КОНЦЕПЦИЯ ГЕНЕТИЧЕСКОЙ ДЕЭВОЛЮЦИИ ПРЕДСТАВЛЕНИЙ ПРОГРАММЫ. Часть 2

Израилов К. Е.[1]

DOI: 10.21681/2311-3456-2024-2-81-86

Цель исследования: *развитие направления реверс-инжиниринга программ, заключающегося в преобразовании их представлений в одно из предыдущих.*

Методы исследования: *системный анализ, мысленный эксперимент, аналитическое моделирование, многокритериальная оптимизация.*

Полученные результаты: *предложена концепция генетической деэволюции представлений программы, предлагающая процесс их восстановления не обратным способом, т.е. от текущего к предыдущему, а прямым – работая с псевдо-предыдущим представлением и оценивая его близость к исследуемому текущему; принцип концепции основан на решении оптимизационной задачи с помощью генетических алгоритмов.*

В первой части статьи [1] было введена онтологическая модель предметной области, в терминах которой предложена высокоуровневая схема (де)эволюции представлений, отражающая преобразования между ними, а также внесение и обнаружение уязвимостей; дано формализованное описание процессов на схеме.

Во второй части статьи предложена низкоуровневая схема генетической деэволюции, описывающая процесс реверс-инжиниринга представлений на базе генетических алгоритмов; дано формализованное описание процессов на схеме, а также шагов деэволюции.

Научная новизна *заключается в качественно новой точке зрения на восстановление представлений – с помощью процесса итеративного подбора предыдущего для соответствия (после эволюции) текущему, при этом, основанного на принципах генетических алгоритмов, а также имеющего полностью формализованный вид.*

Ключевые слова: *концепция, эволюция, реверс-инжиниринг, реинжиниринг, обратная разработка, обратный инжиниринг, генетический алгоритм, уязвимость.*

THE GENETIC DE-EVOLUTION CONCEPT OF PROGRAM REPRESENTATIONS. Part 2

Izrailov K. E.[2]

The goal of the investigation: *development of the programs reverse engineering direction, which consists in transforming their state into one of the previous.*

Research methods: *system analysis, mental experiment, analytical modeling, multicriteria optimization.*

Result: *the genetic de-evolution concept of program representations has been proposed, suggesting a process for their restoration in a backway, i.e. from the current to the previous one, and direct way – working with the pseudo-previous representation and assessing its proximity to the current one being studied; the concept principle is based on solving an optimization problem using genetic algorithms.*

In the first part of the article [1], an ontological model of the subject area was introduced, in terms of which a high-level scheme of (de)evolution of representations was proposed, reflecting the transformations between them, as well as the introduction and detection of vulnerabilities; A formalized description of the processes on the diagram is given.

1 Израилов Константин Евгеньевич, кандидат технических наук, доцент, старший научный сотрудник лаборатории проблем компьютерной безопасности Санкт-Петербургского Федерального исследовательского центра Российской академии наук, Санкт-Петербург, Россия. ORCHID: http://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56122749800. E-mail: konstantin.izrailov@mail.ru

2 Konstantin E. Izrailov, Ph.D, assistant Professor, Senior Researcher of Laboratory of Computer Security Problems of St. Petersburg Federal Research Center of the Russian Academy of Sciences, Saint-Petersburg, Rusia. ORCHID: http://orcid.org/0000-0002-9412-5693. Scopus Author ID: 56123238800. E-mail: konstantin.izrailov@mail.ru

In the second part of the article, a low-level scheme of genetic de-evolution is proposed, which describes the reverse engineering process of representations based on genetic algorithms; a processes formalized description in the diagram, as well as the steps of de-evolution, is given.

The scientific novelty consists in a qualitatively new point of view on the representation's restoration – using the iterative process selection of the previous one to correspond (after evolution) to the current one, at the same time, based on the principles of genetic algorithms, and also having a completely formalized form.

Keywords: concept, evolution, reverse engineering, reengineering, backward engineering, genetic algorithm, vulnerability

Схемы генетической деэволюции

Детализация схемы (де)эволюции программы (см. рис. 2 в [1]) в части получения предыдущего представления из текущего представлена в виде вложенной схемы генетической деэволюции (см. рис. 1).

Дадим ряд необходимых пояснений к схеме деэволюции представлений (см. рис. 1), описав в начале суть генетических алгоритмов. Следуя из названия, генетический алгоритм «позаимствован» у природы и построен на следующих принципах ее эволюции: создание популяции из множества особей, характеризуемых хромосомами, состоящими из последовательности ген (каждый из которых отвечает за некоторую, возможно не очевидную особенность особи); естественный отбор особей, наиболее приспособленных к жизни в среде; скрещивание таких «лучших» представителей популяции путем «перемешивания» их ген; внесение незначительных мутаций в гены [2]; итеративное повторение процесса. Важнейшей операцией в алгоритме является определение приспособленности особи, для чего предназначена соответствующая функция (вычисление ее значения и позволяет делать такой отбор). Как результат, генетические алгоритмы позволяют решать оптимизационные задачи [3]; впрочем, являясь эвристическими, они гарантируют лишь нахождение экстремума (т.е. минимума или максимума), который не обязан быть глобальным [4]. Сам же реверс-инжиниринг (сокр. реинжиниринг, далее – РИ) может быть представлен не в классическом смысле – как анализ конструкций экземпляра

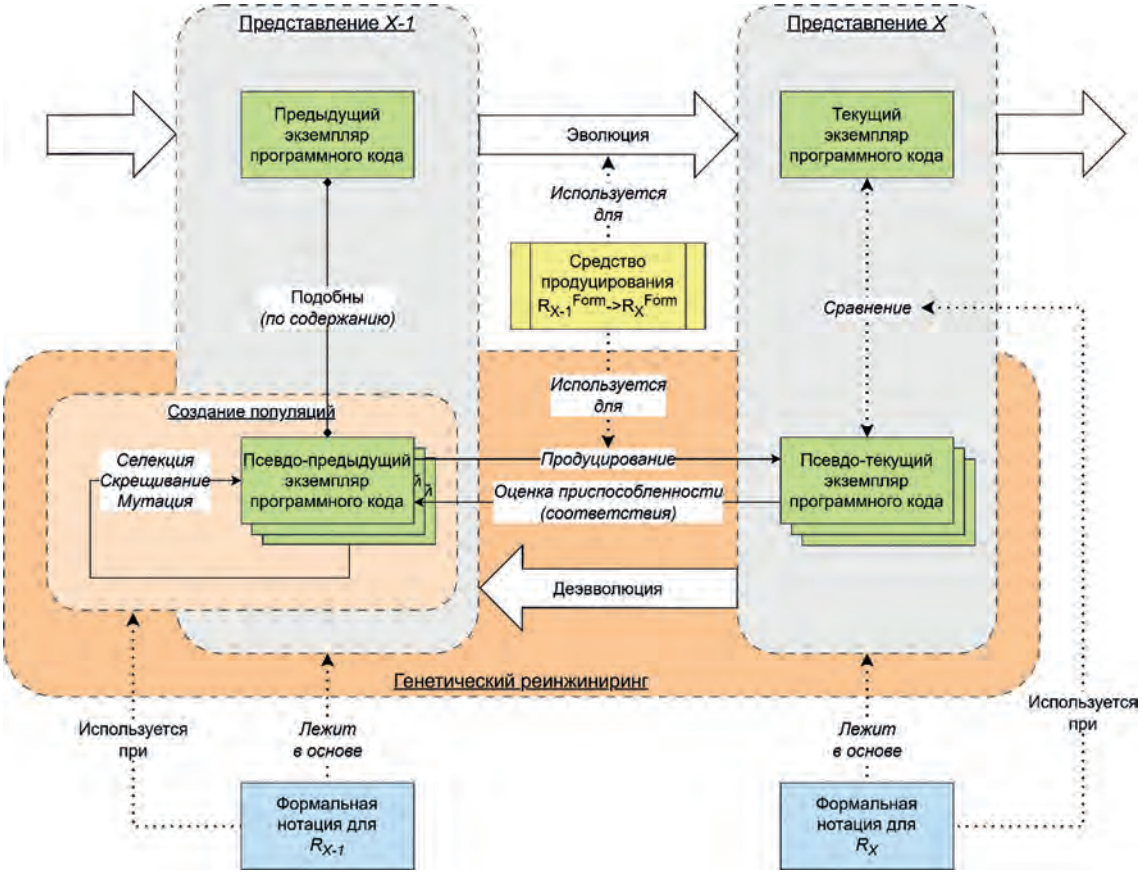

Рис. 1. Схема генетической деэволюции представлений программы

программного кода (далее – ЭПК) в текущем представлении (например, машинного кода) с целью сопоставления ему аналогичных конструкций ЭПК в предыдущем представлении (например, исходного кода), а в новом, «генетическом» – как итеративный подбор или оптимизация ЭПК в предыдущем представлении (т.е. искомого), который бы в точности преобразовывался в имеющийся ЭПК в текущем представлении [5, 6]. Таким образом, суть генетического РИ заключается в создании множества (т.е. популяции) псевдо-предыдущих ЭПК (в представлении X–1) при учете его формальной нотации (т.е. для R_{X-1}). Затем, из таких ЭПК создаются подобные им псевдо-текущие ЭПК (в представлении X) путем применения средства продуцирования, преобразующего формы: с R_{X-1}^{Form} в R_X^{Form}. Для полученных подобным образом ЭПК оценивается их близость с имеющимся текущим ЭПК (также с использованием формальной нотации для т.е. для R_X). По результатам оценки селектируются (т.е. отбираются) наиболее приспособленные псевдо-предыдущие ЭПК, осуществляется их скрещивание друг с другом и мутация, в результате чего создается новая популяция такого же размера. Затем процесс повторяется, пока не будет найден псевдо-предыдущий ЭПК, в точности преобразуемый в имеющийся ЭПК в представлении X. Функция же приспособленности должна определять «близость» псевдо-текущего ЭПК, полученного из каждого варианта псевдо-предыдущего ЭПК, с имеющимся ЭПК в текущем представлении; максимальное значение данной функции и будет сигналом к выполнению задачи РИ. Отметим, что в данном случае термин «экземпляр» введен специально, чтобы указать на многообразие вариаций программного кода программы в некотором представлении, а термин «псевдо» говорит о том, что каждая вариация ЭПК не обязана быть тождественна текущей программе, подвергаемой (ре)инжинирингу.

Такой итеративный процесс генетического РИ, приведенный выше, можно описать следующей последовательностью шагов (включающей их формальную часть).

Шаг 1. Создание множества псевдо-предыдущих экземпляров программного кода

Первоначально создается разнородное множество (т.е. популяция) псевдо-предыдущих ЭПК – т.е. их вариация в некотором представлении; например, случайно сгенерированных, но синтаксически корректных, исходных кодов С-программы. Такие ЭПК не являются истинно предыдущим, а только стремятся им стать в процессе РИ. Корректность и вариативность же генерации множества ЭПК обеспечивается учетом формальной нотации, принятой в данном представлении [7]; например, за счет формализации лексических токенов и синтаксиса языка программирования С.

Шаг ($Step_1$) имеет следующую формальную запись:

$$Step_1 : \{e_i^{x-1} : i \in \mathbb{N},\ i \le N\} \equiv \{e_i^{x-1}\}_N = Random(N, FN_{x-1}),$$

где e_i^{x-1} – i-й псевдо-предыдущий ЭПК из множества в x-ом представлении; $\mathbb{N}$ – множество натуральных чисел; N – число ЭПК в множестве (то есть общее число особей в популяции); N в нижнем индексе – размер множества; $Random(N, FN_{x-1})$ – функция генерации случайных ЭПК числом N и в соответствии с формальной нотацией FN_{x-1} (аббр. от англ. Format Notation) для x–1-го представления. Соответственно, таким же образом можно сгенерировать множество ЭПК для x–2-го представления и т.д.

Шаг 2. Продуцирование множества псевдо-текущих экземпляров программного кода

Для каждого псевдо-предыдущего ЭПК происходит продуцирование соответствующего ему псевдо-текущего ЭПК, для чего используется стандартное средство продуцирования, точно такое, как и при получении представлений в рамках обычного инжиниринга; например, классический компилятор для языка С со встроенным ассемблером (для получения сразу машинного кода в обход ассемблерного) [8].

Шаг (Step_2) имеет следующую формальную запись:

$$Step_2 : e_i^x = Production(e_i^{x-1}, U_{x-1}),$$

где $Production(e_i^{x-1}, U_{x-1})$ – функция получения псевдо-текущего (т. е. i-го) ЭПК из предыдущего (т.е. x–1-го) с помощью средства продуцирования U_{x-1}, ориентированного на обработку x–1-го представления. Соответственно, таким же образом можно получить множество ЭПК для x–1-го представления из x–2-го.

Шаг 3. Сравнение псевдо-текущих экземпляров программного кода с исследуемым текущим

Вычисляется функция приспособленности для каждого псевдо-текущего ЭПК, полученного на Шаге 2. Данная функция определяет близость ЭПК к тому, РИ для которого необходимо произвести [9, 10]. Соответственно, если совпадение будет точным, то это означает, что его псевдо-предыдущий ЭПК и есть результат деэволюции. Так, например, если текущий машинный код состоит из сложения аргументов подпрограммы, а псевдо-текущий – из множества циклов и условных переходов, то, очевидно, значение функции будет малым; и, наоборот, для сгенерированного псевдо-текущего машинного кода из набора сложений и вычитаний аргументов подпрограммы, его фитнес функция примет большие значения. Соответственно, если функция приспособленности вернет максимальное значение, то это

будет означать обнаружение искомого предыдущего ЭПК, который в точности преобразуется в текущий.

Шаг ($Step_3$) имеет следующую формальную запись:

$$Step_3 : f_i^x = Fitness(e_i^x, FN_x),$$

где f_i^x – значение приспособленности, полученное при вычислении соответствующей функции; $Fitness(e_i^x, FN_x)$ – функция приспособленности (перев. на англ. *Fitness Function* с транскрипцией, как «Фитнес-функция» – альтернативное используемое название) для i-го ЭПК в x-ом представлении с использованием формальной нотации (FN) для этого же представления. Соответственно, таким же образом можно получить значение приспособленности $x-1$-го представления.

Прекращение выполнения процесса генетического РИ произойдет при получении такого псевдо-предыдущего ЭПК, из которого бы получался псевдо-текущий ЭПК, тождественный текущему исследуемому – т.е. когда его приспособленность в текущем представлении является «идеальной» или максимальной:

$$f_i^x = f_0^x \Rightarrow f_i^{x-1}$$

искомый предыдущий экземпляр программного кода, где f_0^x – «идеальная» приспособленность особи.

Шаг 4. Селекция псевдо-текущих экземпляров программного кода

Из всего множества псевдо-текущих ЭПК отбирается подмножество, псевдо-текущие ЭПК которых обладают наивысшими значениями функции приспособленности. Таким образом, тот программный код, который далек от искомого, по возможности убирается из популяции (т.е. отсекается из списка будущих вариантов решения). Так, в примере из Шага 3 будет «отсечен» код с условными ветвлениями.

Шаг ($Step_4$) имеет следующую формальную запись:

$$\text{Step4} : \{e_i^{x-1} : i \in \mathbb{N},\ i \le M < N\}' \equiv \{e_i^{x-1}\}'_M = \\ = Selection_M(\{e_i^{x-1}\}_N, \{f_i^x\}_N, Setting_S),$$

где M – количество элементов в новом множестве (очевидно, меньшее N), полученном в результате скрещивания; $Selection_M(\{e_i^{x-1}\}_N, \{f_i^x\}_N, \dots)$ – операция отбора M ЭПК-особей (из множества мощностью N) в $x-1$-ом представлении, имеющих наилучшую приспособленность, определяемую по значению соответствующей функции в x-ом представлении; $Setting_S$ – настройки операции селекции, определяющие параметры ее работы (например, способ или уровень отсечения нескольких ЭПК по значению приспособленности); верхний индекс «’» (а также «’’» и «’’’», используемые далее) отражает факт получения новых ЭПК из существующих в одном представлении в рамках генетического реинжиниринга, а также служит для указания отличия множеств. При этом, для работы операции формальная нотация представления не является необходимой. Соответственно, таким же образом можно произвести селекцию и для $x-2$-го представления.

Шаг 5. Скрещивание псевдо-текущих экземпляров программного кода

ЭПК, оставшиеся после Шага 4, будут скрещены друг с другом, что позволит получить новое множество ЭПК, которые обладают особенностями, позволяющими им преобразовываться в ЭПК, близкий к текущему. Это возможно путем интерпретации ген у хромосомы особи, как отдельных конструкций программного кода в соответствии с используемой формальной нотацией. Так, например, если после Шага 4 были выделены следующие ЭПК на языке C (гены которых отмечены своим цветом):

```
1) A = X + 1
2) B = Y – 2
```

то после скрещивания (путем составления нового кода из двух частей – начала 1-го и конца 2-го) могут появиться следующие ЭПК:

```
1) A = Y – 2
2) A = X – 2
3) A = X + 2
```

Естественно, должна обеспечиваться корректность новых ЭПК путем учета формальной нотации.

Шаг ($Step_5$) имеет следующую формальную запись:

$$Step_5 : \{e_i^{x-1}\}''_N = Crossing_N(\{e_i^{x-1}\}'_M, FN_{x-1}, Setting_C),$$

где $Crossing_N(\{e_i^{x-1}\}'_M, \dots, \dots)$ – операция скрещивания M ЭПК-особей (полученных после селекции) в $x-1$-ом представлении для получения полного набора их популяции размером N, особи которой частично обладают генами, «показавшими» лучшую приспособленность; $Setting_C$ – настройки операции скрещивания, определяющие параметры ее работы (например, применяемый алгоритм). При этом в работе операции учитывается формальная нотация (FN_{x-1}) представления этого множества ЭПК, что позволяет заведомо избежать некорректного программного кода. Соответственно, таким же образом можно произвести скрещивание и для $x-2$-го представления. Конкретные алгоритмы скрещивания выходят за рамки настоящей статьи (хотя существует их целый пул [11, 12]), а в качестве примера можно привести составление нового программного кода из начала и конца других ЭПК.

Шаг 6. Мутация псевдо-текущих экземпляров программного кода

В каждом ЭПК, полученном после Шага 5, может быть произведена мутация путем случайной замены его конструкции (или их множества) на другую, как и ранее, корректную [13]. Например, в примере из Шага 5 первая конструкция кода для 2-го ЭПК может быть поменяна с «A» на «B» (отмечено желтым цветом), что позволит получить следующий код:

```
3) B = X - 2
```

Данный шаг как раз предназначен для выхода из «неглубоких» локальных экстремумов.

Шаг ($Step_6$) имеет следующую формальную запись:

$$\mathrm{Step}_6 : \{e_i^{x-1}\}_N''' = Mutation_N(\{e_i^{x-1}\}_N'', FN_{x-1}, Setting_M),$$

где $Mutation_N(\{e_i^{x-1}\}_N'', \dots, \dots)$ – операция мутации ген для N ЭПК-особей (полученных после скрещивания) в $x-1$-ом представлении; $Setting_M$ – настройки операции мутации, определяющие параметры ее работы (например, количество мутирующих особей и их ген). Как и в случае скрещивания, в работе операции используется формальная нотация (FN_{x-1}) представления. Соответственно, таким же образом можно произвести скрещивание и для $x-2$-го представления.

Шаг 7. Повторение создания популяции

Данный шаг является формальным и обеспечивает итеративность выполнения генетического РИ; он приводит к безусловному переходу на Шаг 2. Как результат – полученное на выходе Шага 7 множество ЭПК (после 1-й итерации селекции, скрещивания и мутации) становится входным для Шага 2.

Шаг ($Step_7$) имеет следующую формальную запись:

$$Step_7 : \{e_i^{x-1}\}_N''' \rightarrow \{e_i^{x-1}\}.$$

Как было указано выше, выход из итеративного выполнения генетического РИ происходит на Шаге 3 при получении ЭПК с «идеальной» приспособленностью.

Отметим, что выполнение генетических алгоритмов однозначно будет работать значительно быстрее полного перебора всех вариантов ЭПК [14], что частично и обосновывает его применимость в интересах РИ.

Заключение

В работе предложена концепция восстановления представлений программы в интересах дальнейшего анализа, в том числе экспертным способом. В основу концепции положена адаптация генетических алгоритмов путем итеративного подбора экземпляров программы в предыдущем представлении для их полного соответствия текущему. Классической задачей применения концепции является декомпиляция машинного кода программы в псевдоисходный код. Концепция представлена в виде двух схем – высокоуровневой, отражающей общую (де)эволюцию представлений, и низкоуровневой – описывающей сам процесс такого генетического реверс-инжиниринга.

Основным научным результатом исследования, представленного в статье, является концепция (включая ее схемы), предлагающая иную точку зрения на процесс восстановления представлений – не обратным способом, т.е. от текущего к предыдущему, а прямым – работая с псевдо-предыдущим и оценивая его близость к исследуемому текущему. Именно такой качественно новый подход к деэволюции представлений и определяет новизну исследования. Также в отличии от преобладающего множества существующих способов, использованный в данном исследовании способ описания процесса (де)эволюции представлений (включая предлагаемый реинжиниринг) впервые имеет формализованный вид.

Теоретической значимостью результата является расширение комплекса взглядов на процесс деэволюции программ – с классических исходного и машинного кода до любого из набора в рамках жизненного цикла программы – что удалось достигнуть за счет заимствования подхода естественного отбора у природы.

Практическая значимость результата заключается в получении концептуальной и теоретической базы для создания программных средств по проведению деэволюции представлений, что, в том числе, призвано существенно повысить эффективность поиска в них уязвимостей [15, 16].

Продолжением исследования планируется создание метода (или комплекса методов) по генетическому проведению реверс-инжиниринга в идеале на каждом из возможных представлений программы, но как минимум из набора классических.

Литература

1. *Израилов К. Е. Концепция генетической деэволюции представлений программы. Часть 1 // Вопросы кибербезопасности. 2024. № 1. С. 61–66. DOI: 10.21681/2311-3456-2024-1-61-66*
2. *Загинайло М. В., Фатхи В. А. Генетический алгоритм как эффективный инструмент эволюционных алгоритмов // Инновации. Наука. Образование. 2020. № 22. С. 513–518.*
3. *Аралбаев Р. А., Тарасов А. А. Задачи оптимизации и применение алгоритмов генетический алгоритм на практике // Инновации. Наука. Образование. 2021. № 48. С. 1645–1653.*
4. *Казакевич А. В., Кораченцов А. А. Автоматизация поиска глобального экстремума функции с использованием генетических алгоритмов // Colloquium-Journal. 2019. № 26-2 (50). С. 75–77. DOI: 10.24411/2520-6990-2019-10931.*
5. *Израилов К. Е. Концепция генетической декомпиляции машинного кода телекоммуникационных устройств // Труды учебных заведений связи. 2021. Т. 7. № 4. С. 10–17. DOI:10.31854/1813-324X-2021-7-4-95-109.*
6. *Израилов К. Е. Применение генетических алгоритмов для декомпиляции машинного кода // Защита информации. Инсайд. 2020. № 3 (93). С. 24–30.*
7. *Федорченко Л. Н., Афанасьева И. В. О построении систем со сложным поведением на принципах синтаксически ориентированного управления // Вестник Бурятского государственного университета. Математика, информатика. 2020. № 2. С. 15–35. DOI: 10.18101/2304-5728-2020-2-15-35.*
8. *Миронов С. В., Батраева И. А., Дунаев П. Д. Библиотека для разработки компиляторов // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 77–88. DOI: 10.15514/ISPRAS-2022-34(5)-5.*
9. *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.*
10. *Борисов П. Д., Косолапов Ю. В. Способ оценки похожести программ методами машинного обучения // Труды Института системного программирования РАН. 2022. Т. 34. № 5. С. 63–76. DOI: 10.15514/ISPRAS-2022-34(5)-4.*
11. *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.*
12. *Марков А. Д., Повираева М. Л., Дробышева В. О. Генетические алгоритмы. Бинарные операторы скрещивания // Научный электронный журнал Меридиан. 2020. № 9 (43). С. 66–68.*
13. *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.*
14. *Федоров Е. А. Исследование скорости работы генетического алгоритма и алгоритма полного перебора // Сборник избранных статей научной сессии ТУСУР. 2019. № 1-2. С. 107–109.*
15. *Devine T. R., Campbell M., Anderson M., Dzielski D. SREP+SAST: A Comparison of Tools for Reverse Engineering Machine Code to Detect Cybersecurity Vulnerabilities in Binary Executables // In proceedings of the International Conference on Computational Science and Computational Intelligence (Las Vegas, NV, USA, 14-16 December 2022). 2022. PP. 862–869. DOI: 10.1109/CSCI58124.2022.00156.*
16. *Васильев В. И., Вульфин А. М., Кучкарова Н. В. Автоматизация анализа уязвимостей программного обеспечения на основе технологии Text Mining // Вопросы кибербезопасности. 2020. № 4 (38). С. 22-31. DOI: 10.21681/2311-3456-2020-04-22-31*