УДК 004.051

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ВЫБОРА КОМПОНЕНТОВ ИНФОРМАЦИОННЫХ СИСТЕМ НА ОСНОВЕ ЭКСПЕРИМЕНТАЛЬНЫХ ОЦЕНОК КРИТЕРИЕВ КАЧЕСТВА[1]

Статья поступила в редакцию 19.04.2019, в окончательном варианте – 10.05.2019.

Гусев Александр Алексеевич, Кубанский государственный университет, 350040, Российская Федерация, г. Краснодар, ул. Ставропольская, 149,

аспирант, https://orcid.org/0000–0003–2437–8537, https://elibrary.ru/author_items.asp?authorid=835966, e-mail: alexandrgsv@gmail.com

Ильин Дмитрий Юрьевич, МИРЭА – Российский технологический университет, 119454, Российская Федерация, г. Москва, пр. Вернадского, 78,

аспирант, http://orcid.org/0000–0002–0241–2733, https://elibrary.ru/author_items.asp?authorid=892115, e-mail: i@dmitryilin.com

Никульчев Евгений Витальевич, МИРЭА – Российский технологический университет, 119454, Российская Федерация, г. Москва, пр. Вернадского, 78,

доктор технических наук, профессор, профессор кафедры управления и моделирования систем, http://orcid.org/0000–0003–1254–9132, https://elibrary.ru/author_items.asp?authorid=396636, e-mail: nikulchev@mail.ru

В статье представлена методика оценки эффективности набора компонентов программного обеспечения на основе вычислительных экспериментов, воспроизводимых под управлением генетического алгоритма. Для представления наборов компонентов в виде натуральных генотипов вводится отображение кодирования. Обратное отображение используется для дешифровки генотипа. На первом шаге методики генетический алгоритм создает начальную популяцию случайных генотипов, преобразуемых в оцениваемые наборы компонентов программного обеспечения. Затем происходит инициализация каждого набора и исполнение заданного перечня операций со снятием экспериментальных замеров для вычислительных экспериментов с информационной системой по 14 заданным частным критериям эффективности. На их основе, с учетом весовых коэффициентов, задающих цели в области управления качеством обслуживания, генетический алгоритм рассчитывает интегральный функционал качества для каждого исследуемого набора компонентов информационной системы. После этого выполняются генетические операторы, и происходит генерация нового (усовершенствованного) поколения генотипов с последующими экспериментальными измерениями для соответствующих наборов компонентов. Процедура повторяется вплоть до выполнения условий остановки. В статье показано применение предлагаемой методики к оценке эффективности выбора компонентов Node.js. Для этой цели разработана MATLAB-программа генетического поиска и сценарий эксперимента для виртуальной машины, работающей под управлением операционной системы Ubuntu 16.04 LTS. Последняя развертывается с помощью средства конфигурирования виртуальной среды разработки Vagrant.

Ключевые слова: качество систем и программ, эффективность взаимодействия программ, генетический алгоритм, эволюционные вычисления, вычислительные эксперименты, информационная система

GENETIC ALGORITHM OF SELECTING INFORMATION SYSTEMS COMPONENTS BASED ON EXPERIMENTAL EVALUATIONS OF QUALITY CRITERIA

The article was received by editorial board on 19.04.2019, in the final version – 10.05.2019.

Gusev Aleksandr A., Kuban State University, 149 Stavropolskaya St., Krasnodar, 350040, Russian Federation,

post-graduate student, https://orcid.org/0000–0003–2437–8537, https://elibrary.ru/author_items.aspauthorid=835966, e-mail: alexandrgsv@gmail.com

Ilin Dmitry Yu., MIREA – Russian Technological University, 78 Vernadskiy Ave., Moscow 119454, Russian Federation,

Post-graduate student, http://orcid.org/0000–0002–0241–2733, https://elibrary.ru/author_items.asp?authorid=892115, e-mail: i@dmitryilin.com

Nikulchev Evgeny V., MIREA – Russian Technological University, 78 Vernadskiy Ave., Moscow 119454, Russian Federation,

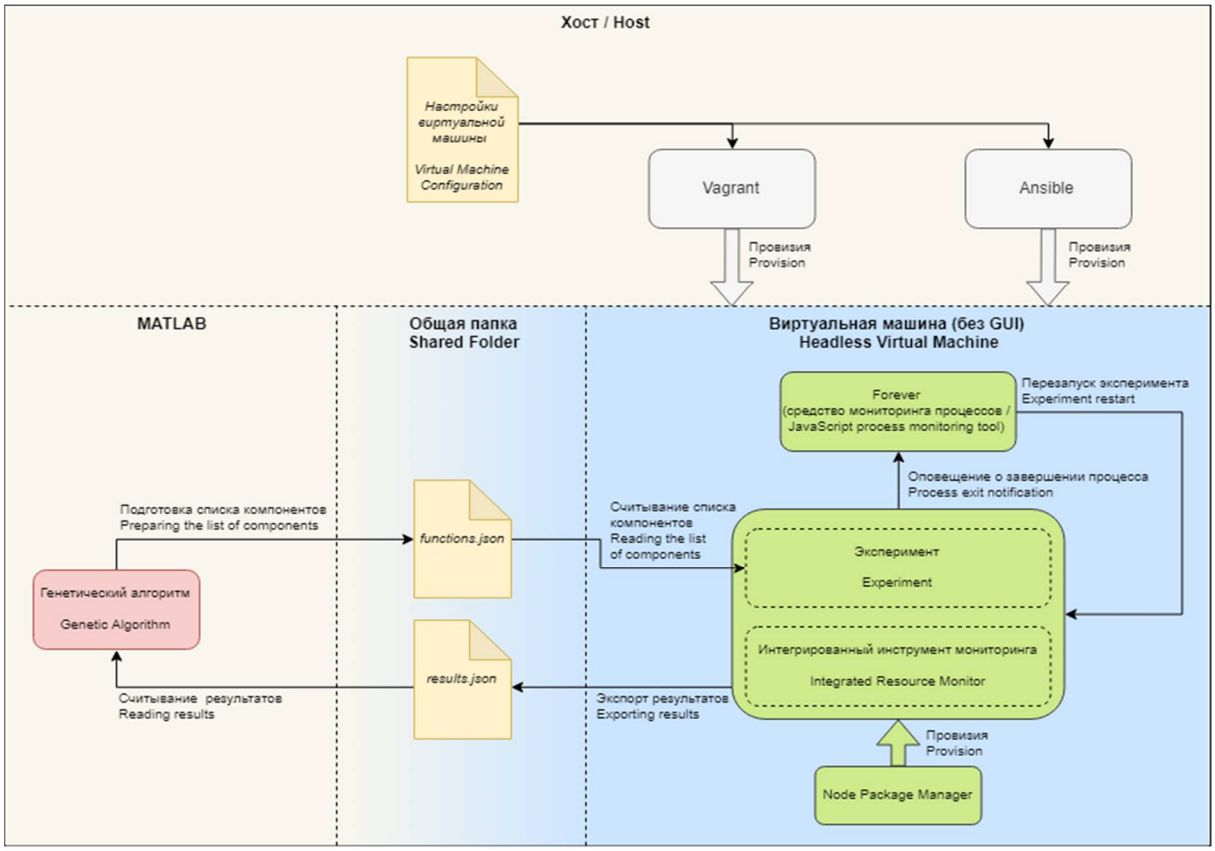
Doct. Sci. (Engineering), Professor, Professor of the Chair of Systems Control and Modelling, http://orcid.org/0000–0003–1254–9132, https://elibrary.ru/author_items.asp?authorid=396636, e-mail: nikulchev@mail.ru

[1] Работа выполнена при финансовой поддержке РФФИ, проект 17-29-02198 (офи_м) «Разработка открытой экспериментально-аналитической веб-платформы для сбора и интеллектуального анализа данных междисциплинарных исследований в области психического здоровья».

The article presents a methodology for assessing the effectiveness of a set of software components based on reproducible experiments under the control of a genetic algorithm. To represent the sets of components in the form of natural genotypes, an encoding mapping and reverse mapping is introduced to decipher the genotype. In the first step of the technique, the genetic algorithm creates an initial population of random genotypes, which are then converted into estimated sets of software components. Then, each set is initialized and a given list of operations is executed with experimental measurements taken according to 14 specified individual performance criteria. Based on experimental measurements, taking into account weighting factors that set goals in the field of Quality of Service, the genetic algorithm calculates the integral quality functional for each set of components of the information system under investigation, after which genetic operators are performed and the generation of improved genotypes is generated and experimental measurements are done for the corresponding sets, the procedure is repeated until the stop conditions are met. The article shows the application of the proposed methodology to evaluating the effectiveness of selecting Node.js components, for which a MATLAB genetic search program was developed as well as the experiment scenario for a virtual machine running the Ubuntu 16.04 LTS operating system. The virtual machine was deployed using the Vagrant virtual environment configuration tool.

Keywords: quality of systems and programs, the effectiveness of program interaction, genetic algorithm, evolutionary computation

Graphical annotation (Графическая аннотация)



Введение. Эффективный выбор компонентов информационных систем (ИС) на основе оценок критериев [36] качества обслуживания становится все более актуальной проблемой в связи с развитием каркасного подхода [4] к проектированию и разработке ИС. В работе данная проблема рассматривается в контексте высоконагруженных распределенных клиент-серверных информационных систем, реализованных на языке JavaScript.

Фреймворк, или каркас [1, 10], – это шаблонное архитектурное решение. Оно позволяет унифицировать процесс разработки ИС на основе комбинирования постоянной части ИС (фреймворка), не меняющейся от конфигурации к конфигурации, и подключаемых компонентов, совместимых с постоянной частью. JavaScript фреймворк – это фреймворк, написанный на языке JavaScript, позволяющий программистам манипулировать набором совместимых компонентов (библиотек) для решения конкретной задачи. Отличие фреймворка от библиотеки JavaScript проявляется в потоке управления [31]: библиотека всегда вызывается своим родительским кодом, в то время как фреймворк определяет общую архитектуру ИС [19] и вызывает те или иные компоненты для реализации функциональности, определенной разработчиком.

Разнообразие задач, решаемых с помощью JavaScript, в последние годы привело к появлению сотен фреймворков. Их можно разделить на две группы: 1) универсальные (например, Node.js [2]). Они

позволяют использовать JavaScript при написании серверной части веб-приложения в качестве языка общего назначения с возможностью взаимодействия с устройствами ввода-вывода; 2) фреймворки для написания браузерных (front-end) приложений, выполняемых на стороне пользователя. Примерами являются Angular.js [27, 44]; Angular [46, 25] (написан на языке TypeScript [20], являющимся обратносовместимой модификацией JavaScript); Vue.js [38, 24]; React.js [22, 43] и многих других [15].

Для поддержки оптимального выбора фреймворка при реализации задач разработки ИС с помощью JavaScript ранее были предложены методика SPEC (Speed, Productivity, Ecosystem, Compatibility) [26] и разнообразные бенчмарк-тесты. Они дают возможность измерить экономичность программного кода [32] и производительность [11, 33, 34] того или иного фреймворка с некоторым набором компонентов на типовых операциях. Указанные методики и тесты позволяют оценить соответствие фреймворка общим потребностям разработчика при заданном наборе компонентов. Однако актуальной задачей является создание для набора альтернативных технологий разработки специализированной методики для оценки обеспечения гарантированного качества обслуживания [7] (Quality of Service, QoS) и эффективности функционирования в заданных условиях. Под ними понимаются конкретное рабочее окружение, вычислительная инфраструктура, вычислительные нагрузки при нормальном и пиковом режиме функционирования и др.

Базовым элементом такой методики должна являться процедура проведения воспроизводимых вычислительных экспериментов [6] по оценке эффективности функционирования компонентов программного обеспечения. При автоматизации данной процедуры в общем случае необходимо решить задачу сокращения числа переборов компонентов программного обеспечения для получения оптимального набора. Решение может быть найдено за счет использования генетических алгоритмов. Они хорошо зарекомендовали себя в задачах многокритериальной оптимизации в области программных разработок: оптимизации усилий по разработке программного обеспечения [29, 30, 48]; оптимизации размещения вычислительных ресурсов [8, 17]; генерации оптимальных тестовых наборов данных [14, 39]; оценке [13] и повышении [23] надежности программного обеспечения; оптимизации разбиения программного обеспечения на модули [35]; при выборе приоритетов выпуска версий [28]; при разделении реализации функциональности выполняемых разработок на программную и аппаратную часть [45]; при рефакторинге программного обеспечения [41]; при решении задач управления проектами [12, 42] и человеческими ресурсами [49, 50]; в других задачах, в том числе связанных с разработкой сервисно-ориентированной архитектуры (SOA) web-сервисов с динамическим выбором компонентов с поддержкой QoS [21, 37, 47].

Целью данной работы является создание и экспериментальная апробация методики генетического поиска эффективного выбора компонентов ИС на основе экспериментальных оценок критериев качества.

Статья состоит из четырех разделов. В первом из них сформулирована постановка задачи. Во втором разделе приводится описание методики генетического поиска и конфигурации экспериментального стенда. В третьем разделе рассмотрены результаты генетического поиска эффективного выбора компонентов ИС на основе экспериментальных оценок. Четвертый раздел содержит обсуждение полученных результатов и завершает статью.

Постановка задачи. Пусть имеется набор $F = (f_1, \dots, f_n)$ требуемых функций ИС, каждую из которых можно реализовать с помощью альтернативных наборов компонентов фреймворка. Сопоставим каждой функции f_i набор альтернативных компонентов $d_i = \left(q_1^i, \dots, q_{j-1}^i, q_j^i, q_{j+1}^i, \dots\right)$, позволяющих реализовать данную функцию. Обозначим все такие наборы как $D = (d_1, \dots, d_n)$, а в виде Ω – множество всех возможных уникальных конфигураций фреймворка, соответствующих выбору по одной альтернативе q_j^i из каждого $d_i \in D$ для реализации каждой функции $f_i \in F$.

Обозначим как $\Psi(\omega), \omega \in \Omega$ – экспериментально оцениваемый общий функционал, определяющий качество ИС:

$$\Psi(\omega) = \varepsilon \cdot Round\left(\sum_{j=1}^{14} w_j R_j\right), \quad (1)$$

где частные критерии качества R_j для рассматриваемой задачи определены в таблице 1. Совокупность этих критериев, по мнению авторов, обладает свойством «необходимости и достаточности», позволяя эффективно различать получающиеся решения в процессе генетического поиска. Весовые коэффициенты w_j, задающие цели в области QoS, указаны в таблице 2. Величина ε – нормировочный множитель, в данном исследовании равный 0,1.

Таблица 1 – Критерии качества информационной системы

Обозначение	Критерий	Единица измерения
R_1	Время работы микропроцессора, затраченное на инициализацию эксперимента	мкс
R_2	Время работы микропроцессора, затраченное на исполнение системных функций в ходе инициализации эксперимента	мкс
R_3	Прирост утилизируемого объема оперативной памяти, отмечаемый по завершении инициализации эксперимента (включая heap, code, segment и stack)	байт
R_4	Прирост размера heap (кучи), отмечаемый по завершении инициализации эксперимента	байт
R_5	Прирост объема используемой heap (кучи), отмечаемый по завершении инициализации эксперимента	байт
R_6	Прирост объема памяти, используемой объектами C++, связанными с JavaScript объектами, отмечаемый по завершении инициализации эксперимента	байт
R_7	Реальное время, затраченное на инициализацию эксперимента	нс
R_8	Время работы микропроцессора, затраченное на эксперимент	мкс
R_9	Время работы микропроцессора, затраченное на исполнение системных функций в ходе эксперимента	мкс
R_{10}	Прирост утилизируемого объема оперативной памяти, отмечаемый по завершении эксперимента (включая heap, code, segment и stack)	байт
R_{11}	Прирост размера heap (кучи), отмечаемый по завершении эксперимента	байт
R_{12}	Прирост объема используемой heap (кучи), отмечаемый по завершении эксперимента	байт
R_{13}	Прирост объема памяти, используемой объектами C++, связанными с JavaScript объектами, отмечаемый по завершении эксперимента	байт
R_{14}	Реальное время, затраченное на эксперимент	нс

Таблица 2 – Весовые коэффициенты

Коэффициент	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
Значение	$\frac{1}{3}10^{-4}$	$\frac{1}{3}10^{-4}$	$\frac{1}{3}10^{-6}$	$\frac{1}{3}10^{-6}$	$\frac{1}{3}10^{-6}$	$\frac{1}{3}10^{-5}$	$\frac{1}{3}10^{-8}$	$\frac{1}{3}10^{-3}$	$\frac{1}{3}10^{-3}$
Коэффициент	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}				
Значение	$\frac{1}{3}10^{-5}$	$\frac{1}{3}10^{-5}$	$\frac{1}{3}10^{-5}$	$\frac{1}{3}10^{-2}$	$\frac{1}{3}10^{-6}$				

Необходимо определить эффективный набор компонентов ИС $\omega_{ef} \in \Omega$,, решая задачу

$$\Psi(\omega) \to \min_{\omega \in \Omega} . \quad (2)$$

Методика экспериментальной оценки. Автоматизированная методика выбора эффективного набора компонентов ИС включает использование простого генетического алгоритма (ПГА) для генерации и экспериментальной оценки конфигураций.

Интеграция компонентов фреймворка реализована с помощью функционального подхода, что является наиболее удобным для комбинирования различных сочетаний. Каждая функция, используемая в ходе проводимого испытания, является своего рода программным интерфейсом, который реализуется с помощью одного из компонентов. Так как компоненты, как правило, предоставляют инструментарий, выходящий за рамки одной функции, то они могут использоваться для реализации нескольких функций. Применение одного компонента для выполнения целого ряда задач в общем случае является предпочтительным, так как это снижает объём необходимой оперативной памяти вычислительной системы. В таблице 3 показан набор используемых функций и компонентов, реализующих эти функции в вычислительном эксперименте.

На этапе инициализации производится вызов функций, задающих базовые настройки компонентов. Каждая из них формирует на выходе новую анонимную функцию [51, разд. 7.1.2], имеющую исключительный доступ к компоненту с заданными настройками. Далее анонимные функции помещаются в единое «пространство имён» с названиями, представленными в таблице 3. Для повышения надёжности получаемых результатов перед инициализацией производится очистка кэша компонентов Node.js.

Таблица 3 – Перечень используемых функций и компонентов

Функция	Компоненты, реализующие функцию	Описание
Filter	Lodash Underscore	Последовательно проверяет все элементы массива на предмет соответствия условию и возвращает массив, состоящий из элементов, для которых проверка дала значение «Истина»
First	Lodash Underscore	Возвращает первый элемент массива
FsRead	Fs-extra Fs	Считывает данные из файла
FsReaddir	Fs-extra	Считывает содержимое каталога, возвращая массив имен файлов и директорий в каталоге
FsReaddirRecursive	Recursive-readdir	Рекурсивно считывает содержимое каталога, возвращая массив имен файлов и директорий в каталоге
HashMD5	Hasha md5 Ts-md5	Вычисляет MD5-хеш от заданного набора данных
Map	Lodash Underscore Средства языка JavaScript	Применяет заданную функцию ко всем элементам массива, возвращая тем самым новый массив, состоящий из преобразованных элементов
PathResolve	Path	Формирует полный путь к файлу или директории на основе заданного массива элементов пути
StringReplace	Средства языка JavaScript	Находит и заменяет подстроку в переданной строке
ZipCompress	Adm-zip Jszip Zipit	Производит архивацию переданного массива файлов и возвращает сформированный Zip-архив

После инициализации начинается этап исполнения фрагмента кода. Для целей исследования используется следующий алгоритм.

1. Формируется путь к каталогу с набором вложенных папок.
2. Считывается список вложенных папок.
3. Исключаются скрытые папки (начинающиеся с символа «.»).
4. Для каждой папки формируется ее путь.
5. Для каждого пути выполняется следующее:

5.1 Рекурсивно считывается весь список вложенных файлов.

5.2 Все файлы считываются с диска и загружаются в оперативную память.

5.3 Производится создание Zip-архива в оперативной памяти.

5.4 Вычисляется MD5-хеш от созданного архива.

После процедур инициализации и исполнения фрагмента кода генерируется json-файл results.json. Он содержит исходные данные для расчета функционала (1). Получение этих данных осуществляется через интерфейс объекта process фреймворка Node.js.

Для численного представления конфигураций вводится отображение кодирования $G: \Omega \to \Lambda \subseteq \mathbb{N}^n$. Таким образом, каждой конфигурации $\omega \in \Omega$, называемой фенотипом, соответствует натуральный набор $\zeta = G(\omega)$, $\omega \in \Omega, \zeta \in \Lambda \subseteq \mathbb{N}^n$, называемый генотипом. В процессе работы ПГА генотипы представляются как

$$\zeta_p^k = \left(\alpha_{1_p}^k \dots \alpha_{n_p}^k\right), k = 1..\left|\Theta_p\right|,$$

где каждая $\alpha_{i_p}^k$ принимает значения от 1 до $|d_i|$, соответствующие порядковому номеру выбранной альтернативы из d_i; Θ_p – множество генотипов (популяция особей), принадлежащих p-му поколению. Вводится также обратное отображение $G^{-1}: \Lambda \to \Omega$, осуществляющее преобразование генотипа конфигурации в соответствующий ему фенотип. С учетом введенных обозначений $\Psi(\omega)$ можно рассматривать как функцию приспособленности ПГА, а исходная задача (2) сводится к задаче

$$\Psi(\omega) \to \min_{\omega \in \{G^{-1}(\zeta), \zeta \in \Theta_{term},\}}, \tag{3}$$

где Θ_{term} – последняя популяция решений перед остановкой ПГА.

Алгоритм генетического поиска при решении задачи (3) выглядит следующим образом.

1. *Создание начальной популяции*: присвоить p = 1; создать N случайных генотипов решений, составляющих начальную популяцию $\Theta_1 = \{\zeta_1^1, \zeta_1^2, \dots, \zeta_1^N\}$, для каждого ζ_1^j определить соответствующий выбор компонентов фреймворка $\omega_1^j = G^{-1}\left(\zeta_1^j\right)$, произвести вычислительный эксперимент и рассчитать

вектор значений функционала качества (1) для всех особей популяции $\mu = (\mu_1, \mu_2, \dots, \mu_N)$, $\mu_i = \Psi(\omega_1^j)$; определить $\mu_{min} = \min_{j=1,N} \mu_j$.

2. *Начало создания нового поколения*: присвоить $k = 1$.

3. *Селекция особи в новое поколение:* присвоить p = p + 1; заданным методом отбора выбрать решение ζ'^k_p.

4. *Кроссинговер*: заданным методом отбора выбрать из популяции второе решение ζ''^k_p. С вероятностью P_K скрестить особи ζ'^k_p и ζ''^k_p оператором кроссинговера. Обозначить результат (потомок) ζ'^k_p.

5. *Мутация*: с вероятностью P_M подействовать на особь ζ'^k_p оператором мутации.

6. *Выбрать следующую особь*: присвоить $k = k + 1$; в случае $k = N$ – перейти к шагу 7, в противном случае – к шагу 3.

7. *Выбор элитарной особи*: из популяции Θ_p отбирается особь ζ_p^i с наименьшим значением функции функционала качества $\mu_i = \min_{j=1,N} \mu_j$

8. *Создание нового поколения:* из отобранных ранее особей создать популяцию $\Theta_{p+1} = \{\zeta'^1_p, \zeta'^2_p, \dots, \zeta'^N_p\}$; для каждого ζ'^j_p определить соответствующий выбор компонентов фреймворка $\omega'^j_p = G^{-1}(\zeta'^j_p)$, произвести эксперимент и рассчитать вектор значений функционала качества (1) популяции $\mu' = (\mu'_1, \mu'_2, \dots, \mu'_N)$, $\mu'_i = \Psi(\omega'^j_p)$; определить $\mu_{min} = \min_{j=1,N} \mu_j$.

9. *Проверка условия завершения алгоритма:* если условие завершения алгоритма не выполнено – переход к шагу 3, в противном случае выдать решение, соответствующее μ_{min} в качестве ответа, и завершить работу алгоритма.

В качестве оператора отбора в общем случае может использоваться турнирная селекция, метод рулетки, метод ранжирования, равномерное ранжирование, сигма-отсечение и модификации этих методов [9].

При кроссинговере (скрещивании) создается новая особь путем обмена подмножествами параметров между двумя родительскими особями [3]. Оператор мутации изменяет генотип особи заранее определенным образом [5].

При решении задач с целочисленными ограничениями для получения целых значений генов используются специальные генетические операторы, рассмотренные подробно в [18].

Остановка алгоритма может осуществляться при достижении предельного количества поколений; при достижении предельного количества стагнирующих поколений (лучшее значение функционала качества в этих поколениях не меняется); при разности между средними по популяции значениями функционала качества двух последовательных поколений меньше установленного порога; при уменьшении выхода за пределы граничных условий до установленного минимального порога; по запросу пользователя; в других случаях, определяемых разработчиком.

Для реализации ПГА при решении задачи (3) используется компонент ga, входящий в состав пакета Global Optimization Toolbox системы MATLAB. Эксперимент осуществляется на виртуальной машине под управлением Ubuntu 16.04 LTS, в которой развернут фреймворк Node.js 10.15.3. Создание виртуальной машины осуществляется с помощью средства конфигурирования виртуальной среды разработки Vagrant. В качестве основной (Хост) операционной системы также используется Ubuntu 16.04 LTS, в которой установлена система MATLAB R2018a. Использование виртуальной машины помогает обеспечить воспроизводимость вычислительного эксперимента, избежать неучтенного изменения параметров, сократить влияние других случайных факторов на результаты эксперимента.

При запуске виртуальной машины осуществляется считывание json-файла functions.json, генерируемого MATLAB-программой, осуществляющей генетический поиск. Файл functions.json является json-представлением исследуемой конфигурации $\omega_i = G^{-1}(\zeta_i)$ и задает набор используемых альтернативных компонентов Node.js для проведения испытаний, состоящих из двух основных этапов: инициализации компонентов и исполнения заданного фрагмента кода.

Методика проведения экспериментального исследования представлена в виде схемы на рисунке 1. Конфигурация экспериментального стенда приведена в таблице 4.

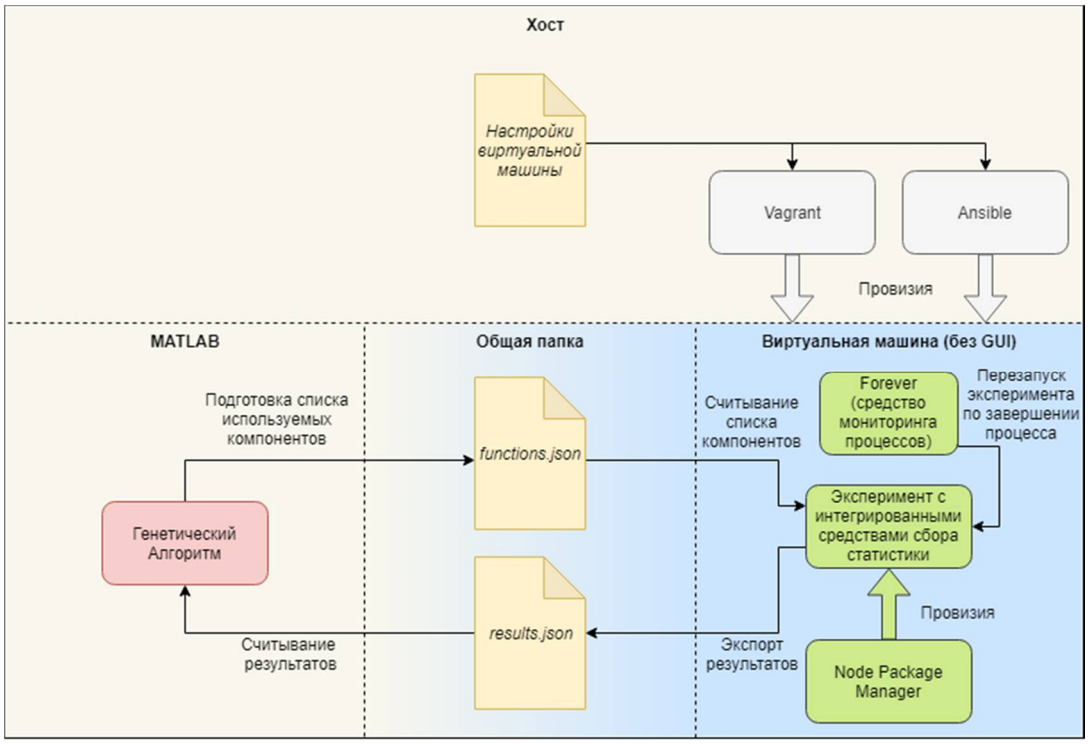


Рисунок 1 – Методика проведения экспериментального исследования

Таблица 4 – Конфигурация экспериментального стенда

Параметр	**Значение**
Микропроцессор	Intel® Core ™ i7–7700
Количество ядер	4
Логических процессоров	8
Тактовая частота	3,60 ГГц
Оперативная память	12,0 ГБ
Операционная система (Хост)	Ubuntu 16.04 LTS
Версия MATLAB	R2018a
Версия Vagrant	2.2.4
Версия Node.js	10.15.3
Параметры виртуальной машины	2 ядра процессора 2,0 ГБ ОЗУ Ubuntu 16.04 LTS
Используемое средство провизии	Ansible
Используемые средства обмена файлами с виртуальной машиной	NFS-сервер + BindFS внутри виртуальной машины
Установленное дополнительное системное программное обеспечение	– git – make – htop – iotop – rsync – node-gyp

Целочисленные ограничения [40]	Все гены принимают целые значения
Оператор отбора особей генетического алгоритма	Турнирная селекция одного из двух потенциальных родителей [18]
Оператор мутации генетического алгоритма	Extended power mutation [18]
Оператор скрещивания генетического алгоритма	Laplace crossover [18]
Вероятность мутации P_M	0,01
Вероятность скрещивания P_K	0,8
Количество отбираемых элитных особей (EliteCount)	1
Размер популяции (PopulationSize, N)	20
Предельное количество поколений	100
Предельное количество стагнирующих поколений	10
Порог изменения среднего значения функционала качества (FunctionTolerance)	$1 \cdot 10^{-6}$
Порог нарушения граничных условий (ConstraintTolerance)	$1 \cdot 10^{-3}$

Результаты и обсуждение. В результате реализации 13 поколений генетического поиска было найдено решение задачи (3), соответствующее значению функционала качества 0,6. Среднее значение функционала качества в терминальном поколении составило 0,805. Генетический поиск занял 78 секунд и завершился по условию снижения среднего изменения функционала качества между поколениями ниже установленного порога (FunctionTolerance). Экспериментальные замеры для терминального поколения генетического поиска представлены в таблице 5, эффективное решение идентифицировано в таблице 6.

Таблица 5 – Взвешенные экспериментальные замеры и значение $\Psi(\omega)$ в терминальном поколении

№ особи, i	w_1R_1	w_2R_2	w_3R_3	w_4R_4	w_5R_5	w_6R_6	w_7R_7	w_8R_8
1	0,6667	0	0,2621	1,5729	0,192	0,2496	0,1633	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5718	0,32	1,5696	0,7	
2	0,5333	0,2667	0,2594	1,5729	0,1869	0,2222	0,1444	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7919	0	0,5663	0,32	1,3938	0,8	
3	0,8	0	0,2567	1,7476	0,1779	0,2234	0,1481	0
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5587	0,32	1,3775	0,6	
4	0,8	0,2667	0,2567	1,7476	0,1864	0,2496	0,2468	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	1,693	0	1,4314	0,32	2,2832	1,1	
5	0,5333	0,1333	0,3427	1,7476	0,2139	0,2467	0,142	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5757	0,32	1,3925	0,7	
6	0,5333	0,1333	0,3168	1,7476	0,2244	0,2739	0,1439	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7919	0	0,5579	0,32	1,4382	0,8	
7	0,5333	0,2667	0,3441	2,0972	0,2251	0,2693	0,1868	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5668	0,32	1,5438	0,8	
8	0,6667	0,1333	0,2567	1,7476	0,2193	0,2498	0,1582	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Phi(\zeta_i)$	
	0	0,7919	0	0,5681	0,32	1,3952	0,8	
9	0,6667	0	0,2567	1,7476	0,1982	0,2231	0,1375	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7919	0	0,5709	0,32	1,5125	0,8	

10	0,4	0,2667	0,3386	2,2719	0,2448	0,2524	0,1661	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5713	0,32	1,6003	0,8	
11	0,6667	0,1333	0,4328	1,7476	0,2954	0,2483	0,1403	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5672	0,32	1,5009	0,7	
12	0,6667	0	0,2526	1,7476	0,2129	0,2696	0,1335	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,8329	0	0,5755	0,32	1,4342	0,8	
13	0,9333	0,1333	0,2649	1,9224	0,24	0,2773	0,6127	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7782	0	0,5654	0,32	1,6094	0,9	
14	0,6667	0,1333	0,3427	1,3981	0,2127	0,25	0,1535	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0	0	0,5803	0,32	1,5205	0,7	
15	0,6667	0,1333	0,3509	1,5729	0,2207	0,2522	0,1335	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,8055	0	1,4403	0,32	2,1216	0,9	
16	0,6667	0	0,2567	1,3981	0,2045	0,248	0,1943	2,6667
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,8055	0	0,5747	0,32	2,2553	1	
17	0,8	0	0,3441	1,5729	0,2145	0,2517	0,1496	0
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	1,3333	0,7919	0	0,5797	0,32	1,5925	0,8	
18	0,6667	0	0,2553	1,7476	0,1877	0,2231	0,127	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7919	0	0,5715	0,32	1,4226	0,8	
19	0,6667	0,1333	0,3536	1,9224	0,24	0,2505	0,1878	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7509	0	0,5582	0,32	1,4831	0,8	
20	0,8	0	0,2553	1,7476	0,1929	0,2227	0,1326	1,3333
	w_9R_9	$w_{10}R_{10}$	$w_{11}R_{11}$	$w_{12}R_{12}$	$w_{13}R_{13}$	$w_{14}R_{14}$	$\Psi(\omega)$	
	0	0,7919	0	0,558	0,32	1,4395	0,8	

Таблица 6 – Найденное эффективное решение

Генотип	Фенотип	
	Функция	Компонент
[2 3 2 1 1 1 2 1 1 1]	Filter	Underscore
	Map	Underscore
	First	Underscore
	PathResolve	Средства языка JavaScript
	StringReplace	Средства языка JavaScript
	ZipCompress	Adm-zip
	HashMD5	Md5
	FsRead	Fs-extra
	FsReaddir	Fs-extra
	FsReaddirRecursive	Recursive-readdir

График эволюции решений, показывающий уменьшение «значения величины штрафа», представлен на рисунке 2.

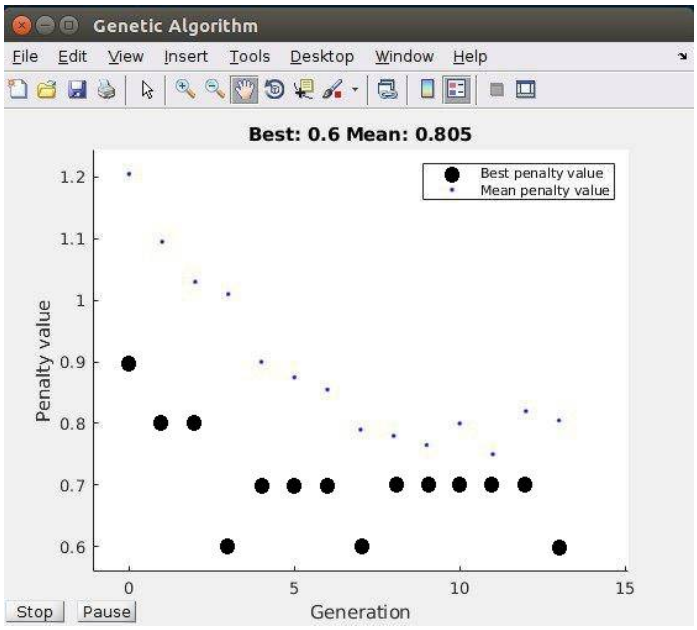


Рисунок 2 – График генетического поиска. Generation – номер поколения. Penalty value – значения функционала $\Psi(\omega)$. Best penalty value – минимальное значение $\Psi(\omega)$ в поколении. Mean penalty value – среднее значение $\Psi(\omega)$ в поколении

Необходимо отметить, что в процессе генетического поиска достаточно быстро появилась лучшая конфигурация с значением $\Psi(\omega) = 0{,}6$. Данная конфигурация выбиралась в следующее поколение как элитарная (см. шаг 7 Алгоритма генетического поиска, приведенного в разделе «Методика экспериментальной оценки»).

Однако при дальнейших прогонах значение $\Psi(\omega) = 0{,}6$ не удавалось получить вновь, что видно на графике (рис. 2). Данная «колебательность» объясняется неустранимым шумом измерений, вызванным небольшими вариациями реального времени исполнения одного и того же процесса машиной, общие причины которых рассмотрены в [16]. Однако генотип лучшего решения из поколения в поколение начинает все больше преобладать (это видно по убыванию среднего значения функционала) и идентификация решения все равно происходит. Причина – увеличивается количество вычислительных экспериментов, приходящихся на лучший генотип, что нивелирует случайные факторы в оценке этого генотипа.

Заключение. Создана методика эффективного выбора компонентов ИС на основе экспериментальных оценок критериев и генетического алгоритма. Методика апробирована в задаче поиска эффективного выбора компонентов Node.js для реализации определенного набора функций в соответствии с заданным функционалом качества. Указаны конфигурация и параметры экспериментального стенда, параметры работы генетического алгоритма. Сформулирован функционал качества, позволяющий учесть вклад набора из 14-ти экспериментально оцениваемых критериев в общую оценку эффективности выбора компонентов информационной системы. Приведены экспериментальные оценки критериев эффективности для терминального поколения генетического поиска. Идентифицирован эффективный выбор компонентов рассматриваемой ИС.

Библиографический список

1. Ахтырченко К. В. Методы и технологии реинжиниринга ИС / К. В. Ахтырченко, Т. П. Сорокваша // Труды Института системного программирования РАН. – 2003. – Т. 4. – С. 141–162.

2. Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript / Итан Браун. – Санкт-Петербург, 2017. – 336 с.

3. Гладков Л. А. Генетические алгоритмы : учебное пособие / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик. – 2-е изд. – Москва : Физматлит, 2006.

4. Горбунов-Посадов М. М. Расширяемые программы / М. М. Горбунов-Посадов. – Москва : Полиптих, 1999. – 336 с.

5. Емельянов В. В. Теория и практика эволюционного моделирования / В. В. Емельянов, В. В. Курейчик, В. М. Курейчик. – Москва : Физматлит, 2003. – 432 с.

6. Ильин Д. Ю. Проведение воспроизводимых экспериментов по оценке эффективности работы компонентов программного обеспечения / Д. Ю. Ильин, А. А. Гусев // Прикладные исследования и технологии ART 2019 : сб. тр. регион. конф. – Москва : МТИ, 2019. – С. 54–56.

7. Ильин Д. Ю. Выбор технологических решений для разработки программного обеспечения распределенных информационных систем / Д. Ю. Ильин, Е. В. Никульчев, П. В. Колясников // Современные информационные технологии и ИТ-образование. – 2018. – Т. 14, № 2. – С. 344–354.

8. Патент US5651099A США. Use of a genetic algorithm to optimize memory space / изобретатель Shane Konsella ; правообладатель Hewlett Packard Development Co LP. – 1995.

9. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – 2-е изд. – Москва : Горячая линия – Телеком, 2008. – 452 с.

10. Фаронов В. В. Создание приложений с помощью C# / В. В. Фаронов. – Москва : Горячая линия – Телеком ЭКСМО, 2008. – 576 с.

11. A comparison of the performance of a few popular javascript frameworks. – Режим доступа: https://github.com/krausest/js-framework-benchmark, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

12. Alba E. Software project management with GAs / Enrique Alba, J. Francisco Chicano // Information Science. – 2007. – Vol. 177, № 11. – P. 2380–2401.

13. Aljahdali S. H. Software reliability prediction using multi–objective genetic algorithm / Sultan H. Aljahdali, Mohammed E. El-Telbany // 2009 IEEE/ACS International Conference on Computer Systems and Applications / IEEE. – Rabat, 2009. – P. 293–300.

14. Bouchachia A. An Immune Genetic Algorithm for Software Test Data Generation / Abdelhamid Bouchachia // 7th International Conference on Hybrid Intelligent Systems (HIS 2007). – Kaiserlautern, Germany : IEEE, 2007. – P. 84–89.

15. Collection: Front-end JavaScript frameworks. – Режим доступа: https://github.com/collections/front-end-javascript-frameworks, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

16. CPU-Time Variation. – Режим доступа: https://www.ibm.com/support/knowledgecenter /en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieag200/cputvari.htm, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

17. Dai Y. S. Optimal testing-resource allocation with genetic algorithm for modular software systems / Y. S. Dai, M. Xie, Poh K. L., Yang B. // Journal of Systems and Software. – 2003. – Vol. 66, № 1. – P. 47–55.

18. Deep K. A real coded genetic algorithm for solving integer and mixed integer optimization problems / Kusum Deep, Krishna Pratap Singh, M. L. Kansal, C. Mohan // Applied Mathematics and Computation. – 2009. – № 212. – P. 505–518.

19. Difference Between Library and Framework. – Режим доступа: https://www.c–sharpcorner.com/UploadFile/a85b23/framework–vs–library/, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

20. Documentation: TypeScript. – Режим доступа: http://www.typescriptlang.org/docs/home.html, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

21. Fanjiang Y.-Y. Genetic algorithm for QoS-aware dynamic web services composition / Y.-Y. Fanjiang [и др.] // 2010 International Conference on Machine Learning and Cybernetics. Qingdao. – IEEE, 2010. – P. 3246–3251.

22. Fedosejev A. React.js essentials / A. Fedosejev. – Packt Publishing, 2015. – 208 p.

23. Feldt R. Generating diverse software versions with genetic programming: an experimental study / R. Feldt // IEEE Proceedings – Software. – 1998. – Vol. 145, № 6. – P. 228–236.

24. Filipova O. Learning Vue.js 2 Learn how to build amazing and complex reactive web applications easily with Vue.js / O. Filipova. – Packt Publishing Ltd, 2016. – 334 p.

25. Frisbie M. Angular 2 Cookbook / M. Frisbie. – Packt Publishing, 2017. – 464 p.

26. GrapeCity Wijmo | Take the SPEC Quiz. – Режим доступа: https://www.grapecity.com/en/spec, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

27. Green B. AngularJS / B. Green, S. Seshadri – O'Reilly Media, 2013. – 196 p.

28. Greer D. Software release planning: an evolutionary and iterative approach / D. Greer, G. Ruhe // Information and Software Technology. – 2004. – Vol. 46, № 4. – P. 243–253.

29. Huang S.-J. Optimization of analogy weights by genetic algorithm for software effort estimation / S.-J. Huang, N.–H. Chiu // Information and Software Technology. – 2006. – Vol. 48, № 11. – P. 1034–1045.

30. Huang S.-J. Integration of the grey relational analysis with genetic algorithm for software effort estimation / S.-J. Huang, N.-H. Chiu, L.-W. Chen // European Journal of Operational Research. – 2008. – Vol. 188, № 3. – P. 898–909.

31. InversionOfControl. – Режим доступа: https://martinfowler.com/bliki/InversionOfControl.html, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

32. Javascript Framework Comparison with Examples (React, Vue & Hyperapp). – Режим доступа: https://hackernoon.com/javascript-framework-comparison-with-examples-react-vue-hyperapp-97f064fb468d, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

33. JavaScript Frameworks, Performance Comparison [Электронный ресурс] Режим доступа: https://www.codementor.io/aminmeyghani/javascript–frameworks–performance–comparison–ntkw9sn66, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения: 16.04.2019).

34. JS web frameworks benchmark – Round 7. – Режим доступа: https://www.stefankrause.net/wp/?p=454, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

35. Kumari A. C. Software module clustering using a hyper–heuristic based multi–objective genetic algorithm / A. Charan Kumari, K. Srinivas, M. P. Gupta // 2013 3rd IEEE International Advance Computing Conference (IACC). – IEEE, 2013. – P. 813–818.

36. Lun L. Coverage Criteria for Component Path-oriented in Software Architecture / Lijun Lun, Xin Chi, Hui Xu // Engineering Letters. – 2019. – Vol. 27, № 1. – P. 40–52.

37. Ma Y. Quick convergence of genetic algorithm for QoS-driven web service selection / Y. Ma, C. Zhang // Computer Networks. – 2008. – Vol. 52, № 5. – P. 1093–1104.

38. Macrae C. Vue.js: Up and Running / C. Macrae. – O'Reilly, 2017. – 219 p.

39. Michael C. C. Generating software test data by evolution / C. C. Michael, G.E. McGraw, M. A. Schatz // IEEE Transactions on Software Engineering. – 2001. – Vol. 27, № 12. – P. 1085–1110.

40. Mixed Integer Optimization – MATLAB & Simulink. – Режим доступа: https://www.mathworks.com/help/gads/mixed-integer-optimization.html, свободный. – Заглавие с экрана. – Яз. англ. (дата обращения 16.04.2019).

41. Ouni A. The use of development history in software refactoring using a multi–objective evolutionary algorithm / Ali Ouni, Marouane Kessentini, Houari Sahraoui, Mohamed Salah Hamdi // Proceedings of the 15th annual conference on Genetic and evolutionary computation (GECCO '13). Christian Blum (Ed.). – ACM, 2013. – P. 1461–1468

42. Reddy J. P. Application of Petri Nets and a Genetic Algorithm to Multi-Mode Multi-Resource Constrained Project Scheduling / J. Prashant Reddy, S. Kumanan, O.V. Krishnaiah Chetty // The International Journal of Advanced Manufacturing Technology. – 2001. – Vol. 17, № 4. – P. 305–314.

43. Robbestad S. A. ReactJS blueprints / S. A. Robbestad. – Packt Publishing, 2016. – 422 p.

44. Ruebbelke L. AngularJS in Action / L. Ruebbelke, B. Ford. – Manning Publications, 2015. – 325 p.

45. Saha D. Hardware software partitioning using genetic algorithm / D. Saha, R. S. Mitra, A. Basu // Proceedings Tenth International Conference on VLSI Design. – IEEE, 1997. – P. 155–160.

46. Seshadri S. Angular: Up and running: Learning angular, step by step / S. Seshadri. – O'Reilly Media, 2018. – 312 p.

47. Shuang K. TTS-Coded Genetic Algorithm for QoS–driven web service selection / K. Shuang [и др.] // 2009 IEEE International Conference on Communications Technology and Applications. – IEEE, 2009. – P. 885–890.

48. Shukla K. K. Neuro-genetic prediction of software development effort / K. K. Shukla // Information and Software Technology. – 2000. – Vol. 42, № 10. – P. 701–713.

49. Stylianou C. A Multi-objective Genetic Algorithm for Software Development Team Staffing Based on Personality Types / Constantinos Stylianou, Andreas S. Andreou // Artificial Intelligence Applications and Innovations. AIAI 2012. IFIP Advances in Information and Communication Technology. – Springer, 2012. – Vol. 381. – P. 37–47.

50. Wena F. Multistage Human Resource Allocation for Software Development by Multiobjective Genetic Algorithm / F. Wena, C.-M. Lin // The Open Applied Mathematics Journal. – 2008. – Vol. 2. – P. 95–103.

51. Flanagan D. JavaScript: The Definitive Guide, 4th Edition / David Flanagan. – O'Reilly, 2001. – 936 p.

References

1. Akhtyrchenko K.V., Sorokvasha T.P. Metody i tekhnologii reinzhiniringa IS [Methods and technologies of information system reengineering]. *Trudy Instituta sistemnogo programmirovaniia RAN* [Proceedings of the Institute for System Programming of the RAS], 2003, vol. 4, pp. 141–162.

2. Brown E. *Veb-razrabotka s primeneniem Node i Express. Polnotsennoe ispolzovanie steka JavaScript* [Web Development with Node and Express. Leveraging the JavaScript Stack]. St. Petersburg, 2017. 336 p.

3. Gladkov L. A., Kureichik V. V., Kureichik V. M. *Geneticheskie algoritmy : uchebnoe posobie* [Genetic algorithms : tutorial]. 2-nd ed. Moscow, Fizmatlit Publ., 2006. 320 p.

4. Gorbunov-Posadov M. M. *Rasshiriaemye programmy* [Expandable programs]. Moscow, Poliptikh Publ., 1999. 336 p.

5. Emelianov V. V., Kureichik V. V., Kureichik V. M. *Teoriia i praktika evoliutsionnogo modelirovaniia* [Theory and practice of evolutionary modelling]. Moscow, Fizmatlit Publ., 2003. 432 p.

6. Ilin D. Iu., Gusev A. A. Provedenie vosproizvodimykh eksperimentov po otsenke effektivnosti raboty komponentov programmnogo obespecheniia [Conducting reproducible experiments to assess the effectiveness of operation of software components]. *Prikladnye issledovaniia i tekhnologii ART 2019 : sbornik trudov regionalnoy konferentsii* [Applied research and technologies ART 2019 : Proceedings of the Regional Conference]. Moscow, MTI Publ., 2019, pp. 54–56.

7. Ilin D. Iu., Nikulchev E. V., Koliasnikov P. V. Vybor tekhnologicheskikh reshenii dlya razrabotki programmnogo obespecheniia raspredelennykh informatsionnykh sistem [Choice of technological solutions for developing distributed information systems software]. *Sovremennye informatsionnye tekhnologii i IT-obrazovanie* [Modern Information Technologies and IT–Education], 2018, vol. 14, no. 2, pp. 344–354.

8. Patent US5651099A USA. Use of a genetic algorithm to optimize memory space. Inventor Shane Konsella; declarer HP Inc; current assignee Hewlett Packard Development Co LP, 1995.

9. Rutkovskaya D., Pilinskiy M., Rutkovskiy L. *Neyronnye seti, geneticheskie algoritmy i nechetkie sistemy* [Neural networks, genetic algorithms and fuzzy systems]. 2nd ed., Moscow, Goriachaia liniya – Telekom Publ., 2008. 452 p.

10. Faronov V. V. *Sozdanie prilozheniy s pomoshchiyu C#* [Building applications with C#]. Moscow, EKSMO Publ., 2008. 576 p.

11. *A comparison of the performance of a few popular javascript frameworks*. Available at: https://github.com/krausest/js-framework-benchmark (accessed 16.04.2019).

12. Alba E., Chicano J. F. Software project management with GAs. *Information Science*, 2007, vol. 177, issue 11, pp. 2380–2401.

13. Aljahdali S. H., El-Telbany M. E. Software reliability prediction using multi-objective genetic algorithm. *IEEE/ACS International Conference on Computer Systems and Applications*. Rabat, IEEE, 2009, pp. 293–300.

14. Bouchachia A. An Immune Genetic Algorithm for Software Test Data Generation. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*. Kaiserlautern, Germany, IEEE, 2007, pp. 84–89.

15. *Collection: Front-end JavaScript frameworks*. Available at: https://github.com/collections/front-end-javascript-frameworks (accessed 16.04.2019).

16. *CPU-Time Variation.* Available at: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieag200/cputvari.htm (accessed 16.04.2019).

17. Dai Y. S., Xie M., Poh K. L., Yang B. Optimal testing-resource allocation with genetic algorithm for modular software systems. *Journal of Systems and Software*, 2003, vol. 66, no. 1, pp. 47–55.

18. Deep K. et al. A real coded genetic algorithm for solving integer and mixed integer optimization problems. *Applied Mathematics and Computation*, 2009, no. 212, pp. 505–518.

19. *Difference Between Library and Framework*. Available at: https://www.c-sharpcorner.com/UploadFile/a85b23/framework-vs-library/ (accessed 16.04.2019).

20. *Documentation : TypeScript.* Available at: http://www.typescriptlang.org/docs/home.html (accessed 16.04.2019).

21. Fanjiang Y.-Y. et al. Genetic algorithm for QoS-aware dynamic web services composition. *2010 International Conference on Machine Learning and Cybernetics*, Qingdao, IEEE, 2010, pp. 3246–3251.

22. Fedosejev A. *React.js essentials*. Packt Publishing, 2015. 208 p.

23. Feldt R. Generating diverse software versions with genetic programming: an experimental study. *IEEE Proceedings – Software*, 1998, vol. 145, no. 6, pp. 228–236.

24. Filipova O. *Learning Vue.js 2 Learn how to build amazing and complex reactive web applications easily with Vue.js*. Packt Publishing Ltd, 2016. 334 p.

25. Frisbie M. *Angular 2 Cookbook*. Packt Publishing, 2017. 464 p.

26. *GrapeCity Wijmo | Take the SPEC Quiz.* Available at: https://www.grapecity.com/en/spec (accessed 16.04.2019).

27. Green B., Seshadri S. *AngularJS.* O'Reilly Media, 2013. 196 p.

28. Greer D., Ruhe G. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 2004, vol. 46, no. 4, pp. 243–253.

29. Huang S.-J., Chiu N.-H. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and Software Technology*, 2006, vol. 48, issue 11, pp. 1034–1045.

30. Huang S.-J., Chiu N.-H., Chen L.-W. Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research*, 2008, vol. 188, no. 3, pp. 898–909.

31. *InversionOfControl.* Available at: https://martinfowler.com/bliki/InversionOfControl.html (accessed 16.04.2019).

32. *Javascript Framework Comparison with Examples (React, Vue & Hyperapp).* Available at: https://hackernoon.com/javascript-framework-comparison-with-examples-react-vue-hyperapp-97f064fb468d (accessed 16.04.2019).

33. *JavaScript Frameworks, Performance Comparison*. Available at: https://www.codementor.io/aminmeyghani /javascript-frameworks-performance-comparison-ntkw9sn66 (accessed 16.04.2019).

34. *JS web frameworks benchmark – Round 7.* Available at: https://www.stefankrause.net/wp/?p=454 (accessed 16.04.2019).

35. Kumari A. C., Srinivas K., Gupta M. P. Software module clustering using a hyper-heuristic based multi–objective genetic algorithm. *2013 3rd IEEE International Advance Computing Conference (IACC)*. Ghaziabad, IEEE, 2013, pp. 813–818.

36. Lun L. et al. Coverage Criteria for Component Path-oriented in Software Architecture. *Engineering Letters*, 2019, vol. 27, no. 1, pp. 40–52.

37. Ma Y., Zhang C. Quick convergence of genetic algorithm for QoS-driven web service selection. *Computer Networks*, 2008, vol. 52, no. 5, pp. 1093–1104.

38. Macrae C. *Vue.js: Up and Running*. O'Reilly, 2017. 219 p.

39. Michael C. C., McGraw G. E., Schatz M. A. Generating software test data by evolution. *Software Engineering IEEE Transactions on*, 2001, vol. 27, no. 12, pp. 1085–1110.

40. *Mixed Integer Optimization – MATLAB & Simulink.* Available at: https://www.mathworks.com/help/gads/mixed-integer-optimization.html (accessed 16.04.2019).

41. Ouni A., Kessentini M., Sahraoui H., Hamdi M. S. The use of development history in software refactoring using a multi–objective evolutionary algorithm. *Proceedings of the 15th annual conference on Genetic and evolutionary computation (GECCO '13),* Christian Blum (Ed.). New York, ACM, 2013, pp. 1461–1468.

42. Reddy J. P., Kumanan S., Chetty O. V. K. Application of Petri Nets and a Genetic Algorithm to Multi–Mode Multi–Resource Constrained Project Scheduling. *The International Journal of Advanced Manufacturing Technology*, 2001, vol. 17, no. 4, pp. 305–314.

43. Robbestad S. A. *ReactJS blueprints*. Packt Publishing, 2016. 422 p.

44. Ruebbelke L, Ford B. *AngularJS in Action*. Manning Publications, 2015. 325 p.

45. Saha D., Mitra R. S., Basu A. Hardware software partitioning using genetic algorithm. *Proceedings Tenth International Conference on VLSI Design*. Hyderabad, India, IEEE, 1997.

46. Seshadri S. *Angular: Up and running: Learning angular, step by step*. O'Reilly Media, 2018. 312 p.

47. Shuang K. et al. TTS-Coded Genetic Algorithm for QoS-driven web service selection. *2009 IEEE International Conference on Communications Technology and Applications*. IEEE, 2009, pp. 885–890.

48. Shukla K. K. Neuro-genetic prediction of software development effort. *Information and Software Technology*, 2000, vol. 42, no. 10, pp. 701–713.

49. Stylianou C., Andreou A. S. A Multi-objective Genetic Algorithm for Software Development Team Staffing Based on Personality Types. *Artificial Intelligence Applications and Innovations. AIAI 2012. IFIP Advances in Information and Communication Technology*. Berlin, Heidelberg, Springer, 2012, vol. 381, pp. 37–47.

50. Wena F., Lin C.-M. Multistage Human Resource Allocation for Software Development by Multiobjective Genetic Algorithm. *The Open Applied Mathematics Journal*, 2008, vol. 2, pp. 95–103.

51. Flanagan D. *JavaScript: The Definitive Guide.* 4th ed. O'Reilly, 2001. 936 p.