Статья опубликована в материалах 1-й Российской мультиконференции по проблемам управления – в сборнике докладов 4-й Всероссийской научной конференции "Управление и информационные технологии" (УИТ-2006). СПбГЭТУ "ЛЭТИ". 2006, с. 144-149.

ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ АВТОМАТИЧЕСКОГО ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ В ЗАДАЧЕ О «ФЛИБАХ»

П. Г. Лобанов, А. А. Шалыто

Санкт-Петербургский государственный
университет информационных технологий механики и оптики

Введение. Существуют сложные задачи, которые эффективно решаются с помощью конечных автоматов [1]. В большинстве случаев их построение выполняется эвристически.

Поэтому весьма актуальной являются задачи формализации методов построения автоматов и автоматизация этих методов.

Известно несколько задач (итерированная дилемма узников [2], задача про «умного» муравья [3], задачи синхронизации [4] и классификации плотности [5, 6] для клеточных автоматов), в которых генетические алгоритмы [7] позволяют автоматически строить автоматы, решающие указанные задачи.

В настоящей работе рассматривается еще одна задача этого класса – задача о флибах [8].

1. Задача о флибах. Рассматриваемая задача – это задача моделирования простейшего живого существа, которое способно предсказывать имеющие периодичность изменения простейшей окружающей среды. При этом живое существо моделируется конечным автоматом, а генетические алгоритмы позволяют автоматически построить автомат, который предсказывает изменения среды с достаточно высокой точностью. Таким образом, при решении данной задачи требуется построить «устройство», которое предсказывает изменения среды с наибольшей вероятностью.

В работе [8] для решения указанной задачи используется один из генетических алгоритмов. При этом точность предсказания построенного автомата является недостаточно высокой. Это во многом связано с подходом, используемым при построении следующего поколения особей (автоматов).

2. Постановка задачи. Одна из важнейших способностей живых существ – предсказание изменений окружающей среды. В качестве простейшей модели такого существа можно использовать конечный автомат. В работе [8] такие конечноавтоматные модели названы ***флибами*** (сокращение от *finite living blobs* – конечные живые капельки).

На вход флиба подается переменная, которая принимает одно из двух значений – ноль или единица. Эта переменная соответствует состоянию окружающей среды в текущий момент времени. Среда настолько проста, что имеет только два состояния. Флиб изменяет свое состояние и формирует значение выходной переменной, принимающей одно из двух указанных значений. Это значение соответствует возможному состоянию среды в следующий момент времени.

Задача флиба – предсказать какое на самом деле состояние окружающей среды наступит в следующий момент времени. Это можно выполнить благодаря периодичности изменений состояний среды. При этом считается, что чем выше вероятность предсказания изменений среды, тем больше вероятность у флиба выжить и оставить потомство.

3. Схемы работы генетических алгоритмов в задаче о флибах. Приведем схемы работы известного и предлагаемого алгоритмов.

3.1. Известный алгоритм. Для поиска оптимального предсказателя в работе [8] предлагается использовать генетический алгоритм.

1. Создается поколение случайных флибов.
2. Подсчитывается сколько изменений состояний окружающей среды правильно предсказывает каждый флибов.
3. Определяются худший и лучший предсказатели в поколении.
4. Лучший предсказатель *скрещивается* со случайно выбранным флибом.
5. Случайным образом определяется необходимость применения к полученному флибу *оператора мутации*. Если это необходимо, то указанный оператор применяется к флибу.
6. Худший предсказатель в поколении заменяется на флиб, полученный в результате скрещивания. После выполненной замены считается, что новое поколение сгенерировано.
7. Если один из флибов достиг уровня предсказания равного 100% или пользователь остановил программу, то алгоритм прекращает работу. В противном случае переходим к пункту 2.

Алгоритм создания случайных флибов, а также работа операторов одноточечного скрещивания и мутации, будут подробно рассмотрены в разд. 4, 7, 8.

3.2. Предлагаемый алгоритм. В настоящей работе предложено применять метод генерации нового поколения, отличающийся от метода, описанного в разд. 3.1. Предлагаемый метод использует турнирный отбор [9] и принцип элитизма (в новое поколение добавляется одна или несколько лучших особей из предыдущего поколения) [10].

Этот алгоритм имеет следующий вид.

1. Создается текущее поколение случайных флибов.
2. Подсчитывается сколько изменений состояний окружающей среды правильно предскажет каждый из этих флибов.
3. Строится новое поколение флибов:
   a. Создается пустое новое поколение и в него добавляется лучший предсказатель из текущего поколения.
   b. Случайным образом из текущего поколения выбираются две пары флибов.
   c. Из каждой пары выбирается лучший предсказатель.
   d. Лучшие предсказатели из указанных пар *скрещиваются*.
   e. Случайным образом определяется необходимость применения к полученному флибу *оператора мутации*. Если это необходимо, то указанный оператор применяется к флибу.
   f. К флибу применяется новая операция – «Восстановление связей между состояниями автомата».
   g. Флиб добавляется в новое поколение.
   h. Переходим к пункту b, если число флибов в новом поколении меньше числа флибов в текущем поколении.
4. Текущее поколение флибов заменяется новым.
5. Если число поколений меньше заданного пользователем, то переходим к пункту 2.

Число флибов в поколении будем называть размером поколения.

4. Реализация флиба. Опишем известный и предлагаемый методы реализации флиба.

4.1. Известный метод. Поведение флибов описывается с помощью таблиц переходов и выходов. В табл. 1 в качестве примера приведено описание поведения флиба с тремя состояниями A, B и C.

Таблица 1. Пример таблицы переходов и выходов флиба

Состояния	0	1
A	1, B	0, A
B	0, C	0, A
C	1, A	0, B

Представим эту таблицу в виде строки: 1B0A0C0A1A0B (такое представление используется в работе [8]). Отметим, что число элементов в приведенной строке в четыре раза больше числа состояний флиба. Пронумеруем состояния флиба и элементы строки, задающей его. Первому состоянию и первому элементу строки присвоим номер ноль. Если флиб находится в состоянии с номером `s` и текущее состояние среды `i`, то состояние, в которое флиб перейдет, содержится в элементе строки, задающей флиб. Этот элемент имеет номер `4s + 2i + 1`. Значение выходной переменной, генерируемой флибом, содержится в элементе с номером `4s + 2i`.

Для генерации случайного флиба требуется задать значения элементов строки случайным образом.

Приведем алгоритм генерации случайного флиба, заданного строкой.

1. Цикл по всем элементам строки, задающей флиб.
    a. Если номер элемента четный, то элементу присваивается одно из возможных состояний окружающей среды, выбранное случайным образом.
    b. Если номер элемента нечетный, то элементу присваивается одно из возможных состояний флиба, выбранное случайным образом.

4.2. Предлагаемый метод. В настоящей работе предлагается использовать другой способ кодирования флиба, реализуемый с помощью трех классов – `Flib`, `State` и `Branch`.

В качестве главного класса при реализации флиба применяется класс `Flib`. Классы `State` и `Branch` реализуют его состояния и переходы соответственно. Во всех этих классах имеется метод `Clone`, предназначенный для создания копий объектов.

Массив `_states` в классе `Flib` содержит состояния флиба. Поле `_curStateIndex` используется для хранения номера текущего состояния флиба в массиве `_states`. Число правильно предсказанных входных символов размещается в поле `_guessCount`. Метод `Step` переводит флиб в новое состояние и вычисляет количество правильно предсказанных символов. Метод `Nulling` устанавливает флиб в начальное состояние.

В массиве `_branches` класса `State` содержатся дуги переходов из данного состояния. Номер элемента в массиве соответствует входной переменной.

Переменные `_stateIndex` и `_output` класса `Branch` содержат номер состояния и значение выходной переменной. Константа `TARGET_COUNT` определяет количества значений, которые может принимать выходная переменная, генерируемая флибом.

Опишем работу программы генерации случайного флиба.

1. Создаются объекты, соответствующие состояниям флиба.
2. Для каждого объекта выполняется следующее:
    a. Для состояния формируются переходы из него.
    b. Для каждого перехода случайным образом определяются номер состояния, в которое переходит флиб и значение выходной переменной.

5. Генератор входного сигнала. В качестве входного сигнала для флибов (как и в работе [8]) используется повторяющаяся последовательность битов, названная битовой маской задающей среду. Эта маска в программе зациклена.

6. Оценочная функция. Лучшим предсказателем является тот флиб, который наиболее точно предсказывает входной сигнал. В качестве оценочной функции используется количество правильно угаданных символов. После формирования нового поколения все флибы в нем устанавливаются в начальное состояние. Затем для определения приспособленности флибов, на вход каждого из них подается несколько входных символов. После этого можно строить новое поколение решений, используя в качестве приспособленности флиба значение поля, содержащее число правильно предсказанных символов.

7. Алгоритмы построения оператора одноточечного скрещивания. Опишем известный и предлагаемый алгоритмы построения оператора одноточечного скрещивания.

7.1. Известный алгоритм. В работе [8] формирование нового флибов из двух существующих родительских выполняется с помощью оператора одноточечного скрещивания. Приведем описание алгоритма работы этого оператора для флиба, закодированного строкой длины `m`.

1. Выбирается случайное число `j` от `0` до `m-1`.
2. Элементы с номерами меньшими либо равными `j` из строки, задающей первый родительский флиб, копируются в строку, задающую новый флиб.
3. Элементы с номерами большими `j` из строки, задающей второй родительский флиб, копируются в строку, задающую новый флиб.

7.2. Предлагаемый алгоритм. Для применения оператора одноточечного скрещивания для флиба, закодированного в виде объекта класса, алгоритм, описанный в работе [8], требуется модифицировать. При этом он имеет следующий вид.

1. Случайным образом выбирается номер одного из состояний нового флиба.
2. В этот флиб добавляются состояния первого родителя, номера которых меньше выбранного номера, и состояния второго родителя, номера которых больше выбранного номера.
3. Формируется и добавляется новое состояние, образованное из состояний первого и второго родителей, которые соответствуют выбранному номеру. Алгоритм формирования состояния аналогичен алгоритму работы оператора одноточечного скрещивания, который описан в разд. 7.1.

8. Алгоритмы построения оператора мутации. Опишем известный и предлагаемый алгоритмы построения оператора мутации.

8.1. Известный алгоритм. Алгоритм оператора мутации, используемый в работе [8], имеет следующий вид.

1. Случайным образом выбирается элемент в строке, задающей флиб.
2. Если номер элемента четный (в элементе содержится значение выходной переменной, генерируемой флибом), то значение переменной инвертируется.
3. Если номер элемента нечетный (в элементе содержится состояние флиба), то номер текущего состояния флиба заменяется следующим.

8.2. Предлагаемый алгоритм. В настоящей работе предлагается использовать следующий алгоритм для оператора мутации.

1. Случайным образом выбирается состояние флиба.
2. Случайным образом выбирается дуга из выбранного состояния флиба.
3. Случайным образом определяется, что будет изменяться – значение выходной переменной, генерируемой флибом, или номер состояния, в которое он попадет при переходе по выбранной дуге.
    a. Если было определено, что изменяется значение выходной переменной, то ей присваивается значение состояния среды, выбранное случайным образом.

b. Если было определено, что изменяется номер состояния, то номеру состояния, в которое переходит флиб, присваивается номер случайно выбранного состояния флиба.

9. Восстановление связей между состояниями. При генерации нового поколения, применении операторов скрещивания и мутации, дуги переходов в флибах изменяются случайным образом. При таком изменении дуг, во флибах могут возникать состояния, в которые невозможно попасть из начального состояния при любой последовательности изменений состояний среды. Будем называть такие состояния *недоступными*. Состояния, в которые можно попасть из начального состояния при некоторой последовательности изменений состояний среды будем называть *доступными*. Алгоритм восстановления связей между состояния изменяет дуги переходов во флибе таким образом, чтобы в нем не было недоступных состояний.

Предлагаемый алгоритм имеет следующий вид.

1. Формируется список *доступных состояний* (метод `InitIndexesList`). Для этого используется функция рекурсивного обхода состояний `AddIndex`.
2. Выполняется цикл по всем состояниям. Если текущее состояние не входит в число *доступных*, то для него выполняются следующие операции:
   a. Случайным образом выбирается состояние из списка *доступных* состояний.
   b. Выбирается случайным образом одна из дуг из выбранного состояния.
   c. В текущем состоянии заменяется одна дуга из этого состояния на дугу, которая ведет в тоже состояние, что и дуга, выбранная в пункте b.
   d. Выбранная в пункте b дугу заменяется дугой, которая ведет в текущее состояние.
   e. Обновляется список *доступных состояний*. В него добавляется текущее состояние и все состояния, в которые можно попасть из него.

10. Общие требования к экспериментам. Все эксперименты проводились при `100` особях в поколении и вероятности применения оператора мутации `0,03`. Число воздействий среды на флиб также выбрано равным `100`. Поэтому число правильно предсказанных символов по величине равно точности предсказания символов в процентах. В таблицах с результатами экспериментов (табл. 2, 3) приводятся минимальная, максимальная и средняя точности предсказания автоматически построенных флибов.

Ниже описаны результаты двух экспериментов, которые отличаются между собой выбранным числом поколений, битовой маской и числом состояний флибов.

10.1. Первый эксперимент. Эксперимент производился для `400` поколений. Битовая маска, задающая среду, имеет вид – `1111010010111101001`. Число состояний флиба – `20`.

Результаты первого эксперимента приведены в табл. 2.

Таблица 2. Результаты первого эксперимента

	Восстановление связей между состояниями	Худший результат	Усредненный результат	Лучший результат
Известный алгоритм	–	70	78,3	88
Предложенный алгоритм	–	83	92,26	100
	+	84	93,48	100

10.2. Второй эксперимент. Эксперимент производился для `1600` поколений. Битовая маска, задающая среду, имеет вид – `10101111011000111101111110011001`. Число состояний флиба – `30`. Отметим, что битовая маска в этом эксперименте имеет большее число разрядов, чем в предыдущем. Результаты второго эксперимента приведены в табл. 3.

Таблица 3. Результаты второго эксперимента

	Восстановление связей между состояниями	Худший результат	Усредненный результат	Лучший результат
Известный алгоритм	–	70	75,86	87
Предложенный алгоритм	–	83	90,44	97
	+	86	92,72	97

Из рассмотрения таблицы следует, что предложенный алгоритм превосходит известный по всем параметрам.

Выводы. Проведенные эксперименты показали, что предложенный метод эффективнее известного как при различных битовых масках, задающих среду, так и при различном числе состояний флибов.

ЛИТЕРАТУРА

1. *Шалыто А.А.* Технология автоматного программирования // Мир ПК. 2003. № 10, с. 74–78. http://is.ifmo.ru/works/tech_aut_prog/
2. *Mitchell M.* An Introduction to Genetic Algorithms. MIT Press. Cambridge. MA, 1996.
3. *Langdon W., Poli R.* Better Trained Ants for Genetic Programming, 1998.
4. *Wolfram S.* A New Kind of Science. Wolfram Media, 2002.
5. *Mitchell M., Crutchfield J., Hraber P.* Evolving cellular automata to perform computations. Physica D. 1993. 75. pp.361–391.
6. *Бедный Ю.Д.* Применение генетических алгоритмов для решения одной задачи на клеточных автоматов. Задача классификации плотности для клеточных автоматов. СПбГУ ИТМО. Бакалаврская работа. 2006. http://is.ifmo.ru, раздел «Работы».
7. *Whitley D.* A Genetic Algorithm Tutorial. Statistics and Computing. 1994. 4, pp. 65-85. http://www.cs.colostate.edu/~genitor/Pubs.html.
8. *Воронин О., Дьюдни А*. Дарвинизм в программировании // Мой компьютер. 2004. № 35. http://www.mycomp.kiev.ua/text/7458.
9. *Miller B., Goldberg M.* Genetic algorithms, tournament selection, and the effects of noise // Complex Systems. 1995. 3, pp. 193–212.
10. *De Jong K.* An analysis of the behavior of a class of genetic adaptive systems. PhD thesis. Univ. Michigan. Ann Arbor, 1975.