# ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ ОБ «УМНОМ МУРАВЬЕ»

**П.Г. Лобанов**
**Научный руководитель – д.т.н., профессор А.А. Шалыто**

Предложен новый генетический алгоритм для решения задачи об «Умном муравье». Выполнены вычислительные эксперименты, демонстрирующие эффективность этого алгоритма по сравнению с известным.

## Введение

В последнее время все шире применяется автоматное программирование, в рамках которого поведение программ описывается с помощью конечных детерминированных автоматов (в дальнейшем автоматов [1]).

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение затруднительно. Известны задачи (итерированная дилемма узников [2], задача о «флибах» [3]), в которых применение генетических алгоритмов позволяет автоматизировать построение конечного автомата. Это один из первых шагов к автоматическому построению программ.

В данной работе рассматривается одна из таких задач – задача об «Умном муравье» [4]. Муравей должен за ограниченное количество ходов съесть всю еду, расположенную определенным образом на поле. Требуется построить конечный автомат, моделирующий поведение муравья. В работе [5] приводится генетический алгоритм для поиска такого автомата. Однако время поиска автомата довольно велико. В данной работе предлагается ряд методов, позволяющих ускорить поиск автомата и уменьшить количество состояний в нем.

## 1. Задача «умного муравья»

На тороидальном поле размером 32×32 вдоль определенной ломаной линии («тропы») расположено 89 ячеек с едой. Расположение ячеек с едой на «тропе» фиксировано. На рис. 1 приведено пример «тропы», называемый в литературе «тропой Джона Мьюра» (John Muir Trail) [5]. Черным цветом изображены ячейки с едой. Пустые ячейки на «тропе» изображены серым цветом. Серые ячейки невидимы для муравья.

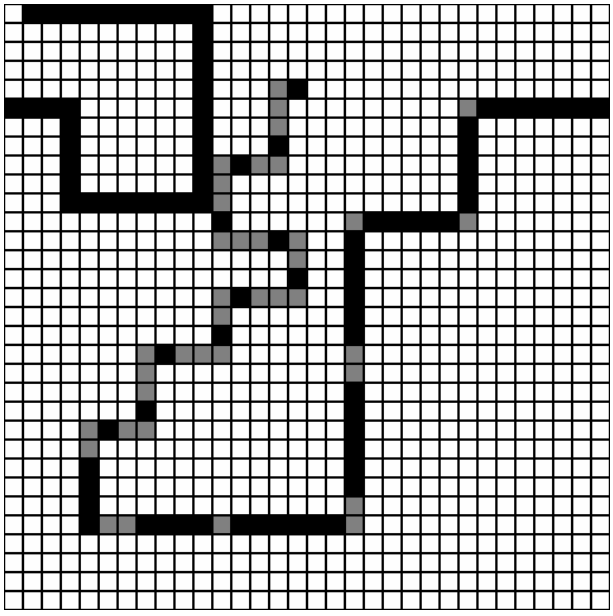

Рис. 1. Тропа Джона Мьюра (John Muir Trail)

Перед первым шагом муравей находится в верхней левой ячейке и смотрит направо. У него есть сенсор, который позволяет ему определить, есть ли еда в ячейке, находящейся непосредственно перед ним. За ход муравей может сделать одно из следующих трех действий – подвинуться вперед, повернуться налево, повернуться направо. Чтобы съесть еду в ячейке, муравью необходимо попасть в нее. После этого ячейка с едой считается пустой и становится для муравья неотличимой от изначально пустых ячеек. Муравей должен съесть всю еду за 200 ходов.

Сенсор муравья определяет значение битовой переменной (ноль – еды в ячейке перед муравьем нет, единица – еда в ячейке перед муравьем есть). Значение этой переменной используется в качестве входного воздействия для конечного автомата, моделирующего поведения муравья. После получения входного воздействия автомат генерирует выходную переменную (действие, которое может осуществить муравей – **вперед, налево** или **направо**) и переходит в новое состояние.

Пример автомата с четырьмя состояниями, моделирующего поведение муравья, построенный в [4] эвристически, приведен на рис 2. Однако муравей с таким поведением задачу не решает, так как за 200 ходов съедает только 42 единицы еды.

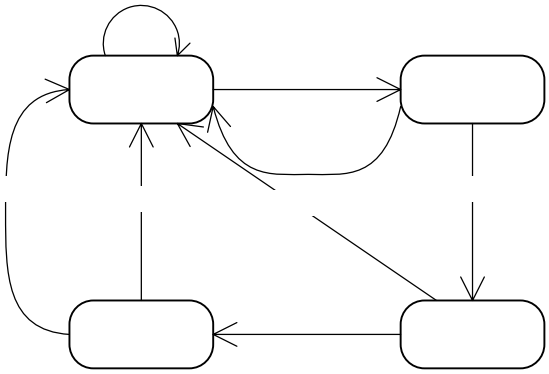

Рис. 2. Автомат с четырьмя состояниями, моделирующий поведение «Умного муравья»

## 2. Известный генетический алгоритм

Конечный автомат, моделирующий поведение муравья, является решением задачи об «Умном муравье». Чем больше ячеек с едой съедает муравей за 200 ходов, тем лучше решение (в качестве целевой функции [4, 5] используется количество съеденных единиц еды). Автомат задается набором состояний, пронумерованных от нуля. Из каждого состояния выходит две дуги, так как возможно два входных воздействия – ноль и единица. Дуга определяет номер состояния, в которое перейдет автомат при переходе по ней, и значение выходной переменной, генерируемой им при переходе (**вперед, налево** или **направо**).

### 2.1. Стратегия отбора

Как и в [5], в данной работе используется стратегия отбора особей для следующего поколения – отбор отсечением. При использовании такого отбора родительские решения выбираются из группы лучших решений текущего поколения. Размер этой группы решений (порог отсечения) обычно составляет от 1/100 до 1/3 от размера поколения. Все решения из группы лучших решений имеют одинаковые шансы быть выбранными в качестве родительских решений.

Приведем описание генетического алгоритма, использующего отбор отсечением.

1. Текущее поколение решений заполняется случайными решениями.
2. Из текущего поколения решений отбирается группа лучших решений.

3. Формируется новое поколение решений:
   a. Создается пустое новое поколение решение.
   b. Из группы лучших решений случайным образом выбирается пара решений.
   c. Формируется новое решение с помощью применения оператора скрещивания [4] к двум выбранным решениям.
   d. К новому решению применяются операторы мутации [4].
   e. Новое решение добавляется в новое поколение решений.
   f. Если размер нового поколения решений меньше размера текущего поколения переходим к пункту b.
4. Новое поколение становится текущим.
5. Если число созданных поколений меньше заданного пользователем, переходим к п. 2.

### 2.2. Оператор скрещивания

В данной работе, как и в работе [5], используется *n*–точечный оператор скрещивания. Приведем описание алгоритма работы этого оператора:

1. В качестве нового решения берется копия первого из выбранных решений.
2. Осуществляется цикл по всем состояниям нового решения. Для каждого состояния:
   a. Выполняется цикл по всем дугам состояния. Для каждой дуги:
      i. Случайным образом определяется, требуется ли изменить номер состояния, в которое переходит автомат по дуге. Если это требуется, то он изменятся на номер состояния из соответствующей дуги второго решения.
      ii. Случайным образом определяется, требуется ли изменить значение выходной переменной, генерируемое при переходе автомата по дуге. Если это требуется, то значение выходной переменной изменяется на значение выходной переменной, полученной из соответствующей дуги второго решения.

Для *n*–точечного оператора скрещивания задается вероятность его применения к каждому элементу автомата.

### 2.3. Оператор мутации

В данной работе, как и в работе [5] используется *n*–точечный оператор мутации. Приведем описание алгоритма этого оператора:

1. Осуществляется цикл по всем состояниям автомата:
   a. Выполняется цикл по всем дугам переходов в состоянии:
      i. Случайным образом определяется необходимость изменение индекса состояния, в которое автомат переходит по дуге.
      ii. Если требуется изменить индекс состояния, то он изменяется на индекс состояния, выбранный случайным образом.
      iii. Случайным образом определятся необходимость изменения значения выходной переменной, генерируемой автоматом при переходе по дуге.
      iv. Если требуется изменить значение выходной переменной, то оно изменяется на одно из возможных значений, выбранное случайным образом.

Для *n*–точечного оператора мутации задается вероятность его применения к элементу автомата. В отличие от обычного оператора мутации, такой оператор может изменить сразу несколько элементов в автомате.

## 3. Предложенные модификации алгоритма

### 3.1. Сортировка состояний в порядке использования

В большинстве решений, которые перебирает генетический алгоритм, автомат в процессе моделирования работы муравья не попадает в часть состояний, количество которых в данном алгоритме задается исходно. На поведение муравья влияют только те

состояния, из которых осуществляется переход по дуге хотя бы на одном из ходов моделирования. Далее такие состояния будем называть **используемыми состояниями**, а все остальные состояния – **неиспользуемыми состояниями**.

Приведем алгоритм сортировки состояний в порядке их использования:

1. Создается пустой словарь пар номеров: [«старый номер состояния» – «новый номер состояния»].
2. Моделируется поведение муравья. Перед каждым переходом по дуге выполняем следующее:
   - если в словаре нет пары, в которой первый элемент равен текущему номеру состояния, то в него добавляется пара [текущий номер состояния – количество пар в словаре].
3. Выполняется цикл по всем состояниям автомата. Для каждого состояния:
   - если в словаре нет пары, в которой первый элемент равен номеру состояния, то в него добавляется пара [номер состояния – количество пар в словаре].
4. Согласно словарю, изменяется порядок состояний и номера состояний в дугах переходов.

Состояния, для которых в словарь добавляются пары в пункте 2 – это и есть **используемые состояния**. Состояния, для которых в словарь добавляются пары в пункте 3 – это **неиспользуемые состояния**.

Используя этот алгоритм, несложно построить автомат, имеющий такое же поведение, как и автомат, найденный с помощью генетического алгоритма, но имеющий меньшее количество состояний. Для этого в автомате, состояния которого отсортированы в порядке использования, достаточно удалить все **неиспользуемые состояния** и заменить переходы в них переходами в состояние с номером ноль.

### 3.2. Всегда вперед, если впереди еда

После изучения автоматов, которые полностью решают задачу об умном муравье, было замечено, что эти муравьи всегда идут вперед, если перед ними еда. Чтобы достичь такого поведения, был добавлен еще один оператор мутации. Этот оператор изменяет значение выходной переменной на **«вперед»** для всех дуг переходов, помеченных входным воздействием единица (в ячейке перед муравьем еда)

### 3.3. Уменьшение количества состояний

В настоящей работе изменена целевая функция из [5] таким образом, что наиболее приспособленными оказываются те решения, у которых меньше **используемых состояний**. Для этого в целевую функцию (количество съеденных единиц еды) добавляется соотношение:

$$\frac{1}{1+useful\_state\_count},$$

где *useful_state_count* – число **используемых состояний**. Так как такая добавка всегда меньше единицы, то наиболее приспособленными будут те муравьи, которые съедают больше еды за 200 ходов. Однако, если количество съеденной пищи одинаково, то предпочтение будет отдаваться автоматам с меньшим числом **используемых состояний**, а как показано в разд. 3.1, число состояний в автомате может быть уменьшено до числа **используемых состояний**.

## 4. Эксперименты

Для проведения экспериментов была написана программа на языке C#, исходные коды которой будут приведены на сайте http://is.ifmo.ru в разделе «Статьи». Программа позволяет задавать размер и максимальное число поколений, устанавливать вероятно-

сти применения для операторов мутации и скрещивания. Также в ней можно изменить порог для стратегии отбора отсечением. При проведении экспериментов при необходимости можно изменять расположение ячеек с едой.

Все эксперименты, о которых идет речь ниже, проводились при размере поколения 10000 и пороге отсечения 2000. Вероятность применения оператора скрещивания к элементу автомата взята равной 0.04, а вероятность применения оператора мутации – 0.02. Число состояний автомата было ограничено двадцатью. Эксперименты проводились на ПК с процессором AMD Athlon 3800+.

В табл. 1 приведены результаты экспериментов, в которых использовалась целевая функция из работы [5]. Число поколений в экспериментах, в которых использовались предложенные в работе операторы мутации, было выбрано равным 100. Число поколений в экспериментах только с *n*–точечным оператором мутации не превышало двухсот. Каждый эксперимент без дополнительных операторов мутации длился примерно 310 с. (время построения двухсот поколений). Продолжительность каждого эксперимента с дополнительными операторами мутации составила около 270 с. (время построения ста поколений).

В столбце «Номер поколения» таблицы приведен номер поколения, в котором был найден лучший автомат в эксперименте. В столбце «Результат» приведено количество ячеек с едой, съеденных лучшим автоматом в эксперименте. Число **используемых состояний** лучшего автомата, полученного в эксперименте, приведено в столбце «Число используемых состояний». В столбце «Время поиска» приводится время в секундах, прошедшее с начала эксперимента, до построения поколения, в котором был найден лучший автомат в эксперименте.

| | Номер эксперимента | Номер поколения | Результат | Число используемых состояний | Время поиска |
|---|---|---|---|---|---|
| Без дополнительных операторов мутации | 1 | 137 | 87 | 15 | 225 |
| | 2 | 180 | 88 | 15 | 281 |
| | 3 | 73 | 88 | 12 | 137 |
| | 4 | 72 | 86 | 12 | 114 |
| | 5 | 68 | 89 | 12 | 114 |
| С дополнительными операторами мутации | 1 | 47 | 89 | 12 | 154 |
| | 2 | 49 | 89 | 11 | 160 |
| | 3 | 56 | 89 | 13 | 182 |
| | 4 | 51 | 89 | 12 | 145 |
| | 5 | 67 | 89 | 13 | 189 |

Таблица 1. Результаты экспериментов с использованием стандартной целевой функции

Как следует из приведенных данных, использование передоложенных операторов мутации позволило при всех запусках найти автомат, решающий поставленную задачу. Известный алгоритм с этим не справился за вдвое большее число поколений в четырех экспериментах из пяти. Это, по всей видимости, связано с тем, что в данных экспериментах размер поколения был значительно меньше размера поколения, использующегося в работе [5].

В табл. 2 приведены результаты экспериментов, проводившихся с использованием предложенной в работе целевой функцией. Число поколений в экспериментах проводившихся, с предложенными в работе операторами мутации, было ограничено сотней. Число поколений в экспериментах, в которых использовался только *n*–точечный оператор мутации, не превышало двухсот. Лучшим результатом считался автомат, который съедает максимальное количество ячеек с едой и имеет при этом минимальное

количество **используемых состояний**. Каждый эксперимент без дополнительных операторов мутации длился примерно 320 с (время построения двухсот поколений). Продолжительность каждого эксперимента с дополнительными операторами мутации составила около 290 с (время построения ста поколений).

| | Номер эксперимента | Номер поколения | Результат | Число используемых состояний | Время поиска |
|---|---|---|---|---|---|
| Без дополнительных операторов мутации | 1 | 143 | 89 | 11 | 237 |
| | 2 | 144 | 89 | 9 | 263 |
| | 3 | 138 | 88 | 10 | 259 |
| | 4 | 99 | 89 | 13 | 152 |
| | 5 | 158 | 89 | 10 | 252 |
| С дополнительными операторами мутации | 1 | 61 | 89 | 10 | 161 |
| | 2 | 85 | 89 | 10 | 263 |
| | 3 | 70 | 89 | 10 | 227 |
| | 4 | 63 | 89 | 10 | 176 |
| | 5 | 96 | 89 | 9 | 269 |

Таблица 2. Результаты экспериментов с использованием предложенной целевой функции

Как следует из приведенных в табл. 2 данных, изменение целевой функции позволило получить автоматы с меньшим числом **используемых состояний** по сравнению с предыдущим случаем. Необходимо отметить, что алгоритм, использующий дополнительные операторы мутации, работает более стабильно и с измененной целевой функцией.

Сравнив данные, приведенные в табл. 1 и 2, отметим, что изменение целевой функции не только не ухудшило сходимость алгоритма, но даже улучшило ее.

В табл. 3 приведен автомат, который решает задачу об «Умном муравье» для «тропы Джона Мьюра» и имеет минимальной число **используемых состояний** из построенных автоматов.

| | Перед муравьем нет еды | | В ячейке перед муравьем еда | |
|---|---|---|---|---|
| Номер состояния | Номер следующего состояния | Выходная переменная | Номер следующего состояния | Выходная переменная |
| 0 | 6 | **направо** | 1 | **вперед** |
| 1 | 2 | **вперед** | 2 | **вперед** |
| 2 | 8 | **вперед** | 3 | **вперед** |
| 3 | 2 | **вперед** | 4 | **вперед** |
| 4 | 6 | **направо** | 5 | **вперед** |
| 5 | 4 | **налево** | 3 | **вперед** |
| 6 | 7 | **направо** | 1 | **вперед** |
| 7 | 2 | **налево** | 1 | **вперед** |
| 8 | 0 | **налево** | 4 | **вперед** |

Таблица 3. Таблица переходов и выходов для лучшего из найденных автоматов

Отметим, что число состояний в автомате, приведенном в таблице 3 (девять состояний) меньше числа состояний в автоматах, полученных в работах [5] и [6] (одиннадцать состояний), которые, кроме того, содержат опечатки.

## Заключение

В работе предложены два новых оператора мутации и целевая функция для генетического алгоритма из работы [6]. Применение предложенных методов позволило сократить время поиска автомата, решающего задачу об «Умном муравье» для «тропы Джона Мьюра».

Написана программа, позволяющая производить эксперименты как с предложенными методами, так и без них. Приведенные экспериментальные данные показывают эффективность предложенных методов по сравнению с известным.

Предложенная в работе целевая функция позволила уменьшить количество состояний у автоматов, получаемых с помощью генетического алгоритма. Это также подтверждается экспериментальными данными.

В заключение отметим, что задача поиска муравья с наименьшим количеством состояний не ставилась. При этом отметим, что известен муравей, решающий задачу, который имеет восемь состояний [7].

## Литература


1. Шалыто А.А. Технология автоматного программирования. / Труды первой Всероссийской конференции «Методы и средства обработки информации». М.: МГУ. 2003. http://is.ifmo.ru/works/tech_aut_prog
2. Mitchell M. An Introduction to Genetic Algorithms. MIT Press. Cambridge. MA, 1996.
3. Лобанов П.Г., Шалыто А.А. Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о «флибах». / Материалы 1-й Российской мультиконференции по проблемам управления. Сборник докладов 4-й Всероссийской научной конференции «Управление и информационные технологии» (УИТ-2006). СПбГЭТУ «ЛЭТИ», 2006. С. 144–149. http://is.ifmo.ru/works/_flib.pdf
4. Koza J.R. Genetic programming. On the Programming of Computers by Means of Natural Selection. The MIT Press, 1998.
5. Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A. The Genesys System: Evolution as a Theme in Artificial Life. // In Langton et al. (ALife II). 1992. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
6. Angeline P.J., Pollack J. Evolutionary Module Acquisition. // Proceedings of the Second Annual Conference on Evolutionary Programming. 2003. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
7. Chambers L. Practical handbook of genetic algorithms, Boca Raton, CRC Press, 1995.