Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра «Компьютерные Технологии»

Р. А. Колганов

Отчет по лабораторной работе
«Построение управляющих автоматов с помощью генетических алгоритмов»

Вариант №30

Санкт-Петербург

2010

Оглавление

Введение

В лабораторной работе требуется реализовать генетический алгоритм построения автомата, решающего задачу об «Умном муравье-3». Для этого реализуется плагин для виртуальной лаборатории «GlOpt» («Global Optimization») на языке C#, который реализует конкретный тип генетического алгоритма и конкретное представление особей-автоматов.

1. Постановка задачи

Действие задачи об «Умном муравье-3» (в модификации, рассматриваемой в данной лабораторной работе) происходит на поверхности тора размером 32×32 клетки. В некоторых клетках находится еда. Муравей видит восемь клеток вокруг себя (рис.1). Изначально он находится в фиксированной стартовой клетке.

За ход муравей может выполнить следующие действия:

- повернуть налево;
- повернуть направо;
- сделать шаг вперед и, если в новой клетке есть еда, съесть ее.

Муравей может сделать не более 200 ходов. Задача заключается в создании конечного автомата, управляющего муравьем так, чтобы тот за минимальное число ходов съел как можно больше яблок [1].

Рис. 1. Фрагмент визуализатора к задаче об «умном муравье-3» среды GlOpt, иллюстрирующий поле зрения муравья

2. Алгоритм

2.1. Представление особи

Особью генетического алгоритма для решения данной задачи является конечный автомат Мили – детерминированный конечный автомат, каждому переходу которого сопоставлено некоторое действие (в данной задаче это либо шаг вперед, либо поворот вправо или влево) [1, 2]. Входным алфавитом этого автомата являются значения восьми логических переменных, обозначающих наличие или отсутствие еды в каждой из видимых муравьем клеток.

Автомат хранится как полная таблица переходов из каждого состояния при всех возможных входных воздействиях. В каждой ячейке таблицы записана информация о переходе из соответствующего ячейке состояния при соответствующем ей воздействии: номер состояния, в которое осуществляется переход, и действие, совершаемое при этом муравьем [3].

Число состояний автомата является настраиваемым параметром алгоритма. Все генерируемые алгоритмом автоматы имеют одинаковое число состояний, не изменяющееся во время выполнения алгоритма.

2.2. Тип генетического алгоритма

При выполнении лабораторной работы использован островной генетический алгоритм. Островной генетический алгоритм отличается от обычного тем, что популяция особей разбита на несколько подпопуляций – «островов». Подпопуляции развиваются независимо друг от друга, однако периодически происходят миграции – обмены между островами некоторыми наиболее развитыми особями [2, 4]. Часто при реализации данного алгоритма развитие подпопуляций распараллеливается. В работе распараллеливание не реализовано ввиду особенностей структуры лаборатории, а именно из-за того, что класс, реализующий алгоритм, должен содержать метод, описывающий очередную итерацию алгоритма, то есть требуется последовательное выполнение шагов алгоритма.

На каждом шаге алгоритма выполняются следующие операции [2]:

- отбор некоторых особей в промежуточное поколение;
- формирование нового поколения путем скрещивания (кроссовера) особей из промежуточного поколения;
- мутация некоторых особей нового поколения;
- миграция (периодически);
- большая мутация (в случае, когда в течение многих поколений длится застой, то есть не происходит обновления лучшей за историю популяции особи);
- сортировка популяции по убыванию значений функции приспособленности особей.

Настраиваемыми постоянными параметрами алгоритма являются:

- число островов;
- число особей на острове (одинаковое для всех островов);
- порог усечения – доля особей, переходящих в промежуточное поколение;
- число поколений между миграциями;
- период застоя перед большой мутацией;
- вероятность мутации;
- вероятность мутации перехода или начального состояния особи;
- вероятность миграции;
- вероятность большой мутации.

2.3. Генетические операции

Формирование начального поколения

Все острова заполняются автоматами, таблицы которых заполнены случайными значениями [2].

Метод отбора

В данной работе реализован метод отбора усечением. Метод заключается в следующем. Сначала выбирается порог усечения – число от нуля до единицы, равное доле наиболее приспособленных особей, попадающих в промежуточное поколение. Далее из промежуточного поколения выбираются пары случайных особей, над которыми производится скрещивание. Образованные после скрещивания дочерние особи попадают в следующее поколение. Это происходит до тех пор, пока следующее поколение не будет заполнено, то есть, пока его размер не достигнет определенного значения [5].

Скрещивание (кроссовер)

При кроссовере двух особей образуются две дочерние особи. В каждую ячейку таблицы первой дочерней особи равновероятно записывается значение соответствующей (по номеру состояния и воздействию) ячейки одного из предков, в ячейку второй – соответствующее значение другого. Аналогично, номером начального состояния первой дочерней особи равновероятно назначается номер начального состояния одного из предков, номером начального состояния второй – номер начального состояния другого предка [1].

Мутация

Каждая особь нового поколения с некоторой небольшой вероятностью может мутировать. Мутация особей заключается в изменении выходного состояния и действия на каждом переходе автомата с некоторой небольшой вероятностью на случайные, а также в изменении с той же вероятностью начального состояния на произвольно определенное [1, 2].

Миграция

Раз в некоторое фиксированное число поколений с каждого острова происходит миграция. Каждая попадающая под порог усечения особь острова с некоторой вероятностью мигрирует на случайный остров, при этом ее место занимает случайная попадающая под порог усечения особь с этого острова [2].

Большая мутация

В случае, когда в течение определенного числа поколений лучшая особь за историю популяции не меняется, происходит большая мутация. Она заключается в замене каждой особи с некоторой небольшой вероятностью на случайно сгенерированную.

Оценка приспособленности

Функцией приспособленности особей в данной задаче является разность числа яблок, съеденных муравьем, управляемым автоматом-особью, и отношения числа

сделанных им шагов к максимально возможному числу шагов (а именно в данном случае к двумстам). Наиболее приспособленной считается особь с максимальным значением функции приспособленности [1].

3. Результаты работы алгоритма

Лучшая особь была выведена в шестом поколении популяции за четыре секунды выполнения алгоритма при параметрах, указанных в таблице.

Таблица. Параметры генетического алгоритма.

Число островов	3
Число особей на острове	75
Порог усечения	0.2
Вероятность мутации	0.1
Вероятность мутации перехода особи	0.1
Вероятность большой мутации	0.1
Вероятность миграции	0.1
Период застоя перед большой мутацией	50
Число поколений между миграциями	100

Лучшая особь является автоматом с одним состоянием. Управляемый им муравей съедает все 89 яблок за 146 шагов.

На основе экспериментов можно сделать вывод, что в данной задаче при данной реализации генетических операторов и метода отбора островной подход бесполезен, достаточно классического, поскольку оптимальный результат достигается через очень малое число поколений.

Далее следует описание лучшей особи. Поскольку у нее одно состояние, то все переходы направлены из него в него же, и имеет значение только совершаемое при переходе действие. Условия действий записаны в форме логических выражений от восьми переменных x_i, обозначающих наличие еды в i-ой клетке из области видимости муравья. Знаки конъюнкции в выражениях опущены, дизъюнкция обозначена как ‘+’, отрицание над несколькими переменными подряд следует понимать как конъюнкцию их отрицаний.

Нумерация клеток области видимости муравья (муравей обозначен символом ‘^’) приведена на рис. 2.

		0		
	1	2	3	
4	5	^	6	7

Рис. 2. Нумерация клеток области видимости муравья.

Условие движения вперед:

$$(x_0 + x_1)\overline{x_2 x_3 x_4 x_5 x_6} + (x_0 + x_2)\overline{x_1 x_3 x_5 x_6 x_7} + (x_0 + x_3)\overline{x_1 x_2 x_4 x_7} + x_0\overline{x_1 x_2 x_3 x_4 x_5 x_6 x_7} +$$
$$+ x_0\overline{x_1 x_2 x_3 x_4 x_5}\,\overline{x_6 x_7} + x_0\overline{x_1 x_2 x_3 x_4 x_5 x_6}\,\overline{x_7} + \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5 x_6 x_7} + \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5 x_6}\,\overline{x_7} +$$
$$+ \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5 x_6 x_7} + \overline{x_0 x_1 x_2 x_3 x_4}\,\overline{x_5 x_6 x_7} + x_0\overline{x_1 x_2 x_3 x_4 x_5}\,\overline{x_6 x_7} + \overline{x_0} x_1 x_2\overline{x_3 x_4 x_5 x_6 x_7} +$$
$$+ \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5 x_6}\,\overline{x_7} + x_0\overline{x_1 x_2 x_3 x_4 x_5}\,\overline{x_6 x_7} + \overline{x_0} x_1 x_2 x_3\overline{x_4}(x_5 + x_6)\overline{x_7} +$$
$$+ \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5 x_6 x_7} + \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5}\,\overline{x_6 x_7} + x_2 x_3 x_4 + \overline{x_0} x_1 x_2 x_3 x_5 + \overline{x_0} x_2 x_3 x_7 + \overline{x_0} x_2 x_4 x_5 +$$
$$+ x_3 x_4\overline{x_5} x_6 + x_0(\overline{x_4} x_5 x_6 + x_4\overline{x_5} x_6 + x_4 x_5\overline{x_6})x_7 + x_1 x_2 x_3 x_7 + \overline{x_0} x_4 x_5 x_6 x_7$$

Условие поворота вправо:

$$\overline{x_0 x_1 x_2 x_3 x_4}\,\overline{x_5 x_6 x_7} + \overline{x_0 x_1 x_2 x_3 x_4 x_5 x_6}\,\overline{x_7} + x_0\overline{x_1 x_2 x_3 x_4 x_5 x_6 x_7} + \overline{x_0} x_1\overline{x_2} x_3\overline{x_4 x_5 x_6 x_7} +$$
$$+ \overline{x_0} x_1\overline{x_2 x_3 x_4 x_5 x_6} x_7 + \overline{x_0 x_1} x_2\overline{x_3 x_4 x_5 x_6 x_7} + \overline{x_0 x_1} x_2\overline{x_3 x_4 x_5 x_6}\,\overline{x_7} + \overline{x_0 x_1 x_2}\,\overline{x_3 x_4 x_5 x_6} x_7 +$$
$$+ \overline{x_0 x_1 x_2} x_3\overline{x_4 x_5}\,\overline{x_6 x_7} + \overline{x_0 x_1 x_2} x_3\overline{x_4 x_5 x_6}\,\overline{x_7} + \overline{x_0 x_1 x_2 x_3}\,\overline{x_4 x_5 x_6 x_7} + \overline{x_0 x_1 x_2 x_3 x_4}(x_5 x_6 +$$
$$+ x_5\overline{x_6} x_7 + \overline{x_5} x_6 x_7) + x_0 x_1 x_2 + x_0 x_2 x_6 + x_0(x_2\overline{x_3} + \overline{x_2} x_3)x_7 + x_1 x_2\overline{x_3} x_4 + x_1 x_2 x_5 +$$
$$+ x_1\overline{x_2} x_3 x_4 + x_1\overline{x_5} x_6 x_7 + x_2 x_3 x_6 + x_4 x_5 x_6\overline{x_7} + x_0 x_2(x_3 + x_4)x_5$$

Если оба этих условия не выполнены, муравей поворачивает влево.

Заключение

Результаты исследований, проведенных в данной лабораторной работе, показали, что генетические алгоритмы достаточно эффективны при построении автомата, решающего задачу об «Умном муравье-3». Муравей, управляемый автоматом, сгенерированным данным алгоритмом, съедает все 89 яблок за 146 шагов. При этом полученный автомат имеет всего одно состояние.

Литература

1. Инструкция по работе с виртуальной лабораторией GlOpt.
   http://is.ifmo.ru/genalg/labs_2010-2011/GlOpt_problems.doc
   http://is.ifmo.ru/genalg/labs_2010-2011/GlOpt_instruction.doc
2. А. А. Давыдов, Д. О. Соколов, Ф. Н. Царев. Применение генетических алгоритмов для построения автоматов мура и систем взаимодействующих автоматов мили на примере задачи об «умном муравье».
   http://is.ifmo.ru/works/_2009_08_12_davydov.pdf
3. Ф. Н. Царев, А. А. Шалыто. Разработка технологии генетического программирования для генерации автоматов управления системами со сложным поведением.
   http://is.ifmo.ru/present/_genetic-itmo.ppt
4. Яминов Б. Генетические алгоритмы.
   http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005
5. И. Л. Каширина. Введение в эволюционное моделирование.
   http://window.edu.ru/window_catalog/pdf2txt?p_id=29587

Приложение. Исходный код

1. Файл Automation.cs

Содержит класс, описывающий особь данного генетического алгоритма – автомат Мили.

```
using ArtificialAnt3.ProblemInfo;
using NewBrain;

namespace ArtificialAnt3.IndividualInfo
{
    /// <summary>
    /// Автомат к задаче об "умном муравье-3"
    /// </summary>
    public class Automation : Individual
    {
        /// <summary>
        /// Матрица состояний автомата
        /// </summary>
        private readonly Transition[,] _transitions;

        /// <summary>
        /// Число входных условий
        /// </summary>
        private static int ConditionCount = 8;

        /// <summary>
        /// Размер алфавита автомата
        /// </summary>
        public static int AlphabetSize
        {
            get { return 1 << ConditionCount; }
            private set { }
        }

        /// <summary>
        /// Новый автомат
        /// </summary>
        /// <param name="initialState">Начальное состояние</param>
        /// <param name="statesCount">Число состояний</param>
        public Automation(int initialState, int statesCount)
        {
            ConditionCount = 8;
            //Размер алвафита равен 2^<число входных условий>
            AlphabetSize = 1 << ConditionCount;

            InitialState = initialState;
            StatesCount = statesCount;
            _transitions = new Transition[statesCount, AlphabetSize];
        }

        /// <summary>
        /// Начальное состояние автомата
        /// </summary>
        public int InitialState { get; set; }

        /// <summary>
        /// Число состояний автомата
        /// </summary>
        public int StatesCount { get; protected set; }

        public Transition GetTransition(int index, int condition)
```

```
{
    return _transitions[index, condition];
}

public void SetTransition(int index, int condition, Transition transition)
{
    _transitions[index, condition] = transition;
}

public override string ToString()
{
    string res = "Initial state: " + InitialState.ToString() + "\r\n   ";
    for (int i = 0; i < StatesCount; i++)
    {
        res += string.Format("{0,4}", i);
    }
    res += "\r\n";
    for (int i = 0; i < AlphabetSize; i++)
    {
        res += string.Format("{0,3}", i);
        for (int j = 0; j < StatesCount; j++)
        {
            res += string.Format("{0,3}", _transitions[j, i].EndState);
            switch (_transitions[j, i].Action)
            {
                case SimpleAntProblem.AntActions.Left:
                    res += "L";
                    break;
                case SimpleAntProblem.AntActions.Right:
                    res += "R";
                    break;
                case SimpleAntProblem.AntActions.Move:
                    res += "M";
                    break;
                default:
                    res += "N";
                    break;
            }
        }
        res += "\r\n";
    }
    return res;
}

/// <summary>
/// Переход, состоящий из действия и конечного состояния
/// по выполнению этого действия.
/// </summary>
public class Transition
{
    public Transition(int endState, SimpleAntProblem.AntActions action)
    {
        EndState = endState;
        Action = action;
    }

    /// <summary>
    /// Конечное состояние
    /// </summary>
    public int EndState { get; private set; }

    /// <summary>
    /// Действие
```

```
            /// </summary>
            public SimpleAntProblem.AntActions Action { get; protected set; }

            public override string ToString()
            {
                return string.Format("-{0}->{1}", Action, EndState);
            }
        }
    }
}
```

2. Файл SimpleAntProblem.cs

Содержит класс, описывающий задачу, в частности, содержащий метод вычисления функции приспособленности особи.

```
using System;
using ArtificialAnt3._Internal.Mover;
using ArtificialAnt3._Internal.Phenotype;
using ArtificialAnt3.IndividualInfo;
using ArtificialAnt3.Properties;
using NewBrain;
using NewBrain.ComponentModel;

namespace ArtificialAnt3.ProblemInfo
{
    [IndividualType(typeof (Automation))]
    [IndividualViewer(typeof(AntViewer))]
    public class SimpleAntProblem : Problem
    {
        #region AntActions enum
        /// <summary>
        /// Возможные типы действий муравья
        /// </summary>
        public enum AntActions : byte
        {
            Left = 0,
            Right,
            Move
        }

        #endregion

        /// <summary>
        /// Направление оптимизации - максимизация функции
        /// </summary>
        public override OptimizationDirection OptimizationDirection
        {
            get { return OptimizationDirection.Maximize; }
        }

        public override string HtmlDescription
        {
            get
            {
                return Resources.artAnt;
            }
        }

        public override string Title
        {
            get { return Properties.Resources.ArtAnt3Problem; }
```

```
        }

        public override string Description
        {
            get { return ""; }
        }

        /// <summary>
        /// Вычисление fitness-функции текущей особи - числа съеденных яблок
        /// на торе за ограниченное число ходов.
        /// </summary>
        /// <param name="individual">Особь для подсчета значения
        /// fitness-функции</param>
        /// <returns>Значение fitness-функции</returns>
        public override double EvaluateIndividual(Individual individual)
        {
            var a = individual as Automation;
            if (a == null)
                return double.NaN;

            var mover = new SimpleMover(a);
            mover.Reset();
            int apples = 0;
            int lastStep = 0;
            for (int i = 0; i < Ant.MaxStepsCount; ++i)
            {
                if (mover.Move())
                {
                    apples++;
                    lastStep = i;
                }
                if (apples == Ant.FoodCount)
                    break;
            }
            return apples - 1.0*lastStep/Ant.MaxStepsCount;
        }
    }

    internal class AntViewer : IIndividualViewer
    {
        /// <summary>
        /// Визуализатор особи - муравья
        /// </summary>
        /// <param name="individual">Особь для визуализации</param>
        public void ViewIndividual(Individual individual)
        {
            Automation a = individual as Automation;
            if (a == null)
                throw new ArgumentException();

            using (var form = new AntViewerForm())
            {
                form.fieldControl.Mover = new SimpleMover(a);
                form.ShowDialog();
            }
        }
    }
}
```

3. Файл GeneticSearchOperatorOnFullTableAutomation.cs

Содержит методы, осуществляющие генетические операции над особями на основе автоматов Мили, представленных в виде полных таблиц переходов: создание особи, мутацию и скрещивание.

```
using System;
using System.ComponentModel;
using ArtificialAnt3.IndividualInfo;
using ArtificialAnt3.ProblemInfo;
using Genetic.Operators;
using NewBrain;
using NewBrain.Attributes;

namespace Genetic
{
    [IndividualType(typeof(Automation))]
    public abstract class SearchOperatorOnFullTableAutomation : SearchOperator
    {
        public SearchOperatorOnFullTableAutomation()
        {
            StatesCount = 8;
            LocalMutationProbability = 0.1;
        }

        [Configurable]
        [Description("Число состояний")]
        public int StatesCount { get; set; }

        [Configurable]
        [Description("Вероятность мутации перехода")]
        public double LocalMutationProbability { get; set; }

        public override Individual Create(Random random)
        {
            Automation automation = new Automation(0, StatesCount);
            for (int i = 0; i < StatesCount; i++)
                for (int j = 0; j < Automation.AlphabetSize; j++)
                    automation.SetTransition(i, j,
                        new Automation.Transition(random.Next(StatesCount),
                            GetRandomAction(random)));

            return automation;
        }

        protected SimpleAntProblem.AntActions GetRandomAction(Random random)
        {
            SimpleAntProblem.AntActions action = SimpleAntProblem.AntActions.Left;
            switch (random.Next(3))
            {
                case 0:
                    action = SimpleAntProblem.AntActions.Left;
                    break;
                case 1:
                    action = SimpleAntProblem.AntActions.Move;
                    break;
                case 2:
                    action = SimpleAntProblem.AntActions.Right;
                    break;
```

```
        }
        return action;
    }
}

[IndividualType(typeof (Automation))]
public class GeneticSearchOperatorOnFullTableAutomation :
    SearchOperatorOnFullTableAutomation, IGeneticSearchOperator
{
    public override string Title
    {
        get { return Properties.Resources.GeneticOperOnAuto; }
    }

    public override string Description
    {
        get { return Properties.Resources.GeneticOperOnAuto; }
    }

    public Individual Mutate(Individual oldIndividual, Random random)
    {
        Automation auto = oldIndividual as Automation;
        if (auto == null)
            return oldIndividual;

        int newInitialState = auto.InitialState;
        if (random.NextDouble() > 1 - LocalMutationProbability)
        {
            newInitialState = random.Next(auto.StatesCount);
        }

        Automation res = new Automation(newInitialState, auto.StatesCount);
        for (int i = 0; i < res.StatesCount; i++)
        {
            for (int j = 0; j < Automation.AlphabetSize; j++)
            {
                if (random.NextDouble() > 1 - LocalMutationProbability)
                {
                    res.SetTransition(i, j,
                        new Automation.Transition(random.Next(auto.StatesCount),
                            GetRandomAction(random)));
                }
                else
                {
                    Automation.Transition t = auto.GetTransition(i, j);
                    res.SetTransition(i, j,
                        new Automation.Transition(t.EndState, t.Action));
                }
            }
        }

        return res;
    }

    public Individual[] Crossover(Individual[] individuals, Random random)
    {
        Automation aa1 = individuals[0] as Automation;
        Automation aa2 = individuals[1] as Automation;

        if (aa1 == null || aa2 == null)
            return individuals;

        Automation[] s = new Automation[2];
```

```
            for (int i = 0; i < 2; i++)
                s[i] = new Automation(0, aa1.StatesCount);

            if (random.Next(2) == 1)
            {
                s[0].InitialState = aa2.InitialState;
                s[1].InitialState = aa1.InitialState;
            }
            else
            {
                s[0].InitialState = aa1.InitialState;
                s[1].InitialState = aa2.InitialState;
            }

            for (int i = 0; i < aa1.StatesCount; i++)
            {
                for (int j = 0; j < Automation.AlphabetSize; j++)
                {
                    if (random.Next(2) == 1)
                    {
                        s[0].SetTransition(i, j, aa1.GetTransition(i, j));
                        s[1].SetTransition(i, j, aa2.GetTransition(i, j));
                    }
                    else
                    {
                        s[0].SetTransition(i, j, aa2.GetTransition(i, j));
                        s[1].SetTransition(i, j, aa1.GetTransition(i, j));
                    }
                }
            }

            return s;
        }
    }
}
```

4. Файл IslandGeneticOptimizationAlgorithm.cs

Содержит класс, реализующий островной генетический алгоритм, использующий метод отбора усечением.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using Genetic.Operators;
using NewBrain;
using NewBrain.Attributes;
using NewBrain.Plugins;
using IslandGenetic.Properties;

namespace IslandGenetic
{
    [SearchOperatorType(typeof(IGeneticSearchOperator))]
    /// <summary>
    /// Островной генетический алгоритм, использующий метод отбора усечением
    /// </summary>
    public class IslandGeneticOptimizationAlgorithm : Algorithm
    {
        [Configurable]
        [Description("Чило особей на одном острове")]
```

```
public int IslandGenerationSize { get; set;}

[Configurable]
[Description("Число островов")]
public int IslandNumber { get; set; }

[Configurable]
[Description("Число поколений между большими мутациями")]
public int BigMutationPeriod { get; set; }

[Configurable]
[Description("Число поколений между миграциями")]
public int MigrationPeriod { get; set; }

[Configurable]
[Description("Порог усечения")]
public double TruncationThreshold { get; set; }

[Configurable]
[Description("Вероятность мутации")]
public double MutationProbability { get; set; }

[Configurable]
[Description("Вероятность миграции")]
public double MigrationProbability { get; set; }

[Configurable]
[Description("Вероятность большой мутации")]
public double BigMutationProbability { get; set; }

public override string Title
{
    get { return Properties.Resources.IslandGenetic; }
}

public override string Description
{
    get { return ""; }
}

public override string HtmlDescription
{
    get
    {
        return Resources.island;
    }
}

/// <summary>
/// Направление оптимизации - максимизировать функцию.
/// </summary>
protected override OptimizationDirection OptimizationDirection
{
    get { return OptimizationDirection.Maximize; }
}

/// <summary>
/// Генетический алгоритм оптимизации
/// </summary>
public IslandGeneticOptimizationAlgorythm()
{
    IslandGenerationSize = 75;
    IslandNumber = 3;
```

```
        BigMutationPeriod = 50;
        MigrationPeriod = 100;
        TruncationThreshold = 0.2;
        MutationProbability = 0.1;
        MigrationProbability = 0.1;
        BigMutationProbability = 0.1;
    }

    protected Random Random { get; private set; }

    /// <summary>
    /// Текущее "поколение" особей на островах для работы алгоритма
    /// </summary>
    protected List<Individual>[] Islands { get; set; }

    protected IGeneticSearchOperator GeneticSearchOperator
    {
        get { return SearchOperator as IGeneticSearchOperator; }
    }

    /// <summary>
    /// Порядковый номер текущего поколения
    /// </summary>
    public int GenerationNumber { get; private set; }

    /// <summary>
    /// Число поколений, среди которых не было улучшения лучшего индивида
    /// </summary>
    private int NoProgressTime;

    /// <summary>
    /// Инициализация начального состояния генетического алгоритма.
    /// Создается первое поколение особей, полученное случайным образом.
    /// </summary>
    protected override void Initialize()
    {
        Random = new Random();
        Islands = new List<Individual>[IslandNumber];
        for (int i = 0; i < IslandNumber; i++)
        {
            Islands[i] = new List<Individual>();
            for (int j = 0; j < IslandGenerationSize; j++)
                Islands[i].Add(SearchOperator.Create(Random));
        }
    }

    /// <summary>
    /// Очередная итерация генетического алгоритма.
    /// Происходит переход к следующему поколению особей.
    /// Осуществляются мутации и миграции между островами.
    /// </summary>
    protected override void NextIteration()
    {
        GenerationNumber++;
        NoProgressTime++;
        NextGeneration();
        if (NoProgressTime >= BigMutationPeriod)
        {
            BigMutation();
            NoProgressTime = 0;
        }
        if (GenerationNumber >= MigrationPeriod)
        {
```

```
            Migration();
            GenerationNumber = 0;
        }
    }

    /// <summary>
    /// Генерация нового поколения заключается в выборе случайным образом
    /// двух особей из старого поколения и получения на их основе
    /// двух новых особей. Далее с вероятносью MutationProbability
    /// осуществляется мутация.
    /// </summary>
    protected void NextGeneration()
    {
        CurrentIndividuals = new List<Individual>();
        for (int i = 0; i < IslandNumber; i++)
        {
            var newGeneration = new List<Individual>();
            int eliteSize = (int)(TruncationThreshold * IslandGenerationSize);
            while (newGeneration.Count < IslandGenerationSize)
            {
                Individual a1, a2;
                a1 = Islands[i][Random.Next(eliteSize)];
                a2 = Islands[i][Random.Next(eliteSize)];
                Individual[] s =
                    GeneticSearchOperator.Crossover(new[] { a1, a2 }, Random);
                newGeneration.AddRange(s);
            }

            if (newGeneration.Count > IslandGenerationSize)
                newGeneration.RemoveAt(newGeneration.Count - 1);

            for (int j = 0; j < newGeneration.Count; j++)
            {
                if (Random.NextDouble() < MutationProbability)
                    newGeneration[j] =
                        GeneticSearchOperator.Mutate(newGeneration[j], Random);
            }

            newGeneration.Sort();
            newGeneration.Reverse();

            Islands[i] = newGeneration;
            CurrentIndividuals.AddRange(Islands[i]);
            if (BestIndividual == null ||
                BestIndividual.CompareTo(Islands[i][0]) < 0)
            {
                BestIndividual = Islands[i][0];
                FileInfo bestIndividualFile =
                    new FileInfo("BestIndividual.txt");
                StreamWriter writer = bestIndividualFile.CreateText();
                writer.Write(BestIndividual.ToString());
                writer.Close();
                NoProgressTime = 0;
            }
        }
    }

    protected void BigMutation()
    {
        lock (Islands)
        {
            for (int i = 0; i < IslandNumber; i++)
            {
```

```
                    for (int j = 0; j < Islands[i].Count; j++)
                        if (Random.NextDouble() > 1 - BigMutationProbability)
                            Islands[i][j] = GeneticSearchOperator.Create(Random);
                    Islands[i].Sort();
                    Islands[i].Reverse();
                }
            }
        }

        protected void Migration()
        {
            int eliteSize = (int) (TruncationThreshold * IslandGenerationSize);
            for (int i = 0; i < IslandNumber; i++)
            {
                for (int k = 0; k < eliteSize; k++)
                    if (Random.NextDouble() > 1 - MigrationProbability)
                    {
                        int otherIsland = Random.Next(IslandNumber);
                        int otherIndividual = Random.Next(eliteSize);
                        Individual tmp = Islands[i][k];
                        Islands[i][k] = Islands[otherIsland][otherIndividual];
                        Islands[otherIsland][otherIndividual] = tmp;
                    }
                Islands[i].Sort();
                Islands[i].Reverse();
            }
        }
    }
}
```