В результате применение генетического алгоритма была получена следующая топология сети (рисунок 3).

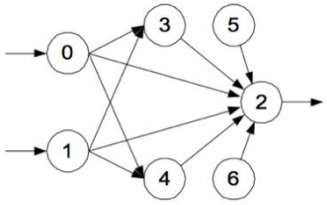

Рис. 3 – Пример автоматически сгенерированной топологии сети

Минимальная сеть, которая способна решать поставленную задачу, состоит из 4-х нейронов и 5-и связей, однако стоит отметить, что полученная сеть была сгенерирована автоматическим образом, что является неплохим результатом

Для дальнейшего улучшения результатов автоматического построения можно провести дополнительные эксперименты по выбору способа кодирования топологии сети. Для этого можно применить косвенный способ кодирования, где строятся определенные грамматики топологических структур или кодируются отдельные блоки.

Список использованных источников:

1. N.J. Radcliffe, Genetic Neural Networks on MIMD computers / Radcliffe N.J., Davis L // University of Edinburgh, Edinburgh, England, 1990 – 65p.
2. K. O. Stanley, Evolving Neural Topologies through Augmenting Topologies / Stanley K. O., Miikkulainen R. // Evolutionary Computation, The MIT Press, 2002. – № 10(2). – p. 99-127.
3. J. R. Koza, Genetic generation of both the weight and architecture for a neural network / Koza J. R., Rice J. P. // International Joint Conference on Neural Networks. 1991. – Vol. 2, P. 397–404.

ЭВОЛЮЦИОННЫЙ АЛГОРИТМ ГЕНЕРАЦИИ ТЕСТОВЫХ СЦЕНАРИЕВ JSON ВЕБ-СЕРВИСОВ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Проведенцев Е. С.

Скобцов В. Ю. – кандидат технических наук

В практике современных информационных технологий и программной инженерии широкое распространение получили веб-сервисы, принимающие в качестве параметров данные в JSON формате. Запросы и ответы подобных сервисов могут быть описаны с помощью JSON-schema.

Для функционального тестирования JSON-сервисов зачастую применяется ручная разработка тестовых сценариев. Автоматизация разработки тестового сценария для JSON-сервиса осложнена, большим количеством входных и выходных параметров, а также возможными зависимостями логики срабатывания одного параметра от другого [1]. Для решения трудно формализуемых задач большой размерности широкое применение получили эволюционные алгоритмы, являющиеся одним из перспективных направлений исследований в области интеллектуальных систем, задач поиска, интеллектуальной обработки данных [2]. В частности, эволюционные алгоритмы эффективно используются в области автоматизации тестирования программного обеспечения [3]. Поэтому в данном исследовании предлагается эволюционный алгоритм генерации тестовых сценариев для подобных веб-сервисов.

В данной работе предлагается алгоритм, использующий JSON-schema для автоматизированного создания тестового сценария JSON-сервиса. Целью данного алгоритма является генерация тестового сценария, обнаруживающего максимальное количество ошибок и, в то же время, содержащего минимальное количество тестовых случаев (test case). В основе данного алгоритма лежит представление тестового случая в качестве экземпляра, каждое поле запроса, которого представляет собой отдельный ген.

Алгоритм работает следующим образом:

1. Генерируется множество-популяция случайных запросов, заполненных в соответствии с предоставленной JSON-схемой. Наполнение запроса должно быть максимальным, то есть каждому полю, по возможности, должно быть присвоено значение. В зависимости от типа поля выбирается стратегия заполнения. Для числовых типов могут выбираться граничные значения, для строковых – значения из предопределенного списка, который составляется в зависимости от специфики программного кода сервиса. Цель данного запроса – задействовать как можно больше ветвей кода, повысив вероятность выявления ошибки. В случае, если данный пункт выполняется не первый раз, то необходимо убедиться, что запрос не

совпадает с уже выполненными на предыдущих итерациях.

2. Сгенерированные запросы отправляются сервису на выполнение.

3. Ответ от сервиса проверяется на соответствие JSON-схеме.

4. Если ответ соответствует JSON-схеме, значит ошибка не была выявлена, и необходимо произвести мутацию запроса. Мутация может выполняться как внутри поля, так и над структурой всего запроса целиком. Возврат к пункту 2.

5. Если ответ не соответствует JSON-схеме, считается, что была обнаружена ошибка. Ошибочный ответ запоминается для дальнейшей обработки.

6. Запускается алгоритм поиска гена, который вызывает ошибку. Для этого из запроса по очереди удаляются все гены (поля) и отправляются на выполнение. Если после удаления очередного гена запрос выполнился без ошибки, то удаленный ген либо сам, либо в комплексе с другими генами является причиной ошибки, производится мутация до тех пор, пока не будет выявлен набор генов, являющийся причиной ошибки. Данный набор добавляется в массив для дальнейшей обработки. Возврат к пункту 1.

В качестве критерия останова цикла 1 – 6 могут выступать временные рамки либо достижение определенного количества мутаций запроса, не приведших к ошибке. После цикла 1 – 6 из наборов генов, полученных в 6 пункте, генерируется минимальное число запросов, соответствующих JSON-схеме. Полученный набор тестовых случаев является результатом работы данного алгоритма.

Подобный алгоритм может быть полезен при генерации smoke-тестов, так как при поиске ошибок задействуется максимально возможное число параметров, следовательно, покрытие функционала тестами стремится к 100%. Значительную пользу может принести набор генов, вычисленный в 6 пункте, являющийся причиной ошибки. Программисту будет значительно проще локализовать ошибку в коде, имея запрос состоящий только из тех полей, которые необходимы для воспроизведения ошибки.

Данный алгоритм может быть использован для регрессионного тестирования: тестирование можно считать успешным при безошибочном выполнении изначального набора запросов и последующих, мутировавших определенное число раз.

Алгоритм можно рассматривать в контексте стратегии мутации при генерации более сложных тестовых сценариев (например, CRUD сценарий) с помощью эволюционных алгоритмов.

Список использованных источников:

1. Тестирование программного обеспечения / Сэм Канер, Джек Фолк, Енг Кек Нгуен / ДиаСофт – Москва, 2001 – 544 с.
2. Ю.А.Скобцов, Д.В.Сперанский. Эволюционные вычисления. Москва: ИНТУИТ, 2014.
3. T. Mantere, J.T. Alander Evolutionary software engineering, a review // Applied Soft Computing 5 (2005) pp.315–331.

АНАЛИЗ ЭФФЕКТИВНОСТИ РАБОТЫ СИСТЕМЫ РАСПРЕДЕЛЕННОЙ ПЕРЕДАЧИ ДАННЫХ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Разумов Е.В.

Иванюк А.А. – д-р. техн. наук

Одним из возможных методов, по средствам которого можно достичь большую устойчивость к MITM-атакам, является метод построения системы распределенной передачи данных по средствам peer-to-peer сети. Основная идея, заложенная в данную систему, заключается в разбиении передаваемой информации на блоки, шифровании каждого из этих блоков и последующей передаче их через отдельный промежуточный узел.

Оценивать эффективность работы такой системы целесообразно по нескольким позициям: скорости работы системы в целом, криптостойкости такого способа пересылки данных, зависимости от внешних факторов, преимуществах перед существующими решениями.

С точки зрения скорости, передача данных между адресатами будет значительно медленнее, чем в системах, которые передают данные от отправителя к получателю напрямую. Зависимость времени передачи данных между адресатами от количества пройденных промежуточных узлов не является линейной. Это объясняется в первую очередь тем, что в реальной системе кроме затрат времени, которые необходимы непосредственно для операций шифрования/расшифровывания, необходимо еще время на передачу блоков зашифрованных данный от одного узла к другому. И это время не является постоянной величиной и зависит от качества сети.

Следующей позицией, по которой можно оценить эффективность разработанной системы, является скорость передачи данных между адресатами в зависимости от числа разбиений исходных данных при условии, что данные передаются через один промежуточный узел. Анализируя результаты, полученные практическим путем, можно сделать вывод, что с увеличением числа разбиений скорость работы системы в целом уменьшается. Причиной тому являются следующие факторы:

- скорость работы сети на каждом из узлов;