Материал опубликован в тезисах научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 174–181.

В. Р. Данилов, А. А. Шалыто

МЕТОД ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ, ПРЕДСТАВЛЕННЫХ ДЕРЕВЬЯМИ РЕШЕНИЙ

ВВЕДЕНИЕ

Автоматное программирование [1] – быстроразвивающаяся парадигма программирования, использующая представление программ в виде формальной модели – автоматов. Применение генетического программирования [2] совместно с автоматным могло бы существенно сократить цикл разработки автоматных программ за счет автоматизации построения автоматов.

Для генерации автоматов могут применяться различные модификации эволюционных алгоритмов [3]. Однако, для большинства из этих методов размер хромосомы, требуемый для хранения автомата, экспоненциально зависит от числа входных переменных.

При решении задачи классификации плотности Дж. Козой [4] применялось представление функции переходов автомата с помощью деревьев разбора [5], представляющих булевы функции. Это представление обладает существенно большей выразительностью по сравнению с традиционными табличными методами, что позволило эффективно решить задачу. Однако этот метод может быть применен только к автоматам, имеющим два состояния. Целесообразно разработать метод эффективного представления автоматов, который может быть применен к решению более сложных задач.

ОСНОВНЫЕ ПОЛОЖЕНИЯ

В настоящей работе предлагается метод генерации автоматов, представленных деревьями решений [6]. Указанный метод позволяет эффективно представлять автоматы с дискретными входными переменными. Разработанный метод сравнивается с генетическими алгоритмами.

Представление автомата с помощью деревьев решений

Дерево решений является удобным способом задания дискретной функции, зависящей от конечного числа дискретных переменных.

Оно представляет собой помеченное дерево, метки в котором расставлены по следующему правилу: внутренние узлы помечены символами переменных, ребра – значениями переменных, листья – значениями искомой функции.

Для определения значения функции по значениям переменных необходимо спуститься от корня до листа, и сформировать значение, которым помечен полученный лист. При этом из вершины, помеченной переменной *x*, переход производится по тому ребру, которое помечено тем же значением, что и значение переменной *x*. Пример дерева решений изображен на рис. 1. Здесь для простоты показано дерево, реализующее булеву функцию от переменных *a*, *b* и *c*.

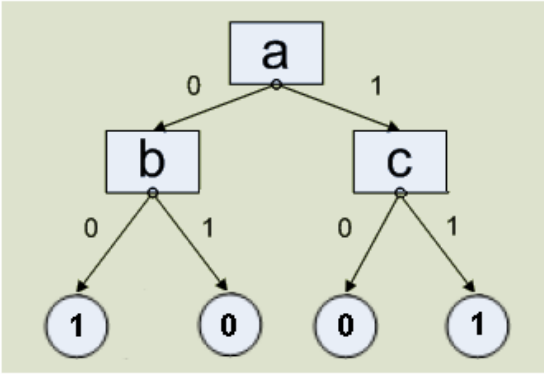

Рис. 1. Пример дерева решений

Опишем предлагаемый метод представления автомата с помощью деревьев решений. Для задания автомата необходимо выразить его функции переходов и выходов с помощью деревьев решений. Возможно осуществить следующее преобразование автомата: вместо задания функций переходов и выходов для автомата в целом, представим эти функции для каждого состояния. Более формально: зададим для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q, x), \lambda(q, x))$ для $\forall x \in X$.

Функции σ_q соответствуют функциям переходов и выходов из состояния q. Каждая из этих функций может быть выражена с помощью дерева решений. В этих деревьях переменными являются входные переменные автомата, а множеством значений – все возможные пары (*Новое Состояние*, *Действие*). **Таким образом, автомат в целом задается упорядоченным набором деревьев решений.**

Генетические операции

Для использования представления автоматов в виде набора деревьев решений в генетических алгоритмах определим следующие операции: случайное порождение автомата – в каждом состоянии создается случайное дерево решений; скрещивание автоматов – скрещиваются деревья решений в соответствующих состояниях; мутация автомата – в случайном дереве решений производится мутация.

Здесь считается, что число состояний в автомате фиксировано, и поэтому противоречий при выполнении определенных таким образом операций не возникнет.

После определения операций над набором деревьев решений, определим генетические операции над собственно деревьями решений. Их можно определить следующим образом:

- Случайное порождение дерева решений. При этом случайным образом выбирается метка: либо одно из возможных значений функции переходов, либо одна из переменных. После этого создается вершина, помеченная выбранной меткой. Если была выбрана переменная, то рекурсивно генерируются дети вершины, иначе вершина становится листом дерева.
- Мутация – выбирается случайный узел в поддереве. После этого поддерево, соответствующее выбранному узлу, заменяется на случайно сгенерированное (рис. 2).
- Скрещивание – в скрещиваемых деревьях выбираются два случайных узла. После этого поддерево, соответствующее выбранному узлу в первом дереве, заменяется поддеревом, соответствующим узлу второго дерева (рис. 3).

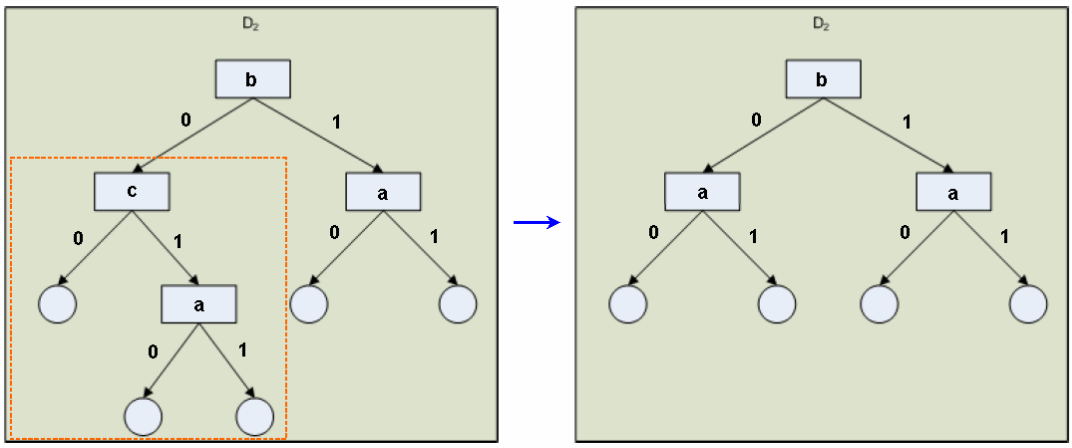

Рис. 2. Мутация деревьев решений

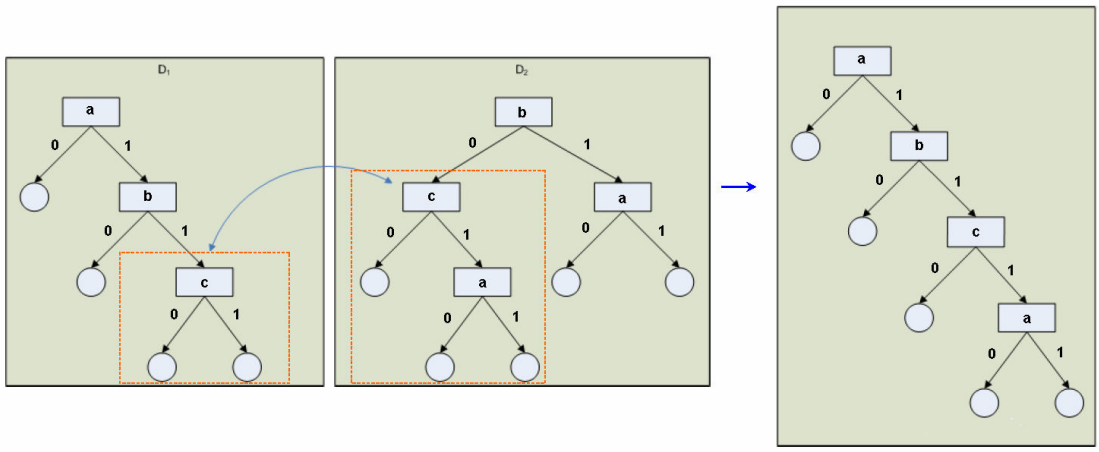

Рис. 3. Скрещивание деревьев решений

Отметим, что заданные таким образом операции могут порождать деревья, в которых некий атрибут встречается на пути от корня до листа дважды (например, дерево, полученное в результате скрещивания на рис. 3). Такие деревья являются корректными – определяют функцию на любом наборе

значений атрибутов единственным образом, однако имеют недостижимые вершины. Появление недостижимых ветвей может влиять на эволюцию отрицательным образом по следующим причинам:

- Появление недостижимых ветвей ведет к увеличению высоты деревьев, экспоненциально увеличивая память, необходимую на хранение популяции. В условиях ограниченной памяти это ведет к уменьшению популяции.
- Недостижимые ветви дерева могут являться плохими приближениями искомой функции (так как не учитываются при подсчете функции приспособленности), но при этом, однако, копироваться в потомков, замедляя генерацию.

Таким образом, необходимо ввести операцию обрезки – удаление недостижимых ветвей. Операция может быть выполнена следующим образом: узел, одна из дочерних вершин которого недостижима, заменяется на свою достижимую дочернюю вершину. На рис. 4 приведен пример обрезки недостижимых ветвей. Операция обрезки должна выполняться после генетических операций скрещивания и мутации.

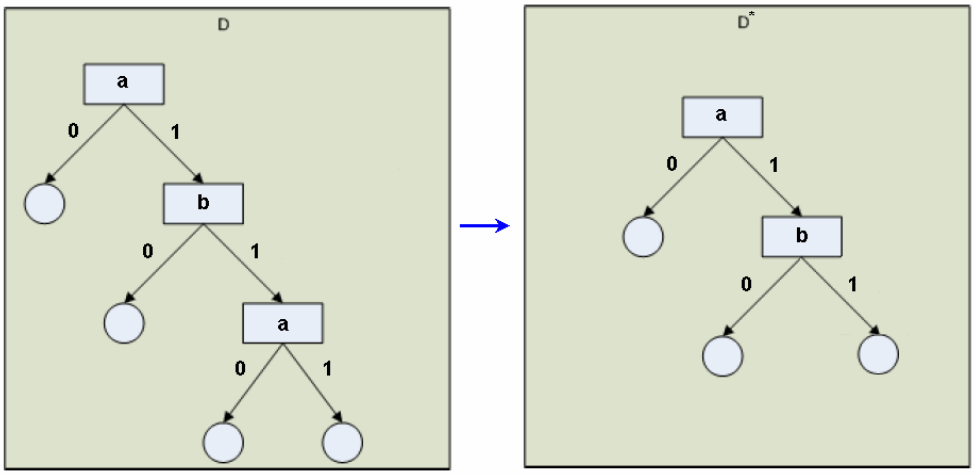

Рис. 4. Обрезка недостижимых ветвей

Апробация

Разработанный подход был протестирован на задаче «*Умный муравей-2*» [7]. Приведем описание задачи. Муравей находится на случайном игровом поле 32×32. При этом муравей видит перед собой некоторую область (рис. 5). Еда в каждой клетке с некоторой вероятностью μ. Значение μ является параметром задачи. Игра длится 200 ходов, за каждый из которых муравей может сделать одно из трех действий: поворот налево или направо, шаг вперед. Если после хода муравей попадает на клетку, где есть еда, то еда съедается. Целью задачи является построение стратегии поведения муравья, при которой матожидание количества съеденной еды максимально.

Рис. 5. Видимая область

Автомат управления муравьем в этой задаче имеет восемь (число видимых клеток) входных переменных – каждая из которых определяет, есть ли еда в клетке, соответствующей переменной. Все входные переменные имеют логический тип.

Предложенный подход сравнивался с генетическими алгоритмами, оперирующими над битовыми строками, и генетическими алгоритмами, оперирующими над таблицами переходов. Эксперимент заключался в сравнении полученных значений фитнесс-функции (объема съеденной еды) за фиксированное число шагов с одинаковыми настройками. Запуск алгоритмов производился со следующими настройками: стратегия отбора – элитизм, для размножения отбираются 25% популяции, имеющих наибольшее значение фитнесс-функции; частота мутации – 2%; размер популяции – 200 особей; число популяций – 100; фитнесс-функция – среднее значение съеденной еды на 200 случайных картах, карты внутри одной популяции совпадают, карты различных популяций различны.

Результаты эксперимента приведены в табл. 1. Последнее измерение фитнесс-функции осуществлялось на случайном наборе из 2000 карт.

Анализ показывает, что в случаях, когда важны значения почти всех предикатов (при $\mu = 0.01$), предлагаемый метод работает хуже известных. Это можно объяснить тем, что искомые автоматы плохо описываются деревьями решений. Однако, при больших значениях μ предложенный метод работает лучше, особенно при большом числе состояний. Таким образом, метод работает в большинстве случаев, за исключением ситуаций, когда функция переходов не может быть эффективно выражена деревом решений.

Интересна зависимость высоты дерева от параметров, приведенная в табл. 2. Можно заметить, что средняя высота деревьев падает с увеличением числа состояний. Это можно объяснить тем, что многие клетки видимые муравьем, были видны ему и раньше – например, после шага в области видимости муравья остается три клетки, а после поворота – пять клеток. Информация о клетках, увиденных раньше, при возрастании числа состояний может храниться в номере состояния. Учитывая то, что в автоматном программировании состояния играют роль неких характеристик уже совершенных действий, можно сказать, что предложенный метод хорошо подходит для генерации автоматов .

Таблица 1. Результаты эксперимента

μ	0.01			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	2.74	2.93	2.83	2.89
Таблица переходов	2.68	3.49	3.74	3.78
Предложенный метод	2.71	2.92	2.88	3.69
μ	0.02			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	7.66	8.38	7.95	6.98
Таблица переходов	6.12	7.32	7.24	7.28
Предложенный метод	7.68	8.04	7.32	8.25
μ	0.03			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	14.46	13.81	13.23	11.93
Таблица переходов	12.48	12.17	11.72	11.15
Предложенный метод	14.14	13.86	13.77	14.18
μ	0.04			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	19.11	18.68	17.47	15.10
Таблица переходов	17.18	15.94	15.03	13.68
Предложенный метод	18.28	20.28	18.60	20.18

Таблица 2. Средняя высота деревьев в автоматах

μ	0.01			
Число состояний	2	4	8	16
Средняя высота	5.5	3.25	2.38	3.63
μ	0.02			
Число состояний	2	4	8	16
Средняя высота	5.0	7.0	2.88	2.0

μ	0.03			
Число состояний	2	4	8	16
Средняя высота	4.5	2.5	1.88	1.56
μ	0.04			
Число состояний	2	4	8	16
Средняя высота	4.0	3.0	1.75	2.63

ЗАКЛЮЧЕНИЕ

В работе предложен метод генетического программирования, использующий высокоуровневое представление автоматов. Это позволяет значительно сократить требуемую память для большинства практических задач. Другим важным достоинством предлагаемого подхода является упрощение представления полученного автомата для человека.

Дальнейшее развитие исследований предполагает адаптацию подхода для генерации систем взаимодействующих автоматов и использование двоичных разрешающих диаграмм [8] для представления автоматов.

ЛИТЕРАТУРА

1. *Шалыто А. А.* Технология автоматного программирования / Труды первой Всероссийской научной конференции «Методы и средства обработки информации» М.: МГУ. 2003. http://is.ifmo.ru/works/tech_aut_prog/

2. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит. 2006.

3. *Andre D., Bennet F., Koza J.* Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem. 1996. http://citeseer.ist.psu.edu/andre96discovery.html

4. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.

5. *К. Шеннон.* Работы по теории информации и кибернетике. М.: Иностранная литература, 1963.

6. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.

7. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». СПбГУ ИТМО, 2007. http://is.ifmo.ru/works/_ant.pdf

8. *Bryant R. E.* Graph-based algorithms for boolean function manipulation // IEEE Transactions on Computers. 1986. V. 35, I. 8, pp. 667−691.