Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. Вып. 53. Автоматное программирование. 2008, с. 114–122.

УДК 004.4`242

МЕТОДЫ ОПТИМИЗАЦИИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ

П. Г. Лобанов

Предложены три метода, позволяющие повысить скорость сходимости стандартных генетических алгоритмов для класса задач, в которых решением является конечный автомат.

Введение

Генетические алгоритмы применяются при решении широкого круга задач. Однако при использовании классического генетического алгоритма часто требуются очень большие вычислительные ресурсы, чтобы получить решение с необходимой степенью точности. В ряде случаев классический алгоритм может не справиться с поставленной задачей за разумное время.

Для повышения скорости сходимости генетического алгоритма и улучшения устойчивости его работы, как правило, применяются модификации, учитывающие особенности решаемой задачи. В работах [1, 2] рассматриваются оценочные функции и способы их адаптации для улучшения эффективности генетических алгоритмов. В работе [3] на примере задачи коммивояжера рассматривается, как зависит эффективность эволюционных методов от выбора структуры хромосом. Необходимо отметить, что выбор структуры хромосом определяет, какие классы операторов скрещивания и мутации могут использоваться.

В данной работе рассматривается три новых модификации генетического алгоритма для класса задач, в которых решением является конечный автомат. Эти модификации могут использоваться как независимо друг от друга, так и совместно.

1. Описание методов

Приведем описание методов, предложенных в данной работе.

1.1. Использование автоматов с флагами

Поведение конечного автомата зависит от его состояний и переходов между ними. Каждый переход определяет, в какое состояние попадет автомат при некотором условии (функция, принимающая значения входных переменных и возвращающая булевское значение) и какие действия выполнятся при переходе. Иногда в качестве входных переменных на переходе используются значения выходных переменных, генерируемых самим автоматом. Такие переменные в работе [4] названы флагами. Значений флагов также влияют на окружение, в котором работает автомат.

Чем более сложную задачу решает автомат, тем, как правило, больше число его состояний. В книге [4] определен класс автоматов с флагами, в которых используется вектор многозначных флаговых переменных `F`, который несет информацию о предыдущих состояниях автомата. Если автоматы без флагов зависят от предыстории (предыдущего состояния), то автоматы с флагами зависят от глубокой предыстории (не только от предыдущего состояния). Автоматы данного класса взаимодействуют с внешними ячейками многозначной памяти.

Флаговые переменные одновременно используются и в качестве входных, и в качестве выходных переменных. Вместе с входными переменными и текущим состоянием автомата, они определяют, в какое состояние перейдет автомат в следующий момент времени и какие выходные переменные он сформирует.

Как показано в работе [4], использование флаговых переменных часто позволяет уменьшить число состояний в автомате. Это, правда, приводит к усложнению его графа переходов. Становится труднее понять алгоритм работы автомата. Однако этот недостаток не является существенным для случая автоматической генерации автоматов.

Рассмотрим, как можно использовать автоматы с флагами при решении задачи о флибах [5]. Поведение флиба описывается с помощью граф переходов. На рис. 1 в качестве примера изображен флиб с тремя состояниями **A**, **B** и **C**.

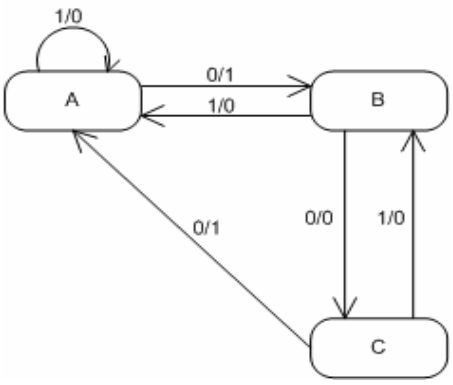

Рис. 1. Граф переходов для флиба с тремя состояниями

Во флибе флаговые переменные можно использовать для хранения предыдущих состояний окружающей среды. Пусть x и $F = (f_1, \ldots, f_n)$ – состояние среды и вектор флаговых переменных на текущий момент времени, соответственно. Тогда вектор флаговых переменных для следующего момента времени будет иметь вид $F' = (f_1' = x, f_2' = f_1, \ldots, f_n' = f_{n-1})$.

На рис. 2 изображен граф переходов для флиба с тремя состояниями и одной флаговой переменной – это переменная, сформированная автоматом на предыдущем шаге, на этом шаге используется в качестве одной из входных переменных.

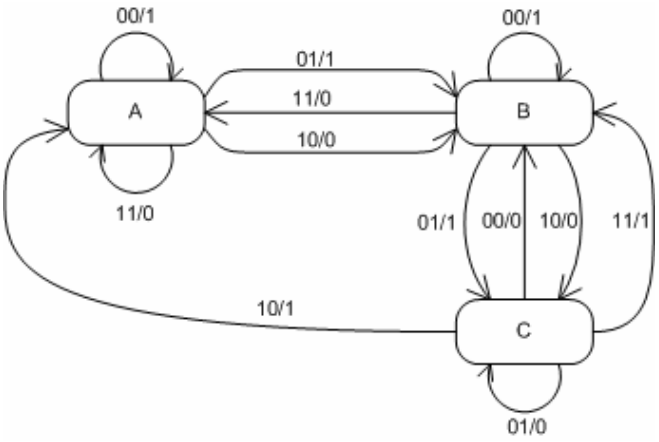

Рис. 2. Граф переходов для флиба с тремя состояниями и одной флаговой переменой

В общем случае входная переменная может принимать произвольное число значений. Пусть существует m допустимых комбинации значений входных переменных, перенумерованных от 0 до $m-1$. Тогда при n флаговых переменных число переходов из каждого состояния автомата будет равно m^{1+n}. Номер перехода, который должен использоваться в текущий момент времени, можно вычислить по формуле $x+m^1 f_1+m^2 f_2+\ldots+m^n f_n$, где x – номер текущей комбинации значений входных переменных, а $F=(f_1,\ldots,f_n)$ – вектор флаговых переменных.

Флаговые переменные позволяют существенно уменьшить число состояний в автомате. Рассмотрим эту особенность на простом примере. Пусть требуется построить флиб для простейшей среды, заданной с помощью повторяющейся битовой маски: 11100. При использовании обычного автомата понадобится, по крайней мере, три состояния, чтобы флиб смог предсказывать изменения среды со сто процентной точностью. Пример такого флиба приведен на рис. 3.

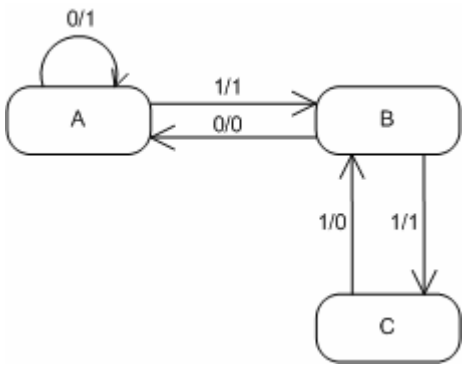

Рис. 3. Граф переходов флиба для среды, заданной битовой маской 11100

Начальное состояние автомата — **A**. Для упрощения понимания переходы между состояниями, которые не используются в случае рассматриваемой среды, были удалены.

При использовании двух флаговых переменных достаточно одного состояния, чтобы построить флиб с точностью предсказания, равной ста процентам. Граф переходов для такого флиба с флагами изображен на рис. 4. Как и в предыдущем случае, все неиспользуемые переходы были удалены, чтобы упростить рисунок.

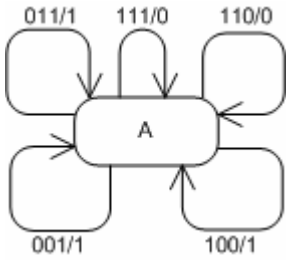

Рис 4. Граф переходов флиба с двумя флаговыми переменными для среды, заданной битовой маской 11100

Как видно из приведенного примера, использование флаговых переменных позволяет уменьшить число состояний автомата за счет увеличения числа переходов между ними.

1.2. Алгоритм восстановления связей между состояниями

При генерации генетическим алгоритмом нового поколения решений, применении операторов скрещивания и мутации переходы в автоматах изменяются случайным образом. При таком изменении переходов в автомате, как правило, возникают состояния, в которые невозможно попасть из начального состояния при любой последовательности значений входных переменных. Будем называть такие состояния **недостижимыми**. Состояния, в которые можно попасть из начального состояния при некоторой последовательности значений входных переменных, будем называть **достижимыми**. Алгоритм восстановления связей между состояниями изменяет переходы в автомате таким образом, чтобы в нем не было **недостижимых состояний**.

Рассмотрим алгоритм восстановления связей между состояния на примере задачи о флибах. Предлагаемый алгоритм имеет следующий вид.

1. Формируется список **достижимых состояний**. Для этого можно, например, использовать обхода графа в глубину.
2. Рассматривается каждое состояние. Если рассматриваемое состояние не входит в число **достижимых состояний**, то для него выполняются следующие операции:
   a. Случайным образом выбирается состояние из списка **достижимых состояний**.
   b. Выбирается случайным образом один переход из выбранного состояния.
   c. В рассматриваемом состоянии заменяется один переход из этого состояния на переход, который ведет в то же состояние, что и переход, выбранный в пункте b.
   d. Выбранный в пункте b переход заменяется переходом, который ведет в рассматриваемое состояние.
   e. Обновляется список **достижимых состояний**. В него добавляется рассматриваемое состояние и все состояния, в которые можно попасть из него.

Рассмотрим работу описанного выше алгоритма на примере. На рис. 5 изображен граф переходов для флиба, полученного в результате работы генетического алгоритма. Начальное состояние автомата – **A**.

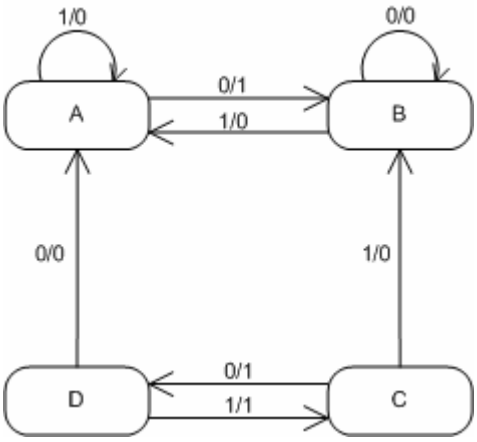

Рис. 5. Граф переходов для флиба, полученного в результате работы генетического алгоритма

Для автомата, граф переходов которого изображен на рис. 5, состояния **A** и **B** являются **достижимыми состояниями**, в то время как состояния **C** и **D** – **недостижимыми**. На рис. 6 **достижимые состояния** закрашены светло-серым цветом, **недостижимые состояния** изображены темно-серыми.

Алгоритм восстановления связей между состояниями перебирает все недостижимые состояния. Первым идет состояние **C**. Случайным образом выбирается состояние из списка **достижимых состояний** (например, состояние **B**). Выбирается случайным образом переход из состояния **B**. В качестве примера выберем переход, который соответствует значению входной переменной `1` и ведет в состояние **A**. Он меняется на переход, соответствующий такому же значению выходной переменной и ведущий в состояние **C**. Один из переходов, ведущих из состояния **C** (для примера возьмем переход, соответствующий значению входной переменной `1`), заменяется аналогичным переходом, ведущим в состояние **A**.

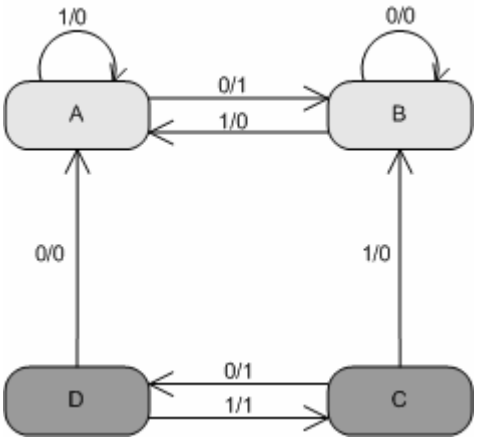

Рис. 6. **Достижимые** (светло-серые) и **недостижимые** (темно-серые) состояния автомата

На рис. 7 показан результат добавления состояния **C** в список **достижимых состояний**. Серым пунктиром показаны переходы, которые были удалены в результате этой операции. Черным пунктиром обозначены новые переходы. В состояние **C** автомат попадет из начального состояния **A**, если на его вход, например, будет подана последовательность значений входной переменной вида `01`.

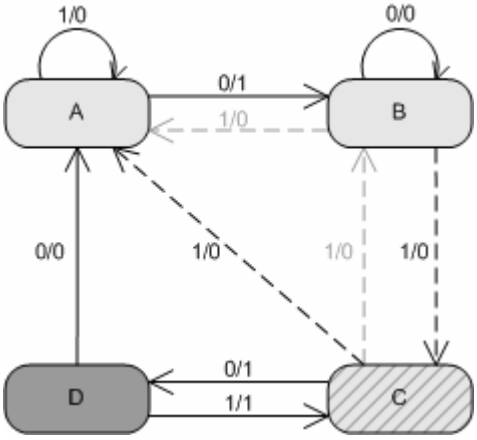

Рис. 7. Добавление состояние **C** в список **достижимых состояний**

После замены переходов состояние **C** стало **достижимым**. В состояние **D** можно попасть из состояния **C**, следовательно, состояние **D** также стало **достижимым**. Действительно, в состояние **D** можно попасть из начального состояния **A**, если подать, например, последовательность значений входной переменной вида `0011`.

Так как после обновления списка **достижимых состояний** все состояния стали достижимыми, то алгоритм завершает свою работу.

Алгоритм восстановления связей между состояниями можно использовать как в качестве дополнительного, так и в качестве основного оператора мутации. Иногда может оказаться целесообразным проверять, не ухудшилась ли приспособленность особи после выполнения восстановления связей между состояниями. Если приспособленность особи ухудшилась, то граф переходов возвращается в исходное состояние.

Часто может оказаться целесообразным увеличить число **достижимых состояний** автомата, оставив часть состояний недостижимыми. Такой подход требует меньше вычислительных ресурсов и вносит меньше изменений в поведение, описываемое этим автоматом.

1.3. Алгоритм сортировки состояний в порядке использования

Во многих задачах в большинстве решений, которые перебирает генетический алгоритм, автоматы в процессе вычисления значения оценочной функции не попадают в часть состояний. Далее состояния, из которых выполняется переход хотя бы на одном шаге моделирования (вычисления оценочной функции), будем называть **используемыми состояниями**, а все остальные состояния – **неиспользуемыми состояниями**.

Если при вычислении оценочной функции на вход автомата подаются все возможные последовательности входных значений, то **неиспользуемые состояния** и переходы, связанные с ними, не влияют на поведение автомата. Примерами таких задач могут служить задача об умном муравье [6] и задача о флибах [5].

Приведем алгоритм сортировки состояний в порядке их использования.

1. Создается пустой словарь пар номеров: [«старый номер состояния» – «новый номер состояния»].
2. Моделируется работа автомата. Перед каждым переходом 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 которые ведут переходы.

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

Рассмотрим работу алгоритма сортировки состояний в порядке использования на примере задачи о флибах. Пусть среда, заданной с помощью повторяющейся битовой маски: `11100`. В процессе работы генетического алгоритма был получен флиб, граф переходов для которого изображен на рис. 8.

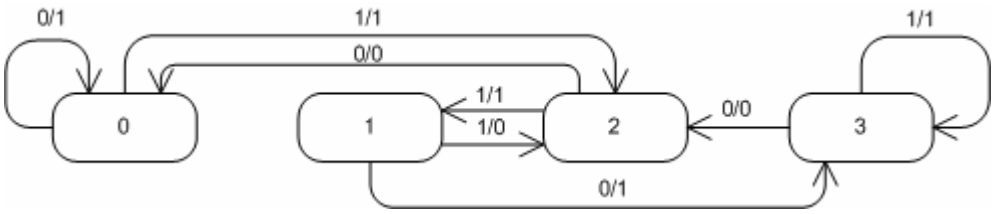

Рис. 8. Граф переходов флиба, полученного в процессе работы генетического алгоритма (среда задана битовой маской 11100)

Состояния флиба пронумерованы цифрами от нуля до трех. Начальное состояние – **0**.

Создается пустой словарь пар [«старый номер состояния» – «новый номер состояния»].

Начинается моделирование работы флиба. На вход автомату подается **1**. Автомат переходит в состояние **2,** и в словарь добавляется пара **[0, 0]**. На следующем шаге моделирования автомат переходит в состояние **1,** и в словарь добавляется пара **[2, 1]**. Далее автомат переходит в состояние **2,** и в словарь добавляется пара **[1, 2]**. При переходе из состояния **2** в состояние **0** в словарь ничего не добавляется, так как для состояний **2** в словаре уже есть пара. При дальнейшем моделирование работы флиба пары в словарь не добавляются, так как при среде, заданной битовой маской `11100`, флиб никогда не попадет в состояние **3**, для всех остальных состояний пары в словаре уже есть.

После моделирования работы флиба словарь пар номеров будет иметь следующий вид: **{[0, 0], [1, 2], [2, 1]}**.

Осуществляется цикл по всем состояниям. Так как в словаре нет пары только для состояний с **3**, то в него добавляется пара **[3, 3]**. Теперь словарь принимает следующий вид: **{[0, 0], [1, 2], [2, 1], [3, 3]}**.

Осталось изменить порядок состояний и номера состояний в переходах согласно словарю. Состояния **1** и **2** меняются местами и номерами. Граф переходов для флиба, получившегося в результате работы алгоритма, приведен на рис. 9.

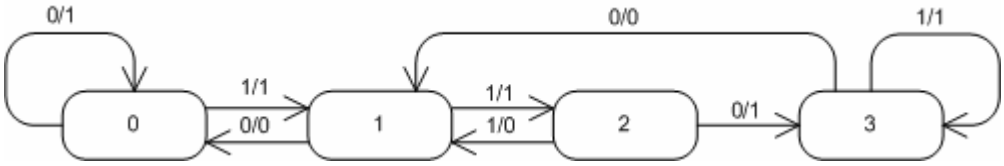

Рис. 9. Граф переходов флиба после применения алгоритма сортировки состояний (среда задана битовой маской 11100)

После применения алгоритма сортировки состояний легко построить автомат, имеющий такое же поведение, как и автомат, найденный с помощью классического 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 начальное состояние.

В нашем случае одно **неиспользуемое состояние** – **3**. В состояние **3** ведет только один переход из состояния **2**. Удалим состояние **3** и заменим переход в него на переход в состояние **0**. Результат изображен на рис. 10.

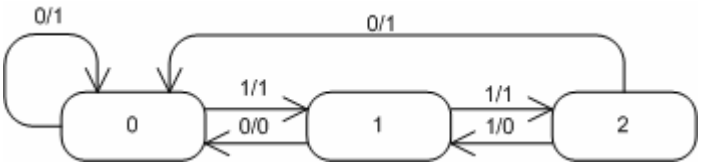

Рис. 10. Граф переходов флиба после удаления **неиспользуемых состояний** (среда заданна битовой маской 11100)

Так как при моделировании работы автомата переход из состояния **2,** соответствующий значению входной переменной **0** ни разу не выполнялся, то поведение автомата не изменилось.

2. Особенности методов оптимизации генетических алгоритмов при построении конечных автоматов

Использование автоматов с флагами вместо обычных автоматов позволяет увеличить число переходов между состояниями и уменьшить число состояний. Благодаря этому в автоматах уменьшается число **недостижимых состояний**. Поведение автоматов с флагами меньше меняется при использовании оператора мутации, чем поведение автоматов без флагов.

Использование алгоритма восстановления связей между состояниями позволяет увеличить среднее число состояний в автоматах поколения. Чем больше состояний у автомата, тем меньшему числу изменений он подвергается, и, как следствие, тем меньше меняется его поведение. В итоге генетический алгоритм начинает отдавать предпочтение автоматам с большим числом состояний и, соответственно, с более сложным поведением. Это полезно, если приближенным решением задачи является автомат с небольшим числом состояний, а любое лучшее решение должно иметь значительно большее число состояний. В этом случае генетический алгоритм может «застрять» в локальном оптимуме. Эта проблема возникает из-за того, что стандартный оператор мутации (изменяет значение выходной переменной на переходе или состояние, в которое осуществляется переход) при однократном применении крайне редко значительно увеличивает число состояний в автомате. Алгоритм восстановления связей между состояниями решает эту проблему.

Алгоритм сортировки состояний в порядке использования приводит к одному виду автоматы с одинаковым поведением, но разным порядком нумерации состояний. Таким образом, использование алгоритма сортировки состояний автомата в порядке использования позволяет сократить пространство поиска, в котором генетический алгоритм перебирает решения.

При n флаговых переменных число переходов из каждого состояния автомата будет равно m^{1+n}, где m – число допустимых комбинации значений входных переменных. Объем памяти, необходимой для хранения графа переходов автомата, пропорционален числу переходов из каждого состояния автомата. Создание новой особи из двух родительских также выполняется за время, пропорциональное числу переходов из каждого состояния автомата. Чтобы генетический алгоритм перебирал решения для автоматов с флагами со скоростью не меньшей, чем для автоматов без флагов, число состояний в автоматах с флагами должно не больше s/m^{1+n}, где s – максимальное число состояний для автомата без флагов.

Все описанные в разд. 1 методы оптимизации генетических алгоритмов подразумевают, что решения, с которыми ведется работа, хранятся в виде графа переходов. Эти ме-

тоды также можно использовать, если структура хромосомы особи позволяет преобразовать ее в граф переходов, а затем выполнить обратную операцию. Например, преобразуем битовую строку в граф переходов, выполняем восстановление связей между состояниями и кодируем получившийся граф переходов с помощью битовой строки. Необходимо отметить, что такой подход связан с увеличением требований к вычислительным ресурсам.

Использовать алгоритм сортировки состояний нецелесообразно, если при вычислении значений оценочной функции на вход автоматам одного поколения подаются различные последовательности значений входных переменных.

Заключение

В работе предложены три новых модификации генетических алгоритмов, которые могут быть использованы для задач, решением которых является конечный автомат. Подробно рассмотрены их алгоритмы.

Рассмотрены особенности предложенных методов. Эти методы могут использоваться как независимо друг от друга, так и совместно.

Литература

1. Huang D. MS Thesis Preproposal: Adaptive Incremental Fitness Evaluation in Genetic Algorithms. NY: Rochester, 2005. http://www.cs.rit.edu/~dxh6185/downloads/MS_Thesis/Documents/Presentation.pdf
2. Linton R. Adapting binary fitness functions in genetic algorithms / Proceedings of the 42nd annual Southeast regional conference. NY: ACM Press. 2004. pp. 391–395.
3. Bryant K. Genetic Algorithms and the Traveling Salesman Problem. Harvey Mudd College: Department of Mathematics, 2000. http://www.math.hmc.edu/math197/archives/2001/kbryant/kbryant-2001-thesis.pdf
4. Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб: Наука. 1998, 626 с.
5. Лобанов П. Г., Шалыто А. А. Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о «Флибах» / 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
6. Царев Ф. Н., Шалыто А. А. Применение генетического программирования для генерации автомата в задаче об «Умном муравье» / Сборник трудов IV-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007, с. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf