Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Ю. Д. Бедный, А. А. Шалыто

Применение генетических алгоритмов для построения автоматов в задаче “Умный муравей”

Редакция 3

Санкт-Петербург
2007

Оглавление

1. Постановка задач

В рамках работы:

1. Рассмотрен ряд задач, в которых возможно использование генетических алгоритмов (ГА) для построения автоматов. Для подробного анализа выбрана одна из наиболее известных в этой области задач – “Умный муравей” (“Artificial Ant”) [1–4].
2. Реализован фреймворк для исследований по этой задаче, а также различных её модификаций.
3. С использованием фреймворка получены решения задачи “Умный муравей” и выполнено сравнение с известными решениями.
4. Рассмотрены модификации задачи “Умный муравей”. Поставлен ряд вопросов и обозначены направления дальнейших исследований.

2. Обзор задачи “Умный муравей”

Приведем краткое описание задачи “Умный муравей” на основе работ [1, 2]. Используется двумерный тор размером 32 на 32 клетки. На некоторых клетках поля расположены яблоки – черные клетки на рис. 1. Яблоки расположены вдоль некоторой ломаной линии, но не на всех ее клетках. Клетки ломаной, на которых их нет – серые. Белые клетки – не принадлежат ломаной и не содержат яблок. Всего на поле 89 яблок.

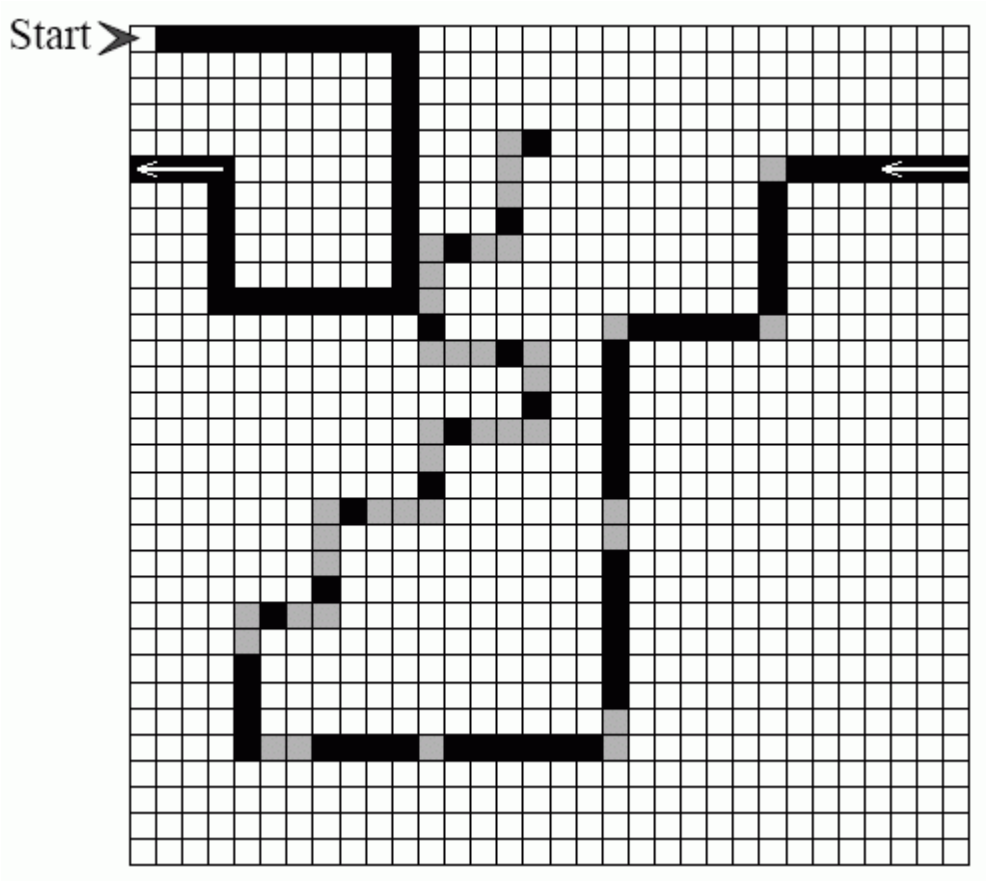

Рис. 1. Поле с яблоками

В клетке с пометкой “Start” находится муравей. Он занимает клетку поля и смотрит в одном из четырех направлений (север, запад, юг, восток). В начале игры муравей смотрит на восток. Он умеет определять находится ли яблоко непосредственно перед ним. За один ход муравей совершает одно из четырех действий:

- идет вперед на одну клетку, съедая яблоко, если оно было перед ним;
- поворачивается вправо;
- поворачивается влево;
- стоит на месте.

Съеденные муравьем яблоки не восполняются. Муравей жив на всем протяжении игры – еда не является необходимым ресурсом для его существования. Никаких других персонажей, кроме муравья, на поле нет. Ломаная **строго задана**. Муравей может ходить по любым клеткам поля.

На рис. 2 изображены муравей, яблоки и часть поля.

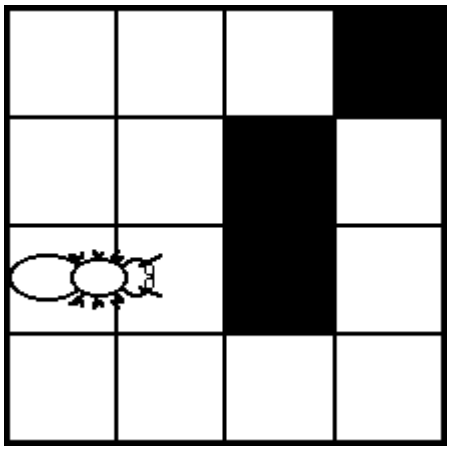

Рис 2. Муравей и яблоки

2.1. Цель игры

Игра длится 200 ходов, на каждом из которых муравей совершает одно из четырех описанных выше действий. В конце игры подсчитывается количество яблок, съеденных муравьем. Это значение – результат игры.

Цель игры – создать муравья, который за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое количество яблок, заканчивают игру с одинаковым результатом вне зависимости от числа ходов, затраченных каждым из них на процесс еды. Однако эта задача может иметь различные модификации, например, такую, в которой при одинаковом количестве съеденных яблок, лучшим считается муравей, съевший яблоки за меньшее число ходов. Ниже будет показано, что поведение муравья может быть задано конечным автоматом. При этом может быть поставлена задача о построении автомата с минимальным числом состояний для муравья, съедающего все яблоки, или автомата для муравья, съедающего максимальное количество яблок при заданном числе состояний.

Ввиду фиксированной топологии и размера поля (тор размером 32 на 32), фиксированного расположения яблок и их невосполнимости, а также примитивности поведения муравья, может показаться, что задача тривиальна.

Для того чтобы убедиться, что это не так, предлагается

сконструировать муравья, который съест хотя бы 82 яблока за 200 ходов. Приведенное выше число 82 выбрано неслучайно, так как существует простая стратегия с результатом 81, которая описана в следующем разделе.

2.2. Конечный автомат, как способ задания поведения муравья

Один из возможных способов задания поведения муравья – конечный автомат.

Придумать муравья с достаточным большим числом состояний несложно. В этом случае муравей имеет следующую стратегию: идет вперед пока есть еда, а на изгибе поворачивает, переходя в новое состояние. Число состояний в этом случае равно длине ломанной (126) плюс число поворотов в ней (20).

Конечный автомат, изображенный на рис. 3, содержит всего пять состояний.

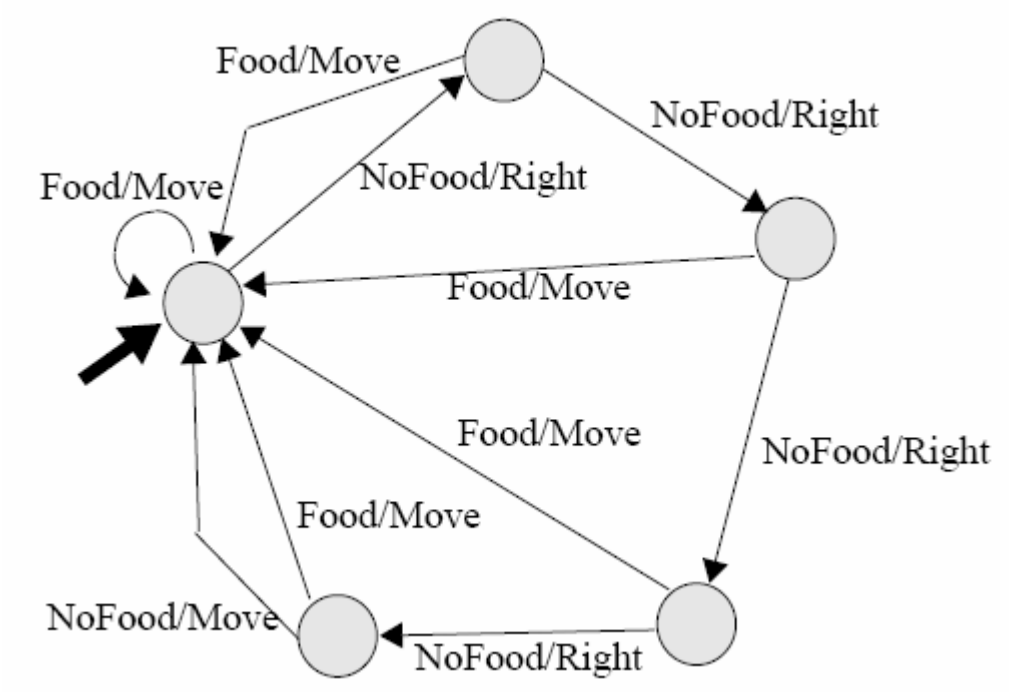

Рис 3. Конечный автомат, задающий муравья

Этот автомат описывает поведение муравья, который неплохо справляется с задачей – за 200 ходов съедает 81 яблоко, а за 314 ходов все 89 яблок. Муравей действует по принципу ”Вижу яблоко – иду вперед. Не вижу

– поворачиваюсь. Сделал круг, но яблок нет – иду вперед”.

Приведем два автомата, которые построены в работах [1, 2] генетическими алгоритмами.

В работе [1] приведен автомат, который решает рассматриваемую задачу за 200 ходов и содержит 13 состояний (рис. 4).

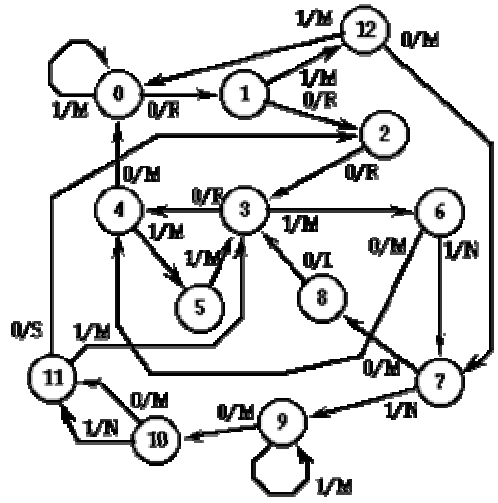

Рис. 4. Конечный автомат с 13 состояниями, задающий поведение муравья, съедающего все яблоки за 200 ходов

На этом рисунке используются следующие обозначения: `N` – непосредственно перед муравьем нет еды, `F` – непосредственно перед муравьем есть еда; `M` – идти вперед, `L` – повернуть налево, `R` – повернуть направо, `NO` – стоять на месте. Можно заметить, что действие `NO` никогда не выполняется.

Этот автомат изображен некорректно – из **одиннадцатого** состояния изображен переход, помеченный входным воздействием 0, с “непонятным” действием `S`. Однако, если это действие заменить на `N`, то автомат корректно решает задачу.

Автомат из работы [2], приведенный на рис. 5, содержит 11 состояний. По утверждению авторов, он съедает все яблоки за 193 хода.

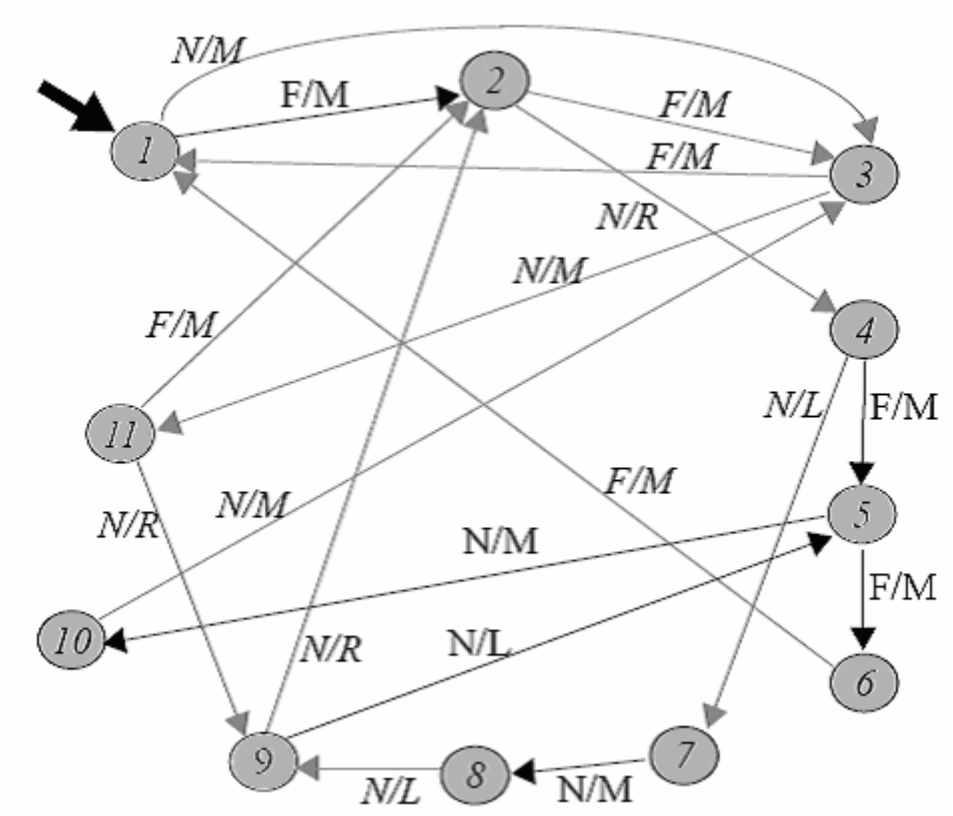

Рис. 5. Конечный автомат с 11 состояниями, задающий поведение муравья, съедающего все яблоки за 193 хода

Этот автомат также изображен некорректно — из **девятого** состояния исходят две дуги, помеченные одинаковым входным воздействием. Однако, если пометку `N/L` на переходе 9–5 заменить на пометку `F/M`, то утверждение авторов становится верным.

2.3. Представление автоматов в виде битовых строк для генерации с помощью генетических алгоритмов

Для построения конечных автоматов, задающего муравья, в работах [1, 2] были применены генетические алгоритмы.

В работе [1] приспособленность особи (*fitness*) определяется как количество яблок, съеденное за 200 ходов. Кодирование автомата, задающего поведение муравья, в особь генетического алгоритма (битовую строку) в этой работе осуществлялось следующим образом: входному воздействию `F` сопоставлялась единица, а `N` – ноль. Каждое из четырех действий муравья

кодировалось двоичными числами: 00 – NO, 01 – L, 10 – R, 11 – M.

Покажем, как выполняется кодирование автомата. На рис. 6 приведен пример графа переходов автомата, описывающего поведение некоторого муравья.

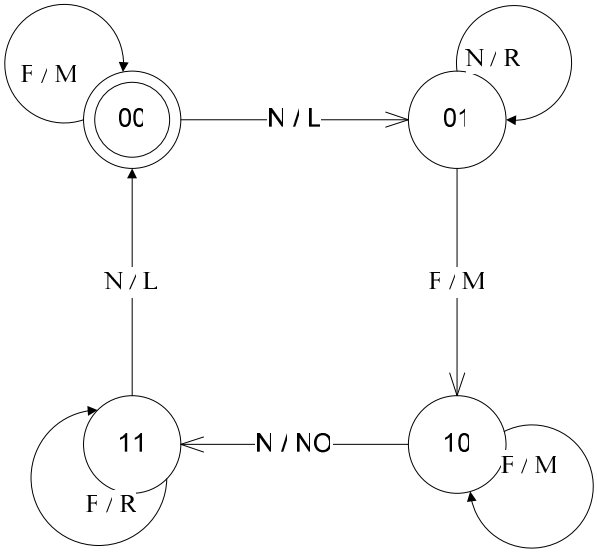

Рис 6. Пример автомата, описывающего поведение муравья

Кодирование этого графа переходов приведено в таблице.

Состояние	Вход	Новое состояние	Действие
00	0	01	01
00	1	00	11
01	0	01	10
01	1	10	11
10	0	11	00
10	1	10	11
11	0	00	01
11	1	11	10

Преобразуем эту таблицу в битовую строку. Для этого сначала требуется запомнить число состояний автомата (в данном случае четыре). В начале строки задан двоичный номер начального состояния автомата (в данном примере 00). Далее записаны пары (Новое состояние, Действие).

Содержимое полей (Состояние, Вход) не записываются, так как они повторяются для каждого автомата с фиксированным числом состояний. Для рассматриваемого примера искомая строка имеет вид:

00 0101 0011 0110 1011 1100 1011 0001 1110

2.4. Идея улучшения работы генетического алгоритма, генерирующего автоматы

Hammerman и *Goldberg* изложили в работе [3] идею, позволяющую улучшить работу генетического алгоритма, особью популяции которого является конечный автомат. Эта идея базируется на следующем наблюдении: существует несколько представлений одного и того же автомата в виде хромосомы (строки фиксированной длины). Например, автоматы на рис. 2.4.1 изоморфны друг другу (обладают одинаковым поведением), но их записи в виде хромосом различаются. Таким образом, путем перенумерация вершин может быть получено $n!$ изоморфных автоматов, кодируемых различными хромосомами.

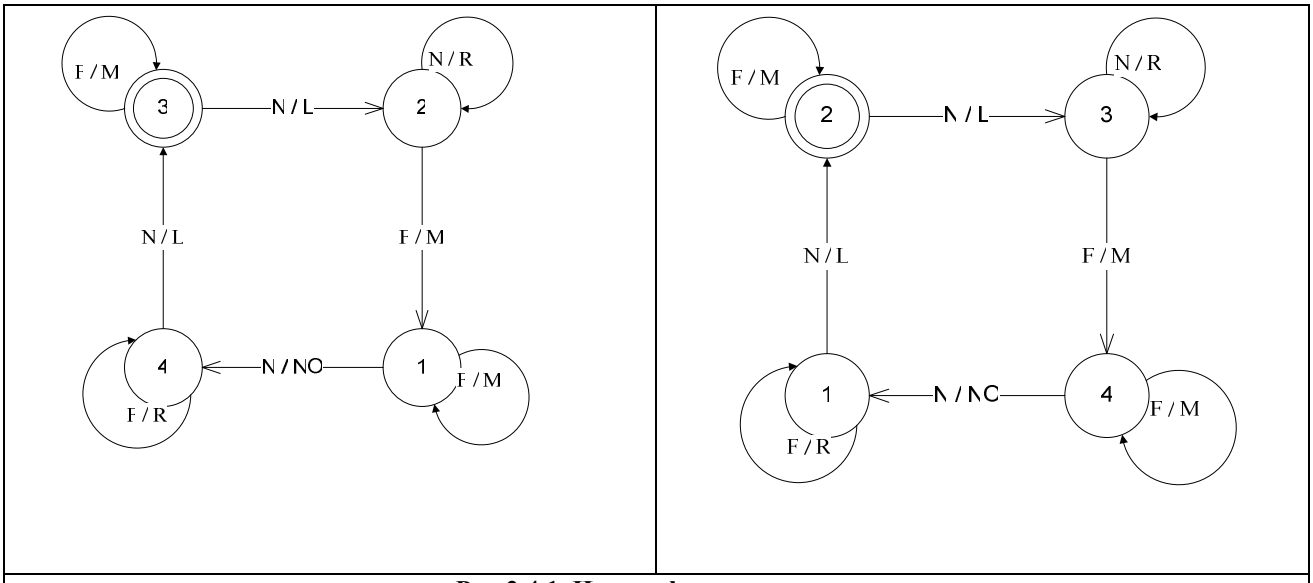

Рис 2.4.1. Изоморфные автоматы

Далее, опираясь на “теорему о схемах” (schemata theorem) [4] (поясняет почему генетические алгоритмы являются эффективным методом решения оптимизационных задач) авторы [3] утверждают, что различные схемы

изоморфных автоматов “соревнуются” друг с другом, замедляя работу генетического алгоритма. Для того, чтобы устранить этот эффект, необходимо приводить автоматы к некоторому каноническому виду (единому для всех изоморфных автоматов), а уже затем кодировать его в виде хромосом.

Для реализации этой идеи авторы [4] предлагают два алгоритма, названные ими *MTF* и *SFS*.

В качестве примера опишем использование алгоритма *MTF*. При его применении на каждом шаге генетического алгоритма выполняется приведение каждого автомата, входящего в популяцию, к каноническому виду. Это выполняется следующим образом. Вершины автомата нумеруются, начиная со стартовой, в соответствии с их порядком при обходе в ширину графа, задающего автомат. На рис 2.4.2 приведен канонический автомат, который изоморфен автоматам, приведенным на рис 2.4.1.

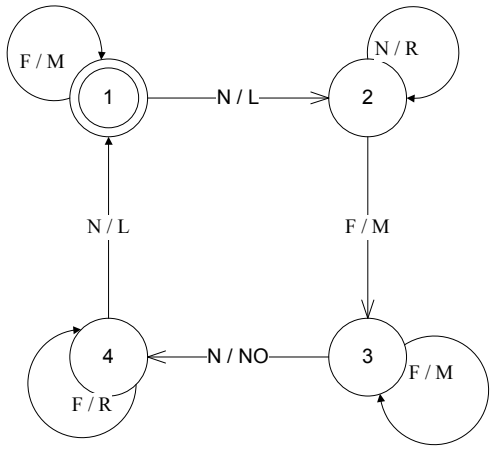

Рис 2.4.2. Канонический автомат

В результате экспериментов, описанных в [3], подтверждается предположение о том, что предложенный метод улучшает (ускоряет) работу генетического алгоритма, особью популяции которого является конечный автомат.

В той же работе приводятся муравьи, полученные в результате работы генетического алгоритма, оптимальные в смыслах: минимизации количества ходов, затрачиваемых на поедание всех яблок; количества состояний автомата, описывающего муравья. На рис. 2.4.3 приведен автомат муравья с 12 состояниями, который съедает все яблоки за 181 ход.

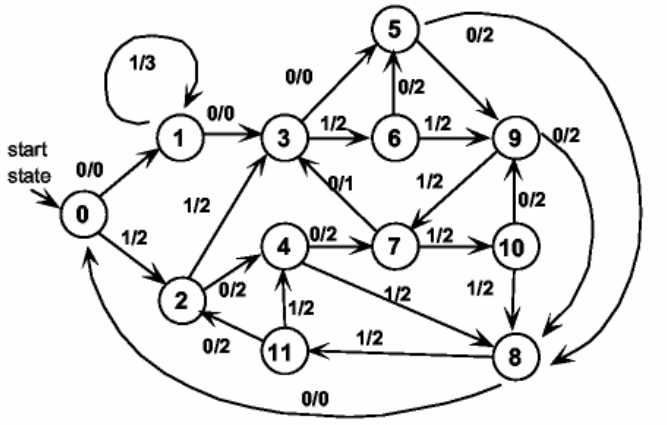

Рис. 2.4.3. Автомат муравья, съедающего все яблоки за 181 ход

Входным воздействиям “Нет еды”, “Есть еда” здесь соответствуют метки на ребрах “0” и “1”, а действиям “Повернуть направо”, “Повернуть налево”, “Идти вперед” – метки “0”, “1”, “2” соответственно.

На рис. 2.4.4 приведен автомат муравья с восемью состояниями, которые решает поставленную задачу.

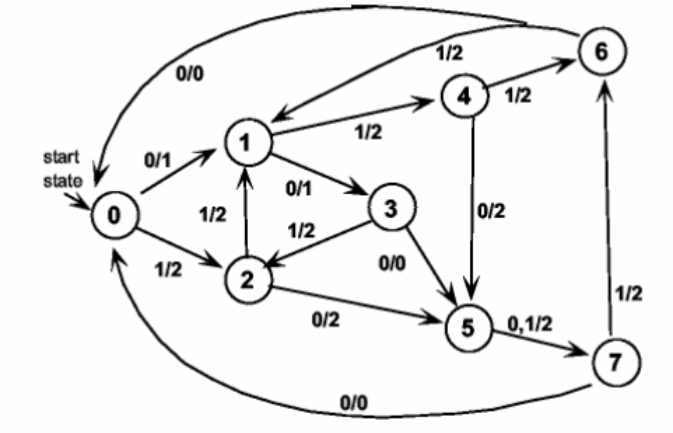

Рис. 2.4.4. Автомат с 8 состояниями, задающий муравья

В настоящее время известен муравей решающий поставленную задачу,

задаваемый автоматом с семью состояниями.

На рис. 2.4.5 изображен граф переходов, построенный генетическим алгоритмом после генерации 160 млн. автоматов. Этот граф переходов описывает поведение автомата с **семью** состояниями, который позволяет муравью съесть всю еду за 190 ходов.

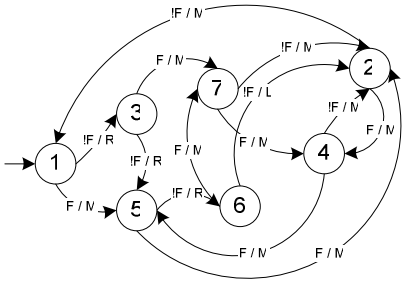

Рис. 2.4.5. Автомат, позволяющий муравью съесть всю еду

Отметим, что при другом запуске этого алгоритма после генерации 250 млн. автоматов был построен еще один автомат с семью состояниями. Эти автоматы были сгенерированы по алгоритму, предложенному Ф. Н. Царевым.

3. Фреймворк для исследований задач о муравье

Для исследований в рассматриваемой области построен фреймворк, обладающий следующими свойствами и функциональностью.

1. Архитектура фреймворка позволяет проводить исследования не только задачи “Умный муравей”, но и ее модификаций. Язык реализации – *Java*, документация – *Javadoc*.
2. Программные интерфейсы фреймворка просты для использования.
3. Фреймворк допускает использование из программ на языке *C++*, реализующих генетические алгоритмы.
4. Фреймворк позволяет визуализировать поведение муравья и задающего его автомата.
5. В фреймворке в качестве первого примера был реализован простой генетический алгоритм для решения задач о муравье (разд. 3.2).

В качестве “макета” фреймворка взят проект “Электрические джунгли” –

http://www.electricjungle.ru. Код фреймворка доступен в Subversion-репозитории по адресу http://neerc.ifmo.ru/svn/automata. Для сборки используется система Maven – http://maven.apache.org.

Основное назначение фреймворка (рис. 7) – предоставить генетическому алгоритму описанный в следующем разделе интерфейс муравья для реализации этого интерфейса, а также возможность вычисления функции приспособленности (*fitness*) для заданных муравья (конкретной реализации) и игры (“Умный муравей”, “Глупый муравей”, “Муравей-2”, т.д.).

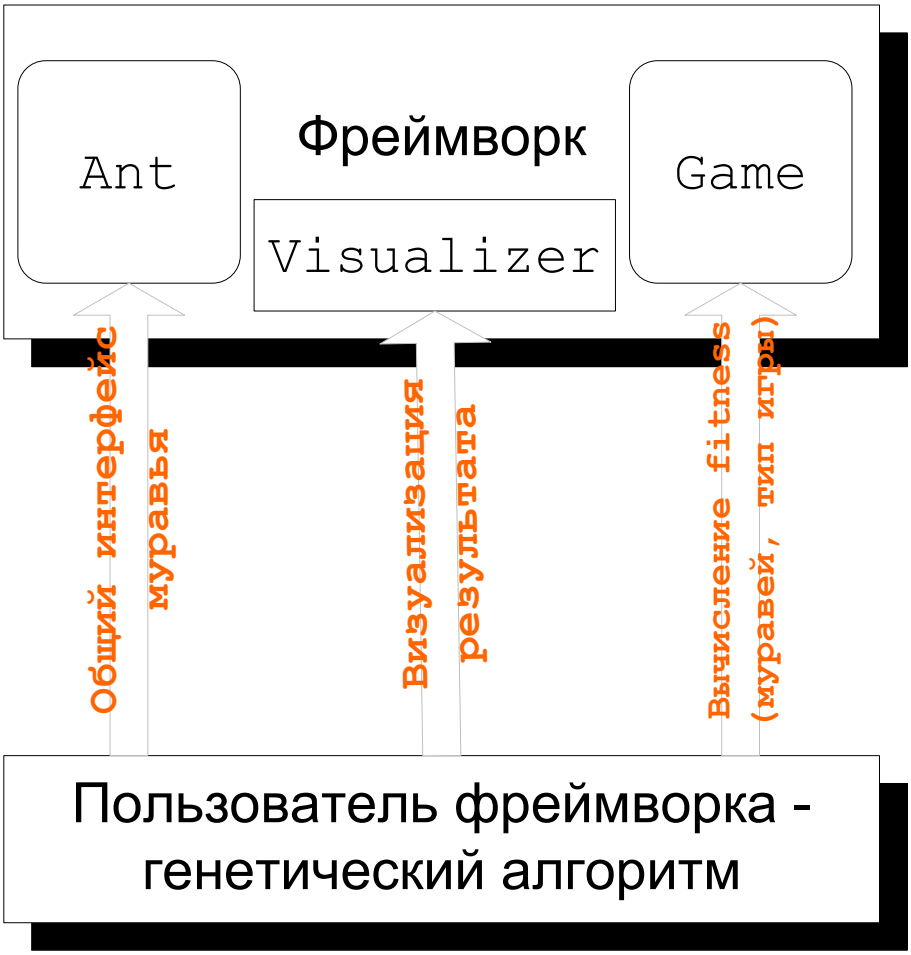

Рис. 7. Фреймворк

3.1. Основные программные интерфейсы

Приведем краткое описание программных интерфейсов, с которыми

работает пользователь фреймворка:

Ant – интерфейс муравья. Реализуя данный интерфейс, пользователь фреймворка задает поведение муравья. Данный интерфейс содержит единственный метод `Action go(Info i)`.

Game – класс, определяющий игру. Этот класс используется генетическим алгоритмом для вычисления функции приспособленности. Передавая в метод `double play(Ant a)` данного класса реализацию интерфейса **Ant**, пользователь получает результат игры для заданных задачи и муравья.

Info – интерфейс для получении информации о наличии еды в обозримых им клетках поля. Для этого используется метод `boolean isFood(Visible v)`.

Visible – перечисление, описывающее клетки, обозримые муравьем.

Action – перечисление, описывающее действие муравья на очередном ходу. Возможны следующие действия: `MOVE, LEFT, RIGHT, NO`.

Information – аннотация, описывающая имя муравья и его разработчика.

Данные интерфейсы и классы приведены на рис. 8. Все они находятся в *package* **ru.ifmo.ctddev.autoant** модуля **autoant-framework**

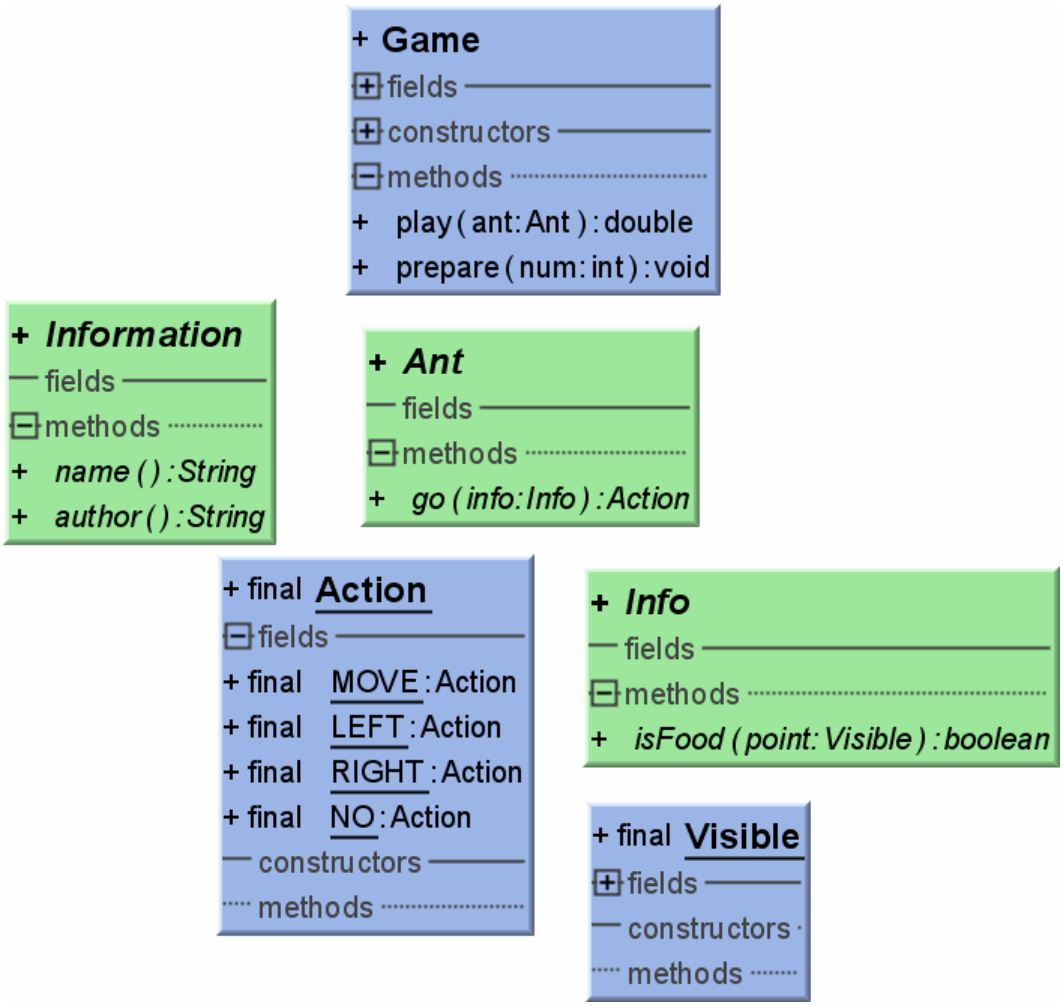

Рис. 8. Диаграмма интерфейсов и классов

3.2. Простой фреймворк для генетических алгоритмов

В рамках фреймворка был создан ряд интерфейсов и классов (*ГА* – фреймворк), которые могут быть использованы для решения довольно широкого класса задач с помощью генетических алгоритмов. Они находятся в *package* **`ru.ifmo.ctddev.autoant.fx`** модуля **`autoant-ga`**.

В качестве теста для проверки фреймворка для генетических алгоритмов была решена задача поиска вектора из всех единиц (**`ru.ifmo.ctddev.autoant.FxTest.testMaxOneProblem`**).

На базе данных классов был создан пример программы (**`ru.ifmo.ctddev.autoant.FixedProblem`** модуля **`autoant-vis`**),

реализующей *ГА* для решения задачи “Умный муравей”, использующий представление автомата в виде битовой строки.

Результаты работы данной программы рассмотрены в главе 4.

3.3. Визуализация полученных автоматов

3.3.1. Визуализация игры

Для лучшего понимая результатов работы генетического алгоритма – эволюционно построенного муравья – был реализован визуализатор игры, который отображает пошаговые действия муравья на поле. Класс **`ru.ifmo.ctddev.autoant.Visualizer`** модуля **`autoant-ga`** использует графический инструментарий *Swing* (http://java.sun.com/docs/books/tutorial/uiswing/index.html) для данных целей. На рис. 9. приведена форма визуализатора. Поддерживается как визуализация муравьев, созданных динамически (через передачу объекта), так и заданных статически.

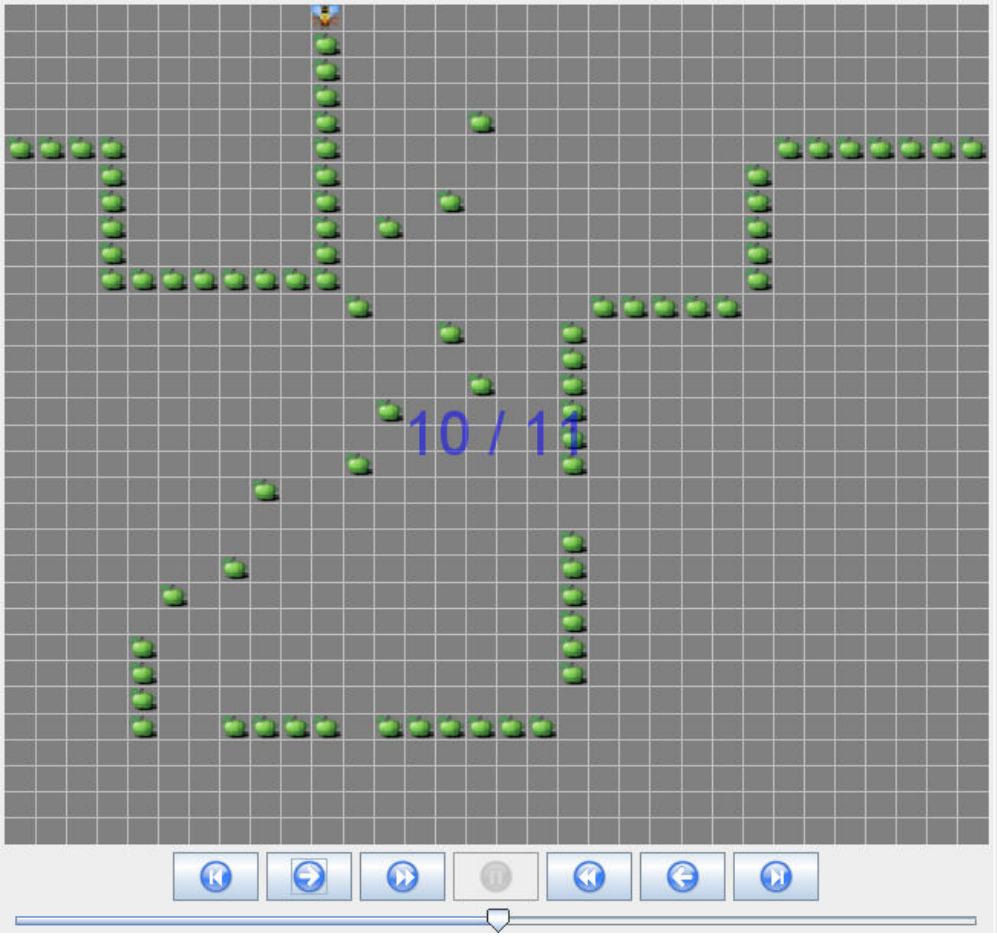

Рис. 9. Визуализация муравья

3.3.2. Визуализация автоматов (*JUNG*)

Описанный выше визуализатор помогает понять работу построенного генетическим алгоритмом автомата, однако этого недостаточно, так как не отображается сам автомат. Для **автоматического отображения** автоматов, генерируемых генетическими алгоритмами, был создан простой визуализатор автоматов на базе библиотеки с открытым кодом *JUNG* – http://jung.sourceforge.net. Пример работы данного визуализатора приведен на рис. 10.

3.4. Взаимодействие с C++ (JNI, JACE)

Несмотря на то, что фреймворк реализован на языке *Java*, существует возможность его использования в программах, реализующих генетические алгоритмы на языке *C++*. Приведем схему взаимодействия частей

программы, написанных на различных языках.

1. Программа на языке *C++* в начале работы вызывает виртуальную *Java* машину.
2. Затем, C++ программа создает объект класса **`Game`**, который полностью аналогичен соответствующему *Java*–классу.
3. Затем, C++ программа передает реализацию класса **`Ant`**, как параметр метода `double Game::play(Ant a)`. Класс **`Ant`** содержит одну чисто-виртуальную функцию `virtual Action go(Info i) = 0;`
4. *Java*–класс **`Game`** оценивает муравья и возвращает результат – вещественное число.

Для организации описанного взаимодействия была использована технология *JNI* (http://java.sun.com/j2se/1.4.2/docs/guide/jni/). Ввиду сложности и неудобства данной технологии, вызванными ее низкоуровневостью, использовалась открытая библиотека *JACE* – http://jace.reyelts.com/jace.

4. Эксперименты и результаты

Для решения задачи “Умный муравей”, описанной выше, был применен генетический алгоритм на битовых строках фиксированный длины, которые кодировали таблицу, задающую автомат муравья. Размер популяции равнялся 5000 хромосом. После 28 минут работы на компьютере с тактовой частотой 1.8GHz был получен автомат, задающий муравья который съедает все яблоки на поле (рис. 10). По времени затрачиваемому на получения муравья, съедающего все яблоки, можно сделать вывод, что реализованный алгоритм оказался эффективнее приведенного в работе [1] и примерно также эффективен, как и генетический алгоритм, описанный в работе [5].

Автомат на рис. 10. имеет 16 состояний.

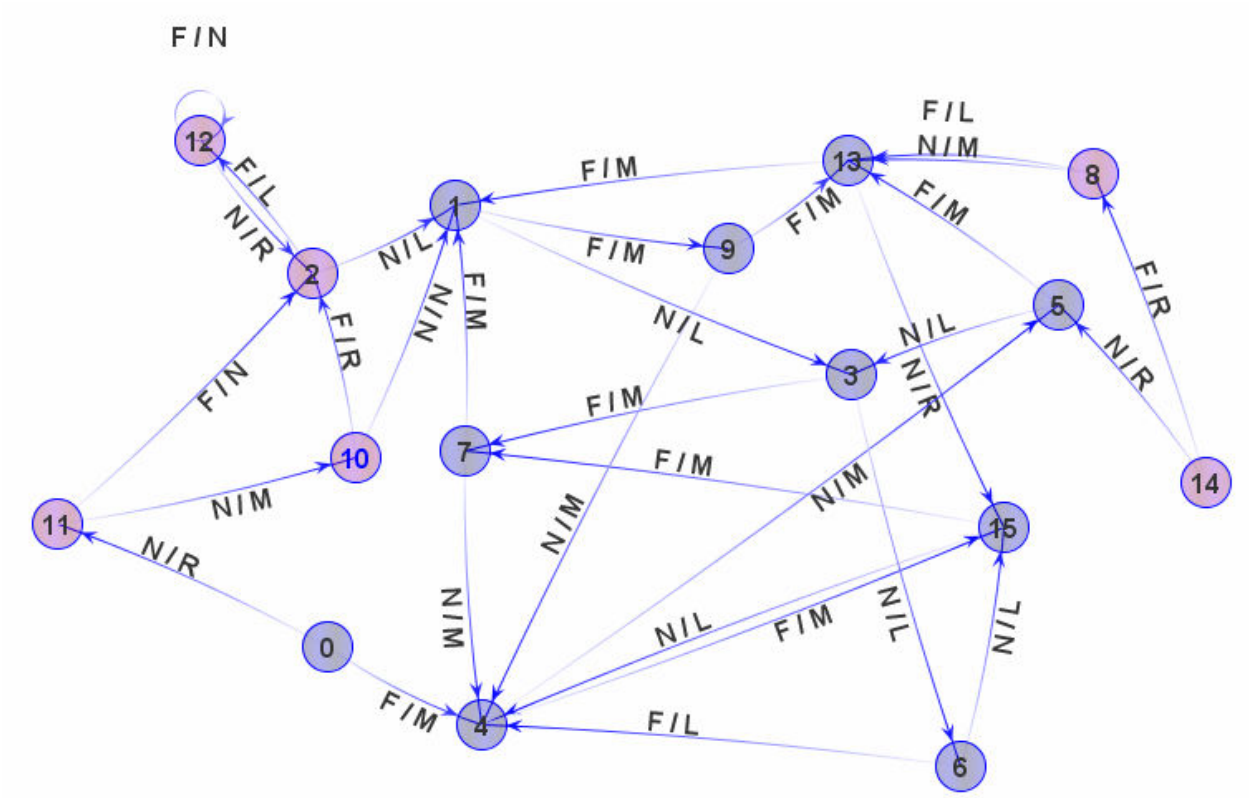

Рис. 10. Автомат с результатом 89 для задачи "Умный муравей"

Стартовое состояние имеет номер ноль. Из состояний этого автомата во время игры используются только 10 – состояния номер 2, 8, 10, 11, 12, 14 не используются. Это можно установить по работе автомата.

Автомат без неиспользуемых состояний приведен на рис. 11.

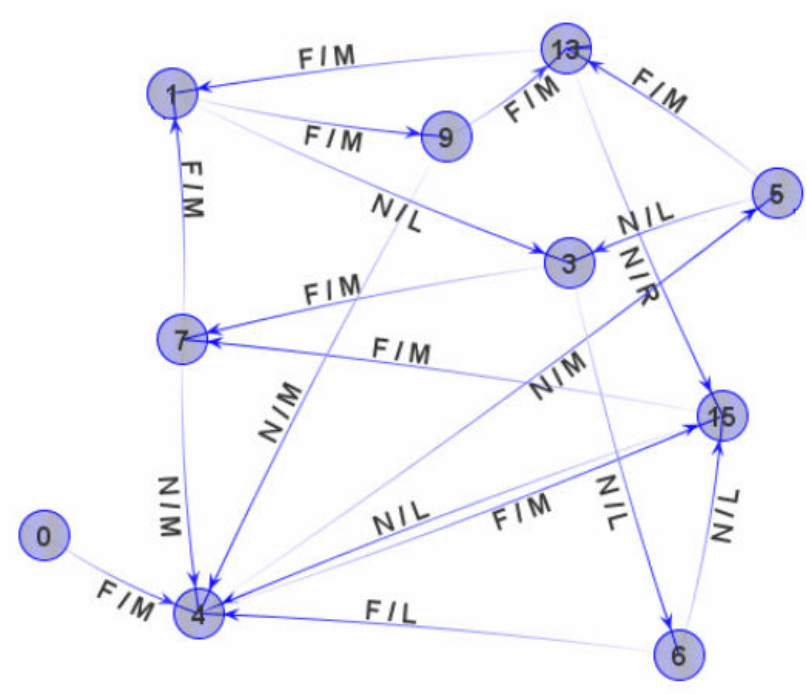

Рис. 11. Автомат с результатом 89 для задачи "Умный муравей" без неиспользуемых состояний

Два вопроса остаются открытыми:

1. Каково минимальное количество состояний автомата, задающего муравья, съедающего все яблоки?
2. Каково минимальное количество ходов, необходимое муравью, съедающему все яблоки?

5. Модификации задачи “Умный муравей”

5.1. Задача “Глупый муравей”

В отличие от предыдущей задачи:

- распределение еды на поле вероятностно – вероятность яблока оказаться в некоторой клетке одинакова для всех клеток поля и равна μ;
- муравей не видит ни одной клетки поля – нет входных воздействий.

Тогда количество еды, съеденной муравьем за 200 ходов, есть случайная величина ξ (определяемая муравьем) на дискретном множестве элементарных исходов Ω – множестве расположений еды – битовых матриц 32 x 32. Каждому исходу ω_i, содержащему k единиц поставим в соответствие

вероятность $p(\omega_i) = \mu^k(1-\mu)^{n-k}$, где n = 32 x 32.

Задача: создать муравья, для которого матожидание количества съеденных яблок Eξ принимает максимальное значение.

Решение: Eξ = μN, где N – количество различных клеток поля, которые посетил муравей за 200 ходов. Из приведенного соотношения следует, что так как μ – константа, необходимо максимизировать N. Это достигается, когда муравей передвигается по траектории “спираль” (рис. 12) Eξ = (200 – количество поворотов) μ= (200 – 200 / 32) μ = 194μ. На рис. 12 желтым цветом показана траектория муравья, который делает 200 ходов и останавливается в клетке, отмеченной оранжевым цветом.

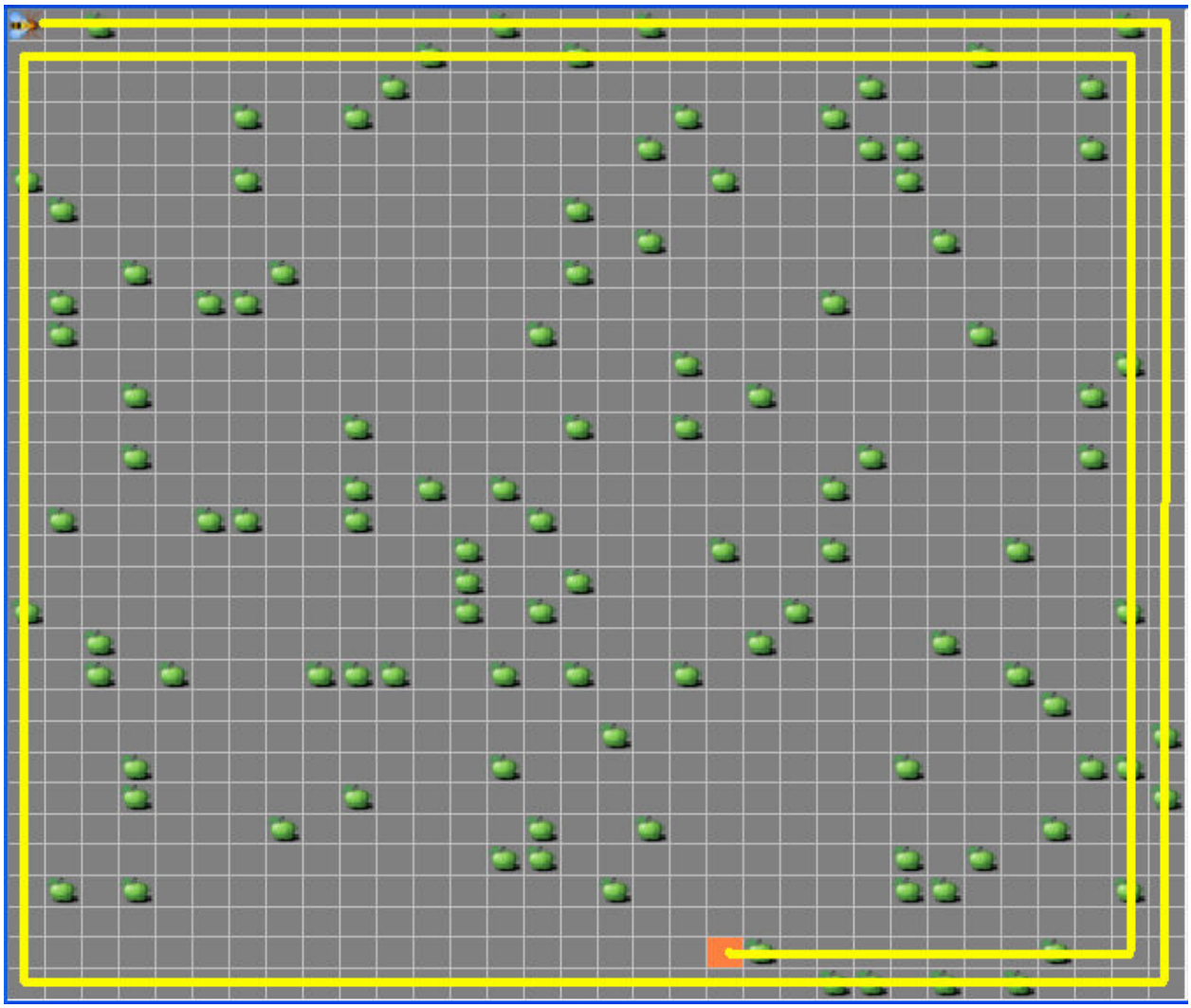

Рис. 12. Траектория “спираль” - лучшее решение задачи "Глупый муравей", μ=0.1

Эксперименты: данная траектория не была получена в результате

работы генетического алгоритма. Это объясняется тем, что муравью требуется слишком много памяти – много состояний, так как другой памяти у него нет. Память на состояниях можно дополнить внешней памятью, например, стековой.

В результате применения генетического алгоритма была получена траектория, количество поворотов в которой при достаточно большом числе состояний лишь немногим превосходит количество поворотов траектории “спираль”. Генетический алгоритм строит умного муравья – он “догадывается”, что поле – тор и существенно это использует.

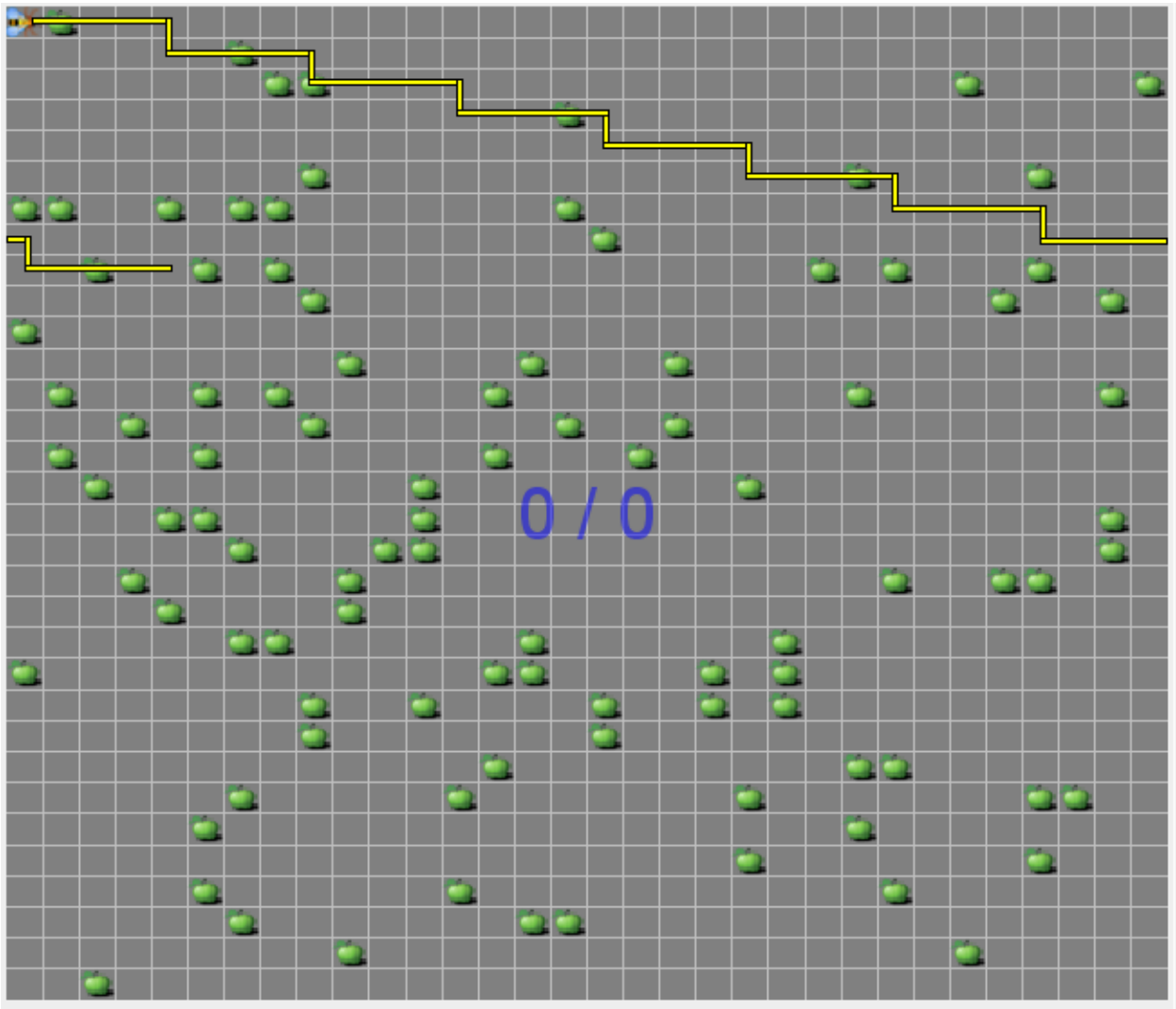

Рис. 12.1. Траектория, полученная ГА для задачи "Глупый муравей", μ=0.1

5.2. Задача “Муравей-2”

В отличие от предыдущей задачи муравей видит одну клетку перед собой, как и в задаче "Умный муравей".

Задача: постановка совпадает с постановкой предыдущей задачи “Глупый муравей”.

Решение: При очень больших μ (почти в каждой клетке есть еда) имеем предельный случай предыдущей задачи. При этом решение не изменяется – траектория “спираль”. Действительно, если в каждой клетке есть еда – гораздо важнее посетить как можно больше различных клеток, нежели исследовать (увидеть).

При малых значениях μ (например, таких, что в среднем на поле всего одна клетка с едой) надо *исследовать* (увидеть) как можно больше клеток, а не посетить их. Траектория “спираль” уже не является лучшим решением, так как перед каждым поворотом, начиная со второго, муравей видит уже ранее исследованную им клетку, находящуюся на “противоположной стороне поля”. Следовательно, надо "не доходить" каждый раз одну клетку до поворота (рис. 13). При этом муравей сделает большее количество поворотов, и поэтому, *посетит* меньшее количество клеток.

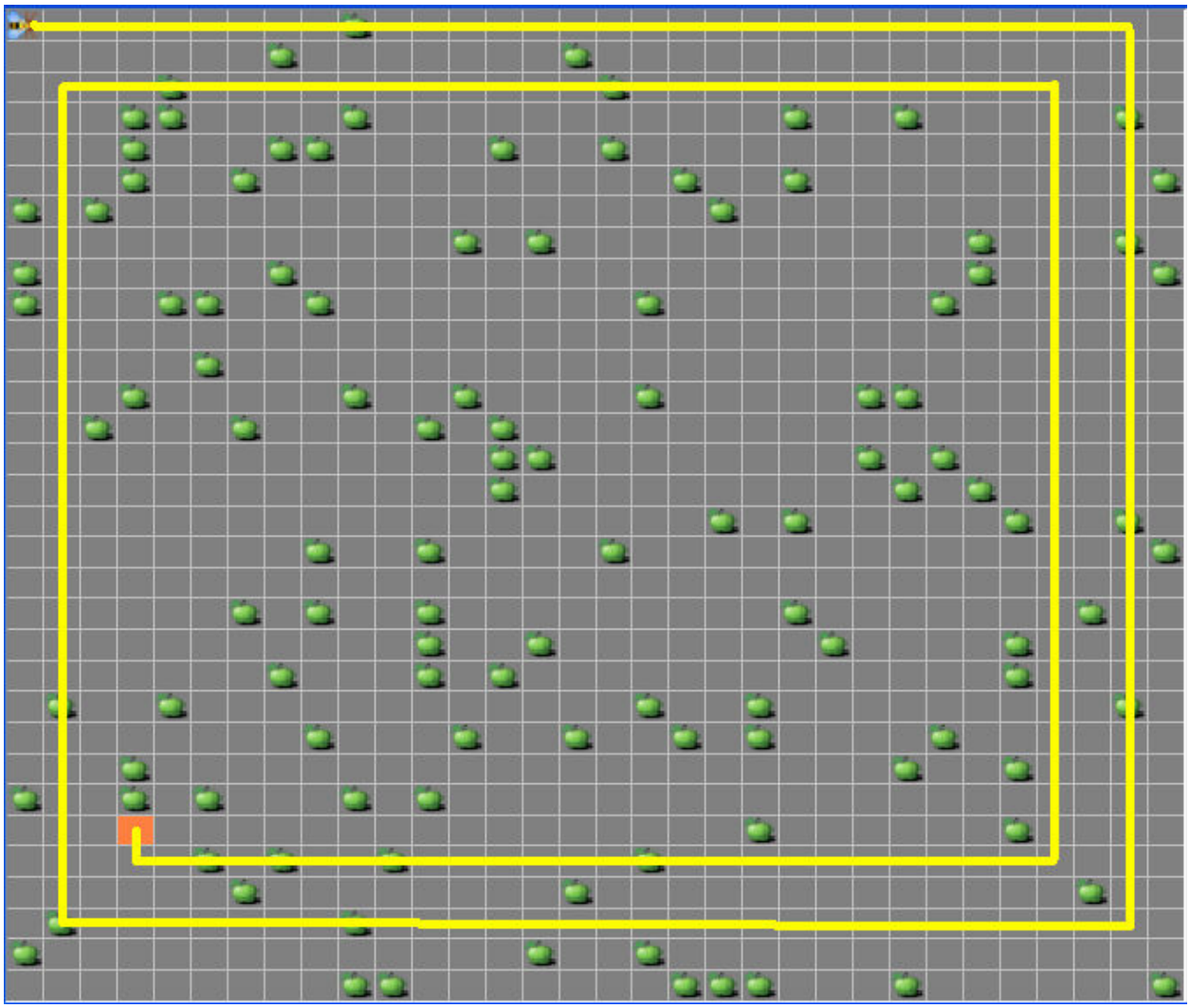

Рис. 13. Наилучшая стратегия муравья в игре “Муравей-2” для малых μ

Открытым остается вопрос: какова граница на μ, на которой надо изменять поведение?

Эксперименты: с использованием фреймворка были выполнены эксперименты при μ = 0.5. При этом автомату максимально разрешалось иметь:

- одно состояние. При этом наилучшей является стратегия “всегда вперед”. Результат при данной стратегии – 16, и он достигается *ГА*.
- два состояния. При этом муравей, построенный *ГА,* дает результат 84.3. Интересно, что этот муравей передвигается по некоторому подобию спирали, имея всего два состояния! Полученный автомат приведен на

рис. 14.

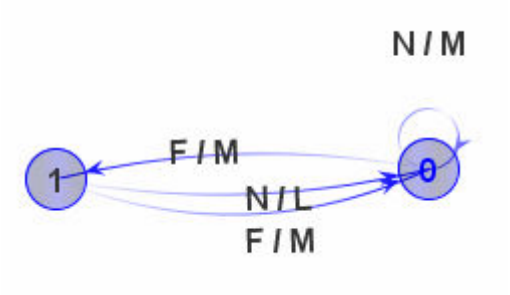

Рис. 14. Лучший муравей с двумя состояниями для задачи "Муравей-2"

- четыре состояния. При этом *ГА* построил муравья с результатом 86.1.

5.3. Задача “Муравей-3”

В отличие от предыдущей задачи расширим обзор муравья – он видит восемь клеток поля (2^8 = 256 входных воздействий). На рис. 15 муравей обозначен красным.

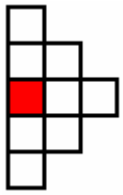

Рис. 15. Область видимости муравья

Задача: если расположение еды на поле будет таким же, как и в задаче “Умный муравей” (фиксированным), то, как показали эксперименты, генетическим алгоритмом быстро выводится автомат всего с одним состоянием, который съедает все 89 яблок за отведенное время (пояснение данного факта приведено ниже).

Поэтому рассмотрим более содержательную задачу, в которой расположение еды задается таким же образом, как и в задаче “Муравей–2”.

Эксперименты: при достаточно больших значениях μ (например, μ = 0.1) наилучшим оказывается автомат лишь с одним состоянием – состояния автомату не требуются. Это объясняется тем, что в области видимости муравья в среднем около 8μ = 0.8 яблок (на самом деле, только перед первым

ходом, а дальше это значение уменьшается, так как муравей постепенно съедает яблоки, но уменьшается незначительно). Следовательно, часто в области видимости муравья находится хотя бы одно яблоко, и он применяет одну из 256 масок поведения **(**маска – одно из четырех действий на входное воздействие**)** для съедания увиденной еды.

Был проведен эксперимент (при $\mu = 0.1$ популяция из 100 особей за 200 итераций при промежуточном вычислении функции на 100 полях и финальном на 1000 полях), который дал следующие результаты:

одно состояние – 46.58;

16 состояний – 40.3.

Исправить ситуацию (заставить ГА строить автоматы с б*о*льшим числом состояний для поедания большего количества яблок) можно уменьшив μ, например до 0.01. Тогда в область обзора муравья попадает в среднем не более $8\mu = 0.08$ яблок (всего на поле оказывается около девяти яблок), и муравей, как правило, на каждом ходу не видит ни одного яблока. В данном случае состояния муравью необходимы для того, чтобы идти по траектории, изображенной на рис 12.1. При параметрах ГА, аналогичных описанным в предыдущем абзаце, были получены следующие результаты:

одно состояние – 2.68;

16 состояний – 5.164.

Таким образом, муравей, задаваемый автоматом с 16 состояниями, съедает почти в два раза больше яблок, чем муравей с одним состоянием. Также заметим, что способность обозревать восемь клеток “очень пригодилась” муравью. Если бы он не видел ни одной клетки (как в задаче “Глупый муравей”), то в среднем результат был бы $200\mu = 2$ – в среднем муравей съедал бы всего два яблок, а не 5.164 (как в рассматриваемом случае).

Однако при анализе поведения построенного муравья, было установлено, что оно довольно примитивно – муравей идет по траектории,

приведенной на рис.12.1 (задача “Глупый муравей”), а, увидев яблоко (яблоки), использует одну из 256 масок поведения для его съедания. Как только яблок в поле зрения больше не остается – муравей вновь идет по траектории того же типа, что и на рис.12.1, иногда изменяя направление траектории с горизонтального на вертикальное и наоборот.

5.4. Модификация задачи с возвращением муравья в начало пути

Была рассмотрена модификация задачи, в которой муравей должен вернуться в начало пути в конце “путешествия”. Функция приспособленности определяется следующим образом. Пусть d – расстояние от точки, в которой муравей закончил игру до начала его пути, а c – количество яблок, съеденное муравьем, тогда: $f = \begin{Bmatrix} -d, & d<0 \\ c, & d=0 \end{Bmatrix}$.

В результате экспериментов не было выведено автомата лучшего, чем тот, который ведет муравья по кругу, не обращая внимания на яблоки – не используя входные воздействия. Таким образом, способность обозревать восемь клеток муравью не пригодилась. Видимо, это объясняется тем, что для эффективного решения поставленной задачи необходима дополнительная память, а наличие памяти только в состояниях не позволяет вывести алгоритм поведения, зависящий от информации получаемой муравьем, обозревающим поле.

5.4. Модификация задачи с генерацией пути в виде ломанной

В фреймворке существует возможность выполнять эксперименты в задаче, в которой расположение еды задается в виде случайной ломанной фиксированной длины, содержащей определенное число яблок. Такой способ генерации поля определен в классе

`ru.ifmo.ctddev.autoant.ant.games.DefaultGame.PolylineField`

5.5. Другие возможные модификация задачи

Вероятностный обзор

Муравей будет получать информацию с некоторой погрешностью. Например, для близлежащих клеток (единичное расстояние до муравья) точно известно (с единичной вероятностью), находится ли там яблоко, но по мере удаления клетки от муравья достоверность информации о наличии в клетке яблока экспоненциально убывает (погрешность оценки соответственно возрастает). Количество входных воздействий при этом не увеличится, но муравью будет необходима память, чтобы переоценить по формуле Байеса вероятность нахождения еды в клетках, увиденных им несколько раз.

6. Способы представления автоматов в виде особей генетического алгоритма

В ходе экспериментов использовались три способа представления автоматов в виде особей ГА:

- в виде битовой строки фиксированной длины;
- в виде графа;
- в виде *Karva*-дерева [6].

Первый способ был подробно описан в разд. 2.3.

Представление автомата в виде графа дает наилучшие результаты. Реализация генетических операций для этого способа представления определена в классе `ru.ifmo.ctddev.autoant.GraphConfiguration`

Представление в виде Karva-дерева

Этот метод основан на программировании с экспрессией генов [6] – разновидности эволюционных алгоритмов, обладающий высокой эффективностью. В отличие как от стандартных генетических алгоритмов, использующих строки фиксированной длины, так и от генетического

программирования, использующего деревья, хромосомы в генетическом программировании с экспрессией генов представляются в виде строк фиксированной длины, которые экспрессируются в *Karva*-деревья.

Предлагаемый метод основан на представления автомата с помощью *Karva*-дерева. Рассмотрим в качестве примера автомат на рис. 16, у которого входные воздействия {N, F}, а действия – {M, L, R}.

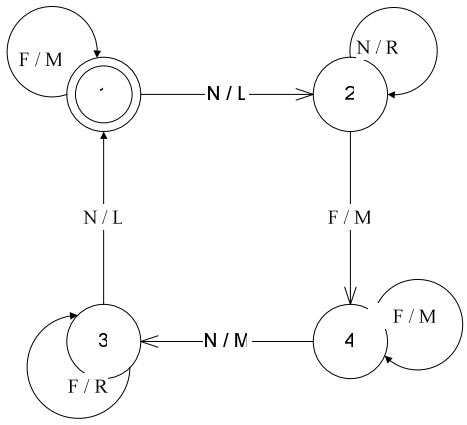

Рис. 16. Пример автомата

На рис.17. этот автомат представлен *Karva*-деревом.

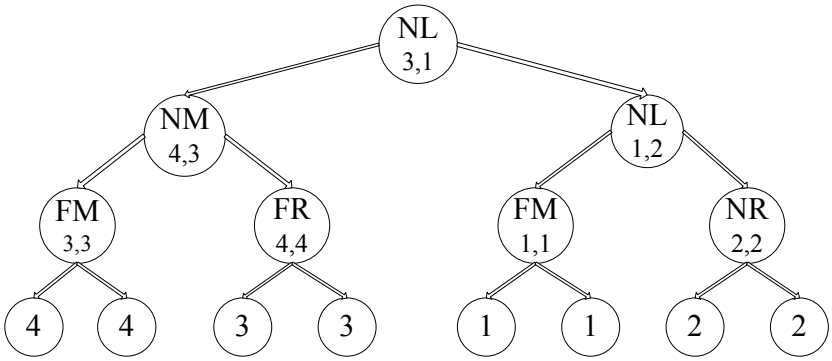

Рис. 17. *Karva*-дерево, задающее автомат

Вершиной этого дерева является либо функция, либо терминал. Множество терминалов – множество состояний автомата {1, 2, …, *n*}. В данном примере – {1, 2, 3, 4}. Множество функций определено следующим образом: каждой паре (входное воздействие v_i, действие z_j) сопоставим функцию $f_{ij}(x, y)$ двух аргументов. В данном примере множество функций – {NL, NR, NM, FL, FR, FM}. Значение функции – автомат, являющийся

объединением автоматов, определяемых аргументами, в котором также задан переход из состояния s в состояние t по входному воздействию v_i с действием z_j. Здесь s – самый правый терминал в левом поддереве, t – самый левый терминал в правом поддереве. На рис. 3 в каждой вершине, являющейся функцией, значения s и t приведены через запятую. Дерево линеаризуется в строку, стандартным для *Karva*-деревьев способом, который изложен в [6]. Строка, соответствующая дереву, изображенному на рис. 3, приведена в таблице.

Таблица

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NL	NM	NL	FM	FR	FM	NR	4	4	3	3	1	1	2	2

При таком представлении удается корректно и просто выполнять генетические операции над строками фиксированной длины, которые экспрессируются в сложные структуры – деревья.

В результате повышение быстродействия обеспечивается за счет применения программирования с экспрессией генов, а проблема большого числа входных воздействий решается за счет того, что из состояния автомата задаются переходы не для всех входных воздействий.

Однако в рассматриваемой задаче о муравье ожидания не оправдались. По-видимому, способ представления автоматов в виде Karva-деревьев, описанный выше, оказался неэффективным.

Литература

1. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. 1992. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
2. *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

3. Chambers *L. D.* Practical Handbook of Genetic Algorithms, Volume 3, Chapter 6 – Algorithms to Improve the Convergence of a Genetic Algorithm with a Finite State Machine Genome. CRC Press, 1999.
4. *Mitchell M.* Introduction to Genetic Algorithms. MA: MIT, 1999.
5. *Guttmann W.* Simulated Evolution of Artificial, Path-Following Ants using a Genetic Algorithm. 2003. http://www.informatik.uni-ulm.de/pm/fileadmin/pm/home/walter/data/ants/
6. *Ferreira C.* Gene Expression Programming: A New Adaptive Algorithm for Solving Problems /Complex Systems. 2001. Vol. 13, issue 2, pp. 87–129. www.gene-expression-programming.com/webpapers/GEP.pdf