УДК 004.054

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ К ГЕНЕРАЦИИ ТЕСТОВ ДЛЯ ТЕСТИРОВАНИЯ ПРОГРАММ ПО КРИТЕРИЮ С1

Ерёмичев В. В., Савкова Е.О.
Донецкий национальный технический университет,
кафедра автоматизированных систем управления
v.eremichev@gmail.com

***С**татья подробно рассматривает этапы подготовки программного кода к тестированию ветвей, а так же алгоритм преобразования задачи тестирования в задачу оптимизации. Описан процесс составления генетического алгоритма, решающего оптимизационную задачу, для формирования значений, удовлетворяющих условиям прохождения программы по выбранной ветви.*

Рассмотрим подробно, каким образом можно применять эволюционный метод в задачах тестирования. Тут нам могут помочь генетические алгоритмы, разработанные для решения задач оптимизации. Следовательно, для того, чтобы заставить работать генетический алгоритм для наших целей, нужно задачу свести к задаче оптимизации.

Поскольку целью работы является создание группы определенных значений, так называемого тестового набора, который бы обеспечивал прохождение той или иной ветви программы на выбор, то рассмотрим конкретно ветвь программы. Ветвью программного обеспечения называется часть кода, которая расположена:

1. От начала программы до первого оператора ветвления.
2. Заключена между двумя операторами ветвления.
3. От последнего оператора ветвления до конца программы.

Первый и последний вариант для нас не представляют особого интереса, поскольку по этим ветвям проходит выполнение программы в любом случае. Второй вариант, когда ветвь представляет собой фрагмент кода между двумя операторами ветвления, интересен тем, что нужно при одном наборе данных существует большая вероятность того, что программа не пройдет по большинству ветвей, следовательно, мы не сможем быть уверены в их работоспособности.

Главным образом нужно обращать внимание на условие, определяющее будет ли пройдена данная ветвь программы или нет. Ниже приведен пример простой программы, на котором мы рассмотрим все дальнейшие операции.

```
void Method (int x)
     {
0          if (x>17)
1               x = 17-x;
2          if (x==-13)
3               x=0;
4     }
```

Для того, чтобы разобраться в ветвях программы, первым делом следует построить управляющий граф, который наглядно сможешь объяснить каким образом можно попасть в ту или иную ветвь. Управляющий граф программы для данного кода приведен на рис. 1.

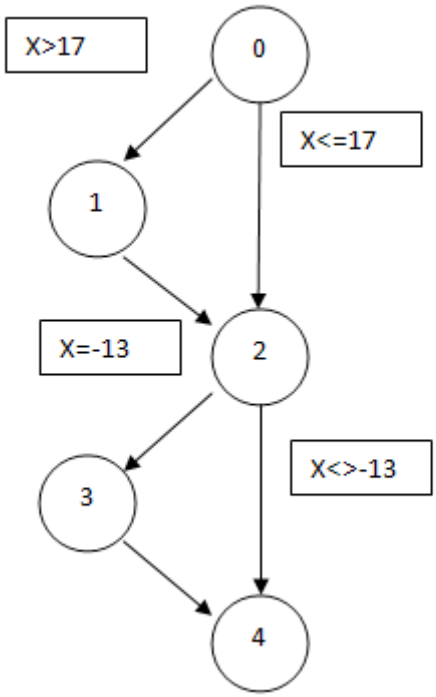

Рисунок 1. Управляющий граф программы для данного кода

Как можно понять, вершины графа представляют собой операторы программы. Они пронумерованы в соответствии со строками. Между вершинами графа есть направленные переходы. Если из вершины исходит два ребра, то между узлами осуществляется благодаря граничным условиям, которые являются аналогией условий операторов ветвления самой программы. Они так же указываются на графе. Таким образом, у нас есть возможность контроля всех ветвей полученной структуры.

Теперь можно приступать к основной задаче – тестированию ветвей. Ее можно разделить на два логических этапа:

1. Разбор условия.
2. Формирование подходящих значений, которые бы активировали граничное условие для перехода по нужной нам ветви.

Первый этап – это подготовительный этап. В нем происходит синтаксический анализ условия, который помогает выделить само налагаемое условие, переменную, значение которой нам нужно будет подобрать для достижения цели и пределы того самого подбираемого значения.

К примеру возьмем вершину ноль. Условие заключенное в переходе $X > 17$. Переменная X имеет целый тип int, шестнадцатиразрядный, имеющий диапазон значений [-32 768;32 767]. Следовательно при генерации значений для прохождения ветви {0-1-2}, значение переменной должно быть в пределах от 18 до 32 767 и от -32 768 до 17 для прохождения по ветви {0-2}.

После того, как этот этап пройден, мы располагаем всеми нужными данными для

осуществления этапа номер два.

Второй этап заключается в нахождении значений, которые соответствуют условиям, описанным выше. С этой целью используется эволюционный алгоритм. При проектировании генетического алгоритма, первым делом следует определиться со структурой хромосомы, как с основным объектом потенциального решения поставленной задачи. В программе мы имеем набор внешних переменных, задействованных на выбранной последовательности переходов. Этот набор можно рассматривать как вектор значений внешних переменных $<x_1, x_2, \ldots, x_n>$, где x_i – значение внешней переменной, а n – число внешних переменных для этого пути. В нашем конкретном случае набор состоит из 1 переменной X. В более сложных условиях, например $((A > B)\&\&(A < C))$, входными значениями будут эти самые A, B и C. Этот набор параметров и будет составлять последующую хромосому. При формировании начальной популяции следует обратить внимание на ограничения на значения элементов вектора.

Следующим шагом следует определиться с функцией приспособленности. Она является функцией оценки хромосомы, показывающей степень приспособленности хромосомы, т.е. на сколько хорошо описываемый хромосомой фенотип решает поставленную задачу. На вход функция приспособленности принимает вектор значений(хромосому) и выдает число, характеризующее приспособленность этого вектора для выполнения заданного пути. Чем меньше это число, тем больше подходит данный вектор значений. Таким образом, задача поиска подходящего набора значений внешних переменных сводится к задаче оптимизации, где требуется найти вектор, которому соответствует минимальное значение функции приспособленности.

Однозначного способа определения функции приспособленности не существует. Тем не менее, существует критерий, который называется расстоянием до условия (Branchdistance), для определения приспособленности хромосом. Расстояние до условия позволяет оценить, насколько близка была данная хромосома к выполнению конкретного условия, которое на практике не было выполнено. Например, для условия $A > B$, расстояние до условия будет вычисляться по формуле $|A - B|$. Чем меньше значение $|A - B|$, тем ближе значение A к B и тем ближе хромосома к тому, чтобы это условие было выполнено. Если условие выполнено, то расстояние до условия ноль. При этом расстояние до условия можно представить как (1)

$$(A \geq B) = \begin{cases} 0, A \geq B \\ |A - B|, A < B \end{cases} \quad (1)$$

Функция приспособленности, основанная на использовании критерия расстояние до условия, успешно применяется для поиска значений для выполнения пути в расширенном конечном автомате, причем эти значения являются граничными для условия, что увеличивает уверенность в правильности тестируемой программы. Для последовательности переходов может быть задано большое число условий, поэтому расстояние до условия всего пути необходимо вычислять по отдельности, рассматривая условия на каждом переходе этого пути. Каждый переход описывается набором параметров:

1. Событие, по которому этот переход может произойти.
2. Охранное условие, которое должно быть выполнено для совершения перехода.

3. Действия на переходе: вызов методов объектов управления, получение значений внешних переменных из среды или изменение значений переменных модели.
4. Предусловия перехода, включающие в себя требования спецификации программы, которые должны быть выполнены для выполнения перехода, и требования спецификации объектов управления, которые должны быть выполнены для вызова методов объектов управления, задействованных при переходе.
5. Постусловия перехода, включающие в себя требования спецификации программы и ее объектов управления.

Таким образом, даже в рамках одного перехода может быть задействовано большое число условий. Для более точного вычисления расстояния до условия перехода в работе каждый переход разбивается на несколько меньших шагов.

После этого исходная последовательность переходов рассматривается как последовательность шагов. Оценка приспособленности всей последовательности шагов вычисляется как сумма оценок для каждого шага по-отдельности. Каждый шаг оценивается по формуле вычисления расстояния для условия. Отметим, что шаги выполняются последовательно и что выполненность шагов в начале пути важнее, чем в его конце. Например, если выполнены условия всех шагов, кроме первого, то сумма расстояний до условия будет небольшой, так как для всех, кроме первого шага, это значение будет равно нолю. На практике эта хромосома не позволяет пройти ни одного шага, так как для того, чтобы успешно пройти второй шаг, необходимо выполнить все условия на первом шаге. Поэтому в предложенном подходе расстояния до условия шагов суммируются с учетом местоположения этих шагов в пути – используется взвешенная сумма (2) приспособленности пути

$$\sum_{i=0...m-1} f_i * d_i, \qquad (2)$$

где m – число шагов в пути, $m = n*3$, здесь n – число переходов в пути; f_i – расстояние до условия для i-го шага; d_i – вес i-го с шага, $d_i = (m - i)^2$.

Если для одного шага задано несколько условий, то расстояние до условия этого шага вычисляется как сумма расстояний до условия каждого из этих условий.

На следующем шаге следует определиться каким образом мы сможем улучшать популяцию, т.е. определиться с операторами кроссинговера и мутации. Поскольку хромосома представляет собой вектор значений, может быть использован классический одноточечный или многоточечный кроссинговер, при котором нужно выбрать точку k - точку пересечения двух хромосом родителей. Эта точка выбирается случайно из интервала [1..n], где n - длина хромосом родителей. После чего потомок A будет наследовать с 0 по k ген родителя 1 и с k+1 по n ген родителя A, а потомок B наследует оставшиеся части обоих родителей, как показано на рис. 2. Мутация же представляет собой случайное изменение гена. Новое значение гена не должно выходить за пределы разрешенного интервала значений генов хромосомы.

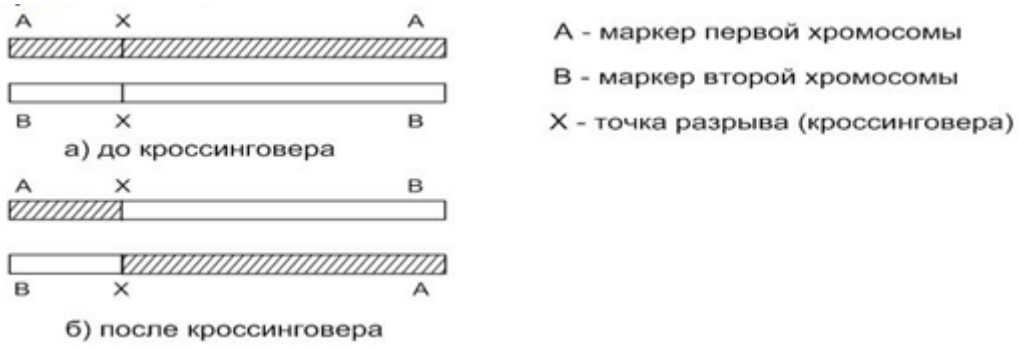

Рисунок 2. Схема одноточечного кроссинговера

Выводы

В рамках данной статьи был описан процесс сведения задачи поиска набора нужных значений, к задаче оптимизации. А так же подробный алгоритм, основанный на эволюционных вычислениях, реализующий автоматический подбор значений внешних переменных, которые позволяют осуществить прохождение выбранной ветви программного кода с целью ее тестирования. Что доказало возможность применения генетических алгоритмов в области тестирования программного обеспечения.

Перечень источников

[1] Wegener J., Buhr K., Pohlheim H., Automatic test data generation for structural testing of embedded software systems by evolutionary testing /In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). NY: Morgan Kauf-mann. 2002, pp. 1233–1240.

[2] Pargas R.P., Harrold M. J., Peck R. R. Test-data generation using genetic algorithms

[3] Кулямин В.В., Тестирование на основе моделей.

[4] Монахов А., Петренко А., Бритвина Е., Петренко О., Грошев С. Тестирование на основе моделей.