DOI: 10.15514/ISPRAS-2025-37(2)-7

Алгоритм выбора семантических мутаций в фаззинге по принципу серого ящика

[1] *Г.Р. Райкин, ORCID: 0009-0004-7783-7818 <gregra@mail.ru>*
[2] *М.С. Пелевин, ORCID: 0009-0007-5762-2918 <maks.pelevin@gmail.com>*
[1] *В.М. Ицыксон, ORCID: 0000-0003-0276-4517 <itsykson@yandex.ru>*

[1] *Университет ИТМО,*
Россия, 197101, Санкт-Петербург, Кронверкский проспект, д.49, л. А.

[2] *СПбГЭТУ «ЛЭТИ»,*
Россия, 197022, Санкт-Петербург, ул. Профессора Попова, д. 5, л. Ф.

Аннотация. С развитием современных информационных систем динамический анализ становится неотъемлемой частью процесса разработки программного обеспечения. Одной из самых эффективных и распространённых техник в этой области является фаззинг-тестирование (фаззинг). Суть этого метода состоит в передаче исследуемой программе большого количества случайных и неожиданных входных данных. Инструменты мутационного фаззинга генерируют тестовые данные, применяя модификации (мутации) к удачным из уже использованных вариантов, повышая таким образом число обнаруженных поведений и покрытие кода. Сами мутации при этом чаще всего выбираются случайным образом. В данной работе предложен метод повышения эффективности мутационного фаззинга с помощью адаптивной стратегии выбора мутации. Предложенный метод апробирован на широко используемых Java-пакетах и продемонстрировал статистически значимый прирост количества найденных ошибок и числа различных поведений (трасс исполнения) тестируемых программ.

Ключевые слова: динамический анализ; тестирование; фаззинг; структурный фаззинг; эволюционный фаззинг; задача о многоруком бандите.

Для цитирования: Райкин Г.Р., Пелевин М.С., Ицыксон В.М. Алгоритм выбора семантических мутаций в фаззинге по принципу серого ящика. Труды ИСП РАН, том 37, вып. 2, 2025 г., стр. 97–114. DOI: 10.15514/ISPRAS–2025–37(2)–7.

Semantic Mutation Strategy in Grey-Box Fuzzing

[1] *G.R. Raykin, ORCID: 0009-0004-7783-7818 <gregra@mail.ru>*
[2] *M.S. Pelevin, ORCID: 0009-0007-5762-2918 <maks.pelevin@gmail.com>*
[1] *V.M. Itsykson, ORCID: 0000-0003-0276-4517 <itsykson@yandex.ru>*

[1] *ITMO University,*
Kronverksky Pr. 49, bldg. A, St. Petersburg, 197101, Russia.

[2] *Saint Petersburg Electrotechnical University LETI,*
Professora Popova St., 5, St. Petersburg, 197022, Russia.

Abstract. With the advancement of modern information technology, dynamic analysis is becoming an essential part of software development. Fuzz testing is one of the most efficient and widely used techniques in this field. The core idea behind this approach is to input a large amount of random data into the program under the test. Mutation-based fuzzing tools generate test data by applying modifications (mutations) to successful variants that have already been identified, thus increasing the number of detected behaviors and code coverage. A common mutation strategy is to randomly select a mutation operator with a predefined probability. This paper proposes a method to improve the effectiveness of mutation fuzzing through an adaptive mutation selection strategy. This approach was tested on commonly used Java packages and showed a statistically significant improvement in the number of errors detected and the diversity of program behaviors (execution traces).

Keywords: dynamic analysis; software testing; fuzzing; structural fuzzing; evolutionary fuzzing; multi-armed bandit problem.

For citation: Raykin G.R., Pelevin M.S., Itsykson V.M. Semantic mutation strategy in grey-box fuzzing. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 2, 2025. pp. 97-114 (in Russian). DOI: 10.15514/ISPRAS-2025-37(2) - 7.

1. Введение

Фаззинг (фаззинг-тестирование) – один из наиболее распространенных методов динамического анализа и тестирования ПО, в основе которого лежит передача тестируемой системе разнообразных входных данных, сгенерированных псевдослучайным образом. Такой подход позволяет проверить поведение системы в большом количестве различных ситуаций, в том числе не предусмотренных ее разработчиками.

Для обеспечения качества генерируемых фаззером входных данных, используется структурный фаззинг, позволяющий генерировать псевдослучайные данные с использованием информации о структурах данных и форматах представления данных. Например, при генерации может учитываться грамматика некоторого языка [1], формат файла [2], спецификация протокола передачи данных [3] и т.д. Однако, зачастую одной корректности входных данных недостаточно для задействования существенной части тестируемой логики, и для проведения объемлющего тестирования необходимо использовать свойства тестируемой программы, которые не отражены в модели [4]. Обнаружению и эксплуатации таких свойств посвящено множество исследований и разработок [5].

В зависимости от способа взаимодействия с тестируемой системой, техники фаззинга условно делят на фаззинг по методу черного, белого и серого ящика. Данная работа фокусируется на фаззинге по методу серого ящика, который предполагает запуск инструментированного кода тестируемой программы и использование информации о трассах исполнения.

Основой всех алгоритмов фаззинга по методу серого ящика является мутационный цикл обратной связи [6]. Эта схема предполагает сохранение наиболее результативных (например, с точки зрения покрытия кода или частотности обнаружения соответствующего поведения) входных данных, и последующую генерацию на основе успешных предыдущих попыток

посредством их незначительного изменения (мутации). Качество и результативность генерируемых фаззером входных данных при этом определяются эффективностью мутаций, проводимых в цикле обратной связи.

При структурном фаззинге применяются мутации, которые меняют исходное значение таким образом, чтобы результат не нарушал заданную моделью спецификацию [7]. Большинство инструментов фаззинга применяют мутации в случайном порядке, таким образом цикл обратной связи культивирует результативные входные значения, а процесс мутаций остается неизменным на всем протяжении тестирования.

В данной работе предложен метод повышения эффективности мутационного фаззинга по методу серого ящика с помощью адаптивной стратегии мутаций. Суть метода состоит в том, чтобы ввести в алгоритм мутационного фаззинга второй цикл обратной связи, который позволит оценивать эффективность мутаций и взвешивать вероятности их применения на основе этой информации. Предложенный метод был реализован на базе эволюционного фаззера и апробирован на широко используемых Java-пакетах. Апробация показала статистически значимый прирост количества найденных ошибок и увеличение числа поведений тестируемых программ в сравнении со случайными мутациями при тестировании с одним и тем же количеством запусков.

2. Алгоритм эволюционного фаззинга

Используемый в данной работе фаззер реализует модифицированный эволюционный алгоритм (μ+1) [8]. Данный алгоритм поддерживает популяцию μ особей (в данном случае в качестве особей выступают входные значения тестируемой программы), каждой из которых ставится в соответствие показатель приспособленности (в контексте задачи фаззинга в качестве приспособленности используется некоторая оценка потенциала значения для дальнейшей эволюции). На каждом шаге алгоритма из популяции выбирается один представитель с вероятностью, пропорциональной показателю приспособленности. К выбранному значению применяется мутация, получившаяся в результате новая особь получает оценку приспособленности, и если этот показатель превосходит минимальное значение из популяции, он добавляется в популяцию на место наименее приспособленного представителя (рис. 1).

В качестве точки входа фаззер использует методы классов в Java проекте. Особенностью такого подхода является необходимость генерации экземпляров объектов произвольных типов в соответствии с сигнатурой тестируемого метода и структурная мутация этих данных в соответствии с их типами. Для этого в фаззере используется рекурсивное представление данных, которое позволяет описать объекты произвольной структуры и мутации, соответствующие их семантике.

Представление значений в фаззере устроено следующим образом. Фаззер содержит внутреннее представление и специальные наборы мутаций для строк, битовых векторов, чисел с плавающей точкой и байтовых буферов. Данные разных типов могут объединяться в составные объекты, а наборы данных одного типа – в коллекции, для коллекций и составных объектов также предусмотрены мутации, меняющие порядок их содержимого или структуру. Мутация данных происходит рекурсивно: сначала мутационный оператор применяются к структуре объекта или коллекции, затем к их содержимому и так далее. Таким образом, фаззер не только генерирует данные в рамках автоматически выведенной спецификации, но и направляет процесс эволюции значений, основываясь на характере их использования.

3. Задача выбора мутаций

Анализ эффективности мутационных операторов на примере фаззера AFL показал, что результативность мутаций с точки зрения генерации входных данных, раскрывающих новые поведения, различна, и их соотношение отличается в зависимости от тестируемой

программы [9]. Эмпирический анализ используемого в данной работе фаззера также показывает, что соотношение эффективностей мутаций может меняться с течением времени. Исходя из этих наблюдений, можно сделать вывод о неэффективности мутационных стратегий с предопределенным соотношением вероятностей выбора.

```
MU_PLUS_ONE_FUZZING(mutations, μ, t_limit):
1   popultion ← init_population(μ)
2   while t_elapsed < t_limit :
3   |  seed ← sample(population)
4   |  mutation ← sample(mutations)
5   |  mutated_seed ← mutate(seed, mutation)
6   |  fitness ← execute(mutated_seed)
7   |  if fitness > population.worst.fitness:
8   |  |  if population.size = μ:
9   |  |  |  population.remove_worst_seed()
10  |  |  population.add(seed, fitness)
```

Рис. 1. Упрощенный алгоритм эволюционного фаззинга (μ+1).
Fig. 1. Simplified (μ+1) evolutionary fuzzing algorithm.

Ниже приведен обзор существующих подходов к выбору мутаций в контексте требований, предъявленных разрабатываемому алгоритму ввиду особенностей использования фаззера:

- Предполагается, что процесс выбора мутаций происходит независимо от выбора мутируемых данных. Это требование обусловлено тем, что алгоритм выбора данных для мутации является предметом многочисленных исследований, и добавление стратегии выбора мутаций не должно ограничивать его дальнейшую модификацию.
- Алгоритм выбора мутаций не должен требовать какого-либо предобучения. Идея использования предобученных на большом количестве программ алгоритмов обсуждается в литературе, однако, применимость и переносимость таких решений очень сложно проверить, поэтому такой подход не получил распространения.
- Выбор мутаций не должен требовать значительных вычислительных ресурсов. Эффективность модификации алгоритма фаззинга должна компенсировать сокращение скорости работы (и, соответственно, количества запусков), к которому она приводит. Проверить такое свойство чрезвычайно сложно ввиду того, что время запуска может существенно варьироваться в зависимости от тестируемой системы и условий эксперимента.

4. Обзор существующих алгоритмов выбора мутаций

4.1 Метод тепловых карт

Метод тепловых карт или «горячих байтов» состоит в том, чтобы найти в кодирующей входное значение последовательности байт таких позиции, что изменение значений в них с большей вероятностью приводит к раскрытию новых поведений и задействованию большего объема тестируемой логики [10,11]. Фокусирование мутаций на «горячих» позициях позволяет повысить эффективность мутаций и сократить количество ненужных запусков тестируемой программы.

Одним из главных ограничений этого метода является необходимость хранить и анализировать информацию о «температуре» наборов данных, особенно если фаззер работает с входными данных большого объема. Кроме того, техника тепловых карт требует, чтобы мутации были параметризованы конкретной позицией, к которой они применяются, что ограничивает использование этого подхода для структурированных значений, а также осложняет задание пользовательских мутаций. Для некоторых семантических мутаций, используемых в данной работе, этот подход не применим.

4.2 Оптимизация роем частиц

Метод оптимизации роем частиц (Particle Swarm Optimization – PSO) – это метаэвристический алгоритм глобальной оптимизации, вдохновленный социальным поведением животных. Он заключается в управлении популяцией частиц, перемещающихся в пространстве возможных решений задачи, итеративно приближаясь таким образом к оптимальному решению.

В рамках алгоритма MOpt [9], реализованного в одном из наиболее широко используемых инструментов фаззинга AFL++, для поиска эффективного соотношения вероятностей выбора мутаций используется модифицированный PSO. В роли частиц в нем выступают мутации, а в качестве пространства решений – вероятности их выбора.

Другой инструмент, применяющий алгоритм оптимизации роем частиц – PSOFuzz [12] использует в качестве частицы однопоточный фаззер, а в качестве пространства решений – соотношения вероятностей выбора мутаций. Таким образом, в качестве роя частиц выступает фаззер, работающий в многопоточном режиме, а искомым оптимумом – набор одновременно применяемых стратегий мутации.

Недостатком этого метода является то, что он требует большого количества обновлений и, соответственно, времени работы для обнаружения приемлемого решения. Также алгоритму PSO свойственна высокая вычислительная сложность, которая влияет на скорость работы фаззера, ввиду чего анализ его производительности показывает противоречивые результаты [13].

4.3 Марковский процесс принятия решения

Наличие цикла обратной связи позволяет формализовать фаззинг по принципу серого ящика как марковский процесс принятия решений. Эта модель описывает задачу оптимизации как задачу поиска оптимальной стратегии принятия решений агентом, находящимся в некоторой среде, которая меняется в зависимости от предпринимаемых им действий. Взаимодействие агента и среды заключается в том, что агент выбирает одно из доступных действий, после чего среда с некоторой вероятностью меняет свое состояние, и в результате изменения состояния агент получает вознаграждение. Вознаграждение определяется определяющей среду функцией награды.

В контексте выбора мутаций в фаззинге в качестве агента выступает планировщик мутаций, который на каждом шаге выбирает одну из доступных мутаций или планировщик

мутируемых значений, который выбирает очередное значение из популяции, а в качестве среды – набор поведений или трасс исполнения тестируемой программы. Таким образом, изменение состояния среды происходит в тот момент, когда очередная мутация приводит появлению входного значения, которое раскрывает неизвестное ранее поведение.

Этот формализм позволяет применить к выбору мутаций ряд мощных методов оптимизации, в частности Q-обучение и нейронные сети [14-16]. Однако, на текущий момент, алгоритмы фаззинга, основанные на глубоком обучении показывают недостаточно впечатляющие результаты и нуждаются в дополнительных исследованиях [17].

4.3 Задача о многоруком бандите

Еще одну модель, которая хорошо подходит для описания процесса фаззинга и, в частности, выбора мутаций, рассматривает задача о многоруком бандите [18]. Аналогично марковскому процессу принятия решений, эта модель описывает агента, которому доступно несколько действий (их также называют руками или рычагами). Каждое действие с некоторой вероятностью генерирует вознаграждение. Цель агента – максимизировать выигрыш, полученный за ограниченное число действий.

Задача о многоруком бандите демонстрирует дилемму исследования и эксплуатации (exploration-exploitation problem): чтобы максимизировать выигрыш, агент должен одновременно изучить распределения вероятностей выигрышей, соответствующих разным действиям, и использовать накопленные знания, выбирая наиболее перспективные действия.

В различных предметных областях применяется множество вариаций и обобщений задачи однорукого бандита. Нестационарная задача о многоруком бандите – обобщение, описывающее агента в динамической среде, свойства которой меняются со временем. Также существует вариант задачи, при котором на среду оказывает влияние некоторая дополнительная информация – контекст. В этом случае, агент должен не только изучить фиксированные свойства среды, основываясь на своих предыдущих действиях, но и выявить закономерности между вариантами контекста и состояниями среды. Такой вариант задачи называют задачей о контекстном многоруком бандите. Наконец, можно рассмотреть многопользовательскую задачу о многоруких бандитах, в которой несколько агентов существуют в одной среде, а изменение среды определяется совокупностью их решений. В зависимости от задач, которые решают агенты, выделяют конкурентную и коллаборативную задачу о многоруких бандитах.

Модель задачи о многоруком бандите можно применить для описания процесса фаззинга различными способами. Наиболее популярный из них – задача выбора значений из корпуса данных [19-21]. В качестве «рук» при этом используются мутируемые значения, а награда вычисляется по результатам исполнения тестируемой программы.

Алгоритм MobFuzz [22] рассматривает выбор значений и выбор мутации как две независимые задачи, описывая таким образом процесс фаззинга как нестационарную многопользовательскую задачу о многоруких бандитах. Аналогичный подход используется в SeamFuzz [23]: этот алгоритм разбивает входные значения на группы и решает задачу выбора мутаций для каждой из групп независимо.

Алгоритмы решения вариантов задачи о многоруком бандите получили широкое распространение в фаззинге благодаря своей экономичности, быстрой сходимости и способности адаптироваться к изменяющимся условиям среды. В данной работе был выбран именно этот подход, так как он соответствует всем описанным выше требованиям: позволяет решать задачу выбора мутаций независимо от выбора мутируемых данных, а также не требует предобучения и существенных вычислительных ресурсов.

5. Выбор мутаций как задача о многоруком бандите

Рассмотрим задачу выбора мутаций в эволюционном фаззинге по методу серого ящика. Агентом в этой модели выступит планировщик, а действиями – выбираемые мутации. Среда задается тестируемой системой и набором ее поведений. Цель агента – выбирать мутации таким образом, чтобы за ограниченное число запусков обнаружить в тестируемой программе как можно больше ошибок и уникальных поведений. Ошибками в данном случае считаются необработанные исключения, они индексируются названием исключения и вершиной стека вызовов. Это некоторое допущение, так как в общем случае разные ошибки могут приводить к одному и тому же исключению, и наоборот, одна и та же ошибка может проявляться разным образом, тем не менее, такой подход позволяет однозначно идентифицировать позиции в коде, в которых возникают ошибки. Различные поведения индексируются трассами исполнения, для оптимизации расхода памяти каждой трассе присваивается хеш-код. Таким образом, выбор мутации влечет один из четырех исходов: обнаружение новой ошибки, обнаружение нового поведения, повторение уже известной ошибки и повторение уже известного корректного поведения.

В данной работе задача выбора мутаций решается независимо от выбора значений (вероятность выбора удачного значения для мутации рассматривается как вероятностное свойство среды), в рамках дальнейшей работы модель может быть приведена к коллаборативной задаче о многоруких бандитах с помощью добавления второго агента, выбирающего исходные значения.

5.1 Свойства процесса фаззинга

Алгоритмы решения вариантов задачи о многоруком бандите опираются на известные свойства среды, с которой взаимодействует агент. Следовательно, и для решения задачи выбора мутаций важно определить характеристики, которыми может руководствоваться планировщик мутаций. Были выделены следующие особенности:

- Распределение вероятностей «успехов» мутаций определяется тестируемой программой, поэтому не стоит закладывать в решение предположения о конкретных распределениях.
- Так как обнаружение нового поведения раскрывает новую тестируемую логику, свойства которой могут отличаться от протестированной до этого, можно сказать, что распределение вероятностей «успехов» мутаций меняется скачкообразно по мере раскрытия новых поведений.
- Количество запусков тестируемой программы, необходимое для обнаружения следующей ошибки или следующего нового поведения, растет экспоненциально [24], при этом количество запусков, воспроизводящих известные ошибки, растет линейно (рис. 2-4).
- Запуски, воспроизводящие ошибки, требуют больше времени и ресурсов, чем те, которые завершаются корректно, поэтому предпочтительнее выполнять больше корректно завершающихся исполнений.

5.2 Жадный алгоритм решения задачи о многоруком бандите

Эпсилон-жадный (ε-greedy) алгоритм решения задачи о многоруком бандите реализует самый простой способ решения проблемы исследования и эксплуатации: большую часть времени жадно выбирает действие, которое приносит наибольшую награду, но с некоторой вероятностью принимает «исследовательские» решения, выбирая случайное действие.

Пусть A – множество возможных действий, $R_t(a)$ – рейтинг действия a на момент времени t, основанный на количестве успехов. На каждом шаге алгоритм разыгрывает случайное $\xi_t \in$ из нормального распределения, и в зависимости от его значения выбирает действие.

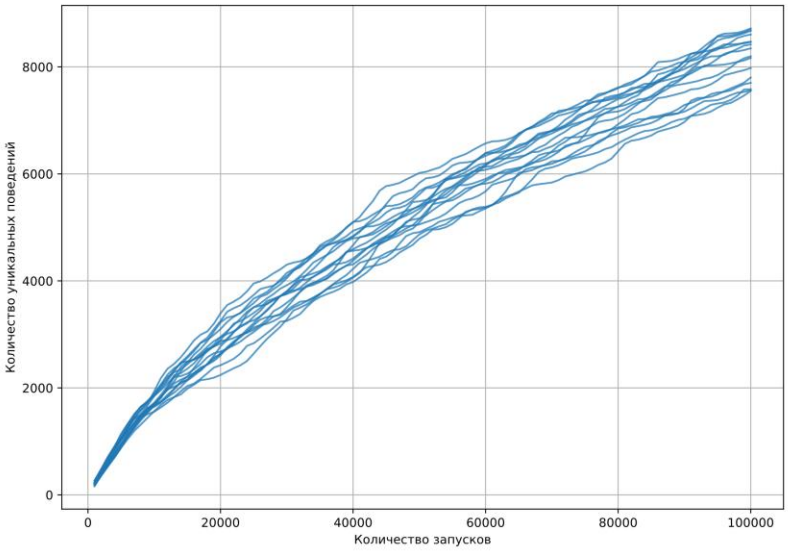

Рис. 2. Зависимость количества обнаруженных уникальных поведений от числа запусков на примере реальной программы (результаты нескольких прогонов).
Fig. 2. Dependence of the number of detected unique behaviors on the number of runs while testing a real program (results of several runs).

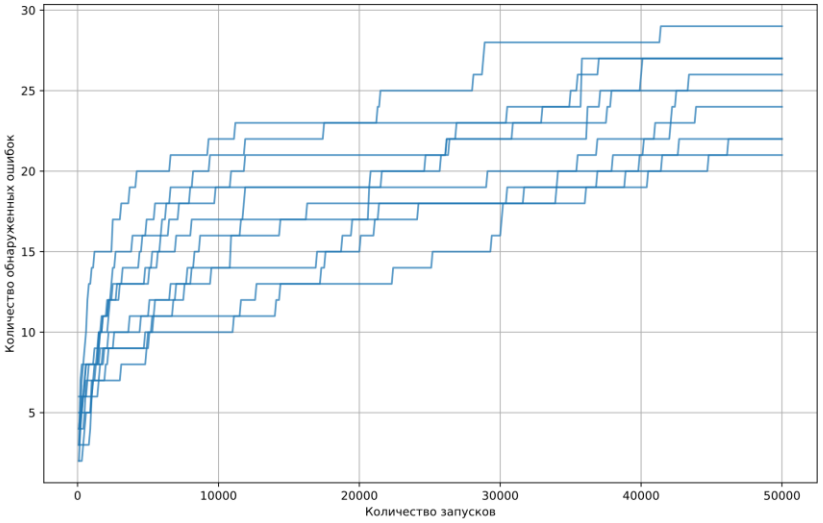

Рис. 3. Зависимость количества найденных ошибок от числа запусков на примере реальной программы.
Fig. 3. Dependence of the number of errors found on the number of runs while testing a real program

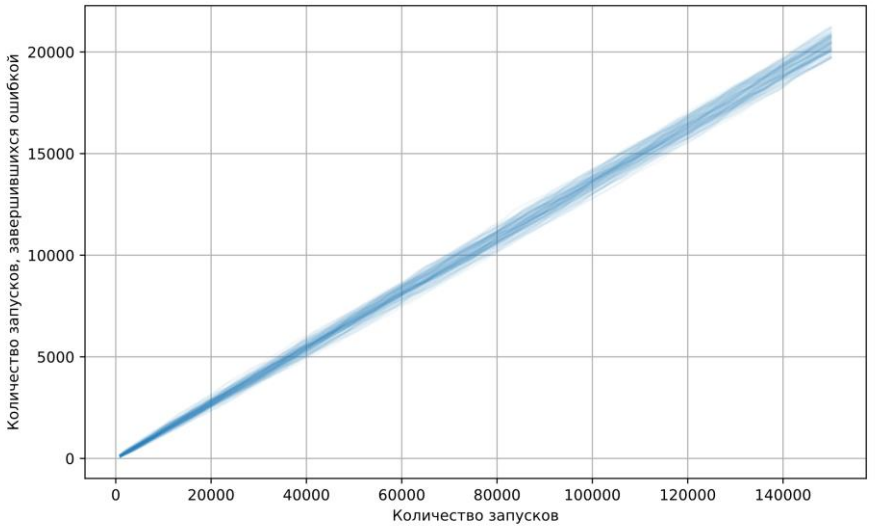

Рис. 4. Зависимость количества запусков, завершившихся ошибкой, от числа запусков на примере реальной программы.
Fig. 4. Dependence of the number of error reproducing runs on the number of runs while testing a real program.

Таким образом, параметр ε задает соотношение вероятностей исследования и эксплуатации:

$$a_t = \begin{cases} argmax_a R_t(a), \text{если} \xi_t \geq \varepsilon, \\ \underset{random}{\leftarrow} A, \text{если} \xi_t < \varepsilon, \end{cases}$$

Идею жадного алгоритма удалось применить для сокращения исполнений, завершающихся ошибкой. Для этого при вычислении «рейтинга» мутации также семплируется $\xi_t \in$, и рейтинг вычисляется как:

$$\varepsilon\_greedy_t = \begin{cases} 1.0, \text{если} \xi \geq \varepsilon, \\ \dfrac{n_t(m)}{e_t^r(m)}, \text{если} \xi < \varepsilon. \end{cases}$$

Здесь $n_t(m)$ – количество применений мутации m на момент времени t, $e_t^r(m)$ – количество применений мутации m, в результате которых была воспроизведена уже известная ошибка. Таким образом, величина $\frac{n_t(m)}{e_t^r(m)}$ обратно пропорциональна вероятности воспроизведения ошибки.

Узким местом этого подхода остается параметр ε, определяющий вероятность того, что алгоритм примет «рискованное» решение. Эмпирически было выбрано значение $\varepsilon = 0.2$ как наиболее удачное.

5.3 Семплирование Томпсона

Семплирование Томпсона – один из наиболее эффективных и широко используемых методов решения задачи о многоруком бандите. Согласно некоторым исследованиям, в алгоритмах фаззинга этот подход превосходит аналоги, такие как Exp3 и UCB (Upper Confidence Bounds) [25, 26].

Алгоритм семплирования Томпсона разыгрывает вероятности выбора действий из бета-распределения, параметры которого итеративно подбираются по мере работы.

Бета-распределение определяется двумя параметрами $\alpha, \beta > 0$, и плотность его вероятности определяется как

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

Бета-функция $B(\alpha, \beta)$ имеет вид:

$$B(\alpha, \beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1} dx.$$

Соответственно, первый и второй моменты случайной величины, принадлежащей бета-распределению, определяются как

$$E[\xi_B] = \frac{\alpha}{\alpha + \beta}$$

$$D[\xi_B] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

Таким образом, в случае многоруких бандитов Бернулли (таких, что каждое действие приводит к успеху или неудаче), если сопоставить α количество успехов, а β – количество неудач, математическое ожидание такого распределения будет стремиться к средней награде, полученной засчет этого действия, а дисперсия будет сокращаться по мере обучения. Более того, даже при изменении распределений оптимальных вероятностей, алгоритм может «переучить» распределение и адаптироваться к новым условиям.

С точки зрения мутаций в фаззинге, успехом можно считать обнаружение нового поведения тестируемой программы (как корректного, так и ошибочного), а неудачей – воспроизведение уже известного поведения. Тогда алгоритм решает задачу поиска новых поведений.

Для выбора мутаций алгоритм семплирования Томпсона был модифицирован с учетом описанных зависимостей числа поведений от числа исполнений следующим образом:

$$\alpha = v_t(m) \cdot e_t(m)$$

$$\beta = log\left(v_t^r(m) \cdot e_t^r(m)\right)$$

Здесь $v_t(m)$ – количество уникальных корректных поведений тестируемой программы, $v_t^r(m)$ – количество вызовов, повторяющих известные корректные поведения, $e_t(m)$ – количество уникальных ошибок и $e_t^r(m)$ – количество воспроизведений известных ошибок мутацией m на момент времени t.

5.4 Итоговое решение

Итоговый алгоритм объединяет описанные выше методы и стремится одновременно сократить количество воспроизведений ошибок и максимизировать количество обнаруженных поведений:

$$p_t(m) = \sqrt{\frac{e_t(m)}{M \vee t}}\left(\varepsilon\_greedy_t(m) + B(\alpha, log\beta)\right)$$

Коэффициент $\sqrt{\frac{e_t(m)}{M \vee t}}$ отвечает за скачкообразное изменение свойств среды. Он используется в алгоритме AdSwitch, который решает нестационарную задачу о многоруком бандите, детектируя «точки изменения» среды [27]. В качестве таких точек в данном случае выступают ошибки ($e_t(m)$ – количество уникальных ошибок, обнаруженных мутацией m на момент времени t, $M \vee$ – количество самих мутаций).

Таким образом, алгоритм принимает следующий вид (рис. 6): после каждого запуска тестируемой программы помимо обновления популяции, обновляется и вероятность выбора примененной мутации. Такой подход позволяет не только культивировать удачные входные данные по мере работы, но и накапливать информацию о более перспективных мутациях, адаптируясь к возможным скачкообразным изменениям их результативности.

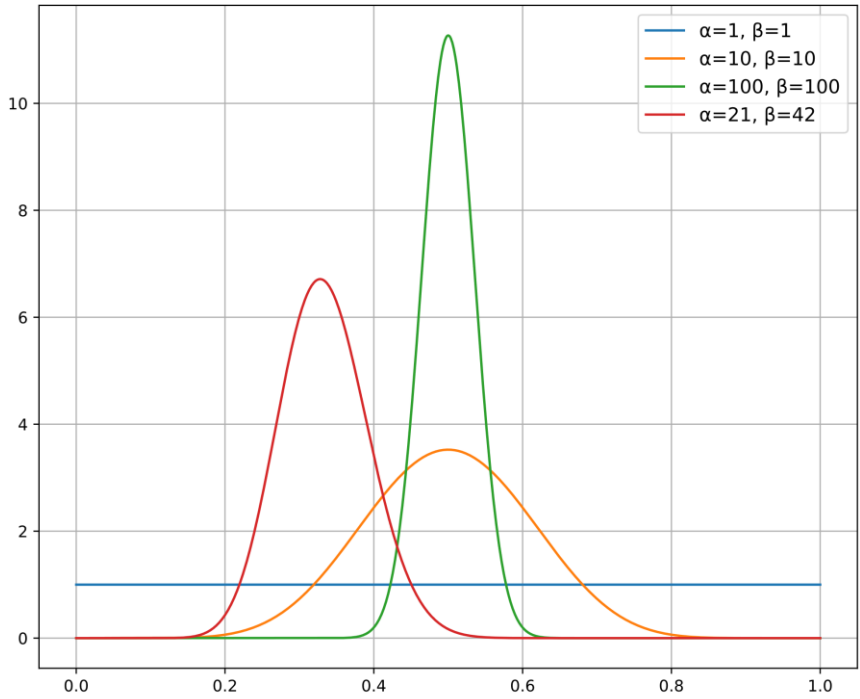

Рис. 5. Плотность вероятности бета-распределения при разных значениях параметров.
Fig. 5. Probability density function of beta distribution for different parameter values.

6. Апробация результатов

6.1 Задача оценки алгоритмов фаззинга

Оценка алгоритмов фаззинга – комплексная задача, которой посвящено множество исследований [28-30].

Прежде всего, важно учесть разнообразие программ, к которым должен быть применим алгоритм. Привлекательной идеей кажется использование для тестирования синтетических программ. Однако, тестирование синтетических программ как способ оценки эффективности алгоритмов фаззинга показало низкую эффективность по сравнению с использованием реальных программ [31].

Следующим важным фактором является продолжительность работы. При прочих равных, предпочтительным считается тот алгоритм, который быстрее достигает высоких значений целевых метрик. Однако, однозначно оценить эту «скорость» невозможно – алгоритмы могут демонстрировать различные результаты в зависимости от отведенных временных

промежутков. Например, AFL разбивает процесс тестирования на несколько последовательных этапов, каждый из которых решает разные задачи.

```
MU_PLUS_ONE_ADAPTIVE_MUTATIONS_FUZZING(mutations, μ, t_limit):
 1  popultion ← init_population(μ)
 2  for mutation in mutations :
 3  |  new_traces[mutation] ← 0
 4  |  known_traces[mutation] ← 0
 5  |  new_errors[mutation] ← 0
 6  |  known_errors[mutation] ← 0
 7  |  rate[mutation] ← 1
 8  while t_elapsed < t_limit :
 9  |  seed ← sample(population)
10  |  mutation ← sample_proportional(mutations, )
11  |  mutated_seed ← mutate(seed, mutation)
12  |  feedback ← execute(mutated_seed)
13  |  if feedback.trace is new:
14  |  |  new_traces[mutation] ← new_traces[mutation] + 1
15  |  else:
16  |  |  known_traces[mutation] ← known_traces[mutation] + 1
17  |  if feedback.error is new:
18  |  |  new_errors[mutation] ← new_errors[mutation] + 1
19  |  else:
20  |  |  known_errors[mutation] ← known_errors[mutation] + 1
21  |  rate[mutation] ← update_mutation_rate(mutation)
22  |  if fitness > population.worst.fitness:
23  |  |  if population.size = μ:
24  |  |  |  population.remove_worst_seed()
25  |  |  population.add(seed, fitness)
```

Рис. 6. Модифицированный алгоритм эволюционного фаззинга (μ+1) с адаптивным выбором мутаций.

Fig. 6. Modified (μ+1) evolutionary fuzzing algorithm with adaptive mutation scheduling.

Одной самодостаточной метрики для оценки качества фаззинга не существует, поэтому при интерпретации результатов необходимо учитывать совокупность нескольких показателей. Самым приоритетным из них считается количество обнаруженных ошибок. Некоторые фаззеры ставят целью обнаружение определенных типов ошибок и уязвимостей.

Еще одним значимым параметром является покрытие кода. Покрытие может быть описано несколькими метриками: покрытием строк, ветвлений, инструкций и т.д. Для оценки эффективности фаззинга рекомендуется использовать хотя бы две из них: покрытие строк и ветвлений. Покрытие строк явно показывает, какая доля кода была выполнена, но не отражает число поведений. Покрытие ветвлений, в свою очередь, лучше отражает число поведений, но не дает информации о том, какая часть кода была задействована. Таким

образом, только комбинация этих метрик позволяет достаточно полно оценить покрытие кода сгенерированными фаззером тестами.

Еще одной сложностью является то, что ввиду стохастической природы фаззинга, результаты тестирования могут значительно различаться от запуска к запуску. В связи с этим, оценивать результативность фаззера можно только на основе многочисленных запусков в одних и тех же условиях.

Сделать предположения о распределении значений описанных метрик не представляется возможным, поэтому для анализа результатов используются непараметрические методы: U-критерий Манна-Уитни и величина эффекта Варга-Делани [32].

Критерий Манна-Уитни – непараметрический ранговый критерий, который предназначен для сравнения двух выборок и проверки нулевой гипотезы – предположения о том, что значения метрик распределены одинаково. Если нулевая гипотеза отвергается с достаточно низким уровнем значимости p (обычно используется значение 0.05), можно сделать вывод о достоверности результатов эксперимента. Важным достоинством критерия Манна-Уитни является то, что он подходит для сравнения выборок небольшого объема. Факта опровержения нулевой гипотезы недостаточно, чтобы сделать вывод о степени выраженности различий выборок, поэтому также оценивают величину эффекта. Наиболее широко используемой при оценке фаззинга является величина эффекта Варга и Делани [33]:

$$\hat{A}_{12} = \left(\frac{R_1}{n_1} - \frac{n_1 + 1}{2}\right) / n_2$$

где R_1 и R_2 – суммы рангов значений выборок, а n_1 и n_2 – объемы выборок.

Интерпретировать значение размера эффекта можно как вероятность того, что алгоритм 1 покажет большее значение оцениваемой метрики, чем алгоритм 2. Например, $\hat{A}_{12} = 0.5$ значит, что алгоритмы работают одинаково, а $\hat{A}_{12} = 0.75$ значит, что первый алгоритм превосходит второй с вероятностью 75%.

6.2 Результаты

Реализованный алгоритм был апробирован на широко используемых Java-пакетах разного назначения (табл. 1-2).

На каждом выбранном пакете фаззер был запущен 30 раз с фиксированным числом запусков. Все эксперименты продемонстрировали значение $p < 0.05$, благодаря чему в соответствии с тестом Манна-Уитни делается вывод о статистической значимости результатов.

Результаты указывают на то, что предложенный алгоритм увеличивает количество обнаруженных ошибок и уникальных поведений тестируемых программ по сравнению со случайным выбором мутаций, как с точки зрения медианных значений, так и с точки зрения величины эффекта. В ходе экспериментов не было обнаружено ни одной программы, на которой разработанный алгоритм выбора мутаций показал бы худшие результаты, чем случайный выбор.

7. Заключение

В работе предложен новый метод планирования семантических мутаций в фаззинге по методу серого ящика, который основан на модели динамической задачи о многоруком бандите и учитывает особенности процесса фаззинга. Предложенный алгоритм продемонстрировал статистически значимое увеличение количества обнаруживаемых поведений и числа найденных ошибок при тестировании широко используемых Java-пакетов различного назначения.

Табл.1. Количество ошибок, найденных ошибок в протестированных пакетах.
Table 1. Number of errors found in tested packages.

	Медианное число ошибок при случайном выборе мутаций	Медианное число ошибок при адаптивном выборе мутаций	$\hat{A}_{12}$
fastjson2 (~ 1.9 млн. строк кода)	25	31	0.81
commons-math3 (~ 431 тыс. строк кода)	3	3	0.65
closure-compiler (~ 600 тыс. строк кода)	1	2	0.74
jsoup (~ 49 тыс. строк кода)	6	7	0.86
Guava (~ 1 млн. строк кода)	17	22	0.74

Табл.2. Количество воспроизведенных поведений (трасс исполнения) протестированных пакетов.
Table 2. Number of behaviors (execution traces) detected in tested packages.

	Медианное число поведений при случайном выборе мутаций	Медианное число поведений при адаптивном выборе мутаций	$\hat{A}_{12}$
fastjson2 (~ 1.9 млн. строк кода)	~ 19 тыс.	~ 29 тыс.	0.81
commons-math3 (~ 431 тыс. строк кода)	~ 181 тыс.	~ 185 тыс.	0.65
closure-compiler (~ 600 тыс. строк кода)	~ 13 тыс.	~ 18 тыс.	0.74
jsoup (~ 49 тыс. строк кода)	~ 44 тыс.	~ 45 тыс.	0.86
Guava (~ 1 млн. строк кода)	80	95	0.74

Разработанный подход легко может быть интегрирован в существующие инструменты фаззинга благодаря тому, что он не зависит от других систем фаззинга, а также не требует значительных вычислительных ресурсов и какого-либо предобучения.

В качестве перспективных направлений развития данного подхода можно выделить:

- Комбинирование с алгоритмами выбора значений. Существует множество алгоритмов выбора входных значений, комбинация которых с алгоритмом выбора мутаций с учетом описанных в данной работе наблюдений о характере процесса фаззинга может повысить результативность фаззинга.

- Позиционирование мутаций: упомянутый выше метод тепловых карт показывает высокую эффективность в фаззинге бинарных данных, но не подходит в явном виде для структурного фаззинга. Адаптированный для структурного фаззинга метод тепловых карт позволит распознавать перспективные точки приложения мутаций и направлять процесс фаззинга более эффективно.
- Выбор мутационных стратегий в соответствии со структурой тестируемого кода. Наблюдения показывают, что эффективные мутационные стратегии различны в зависимости от тестируемых программ, однако на текущий момент нет исследований о том, с чем связано это различие. Исследование зависимости эффективности стратегий фаззинга от структуры тестируемых программ может пролить свет на эту проблему и позволить существенно повысить результативность инструментов фаззера с помощью их гибкой настройки.

Список литературы / References

[1]. M. Eberlein, Y. Noller, T. Vogel, and L. Grunske, «Evolutionary Grammar-Based Fuzzing», ArXiv, 2020. Accessed: https://api.semanticscholar.org/CorpusID:220961614.
[2]. R. Dutra, R. Gopinath, and A. Zeller, «FormatFuzzer: Effective Fuzzing of Binary File Formats», CoRR, 2021. Accessed: https://arxiv.org/abs/2109.11277.
[3]. X. Zhang et al., «A Survey of Protocol Fuzzing», ACM Comput. Surv., v. 57, No. 2, Oct. 2024, doi:10.1145/3696788.
[4]. H. Han, D. Oh, and S. K. Cha, «CodeAlchemist: Semantics-Aware Code Generation to Find Vulnerabilities in JavaScript Engines», Proceedings 2019 Network and Distributed System Security Symposium, 2019. Accessed: https://api.semanticscholar.org/CorpusID:142503428.
[5]. V. J. Manès et al., «The Art, Science, and Engineering of Fuzzing: A Survey», IEEE Transactions on Software Engineering, v. 47, No. 11, pp. 2312–2331, 2021, doi: 10.1109/TSE.2019.2946563.
[6]. M. Eberlein, Y. Noller, T. Vogel, and L. Grunske, «Evolutionary Grammar-Based Fuzzing». 2020 г.
[7]. Н. Ерохина, «Метод мутации сложноструктурированных входных данных при фаззинг-тестировании JavaScript интерпретаторов», Труды Института системного программирования РАН, т. 35, вып. 5, сс. 55–66, 2024, doi: 10.15514/ISPRAS-2022-35(5)-4.
[8]. A. Slowik, and H. Kwasnicka, «Evolutionary algorithms and their applications to engineering problems», Neural Computing and Applications, v. 32, No. 16, pp. 12363–12379, Mar. 2020, doi: 10.1007/s00521-020-04832-8.
[9]. C. Lyu et al., «MOPT: Optimized Mutation Scheduling for Fuzzers», в 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1949–1966. Accessed: https://www.usenix.org/conference/usenixsecurity19/presentation/lyu.
[10]. T. D. Nguyen, L. H. Pham, and J. Sun, «Fuzzing with Quantitative and Adaptive Hot-Bytes Identification». Доступно на: https://arxiv.org/abs/2307.02289.
[11]. L. Binosi, L. Rullo, M. Polino, M. Carminati, and S. Zanero, «Rainfuzz: Reinforcement-Learning Driven Heat-Maps for Boosting Coverage-Guided Fuzzing», в International Conference on Pattern Recognition Applications and Methods, 2023. Доступно на: https://api.semanticscholar.org/CorpusID:257361890.
[12]. C. Chen, V. Gohil, R. Kande, A.-R. Sadeghi, and J. Rajendran, «PSOFuzz: Fuzzing Processors with Particle Swarm Optimization». Accessed: https://arxiv.org/abs/2307.14480.
[13]. P. Jauernig, D. Jakobović, S. Picek, E. Stapf, and A. Sadeghi, «DARWIN: Survival of the Fittest Fuzzing Mutators», Jan. 2023, doi: 10.14722/ndss.2023.23159.
[14]. K. Böttinger, P. Godefroid, and R. Singh, «Deep Reinforcement Fuzzing», CoRR, 2018. Accessed: http://arxiv.org/abs/1801.04589.
[15]. D. She, R. Krishna, L. Yan, S. Jana, and B. Ray, «MTFuzz: fuzzing with a multi-task neural network», в Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, в ESEC/FSE '20. ACM, Nov. 2020. doi: 10.1145/3368089.3409723.

[16]. K. Gong, W. Yang, B. Cui, and C. Chen, «DRLFCfuzzer: fuzzing with Deep-Reinforcement-Learning under Format Constraints», in 2022 2nd International Conference on Electronic Information Engineering and Computer Technology (EIECT), Oct. 2022, pp. 374–380. doi: 10.1109/EIECT58010.2022.00080.
[17]. S. Li et al., «Deep Learning for Coverage-Guided Fuzzing: How Far are We?», IEEE Transactions on Dependable and Secure Computing, pp. 1–13, 2022, doi: 10.1109/TDSC.2022.3200525.
[18]. A. Slivkins, «Introduction to Multi-Armed Bandits», CoRR, 2019. Accessed: http://arxiv.org/abs/1904.07272.
[19]. V. Gohil, R. Kande, C. Chen, A.-R. Sadeghi, and J. Rajendran, «MABFuzz: Multi-Armed Bandit Algorithms for Fuzzing Processors». 2023.
[20]. T. Yue et al., «EcoFuzz: Adaptive Energy-Saving Greybox Fuzzing as a Variant of the Adversarial Multi-Armed Bandit», в 29th USENIX Security Symposium (USENIX Security 20), USENIX Association, Aug. 2020, pp. 2307-2324. Accessed: https://www.usenix.org/conference/usenixsecurity20/presentation/yue.
[21]. Z. Huang, X. Song, Y. Luo, J. Yang, and B. Cui, «Syzballer: Kernel Fuzzing Based on Basic Block Weight and Multi-armed Bandit», in 2022 IEEE 8th International Conference on Computer and Communications (ICCC), Dec. 2022, pp. 2364–2369. doi: 10.1109/ICCC56324.2022.10065711.
[22]. G. Zhang et al., «MobFuzz: Adaptive Multi-objective Optimization in Gray-box Fuzzing», в Proceedings 2022 Network and Distributed System Security Symposium, в NDSS 2022. Internet Society, 2022. doi: 10.14722/ndss.2022.24314.
[23]. M. Lee, S. Cha, и H. Oh, «Learning Seed-Adaptive Mutation Strategies for Greybox Fuzzing», в 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023, pp. 384–396. doi: 10.1109/ICSE48619.2023.00043.
[24]. M. Böhme, and B. Falk, «Fuzzing: on the exponential cost of vulnerability discovery», 2020, pp. 713–724. doi:10.1145/3368089.3409729.
[25]. S. Karamcheti, G. Mann, and D. S. Rosenberg, «Adaptive Grey-Box Fuzz-Testing with Thompson Sampling», CoRR, 2018. Accessed: http://arxiv.org/abs/1808.08256.
[26]. N. Gupta, O.-C. Granmo, и A. Agrawala, «Thompson Sampling for Dynamic Multi-armed Bandits», Machine Learning and Applications, Fourth International Conference on, v. 1, pp. 484–489, 2011, doi: 10.1109/ICMLA.2011.144.
[27]. P. Auer et al., «Achieving Optimal Dynamic Regret for Non-stationary Bandits without Prior Information», в Proceedings of the Thirty-Second Conference on Learning Theory, A. Beygelzimer and D. Hsu, Ed., in Proceedings of Machine Learning Research, vol. 99. PMLR, 2019, pp. 159–163. Accessed https://proceedings.mlr.press/v99/auer19b.html.
[28]. M. Eceiza, J. L. Flores, and M. Iturbe, «Improving fuzzing assessment methods through the analysis of metrics and experimental conditions», Computers & Security, v. 124, p. 102946, 2023, doi: https://doi.org/10.1016/j.cose.2022.102946.
[29]. Y. Li et al., «UNIFUZZ: A Holistic and Pragmatic Metrics-Driven Platform for Evaluating Fuzzers», в 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, Aug. 2021, pp. 2777–2794. Доступно на: https://www.usenix.org/conference/usenixsecurity21/presentation/li-yuwei.
[30]. D. Paaßen, S. Surminski, M. Rodler, and L. Davi, «My Fuzzer Beats Them All! Developing a Framework for Fair Evaluation and Comparison of Fuzzers», CoRR, 2021. Accessed: https://arxiv.org/abs/2108.07076.
[31]. J. Bundt, A. Fasano, B. Dolan-Gavitt, W. Robertson, and T. Leek, «Evaluating Synthetic Bugs», в Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, в ASIA CCS '21. Virtual Event, Hong Kong: Association for Computing Machinery, 2021, pp. 716–730. doi: 10.1145/3433210.3453096.
[32]. Arcuri and L. Briand, «A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering», Software Testing, Verification and Reliability, v. 24, No. 3, pp. 219–250, 2014, doi: https://doi.org/10.1002/stvr.1486.
[33]. A. Vargha and H. Delaney, «A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong», Journal of Educational and Behavioral Statistics – J EDUC BEHAV STAT, v. 25, No. 2, pp. 101-132, 2000, doi: 10.2307/1165329.

Информация об авторах / Information about authors

Григорий Романович РАЙКИН – аспирант института компьютерных наук ИТМО. Сфера научных интересов: статический и динамический анализ программ, фаззинг, формальная спецификация программ.

Grigoriy Romanovich RAYKIN – postgraduate student at the ITMO Institute of Computer Science. Research interests: static and dynamic software analysis, fuzzing, formal software specification.

Максим Сергеевич ПЕЛЕВИН – магистр факультета компьютерных технологий и информатики СПбГЭТУ "ЛЭТИ". Сфера научных интересов: динамический и статический анализ программ, фаззинг.

Maksim Sergeevich PELEVIN – master of the Faculty of Computer Science and Technology at ETU "LETI". Research interests: dynamic and static software analysis, fuzzing.

Владимир Михайлович ИЦЫКСОН – кандидат технических наук, доцент института прикладных компьютерных наук ИТМО. Сфера научных интересов: статический и динамический анализ программ, верификация программного обеспечения, методы обнаружения дефектов в исходном коде, методы автоматизации тестирования программ.

Vladimir Mikhailovich ITSYKSON – Cand. Sci. (Tech.), associate professor of the Institute of Applied Computer Science ITMO. Research interests: static and dynamic software analysis, software verification, methods for detecting defects in source code, methods for automating software testing.