КОМПЛЕКС МЕТОДОВ ГЕНЕТИЧЕСКОЙ ДЕЭВОЛЮЦИИ ПРЕДСТАВЛЕНИЙ ПРОГРАММЫ

Израилов К. Е.[1]

DOI: 10.21681/2311-3456-2025-4-93-106

Цель исследования: *повышение эффективности нейтрализации уязвимостей программы за счет интеллектуализации ее реверс-инжиниринга с помощью генетических алгоритмов*

Методы исследования: *системный анализ и методы оптимизации, теория графов, функциональный и структурный синтез, общая методология программирования и теория компиляторов.*

Полученные результаты: *синтезирован иерархический трехуровневый комплекс методов, состоящий из метода генетического реверс-инжиниринга программы, метода генетической деэволюции ее соседних представлений (машинного и исходного кода, алгоритмов, архитектуры и т.д.), и группы методов для реализации основополагающих операций генетических алгоритмов.*

Новизна *комплекса методов заключается в их ориентированности на решение задачи реверс-инжиниринга путем прямых преобразований программы в последующие представления, что отличает их от классических, выполняющих обратные преобразования. Также, алгоритмы группы методов комплекса основаны на работе с оригинальной моделью исходного кода, представляющей его как последовательность ген.*

Ключевые слова: *нейтрализация уязвимостей, реверс-инжиниринг, искусственный интеллект, генетические алгоритмы, комплекс методов.*

Введение

Наличие уязвимостей в программном обеспечении является актуальной проблемой современного информационно-технологического мира [1]. Одним из путей разрешения данной проблемы считается непосредственное обнаружение и устранение уязвимостей в различных представлениях программы [2] – машинном, исходном и байткоде, алгоритмах, архитектуре и др. При этом, для достижения высокой эффективности такой нейтрализации уязвимостей необходимо предварительное получение представлений программы, в которых уязвимости были внедрены; каждое же такое преставление после исправления должно преобразовываться в конечную программу. В интересах этого требуется создание нового подхода к реверс-инжинирингу программ, поскольку существующие не удовлетворяют указанным условиям [3] – часть подходов не позволяет работать со всем набором предыдущих представлений программы, а другая часть получает псевдо-представления, не преобразуемые в конечное; так, например, в описании коммерческого продукта для анализа машинного кода IDA Pro изначально заявлено, что он восстанавливает псевдо-исходный код, который в принципе не обязан быть даже компилируемым, не говоря уже о тождественности машинному. Предлагаемый автором подход основан на решении оптимизационной задачи по подбору конструкций предыдущего представления для его полного соответствия заданному, что как раз и позволяет получать высокоуровневые представления, подходящие для поиска уязвимостей, которые затем могут быть «собраны» (например, компиляцией) в выполняемую программу с функциональностью, идентичной первоначальной; такой процесс обратных преобразований между соседними представлениями назван их деэволюцией. Исходя из того, что решение указанной оптимизационной задачи основано на применении генетических алгоритмов (точнее их модифицированной версии), как деэволюция представлений, так и полная их цепочка – т. е. реверс-инжиниринг, были названы генетическими (сокр. ГДЭ и ГРИ, соответственно); при этом, частный случай получения исходного кода программы по ее машинному коду логично был назван генетической декомпиляцией (сокр. ГДК). Проведенные эксперименты показали как работоспособность данной концепции реверс-инжиниринга, так и ее практическую реализуемость и превосходство над аналогами. Описанию же самого подхода ГРИ в виде комплекса методов и посвящена данная статья.

Генетический подход

Прежде чем перейти к описанию предложенного подхода (в виде комплекса методов), рассмотрим основные исторические предпосылки, лежащие в его основе. Также в качестве примеров применения подхода выберем декомпиляцию машинного кода в исходный, как крайне востребованную на сегодняшний день задачу реверс-инжиниринга.

1 Израилов Константин Евгеньевич, кандидат технических наук, доцент, профессор кафедры прикладной математики и безопасности информационных технологий Санкт-Петербургского университета государственной противопожарной службы МЧС России, Санкт-Петербург, ORCHID: http://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56122749800. E-mail: konstantin.izrailov@mail.ru

Существуют различные подходы к проведению реверс-инжиниринга, которые с точки зрения их исторического появления можно упорядочить следующим образом: ручной, алгоритмический, полный перебор (как гипотетический, практически не применимый на практике) и интеллектуальный. При этом у каждого их них есть свои сильные и слабые стороны, которые не позволяют обеспечить качественного повышения эффективности всего процесса нейтрализации уязвимостей.

Рассмотрим более подробно подход, использующий искусственный интеллект (далее – ИИ), поскольку он является современным трендом информационных технологий, востребованным в огромном количестве областей [4].

ИИ активно исследуется и используется при решении относительно частных задач области реверс-инжиниринга, а именно, следующих: глубокое машинное обучение для анализа машинного кода (декомпиляция, восстановление метаданных) [5], декомпиляция машинного кода с применением малых и больших языковых моделей [6, 7], именование функций машинного кода с поддержкой оптимизаций компилятора на базе графовых нейронных сетей [8], восстановление отладочной информации (имен и типов переменных) с использованием машинного обучения [9], идентификация функций в машинном коде на основе рекуррентных нейронных сетей [10]. Впрочем, использование ИИ для деэволюции представлений программы, как правило, ведет к ряду существенных недостатков, таких, как необходимость в датасетах огромного размера, высокая вычислительная стоимость и ресурсоемкость, слабая интерпретируемость получаемых результатов (т. е. отсутствие ответа на вопрос, почему ИИ восстановил по машинному коду именно такой код программы), эффект «галлюцинирования» (когда создается выдуманная информация, не имеющая ничего общего с реальным кодом). Также, ИИ может ошибаться при декомпиляции программ со сложной логикой работы; например, следующий «запутанный» исходный код:

```
int funct(int p) {
  goto aaa2;

aaa1:;
  p++;
  goto aaa3;

aaa2:;
  goto aaa1;

aaa3:;
  return p;
}
```

производит увеличение аргумента «p» функции funct(...) на 1, возвращая тем самым всегда значение «p + 1»; соответствующий ему машинный код для архитектуры x86-64 является следующим:

```
push ebp
mov  ebp, esp

jmp  SHORT $aaa2$3

$aaa1$4:
jmp  SHORT $aaa3$5

$aaa2$3:
mov  eax, DWORD PTR _p$[ebp]
add  eax, 1
mov  DWORD PTR _p$[ebp], eax

jmp  SHORT $aaa1$4

$aaa3$5:
mov  eax, DWORD PTR _p$[ebp]

pop  ebp
ret  0
```

Запрос же на декомпиляцию приведенного машинного кода для генеративного ИИ с моделью GPT-4o mini привел к выдаче следующего результата: «Этот код реализует бесконечный цикл, который увеличивает значение переменной _p$ на 1. Так как после увеличения значения происходит переход обратно к метке $aaa1$4, которая снова переходит к $aaa3$5, где значение снова загружается, цикл не завершится, пока функция не будет принудительно остановлена». Таким образом, данная модель произвела некорректную декомпиляцию (а, точнее, не смогла ее произвести).

Аналогичный запрос для другого популярного генеративного ИИ с моделью DeepSeek-R1 выдал следующий исходный код:

```
int func(int *p) {
    return ++(*p);
}
```

что, конечно, близко к логике истинного исходного кода, но не соответствует ей, т. к. аргумент функции передается по указателю, а не по значению, а сама некорректно восстановленная функция тем самым обладает эффектом изменения внешних данных.

При этом остается «открытым» вопрос полной корректности восстанавливаемых с помощью ИИ представлений программы – т. е., например, тождественность полученного исходного кода декомпилируемому машинному коду.

Перейдем к описанию самого предложенного подхода ГДЭ, частично лишенного недостатков, свойственных существующим интеллектуальным.

Как указывалось ранее, суть подхода заключается в решении оптимизационной задачи по подбору конструкций предыдущего представления, которые бы были тождественны совокупности конструкций заданного представления; само же решение основано на генетических алгоритмах [11, 12]. Так, если требуется получить исходный код по машинному, то создается случайная популяция экземпляров исходного кода, которые компилируются в машинный и сравниваются с заданным. Из исходных кодов отбираются те, машинный код которых наиболее близок к заданному – производится их селекция, над которыми затем осуществляются операции скрещивания и мутации, получая новое поколение. Данный итеративный процесс завершится, когда будет получен исходный код, дающий после компиляции машинный код, полностью идентичный заданному. При этом существует достаточно большое количество исследований, посвященных оптимизации работы самих генетических алгоритмов [13, 14, 15].

Для работы ГДЭ необходимы синтаксисы соседних представлений программы, которые предварительно преобразуются в соответствующие графы синтаксических правил (далее – ГСП); при этом, сам код программы имеет запись в виде хромосомы, в которой каждый ген определяет выбор пути по данному ГСП. Сам выбор целесообразно делать только в тех узлах ГСП, в которых существуют различные пути продолжения синтаксических правил, т. е. в узлах-альтернативах; так, например, в математических выражениях после переменной может идти целый набор различных бинарных операций (что в ГСП является узлом-альтернативой), выбор каждой из которых и задается геном хромосомы-пути. Таким образом, по сравнению с классическим реверс-инжинирингом, преобразующим текущее представление программы в ее предыдущее, предлагаемый подход имеет противоположную направленность. Особенность подхода и его отличие от альтернативных может быть представлено следующим формальным образом.

Предположим, что представление программы является совокупностью двух неразделимых компонентов: формы и содержания, первый из которых определяет внешнее представление программы, а второй – ее внутреннюю суть или логику; тогда представление программы может быть записано следующим образом:

$$Rep_x = \langle Rep_x^{Form}, Rep_x^{Content} \rangle, \quad (1)$$

где Rep_x – x-ое представление программы, определяемое кортежем из его формы Rep_x^{Form} и содержания $Rep_x^{Content}$. Очевидно, что в процессе разработки логика программы должна сохраняться, а ее форма будет постепенно меняться или эволюционировать (точнее переходить от человеко-ориентированной в машинно-ориентированную), т. е.:

$$\begin{cases} Rep_x^{Form} \neq Rep_{x+1}^{Form} \\ Rep_x^{Content} = Rep_{x+1}^{Content} \end{cases}, \quad (2)$$

где x и $x + 1$ – индексы текущего и последующего представлений программы.

Тогда весь процесс программного инжиниринга может быть записан следующим образом:

$$\begin{cases} Rep_x \longrightarrow Rep_{x+1} \\ Rep_x^{Form} \longrightarrow Rep_{x+1}^{Form} \\ Rep_x^{Content} = Rep_{x+1}^{Content} \end{cases}, \quad (3)$$

где «⟶» – операция прямого преобразования представлений и их компонентов.

Процесс же реверс-инжиниринга в этом случае имеет следующую запись:

$$\begin{cases} Rep_{x-1} \longleftarrow Rep_x \\ Rep_{x-1}^{Form} \longleftarrow Rep_x^{Form} \\ Rep_{x-1}^{Content} = Rep_x^{Content} \end{cases}, \quad (4)$$

где «⟵» – операция обратного преобразования представлений и их компонентов.

Принцип действия классического реверс-инжиниринга (подходы, применяемые в котором уже были упомянуты), можно записать следующим образом:

$$Rep_{x-1}^{Form} = Reverse(Rep_x^{Form}), \quad (5)$$

где $Reverse(\ldots)$ – операция обратного ручного, алгоритмического или интеллектуального преобразования формы представления.

Отличие же подхода ГДЭ от классических в этом случае видно по следующей его записи:

$$\{Rep_{x-1}\}^{n+1} = GenAlgOpt(\{Rep_{x-1}\}^n) : Compile(Rep_{x-1}) = Rep_x, \quad (6)$$

где $\{\ldots\}^n$ и $\{\ldots\}^{n+1}$ – n-ая и $n+1$-ая популяции экземпляров программы в предыдущем представлении (например, исходного кода); $GenOpt(\ldots)$ – итерационная операция решения оптимизационной задачи с помощью генетического алгоритма: $Compile(\ldots)$ – операция получения из программы в предыдущем представлении (например, исходного кода) его текущего представления (например, машинного кода) прямым преобразованием (например, компиляцией).

Предлагаемый подход ГДЭ является некоторым развитием существующих и активно используемых – алгоритмического и интеллектуального (при этом, хотя ручной подход также применяется, однако его использование скорее связано с безысходностью по причине невозможности повеления реверс-инжиниринга остальными, поскольку он крайне время- и ресурсозатратен). Так, хотя алгоритмический подход дает гарантированный результат, тем не менее он основан на заложенных экспертом правилах,

которые не могут покрыть все множество необходимых преобразований машинного кода в исходный. В противоположность этому, хотя интеллектуальный подход способен избегать шаблонных конструкций машинного кода, однако получаемый с помощью него результат не всегда соответствует требуемому – возникают ошибки восстановления исходного кода, который не гарантированно соответствует машинному. ГДК (как частный случай ГДЭ) представляет собой комбинацию наиболее успешных возможностей данных подходов следующим образом. Во-первых, применение средств компиляции при решении оптимизационной задачи позволяет утверждать о тождественности полученного исходного кода заданному машинному – специфика алгоритмического подхода. А, во-вторых, вариативность в работе генетического алгоритма (поскольку, операции скрещивания и мутации построены в том числе на случайных выборах генов) позволяет расширять строгие правила преобразования внешаблонными решениями – специфика интеллектуального подхода.

Для обоснования работоспособности подхода ГДЭ, центрального во всем ГРИ, был создан программный прототип ГДК, производящий декомпиляцию различных экземпляров машинного кода для заданных синтаксисов. При реализации прототипа использовался язык программирования Python версии 3.11, авторский код модифицированного ядра генетических алгоритмов и его основных операций, а также вспомогательные библиотеки: collections, copy, enum, os, random, subprocess и др.

Комплекс методов генетической деэволюции представлений

Для работы всего ГРИ необходим целый комплекс соответствующих методов (далее – Комплекс), связанных иерархически – т. е. каждый вышестоящий метод при выполнении использует результаты работы нижестоящего (или их группу), как параметр. Так, на верхнем уровне расположен метод генетического реверс-инжиниринга программы (далее – Метод-ГРИ), который в процессе работы для обратного преобразования соседних представлений применяет метод их генетической деэволюции (далее – Метод-ГДЭ), который в свою очередь использует группу методов (далее – Группа методов), предназначенных для реализации основополагающих операций генетических алгоритмов, основанных на генетическом представлении особей популяции [16]: методы вычисления метрики близости (двух экземпляров программ в текущем представлении), а также скрещивания и мутации ген особей; пользователем Комплекса является эксперт по информационной безопасности и реверс-инжинирингу (далее – Эксперт). Блок-схемы и псевдокод методов с необходимыми пояснениями приводятся далее.

Метод генетического реверс-инжиниринга программы

Принцип работы Метода-ГРИ заключается в итерационном (т. е. последовательном однотипном) преобразовании представлений программы из текущего (как правило, конечного, т. е. выполняемого, такого, как машинный код) во все более высокоуровневые (например, в исходный код, алгоритмы, архитектуру) с нейтрализацией в них уязвимостей. Затем, требуется пересборка конечного представления программы, которое тем самым становится безопасным. Соответствующая блок-схема Метода-ГРИ приведена на рис. 1; здесь и далее белым фоном обозначены основные шаги метода, зеленым – ввод и вывод данных, желтым – специализированные конструкции (начало, конец, границы цикла), синим – вызовы внешних методов, оранжевым – условный переход, серым – хранилища данных; пунктирные линии обозначают связь по данным.

На блок-схеме (см. рис. 1) присутствуют следующие элементы работы Метода-ГРИ; здесь и далее операция «⟶» означает получение одного объекта из другого, операция «+=» – добавление элемента к набору, а пометка «*» – безопасную версию представления:

1) «Начало» – запуск метод Экспертом;

2) «Ввод конечного (небезопасного) представления программы: Rep_i» – пользователь метода выбирает имеющееся i-е представление программы (Rep_i), реверс-инжиниринг которого с последующим обнаружением и устранением уязвимостей необходимо произвести;

3) «Нейтрализация уязвимостей в конечном представлении: $Rep_i \longrightarrow Rep_i^*$» – внешний вызов процесса нейтрализации уязвимостей в текущем представлении (Rep_i) с получением его безопасной версии (Rep_i^*), поскольку существует определенное количество способов их обнаружения и устранения в конечных представлениях программы (например, статические анализаторы машинного кода);

4) «Идентификация синтаксиса текущего (конечного) представления: $Rep_i^* \longrightarrow Syntax_i$» – получение синтаксиса ($Syntax_i$) текущего представления программы (Rep_i^*), такого, как язык программирования или процессорная архитектура [17], что необходимо для дальнейшей деэволюции представлений;

5) «Модель жизненного цикла программы: $\{Rep_x, x \longrightarrow y\}$» – информационный объект ($\{\ldots\}$), хранящий возможные представления программ (Rep_x) и преобразования между ними ($x \longrightarrow y$) [18, 19];

6) «Существует предыдущее представление: Rep_{i-1}?» – проверка согласно модели жизненного цикла программы наличия предыдущего представления программы (Rep_{i-1}), в котором требуется нейтрализация уязвимостей;

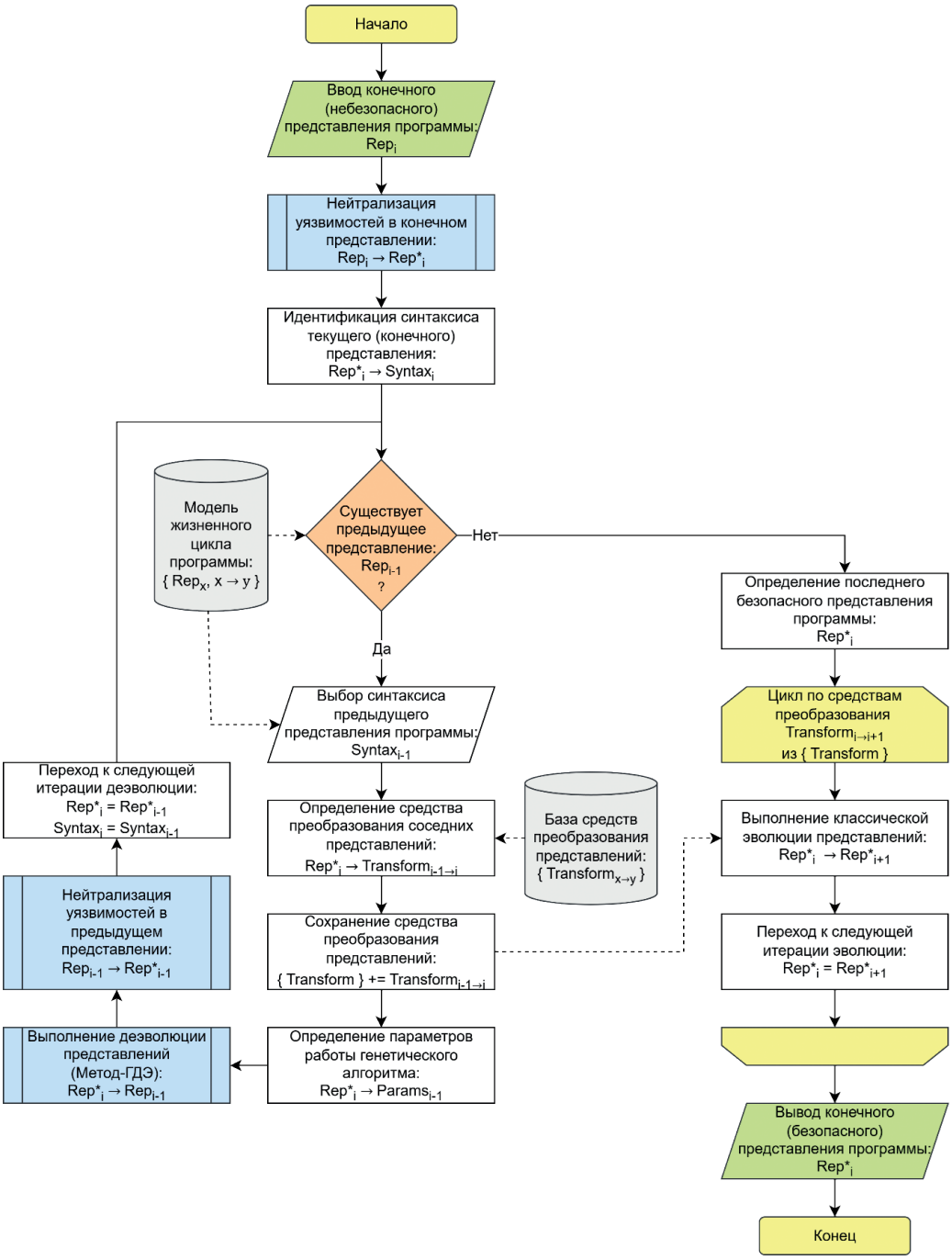

Рис. 1. Блок-схема генетического реверс-инжиниринга программы (с нейтрализацией уязвимостей)

7) «Выбор синтаксиса предыдущего представления программы: $Syntax_{i-1}$» – получение согласно модели жизненного цикла программы синтаксиса ее предыдущего представления ($Syntax_{i-1}$), что необходимо для работы ГДЭ;

8) «База средств преобразования представлений: $\{Transform_{x \to y}\}$» – информационный объект ($\{\ldots\}$) с существующими средствами получения ($Transform$) последующих представлений из текущих ($x \longrightarrow y$), такими, как генераторы исходного кода, средства компиляции, ассемблирования и пр.;

9) «Определение средства преобразования соседних представлений: $Rep_i^* \longrightarrow Transform_{i-1 \to i}$» – определение согласно базе средств преобразования представлений необходимого для получения текущего представления (Rep_i^*) из предыдущего ($i-1 \longrightarrow i$);

10) «Сохранение средства преобразования представлений: $\{Transform\}$ += $Transform_{i-1 \to i}$» – добавление определенного средства преобразования представлений ($Transform_{i-1 \to i}$) в набор ($\{\ldots\}$) таких средств ($Transform$);

11) «Определение параметров работы генетического алгоритма: $Rep_i^* \longrightarrow Params_{i-1}$» – определение по текущему представлению (Rep_i^*) таких параметров генетического алгоритма, работающего с предыдущим представлением ($Params_{i-1}$), как длина хромосомы, алгоритмы операций селекции, скрещивания и мутации, частота выполнения последних операций и др.;

12) «Выполнение деэволюции представлений (Метод-ГДЭ): $Rep_i^* \longrightarrow Rep_{i-1}$» – внешний вызов второго метода в иерархии Комплекса, производящего непосредственное преобразование текущего представления с нейтрализованными уязвимостями (Rep_i^*) в потенциально небезопасное предыдущее (Rep_{i-1}); при этом, в Метод-ГДЭ передаются такие настройки, как программа в текущем представлении (Rep_i^*), синтаксисы текущего ($Syntax_i$) и предыдущего ($Syntax_{i-1}$) представлений, средство преобразования предыдущего представления в текущее ($Transform_{i-1 \to i}$) и параметры генетического алгоритма ($Params_{i-1}$);

13) «Нейтрализация уязвимостей в предыдущем представлении: $Rep_{i-1} \longrightarrow Rep_{i-1}^*$» внешний вызов процесса нейтрализации уязвимостей в предыдущем представлении (Rep_{i-1}) с получением его безопасной версии (Rep_{i-1}^*) по аналогичным с элементом 3 причинам;

14) «Переход к следующей итерации деэволюции: $Rep_i^* = Rep_{i-1}^*$, $Syntax_i = Syntax_{i-1}$» – установление в качестве текущего представления (Rep_i^*) и его синтаксиса ($Syntax_i$) тех, которые ранее являлись предыдущими (Rep_{i-1}^* и $Syntax_{i-1}$), обеспечивая тем самым переход к следующей итерации реверс-инжиниринга программы;

15) «Определение последнего безопасного представления программы: Rep_i^*» – установление в качестве представления для сборки конечной безопасной программы того, которое обрабатывалось на последней итерации реверс-инжиниринга (Rep_i^*);

16) «Цикл по средствам преобразования $Transform_{i \to i+1}$ из $\{Transform_{x \to y}\}$» – осуществление циклического перебора средств трансформации представлений ($Transform_{i \to i+1}$), сохраненных в наборе ранее ($\{Transform_{x \to y}\}$) при осуществлении их деэволюции;

17) «Выполнение классической эволюции представлений: $Rep_i^* \longrightarrow Rep_{i+1}^*$» – получение в цикле каждого последующего представления программы (Rep_{i+1}^*) по текущему (Rep_i^*), например, путем генерации исходного кода, компиляции или ассемблирования;

18) «Переход к следующей итерации эволюции: $Rep_i^* = Rep_{i+1}^*$» – установление в качестве текущего представления (Rep_i^*) того, которое ранее считалось последующим (Rep_{i+1}^*), обеспечивая тем самым переход к следующей итерации получения конечного (безопасного) представления программы;

19) «Вывод конечного (безопасного) представления программы: Rep_i^*» – вывод представления программы, полученного на последней итерации цикла, которое является конечным и безопасным, т. е. результатом ГРИ, в процессе которого была произведена нейтрализация уязвимостей;

20) «Конец» – завершение метода.

Согласно блок-схеме, для работы Метода-ГРИ необходимы методы нейтрализации (т. е. обнаружения и устранения) уязвимостей во всех представлениях программы, а также, второй элемент Комплекс – Метод-ГДЭ.

Метод генетической деэволюции соседних представлений программы

Принцип работы Метода-ГДЭ заключается в итерационном подборе конструкций предыдущего представления программы по мере приближения полученного из них представления к текущему, деэволюцию которого необходимо произвести. При этом, наилучшие (отобранные) экземпляры в предыдущем представлении будут скрещиваться друг с другом и мутировать для получения наиболее близких к искомой программ. Так, например, метод будет подбирать ключевые слова, переменные и константы языка программирования С, компилируя их в машинный код, сравнивая с заданным, исходя из этого отбирая наиболее близкие, «перемешивая» между собой их конструкции и внося случайные изменения. Такой эволюционный (а, точнее, генетический) процесс решения оптимизационной задачи завершится, когда будет найдена программа в предыдущем представлении (например, ее исходный код), которая после преобразования тождественна заданной программе в текущем представлении (например, машинному коду). Соответствующая блок-схема Метода-ГДЭ приведена на рис. 2.

На блок-схеме (см. рис. 2) присутствуют следующие элементы работы Метода-ГДЭ:

1) «Начало» – запуск метод;

2) «Ввод настроек метода: текущее представление программы (Rep_i), синтаксис текущего ($Syntax_i$) и предыдущего представлений ($Syntax_{i-1}$), средство их преобразования ($Transform_{i-1 \to i}$), параметры генетического алгоритма ($Params$)» – в метод передаются параметры, необходимые для получения предыдущего i–1-го представления по i-му текущему (Rep_i) с помощью генетического алгоритма, в операциях которого используются синтаксисы обоих представлений ($Syntax_{i-1}$ и $Syntax_i$) и средство получения текущего представления программы, соответствующего ее некоторому экземпляру в предыдущем ($Transform_{i-1 \to i}$); управление работой

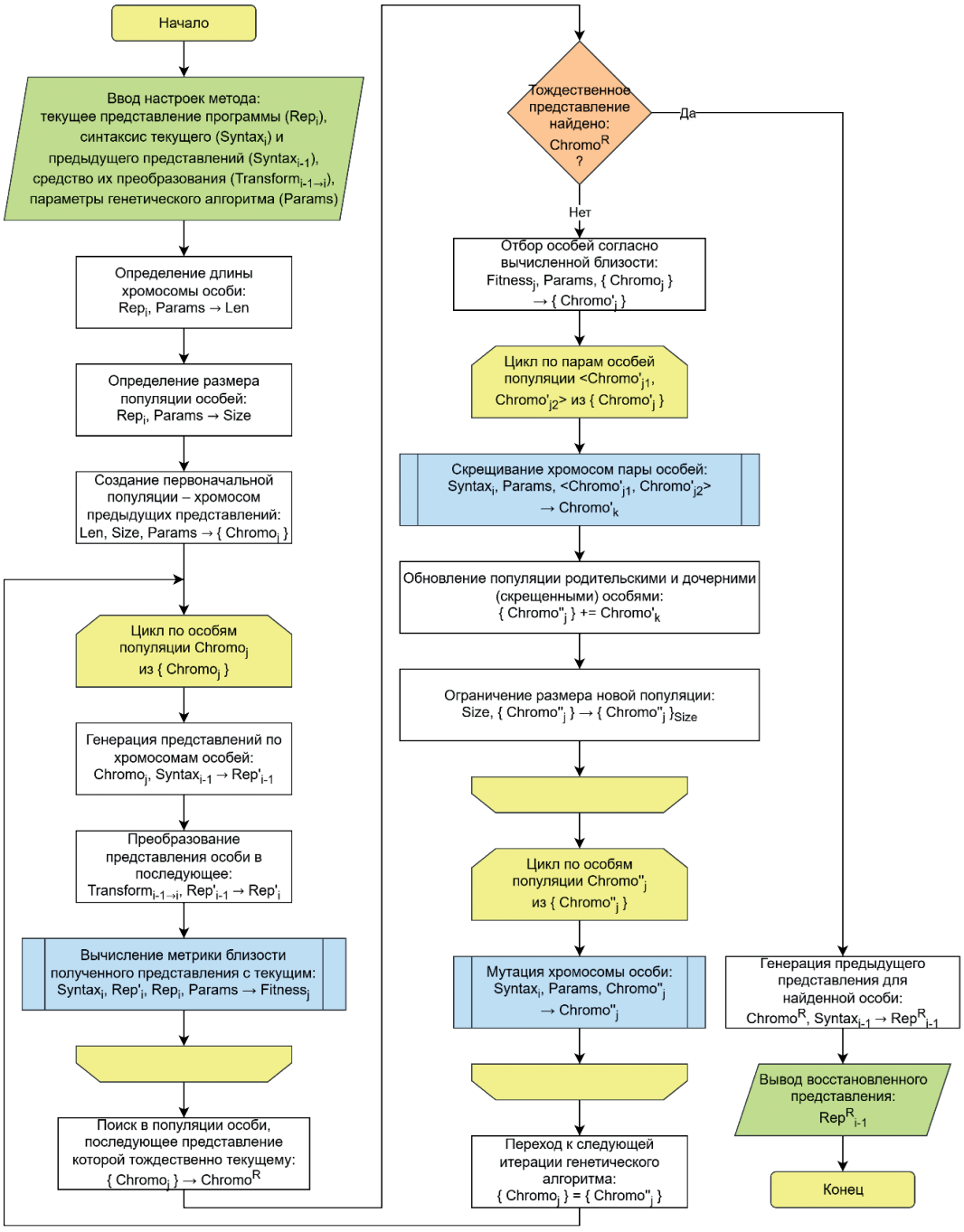

Рис. 2. Блок-схема генетической деэволюции представлений программы

метода осуществляется через переданные в него параметры (*Params*);

3) «Определение длины хромосомы особи: Rep_i, $Params \longrightarrow Len$» – определение близкой к истинной длины (*Len*) хромосомы особи, вычисляемой на основании экземпляра программы в текущем представлении (Rep_i) и заданных настроек метода (*Params*);

4) «Определение размера популяции особей: Rep_i, $Params \longrightarrow Size$» – определение размера популяции особей (*Size*), близкого к оптимальному и вычисляемому на основании экземпляра программы в текущем представлении (Rep_i) и заданных параметров метода (*Params*) [20];

5) «Создание первоначальной популяции – хромосом предыдущих представлений: $Len, Size, Params \longrightarrow \longrightarrow \{Chromo_j\}$» – создание первоначальной популяции особей, как множества хромосом, на основании определенных длины хромосомы (*Len*), размера популяции (*Size*) и заданных параметров метода (*Params*);

6) «Цикл по особям популяции $Chromo_j$ из $\{Chromo_j\}$» – осуществление циклического перебора

хромосом особей ($Chromo_j$) из текущей популяции ($\{Chromo_j\}$);

7) «Генерация представлений по хромосомам особей: $Chromo_j$, $Syntax_{i-1} \longrightarrow Rep'_{i-1}$» – создание предыдущего представления программы (Rep'_{i-1}) на основании ее генетической записи в виде хромосомы ($Chromo_j$) с использованием ГСП, заданного синтаксисом этого представления ($Syntax_{i-1}$);

8) «Преобразование представления особи в последующее: $Transform_{i-1\to i}$, $Rep'_{i-1} \longrightarrow Rep'_i$» – получение программы в текущем представлении (Rep'_i) из ее предыдущего представления (Rep'_{i-1}) с помощью средства преобразования ($Transform_{i-1\to i}$);

9) «Вычисление метрики близости полученного представления с текущим: $Syntax_i$, Rep'_i, Rep_i, $Params \longrightarrow Fitness_j$» – внешний вызов 1-го метода Группы для вычисления численной метрики близости двух программ (которая в теории генетических алгоритмов соответствует приспособленности особи или ее фитнес-функции – $Fitness_j$) в текущем представлении с использованием его синтаксиса ($Syntax_i$), одна из которых задана и требует деэволюции (Rep_i), а другая получена преобразованием из особи популяции (Rep'_{i-1}); управление работой метода осуществляется путем передачи необходимых параметров ($Params$);

10) «Поиск в популяции особи, последующее представление которой тождественно текущему: $\{Chromo_j\} \longrightarrow Chromo^R$» – поиск в текущей популяции особи, метрика близости к которой предельно максимальна (например, равна 1, в случае нормированной к единице метрики), что означает тождественность некоторого экземпляра программы ($Chromo^R$) после преобразования к заданному;

11) «Тождественное представление найдено: $Chromo^R$?» – проверка нахождения в популяции экземпляра программы, тождественного заданному, что означает решение задачи деэволюции ($Chromo^R$) и приводит к последующему завершению метода;

12) «Отбор особей согласно вычисленной близости: $Fitness_j$, $Param$, $\{Chromo_j\} \longrightarrow \{Chromo'_j\}$» – отбор особей из текущей популяции ($\{Chromo_j\}$) в новую популяцию ($\{Chromo'_j\}$) исходя из значения метрики близости каждой из них ($Fitness_j$); в простейшем случае может осуществляться сортировкой по убыванию и выбором первых N-особей;

13) «Цикл по парам особей популяции $\langle Chromo'_{j1}, Chromo'_{j2}\rangle$ из $\{Chromo'_j\}$» – осуществление циклического перебора двух хромосом особей (кортеж из $Chromo'_{j1}$ и $Chromo'_{j2}$) из текущей популяции ($\{Chromo'_j\}$) для «смешивания» их ген;

14) «Скрещивание хромосом пары особей: $Syntax_i$, $Params$, $\langle Chromo'_{j1}, Chromo'_{j2}\rangle \longrightarrow Chromo'_k$» – внешний вызов 2-го метода Группы для выполнения операции скрещивания (в рамках генетического алгоритма) хромосом двух родительских особей ($Chromo'_{j1}$ и $Chromo'_{j2}$), приводящей к получению дочерней особи с генами, содержащими родительские; управление работой метода осуществляется путем передачи синтаксиса текущего представления ($Syntax_i$) и необходимых параметров ($Params$);

15) «Обновление популяции родительскими и дочерними (скрещенными) особями: $\{Chromo''_j\}$ += $Chromo'_k$» – формирование новой текущей популяции ($\{Chromo''_j\}$) из отобранных родительских особей и их дочерних ($Chromo'_k$);

16) «Ограничение размера новой популяции: $Size$, $\{Chromo''_j\} \longrightarrow \{Chromo''_j\}_{Size}$» – ограничение результатов работы операции скрещивания, создающей дочерние особи, для предотвращения превышения текущей популяцией ($\{Chromo''_j\}$) определенного ранее размера ($Size$), гарантируя тем самым постоянное количество особей популяции ($\{Chromo''_j\}_{Size}$);

17) «Цикл по особям популяции $Chromo''_j$ из $\{Chromo''_j\}$» – осуществление циклического перебора хромосом особей ($Chromo''_j$) из текущей популяции ($\{Chromo''_j\}$) для мутации ген их хромосом;

18) «Мутация хромосомы особи: $Syntax_i$, $Params$, $Chromo''_j \longrightarrow Chromo''_j$» – внешний вызов 3-го метода Группы для выполнения операции мутации (в рамках генетического алгоритма) генов хромосомы особи ($Chromo''_j$), приводящей к их случайному изменению; управление работой метода осуществляется путем передачи синтаксиса текущего представления ($Syntax_i$) и необходимых параметров ($Params$);

19) «Переход к следующей итерации генетического алгоритма: $\{Chromo_j\} = \{Chromo''_j\}$» – формальный переход к новой эпохе эволюции, в которой предыдущая популяция ($\{Chromo''_j\}$), полученная операциями селекции, скрещивания и мутации, становится текущей ($\{Chromo_j\}$);

20) «Генерация предыдущего представления для найденной особи: $Chromo^R$, $Syntax_{i-1} \longrightarrow Rep^R_{i-1}$» – создание предыдущего представления программы (Rep^R_{i-1}) на основании особи в популяции с максимально возможной метрикой близости к заданной программе в текущем представлении ($Chromo^R$) с использованием ГСП, заданного синтаксисом этого представления ($Syntax_{i-1}$); полученное таким образом представление программы является искомым результатом работы метода;

21) «Вывод восстановленного представления: Rep^R_{i-1}» – вывод из метода найденного представления программы, которое после преобразования тождественно заданному;

22) «Конец» – завершение метода.

Согласно блок-схеме, для работы Метода-ГДЭ необходима Группа методов, отвечающих за операции

вычисления метрики близости экземпляров программы, а также скрещивания и мутации их хромосом.

Группа методов операций генетических алгоритмов

Группа методов на нижнем уровне в иерархии Комплекса состоит из методов для выполнения операций вычисления метрики близости, скрещивания и мутации, работа которых основана на генетическом представлении программы – как пути по ГСП синтаксиса предыдущего представления. Поскольку блок-схема не даст достаточного понимания принципов и особенностей методов Группы, приведем далее интуитивно-понятный псевдокод алгоритмов работы каждого из них.

Первый метод Группы предназначен для определения близости экземпляра программы, полученного из хромосомы особи с заданной программой (т. е. той, деэволюцию которой необходимо произвести). Работа метода построена на определении признаков каждого из экземпляров, которые являются параметрами модели, соответствующей синтаксису текущего представления программы. Так, например, если представлением является исходный код, а экземпляры имеют текстовую запись программы на соответствующем языке программирования, то параметрами может быть топология дерева абстрактного синтаксиса и свойства его узлов; в ряде случаев, можно использовать более простые модели, такие как списки символьных строк [21]. Затем, используя такие модельные представления экземпляров программ, производится вычисление метрики их близости, целесообразность чего обосновывается использованием единого базиса сравнения (т. е. формальной записью синтаксиса).

Псевдокод алгоритма для метода вычисления метрики близости (ProximityMetric) в Python-подобном стиле приведен ниже; помимо типовых, используются следующие языковые конструкции: «**List<>**» – список элементов, «**double**» – тип для чисел с плавающей точкой.

```
   Name: ProximityMetric()
   Input:
      Syntax - синтаксис представления программы
      Rep1 - представление первой программы
      Rep2 - представление второй программы
      Params - параметры определения метрики близости
   Output:
      Metric - метрика близости двух программ (в диапазоне [0, 1])
   Begin
         // Шаг 1. Получение внутреннего представления
    1:   parser = Params.Parser(Syntax);
    2:   features1 = parser.Run(Rep1);
    3:   features2 = parser.Run(Rep2);

         // Шаг 2. Нормализация внутреннего представления
    4:   normal = Params.Normalizer(Syntax);
    5:   features_normal1 = normal(features1);
    6:   features_normal2 = normal(features2);

         // Шаг 3. Вычисление частных метрик
    7:   List<double> metrics;
    8:   For comparator In Params.Comparators:
    9:      metric = comparator.Eval(features_normal1, features_normal2);
   10:      metrics += metric;
   11:   End For

         // Шаг 4. Вычисление интегральной метрики
   12:   integrator = Params.Integrator();
   13:   Metric = integrator.Combine(metrics);

   14:   Return Metric
   End
```

Метод на вход принимает синтаксис представления программы, два ее представления и параметры; а на выходе возвращает метрику близости этих программ.

Согласно псевдокоду алгоритма ProximityMetric(), он состоит из 4 следующих шагов: 1) экземпляры программ на основе формального синтаксиса преобразуются во внутреннее представление, например, дерево абстрактного синтаксиса [22]; 2) производится нормализация внутреннего представления приведением их к более обобщенному виду, например, сортировкой операндов коммутативных операций или более сложным образом [23]; 3) вычисляются различные частные метрики близости программ, например, с использованием алгоритмов схожести графов [24]; 4) вычисленные частные метрики интегрируются в единую, множество значений которой находится в отрезке от 0 до 1 и которая считается результатом работы метода.

Второй метод Группы скрещивания предназначен для получения дочернего экземпляра программы по хромосомам двух родительских особей-программ в текущей популяции [25]. Метод анализирует два родительских пути и находит в них позиции, которые ведут на один узел-альтернативу в ГСП. Затем, случайным образом выбирается одна из таких позиций, относительно которой происходит взаимный «обмен» частями хромосом родительских узлов для их поддеревьев в ГСП, с получением пары дочерних. В результате, начальная (до общей позиции и поддеревьев ГСП) и конечная части (после поддеревьев ГСП) хромосом дочерних узлов совпадают с аналогичными частями 1-го и 2-го родителя соответственно, а средняя часть хромосом (после общей позиции и в рамках поддерева ГСП) – с противоположными родителями, т.е. с 2-м и 1-м соответственно. Например, если согласно хромосомам родителей их пути

по ГСП проходят через узлы [1 2 9 3 4] и [5 6 9 7 8] (т. е. в которых общий узел 9), то после скрещивания могут получиться две дочерних особи, пути которых проходят через следующие узлы ГСП – [1 2 9 7 8] и [5 6 9 3 4].

Псевдокод алгоритма для метода скрещивания хромосом (ChromosomeCrossover) в Python-подобном стиле приведен ниже; языковые конструкции, указанные для алгоритма ProximityMetric(), дополнены следующими: «**Tuple<>**» – кортеж элементов, «**integer**» – целочисленный тип, «**var_list[p1 : p2]**» – часть списка «**var_list**» с $p1$-го по $p2$-ой элементы (второй не включительно).

```
   Name: ChromosomeCrossover()
   Input:
      Syntax - синтаксис представления программы
      Chromosome1 - хромосома первой особи
      Chromosome2 - хромосома второй особи
      Params - параметры скрещивания хромосом
   Output:
      Chromosome_C1 - хромосома после скрещивания первой особи
      Chromosome_C2 - хромосома после скрещивания второй особи
   Begin:

         // Шаг 1. Инициализация
    1:   random = Random();
    2:   List<Tuple<integer, integer>> points;
    3:   List<Tuple<ChromosomeType, ChromosomeType>> crosses;

         // Шаг 2. Обход генов первой хромосомы
    4:   For i1 In Chromosome1.Gens.Length:
    5:      gen1 = Chromosome1.Gens[i1];

         // Шаг 2.1. Обход генов второй хромосомы
    6:   For i2 In Chromosome2.Gens.Length:
    7:      gen2 = Chromosome2.Gens[i1];

         // Шаг 2.1.1. Поиск генов, связанных с одинаковыми узлами-альтернативами
    8:      If gen1.NodeId == gen2.NodeId Then:

            // Шаг 2.1.2. Получение частей хромосом для обмена между особями
    9:         cross_1 = Chromosome2.Gens[i1 : i1 + gen1.SubTreeLen];
   10:         cross_2 = Chromosome2.Gens[i2 : i2 + gen2.SubTreeLen];

            // Шаг 2.1.3. Проверка отличности частей хромосом
   11:         If cross_1 != cross_2 Then:
            // Шаг 2.1.4. Выбор и сохранение частей хромосом
   12:         If Params.CrossoverRate >= random.float(0, 1) Then
   13:            points += [ i1, i2 ];
   14:            crosses += [ cross_1, cross_2 ];
   15:               End If
   16:            End If
   17:         End If
   18:      End If
   19:   End If

         // Шаг 3. Проверка наличия подходящих хромосом для обмена генами
   20:   If points.Length == 0 Then:
   21:      Return None;
   22:   End If

         // Шаг 4. Выбор части хромосом для обмена генами
   23:   j = random.int(0, points.Length);
   24:   point = points[j];
   25:   cross = crosses[j];

         // Шаг 5. Конструирование дочерних хромосом из родительских
   26:   Chromosome_C1.Gens = Chromosome1.Gens[0 : point.Element1] + cross.Element2 + Chromosome1.Gens[point.Element1 + cross.Element2.Length : Chromosome1.Gens.Length];
   27:   Chromosome_C2.Gens = Chromosome2.Gens[0 : point.Element2] + cross.Element1 + Chromosome2.Gens[point.Element2 + cross.Element1.Length : Chromosome2.Gens.Length];

   28:   Return Chromosome_C1, Chromosome_C2;
   End
```

Метод на вход принимает синтаксис представления программы, хромосомы двух особей и параметры; а на выходе возвращает хромосомы двух дочерних (сгенерированных) особей.

Согласно псевдокоду алгоритма ChromosomeCrossover(), он состоит из 5 следующих шагов: 1) инициализация генератора случайных чисел, создание вспомогательных хранилищ точек пересечения хромосом и самих частей хромосом; 2) обход генов обоих хромосом с целью определения их точек пересечения для обмена при скрещивании; 3) проверка наличия точек пересечения и в ином случае возврат негативного результата скрещивания; 4) случайный выбор частей хромосом для обмена между особями; 5) конструирование пары новых дочерних особей, составленных из начальных и конечных частей хромосом своих родителей с обменом средними частями. При этом, шаг 2 представляет собой два вложенных цикла по генам хромосом особей, в теле последнего из которых выполняются следующие вложенные шаги: 1) поиск генов двух хромосом, соответствующих одному узлу-альтернативе; 2) сохранение точек пересечения и частей хромосом особей, соответствующих поддеревьям ГСП для найденных узлов-альтернатив; 3) обеспечение отличности полученных частей-хромосом для обеих особей (в ином случае, скрещивание не будет иметь эффекта); 4) выбор частей хромосом для скрещивания, используя

случайно сгенерированное число (на отрезке [0,1]) и частоту скрещивания.

Третий метод Группы предназначен для мутации хромосомы отдельной особи [26]. Метод случайным образом (или согласно некоторой логике) меняет гены хромосомы, что приводит к изменению выбора дочерних веток в ГСП в узлах-альтернативах. Так, замена элемента пути на другой случайный, в котором содержатся другие альтернативы, перестроит всю логику обхода ГСП, потенциально увеличив или уменьшив длину хромосомы особи. Естественно, такое произвольное изменение ген может привести к необходимости выбора значений для новых альтернатив, для чего в методе присутствует специальная операция добавления корректного (случайного) окончания пути. Например, если изначально альтернатива вела к сложению с числом, а после мутации – к сложению с функцией, выражение для которой содержит множество пока еще не выбранных альтернатив (например, параметров), то окончание хромосомы будет перегенерировано, а ее длина увеличится.

Псевдокод алгоритма (ChromosomeMutation) для метода мутации хромосомы приведен ниже.

```
Name: ChromosomeMutation()
Input:
   Syntax - синтакс представления программы
   Chromosome - хромосома особи
   Params - параметры мутации хромосом
Output:
   Chromosome_M - хромосома смутировавшей особи
Begin
      // Шаг 1. Инициализация
 1:   random = Random();

      // Шаг 2. Обход генов хросомосы
 2:   For i In Chromosome.Gens.Length:
 3:      gen = Chromosome.Gens[i];

      // Шаг 2.1. Выбор гена для мутации
 4:      If Params.MutationRate >= random.
float(0, 1) Then:

      // Шаг 2.2. Выбор нового значения гена
 5:      j = random.int(0, gen.MaxValue - 1):

      // Шаг 2.3. Проверка отличности нового
гена от имеющегося
 6:      If j >= gen.Value Then:
 7:         j += 1;
 8:      End If

      // Шаг 2.4. Обновление значение гена
 9:      Chromosome.Gens[i].Value = j;
10:      End If

      // Шаг 2.5. Построение корректного пути
по ГСП с учетом нового гена
11:      ChromosomeM = Syntax.CompletePathFrom-
Gen(Chromosome, I, Param);

12:   Return ChromosomeM
End
```

Метод на вход принимает синтаксис представления программы, хромосому и параметры; а на выходе возвращает мутированную хромосому.

Согласно псевдокоду алгоритма ChromosomeMutation(), он состоит из 2 следующих шагов, на которых производится инициализация генератора случайных чисел и обход генов хромосомы для их мутации. Также, второй шаг поделен на следующие вложенные шаги: 1) выбор гена для мутации, используя случайно сгенерированное число (на отрезке [0,1]) и заданную частоту мутации; 2) выбор нового мутированного значения гена, используя случайно сгенерированное число в диапазоне 0 до максимально возможного значения гена (определяемого количеством возможных путей из узла-альтернативы), уменьшенного на 1; 3) обеспечение отличности нового значения гена от имеющегося в хромосоме путем увеличения его значения на 1 в случае равенства или превышения имеющегося значения (для этого было уменьшение диапазона случайных чисел на 1 на предыдущем шаге); 4) обновление имеющегося значения гена хромосомы тем, которое было выбрано случайно; 5) построение корректного окончания генов хромосомы, начиная с мутировавшего, поскольку сделанное таким образом изменение пути по ГСП, как правило, влияет на последующие узлы-альтернативы.

Заключение

В работе описывается авторский подход генетической деэволюции, основным применением которого является получение представлений программы для обнаружения и устранения в них уязвимостей. Подход основан на решении оптимизационной задачи подбора синтаксических конструкций, которые бы соответствовали программе в предыдущем представлении, тождественной имеющейся программе в текущем.

Основным результатом проведенного исследования является комплекс методов генетической деэволюции, состоящих из общего метода проведения реверс-инжиниринга программы, использующего метод последовательной деэволюции соседних представлений программы, который в свою очередь основан на применении генетических алгоритмов с помощью операций, реализованных в виде метода вычисления метрики близости программ в одном представлении, а также методов скрещивания и мутации хромосом их особей.

Значимость полученных результатов заключается как в теоретической интеллектуализации процесса реверс-инжиниринга программ, так и в практическом улучшении результативности нейтрализации уязвимостей программы при неухудшении оперативности и ресурсоэкономности процесса.

Исходя из того, что все методы имеют реализацию в виде программного прототипа, продолжением работы должно стать проведение серии экспериментов, как для подтверждения их работоспособности, так и для оценок границ применимости.

Литература

1. Абдуллин Т. И., Баев В. Д., Буйневич М. В. и др. Цифровые технологии и проблемы информационной безопасности / Санкт-Петербург: Санкт-Петербургский государственный экономический университет, 2021. 163 с.
2. Шимчик Н. В., Игнатьев В. Н., Белеванцев А. А. IRBIS: Статический анализатор помеченных данных для поиска уязвимостей в программах на C/C++ // Труды Института системного программирования РАН. 2022. Т. 34. № 6. С. 51–66. DOI: 10.15514/ISPRAS-2022-34(6)-4.
3. David A. Ghidra Software Reverse Engineering for Beginners: Analyze, identify, and avoid malicious code and potential threats in your networks and systems. UK: Packt Publishing Ltd, 2021. 322 p.
4. Жилин В. В., Сафарьян О. А. Искусственный интеллект в системах хранения данных // Вестник Донского государственного технического университета. 2020. Т. 20. № 2. С. 196–200. DOI: 10.23947/1992-5980-2020-20-2-196-200.
5. Artuso F. Deep Learning Based Binary Code Analysis: Ph.D. Program in Engineering in Computer Science / Sapienza University of Rome, 2025. 155 p.
6. Armengol-Estape J., Woodruff J., Cummins C., O'Boyle M. F. SLaDe: A Portable Small Language Model Decompiler for Optimized Assembly // The proceedings of IEEE/ACM International Symposium on Code Generation and Optimization (Edinburgh, United Kingdom, 2–6 March 2024). 2024. PP. 67–80.
7. Tan H., Luo Q., Li J., Zhang Y. LLM4Decompile: Decompiling Binary Code with Large Language Models // The proceedings of Conference on Empirical Methods in Natural Language Processing (USA, Miami, Florida, 12–16 November 2024). 2024. PP. 3473–3487.
8. Zhang X., Xu Z., Yang S., Li Z., Shi Z., Sun L. Enhancing Function Name Prediction using Votes-Based Name Tokenization and Multi-task Learning // The proceedings of ACM on Software Engineering. Vol. 1. No. 75. PP. 1679–1702.
9. He J., Ivanov P., Tsankov P., Raychev V., Vechev M. Debin: Predicting Debug Information in Stripped Binaries // The proceedings of ACM SIGSAC Conference on Computer and Communications Security (Canada, Toronto, 15–19 October 2018). 2018. P. 1667–1680.
10. Shin E. C. R., Song D., Moazzezi R. Recognizing functions in binaries with neural networks // The proceedings of 24th USENIX Conference on Security Symposium (USA, Washington, D.C., 2015 August 12–14). 2015. PP. 611–626.
11. Израилов К. Е. Концепция генетической деэволюции представлений программы. Часть 1 // Вопросы кибербезопасности. 2024. № 1(59). С. 61–66. DOI: 10.21681/2311-3456-2024-1-61-66.
12. Израилов К. Е. Концепция генетической деэволюции представлений программы. Часть 2 // Вопросы кибербезопасности. 2024. № 2(60). С. 81–86. DOI: 10.21681/2311-3456-2024-2-81-86.
13. Силенко Д. И., Лебедев И. Г. Алгоритм глобальной оптимизации, использующий деревья решений для выявления локальных экстремумов // Проблемы информатики. 2023. № 2(59). С. 21–33. DOI: 10.24412/2073-0667-2023-2-21-33.
14. Пикалов М. В., Письмеров А. М. Настройка параметров генетического алгоритма при помощи анализа ландшафта функции приспособленности и машинного обучения // Известия ЮФУ. Технические науки. 2024. № 2(238). С. 221–228. DOI: 10.18522/2311-3103-2024-2-221-228.
15. Петросов Д. А. Анализ и выбор методов представления характеристик состояния популяции генетического алгоритма // Оригинальные исследования. 2023. Т. 13. № 10. С. 235–239.
16. Безгачев Ф. В., Галушин П. В., Рудакова Е. Н. Эффективная реализация инициализации и мутации в генетическом алгоритме псевдо-булевой оптимизации // E-Scio. 2020. № 4(43). С. 224–231.
17. Pan Z., Yan Y., Yu L., Wang T. Identification of binary file compilation information // Proceedings of the IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (Chongqing, China, 16–18 December 2022). 2022. PP. 1141–1150. DOI: 10.1109/IMCEC55388.2022.10019958.
18. Израилов К. Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 1. Схема жизненного цикла // Труды учебных заведений связи. 2023. Т. 9. № 1. С. 75–93. DOI: 10.31854/1813-324X-2023-9-1-75-93.
19. Израилов К. Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 2. Аналитическая модель и эксперимент // Труды учебных заведений связи. 2023. Т. 9. № 2. С. 95–111. DOI: 10.31854/1813-324X-2023-9-2-95-111.
20. Цыганков В. А., Шабалина О. А., Катаев А. В. Исследование воздействия размера популяции на быстродействие генетического алгоритма // Известия ЮФУ. Технические науки. 2024. № 3(239). С. 168–176. DOI: 10.18522/2311-3103-2024-3-168-176.
21. Буйневич М. В., Израилов К. Е. Авторская метрика оценки близости программ: приложение для поиска уязвимостей с помощью генетической деэволюции // Программные продукты и системы. 2025. Т. 38. № 1. С. 89–99. DOI: 10.15827/0236-235X.149.089-099.
22. Грибков Н. А., Овасапян Т. Д., Москвин Д. А. Анализ восстановленного программного кода с использованием абстрактных синтаксических деревьев // Проблемы информационной безопасности. Компьютерные системы. 2023. № 2(54). С. 47–60. DOI: 10.48612/jisp/ruar-u6he-kmd4.
23. Allamanis M., Brockschmidt M., Khademi M. Learning to Represent Programs with Graphs // In proceedings of the 6th International Conference on Learning Representations (Vancouver, Canada, 20 April–3 May 2018). 2018. PP. 1–17. DOI: 10.48550/arXiv.1711.00740.
24. Ормонова Э. М. Определение качества программного продукта на основе теории графов // Наука. Образование. Техника. 2021. № 1(70). С. 37–44.
25. Тотухов К. Е., Романов А. Ю., Лукьянов В. И. Исследование эффективности работы генетических алгоритмов с различными методами скрещивания и отбора // Электронный сетевой политематический журнал «Научные труды КубГТУ». 2022. № 6. С. 98–109.
26. Доманов К. И. Сравнительный анализ эффективности работы генетического алгоритма при модификации оператора мутации в задаче коммивояжера // Политехнический молодежный журнал. 2022. № 1(66). DOI: 10.18698/2541-8009-2022-1-760.

A COMPLEX OF METHODS FOR GENETIC DE-EVOLUTION OF PROGRAM REPRESENTATIONS

Izrailov K. E.[2]

Keywords: *vulnerability neutralization, reverse engineering, artificial intelligence, genetic algorithms, complex of methods.*

The goal of the research: *increasing the efficiency of neutralizing program vulnerabilities by intellectualizing its reverse engineering using genetic algorithms*

Research methods: *system analysis and optimization methods, graph theory, functional and structural synthesis, general programming methodology and compiler theory.*

Results: *a hierarchical three-level set of methods was synthesized, consisting of a genetic reverse-engineering program method, a genetic de-evolution method of its neighboring representations (machine and source code, algorithms, architecture, etc.), and a group of methods for implementing the genetic algorithms fundamental operations.*

The scientific novelty *of the complex methods lies in their focus on solving the reverse engineering problem by direct transformations of the program into subsequent representations, in contrast to classical ones that perform inverse transformations. Also, the algorithms of the methods group of the complex are based on working with the original source code model, representing it as a genes sequence.*

References

1. Abdullin T. I., Baev V. D., Bujnevich M. V. i dr. Cifrovye tehnologii i problemy informacionnoj bezopasnosti / Sankt-Peterburg: Sankt-Peterburgskij gosudarstvennyj jekonomicheskij universitet, 2021. 163 s.
2. Shimchik N. V., Ignat'ev V. N., Belevancev A. A. IRBIS: Staticheskij analizator pomechennyh dannyh dlja poiska ujazvimostej v programmah na C/C++ // Trudy Instituta sistemnogo programmirovanija RAN. 2022. T. 34. № 6. S. 51–66. DOI: 10.15514/ISPRAS-2022-34(6)-4.
3. David A. Ghidra Software Reverse Engineering for Beginners: Analyze, identify, and avoid malicious code and potential threats in your networks and systems. UK: Packt Publishing Ltd, 2021. 322 p.
4. Zhilin V. V., Safar'jan O. A. Iskusstvennyj intellekt v sistemah hranenija dannyh // Vestnik Donskogo gosudarstvennogo tehnicheskogo universiteta. 2020. T. 20. № 2. S. 196–200. DOI: 10.23947/1992-5980-2020-20-2-196-200.
5. Artuso F. Deep Learning Based Binary Code Analysis: Ph.D. Program in Engineering in Computer Science / Sapienza University of Rome, 2025. 155 p.
6. Armengol-Estape J., Woodruff J., Cummins C., O'Boyle M. F. SLaDe: A Portable Small Language Model Decompiler for Optimized Assembly // The proceedings of IEEE/ACM International Symposium on Code Generation and Optimization (Edinburgh, United Kingdom, 2–6 March 2024). 2024. PP. 67–80.
7. Tan H., Luo Q., Li J., Zhang Y. LLM4Decompile: Decompiling Binary Code with Large Language Models // The proceedings of Conference on Empirical Methods in Natural Language Processing (USA, Miami, Florida, 12–16 November 2024). 2024. PP. 3473–3487.
8. Zhang X., Xu Z., Yang S., Li Z., Shi Z., Sun L. Enhancing Function Name Prediction using Votes-Based Name Tokenization and Multi-task Learning // The proceedings of ACM on Software Engineering. Vol. 1. No. 75. PP. 1679–1702.
9. He J., Ivanov P., Tsankov P., Raychev V., Vechev M. Debin: Predicting Debug Information in Stripped Binaries // The proceedings of ACM SIGSAC Conference on Computer and Communications Security (Canada, Toronto, 15–19 October 2018). 2018. P. 1667–1680.
10. Shin E. C. R., Song D., Moazzezi R. Recognizing functions in binaries with neural networks // The proceedings of 24th USENIX Conference on Security Symposium (USA, Washington, D.C., 2015 August 12–14). 2015. PP. 611–626.
11. Izrailov K. E. Koncepcija geneticheskoj dejevoljucii predstavlenij programmy. Chast' 1 // Voprosy kiberbezopasnosti. 2024. № 1(59). S. 61–66. DOI: 10.21681/2311-3456-2024-1-61-66.
12. Izrailov K. E. Koncepcija geneticheskoj dejevoljucii predstavlenij programmy. Chast' 2 // Voprosy kiberbezopasnosti. 2024. № 2(60). S. 81–86. DOI: 10.21681/2311-3456-2024-2-81-86.
13. Silenko D. I., Lebedev I. G. Algoritm global'noj optimizacii, ispol'zujushhij derev'ja reshenij dlja vyjavlenija lokal'nyh jekstremumov // Problemy informatiki. 2023. № 2(59). S. 21–33. DOI: 10.24412/2073-0667-2023-2-21-33.
14. Pikalov M. V., Pis'merov A. M. Nastrojka parametrov geneticheskogo algoritma pri pomoshhi analiza landshafta funkcii prisposoblennosti i mashinnogo obuchenija // Izvestija JuFU. Tehnicheskie nauki. 2024. № 2(238). S. 221–228. DOI: 10.18522/2311-3103-2024-2-221-228.
15. Petrosov D. A. Analiz i vybor metodov predstavlenija harakteristik sostojanija populjacii geneticheskogo algoritma // Original'nye issledovanija. 2023. T. 13. № 10. S. 235–239.
16. Bezgachev F. V., Galushin P. V., Rudakova E. N. Jeffektivnaja realizacija inicializacii i mutacii v geneticheskom algoritme psevdo-bulevoj optimizacii // E-Scio. 2020. № 4(43). S. 224–231.
17. Pan Z., Yan Y., Yu L., Wang T. Identification of binary file compilation information // Proceedings of the IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (Chongqing, China, 16–18 December 2022). 2022. PP. 1141–1150. DOI: 10.1109/IMCEC55388.2022.10019958.
18. Izrailov K. E. Modelirovanie programmy s ujazvimostjami s pozicii jevoljucii ee predstavlenij. Chast' 1. Shema zhiznennogo cikla // Trudy uchebnyh zavedenij svjazi. 2023. T. 9. № 1. S. 75–93. DOI: 10.31854/1813-324X-2023-9-1-75-93.

2 Konstantin E. Izrailov, Ph.D., Docent, Professor of the Department of Applied Mathematics and Information Technologies Security of the Saint-Petersburg University of State Fire Service of EMERCOM of Russia, Saint-Petersburg. ORCHID: http://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56123238800. E-mail:konstantin.izrailov@mail.ru

19. Izrailov K. E. Modelirovanie programmy s ujazvimostjami s pozicii jevoljucii ee predstavlenij. Chast' 2. Analiticheskaja model' i jeksperiment // Trudy uchebnyh zavedenij svjazi. 2023. T. 9. № 2. S. 95–111. DOI: 10.31854/1813-324X-2023-9-2-95-111.
20. Cygankov V. A., Shabalina O. A., Kataev A. V. Issledovanie vozdejstvija razmera populjacii na bystrodejstvie geneticheskogo algoritma // Izvestija JuFU. Tehnicheskie nauki. 2024. № 3(239). S. 168–176. DOI: 10.18522/2311-3103-2024-3-168-176.
21. Bujnevich M. V., Izrailov K. E. Avtorskaja metrika ocenki blizosti programm: prilozhenie dlja poiska ujazvimostej s pomoshh'ju geneticheskoj dejevoljucii // Programmnye produkty i sistemy. 2025. T. 38. № 1. S. 89–99. DOI: 10.15827/0236-235X.149.089-099.
22. Gribkov N. A., Ovasapjan T. D., Moskvin D. A. Analiz vosstanovlennogo programmnogo koda s ispol'zovaniem abstraktnyh sintaksicheskih derev'ev // Problemy informacionnoj bezopasnosti. Komp'juternye sistemy. 2023. № 2(54). S. 47–60. DOI: 10.48612/jisp/ruar-u6he-kmd4.
23. Allamanis M., Brockschmidt M., Khademi M. Learning to Represent Programs with Graphs // In proceedings of the 6th International Conference on Learning Representations (Vancouver, Canada, 20 April–3 May 2018). 2018. PP. 1–17. DOI: 10.48550/arXiv.1711.00740.
24. Ormonova Je. M. Opredelenie kachestva programmnogo produkta na osnove teorii grafov // Nauka. Obrazovanie. Tehnika. 2021. № 1(70). S. 37–44.
25. Totuhov K. E., Romanov A. Ju., Luk'janov V. I. Issledovanie jeffektivnosti raboty geneticheskih algoritmov s razlichnymi metodami skreshhivanija i otbora // Jelektronnyj setevoj politematicheskij zhurnal «Nauchnye trudy KubGTU». 2022. № 6. S. 98–109.
26. Domanov K. I. Sravnitel'nyj analiz jeffektivnosti raboty geneticheskogo algoritma pri modifikacii operatora mutacii v zadache kommivojazhera // Politehnicheskij molodezhnyj zhurnal. 2022. № 1(66). DOI: 10.18698/2541-8009-2022-1-760.