АВТОМАТИЗАЦИЯ ПРОЦЕССА НАГРУЗОЧНОГО ТЕСТИРОВАНИЯ JSON ВЕБ-СЕРВИСА С ПОМОЩЬЮ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ

AUTOMATIZATION OF JSON WEB SERVICES PERFORMANCE TESTING USING EVOLUTIONARY ALGORITHMS

E. Provedentsev

Summary. Many web services nowadays should serve huge amount of requests simultaneously. Measuring the limit amount of requests is crucial for stable and faultless work of the web service. Such measuring requires investing much time resources and not always possible to do at the developing stage. This article proposes the way to automatize this process using evolutionary algorithms.

Keywords: performance testing, web service, evolutionary algorithms.

Проведенцев Евгений Сергеевич
Аспирант, Белорусский государственный университет информатики и радиоэлектроники
evgenprovedencev@gmail.com

Аннотация. В настоящее время множество веб-сервисов обязаны обрабатывать большое количество запросов одновременно. Для обеспечения непрерывной и безошибочной работы подобных веб-сервисов необходимо прогнозирование максимально выдерживаемой нагрузки. Подобное прогнозирование весьма трудоемко и не всегда возможно на этапе проектирования системы. В данной работе предлагается способ автоматизации данного процесса с помощью эволюционных алгоритмов.

Ключевые слова: нагрузочное тестирование, веб-сервис, эволюционные алгоритмы.

Введение

Интернет в наши дни используется повсеместно. На многочисленные веб-сервисы возложена задача удовлетворять нужды финансовых, рыночных, банковских сфер, сферы образования и здравоохранения. Число потребителей веб-сервисов растет из года в год в соответствии с ростом населения, которое имеет доступ к сети Интернет. Невозможно представить успешное существование продукта и его конкурентоспособности без тщательного планирования развития. Одним из ключевых аспектов подобного планирования является инженерия производительности. Она включает в себя прогнозирование загрузки, надежности работы продукта, его стабильность и производительность. Подобные процессы весьма трудоемки и требуют, как правило, участия квалифицированных и компетентных в этой области кадров, что влечет за собой вложение немалых денежных средств. В данной работе представлен способ автоматизации процессов прогнозирования загрузки и производительности JSON [1] веб-сервиса, запрос и ответ которого подчиняется некоторой схеме. В данной работе используется веб-сервис, вход которого описан с помощью JSON-schema [2] документа. На вход алгоритма подается JSON-schema сервиса, а также набор запросов к сервису — начальная популяция. Целью алгоритма является выведение популяции максимально соответствующей критериям отбора, а также определение величин выдерживаемой нагрузки и ожидаемой производительности.

Описание процесса тестирования

В данной работе рассмотрено определение следующих метрик: максимальная загрузка, при которой гарантируется стабильная работа сервиса и среднее времени обработки запроса сервисом.

Существует две модели нагрузки: открытая и закрытая [3]. При закрытой модели нагрузки число пользователей сервиса ограничено. Время обработки запроса не критично для пользователей. Рост времени обработки запросов сервисом влечет за собой снижение интенсивности запросов пользователей; пользователи в этой модели дожидаются обработки предыдущего запроса, прежде чем отправить новый. В качестве примера системы с такой нагрузкой можно привести банковское ПО, где пользователями будут сотрудники банка. При открытой модели нагрузки количество пользователей не ограничено. Такая модель характеризуется нагрузкой, не коррелирующей со временем обработки запросов. В качестве примера может выступать практически любое веб-приложение, находящееся в открытом доступе в сети Интернет.

В закрытой модели увеличение нагрузки осуществляется путем добавления виртуальных пользователей. Эффективность сервиса в такой модели зависит от степени параллелизма системы. В открытой модели увеличение нагрузки растет с увеличением количества запросов в секунду. Эффективность системы в этом

случае напрямую зависит от объема критического ресурса.

Алгоритм 1. Определение максимальной загрузки сервиса в закрытой модели нагрузки VU_{max}. Запускается заданное количество процессов — виртуальных пользователей VU_{start}, отправляющих запросы к сервису в синхронном режиме, т.е. каждый следующий запрос оправляется только после того, как был получен ответ на предыдущий; затем через каждый промежуток времени ΔT количество виртуальных пользователей увеличивается на определенную величину ΔVU до тех пор, пока сервис справляется с обработкой запросов в полном объеме и при этом среднее время обработки запроса не превышает некий заданный уровень. VU_{max} — число виртуальных пользователей на последней итерации.

Алгоритм 2. Определение максимальной загрузки сервиса в открытой модели нагрузки RPS_{max}. Нагрузка на сервер производится в асинхронном режиме: запросы отправляются с определенной частотой RPS, количество одновременно работающих виртуальных пользователей не важно. Так же, как и в методе определения VU_{max}, задается начальная частота запросов VU_{start}, шаг времени ΔT и шаг частоты запросов ΔRPS, такие же критерии останова. RPS_{max} — частота запросов на последней итерации.

Алгоритм 3. Определение среднего времени обработки запроса L_{avg}. Задается RPS равная половинеRPS_{max}. На каждой итерации k после промежутка времени ΔT измеряется $L_{avg\,k}$ за время $\Delta T \times k$. Критерий останова:

$$\frac{|L_{avg\,k} - L_{avg\,k-1}|}{L_{avg\,k-1}} < p \qquad (1)$$

где p — величина, заданная на старте. За L_{avg} принимаем $L_{avg\,k}$ полученное на последней итерации.

Во всех трех алгоритмах может применяться либо сценарный подход — задается набор запросов, либо hit-based подход — задается единственный запрос. В данной работе рассматривается только hit-based подход, так как любой сценарий можно свести к нескольким запускам hit-based тестирования запросов из сценария, что позволяет более точно интерпретировать результаты тестирования.

Применение эволюционного подхода

Определим сущности эволюционного алгоритма в контексте нагрузочного тестирования веб-сервиса. Особью (хромосомой) назовем отдельный запрос к сервису. В качестве популяции будет выступать набор различных запросов. Основной целью эволюции является поиск запроса, который быстрее всего утилизирует критический ресурс системы. Это связанно с тем, что производительность системы масштабируется по производительности самого «узкого» элемента системы (bottleneck), использующего критический ресурс [4]. Следовательно в качестве фитнесс-функции на этапе определения VU_{max} необходимо использовать функцию обратно пропорциональную VU_{max}, на этапе определения RPS_{max} — функцию обратно пропорциональную RPS_{max}, а на этапе определения L_{avg} функцию пропорциональную L_{avg}.

Алгоритм 4. Алгоритм применения эволюционного подхода.

1. Генерируется популяция — определенное число запросов, соответствующих заданной схеме. Для ускорения работы алгоритма запросы могут задаваться вручную, исходя из эвристических предположений.

2. Для каждого запроса запускается соответствующий алгоритм из предыдущего раздела.

3. Для каждого запроса вычисляется значение фитнесс-функции.

4. Если значение фитнесс-функции лучшего запроса из текущей популяции α_i не превышает значение фитнесс-функции лучшего запроса из предыдущей популяции α_{i-1}, то метрика, вычисленная для α_{i-1} (VU_{max}, RPS_{max} или L_{avg}), является результатом работы алгоритма, а запрос α_{i-1} может использоваться для тестирования продукта во время дальнейшей работы над производительностью.

4. Генерация следующей популяции.

5. Возврат к п. 2

Кодирование JSON запроса. Операторы.

Кодирование происходит следующим образом. Все поля JSON запроса достижимые через обращение только к ключам объекта добавляются в словарь, ключом при этом являются все ключи, через которые было достигнуто поле, записанные через точку.

Пример исходного запроса:

```
{
«A»: true,
«C»: {
«D»: null,
«E»: {
«F»: 2
```

```
},
«G»: [
{«r»: 1, «q»: 2},
{«s»: 1, «t»: 2}
]
}
}
```

Результирующий словарь:

```
{
«A»: true,
«C.D»: null,
«C.E.F»: 2,
«C.G»: [
{«r»: 1, «q»: 2},
{«s»: 1, «t»: 2}
]
}
```

Каждому ключу в словаре соответствует значение одного из пяти типов: простые типы — null, float, Boolean, string и сложный тип array; если значение простого типа, то запись рассматривается как отдельный ген, если тип значения array — запись является набором генов.

Мутация. В кодированном запросе каждое поле с вероятностью P_{mf} принимает случайное значение из множества действительных значений для типа данного поля; поля типа array с вероятностью P_{mf} меняют свою длину: теряют один элемент с вероятностью 0.5, если длина массива не меньше минимального количества элементов массива, иначе получают еще один элемент, поля которого проинициализированы случайными значениями.

Рекомбинация. В данной работе применялась одноточечная рекомбинация [5]. В качестве потомков копируются два JSON запроса родителя. Ключи простых полей кодированных структур родителей записываются в отдельный список в случайном порядке. Выбирается точка скрещивания. Затем поля первого родителя, соответствующие ключам из списка до точки скрещивания, присваиваются второму потомку, а поля второго родителя, соответствующие ключам после точки скрещивания, присваиваются первому потомку. Массивы скрещиваются следующим образом: элементы массивов обоих родителей, соответствующих определенному ключу, записываются в общий массив в случайном порядке, затем половина элементов общего массива присваивается первому потомку, вторая половина — второму.

Тестовый стенд

В качестве тестового стенда был использован компьютер Raspberry Pi Model B Rev 2, с установленной на него ОС Raspbian. В качестве объекта тестирования был использован тестовый веб-сервис (далее «мишень»). Мишень представляет собой веб-сервис, принимающий HTTP запросы с параметрами, указанными в теле запроса в виде JSON документа. Каждый запрос инициирует на мишени выполнение двух алгоритмов: задача о ранце 0/1 [6] и решето Эратосфена. Параметры, переданные в запросе, используются как входные данные для алгоритмов. Список параметров и их граничные значения описаны следующей JSON-схемой:

```
{
«type»: «object»,
«properties»:{
«Size»:{
«type»: «integer»,
«maximum»: 120,
«minimum»: 40
},
«NotPrecise»:{
«type»: «boolean»
},
«EratOptimized»:{
«type»: «boolean»
},
«EratTop»:{
«type»:»integer»,
«maximum»: 1000000,
«minimum»: 200000
},
«KnapN»:{
«type»:»integer»,
«maximum»: 100,
«minimum»: 0
},
«Items»:{
«type»: «array»,
«minItems»: 2,
«maxItems»: 16,
«items»:{
«type»: «object»,
«properties»:{
«Weight»:{
«type»: «integer»,
«maximum»: 30,
«minimum»: 1
},
«Value»:{
«type»: «integer»,
«maximum»: 100,
«minimum»: 1
}
},
«required»:[
«Weight»,
«Value»
]
```

```
}
}
},
«required»:[
«Size»,
«Items»
]
}
```

Size — задает размер рюкзака, принимает значения от 40 до 120.

KnapN — на основании данного параметра вычисляется количество повторений алгоритма рюкзака по формуле 100 — KnapN, принимает значения от 0 до 100.

Items — набор предметов для упаковки в рюкзак, массив от 2 до 16 элементов, каждый из которых состоит из двух параметров:

Weight — вес предмета, принимает значения от 1 до 30;

Value — цена предмета, принимает значения от 1 до 100.

NotPrecise — принимает значения true, false, null; если значение параметра true, то применяется оптимизация алгоритма упаковки рюкзака.

EratTop — число, меньше которого необходимо обнаружить все простые числа, принимает значения от 200000 до 1000000.

EratOptimized — принимает значения true, false, null; если значение параметра true, то применяется оптимизация алгоритма решето Эратосфена.

Подбор граничных значений параметров осуществлялся по результатам профилирования мишени по памяти и времени CPU. Значения были выбраны таким образом, чтобы затраты по памяти и времени CPU алгоритмов были приблизительно равны между собой, а также были существенны по сравнению с общими затратами на обработку запросов. Также по результатам профилирования были выбраны эталонные значения параметров: значения, при которых затраты ресурсов максимальны. KnapN — 0; чем меньше данный параметр, тем больше раз выполнится упаковка рюкзака. Items — 16. В связи с тем, что алгоритм упаковки рюкзака представляет собой неоптимизированный перебор всех вариантов, то чем больше размер рюкзака и чем меньше вес предметов, тем дольше будет работать алгоритм. Size — 120. Weight — 1. Value — 1. NotPrecise — false. EratTop — 1000000. EratOptimized — false. В данном примере при использовании запроса с эталонными значениями затраты на выполнение алгоритмов составили 95–98% от общих затрат на обработку запроса, при этом затраты делятся приблизительно поровну между двумя алгоритмами.

Эксперименты

В ходе экспериментов был исследован алгоритм определения максимальной загрузки сервиса в открытой модели нагрузки. Было произведено сравнение и анализ результатов тестирования, проведенного тремя подходами, различающимися алгоритмом генерации следующего поколения. *В первом подходе* к текущему поколению добавлялись n (размер популяции) новых особей, путем скрещивания n пар особей, выбранных турнирной селекцией [5] из текущего поколения, затем каждая новая особь с вероятностью 25% проходила мутацию, после чего n лучших особей становились новым поколением. Отличие *второго подхода* заключается в том, что каждая пара родителей скрещивалась с вероятностью 50%. В *третьем подходе* путем турнирной селекции отбиралось n особей, каждая из которых копировалась и проходила мутацию, оператор скрещивания не применялся.

В ходе экспериментов критерий останова Алгоритма 4 был изменен. Если на протяжении 10 поколений значение ф.ф. лучшей особи не меняется, то считается, что алгоритм достиг точки останова, а данная особь и ее ф.ф. является результатом работы алгоритма. Это связано с тем, что для данного конкретного примера описанных выше мишени и пулемета значение ф.ф. особей проявило высокую вероятность остановки в локальном максимуме. По той же причине вероятность срабатывания мутации гораздо выше значений свойственных для эволюционных алгоритмов в целом. Подобные изменения позволяют алгоритму эффективнее преодолевать локальные максимумы ф.ф.

Целью алгоритма в контексте вышеописанного тестового стенда является нахождение запроса со значениями параметров, максимально приближенными к эталонным, за минимальное время.

Для сравнения были использованы результаты 5 прогонов каждого подхода.

Как видно из графика на рисунке 1, за 10 поколений параметр KnapN в большинстве случаев стал оптимальным (оптимальное значение — 0), только в одном из прогонов параметр остался в локальном оптимуме. Параметр EratTop в среднем остановился в 30–35% от оптимума (оптимальное значение — 1000000). В первом прогоне оба параметра остановились дальше от оптимума, чем в остальных прогонах; скорее всего это вызвано утилизацией ресурсов сторонними процессами. В целом данный подход показал свою эффективность при нахождении оптимальных значений численных параметров. Из минусов данного подхода стоит отметить плохую способность покидать локальный оптимум.

В рамках данного исследования второй подход показал результаты близкие к результатам первого подхода. В теории больший шанс мутации должен обеспечить

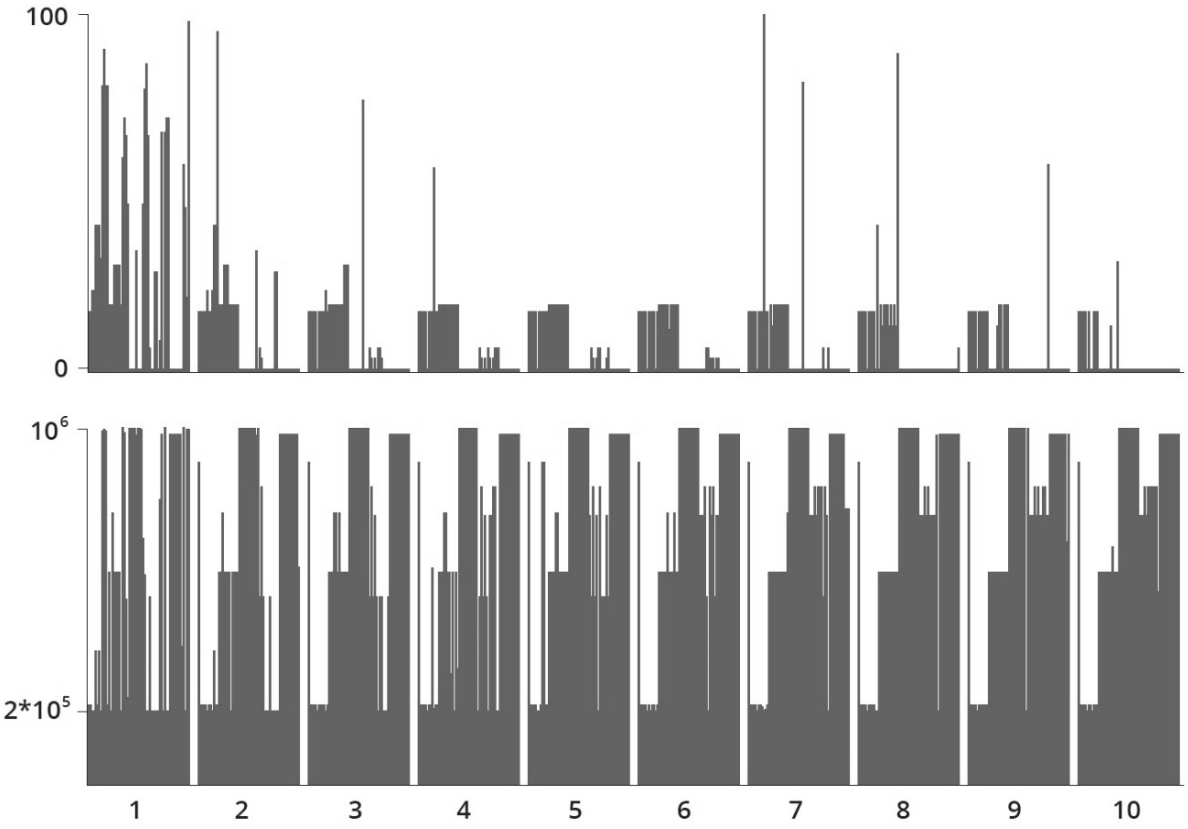

Рис. 1. Значения параметров 10 первых поколений, 1 подход, 100% скрещивание, 25% мутация. (Сверху — KnapN, снизу — EratTop)

улучшение способности покидания локального оптимума. Данный эффект отчасти можно наблюдать на графике параметра EratTop: по сравнению с графиком первого подхода, на графике второго подхода видно более сильное отклонение значений от среднего значения для каждого отдельно взятого прогона.

Результаты тестирования с помощью третьего подхода оказались наиболее близки к оптимуму. 100% шанс мутации гарантирует покидание локального оптимума через минимальное число поколений. Однако улучшение результата по сравнению с первыми двумя подходами недостаточно, чтобы утверждать неэффективность применения оператора скрещивания, и требует более детального сравнения на большем наборе данных.

Все три подхода продемонстрировали также способность находить оптимальные значения параметров в массиве. Несмотря на то, что оптимальные значения параметров Items.Weight и Items.Value не были спрогнозированы по результатам профилирования, в ходе экспериментов значения этих параметров стремились к 1. Как выяснилось в дальнейшем, из-за применения неэффективного алгоритма упаковки рюкзака, 1 — действительно является оптимальным значением для данных параметров.

На тестирование с помощью каждого подхода было затрачено в среднем 2–3 часа на один прогон (10–15 часов на один подход).

Выводы

Данное исследование показало возможность использования эволюционных алгоритмов для автоматизации нагрузочного тестирования при использовании показателей утилизации оперативной памяти и CPU в качестве опорных метрик. Эффективность данного метода напрямую коррелирует с количеством времени, затраченным на тестирование. В данном исследовании на каждый подход было затрачено до 15 часов (2 человеко-дня). Подобный метод будет полезен в случае тестирования уже разработанного веб-сервиса с нуля, т.е. если нагрузочное тестирование не проводилось в процессе разработки, и/или тестирование поручается отделу, не вовлеченному в разработку, так же подобный метод полезен в случае тестирования веб-сервиса без доступа к его реализации (black box).

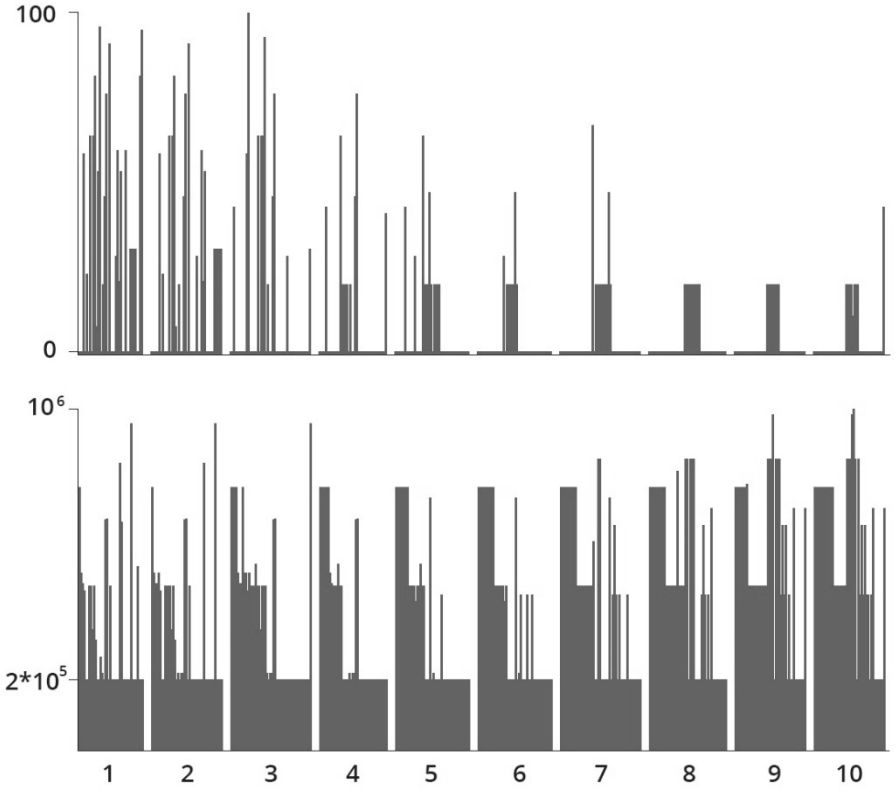

Рис. 2. 2 подход, 50% скрещивание, 50% мутация

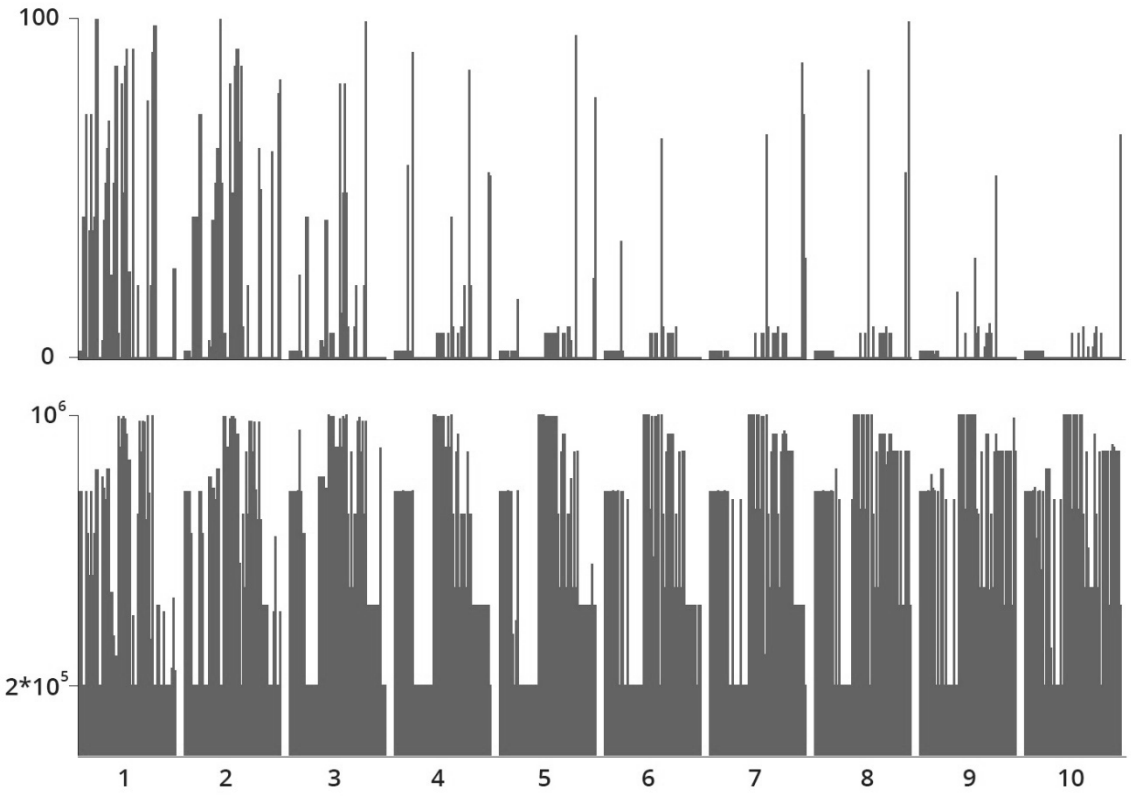

Рис. 3. 3 подход, без скрещивания, 100% мутация

ЛИТЕРАТУРА

1. T. Bray The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force, March 2014, ISSN: 2070–1721 https://tools.ietf.org/html/rfc7159.
2. A. Wright, H. Andrews JSON Schema: A Media Type for Describing JSON Documents. Internet Engineering Task Force, April 15, 2017. https://tools.ietf.org/html/draft-wright-json-schema-01.
3. Wescott, Bob. The Every Computer Performance Book, CreateSpace, 2013. ISBN1482657759.
4. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд.»Вильямс», 2010.— 464 с.— (Addison-Wesley Signature Series).— 1000 экз.— ISBN978–5–8459–1625–9.
5. 2. Ю. А. Скобцов, Д. В. Сперанский. Эволюционные вычисления. Москва: ИНТУИТ, 2014.
6. Silvano Martelo, Paolo Toth. Knapsack problems. Great Britain: Wiley, 1990. ISBN0–471–92420–2.

Журнал «Современная наука: актуальные проблемы теории и практики»

Белорусский государственный университет информатики и радиоэлектроники