEAPP	36	**2**	**37**	**2**
NYTIMES	**34**	**6**	27	**6**
WIKIPEDIA	**118**	**11**	65	9
WSJ	36	7	**53**	**8**

Таблица 1: Независимые от приложения стратегии тестирования. Метрика слева – уникальные состояния, справа – уникальные Активности.

Можно заметить, что на 6 из 10 приложениях подход основанный на использовании нейронной сети уступает классическому алгоритму Q-learning. В результате анализа



(a) Метрика: уникальные состояния.



(b) Метрика: уникальные Активности.

Рис. 6: Независимые от приложения стратегии тестирования.

было выяснено, что за время, отведенное под тестирование (12 минут), алгоритм не успевает собрать достаточно данных для обучения, и тем самым нейронная сеть не успевает выучить нужные шаблоны тестирования. Однако, на четырех приложениях нейронная сеть оказалась лучше. Это связано с тем, что на этих приложениях визуальная составляющая состояний не сильно изменяется при переходе в разные Активности. Можно заметить, что все четыре приложения имеют малое количество уникальных Активностей, что говорит о низкой вариативности структуры графического интерфейса. Это облегчает обучение нейронной сети.

4.6 Зависимые от приложения подходы

Повышения производительности тестирования приложения можно добиться, если перейти от идеи тестирования произвольного приложения к тестированию конкретного приложения. С целью улучшения метрик качества, перейдем к рассмотрению моделей, способных тестировать только то приложение, под которое они обучаются. Идея этих методов в том, что состояния и действия в Q-таблице теперь будут означать конкретные состояния и действия текущего приложения. Такая персонализация позволит обучить модель с учетом тонкостей приложения.

Как было сказано ранее, функция награды полностью определяет цель тестирования. Рассмотрим несколько функций наград, способных увеличить эффективность тестирования приложения.

4.6.1 Награда: частота нажатий

Наиболее простым Q-learning подходом для этих целей является модель [19]. У этого метода функция награды содержит только частоту нажатия определенного действия. Суть метода в том, что изначально Q-значение за любое действие в любом состоянии равно константе. Однако при очередном выборе действия Q-значение за это действие корректируется: уменьшается пропорционально количеству выполнений этого действия в текущем состоянии. Таким образом, уже протестированные действия текущего состояния будут иметь меньшее Q-значение. Q-значения в этом подходе корректируется согласно формуле (1). Функция награды в этом подходе выглядит следующим образом

$$R(s_t, a_t) = \frac{1}{count(s_t, a_t)}, \tag{2}$$

где $count(s_t, a_t)$ – количество раз, которое действие a_t воспроизводилось в состоянии s_t. Такой подход будет повторно посещать состояние только в том случае, если он уже посетил все оставшиеся состояния. Результаты тестирования этой стратегии приведены в Таблице 2.

Исходя из формулы (2) можно заметить, что такой выбор функции награды и обновления Q-значения – далекий от интеллектуальности подход. Действительно, функция награды зависит только от количества нажатий на интерактивный элемент текущего состояния, что неизбежно приводит к обычному обходу графа состояний. Единственный

момент, который отличает эту систему от стандартного обхода – различные значения γ для разных состояний. В этом и следующих подходах фактор дисконтирования γ формируется согласно следующей формуле

$$\gamma = 0.9e^{-0.1|events(s_{t+1})|},$$

где $events(s_{t+1})$ – все возможные взаимодействия в состоянии s_{t+1}. Такое определение γ обоснованно интуитивным предположением о том, что для состояний с меньшим количеством интерактивных элементов нужно отдавать предпочтение будущим наградам. Однако фактор дисконтирования влияет только на степень учета будущих наград. Для увеличения эффективности необходимо усовершенствовать функцию награды. Будем использовать некоторые эвристические предположения для дальнейшего улучшения метрик.

Приложения	Обратная частота нажатий		Кол-во интерактивных элементов		Обратное кол-во инт. элементов		Расстояние между состояниями	
ALIEXPRESS	**37**	**11**	34	10	32	9	30	9
APPLEBEES	32	7	30	7	**34**	**8**	32	7
BOOKING	**116**	**18**	85	**18**	85	16	86	16
COLORNOTE	142	5	153	**6**	**183**	5	164	5
DOMINOS	**35**	**1**	34	**1**	30	**1**	30	**1**
EBAY	**67**	**11**	45	10	40	**11**	46	**11**
FACEAPP	**54**	**2**	48	**2**	48	**2**	39	**2**
NYTIMES	**77**	11	45	11	49	**12**	74	11
WIKIPEDIA	74	9	77	12	**95**	**14**	46	10
WSJ	96	7	**117**	**8**	85	7	82	6

Таблица 2: Зависимые от приложения стратегии тестирования. Метрика слева – уникальные состояния, справа – Активности.

4.6.2 Награда: количество интерактивных элементов

Чтобы алгоритм стремился научиться открывать больше новых состояний, ему нужно давать большую награду за переход в состояния, которые потенциально могут привести к более широкому исследованию. Модифицируем функцию награды таким образом, чтобы алгоритм получал большую награду за переход в состояния с большим числом интерактивных элементов. Это позволит учитывать количество интерактивных элементов не только в долгосрочной перспективе (за счет фактора дисконтирования), но и в мгновенной награде. Функция награды в этом случае будет выглядеть так

$$R(s_t, a_t) = \frac{|events(s_{t+1})|}{count(s_t, a_t)},$$

где $count(s_t, a_t)$ – количество раз, которое действие a_t воспроизводилось в состоянии s_t, $events(s_{t+1})$ – все возможные взаимодействия в состоянии s_{t+1}. Результаты тестирования этой стратегии приведены в Таблице 2.

Можно рассмотреть обратную мотивацию. Посещение состояний с большим количеством интерактивных элементов находится в приоритете для Q-learning алгоритма, так как Q-значение вычисляется с учетом максимума на следующем состоянии. Таким образом, вероятность, что максимум будет больше для состояния с большим количество интерактивных элементов, выше, чем для состояний с малым количеством интерактивных элементов. Следовательно, алгоритм может пропускать такие «бедные» состояния. Попробуем исправить эту ситуацию тем, что увеличим награду за посещение состояний с малым количеством интерактивных элементов. Это способствует исследованию граничных состояний перед тем, как заходить в потенциально «богатые» состояния. Функция награды выглядит следующим образом

$$R(s_t, a_t) = \frac{1}{|events(s_{t+1})|count(s_t, a_t)},$$

где $count(s_t, a_t)$ – количество раз, которое действие a_t воспроизводилось в состоянии s_t, $events(s_{t+1})$ – все возможные взаимодействия в состоянии s_{t+1}. Результаты тестирования этой стратегии приведены в Таблице 2.

4.6.3 Награда: расстояние между состояниями

Можно рассмотреть еще одну эвристику – если нажатие на экран приводит к состоянию, близкому к предыдущему (например, открыт выпадающий список), то скорее всего это действие менее ценное, чем то, которое привело бы к кардинально новому состоянию (новой Активности). Состояния приложения можно представлять в виде дерева, в котором интерактивные элементы с их атрибутами находятся в узлах, а вложенные элементы порождают иерархию дерева. Следовательно, мерой расстояния между состояниями может служить расстояние редактирования графов. Установим это расстояние как функцию награды и проведем измерения эффективности тестирования. Функция награды в этом случае будет выглядеть следующим образом

$$R(s_t, a_t) = \frac{dist(s_t, s_{t+1})}{count(s_t, a_t)},$$

где $count(s_t, a_t)$ – количество раз, которое действие a_t воспроизводилось в состоянии s_t, $dist(s_t, s_{t+1})$ – расстояние редактирования между графами состояний s_t и s_{t+1}, которое описывается следующей формулой

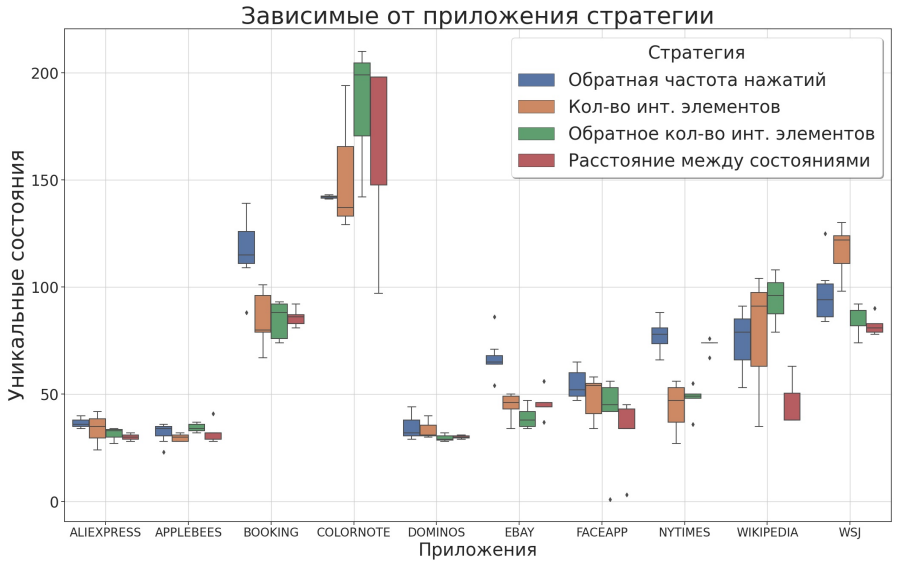
$$dist(s_t, s_{t+1}) = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(s_t,s_{t+1})} \sum_{i=1}^{k} c(e_i),$$

где $\mathcal{P}(s_t, s_{t+1})$ означает набор преобразования графа s_t к графу s_{t+1}, а $c(e) \geqslant 0$ равна стоимости каждой операции преобразования e. Стоимость вставки и удаления вершины в этой работе равна единице. Результаты тестирования этой стратегии приведены в Таблице 2.

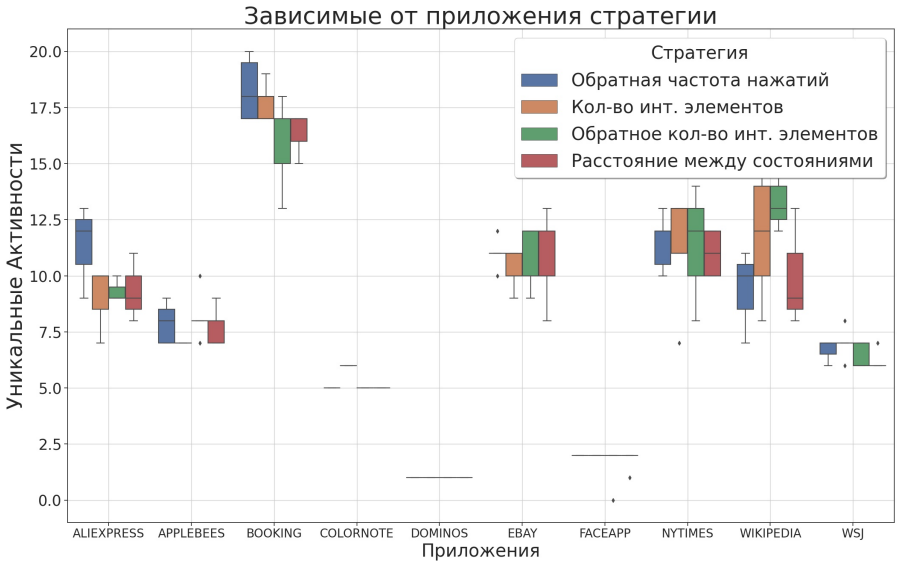
Для наглядности результаты тестирования всех стратегий, зависящих от приложения, представлены в виде диаграмм размаха на Рис. 7.

В сравнении результатов стратегий, независимых от приложения (Таблица 1), с результатами последних стратегий (Таблица 2) очевидно превосходство вторых. Несмотря на то, что приложение Wikipedia немного снизило показатели, метрики на остальных приложениях сильно возросли. Как и ожидалось, переход на обучение под конкретное приложение позволил повысить эффективность тестирования.

Если сравнивать между собой эвристики, которые использовались в функциях наград, то однозначного лидера выделить нельзя. С точки зрения метрики уникальных состояний, хороший результат показали награды за обратную частоту нажатий и об-



(a) Метрика: состояния.



(b) Метрика: Активности.

Рис. 7: Зависимые от приложения стратегии тестирования.

ратное количество интерактивных элементов. При анализе результатов на приложении Wall Street Journal лучшей стратегией оказывается та, которая считает более ценные

состояния с большим количеством интерактивных элементов. Функция наград с расстоянием между состояниями показала хороший результат на приложениях The New York Times и ColorNote, но все равно не стала лидером. Анализ показал, что близкие на внешний вид состояния могут иметь большое расстояние редактирования между деревьями этих состояний. Это не позволяет в полной мере использовать эту эвристику.

Для дальнейшего улучшения качества тестирования можно прибегнуть к приемам, которые используются во многих приложениях обучения с подкреплением и машинного обучения в целом. Рассмотрим применение некоторых из них в поставленной задаче.

4.7 Эпсилон жадная стратегия

Хорошей практикой обучения с подкреплением является добавление некоторой случайности на первых этапах обучения. Это позволяет агенту лучше изучить окружение, до того момента, пока алгоритм не обучится. Такая стратегия называется ε-жадная. Суть ее в том, что перед обучением задается число ε близкое к единице, его роль – определять вероятность, с которой агент будет делать случайное действие в текущем состоянии вместо того, чтобы соблюдать стратегию. По ходу обучения это число уменьшается, и через определенное количество времени агент действует только согласно стратегии. Такой подход позволяет на начальных этапах отдать предпочтение случайным действиям, нежели еще необученным алгоритмам. Этот метод позволяет производить более широкое исследование за счет смешивания двух алгоритмов: случайного и основанного на модели. Будем использовать в этом подходе функцию награды, описываемую формулой (2). Результаты тестирования этой стратегии приведены в Таблице 3.

4.8 Предобучение модели

В теории машинного обучения принято разделять этапы обучения и использования модели. В поставленной задаче можно также воспользоваться этим правилом. Сначала на длительное время запустить изучение приложения для формирования Q-таблицы, а потом прочитать из файла готовую таблицу и на ней проводить тестирование. Важно понимать, что при формировании таблицы не нужно учитывать количество посещений каждого состояния, иначе на этапе тестирования будет сложно достичь часто посещаемых состояний.

В дополнение к предыдущему приему, можно выполнять тестирование в несколько эпизодов. Одна и та же Q-таблица будет переходить от одного эпизода к другому, а каждый эпизод будет начинать тестирование приложения с самого начала. Таким образом, во время тестирования можно имитировать несколько сессий взаимодействия пользователя, что должно повысить эффективность тестирования. Будем проводить тестирование в пять эпизодов, первые четыре из которых будут этапами обучения Q-таблицы, а последний этапом тестирования.

Приложения	Эпсилон жадная стратегия		Предобученная модель		Предобученная модель с эпсилон жадной стратегией	
ALIEXPRESS	30	13	**80**	16	63	**18**
APPLEBEES	35	**9**	**68**	8	63	8
BOOKING	94	17	**107**	15	97	**18**
COLORNOTE	130	5	189	5	**190**	**6**
DOMINOS	37	**1**	**39**	**1**	36	**1**
EBAY	**60**	**12**	40	9	34	11
FACEAPP	63	**2**	**79**	**2**	59	**2**
NYTIMES	**68**	**13**	53	10	60	11
WIKIPEDIA	101	13	139	**14**	**154**	12
WSJ	84	8	117	8	**134**	**9**

Таблица 3: Улучшение стратегий тестирования. Метрика слева – уникальные состояния, справа – Активности.

На эпизодах обучения будем использовать функцию награды, равную числу интерактивных элементов в новом состоянии без учета количества посещений состояний. Таким образом удастся сформировать честную Q-таблицу, описывающую только количество интерактивных элементов. На эпизоде тестирования будем использовать обученную Q-таблицу, однако для предотвращения выбора одних и тех же действий теперь функция награды будет обратно пропорциональной количеству взаимодействий (2). Результаты этого подхода представлены в Таблице 3.

Теперь попробуем добавить в эту стратегию элемент случайности. Объединим стра-

тегию обучения в несколько эпизодов с эпсилон жадной стратегией. Будем использовать описанную ранее ε-жадную стратегию, однако в рамках одного эпизода значение ε уменьшаться не будет. Уменьшение значения ε будет происходить после эпизода. Таким образом, первый эпизод тестирования будет почти полностью случайным, а последний – полностью подчиняться Q-learning стратегии. В итоге последний этап тестирования будет использовать обученную ε-жадной стратегией Q-таблицу. Результаты этого подхода представлены в Таблице 3.

Результаты последних трех подходов либо превосходят все рассмотренные ранее стратегии, либо приближаются к лучшим результатам на них. Таким образом, предобучение модели и добавление в нее элемента случайности значимо повысило эффективность тестирования. Интересно заметить, что в обоих случайных подходах метрика Активностей выше, чем в обычной предобученной модели. Из этого следует, что случайность на первых этапах тестирования положительно сказывается на увеличении метрики Активностей.

4.9 Анализ результатов

Перед тем как сравнивать подходы обучения с подкреплением с другими инструментами тестирования мобильных приложений, проведем анализ полученных результатов и выберем подход, который производит наиболее эффективное тестирование.

Для начала стоит отметить, что стратегии, которые способны тестировать произвольное приложение, уступают тем, которые обучаются под единственное приложение. Этого следовало ожидать, так как реализация этих алгоритмов связана с рядом упрощений для совместимости с любым приложением.

Улучшение зависимых от приложения стратегий за счет добавления классических приемов машинного обучения приводит к увеличению метрик качества. Так три последние стратегии показали высокую производительность тестирования и превзошли почти все предыдущие стратегии. Можно заметить, что добавление случайности в алгоритм способствует повышению метрики Активности. Однако смешивание двух стратегий (предобучение и случайность) не позволило взять лучшее от каждого из подходов, и результат в среднем оказался ниже. Исключение составили три приложения: Wall Street Journal, ColorNote и Wikipedia. Однако их превосходство над стратегией с предобучением незначительное.

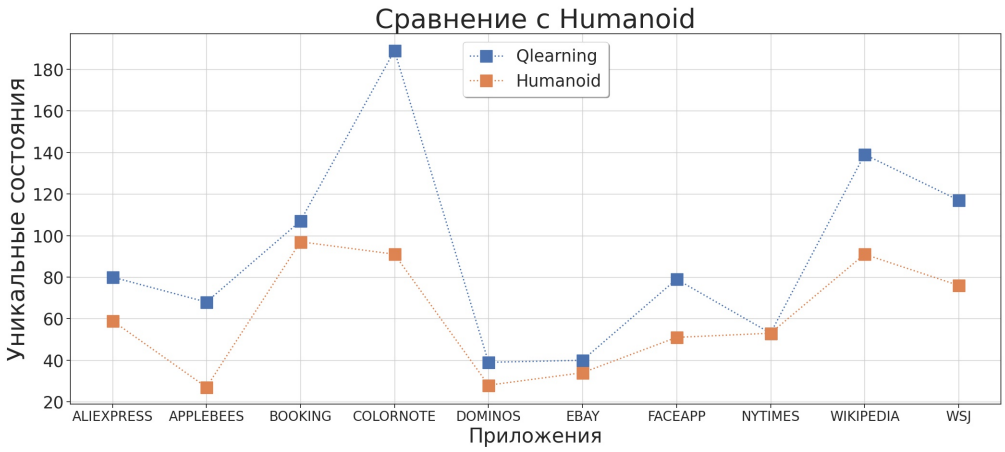
Приложения	Обратная частота нажатий		Обратное кол-во инт. элементов		Предобученная модель		Предобученная модель с эпсилон жадной стратегией	
ALIEXPRESS	37	11	34	10	**80**	16	63	**18**
APPLEBEES	32	7	34	**8**	**68**	**8**	63	**8**
BOOKING	**116**	**18**	85	16	107	15	97	**18**
COLORNOTE	142	5	183	5	189	5	**190**	**6**
DOMINOS	35	**1**	30	**1**	**39**	**1**	36	**1**
EBAY	**67**	**11**	40	**11**	40	9	34	**11**
FACEAPP	54	**2**	48	**2**	**79**	**2**	59	**2**
NYTIMES	**77**	11	49	**12**	53	10	60	11
WIKIPEDIA	74	9	95	**14**	139	**14**	**154**	12
WSJ	96	7	85	7	117	8	**134**	**9**

Таблица 4: Сравнение лучших стратегий тестирования. Метрика слева – уникальные состояния, справа – Активности.

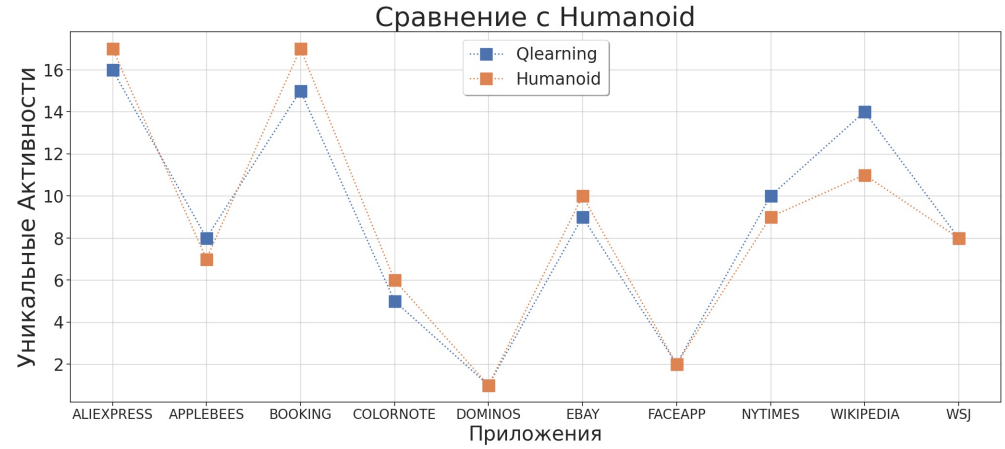
В результате анализа двух метрик качества на десяти приложениях можно выделить четыре лидирующих подхода: подход с наградой за обратную частоту нажатий (пункт 4.6.1), подход с наградой за обратное количество интерактивных элементов (пункт 4.6.2), подход с предобучением Q-таблицы (пункт 4.8), подход с предобучением и случайностью (пункт 4.8). Результаты экспериментов этих стратегий можно увидеть в таблице 4. Сравнивая их метрики качества, можно заметить, что подход с предобучением Q-таблицы ведет себя наиболее стабильно. Даже если его результаты ниже конкурентов, они все равно довольно близки к лидирующим и высоки в сравнении с другими алгоритмами. Таким образом, лучшим из разработанных в данной работе методов является метод с предобучением таблицы. Именно он будет сравниваться с другим инструментом тестирования в следующем пункте.

4.10 Сравнение с Humanoid

В предыдущем пункте удалось выделить лидирующий Q-learning подход. Будем использовать его для сравнения подходов обучения с подкреплением с современным подходом к тестированию, основанном на глубоких нейронных сетях Humanoid [8]. Humanoid обучается таким образом, чтобы максимально соответствовать человеческому поведению.



(a) Метрика: состояния.



(b) Метрика: Активности.

Рис. 8: Сравнение Q-learnign подхода с Humanoid подходом.

Эксперименты с инструментом Humanoid будем проводить при тех же условиях, что использовались для других экспериментов (5 запусков по 12 минут). Результаты сравнения можно видеть на Рис. 8. Подход обучения с подкреплением с предобучени-

ем Q-таблицы превзошел инструмент тестирования Humanoid по метрике уникальных состояний. Если сравнивать два подхода по метрике уникальных Активностей, то ни один из подходов значимо не превосходит другой.

5 Описание практической части

Для реализации алгоритмов в работе использовался язык программирования Python. Основной причиной в пользу выбора этого языка был тот факт, что тестирующая система DroidBot написана на Python, и для удобного внедрения было принято решение все алгоритмы писать на этом языке. Также Python удобен тем, что предоставляет обширную стандартную библиотеку и множество дополнительных модулей. Это позволяет вести разработку с большой скоростью, без реализации классических алгоритмов.

Реализация большинства подходов ограничивалась стандартной библиотекой Python. Однако, для некоторых алгоритмов приходилось использовать библиотеки для обработки и визуализации данных: numpy, pandas, matplotlib. Так же для подсчета расстояния редактирования графов была подключена библиотека zss, а для реализации сверточной нейронной сети использовалась библиотека keras.

Все алгоритмы тестирования встраивались в тестирующую систему DroidBot [13] (Рис. 2). Тестирующая система использовалась для того, чтобы основное внимание при решении задачи уделялось реализации алгоритмов тестирования, а не проблемам взаимодействия с устройством. DroidBot состоит из нескольких основных компонентов: `Device`, `DeviceState`, `Environment`, `InputEvent`, `InputPolicy`. Все они являются классами языка Python. Класс `Device` описывает взаимодействия которые происходят с устройством: установка/удаление приложения, получение текущего состояния, получение метаинформации о состоянии, воспроизведение нажатий. Как отмечалось ранее, эти действия происходят через низкоуровневый инструмент Android Debug Bridge. Класс `DeviceState` отвечает за представление состояния на устройстве, через него также можно получить список интерактивных элементов и возможные взаимодействия с ними. В классе `Environment` происходит основное взаимодействие тестирующего алгоритма и устройства. В нем задаются настройки тестирования и сохраняются результаты. `InputEvent` представляет из себя базовый класс для представления произвольного взаимодействия с устройством. Все жесты наследуются от этого класса.

Самым важным для этой работы классом в системе тестирования DroidBot является класс `InputPolicy`. Это базовый класс всех стратегий тестирования. Для генерации нажатия вызывается метод этого класса `generate_event`, который получает на вход обработанное состояние приложения и возвращает действие, которое нужно воспроизвести на устройстве. Это и есть основный цикл, который проходят алгоритмы тестирования.

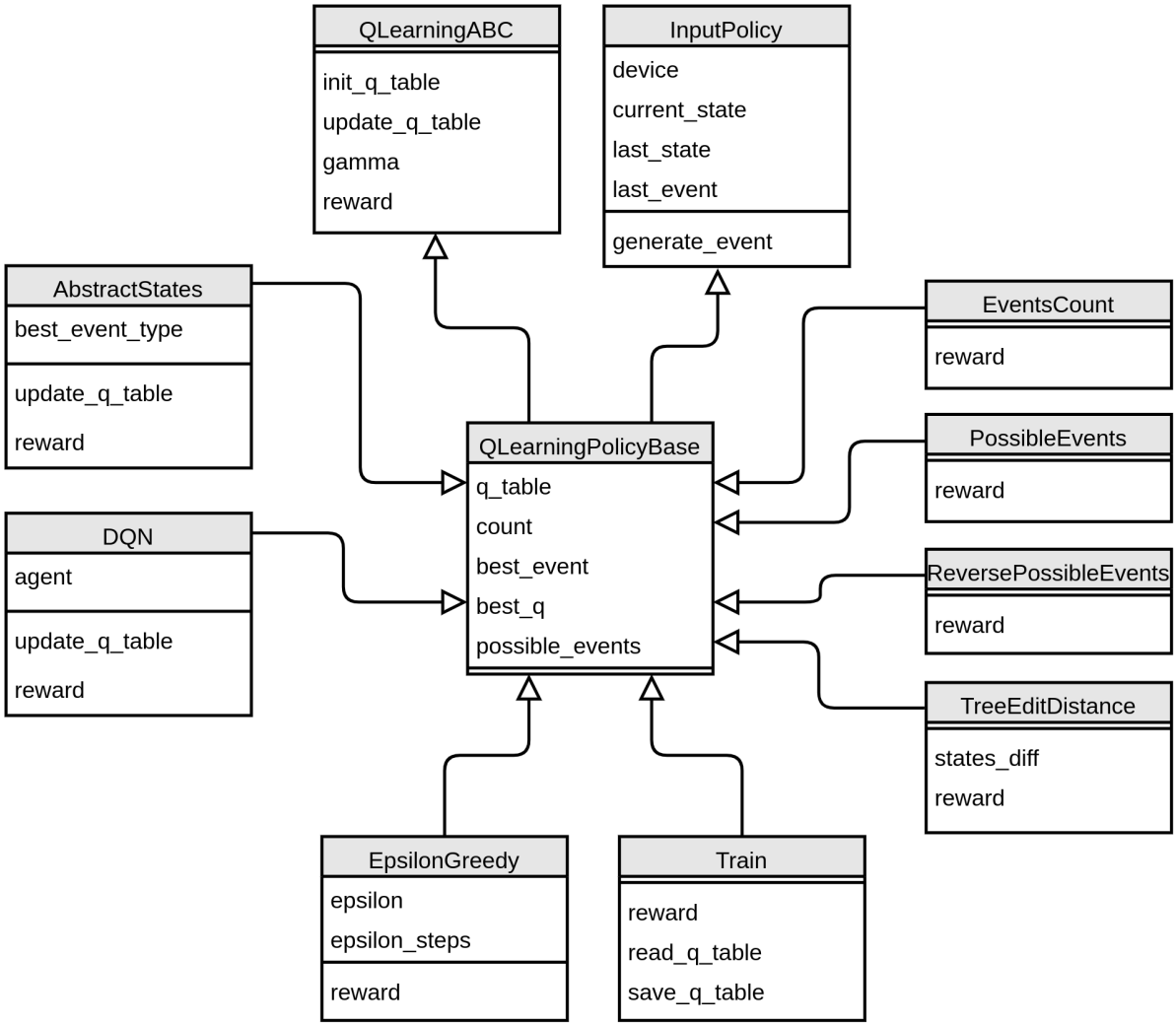


Рис. 9: UML диаграмма классов стратегий тестирования.

Алгоритмы обучения с подкреплением реализовывались отдельными классами. Их UML диаграмма представлена на Рис. 9. На этой диаграмме можно увидеть два базовых класса: `QLearningABC` и `InputPolicy`. Первый содержит абстрактные методы, которые должны быть перегружены в любой Q-learning стратегии, второй – весь необходимый функционал для встраивания стратегии в DroidBot. Класс `QLearningPolicyBase` наследует эти два класса и описывает базовую функциональность произвольного Q-learning алгоритма: создание таблицы, подсчет числа взаимодействий, алгоритмы поиска максимального Q-значения и т. д. Все остальные классы на UML диаграмме являются реализацией определенной стратегии из предыдущей главы и наследуются от `QLearningBase`.

Все алгоритмы высокоуровнево можно описать одной и той же последовательностью действий:

1. Ожидание запроса для генерации действия (`generate_event`)
2. Получение запроса вместе с текущим состоянием (`generate_event_based_on_utg`)
3. Получение списка возможных взаимодействия в текущем состоянии (`possible_events`)
4. Поиск лучшего действия согласно Q-таблице (`update_best`)
5. Обновление Q-значения для предыдущего состояния и действия (`update_q_table`)
6. Возврат лучшего действия для воспроизведения на устройстве (`best_event`)

6 Заключение

В данной работе были исследованы современные методы динамического тестирования мобильных приложений через взаимодействие с графическим интерфейсом. Основное внимание было уделено изучению перспективных подходов, основанных на алгоритмах обучения с подкреплением.

В ходе работы были изучены основы функционирования алгоритмов обучения с подкреплением и популярный подход Q-learning. Были реализованы и протестированы существующие подходы для решения поставленной задачи. Также проводились эксперименты с изменением и улучшением существующих подходов, путем добавления некоторых эвристик, в том числе основанные на сверточных нейронных сетях. Были реализованы подходы способные тестировать целый набор приложений, используя одну модель обучения с подкреплением для всех приложений. Для повышения качества тестирования были реализованы алгоритмы, способные тестировать единственное приложение, под которое они обучаются. Также проводились эксперименты по внедрению идей из классического машинного обучения.

Результаты показывают, что подходы, предназначенные для тестирования одного приложения, превосходят обобщенные модели, способные тестировать набор приложений. Также добавление логичных эвристик даже в самые простые реализации способно повысить производительность тестирования. Можно сделать вывод, что не все современные подходы способны удовлетворять требованиям практического применения алгоритмов. В результате экспериментов удалось выделить алгоритм, основанный на предварительном обучении. Он значимо превосходит эвристические предположения.

В результате сравнения с современным подходом к тестированию Humanoid, лучший из методов обучения с подкреплением оказался существенно эффективней в рамках поставленной задачи.

Результаты этой работы будут использованы в промышленном продукте, разрабатываемом в Институте системного программирования им. В.П. Иванникова РАН. В перспективе можно рассмотреть влияние альтернативного представления состояний приложения. В том числе представление на основе графовых вложений.

Список литературы

[1] *Clement, J.* Number of apps available in leading app stores 2020. — 2020.

[2] *Packard, Hewlett.* Failing to meet mobile app user expectations: a mobile user survey / Hewlett Packard // *Tech. rep.* — 2015.

[3] Understanding the test automation culture of app developers / Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan et al. // 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST) / IEEE. — 2015. — Pp. 1–10.

[4] *Arnatovich, Yauhen Leanidavich.* A systematic literature review of automated techniques for functional gui testing of mobile applications / Yauhen Leanidavich Arnatovich, Lipo Wang // *arXiv preprint arXiv:1812.11470.* — 2018.

[5] *Haoyin, LV.* Automatic android application GUI testing—A random walk approach / LV Haoyin // 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET) / IEEE. — 2017. — Pp. 72–76.

[6] *Machiry, Aravind.* Dynodroid: An input generation system for android apps / Aravind Machiry, Rohan Tahiliani, Mayur Naik // Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. — 2013. — Pp. 224–234.

[7] *Studio, Android.* Ui/application exerciser monkey / Android Studio // *Retrieved February.* — 2017. — Vol. 10. — P. 2020.

[8] A deep learning based approach to automated android app testing / Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen // *arXiv e-prints.* — 2019. — Pp. arXiv–1901.

[9] *Azim, Tanzirul.* Targeted and depth-first exploration for systematic testing of android apps / Tanzirul Azim, Iulian Neamtiu // Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications. — 2013. — Pp. 641–660.

[10] Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps / Shuai Hao, Bin Liu, Suman Nath et al. // Proceedings of the 12th annual international conference on Mobile systems, applications, and services. — 2014. — Pp. 204–217.

[11] *Mao, Ke.* Sapienz: Multi-objective automated testing for android applications / Ke Mao, Mark Harman, Yue Jia // Proceedings of the 25th International Symposium on Software Testing and Analysis. — 2016. — Pp. 94–105.

[12] *Choudhary, Shauvik Roy.* Automated test input generation for android: Are we there yet?(e) / Shauvik Roy Choudhary, Alessandra Gorla, Alessandro Orso // 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) / IEEE. — 2015. — Pp. 429–440.

[13] Droidbot: a lightweight ui-guided test input generator for android / Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen // 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C) / IEEE. — 2017. — Pp. 23–26.

[14] *Memon, Atif M.* Coverage criteria for GUI testing / Atif M Memon, Mary Lou Soffa, Martha E Pollack // Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering. — 2001. — Pp. 256–267.

[15] *Szepesvári, Csaba.* Algorithms for reinforcement learning / Csaba Szepesvári // *Synthesis lectures on artificial intelligence and machine learning.* — 2010. — Vol. 4, no. 1. — Pp. 1–103.

[16] An empirical study of android test generation tools in industrial cases / Wenyu Wang, Dengfeng Li, Wei Yang et al. // 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE) / IEEE. — 2018. — Pp. 738–748.

[17] DRIFT: Deep Reinforcement Learning for Functional Software Testing / Luke Harries, Rebekah Storan Clarke, Timothy Chapman et al. // *arXiv preprint arXiv:2007.08220.* — 2020.

[18] QBE: QLearning-based exploration of android applications / Yavuz Koroglu, Alper Sen, Ozlem Muslu et al. // 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) / IEEE. — 2018. — Pp. 105–115.

[19] Reinforcement learning for android gui testing / David Adamo, Md Khorrom Khan, Sreedevi Koppula, Renée Bryce // Proceedings of the 9th ACM SIGSOFT International

Workshop on Automating TEST Case Design, Selection, and Evaluation. — 2018. — Pp. 2–8.

[20] *Vuong, Thi Anh Tuyet.* A reinforcement learning based approach to automated testing of android applications / Thi Anh Tuyet Vuong, Shingo Takada // Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation. — 2018. — Pp. 31–37.