Для предложения «На улице идёт сильный дождь» можно выделить отношения, представленные на рисунке 1.

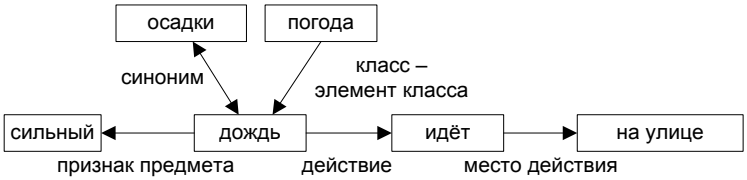

Рис. 1 – Пример семантической сети

В соответствии с алгоритмом, в сеть последовательно добавлены слова «дождь», «осадки», «погода», «идёт», «сильный», «на улице». Словарь «класс-элемент класса» может быть создан пользователем-экспертом либо вычеркнут из рассмотрения в зависимости от специфики решаемой задачи. Следует отметить, что предлог «на» перед существительным («на улице») означает указание места. Для повышения точности определения данного отношения можно использовать некоторый алгоритм вычисления падежа существительного.

Покажем, что данная сеть не только довольно подробно учитывает смысл предложения, но и содержит достаточно информации для ответа на вопрос «Где идут осадки?».

«Где» - вопрос определения места. «Осадки» – это синоним слова «дождь». «Идёт» - действие, выполняемое дождём. Место, связанное с «идёт» - на улице.

Таким образом, информации, отражённой в данной сети, достаточно, чтобы в ответ на вопрос «Где идут осадки?» можно было сформировать предложение «Дождь идёт на улице». Для ответа на вопрос понадобились отношения «Синоним», «Действие», «Место действия». Следовательно, данных типов отношений может быть достаточно для ответа на вопрос определения места.

Итак, построение рассмотренной сети является вполне автоматизируемой задачей. Предложенная семантическая сеть способна довольно подробно отразить смысл текста. Благодаря достаточно большому количеству типов отношений она позволяет сформировать ответ на вопрос. Тем не менее, её реализация представляет собой довольно сложную задачу.

Список использованных источников:

1. Базы знаний экспертных систем. [Электронный ресурс]. – Электронные данные. – Режим доступа: http://daxnow.narod.ru/index/0-18. Дата доступа : 25.03.2017.
2. Гаврилова, Т.А. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский – СПб. : Питер, 2000. – 384 с.
3. Масленникова, О.Е. Основы искусственного интеллекта [Электронный ресурс] : учеб. пособие / О.Е. Масленникова, И.В. Гаврилова. – М.: ФЛИНТА, 2013. – Режим доступа: http://search.rsl.ru/ru/record/01007574162.
4. Потараев, В.В. Метод классификации текстовой информации на основе семантической сети / В.В. Потараев // Апробация.– 2016. – №1. – С. 56-58.
5. Рахимова Д. Р. Построение семантических отношений в машинном переводе // Вестник КазНУ им. аль-Фараби. Серия «Математика, механика и информатика». – Алматы, 2014. – №1. – С. 90-101.

ЭВОЛЮЦИОННЫЙ ПОДХОД В АВТОМАТИЗАЦИИ ПРОЦЕССА НАГРУЗОЧНОГО ТЕСТИРОВАНИЯ

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

Проведенцев Е. С.

Скобцов В. Ю. – канд. техн. наук

Невозможно представить успешное существование современного программного продукта и его конкурентоспособности без тщательного планирования развития. Одним из ключевых аспектов подобного планирования является инженерия производительности. Она включает в себя тестирование и прогнозирование загрузки, надежности работы продукта, его стабильности и производительности. Подобные процессы весьма трудоемки и требуют, как правило, участия квалифицированных и компетентных в этой области кадров, что влечет за собой вложение немалых денежных средств. Поэтому важной задачей является разработка алгоритмов и методов, позволяющих автоматизировать процессы тестирования и прогнозирования загрузки и производительности веб-сервиса, а также построение запросов, которые могут применяться для дальнейшего мониторинга производительности веб-сервиса.

В данной работе представлен способ автоматизации процессов тестирования и прогнозирования максимальной нагрузки и производительности веб-сервиса, а так же построение тестовых сценариев для мониторинга производительности веб-сервиса. На вход алгоритма подается JSON-schema сервиса, а также набор запросов к сервису – начальная популяция. Целью алгоритма является выведение популяции

запросов, максимально утилизирующих критические ресурсы системы, а также определение величин максимальной нагрузки и ожидаемой производительности.

Определим сущности эволюционного алгоритма в контексте нагрузочного тестирования веб-сервиса. Особью (хромосомой) назовем отдельный JSON-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. Следовательно, в качестве фитнесс-функции на этапе определения максимальной загрузки сервиса в закрытой модели (далее *VUmax)* необходимо использовать функцию обратно пропорциональную *VUmax*, на этапе определения максимальной загрузки сервиса в открытой модели (далее *RPSmax)* – функцию обратно пропорциональную *RPSmax*, а на этапе определения среднего времени обработки запроса (далее *Lavg)* функцию пропорциональную *Lavg*.

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 (описаны далее), алгоритм 1 – для нахождения *VUmax,* алгоритм 2 – для нахождения *RPSmax,* алгоритм 3 – для нахождения *Lavg.*
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)* на протяжении определенного количества итераций, то метрика, вычисленная для *α(i-1)* (*VUmax*, *RPSmax* или *Lavg)*, является результатом работы алгоритма, а запрос *α(i-1)* может использоваться для тестирования продукта во время дальнейшей работы над производительностью.
4. Генерация следующей популяции. Последовательное выполнение эволюционных операторов.
5. Возврат к п. 2

Алгоритм 1. Определение *VUmax*. Запускается заданное количество процессов – виртуальных пользователей *VUstart*, 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 *dT* количество виртуальных пользователей увеличивается на определенную величину *dVU* до тех пор, пока сервис справляется с обработкой запросов в полном объеме и при этом среднее время обработки запроса не превышает некий заданный уровень. *VUmax* – число виртуальных пользователей на последней итерации.

Алгоритм 2. Определение *RPSmax*. 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 *VUmax*, задается начальная частота запросов *RPSstart*, шаг времени *dT* и шаг частоты запросов *dRPS*, такие же критерии останова. *RPSmax* – частота запросов на последней итерации.

Алгоритм 3. Определение *Lavg*. Задается *RPS*равная половине *RPSmax*. На каждой итерации *k* после промежутка времени *dT* измеряется *Lavgk* за время *dT * k*. Критерий останова:

$$\frac{|L_{avgk} - L_{avgk-1}|}{L_{avgk-1}} < p \qquad (1)$$

где *p* – величина, заданная на старте. За *Lavg* принимаем *Lavgk*полученное на последней итерации.

Данный алгоритм был протестирован на веб-сервисах с 5-15 параметрами различных типов. Эффективность алгоритма напрямую зависит от выбора ряда характеристик: вероятности срабатывания эволюционных операторов, последовательности их выполнения, способа задания начальной популяции, стратегии мутации различных типов данных. Во время тестирования данные характеристики выбирались эвристическим методом. Алгоритм показал эффективность в нахождении предельно допустимой нагрузки на веб сервис. Значения близкие к эталонным были получены в среднем на 6-7 итерации алгоритма. Исследование показало, что время, затраченное на тестирование веб-сервиса с помощью подобного алгоритма, может быть значительно сокращено по сравнению с тестированием того же веб-сервиса производимое вручную.

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.