УДК 004.415.53:004.832.23

ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ ОЛИМПИАДНЫХ ЗАДАЧ ПО ТЕОРИИ ГРАФОВ С ИСПОЛЬЗОВАНИЕМ ЭВОЛЮЦИОННЫХ СТРАТЕГИЙ

М.В. Буздалов

Предлагается метод автоматизированной генерации тестов для олимпиадных задач по теории графов, предназначенный для выявления неэффективных решений. Метод основан на использовании эволюционных стратегий. Описывается использование предлагаемого метода для генерации новых тестов к олимпиадной задаче из Интернет-архива acm.timus.ru.

Ключевые слова: эволюционные стратегии, олимпиады по программированию, тестирование.

Введение

В мире проводится большое число олимпиад по программированию как для студентов [1, 2, 4, 5], так и для школьников [3, 6]. На этих олимпиадах участникам предлагается решить одну или несколько задач. Решением задачи является программа, написанная на одном из широко используемых языков программирования – например, в соревновании ACM ICPC разрешено использование языков C, C++ и Java [7]. В работе [8] описано, как решаются олимпиадные задачи, а в [9] описан подход к эффективной командной работе на олимпиаде.

Проверка корректности решения задачи, как правило, производится автоматически, путем запуска решения на одном или нескольких наборах входных данных – тестах – и проверки соответствия выходных данных решения условию задачи. Тесты одинаковы для всех решений конкретной задачи, неизвестны участникам и не изменяются в течение всей олимпиады. Если хотя бы на одном тесте программа проработала более определенного времени, использовала больше оперативной памяти, чем разрешено, или завершилась с ошибкой, то решение признается некорректным [7]. По этой причине качество тестов непосредственно влияет на уровень проведения олимпиады.

Описание процесса генерации тестов для олимпиадных задач, а также мотивация автоматизации поиска тестов приведены в работе [10].

Описание предлагаемого подхода

В настоящей работе рассматривается генерация тестов, на которых неэффективные решения работают как можно дольше. Качество таких тестов может характеризоваться величинами с большим диапазоном значений, что делает поиск требуемого теста проще.

При генерации теста против неэффективного решения предполагается, что такое решение уже имеется. Это допущение в некотором смысле противоречит цели получаемых тестов – выявлять различные неэффективные решения, реализуемые участниками с применением различных, зачастую непредсказуемых и нестандартных идей. Тем не менее, тест, сгенерированный против некоторого решения, обычно оказывается «фатальным» для достаточно большого множества неэффективных решений. При разумном выборе решений, против которых генерируются тесты, возможно «покрыть» почти все неэффективные решения.

При описываемом подходе для поиска теста применяется эволюционная стратегия [11]. Применение вероятностных алгоритмов, таких как эволюционная стратегия, для поиска теста оправдано тем, что определение сложности теста для конкретного решения, а, следовательно, аналитический поиск сложного теста является, по теореме Райса [12], алгоритмически неразрешимой задачей. Кроме того, при генерации теста ставится задача найти не наиболее сложный тест, а достаточно сложный для того, чтобы отфильтровать неэффективные решения.

Для применения эволюционной стратегии необходимо разработать подходящий способ кодирования теста и оптимизируемую функцию – функцию приспособленности теста.

Способ кодирования теста зависит от задачи, для которой генерируются тесты. Он должен эффективно учитывать ограничения, заданные в условии задачи. Для повышения эффективности подхода следует проанализировать возможные способы кодирования и выбрать тот из них, который обеспечивает большую долю потенциально эффективных тестов.

Большую трудность составляет выбор функции приспособленности. В качестве функции приспособленности нельзя выбирать время работы решения на тесте, поскольку это время может случайным образом изменяться в зависимости от загруженности различных узлов компьютера, а также имеет низкую точность измерения.

Выбор в качестве функции приспособленности числа выполненных инструкций кода решает описанные выше проблемы. Однако во многих случаях то, что функция приспособленности пропорциональна времени работы программы, может привести к тому, что алгоритм оптимизации сделает выбор в пользу «больших» тестов, игнорируя «качественные», иными словами, предпочтет количество качеству.

В связи с этим предлагается проанализировать решение, против которого требуется сгенерировать тест, и модифицировать его исходный код так, чтобы в процессе работы решения вычислялась та харак-

теристическая величина, которая будет использована в качестве функции приспособленности. Эта величина может лучше выражать качество теста, что ускоряет процесс поиска.

Применение подхода к олимпиадной задаче «Work for Robots»

Для апробации описываемого подхода в условиях реальных олимпиадных задач была выбрана задача «Work for Robots». С текстом условия этой задачи можно ознакомиться на сайте Timus Online Judge [13], где она размещена под номером 1695 [14]. Формализованная версия условия задачи звучит следующим образом.

Дан неориентированный граф без петель и кратных ребер. Число вершин графа N находится в диапазоне от одной до 50. Необходимо найти число клик в этом графе (кликой [11] называется полный подграф данного графа). Ограничение по времени – две секунды, ограничение по памяти – 64 МБ.

Данная задача является NP-полной [12]. Решение этой задачи основано на известном среди участников олимпиад по программированию приеме «meet-in-the-middle»: вершины графа делятся на два приблизительно равных по числу вершин множества, для каждого подмножества одного из этих множеств вычисляется число клик в нем, а для каждого подмножества второго множества выясняется, является ли оно кликой, после чего ответ на задачу возможно получить путем «слияния» результатов вычислений для двух половин.

По состоянию на 1 августа 2009 года по этой задаче было засчитано множество решений, основанных на неэффективных алгоритмах. К таким алгоритмам относится, например, перебор с запоминанием – число состояний, которые требуется запомнить, в худшем случае чрезвычайно велико, но на всех имевшихся тестах это число было небольшим. Были также сданы некоторые решения, эффективные по времени, но потенциально использующие больше памяти, чем разрешено.

Для генерации тестов к этой задаче была использована (1+1)-эволюционная стратегия, описанная в работе [11].

Кодирование теста. Тесты для данной задачи предлагается кодировать в виде матрицы булевых значений размера N, являющейся матрицей смежности графа, данного в тесте. При этом матрица является симметричной, а на ее диагонали всегда стоят нули. С одной стороны, такой способ кодирования является достаточно простым, а с другой стороны, попытки закодировать тест другим способом оказались недостаточно эффективными. Для того чтобы получающиеся тесты были как можно более трудными, для генерации тестов взято максимально возможное число вершин в графе ($N = 50$).

Оператор мутации. Для описанного выше способа кодирования разработано два оператора мутации. Первый оператор инвертирует один элемент матрицы, не стоящий на диагонали, и симметричный ему. Второй оператор симметрично инвертирует случайно выбранные элементы матрицы в количестве, не превышающем одной десятой от числа всех элементов матрицы. Данные операторы применяются поочередно.

Целью первого из операторов является попытка перепробовать все возможные «единичные» изменения теста. Для этого элементы матрицы, подлежащие изменению, перебираются в некотором, заранее определенном порядке, например, $A_{2,1}$, $A_{3,1}$, $A_{3,2}$ и т.д. Второй оператор делает более глобальные изменения, которые в некоторых случаях могут быстрее привести к цели.

Схема эволюционной стратегии. В данной работе используется (1+1)-эволюционная стратегия, описанная в работе [11]. Сначала генерируется случайным образом начальный тест и вычисляется его функция приспособленности. Далее к текущему тесту применяется один из операторов мутации (мутации с нечетным порядковым номером проводятся первым оператором мутации, с четным – вторым оператором). Если функция приспособленности полученного теста больше, чем у текущего теста, то новый тест замещает текущий, иначе текущий тест остается неизменным. Если в течение некоторого времени приспособленность тестов не увеличивается, то алгоритм запускается с новым начальным тестом.

Выбор функции приспособленности. Решения, использующие перебор с запоминанием, реализованы, как правило, в виде рекурсивной процедуры, которая вычисляет ответ для подграфа исходного графа, построенного на некотором подмножестве его вершин. Если для данного подграфа ответ еще не вычислялся, то производится вычисление и полученный ответ сохраняется, иначе возвращается сохраненный результат. При каждом вызове такой процедуры выполняется $O(1)$ или $O(\log N)$ действий (где N – число вершин в графе) в зависимости от структуры данных, используемой для хранения подсчитанной информации.

Решения, следующие описанной стратегии, можно условно разделить на три типа.

1. Запоминание вычисленных значений в ассоциативном массиве. Такие решения используют объем памяти, прямо пропорциональный числу вычисленных значений. Это число может быть очень большим, кроме этого, операции добавления и поиска в таких структурах, как ассоциативный массив, имеют большой скрытый множитель в асимптотической оценке, поэтому такие решения неэффективны.

2. Запоминание вычисленных значений в массиве небольшого размера. Для решений такого типа массив должен иметь размер, равный степени двойки. Небольшим считается размер массива, не превосходящий 32 МБ, так как массивы размером 64 МБ и более не умещаются в разрешенном объеме памяти вместе с исполняемым кодом программы. В этом случае в массиве можно запомнить не более 2^{23} значений. Хотя существуют корректные решения такого типа, многие решения, следующие этой стратегии, не укладываются в ограничение по времени, поскольку вынуждены много раз вычислять одни и те же значения, которые они не способны запомнить.
3. Запоминание вычисленных значений в массиве большого размера (64 МБ и более). Используя тот факт, что на *x*86-совместимых архитектурах компьютера для статических массивов память выделяется страницами по 4 КБ, можно определить в программе массив размером 64 МБ и более. Если не все ячейки массива использовались в процессе работы программы, объем использованной памяти, равный суммарному объему задействованных страниц, может быть намного меньше, в частности, удовлетворять ограничениям задачи. Однако существуют тесты, которые приводят к более плотному заполнению массива и нарушению ограничений на потребляемую память.

Для генерации тестов против решений первого и второго типа значение функции приспособленности равняется числу вызовов главной рекурсивной процедуры. Для достаточно сложных тестов это число имеет порядок нескольких десятков или сотен миллионов, в то время как для простых тестов это число не превышает нескольких десятков.

Для генерации тестов против решений третьего типа используемый массив разбивается на участки по 1 КБ, и функция приспособленности равняется числу таких участков, использованных при работе программы.

Для решений, не подходящих под вышеуказанные описания, используется индивидуальный подход к разработке функции приспособленности.

Результаты экспериментального исследования

Всего на момент начала эксперимента было засчитано 86 решений различных авторов. Для генерации тестов были отобраны 10 решений, которые, по предположению автора работы, представляли собой достаточно репрезентативную выборку некорректных решений. Из них одно решение относилось к третьему типу, большинство остальных решений примерно поровну распределялись между первым и вторым типами.

Против одного из отобранных решений не удалось сгенерировать тест, и после более тщательного теоретического анализа реализованного в нем алгоритма это решение было признано корректным. Против остальных решений было сгенерировано в совокупности 10 тестов, получивших номера с 43 по 52. В таблице для каждого теста *T* приведено число решений, которые прошли все тесты с номерами, меньшими *T*, и не прошли тест *T*, включая отдельную статистику для каждого возможного результата прохождения теста.

Таким образом, с помощью описанного метода сгенерированы тесты, позволившие почти полностью устранить имеющиеся на время проведения эксперимента зачтенные неверные решения, которых было более 50% (45 из 86).

№ теста	Число решений, не прошедших тест[1]			
	Все	WA[2]	TL[3]	ML[4]
43	1	0	1	0
44	2	0	2	0
45	2	0	2	0
46	1	0	1	0
47	0	0	0	0
48	9	0	8	1
49	13	0	13	0
50	12	0	12	0
51	2	2	0	0
52	3	0	0	3
Всего	45	2	39	4

[1] Если решение не прошло тест с номером *N*, то оно не учитывается в статистике для тестов с большими номерами.

[2] Вердикт WA означает, что решение выдало неверный ответ.

[3] Вердикт TL означает, что решение не уложилось в ограничение по времени.

[4] Вердикт ML означает, что решение не уложилось в ограничение по памяти.

Таблица. Эффективность сгенерированных тестов

На текущий момент процент зачтенных неверных решений составляет не более 25%, что существенно ниже, чем до использования описанного метода. Тем не менее, полностью устранить неверные решения по этой задаче не получилось. Связано это с тем, что новые некорректные решения существенно используют рандомизацию (перестановку номеров вершин), которой в данной задаче пока не удается противопоставить эффективные меры.

Заключение

В работе описан метод, позволяющий автоматически генерировать тесты для олимпиадных задач по программированию с использованием эволюционной стратегии. Данный метод был применен для генерации тестов для реальной олимпиадной задачи, где показал высокое качество тестов, создаваемых с его помощью. Полученные результаты позволяют утверждать, что описанный метод является достаточно перспективным в плане его применения при подготовке тестов для задач по олимпиадному программированию в целях повышения качества олимпиад.

Работа выполнена в рамках реализации Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы.

Литература

1. ACM International Collegiate Programming Contest [Электронный ресурс]. – Режим доступа: http://en.wikipedia.org/wiki/ACM_ICPC, свободный. Яз. англ. (дата обращения 26.09.2011).
2. TopCoder [Электронный ресурс]. – Режим доступа: http://www.topcoder.com/tc, свободный. Яз. англ. (дата обращения 26.09.2011).
3. International Olympiad in Informatics [Электронный ресурс]. – Режим доступа: http://www.ioinformatics.org, свободный. Яз. англ. (дата обращения 26.09.2011).
4. Google Code Jam [Электронный ресурс]. – Режим доступа: http://code.google.com/codejam, свободный. Яз. англ. (дата обращения 26.09.2011).
5. Russian Code Cup [Электронный ресурс]. – Режим доступа: http://russiancodecup.ru, свободный. Яз. рус. (дата обращения 26.09.2011).
6. Интернет-олимпиады по информатике [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/school/io/, свободный. Яз. рус. (дата обращения 26.09.2010).
7. Правила проведения полуфинала NEERC [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/information/contest-rules.html, свободный. Яз. рус., англ. (дата обращения 26.09.2010).
8. Оршанский С.А. О решении олимпиадных задач по программированию формата ACM ICPC // Информатика. – 2006. – № 1. – С. 21–26.
9. Акишев И.Р. Об опыте участия в командных соревнованиях по программированию формата ACM // Информатика. – 2008. – № 19. – С. 20–28.
10. Буздалов М.В. Генерация тестов для олимпиадных задач по программированию с использованием генетических алгоритмов // Научно-технический вестник СПбГУ ИТМО. – 2011. – № 2 (72). – С. 72–77.
11. Baeck T., Hoffmeister F., Schwefel H.-P. A Survey of Evolution Strategies // Proceedings of the Fourth International Conference on Genetic Algorithms. – Morgan Kauffman, 1991. – P. 2–9.
12. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М: Мир, 1982. – 416 с.
13. Timus Online Judge. Архив задач с проверяющей системой [Электронный ресурс]. – Режим доступа: http://acm.timus.ru, свободный. Яз. рус., англ. (дата обращения 26.09.2010).
14. Задача «Work for Robots» [Электронный ресурс]. – Режим доступа: http://acm.timus.ru/problem.aspx?num=1695, свободный. Яз. рус., англ. (дата обращения 26.09.2011).

Буздалов Максим Викторович – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, аспирант, mbuzdalov@gmail.com