ИСПОЛЬЗОВАНИЕ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ ПРИСПОСОБЛЕННОСТИ ДЛЯ ТЕСТИРОВАНИЯ РЕШЕНИЙ ОЛИМПИАДНЫХ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Буздалова А.С.

студентка кафедры компьютерных технологий НИУ ИТМО,
abuzdalova@gmail.com

Буздалов М.В.

ассистент кафедры компьютерных технологий НИУ ИТМО,
mbuzdalov@gmail.com

Аннотация: Рассматривается задача автоматической генерации тестов для олимпиадных задач по программированию с помощью эволюционных алгоритмов. Для повышения эффективности генерации предлагается использовать вспомогательные функции приспособленности. Рассматриваются различные способы их использования, такие как многокритериальная оптимизация со случайным выбором критериев и ранее разработанный метод EA+RL, применимый как к однокритериальным, так и к многокритериальным алгоритмам. Результаты экспериментов подтверждают, что использование вспомогательных функций приспособленности позволяет получать достаточно качественные тесты за меньшее число поколений эволюционного алгоритма.

Введение

Данная работа посвящена генерации тестов, применяющихся для оценки решений задач по олимпиадному программированию [1, 2]. Как правило, в состав задания по олимпиадному программированию входит формулировка задачи, которую необходимо решить, и форматы входных и выходных данных. Тесты представляют собой наборы входных данных, для которых известны соответствующие выходные данные — правильные ответы на поставленную задачу. Для того чтобы решение успешно прошло тест, необходимо, чтобы оно позволило получить правильный ответ в пределах максимального установленного времени и фиксированного объема используемой памяти.

Целью данной работы является автоматическая генерация тестов производительности. Будем называть *успешным* запуском алгоритма генерации запуск, в результате которого получен тест, время работы решения на котором превышает максимальное установленное время.

Для генерации тестов используются эволюционные алгоритмы [3]. Этот подход был предложен в работе одного из авторов данной статьи [4]. Задачей эволюционного алгоритма является максимизация *целевой*

функции приспособленности (в данном случае времени работы решения олимпиадной задачи). Алгоритм оперирует с *особями* — кандидатами в решение задачи оптимизации (определенным образом закодированными тестами). Особи подвергаются *мутации* и *скрещиванию*. Таким образом, образуются потомки. Наиболее приспособленные особи из числа потомков и родителей отбираются для формирования следующего *поколения*. Описанный процесс повторяется до тех пор, пока не будет получена достаточно приспособленная особь или же не будет превышено максимальное число поколений.

Несмотря на то, что время работы решения является целевой функцией приспособленности, использование ее в эволюционном алгоритме не является эффективным подходом по ряду причин: время работы решения платформозависимо, зашумлено и квантовано [4]. Для повышения эффективности эволюционного алгоритма предлагается использовать вспомогательные функции приспособленности [5–8], коррелирующие с целевой функцией, но при этом лишенные ее недостатков. Рассматривается как случайный выбор вспомогательных функций приспособленности, предложенный в работе [8], так и выбор наиболее эффективных функций приспособленности, осуществляемый с помощью метода EA+RL, предложенного ранее авторами данной работы [5, 6].

Цель работы

Целью работы является исследование эффективности различных подходов к генерации тестов, основанных на использовании эволюционных алгоритмов. Особое внимание уделяется применению вспомогательных функций приспособленности. Предлагается генерировать тесты производительности для двух примеров решений олимпиадной задачи. В следующих разделах описаны олимпиадная задача, ее решения, а также соответствующие решениям вспомогательные функции приспособленности.

Олимпиадная задача по программированию

В качестве примера, рассмотрим задачу «Ships. Version 2», расположенную на сайте Timus Online Judge [9] под номером 1394. Задача формулируется следующим образом: даны N кораблей, каждый длины s_i, и M гаваней, каждая длины h_j. Необходимо распределить корабли по гаваням таким образом, чтобы суммарная длина всех кораблей, размещенных в j-ой гавани, не превосходила значения h_j. Гарантируется, что корректное размещение всегда существует. Вводятся следующие ограничения:

- $N \leqslant 99, 2 \leqslant M \leqslant 9, 1 \leqslant s_i \leqslant 100$;
- $\sum s_i = \sum h_j$;

- ограничение на время работы решения 1 с;
- ограничение на объем используемой памяти 64 Мб.

Описанная задача является частным случаем задачи о мультирюкзаке, которая, в свою очередь, является NP-трудной в строгом смысле [10] (другими словами, не известно ни одного решения это задачи, время работы которого было бы полиномиальным относительно любого набора входных значений). По этой причине является крайне маловероятным, что каждый возможный экземпляр задачи может быть решен в рамках заданных ограничений на время и память. Однако в случае наиболее удачных решений весьма сложно составить такой тест, на котором решение превысит заданный предел времени работы.

Вспомогательные функции приспособленности

Как было сказано выше, целевой функцией приспособленности является время работы решения на тесте, однако выращивать особи, руководствуясь значениями этой функции, неэффективно. Предлагается использовать вспомогательные функции приспособленности. Для вычисления значений этих функций в код решения встраиваются счетчики, коррелирующие со временем выполнения кода. Например, если в алгоритме решения часто вызывается некоторая рекурсивная функция, можно ввести счетчик числа вызовов этой функции. Значения, принимаемые таким счетчиком, не зависят от платформы, на которой запущено решение, а также не подвержены зашумлению. Ниже представлены псевдокоды двух решений задачи, описанной в предыдущем разделе, с включенными в них счетчиками. Далее будем называть вспомогательные функции приспособленности так же, как соответствующие счетчики.

В *Решении 1* используются счетчики I, P, R:

```
Read the input data
I := 0, P := 0, R := 0
while(solution not found)
  Randomly shuffle ships
  Call the recursive DP-based ship arranging procedure
     For each call to this procedure, R := R + 1
     In each innermost loop, P := P + 1
  if (solution is found)
     Write the answer
  else
     I := I + 1
  end if
end while.
```

В *Решении 2* используются счетчики Iterations, Length, Tuple:

```
Read the input data
iterations := 0, length := 0, last := 0
while (solution not found) do
  Randomly shuffle ships and havens
last := 0
  Call the recursive ship arranging procedure
     For each call to this procedure, last:= last + 1
  if (solution is found) then
     Write the answer
  else
     iterations := iterations + 1
     length := length + last
     last := 0
  end if
end while
tuple := 1000000000 * iterations + last.
```

Главное отличие между двумя решениями заключается в реализации рекурсивной функции распределения кораблей. Особенностью первого решения является то, что для реализации этой функции используется подход, основанный на динамическом программировании. К сожалению, привести более детальное описание не представляется возможным, так как правила олимпиадных соревнований по программированию не позволяют публиковать решения участников. Однако из результатов экспериментов будет видно, что, с точки зрения сложности генерации тестов, решения различаются.

Описание предлагаемого подхода

В данном разделе описаны различные подходы к генерации тестов, основанные на применении эволюционных алгоритмов, которые могут использоваться как для максимизации одной функции приспособленности (*однокритериальные*), так и для максимизации нескольких функций приспособленности (*многокритериальные*). Функция приспособленности может использоваться постоянно (*фиксированная*) или выбираться из заранее подготовленного набора в течение работы эволюционного алгоритма (*динамическая*). Применяемые в работе эволюционные алгоритмы, а также стратегии выбора динамической функции приспособленности.

Эволюционные алгоритмы

Используются как однокритериальные, так и многокритериальные эволюционные алгоритмы. Во всех используемых алгоритмах особи кодируют-

ся одинаковым образом, а также применяются одни и те же операторы мутации и скрещивания, описанные в работе [11]. Размер поколений составляет 200 особей. Ниже описаны используемые алгоритмы.

Однокритериальный генетический алгоритм (GA). В данном алгоритме для формирования следующего поколения применяется турнирный отбор с размером турнира 2 и с вероятностью выбора лучшей особи, равной 0,9. Операторы мутации и скрещивания применяются с вероятностью 1,0. Используется элитизм, отбирается пять лучших особей [3]. Если на протяжении 10 000 поколений лучшее полученное значение функции приспособленности не изменяется, то текущее поколение очищается и заполняется случайными новыми особями.

Многокритериальный генетический алгоритм (NSGA-II). Для одновременной оптимизации нескольких функций приспособленности используется ускоренная вариация алгоритма NSGA-II [12]. Применяется стратегия выбора особей, основанная на доминировании по Парето.

Стратегии выбора вспомогательных функций приспособленности

Стратегия, предложенная M.T. Jensen [8]. Рассмотрим две стратегии выбора вспомогательных функций приспособленности. Первая из них была предложена в работе [8]. Согласно этой стратегии, вспомогательная функция приспособленности выбирается случайным образом и используется в течение фиксированного числа поколений. Затем выбирается следующая функция приспособленности, и так далее, пока все вспомогательные функции приспособленности не будут использованы.

Метод EA+RL. В качестве второй стратегии выбора использовался предложенный авторами данной статьи метод EA+RL [5, 6]. В контексте данной работы он называется GA+RL и NSGA-II+RL, в соответствии с теми эволюционными алгоритмами, совместно с которыми был применен. Функция приспособленности выбирается с помощью обучения с подкреплением [13] из множества, включающего как целевую функцию приспособленности, так и вспомогательные. На выбор влияет вознаграждение, которое зависит от роста целевой функции приспособленности. Целью алгоритма обучения с подкреплением является максимизация суммарного вознаграждения. Таким образом, выбираются функции приспособленности, способствующие росту целевой функции приспособленности.

В качестве алгоритма обучения используется алгоритм *Delayed Q-learning* $m=0$, $\varepsilon=0.001$, $\gamma=0.1$ [14]. Алгоритм перезапускается каждые 50 поколений с целью предотвращения стагнации.

Результаты эксперимента

Для каждого из рассмотренных решений были сгенерированы тесты. Использовались все рассмотренные алгоритмы с каждой из предложенных

функций. Каждый алгоритм был запущен 100 раз, затем результаты были усреднены. Алгоритм останавливался, если был получен тест, на котором время выполнения решения превышало ограничение по времени, или же если было достигнуто максимальное число поколений, равное 10 000.

Результаты экспериментов приведены в Таблицах 1, 2. Используются следующие обозначения: ФП — функция приспособленности, используемая для оценки особей; T — целевая функция приспособленности (время выполнения решения на тесте); σ — среднеквадратичное отклонение числа поколений, необходимого для генерации эффективного теста. Алгоритм тем эффективнее, чем меньше среднее число понадобившихся для генерации поколений.

Рассмотрим результаты генерации тестов для Решения 1, приведенные в Таблице 1. Как и ожидалось, использование времени выполнения решения в качестве единственной функции приспособленности (алгоритм №4) неэффективно.

В случае фиксированной ФП, многокритериальная оптимизация (№5–7) превосходит по производительности однокритериальную (№1–4) вне зависимости от используемой вспомогательной функции приспособленности. В случае динамической ФП многокритериальная оптимизация также достаточно эффективна, хотя и уступает производительности многокритериального алгоритма в случае фиксированной ФП. Однако использование динамической ФП в общем случае может быть более предпочтительным подходом, так как заранее неизвестно, какая из фиксированных ФП эффективна. Пример случая, когда не все вспомогательные ФП эффективны, соответствует Решению 2.

№	Алгоритм	ФП	Успешные запуски, %	Число поколений	
				среднее	σ
Фиксированная ФП					
1	GA	I	99	2999	1986
2	GA	R	93	3153	3742
3	GA	P	54	12621	12770
4	GA	T	0	—	—
5	NSGA-II	T, I	100	203	119
6	NSGA-II	T, R	100	440	381
7	NSGA-II	T, P	100	448	360
Динамически выбираемая ФП					
8	GA+RL	все	65	9636	9538
9	NSGA-II+RL	все	99	895	1215
10	NSGA-II+Jensen	все	100	882	786

Таблица 1. Результаты генерации тестов для Решения 1

Результаты генерации тестов для Решения 2 приведены в Таблице 2. В случае фиксированной ФП, только Tuple оказывается эффективной вспо-

могательной функцией (№ 1 – 5). Это невозможно узнать, не запустив алгоритмы с каждой из вспомогательных функций.

В то же время, все приведенные методы, использующие динамический выбор ФП, достаточно эффективны (№ 8 – 10). В случае использования динамической ФП достаточно одного запуска любого из приведенных алгоритмов. Механизм автоматического выбора наиболее эффективной функции приспособленности позволяет не осуществлять этот выбор вручную. Таким образом, использование динамически выбираемой вспомогательной функции приспособленности является достаточно универсальным и эффективным методом.

№	Алгоритм	ФП	Успешные запуски, %	Поколения	
				среднее	σ
Фиксированная ФП					
1	GA	Tuple	95	3815	3466
2	GA	Ierations	54	12669	12873
3	GA	Length	51	13755	14082
4	GA	T	0	-	-
5	NSGA-II	T, Tuple	95	2217	3136
6	NSGA-II	T, Iterations	45	15861	16723
7	NSGA-II	T, Length	20	41330	44768
Динамически выбираемая ФП					
8	GA+RL	все	80	5817	6160
9	NSGA-II+RL	все	72	6679	7764
10	NSGA-II+Jensen	все	75	6103	7076

Таблица 2. Результаты генерации тестов для Решения 2

Заключение

Было проведено сравнение некоторых подходов к генерации тестов для задач по олимпиадному программированию. Показано, что использование вспомогательных функций приспособленности позволяет существенно повысить производительность эволюционных алгоритмов, применяемых для решения рассмотренной задачи генерации. Наиболее универсальной следует считать стратегию динамического выбора вспомогательной функции приспособленности, так как для ее использования не требуется предварительных знаний от том, какая из вспомогательных функций приспособленности оптимальна. Результаты эксперимента показывают, что в среднем наиболее эффективно использовать данную стратегию совместно с алгоритмами многокритериальной оптимизации. Дальнейшие исследования предполагают разработку метода автоматического добавления в код тестируемого решения счетчиков, позволяющих вычислять значения вспомогательных функций приспособленности.

Литература

1. ACM International Collegiate Programming Contest. http://en.wikipedia.org/wiki/ACM_ICPC [дата просмотра: 23.04.2013].
2. International Olympiad in Informatics. http://www.ioinformatics.org [дата просмотра: 23.04.2013].
3. *Eiben A.E., Smith J.E.* Introduction to Evolutionary Computing. — Springer-Verlag, Berlin, Heidelberg, New York. — 2007.
4. *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.
5. *Buzdalova A., Buzdalov M.* Increasing Efficiency of Evolutionary Algorithms by Choosing between Auxiliary Fitness Functions with Reinforcement Learning // Proceedings of the Eleventh International Conference on Machine Learning and Applications, ICMLA 2012. — Boca Raton: IEEE Computer Society, 2012. — Vol. 1. — P. 150–155.
6. *Буздалова А.С., Буздалов М.В.* Метод повышения эффективности эволюционных алгоритмов с помощью обучения с подкреплением // Научно-технический вестник информационных технологий, механики и оптики — Санкт-Петербург, 2012. — №5(81) — С. 115–119.
7. *Knowles J.D., Watson R.A., Corne D.* Reducing Local Optima in Single-Objective Problems by Multi-objectivization // Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization EMO '01. — London, UK: Springer-Verlag. — 2001. — P. 269–283.
8. *Jensen M.T.* Helper-Objectives: Using Multi-Objective Evolutionary Algorithms for Single-Objective Optimisation: Evolutionary Computation Combinatorial Optimization // Journal of Mathematical Modelling and Algorithms. — 2004. — №4. — P. 323–347.
9. Timus Online Judge. Архив задач с проверяющей системой. http://acm.timus.ru [дата просмотра 23.04.2013]
10. *Pisinger D*. Algorithms for Knapsack Problems: PhD. thesis / University of Copenhagen. — 1995.
11. *Buzdalova A.S., Buzdalov M.V.* Adaptive Selection of Helper-Objectives for Test Case Generation // Proceedings of the IEEE Congress on Evolutionary Computation. — 2013 [to be published].
12. *R.G. L. D'Souza, K.C. Sekaran*, and *A. Kandasamy*. Improved NSGA-II based on a novel ranking scheme. Computing Research Repository, abs/1002.4005. — 2010.
13. *Sutton R.S., Barto A.G.* Reinforcement Learning: An Introduction. — MIT Press, Cambridge, MA, 1998. — 322 p.
14. *Strehl A.L., Li L., Wiewora E., Langford J., Littman M.L.* PAC Model-Free Reinforcement Learning // Proceedings of the 23rd International Conference on Machine learning. ICML'06. — 2006. — P. 881–888.