Критерии полноты тестового покрытия в генетических алгоритмах генерации тестов[1]

М.А. Владимиров
mvlad@ispras.ru

Аннотация. В статье описывается применение генетических алгоритмов для автоматической генерации тестов. Проводится анализ некоторых широко распространённых критериев полноты на предмет их применимости для построения тестов с помощью генетических алгоритмов. Строятся оценочные функции, соответствующие этим критериям.

1. Введение

При разработке и сопровождении программного обеспечения, значительная часть усилий тратится на поиск и устранение ошибок. Самым распространённым методом поиска ошибок является тестирование, то есть процесс выполнения программ с целью обнаружения ошибок [1]. Здесь слово «программа» понимается в широком смысле, как любая запись алгоритма. В частности, программами являются отдельные процедуры, функции, классы и т.д.

Процесс тестирования включает выполнение некоторого набора тестов и анализ полученных результатов. Тест — это последовательность обращений к тестируемой программе. Результатом выполнения теста является решение (вердикт) о том, отработала ли программа корректно или некорректно. Основной характеристикой тестового набора, определяющей качество тестирования, является класс возможных ошибок в программе, которые данный тестовый набор способен обнаружить.

Для количественной оценки качества тестирования используются различные метрики тестового покрытия [9]. Для качественного тестирования необходимо построить полный тестовый набор, то есть набор, удовлетворяющий некоторому критерию полноты. Зачастую критерий полноты для тестового набора определяют через пороговое значение метрики тестового покрытия.

Построение полного тестового набора для больших систем вручную может быть крайне трудоёмкой задачей. Автоматизация этого процесса позволяет

[1] Работа поддержана грантом РФФИ (05-01-999).

существенно снизить затраты на тестирование. Существуют различные подходы к решению задачи автоматической генерации тестов: [3,4,6]. Один из них основан на применении генетических алгоритмов [8]. Этот подход во многих случаях даёт хорошие результаты. К сожалению, его эффективность существенно зависит от используемого критерия полноты. Цель данной статьи — проанализировать некоторые широко распространённые критерии полноты тестового набора на их применимость при использовании генетических алгоритмов для генерации тестов.

2. Основные понятия

2.1. Генетические алгоритмы

Генетические алгоритмы — это метод решения задач оптимизации. В методе используются идеи, почёрпнутые из эволюционной биологии: наследование признаков, мутация, естественный отбор и кроссовер [7]. Определяется множество кандидатов, среди которых ищется решение задачи. Кандидаты представляются в виде списков, деревьев или иных структур данных.

Общая схема генетического алгоритма выглядит следующим образом:

- создать начальный набор кандидатов;
- оценить качество каждого кандидата в текущем наборе;
- выбрать пары наиболее качественных кандидатов для воспроизводства;
- применить оператор кроссовера;
- применить оператор мутации;
- если не выполнено условие останова, перейти к шагу 2.

Начальный набор кандидатов, как правило, формируется случайным образом. На множестве кандидатов определяется оценочная функция, задающая качество кандидата, то есть то, насколько он близок к верному решению. При выборе кандидатов для воспроизводства более качественные кандидаты имеют больше шансов. По двум выбранным кандидатам предыдущего поколения оператор кроссовера строит кандидата следующего поколения. Оператор мутации вносит малые случайные изменения кандидатов. Алгоритм завершается, когда выполняется условие останова. Часто используются следующие условия останова:

- достигается заданное количество поколений;
- найдено верное решение;
- за заданное количество итераций максимальное качество кандидатов в популяции не улучшилось;
- различные комбинации предыдущих условий.

Генетические алгоритмы позволяют решать задачи, для которых не применимы традиционные методы оптимизации. Одной из областей применения генетических алгоритмов является автоматическая генерация тестов для программного обеспечения.

2.2. Критерии полноты тестового покрытия

Для тестирования программного обеспечения требуется создать репрезентативный набор тестов, то есть набор, охватывающий все возможные сценарии работы системы. Для оценки репрезентативности тестовых наборов используются различные критерии полноты тестового покрытия.

Пусть P — множество программных систем, T — множество тестов, а Σ — множество тестовых наборов, то есть множество всех конечных подмножеств множества T. Тогда задача генерации тестов может быть сформулирована следующим образом: для заданной тестируемой системы $S \in P$ построить тестовый набор $\sigma \in \Sigma$, удовлетворяющий заданному критерию полноты тестового покрытия $F : P \times \Sigma \to \{\bullet, \perp\}$, то есть такой набор σ, для которого $F(S,\sigma) = \bullet$.

Многие критерии полноты тестового покрытия, имеющие практическое применение, строятся по следующей схеме: для тестируемой системы S критерий F определяет множество элементов тестового покрытия Q_S^F. Элементом тестового покрытия можно считать некоторый класс событий, которые могут произойти в ходе работы тестируемой программной системы. По появлению в процессе исполнения программы элементов тестового покрытия и различных их комбинаций можно судить о полноте или качестве проверки, которую выполняет данный тестовый набор. Например, элементами тестового покрытия могут быть исполняемые строки исходного кода (соответствующие событиям их исполнения); рёбра графа потока управления; пути в графе потока управления; логические выражения, встречающиеся в исходном коде и т.п. Кроме того, критерий F определяет логическую функцию $f : Q_S^F \times T \to \{\bullet, \perp\}$, которая принимает значение $f(q,t) = \bullet$, если элемент тестового покрытия q покрывается тестом t. Тестовый набор σ для системы S удовлетворяет критерию полноты тестового покрытия F, если каждый элемент тестового покрытия из множества Q_S^F покрывается хотя бы одним тестом из тестового набора σ. Иными словами:

$$F(S,\sigma) \equiv \forall q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet \qquad (1)$$

Приведём несколько примеров часто упоминаемых критериев полноты тестового покрытия:

- каждый оператор в исходном коде выполняется хотя бы один раз;
- каждая ветвь графа потока управления выполняется хотя бы один раз;
- каждый путь графа потока управления выполняется хотя бы один раз;
- каждое логическое выражение хотя бы один раз вычисляется со значением «истина» и хотя бы один раз – со значением «ложь»;

- тестовый набор убивает всех мутантов из заданного набора.

Заметим, что все критерии, приведённые в качестве примеров, соответствуют ранее изложенной схеме.

2.3. Метрики тестового покрытия

Со многими критериями полноты тестового покрытия можно связать соответствующую метрику тестового покрытия. Метрика тестового покрытия — это функция вида $M : P \times \Sigma \to R$. Значение этой функции $M(S,\sigma)$ имеет смысл числовой оценки того, насколько хорошо тестовый набор σ покрывает тестируемую систему S. Сам критерий при этом можно записать в виде $M(S,\sigma) \geq \alpha_S$, где α_S — это минимальное пороговое значение метрики M для тестируемой системы S.

В частности, для критерия полноты тестового покрытия F, представимого в виде (1), можно ввести следующую метрику:

$$M^F(S,\sigma) = |\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet\}| \qquad (2)$$

Сам критерий при этом примет вид:

$$|\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet\}| \geq |Q_S^F| \qquad (3)$$

В некоторых случаях, когда не удаётся построить тестовый набор, удовлетворяющий такому критерию полноты тестового покрытия, можно использовать ослабленный критерий:

$$|\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet\}| \geq \lambda |Q_S^F| \qquad (4)$$

Параметр $\lambda \in (0,1]$ указывает, какая доля элементов тестового покрытия должна быть покрыта тестовым набором. Приведём несколько примеров часто упоминаемых метрик тестового покрытия:

- количество покрытых (выполненных хотя бы один раз) операторов в исходном коде;
- количество покрытых ветвей графа потока управления;
- количество покрытых путей графа потока управления;
- количество распознанных мутантов (версий тестируемой системы с искусственно привнесёнными ошибками).

Все эти метрики могут быть представлены в виде (2). Подробное описание этих и других, используемых на практике, метрик полноты тестового покрытия можно найти в [9].

3. Генетический алгоритм генерации тестов

Рассмотрим следующую задачу генерации тестов:

Задача 1. Для заданной тестовой системы S построить тестовый набор $\sigma \in \Sigma$, удовлетворяющий критерию (4).

Для построения генетического алгоритма решения этой задачи необходимо определить:

- множество кандидатов;
- структуру представления кандидатов;
- оценочную функцию;
- оператор кроссовера;
- оператор мутации;
- условие останова.

3.1. Простейший алгоритм

Рассмотрим простейший генетический алгоритм для решения задачи 1. В качестве множества кандидатов возьмём множество Σ; в качестве оценочной функции возьмём метрику тестового покрытия $M^F(S,\sigma)$ для заданной тестируемой системы S. Условием останова будет наличие в текущем поколении решения σ, удовлетворяющего критерию (4). Структуру представления кандидатов, а также операторы кроссовера и мутации мы пока уточнять не будем.

Заметим, что такой алгоритм допускает ситуацию, в которой критерий (4) не выполняется ни для одного тестового набора из текущего поколения, но, тем не менее, выполняется для некоторого объединения тестовых наборов из текущего и предшествующих поколений. Иными словами, все тесты, необходимые для построения решения, уже найдены, но само решение ещё не построено. В этой ситуации алгоритм не способен эффективно построить искомое решение, целенаправленно объединив подходящие тесты из разных тестовых наборов. Причина проблемы в том, что при построении алгоритма не использовалась имеющаяся информация о структуре критерия (4).

Заметим также, что каждое последующее поколение тестов формируется путём применения операторов кроссовера и мутации к тестам из предыдущего поколения. Если в предыдущем поколении не было ни одного теста, покрывающего некоторый элемент тестового покрытия q, то в последующем поколении такой тест может появиться только как результат кроссовера или мутации тестов, не покрывающих q. Как бы мы не определяли операторы кроссовера и мутации, нет никаких оснований полагать, что получить таким способом тест, покрывающий q, проще, чем при полностью случайной генерации.

Из этих замечаний следует, что эффективность данного генетического алгоритма, вообще говоря, не выше, чем у полностью случайного алгоритма генерации тестов.

3.2. Целенаправленный поиск

Учитывая структуру критерия (4), из задачи 1 можно выделить следующую подзадачу:

Задача 2. Для заданной тестовой системы S и заданного элемента тестового покрытия q, построить тест $t \in T$, удовлетворяющий условию $f(q,t) = \bullet$.

Для решения исходной задачи 1, достаточно решить задачу 2 для $n \geq \lambda \mid Q_S^F \mid$ попарно различных элементов тестового покрытия $q_1, q_2, ..., q_n \in Q_S^F$, то есть построить тесты $t_1, t_2, ..., t_n$ такие, что

$$\begin{cases} f(q_1,t_1) = \bullet, \\ f(q_2,t_2) = \bullet, \\ \quad \ldots \\ f(q_n,t_n) = \bullet. \end{cases}$$

Решением задачи 1 будет множество $\{t_1, t_2, ..., t_n\}$.

Рассмотрим генетический алгоритм решения задачи 2. В качестве множества кандидатов возьмём множество тестов T. Условие останова: в текущей популяции присутствует тест q такой, что $f(q,t) = \bullet$. Оценочная функция $m_q : T \to R$ каждому тесту t ставит в соответствие числовую меру $m_q(t)$ того, насколько тест t близок к тому, чтобы покрыть элемент тестового покрытия q. При этом оценочная функция f_q достигает своего максимального значения на тех и только на тех тестах, которые удовлетворяют условию $f(q,t) = \bullet$. Иными словами:

$$f(q,t) = \bullet \Leftrightarrow m_q(t) = \max_{t \in T} m_q(t) \qquad (5)$$

В частности, в качестве оценочной функции можно использовать следующую функцию, удовлетворяющую условию (5):

$$m_q(t) = \begin{cases} 0, \text{при } f(q,t) = \perp, \\ 1, \text{при } f(q,t) = \bullet. \end{cases}$$

В такой оценочной функции считается, что все тесты, не покрывающие элемент тестового покрытия q, одинаково далеки от того, чтобы покрыть элемент q. При использовании этой оценочной функции эффективность генетического алгоритма будет не выше, чем при случайном поиске. Примеры более эффективных оценочных функций для некоторых метрик полноты тестового покрытия можно найти в [8,5,2].

4. Оценочные функции

В этом разделе подробно рассматриваются три известных критерия полноты тестового покрытия, и для каждого из них предлагается оценочная функция.

4.1. Покрытие операторов исходного кода

Тестовый набор удовлетворяет критерию покрытия операторов исходного кода, если при выполнении этого тестового набора каждый оператор исходного текста программы выполняется хотя бы один раз. Элементами тестового покрытия в данном случае являются операторы исходного текста. Для заданного оператора q значение оценочной функции $m_q(t)$ тем больше, чем ближе тест t к тесту, покрывающему оператор q. Для построения оценочной функции рассмотрим граф потока управления тестируемой системы S. Вершинами графа являются операторы исходного кода, то есть множество Q_S^F. В графе существует ребро, идущее из вершины q_1 в вершину q_2 тогда и только тогда, когда оператор q_2 может быть выполнен непосредственно после оператора q_1. Пусть $\Pi(q,t)$ — это множество всех элементов q' из Q_S^F, для которых выполняются следующие условия:

- существует путь в графе потока управления ведущий из q' в q или $q' = q$;
- $f(q',t) = \bullet$.

Обозначим через $dist(q',q)$ длину кратчайшего пути в графе потока управления, ведущего из q' в q ($dist(q,q) \equiv 0$). Тогда оценочную функцию можно определить следующим образом:

$$-m_q(t) = \min_{q' \in \Pi(q,t)} dist(q',q) \qquad (6)$$

Выражение, стоящее справа, определяет, за какое минимальное количество переходов можно добраться до элемента покрытия q от уже покрытых элементов из множества Q_S^F. Функция $m_q(t)$ принимает значение 0 на тех и только тех тестах, которые покрывают элемент q. Заметим, что

$$\begin{cases} m_q(t) = 0 & \Leftrightarrow \quad f(q,t) = \bullet \ ; \\ m_q(t) < 0 & \Leftrightarrow \quad f(q,t) = \perp . \end{cases}$$

4.2. Покрытие ветвей потока управления

Тестовый набор удовлетворяет критерию покрытия ветвей потока управления, если при выполнении этого тестового набора управление хотя бы один раз проходит по каждому ребру графа потока управления. Заметим, что любой

тестовый набор, удовлетворяющий этому критерию, удовлетворяет также и критерию покрытия операторов исходного кода. Обратное утверждение, однако, неверно [9]. Элементами тестового покрытия являются переходы в графе потока управления. С каждым переходом в графе потока управления можно связать условие, при котором этот переход может быть выполнен. Переход от оператора q к оператору r, с которым связано условие p, обозначим как $q \xrightarrow{p} r$. Для выполнения перехода $q \xrightarrow{p} r$ необходимо и достаточно, чтобы был выполнен оператор q, и чтобы после этого условие p обратилось в истину. Соответственно, для тестов, не покрывающих оператор q, в качестве оценочной подходит функция m_q, определённая уравнением (6), так как истинность условия p для оценки таких тестов роли не играет. Для тестов, покрывающих оператор q, функция $m_q(t)$ обращается в 0. Для таких тестов оценочная функция должна определять, насколько близок тест к тесту, для которого после выполнения оператора q будет истинным условие p. Таким образом, в общем виде оценочную функцию для критерия покрытия ветвей потока управления можно определить следующим образом:

$$m_{q \xrightarrow{p} r}(t) = \begin{cases} m_q(t), \text{при } f(q,t) = \perp, \\ \mu_{q,p}(t), \text{при } f(q,t) = \bullet \end{cases} . \qquad (7)$$

Значение функции $\mu_{q,p} : T \to R^+$ тем больше, чем ближе заданный тест к тесту, в котором условие p выполняется после выполнения оператора q. При этом функция $\mu_{q,p}(t)$ достигает своего максимума на тех и только тех тестах, в которых после выполнения оператора q выполняется условие p, то есть тех, которые покрывают переход $q \xrightarrow{p} r$.

Функцию $\mu_{q,p}(t)$ можно определять по-разному в зависимости от характера условия p. Если условие имеет форму простого (не)равенства $x \Diamond y$, где «$\Diamond$» обозначает одно из отношений «$<$», «$>$», «$=$», «$\le$» или «$\ge$», то для определения функции $\mu_{q,p}(t)$ можно использовать значение $|x - y|$, например, следующим образом:

$$\mu_{q,p}(t) = \begin{cases} 2, \text{при } x \Diamond y, \\ e^{-|x-y|}, \text{при } \neg(x \Diamond y). \end{cases}$$

Если условие представляет собой конъюнкцию $p = p_1 \wedge p_2 \wedge \ldots \wedge p_n$, то в качестве значения функции $\mu_{q,p}(t)$ можно взять количество членов этой

конъюнкции, принимающих значение «истина». В общем случае эффективно определить функцию $\mu_{q,p}(t)$ затруднительно.

4.3. Покрытие путей потока управления

Тестовый набор удовлетворяет критерию покрытия путей потока управления, если его выполнение хотя бы один раз проходит по каждому возможному пути в графе потока управления ведущему от точки входа до точки завершения работы. Этот критерий сильнее критерия покрытия ветвей потока управления. Каждый путь представляет собой последовательность переходов $R = \tau_1, \tau_2, ..., \tau_n$, где τ_i имеет вид $q_i \xrightarrow{p_i} q_{i+1}$, при $1 \le i \le n$. Упорядоченным подмножеством пути $\tau_1, \tau_2, ..., \tau_n$ назовём последовательность $\tau_{i_1}, \tau_{i_2}, ..., \tau_{i_m}$ такую, что $1 \le i_1 < i_2 < ... < i_m \le n$. Заметим, что в упорядоченном подмножестве пути конечный оператор перехода может не совпадать с начальным оператором следующего за ним перехода.

Пусть есть два пути $R' = \tau'_1, \tau'_2, ..., \tau'_l$ и $R'' = \tau''_1, \tau''_2, ..., \tau''_k$, и пусть

$$\begin{cases} \tau'_{i_1} = \tau''_{j_1}, \\ \tau'_{i_2} = \tau''_{j_2}, \\ \quad \dots \\ \tau'_{i_m} = \tau''_{j_m}, \end{cases}$$

причём $1 \le i_1 < i_2 < ... < i_m \le l$ и $1 \le j_1 < j_2 < ... < j_m \le k$. Тогда пути R' и R'' имеют общее упорядоченное подмножество размера m.

Обозначим через $length(R)$ длину пути R, а через $common(R', R'')$ – максимальный размер общего упорядоченного подмножества путей R' и R''. Определим оценочную функцию для критерия покрытия путей потока управления следующим образом:

$$-m_R(t) = length(R) + length(path(t)) - 2 \cdot common(R, path(t)).$$

Здесь $path(t)$ — это путь, по которому приходит управление при выполнении теста t. Значение в правой части равно количеству переходов в путях R и $path(t)$, не входящих в максимальное общее упорядоченное подмножество этих путей. Оно равно 0 тогда и только тогда, когда пути R и $path(t)$ совпадают.

5. Заключение

Применение генетических алгоритмов для генерации тестов предъявляет дополнительные требования к используемым критериям полноты тестового покрытия. Это вызвано тем, что критерий полноты используется не только для оценки качества сгенерированных тестов, но и непосредственно в процессе генерации для оценки близости полученных тестов к нужным результатам. Таким образом, нужно иметь оценочную функцию, позволяющую измерить эту близость, определить, насколько перспективными являются уже построенные тесты с точки зрения их использования в качестве основы для построения новых тестов. Кроме того, нужно иметь в виду, что тривиальные решения — функции вида «покрыто – 0, не покрыто – 1» — работают очень плохо.

Для критериев, связанных с покрытием тех или иных путей в коде программы, удается построить достаточно удобные оценочные функции, основанные на количестве непокрытых дуг в пути, который нужно покрыть. В статье построены такие функции для некоторых широко распространённых критериев полноты тестового покрытия.

Литература

1. Г. Мейерс. *Искусство тестирования*. М.: Финансы и статистика, 1982.
2. André Baresel, Harmen Sthamer, and Michael Schmidt. Fitness function design to improve evolutionary structural testing. In *GECCO*, pages 1329–1336, 2002.
3. Chandrasekhar Boyapati, Sarfraz Khurshid, and Darko Marinov. Korat: Automated testing based on Java predicates. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2002)*, Rome, Italy, 22–24 July 2002. IEEE.
4. Ugo A. Buy, Alessandro Orso, and Mauro Pezzè. Automated testing of classes. In *ISSTA*, pages 39–48, 2000.
5. Yoonsik Cheon and Myoung Kim. A fitness function for modular evolutionary testing of object-oriented programs. Technical Report 05-35, Department of Computer Science, University of Texas El Paso, Nov 2005.
6. Yoonsik Cheon, Myoung Kim, and Ashaveena Perumendla. A complete automation of unit testing for Java programs. In Hamid R. Arabnia and Hassan Reza, editors, *Proceedings of the 2005 International Conference on Software Engineering Research and Practice (SERP '05), Volume I, Las Vegas, Nevada, June 27-29, 2005*, pages 290–295. CSREA Press, 2005.
7. John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.
8. Paolo Tonella. Evolutionary testing of classes. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 119–128, New York, NY, USA, 2004. ACM Press.
9. Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997.