Исследование выполнено по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы» в рамках государственного контракта П2373 от 18 ноября 2009 года.

Литература

1. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2010. – 176 с.
2. Кларк Э.М, Грамберг О., Пелед Д. Верификация моделей программ. Model Checking. – М.: МЦНМО, 2002. – 416 с.
3. Myers G. The Art of Software Testing. – John Wiley & Son. Inc, 2004.
4. McMinn P. Search-based software test data generation: a survey: Research Articles // Software Testing, Verification & Reliability. – 2004. – № 14 (2). – P. 105–156.
5. Степанов О.Г. Методы реализации автоматных объектно-ориентированных программ. Диссертация на соискание ученой степени кандидата технических наук. – СПб: СПбГУ ИТМО, 2009. – Режим доступа: http://is.ifmo.ru/disser/stepanov_disser.pdf, своб.
6. Meyer B. Applying design by contract // Computer. – 1992. – 25(10). – P. 40–51.
7. Kalaji A.S., Hierons R.M. and S. Swift. Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM) // Software Testing, Verification, and Validation (ICST). 2nd International IEEE Conference. – 2009. Denver, Colorado: IEEE.
8. Duale Y., Ümit Uyar M. A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models // IEEE Transactions on Computers. – 2004. – Vol. 53. – № 5. – P.614–627.
9. Jacky J., Veanes M., Campbell C., Schulte W. Model-Based Software Testing and Analysis with C#. – Cambridge University Press, 2008.
10. Mark U., Legeard B. Practical Model-Based Testing: A Tools Approach. – Morgan–Kaufmann, 2007.
11. Bourdonov I., Kossatchev A., Kuliamin V., Petrenko A. UniTesK Test Suite Architecture // Proc. of FME 2002. LNCS 2391. – Springer-Verlag, 2002. – P. 77–88.
12. Wegener J., Buhr K., Pohlheim H. Automatic test data generation for structural testing of embedded software systems by evolutionary testing // Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). – NY. – 2002. – P. 1233–1240.
13. Веб-сайт yEd – Graph Editor [Электронный ресурс]. – Режим доступа: www.yworks.com/en/products_yed_about.html, своб.
14. Cheon Y., Leavens G.T. A Runtime Assertion Checker for the Java Modeling Language (JML) // Proceedings of the SERP '02, Las Vegas, Nevada, USA. – CSREA Press, June 2002. – P. 322–328.

Законов Андрей Юрьевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, andrew.zakonov@gmail.com

Шалыто Анатолий Абрамович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.415.53:004.832.23

ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ С ИСПОЛЬЗОВАНИЕМ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

М.В. Буздалов

Предлагается метод автоматизированной генерации тестов для олимпиадных задач по программированию, предназначенный для выявления неэффективных решений. Этот метод основан на использовании генетических алгоритмов. Описывается использование предлагаемого метода для генерации новых тестов к олимпиадной задаче из Интернет-архива acm.timus.ru, при этом ни одно из имевшихся решений не прошло построенный набор тестов.
Ключевые слова: генетические алгоритмы, олимпиады по программированию, олимпиадные задачи, тестирование.

Введение

В мире проводится большое число олимпиад по программированию. Они способствуют выявлению талантливых программистов среди школьников и студентов. Среди них можно отметить международную студенческую олимпиаду по программированию International Collegiate Programming Contest [1], проводимую Association for Computing Machinery (далее олимпиада будет упоминаться как ACM ICPC), с развитой сетью отборочных соревнований, международную олимпиаду школьников по информатике [2], соревнования, проводимые компанией TopCoder [3], Интернет-олимпиады по информатике и программированию [4] и многие другие.

На олимпиадах по программированию предлагается решить одну или несколько задач. Формулировка задачи предполагает чтение входных данных, удовлетворяющих условию задачи, получение требуемых результатов на основе этих данных и вывод результатов в формате, указанном в условии задачи. Решением задачи является программа, написанная на одном из языков программирования (например, в соревновании ACM ICPC используются языки C, C++ и Java [5]).

Программа тестируется на наборе тестов, не известных участникам. На работу программы накладываются определенные ограничения, такие как максимальное время выполнения и максимальный объем используемой памяти.

Решение считается прошедшим определенный тест, если оно при работе с ним не нарушило ограничений, завершилось корректно (без ошибок времени выполнения) и его ответ признан правильным. О конкретных видах задач и ограничениях можно прочитать, например, на сайте олимпиады [5]. В статье [6] изложен процесс решения отдельно взятой задачи при одиночной работе участника, а в статье [7] описана работа в команде.

Подготовка задач для олимпиад по программированию

Проведение соревнований на высоком уровне предполагает качественную подготовку задач. Этот процесс включает в себя выбор интересных идей для будущих задач, написание условий и решений, а также составление тестов. В настоящей работе предлагается новый метод автоматизированного составления тестов против неэффективных решений – таких решений, которые всегда выдают верный ответ, но на некоторых тестах не укладываются в ограничения по времени или памяти.

Тесты для олимпиадных задач

В условии задачи на входные данные накладываются некоторые ограничения. Тем не менее, для большинства задач тестирование решения на всех возможных тестах, удовлетворяющих этим ограничениям, не представляется возможным, так как число таких тестов чрезмерно велико. В силу этого из всех возможных тестов необходимо выбрать только те, которые позволят установить, является ли некоторое решение корректным. Однако, согласно теореме Райса [8], данная задача в общем случае является алгоритмически неразрешимой. Следовательно, возможно выбрать набор тестов, который лишь с некоторой долей уверенности позволяет утверждать о корректности решения. Цель подготовки тестов состоит в том, чтобы сделать эту уверенность как можно большей.

Подготовка тестов

Подготовка тестов к олимпиадной задаче является творческим процессом. Для каждой задачи в обязательном порядке пишутся несколько решений, называемых *решениями жюри*. Среди них должны быть как корректные, так и неверные и неэффективные решения. Цель решений жюри – проверка тестов: любое корректное решение должно пройти все тесты, для любого некорректного решения должно существовать определенное ненулевое число тестов, которое оно не проходит.

Часть тестов пишется вручную. Такие тесты проверяют решения на корректность разбора случаев, встречающихся в задаче. К таким тестам относятся и так называемые «минимальные» тесты, которые проверяют корректность работы решений около нижних границ ограничений.

Тесты большого размера, как правило, генерируются согласно некоторому шаблону. Так, например, если в условии задачи фигурирует некоторый граф, то можно сгенерировать двоичное дерево, полный двудольный граф и другие виды графов [9]. Для покрытия тех ошибок, которые могут быть не найдены с помощью описанных выше тестов, создаются «случайные» тесты. В общем случае большие тесты создаются программами, написанными членами жюри.

Для некоторых задач может быть написано некорректное решение, которое не было предусмотрено жюри. В этом случае в наборе тестов может и не оказаться такого теста, на котором это решение не работает. Авторы задач стремятся предотвратить такое развитие событий, для чего реализуют эвристические решения и генерируют тесты против них. Однако не все идеи эвристических решений могут быть найдены или реализованы, а в отдельных случаях поиск тестов против них может затянуться на неопределенное время.

По указанным причинам на олимпиадах иногда встречаются задачи со «слабым» набором тестов, которые пропускают некорректные решения. Это приводит к тому, что наиболее подготовленные участники олимпиад, ищущие корректные решения, не решают такие задачи, в то время как менее подготовленные участники успешно сдают некорректные решения, что противоречит самой идее олимпиады.

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 упрощает поиск требуемого теста.

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 тест кодируется в виде особи генетического алгоритма [10, 11]. Способ кодирования зависит от задачи, для которой генерируются тесты. Кроме этого, кодирование должно эффективно учитывать ограничения, данные 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. Также измеренное время работы, как правило, пропорционально некоторому минимальному интервалу – «кванту» времени, выделяемому операционной системой для работы программы, поэтому число различных значений функции приспособленности снижается.

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

Применение подхода к олимпиадной задаче «Ships. Version 2»

Для апробации описываемого подхода в условиях реальных олимпиадных задач была выбрана задача «Ships. Version 2». С текстом условия этой задачи можно ознакомиться на сайте *Timus Online Judge* [12], где она размещена под номером 1394 [13]. Условие задачи состоит в следующем.

Дано N $(2 \le N \le 99)$ предметов с весами w_i, $1 \le w_i \le 100$. Также дано M $(2 \le M \le 9)$ рюкзаков с вместимостями $c_j \ge 1$. Известно, что $\sum_{i-1}^{N} w_i = \sum_{j=1}^{M} c_j$. Требуется поместить все данные предметы в данные рюкзаки. Гарантируется, что искомое размещение существует. Ограничение по времени – одна секунда, ограничение по памяти – 16 Мбайт.

Из изложенного следует, что рассматриваемая задача является частным случаем задачи о мультирюкзаке [14] с дополнительными ограничениями: для всех предметов их вес равен стоимости, и все рюкзаки должны быть заполнены.

Задача о мультирюкзаке NP-полна, так как она принадлежит классу NP и к ней сводится NP-полная задача о сумме подмножеств. Кроме того, она является NP-трудной *в* сильном смысле [14], т.е. для нее неизвестны решения, работающие с полиномиальной оценкой от размерности задачи и ограничений на веса и стоимости предметов. Вполне естественно ожидать от рассматриваемой задачи такого же свойства. Из этого следует, что при данных ограничениях задачи маловероятно существование решения, которое укладывалось бы в ограничения по времени и памяти на всех возможных тестах. Однако существует большое число различных эвристических решений, для которых трудно составить такой тест. По состоянию на 15 июня 2009 года из 3100 посланных на проверку решений было принято 260.

Для генерации тестов к этой задаче был применен генетический алгоритм.

Представление теста в виде особи генетического алгоритма

В условии задачи сказано, что сумма весов предметов должна равняться сумме вместимостей рюкзаков, и существует способ разместить все предметы по рюкзакам. Для случайно сгенерированного теста это условие с большой вероятностью не выполняется. Целесообразно закодировать тестовые данные таким образом, чтобы эти условия выполнялись по построению.

Предлагается следующий способ построения теста по последовательности чисел. Будем считать, что последовательность состоит из целых чисел из диапазона $[0;100]$. При этом положительным числам соответствуют предметы с весом, равным соответствующему числу. Нули же являются разделителями последовательности на непрерывные группы положительных чисел. Каждой такой группе сопоставлен рюкзак с вместимостью, равной сумме чисел этой группы. На рис. 1 проиллюстрирован пример последовательности и соответствующего ей теста.

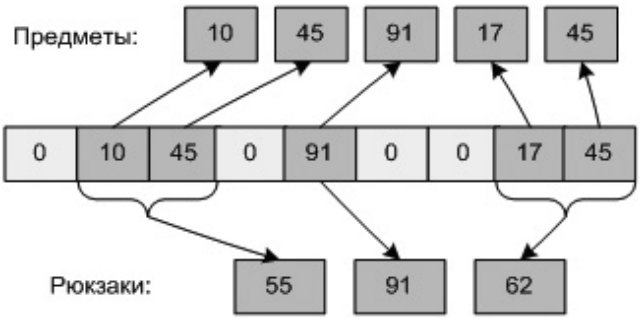

Рис. 1. Числовая последовательность и генерируемый ей тест

Любой тест, удовлетворяющий ограничениям, может быть закодирован в виде последовательности чисел длиной не более $N + M - 1$. Напротив, не любая возможная последовательность генерирует тест, удовлетворяющий ограничениям: число предметов и число рюкзаков могут быть как слишком малыми, так и слишком большими. В связи с этим предлагается обнулять значение приспособленности тех последовательностей, которые генерируют тесты, не удовлетворяющие ограничениям.

Согласно теореме о схемах [10], в классическом генетическом алгоритме, работающем со строками, «выживают» короткие по длине, малые по числу элементов схемы с большой приспособленностью. В данной же задаче после предварительного анализа было выяснено, что общую приспособленность определяют большие группы элементов последовательности.

Чтобы иметь возможность группировать блоки элементов последовательности, предлагается использовать новый вид кодирования последовательности – древовидные генераторы последовательностей (рис. 2).

Каждое поддерево генерирует некоторую последовательность, определяемую следующими правилами. Лист дерева генерирует последовательность из одного целого числа, хранящегося в нем. Внутренняя вершина дерева генерирует последовательность, образованную конкатенацией результатов генерации детей этой вершины.

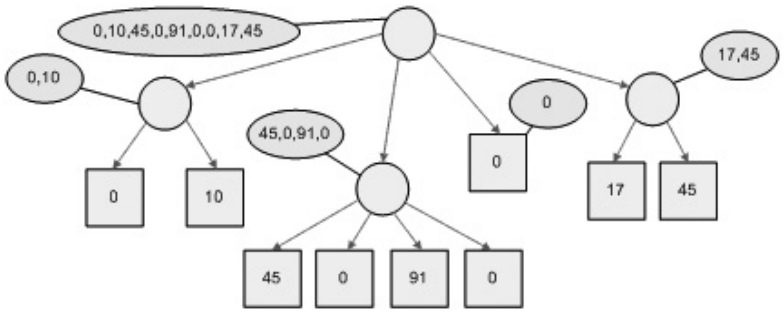

Рис. 2. Пример древовидного генератора

Древовидные генераторы напоминают деревья разбора, используемые в генетическом программировании [15], с той разницей, что результатом «выполнения» поддерева является не число, а последовательность.

Функция приспособленности

Для любого решения в качестве функции приспособленности можно выбрать число выполнений «узких мест» алгоритма. При таком подходе счетчик числа операций увеличивается внутри самого глубокого уровня вложенности некоторого числа циклов, встречающихся в программе. Итоговое значение счетчика в этом случае почти точно пропорционально времени работы программы. Однако этот общий подход работает не для всех решений.

Для рекурсивных алгоритмов решения в качестве функции приспособленности можно выбрать число вызовов рекурсивной функции (или одной из таких функций в случае, если их несколько) в течение времени работы алгоритма.

Часто также встречаются решения, запускающие один и тот же алгоритм на разных вариантах упорядочения входных данных. Так, многие эвристические решения перебирают случайные перестановки предметов и (или) рюкзаков и для каждой такой перестановки пытаются произвести поиск ответа. В таких случаях в качестве функции приспособленности разумно выбрать число таких запусков в процессе работы решения.

В некоторых случаях целесообразно использовать комбинации описанных выше подходов. Пусть F_1 – функция приспособленности, соответствующая одному подходу, F_2 – другому, F_n – n-му. Тогда функция приспособленности для комбинации подхода будет представлять собой вектор $(F_1, F_2, \ldots, F_n)$. Значения этой функции сравниваются лексикографически.

В качестве оператора селекции используется турнирный отбор [10], в котором с вероятностью 0,9 выбирается более приспособленная особь.

Операторы скрещивания и мутации

В качестве оператора скрещивания используется стандартный для генетического программирования оператор обмена поддеревьями. Выбор поддерева описывается следующим образом: если в данный момент алгоритм рассматривает лист дерева, то он и будет выбран в качестве поддерева. Если же алгоритм рассматривает узел дерева, то с вероятностью 0,5 выбирается поддерево с корнем в этом узле, иначе равновероятно выбирается один из потомков, и процедура выбора продолжается. Оператор мутации заменяет случайным образом выбранное поддерево на сгенерированное случайным образом дерево той же величины.

Описание и результаты эксперимента

Целью эксперимента было сгенерировать такие тесты, чтобы максимально возможное число решений, прошедших уже имеющиеся тесты, не прошло хотя бы один из таких тестов. Для этого из имеющихся на сервере зачтенных решений было выбрано 25 решений. Некоторые из них были выбраны для генерации тестов против них, остальные решения использовались для оценки эффективности полученных тестов. Генерация каждого теста производилась против одного решения за время от одного часа до суток.

Всего было сгенерировано 28 тестов. Три из них генерировались на основе трех решений, показавших на ранее существовавших тестах наихудшее время работы. Остальные тесты генерировались против семи других решений, против каждого из них было сгенерировано от одного до трех тестов. Решения, против которых тесты не генерировались, использовались для оценки эффективности получаемых тестов.

В результате перетестирования решений из имевшихся на момент начала тестирования принятых решений ни одно не прошло сгенерированный набор тестов. По результатам тестирования среди сгенерированных тестов было отобрано 11 лучших. Они получили номера с 48 по 58 в порядке уменьшения сложности.

На настоящее время набор тестов, в который входят и сгенерированные генетическим алгоритмом тесты, прошло всего 8 решений, что говорит о высоком качестве набора тестов.

Заключение

В работе описан метод, позволяющий автоматически генерировать тесты для олимпиадных задач по программированию с использованием генетических алгоритмов. 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.

Литература

1. ACM International Collegiate Programming Contest [Электронный ресурс]. – Режим доступа: http://en.wikipedia.org/wiki/ACM_ICPC, свободный. Яз. англ. (дата обращения 21.09.2010).
2. International Olympiad in Informatics [Электронный ресурс]. – Режим доступа: http://www.ioinformatics.org, свободный. Яз. англ. (дата обращения 21.09.2010).
3. TopCoder [Электронный ресурс]. – Режим доступа: http://www.topcoder.com/tc, свободный. Яз. англ. (дата обращения 21.09.2010).
4. Интернет-олимпиады по информатике [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/school/io/, свободный. Яз. рус. (дата обращения 21.09.2010).
5. Правила проведения полуфинала NEERC [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/information/contest-rules.html, свободный. Яз. рус., англ. (дата обращения 21.09.2010).
6. Оршанский С.А. О решении олимпиадных задач по программированию формата ACM ICPC // Мир ПК. – 2005. – № 9.
7. Акишев И.Р. Об опыте участия в командных соревнованиях по программированию формата ACM // Методическая газета для учителей «Информатика». – 2008. – № 19. – С. 20–28.
8. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982.

9. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ / Пер. с англ. – 2-е изд. – М.: Вильямс, 2005. – 1296 с.
10. Holland J.P. Adaptation in Natural and Artificial Systems. – The University of Michigan Press, 1975.
11. Mitchell M. An Introduction to Genetic Algorithms. – MA: MIT Press, 1996.
12. Timus Online Judge. Архив задач с проверяющей системой [Электронный ресурс]. – Режим доступа: http://acm.timus.ru, свободный. Яз. рус., англ. (дата обращения 21.09.2010).
13. Задача «Ships. Version 2» [Электронный ресурс]. – Режим доступа: http://acm.timus.ru/problem.aspx?space=1&num=1394, свободный. Яз. рус., англ. (дата обращения 21.09.2010).
14. Pisinger D. Algorithms for Knapsack Problems: PhD. Thesis. – University of Copenhagen, 1995.
15. Koza J.R. Genetic programming: On the Programming of Computers by Means of Natural Selection. – MA: The MIT Press, 1998.

Буздалов Максим Викторович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, mbuzdalov@gmail.com

УДК 004.05

СОВМЕСТНОЕ ПРИМЕНЕНИЕ КОНТРАКТОВ И ВЕРИФИКАЦИИ ДЛЯ ПОВЫШЕНИЯ КАЧЕСТВА АВТОМАТНЫХ ПРОГРАММ

А.А. Борисенко, В.Г. Парфенов

При создании систем со сложным поведением важную роль играет контроль качества разрабатываемых программ. Цена ошибки в таких системах может быть слишком велика, поэтому важно не просто проверить соответствие создаваемой программы всем предъявленным к ней требованиям, но и сделать этот процесс эффективным, максимально автоматизировав его. На практике этого можно добиться, формализовав все требования к программе и храня полученную исполнимую спецификацию непосредственно вместе с кодом программы.

Рассмотрены существующие методы контроля качества современных программных систем и автоматных программ, а также описан процесс создания среды, позволяющей поддержать сразу три подхода к проверке качества программ с явным выделением состояний: проверку на модели, модульное тестирование и контракты. Предложенный подход позволяет сохранить корректность записи сформулированных требований при изменении самой программы, а также интерактивно контролировать ее качество.

Ключевые слова: контроль качества, соответствие спецификации.

Введение

Качественное программное обеспечение (ПО) – это, прежде всего, надежное программное обеспечение. Зачастую системы, требующие высокого уровня надежности, представляют собой системы со сложным поведением [1], а цена ошибки в таких проектах может быть слишком высокой [2]. При разработке систем со сложным поведением важное место занимает стадия тестирования, а одним из распространенных методов разработки таких систем является автоматное программирование [1].

Другой важной чертой современных программных проектов является их частое изменение: модифицируются требования к системе, находятся и исправляются ошибки. Для контроля качества ПО, соответствия его реализации и спецификации, в современных проектах используется ряд методов: ручное и автоматизированное тестирование, контрактное программирование и верификация [3].

Контроль качества автоматных программ

К автоматным программам могут быть успешно применены следующие методы анализа корректности [4]: тестирование, верификация (проверка на модели и доказательная верификация), контракты. Рассмотрим последовательно каждый из них, выделяя при этом характерные для данного метода преимущества и недостатки.

Тестирование. Тестирование – процесс выявления ошибок в ПО. Запуск программы на определенных входных данных, а также проверка различных сценариев выполнения позволяют достаточно быстро (по сравнению с другими методами поиска ошибок) убедиться в корректности обработки заданных сценариев [5].

Тестирование, применяемое после окончательного написания программы, не способно найти все ошибки. Как заметил Э. Дейкстра, если при тестировании ошибки в программе не найдены, это еще не означает, что их там нет.

Верификация. *Артефактами* жизненного цикла ПО называются различные информационные сущности, документы и модели, создаваемые или используемые в ходе разработки и сопровождения ПО [6].