Сборник статей третьей Всероссийской научной конференции «Нечеткие системы и мягкие вычисления» (НСМВ-2009). Волгоград: 2009. Том II, с 16–24.

УДК 004.02+004.05

Применение генетических алгоритмов, выявляющих неэффективные решения олимпиадных задач по программированию, на примере задачи о рюкзаке

М. В. Буздалов

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

mbuzdalov@gmail.com

Тестирование является важной частью цикла разработки программного обеспечения (ПО). Оно занимает около 50 % времени и более 50 % стоимости разработки ПО [1]. Так как качество тестирования сильно зависит от человеческого фактора, исследователи и разработчики сосредоточили большие усилия над автоматизацией процесса тестирования. Среди новейших тенденций в этой области встречаются работы по эволюционному тестированию программ [2, 3].

Олимпиадное движение в области информатики и программирования активно развивается как в России, так и в мире. Олимпиады способствуют выявлению талантливых программистов среди школьников и студентов. Среди известных олимпиад можно отметить международную студенческую олимпиаду по программированию *International Collegiate Programming Contest* [4], проводимую *Association for Computing Machinery* (*ACM ICPC*), международную олимпиаду школьников по информатике [5], соревнования, проводимые компанией *TopCoder* [6], интернет-олимпиады по информатике и программированию [7] и многие другие.

На большинстве олимпиад предлагается решить одну или несколько задач. Решение задачи, как правило, предполагает написание программы на одном из алгоритмических языков (так, на олимпиаде *ACM ICPC* разрешено использование языков *C, C++* и *Java* [4]), которая читает из файла входные данные, получает на их основе выходные данные и также записывает их в файл. Входные и выходные данные имеют определенный, строго соблюдаемый формат, а на время работы программы и потребляемые ей ресурсы накладываются жесткие ограничения.

По заданному решению невозможно алгоритмически определить, корректно ли оно, согласно теореме Райса [8]. Поэтому корректность решения проверяется на одном или нескольких тестах, подготовленных заранее и неизвестных участникам олимпиады. На некоторых олимпиадах результаты

проверки сообщаются участникам немедленно по окончании проверки, на других – только после завершения олимпиады.

Тестирование решения на уже готовых тестах может быть полностью автоматизировано, и на большинстве олимпиад процесс тестирования не требует участия человека. Однако подготовка тестов является сложным и ответственным процессом, который в настоящее время выполняется вручную. В самом деле, для большинства олимпиадных задач число возможных тестов чрезвычайно велико для того, чтобы можно было протестировать решение на всех таких тестах. Поэтому необходимо составить такой набор тестов, чтобы как можно больше неверных или неэффективных решений его не прошло.

Из вышесказанного следует, что автоматизация составления тестов приводит к повышению качества тестового набора и задачи в целом. Однако полная автоматизация этого процесса для произвольно выбранной задачи невозможна. В связи с этим приходится ограничиваться выбором классов олимпиадных задач и классов решений таких задач, против которых необходимо генерировать тесты.

Среди всех неэффективных решений можно выделить определенный класс решений, трудных для генерации тестов против них – так называемые эвристические решения. Такие решения всегда выдают корректные ответы, но асимптотическая оценка на их время работы слишком велика для того, чтобы можно было сделать вывод о достаточной эффективности данного решения. Однако на большинстве тестов такие решения работают за небольшое время, и лишь крайне малое число тестов позволяет выявить неэффективность решения. Даже против известного решения такого рода найти хороший тест крайне трудно, еще труднее найти такой тест для множества неизвестных решений.

Предлагается производить поиск тестов, выявляющих неэффективность некоторого конкретного решения, **с помощью генетических алгоритмов** [9]. В излагаемом методе особь генетического алгоритма кодирует тест, а фитнесс-функция характеризует время, затраченное решением на работу с этим тестом. Более приспособленные особи соответствуют тестам, время работы на которых больше.

Для повышения эффективности работы генетического алгоритма целесообразно разработать такое кодирование теста в виде особи, которое позволило бы, с одной стороны, снизить долю некорректных тестов среди генерируемых, а с другой – ввести операторы мутации и кроссовера, отражающие структуру теста.

Примером задачи, для которой существуют эвристические решения, может служить задача о рюкзаке [10]. Приведем ее краткую формулировку в виде олимпиадной задачи:

- Дано:
  - N предметов, каждый стоимостью P_i и весом W_i.
  - Рюкзак вместимостью W.
- Найти:

    - Набор предметов с суммарным весом, не превосходящим W, и максимальной стоимостью.
- Ограничения на входные данные:
    - $1 \le N \le N_{max}$;
    - $1 \le P_i \le P_{max} = 10^4$;
    - $1 \le W_i \le W_{max} = 10^4$.
- Ограничение на время работы: T_{max} секунд (с).

Хотя задача является *NP*-полной [11] – для любого известного решения необходимые время и объем ресурсов в худшем случае возрастают экспоненциально от размера теста. В настоящее время существует множество решений (например, приведенные в работе [10]), которые на почти всех входных данных работают за полиномиальное время от их размера.

Предлагается следующая схема кодирования тестов особями генетического алгоритма.

1. Вместимость рюкзака W берется как половина суммы весов предметов. С одной стороны, это упрощает кодирование теста, а с другой стороны, такие тесты обычно являются более трудными для большинства решений задачи. Таким образом, кодирование теста сведено к кодированию множеству предметов.
2. В качестве способа закодировать множество предметов предлагается использовать древовидные генераторы последовательностей. При таком подходе особь генетического алгоритма является подвешенным деревом. Каждая вершина дерева (генератор) генерирует некоторую последовательность предметов по следующему правилу:
    - лист дерева генерирует последовательность, состоящую из одного предмета.
    - последовательность, генерируемая узлом дерева, строится следующим образом:
        - все последовательности, сгенерированные потомками данного узла, конкатенируются;
        - к каждому элементу получившейся последовательности применяется некоторое преобразование, хранящееся в узле;
        - итоговая последовательность является результатом генерации.
3. Тест, генерируемый особью, составляется из элементов последовательности, сгенерированной корневым генератором особи.

Приведенная схема кодирования обладает одним существенным преимуществом перед строковым кодированием: появляется возможность манипулировать не только отдельными предметами, но и их группами.

Операторы кроссовера и мутации являются аналогами соответствующих операторов в генетическом программировании. Оператор кроссовера производит обмен случайно выбранными поддеревьями. Используется два оператора мутации: один заменяет случайно выбранную вершину дерева на случайно сгенерированную вершину того же типа, другой заменяет случайно выбранное поддерево на случайно сгенерированное дерево такого же размера.

В результате действия какого-либо оператора может получиться особь, генерирующая тест, который не удовлетворяет ограничениям. Для того чтобы удовлетворять ограничению $N \leq N_{max}$, дерево генераторов последовательности усекается так, чтобы длина генерируемой им последовательности не превосходила N. Ограничения на веса и стоимости предметов удовлетворяются следующим образом: в сгенерированной последовательности предмет со стоимостью P_i и весом W_i заменяется на предмет со стоимостью $\max(1, \min(P_{max}, P_i))$ и весом $\max(1, \min(W_{max}, W_i))$.

Построение следующего поколения происходит по следующему алгоритму:

1. Число генерируемых потомков выбирается равным числу особей в текущем поколении. Если это число является нечетным, то оно увеличивается на единицу.
2. Для генерации пары потомков родители выбираются с помощью турнирного отбора. Сначала к родителям применяется оператор кроссовера, потом к каждому из получившихся потомков применяется один из операторов мутации.
3. В новое поколение выбирается не более 100 лучших особей из предков, потомков и 25 случайно сгенерированных особей.

Алгоритм начинает работу с пустой популяции и заканчивает работу, когда выполнен хотя бы один из следующих критериев:

- найден тест, время работы на котором превышает ограничение по времени;
- закончилось время, отведенное генетическому алгоритму для работы.

Для того, чтобы оценить эффективность данного подхода, был проведен поиск тестов аналитическими методами, а также с помощью шаблонов.

I. Аналитическое построение тестов для задачи о рюкзаке описано в работе [12]. Характерная черта таких тестов – экспоненциальный рост весов и стоимостей предметов с ростом числа предметов N. Так, тесты Тодда дают веса порядка $3 \cdot 10^7$ при сравнительно небольшом значении N $(N = 20)$. Это неприемлемо для выбранных ограничений. Поэтому аналитически построенные тесты на практике оказываются *малоприменимыми.*

II. Для задачи о рюкзаке известно [10] несколько шаблонов для генерации тестов:

- Некоррелированные тесты. Веса и стоимости выбираются равновероятно и независимо из интервалов $[1; W_{max}]$ и $[1; P_{max}]$ соответственно.
- Слабо коррелированные тесты. Вес каждого предмета W_i выбирается равновероятно из интервала $[1 + \beta; W_{max} - \beta]$, а стоимость P_i выбирается равновероятно из интервала $[W_i - \beta; W_i + \beta]$, где β – небольшая положительная константа (порядка 50).
- Сильно коррелированные тесты. Вес каждого предмета W_i выбирается равновероятно из интервала $[1; W_{max} - \beta]$, а стоимость P_i выбирается равной $W_i + \beta$, где β – небольшая константа (порядка 10).
- Задачи о «сумме подмножеств». Вес предмета выбирается равным стоимости.

По сравнению с другими шаблонами, шаблон «сильно коррелированные тесты» порождает самые сложные тесты для многих решений задачи о рюкзаке.

Поиск требуемых тестов на основе того или иного шаблона производился следующим образом:

1. Случайным образом на основе шаблона конструируется тест.
2. Исследуемое решение запускается на этом тесте.
3. Если время, затраченное решением на этом тесте, больше, чем на лучшем из ранее сгенерированных тестов, то текущий тест объявляется лучшим.
4. Если время, отведенное на поиск теста, не истекло, то процесс повторяется сначала.

III. Описанные подходы (с применением генетического алгоритма и с использованием каждого из упомянутых шаблонов) были использованы для поиска тестов для задачи о рюкзаке с различными ограничениями на число предметов, время выполнения решения и время поиска тестов. Для тестирования использовались решения, реализующие алгоритмы *EXPKNAP* и *HARDKNAP* из работы [10]. При этом для каждого из алгоритмов рассматривалось по одной корректной и по одной частичной реализации. Некоторые из полученных результатов приведены в таблице.

Таблица. Результаты поиска тестов

Тип тестов	*ExpKnapPart*	*ExpKnap*	*HardKnapPart*	*HardKnap*
$N_{max} = 25$, $T_{max} = 2$ с, время поиска теста – 1 час				
Некоррелированные	0 мс	0 мс	0 мс	0 мс
Коррелированные	20 мс	20 мс	0 мс	20 мс
Сумма подмножеств	20 мс	20 мс	10 мс	10 мс
Генетические	380 мс	570 мс	10 мс	20 мс
$N_{max} = 30$, $T_{max} = 5$ с, время поиска теста – 2 часа				
Некоррелированные	0 мс	0 мс	0 мс	0 мс
Коррелированные	120 мс	110 мс	50 мс	100 мс
Сумма подмножеств	20 мс	20 мс	50 мс	100 мс
Генетические	> 5 с	> 5 с	50 мс	90 мс

Из таблицы следует, что тесты, сгенерированные генетическим алгоритмом, по сложности не уступают тестам, сгенерированным по шаблонам, а для алгоритмов семейства *EXPKNAP* – превосходят по сложности на порядки. При дальнейшем анализе полученных тестов был выделен новый шаблон для генерации тестов – «тесты с малым разнообразием предметов». Тесты, генерируемые по этому шаблону, являются крайне сложными для решений, реализующих алгоритмы семейства «перебор с возвратом и отсечениями».

Подход к генерации тестов с помощью генетических алгоритмов был также применен при генерации дополнительных тестов к задаче *Ships. Version 2*. Эта задача размещена на сервере для решения задач [13] под номером 1394.

Формулировка этой задачи такова:

- Дано N кораблей, каждый корабль имеет длину S_i.
- Дано M гаваней, каждая гавань имеет длину L_i.
- Сумма длин всех кораблей равна сумме длин всех гаваней.
- Требуется расставить корабли по гаваням так, чтобы сумма длин кораблей, стоящих в i-той гавани, равнялась L_i.
- Гарантируется, что ответ всегда существует.
- Ограничения на входные данные:
  - $2 \le N \le 99$;
  - $2 \le M \le 9$;
  - $1 \le S_i \le 100$.
- Ограничение на время работы: одна секунда

Из формулировки следует, что задача является частным случаем задачи о мультирюкзаке. Эта задача является *NP*-полной в сильном смысле – для ее решения неизвестны алгоритмы, работающие за время, полиномиальное не только от длины входа, но и от иных ограничений задачи. Отсюда следует, что при данных ограничениях маловероятно существование решения, проходящего все тесты и укладывающегося во все ограничения.

По состоянию на 3 июня 2009 года 258 решений прошло все имевшиеся на проверяющем сервере тесты. Столь большое число зачтенных решений свидетельствует о том, что имеющиеся тесты достаточно слабы.

С помощью модификации подхода на основе генетического алгоритма, описанного выше, который был применен для генерации тестов к задаче о рюкзаке, найден набор более сильных тестов.

Эта модификация состоит в следующем. Будем в этом случае представлять хромосому, кодирующую тест, последовательностью чисел от нуля до 100. Каждое ненулевое число последовательности соответствует кораблю, каждая непрерывная группа ненулевых чисел – гавани. Закодированный таким образом тест не нуждается в проверке на существование ответа и на равенство сумм длин, а все остальные ограничения достаточно просто удовлетворяются. Последовательность чисел, в свою очередь, кодируется описанным ранее древовидным генератором последовательности. Схема генетического алгоритма не отличается от ранее описанной за исключением того, что максимальный размер поколения составляет 500 особей.

В результате эксперимента удалось найти набор из 11 тестов, который не проходит ни одно из посланных на сервер решений. Эти тесты были добавлены на сервер и получили номера с 48 по 58 включительно. В настоящее время ни одно решение не прошло все имеющиеся тесты, что свидетельствует о высоком качестве полученного набора.

Дальнейшая работа включает в себя следующие направления:

- генерация тестов для других классов задач;
- генерация тестов, трудных одновременно для нескольких решений;
- исследование верхних оценок на время работы различных алгоритмов с помощью генетических алгоритмов.

Литература

1. *Myers G. J.* The Art of Software Testing. John Wiley & Sons, Inc., 2004.
2. *Tonella P.* Evolutionary testing of classes //ISSTA. 2004, pp. 119-128.
3. *Alander J. T., Mantere T., Turunen P.* Genetic Algorithm Based Software Testing. http://citeseer.ist.psu.edu/40769.html
4. *ACM International Collegiate Programming Contest*. http://icpc.baylor.edu
5. *International Olympiad in Informatics*. http://ioinformatics.org
6. *TopCoder*. http://www.topcoder.com/tc
7. *Интернет-олимпиады* по информатике и программированию. http://neerc.ifmo.ru/school/io
8. *Игошин В. И.* Математическая логика и теория алгоритмов: учебное пособие для студентов высших учебных заведений. М.: Academia, 2008, 448 с.
9. *Скобцов Ю. А.* Основы эволюционных вычислений. Донецк.: ДонНТУ, 2008.
10. *Pisinger D.* Algorithms for Knapsack Problems: Ph.D. thesis. University of Copenhagen. 1995.

11. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
12. *Chvatal V.* Hard knapsack problems //Operations Research. 1980. Vol. 28, no. 6, pp. 1402–1411.
13. *Timus Online Judge*. Архив задач с проверяющей системой. http://acm.timus.ru