МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

# **Кафедра вычислительных технологий**

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК

Заведующий кафедрой д.ф.- м.н., профессор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Миков \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

# **СРАВНИТЕЛЬНЫЙ АНАЛИЗ СЛОЖНОСТИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ НА ПРИМЕРЕ ЗАДАЧИ КОММИВОЯЖЕРА**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И. И. Ронжин

Факультет компьютерных технологий и прикладной математики Направление 02.03.02 – Фундаментальная информатика и информационные технологи

Научный руководитель В. В. Пашенцева

Нормоконтролер\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. С. Фисун

Краснодар 2016

# СОДЕРЖАНИЕ

Введение ....................................................................................................................... 3

1. Постановка задачи .................................................................................................. 5
2. Структура работы генетического алгоритма........................................................ 8
   1. Основные понятия .......................................................................................... 12
   2. Типы селекций ................................................................................................ 13
      1. Метод рулетки ...................................................................................... 13
      2. Ранговый метод .................................................................................... 15
      3. Турнирный метод ................................................................................. 16
   3. Типы скрещивания ......................................................................................... 18
      1. Одноточечное скрещивание ................................................................ 18
      2. Двухточечное скрещивание ................................................................ 18
      3. Одноэлементное скрещивание ........................................................... 19
      4. Поэлементное (равномерное) скрещивание ...................................... 20
   4. Влияние мутации ............................................................................................ 20
3. Сравнительный анализ генетического алгоритма ............................................. 22
   1. Размер популяции .......................................................................................... 24
   2. Количество поколений .................................................................................. 25
   3. Метод селекции .............................................................................................. 27
   4. Тип скрещивания ............................................................................................ 28
   5. Мутация ........................................................................................................... 30

Заключение ................................................................................................................ 32

Список использованных источников ...................................................................... 33

Приложение А Класс программы - GeneticAlg ...................................................... 35

Приложение Б Класс программы - FitnessFunction ............................................... 51

Приложение В Класс программы - Main ................................................................ 57

ВВЕДЕНИЕ

Генетический алгоритм – это эвристический алгоритм поиска, используется для решения задач оптимизации и моделирования путем случайного подбора, комбинирования и вариации искомых параметров с применением механизмов, напоминающих биологическую эволюцию. Генетические алгоритмы достаточно слабо развиты и нуждаются в более подробном изучении, как с теоретической стороны, так и с практической.

Эволюционные методы разработаны в 1960-х гг., однако только в настоящее время им нашли применение в силу высокой производительности ЭВМ. Многие современные компании используют такие алгоритмы для задач, для которых не разработаны специальные локальные методы или эти методы являются не эффективными.

Такой является задача коммивояжера. Проблема в том, что количество возможных маршрутов очень быстро возрастает с ростом n (оно равно n! — количеству способов упорядочения пунктов), и если количество городов будет более 100, то никакая современная ЭВМ не решит эту задачу перебором всех вариантов за объективное время.

В таком случае следует отказаться от попыток отыскать точное решение задачи коммивояжёра и сосредоточиться на поиске приближённого — пускай не оптимального, но хотя бы близкого к нему. В виду большой практической важности задачи полезными будут и приближённые решения.

Генетические алгоритмы позволяют приближенно решить задачу коммивояжера.

Целью исследования является разработка программы и проведение экспериментов для проведения сравнительного анализа генетического алгоритма.

Объектом исследования является задача коммивояжера.

Предметом исследования является решение задачи коммивояжера с помощью генетических алгоритмов.

В соответствии с объектом, предметом и целью исследования были определены следующие задачи исследования:

1. Реализация графовой модели;
2. Реализация генетического алгоритма;
3. Применение генетического алгоритма к графам;
4. Анализ полученных результатов.

1 Постановка задачи

Задача коммивояжёра — важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры выгодности маршрута (точнее говоря, невыгодности) может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута. Пример решения задачи коммивояжера изображен на рисунке 1.

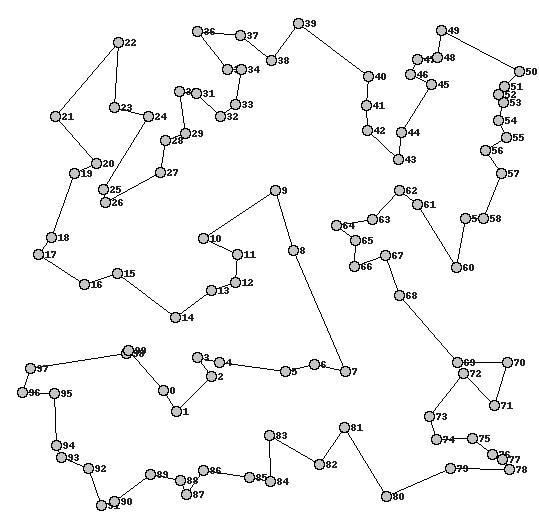


Рисунок 1 – Пример решения задачи коммивояжера

Совершенно очевидно, что задача может быть решена перебором всех вариантов объезда и выбором оптимального. Проблема в том, что количество возможных маршрутов очень быстро возрастает с ростом *n* (оно равно — количеству способов упорядочения пунктов). К примеру, для 100 пунктов количество вариантов будет представляться 158-значным числом — не выдержит ни один калькулятор. Мощная ЭВМ, способная перебирать миллион вариантов в секунду, будет биться с задачей на протяжении примерно *3\*10144* с. Увеличение производительности ЭВМ в 1000 раз даст хоть и меньшее в 1000 раз, но по-прежнему чудовищное время перебора вариантов. Не спасает ситуацию даже то, что для каждого варианта маршрута имеется *2n* равноценных, отличающихся выбором начального пункта (*n* вариантов) и направлением обхода (2 варианта). Перебор с учётом этого наблюдения сокращается незначительно — до вариантов.

Алгоритм, основанный на полном переборе вариантов, не является самым эффективным (в смысле быстродействия) для решения задачи коммивояжёра. Однако не существует алгоритма решения, имеющего степенную сложность (то есть требующего порядка операций для некоторого *a*) — любой алгоритм будет хуже. Всё это делает задачу коммивояжёра безнадёжной для ЭВМ с последовательным выполнением операций, если n хоть сколько-нибудь велико.

В таком случае следует отказаться от попыток отыскать точное решение задачи коммивояжёра и сосредоточиться на поиске приближённого — пускай не оптимального, но хотя бы близкого к нему. В виду большой практической важности задачи полезными будут и приближённые решения.

Заметим, что интеллект человека, не вооружённый вычислительной техникой, способен отыскивать такие приближённые решения задач, требующих огромного перебора вариантов в поисках оптимального. Например, шахматы. Человек может весьма успешно соперничать в этой игре с вычислительной машиной, либо вовсе не прибегая к перебору, либо сводя его к минимуму. Человек руководствуется при этом интуицией и набором эвристик (находок) — правил, которые обычно помогают в решении задач, хотя эффективность таких правил и не имеет достаточного обоснования.

В связи с этим, необходимо использовать алгоритм нахождения приближенного решения задачи Коммивояжера. В такую категорию входят генетические алгоритмы, решение которых основывается на постепенном изменении значений популяций. Эти алгоритмы позволяют находить как глобальный минимум функции, так и максимум, в зависимости от выбора типа селекции.

Эвристические алгоритмы позволяют решать практически любые задачи оптимизации, но их эффективность ниже, чем у локальных методов, что говорит о том, что данные алгоритмы чаще всего применяются к задачам, для которых не разработаны специальные локальные методы или решение такими методами являются неэффективными при заданных параметрах.

В общем случае задача коммивояжера – это задача отыскания кратчайшего гамильтонова цикла в полном графе.

Существует несколько частных случаев общей постановки задачи, в частности:

1. Геометрическая задача коммивояжера (планарная или евклидова, когда матрица расстояний отражает расстояния между точками на

плоскости);

1. Треугольная задача коммивояжера (когда на матрице стоимостей выполняется неравенство треугольника);
2. Симметричная задача коммивояжера (неориентированный граф);
3. Ассиметричная задача коммивояжера (ориентированный граф);
4. Обобщенная задача коммивояжера.

2 Структура работы генетического алгоритма

Эволюционные методы оптимизации представляют большой интерес. Упоминания о применении генетических алгоритмов для решения задачи оптимизации относится к концу 1960-х гг. Эволюционные методы основываются на примере работы эволюции и обучения, в частности, к таким методам относятся генетические алгоритмы, нейронные сети и эволюционные алгоритмы [9].

Генетический алгоритм – это эвристический алгоритм поиска, используется для решения задач оптимизации и моделирования путем случайного подбора, комбинирования и вариации искомых параметров с применением механизмов, напоминающих биологическую эволюцию. Этот алгоритм является разновидностью эволюционных вычислений. Отличительной чертой генетического алгоритма является то, что акцент падает на использование оператора скрещивания, производящего операцию рекомбинации решений-кандидатов (геномы), роль которой аналогична роли скрещивания в живой природе. Это новый, но не единственный способ решения задач оптимизации [8].

Генетические алгоритмы возникли в результаты наблюдения и попыток копирования естественных процессов, происходящих в мире живых организмов, в частности, эволюции и связанной с ней селекции популяций живых существ. При сопоставлении нейронных сетей и генетических алгоритмов следует упомянуть, что у них принципиальные различия в длительности протекания упоминаемых естественных процессов, т.е. на чрезвычайно быструю обработку информации в нервной системе и очень медленных процесс естественной эволюции. Однако при компьютерном моделировании эти различия оказывается несущественными.

В генетических алгоритмах применяется ряд терминов, заимствованных из генетики, прежде всего гены и хромосомы, а также популяция, особь, аллель, генотип и т.д.

Для применения алгоритма задачи приводятся к виду, при котором решение может быть представлено как набор более мелких составных частей (аналог генотипа и его частей – генов). Длины этих генотипов могут быть как фиксированной величиной, так и переменной.

Алгоритм состоит из нескольких основных шагов:

1. Подготовка – формирование начальной популяции, т.е. начальный набор решений. Чаще всего используют случайный процесс формирования с целью охватить большее разнообразие для поиска решений, однако алгоритм для формирования может быть различным. Так, например, формирование может происходить с заранее известными свойствами, но следует иметь в виду, что это может повлиять на ход развития системы.
2. Оценка приспособленности особей в популяции и их отбор – важный этап алгоритма, где происходит выбор направления развития популяций. Чаще всего отбрасываются решения с низким значением фитнесс-функции (или с высоким, в зависимости от того, в чем состоит задача – в минимизации или в максимизации функции). Это способствует улучшению средней приспособленности всей популяции.
3. Проверка условия остановки алгоритма – происходит определение условия остановки генетического алгоритма. Это условие зависит от конкретного применения алгоритма.
4. Селекция особей – этап, в котором выбираются хромосомы, которые будут участвовать в создании потомков для следующей популяции, т.е. для очередного поколения. Такой выбор производится согласно принципу естественного отбора, по которому наибольшие шансы на участи в создании новых особей имеют родители с наибольшими значениями фитнесс-функции. 5. Скрещивание и мутация – этап, на котором происходит образование новых решений в популяции, прошедшей через отбор, для восстановления численности после этапа отбора. При скрещивании берутся два или более существующих генома из популяции, затем определенным образом происходит соединение составных частей в новый геном, который остается в популяции и будет являться новым решением. Затем геном подвергается частичным изменением. Этот этап позволяет популяциям быть более разнообразными.
5. Формирование новой популяции после всех внесенных изменений.
6. Оценка решений, выбор наилучшей особи – после выполнение предыдущих этапов проверяется было ли найдено необходимое решение. Однако чаще для решения этой задачи применяется подход с числом поколений (число создаваемых популяций).

На рисунке 2 изображена структура работы алгоритма.

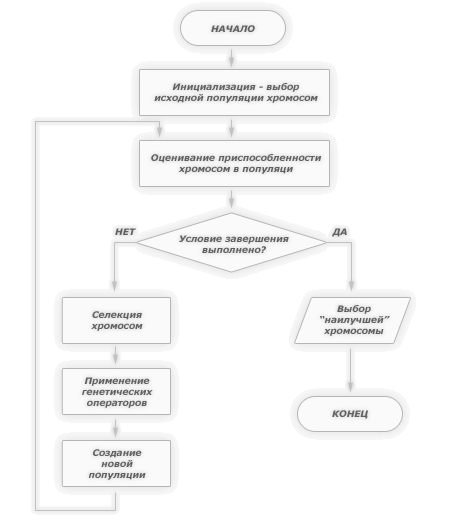


Рисунок 2 – Структура генетического алгоритма

В коде программы за последовательность выполнения этапов алгоритма отвечает класс GeneticAlg в методе run (Приложение А).

2.1 Основные понятия

Для применения генетического алгоритма необходимо определить основные структурные элементы: вид элемента популяции (генотип, особь, хромосома), процесс скрещивания, селекция, мутация, вид фитнесс-функции.

Особью будет являться элемент популяции, за который будем принимать маршрут через все города. Каждый такой маршрут является возможным решением и не противоречит условиям задачи, хотя это и не обозначается, что он будет являться оптимальным. Предполагаем, что все элементы популяции корректны, т.е. все они – потенциальные решения поставленной задачи и не являются противоречивыми. Они представляются хромосомами с закодированными в них множествами параметров задачи, т.е. решений, которые иначе называются точками в пространстве поиска.

Фитнесс-функция (функция приспособленности) – представляет меру приспособленности данной особи в популяции. Эта функция играет важнейшую роль, поскольку позволяет оценить степень приспособленности конкретных особей в популяции и выбрать из них наиболее приспособленные в соответствии эволюционным принципом выживания «сильнейших». Фитнесс-функция принимает вид формулы (1).

(1)

где P – множество всех связей в маршруте. Значение этой функции определяет длину пути. Соответственно, необходимо будет сортировать значения необходимым образом для нахождения минимума или максимума, что позволит оценить получаемые решения и определить минимизируемый параметр.

В коде программы класс FitnessFunction (Приложение Б) отвечает за работу с фитнесс-функциями.

2.2 Типы селекций

2.2.1 Метод рулетки

Основан на принципе колеса рулетки. Этот метод считается для генетических алгоритмов основным. Выбор особей происходит случайным образом, однако родительские особи выбираются пропорционально значениям их фитнесс-функции.

Каждой особи сопоставлен сектор колеса рулетки, величина которого устанавливается пропорционально значению фитнесс-функции этой хромосомы, поэтому, чем больше значение функции приспособленности, тем больше сектор на колесе рулетки. Из этого следует то, что чем больше сектор на колесе рулетки, тем выше шанс, что особи с очень малым значением фитнесс-функции довольно быстро исключаются из данной популяции. Это может привести к преждевременной сходимости генетического алгоритма.

Общий алгоритм работы:

1. Вычисляем значение фитнесс-функции (fv) для каждой особи в популяции.
2. Находим сумму всех функций приспособленности (Sf) в популяции, формула (2).

(2)

1. Вычисляем среднее значение фитнесс-функции (Af) на популяции, формула (3).

(3)

1. Находим ожидаемое значение функции приспособленности (Ef) для каждой особи в популяции, формула (4).

(4)

1. Вычисляем сумму ожидаемых фитнесс-функций (Sum Ef) для всех

хромосом, формула (5).

(5)

1. Генерируем случайное число (G), которое входит в множество [0, SumEF], формула (6).

(6)

1. Выбираем особь, которая будет соответствовать случайному числу на рулетке, добавляем ее в родительский пул.
2. Возвращаемся к шестому шагу n раз, где n – размер популяции.

Таким образом, получим рулетку, где каждой особи будет

соответствовать сегмент, величина которого будет равна величине фитнессфункции этой хромосомы, и затем случайным образом будут выбираться потомки для дальнейшего скрещивания [14].

На рисунках 3 и 4 изображен пример распределения вероятностей родителей.

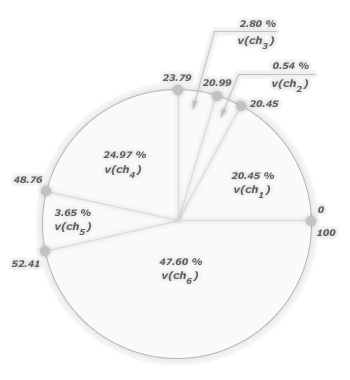


Рисунок 3 – Механизм распределения родителей на рулетке

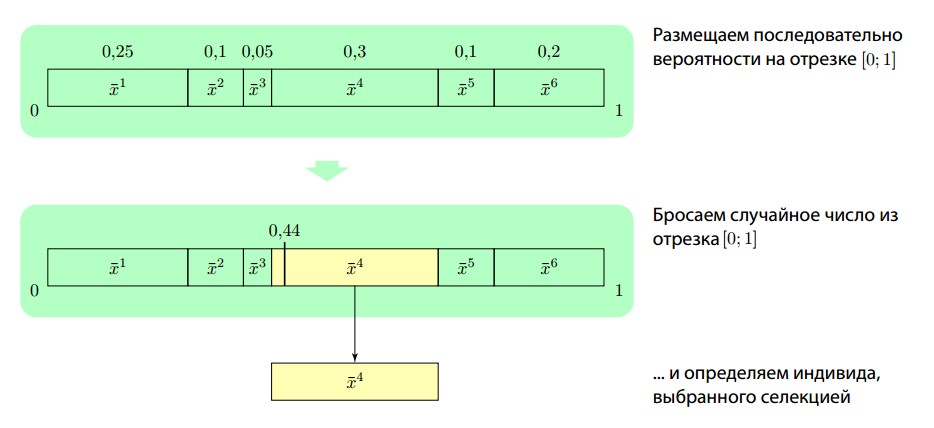


Рисунок 4 – Механизм работы метода Рулетка [10]

Имеет асимптотическую сложность O(*n*2), так как требуется массив для вычисления и хранения вероятности выпадения.

2.2.2 Ранговый метод

При ранговой селекции хромосомы ранжируются по их значениям функции приспособленности. В итоге получается отсортированный список особей, упорядоченных от наиболее приспособленных к наименее приспособленным, или наоборот. Каждой особи приписывается ранг (число), которое соответствует ее месту в списке. Если в списке есть некоторое количество одинаковых, по фитнесс-функции, элементов, то ранг им присваивается как среднеарифметический ранг этих двух элементов.

Основным преимуществом этого метода является тот факт, что его возможно применять как для нахождения минимума функции, так и для нахождения максимума [10].

На рисунке 5 представлен пример работы ранговой селекции.



## Рисунок 5 – Механизм работы ранговой селекции [10]

Имеет асимптотическую сложность O(*n*2), так как требуется массив для хранения и сортировки рангов.

2.2.3 Турнирный метод

Метод заключается в том, что случайным образом выбираются две особи, после чего происходит сравнение их фитнесс-функций. Хромосомы с большим значением функции приспособленности выбирается далее, для скрещивания.

Этот метод имеет аналогичное преимущество – используется и для минимизации, и для максимизации функции. Допускается изменение размера подгрупп особей, на которые разделяется популяция.

Общий алгоритм работы:

1. Случайным образом выбираются несколько (обычно две) хромосом.
2. Выбирается хромосома с большей фитнесс-функцией.
3. Добавление особи в родительский пул.

Преимуществом данного метода является отсутствие преждевременной сходимости. Так как особи выбираются случайным образом, то вероятность того, что слабые особи полностью покинут популяцию несколько ниже, чем в других методах. Кроме этого, можно отметить его простоту и тот факт, что не требуется глобальное переупорядочивание всех хромосом. Турнирная селекция имеет более глубокие аналоги в биологии [9]. На рисунке 6 изображен механизм работы турнирной селекции.

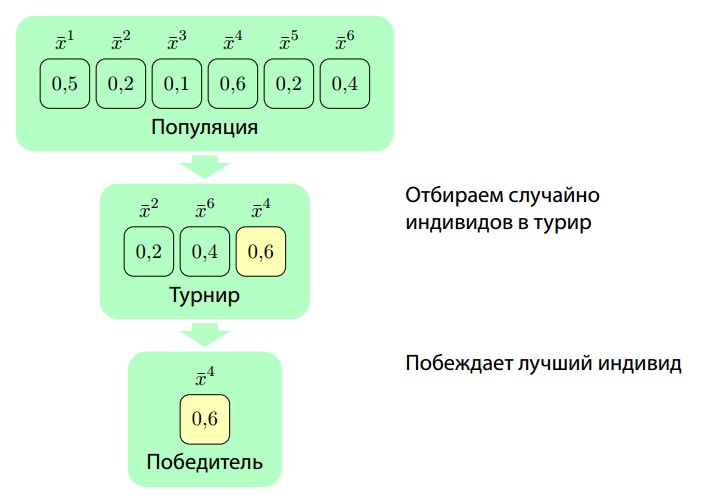


Рисунок 6 – Механизм работы турнирной селекции [10]

Имеет асимптотическую сложность O(1).

2.3 Типы скрещивания

* + 1. Одноточечное скрещивание

Представляет собой метод, при котором хромосомы родителей разрезаются в точке, которая выбирается случайным образом, затем происходит обмен правыми частями хромосом (рисунок 7).

Сложность алгоритма: О(L), где L – длина хромосомы.



Рисунок 7 – Механизм работы одноточечного скрещивания

* + 1. Двухточечное скрещивание

Хромосомы рассматриваются как кольца, где первые и последние гены связываются. После этого кольцо разделяется на две части, затем между частями происходит обмен (рисунок 8).

Сложность алгоритма: О(L), где L – длина хромосомы.



Рисунок 8 – Механизм работы двухточечного скрещивания

* + 1. Одноэлементное скрещивание

Случайным образом выбирается один геном хромосомы, после чего происходит обмен между особями (рисунок 9).

Сложность алгоритма: О(1), так как происходит обмен одним геном.

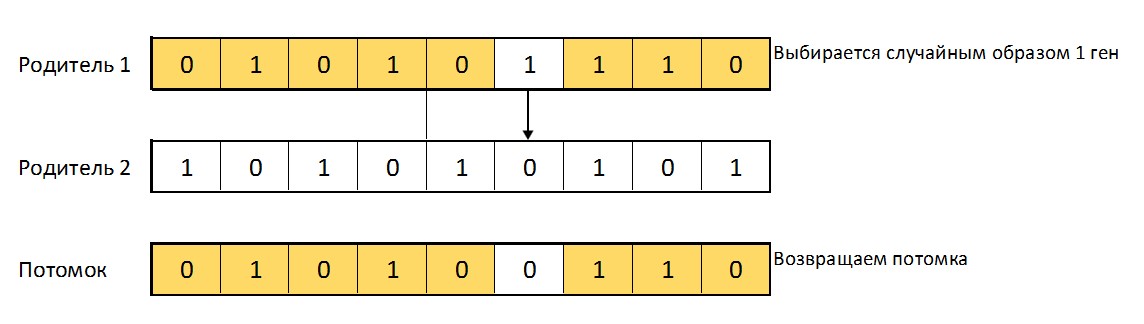


Рисунок 9 – Механизм работы одноэлементного скрещивания

* + 1. Поэлементное (равномерное) скрещивание

Каждый ген хромосомы с определенной вероятностью скрещивается с другим геном хромосомы (рисунок 10).

Сложность алгоритма: О(L), где L – длина хромосомы.

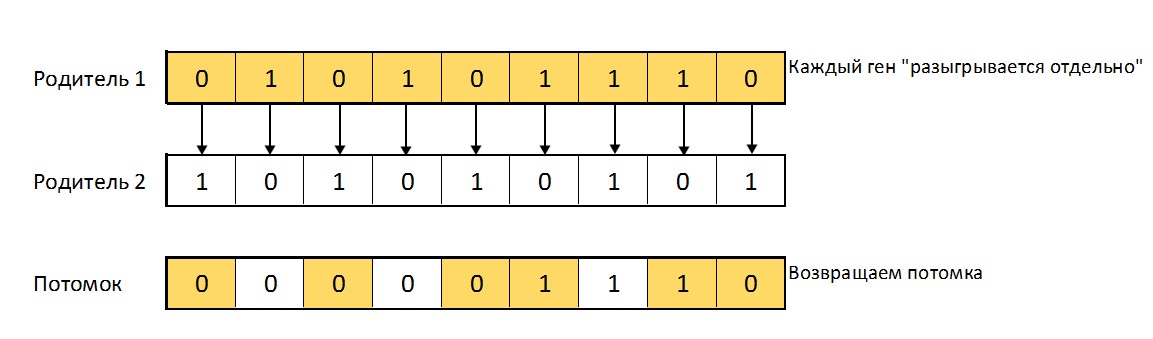


Рисунок 10 – Механизм работы поэлементного скрещивания

2.4 Влияние мутации

Мутация – оператор, который отвечает за случайное изменение всех потомков из популяции. Цель этого оператора стоит в том, что необходимо вносить разнообразие в популяцию, как это происходит случайным образом в эволюции.

Мутация, обычно, предполагает небольшое изменение хромосомы. При осуществлении мутации каждый ген хромосомы с определенной заданной вероятностью мутирует, то есть его значение меняется на противоположное.

Различают 3 вида мутации: слабая, средняя, сильная.

Вероятность мутации (M) определяется формулой (7).

(7)

где n – длина хромосомы [10].

3 Сравнительный анализ генетического алгоритма

Для каждой задачи оптимизации необходимо найти такие характеристики, при котором поиск решения будет занимать минимальное время. Некоторые задачи будут решаться быстрее при выборе определенного метода селекции, процента мутации и т.д. Необходимо провести анализ каждой характеристики, определить путь, при котором решение задачи коммивояжера будет происходить оптимальным образом.

Следует провести необходимое количество экспериментов с разным набором характеристик.

Так же требуется использовать графы с разным количеством вершин для адекватного сравнения. Будут использоваться графы с количеством вершин: 256, 512, 1024, 2048, 4096, 8192.

Введем коэффициент K, который имеет вид функции (8).

(8)

Его значение определяет отношение длины среднего случайного пути к длине найденного пути. Чем больше это значение, тем короче является найденный путь. Коэффициент К необходим для более наглядного представления информации на графике.

На рисунках 11 и 12 изображен пример работы программы при разных количествах поколений.

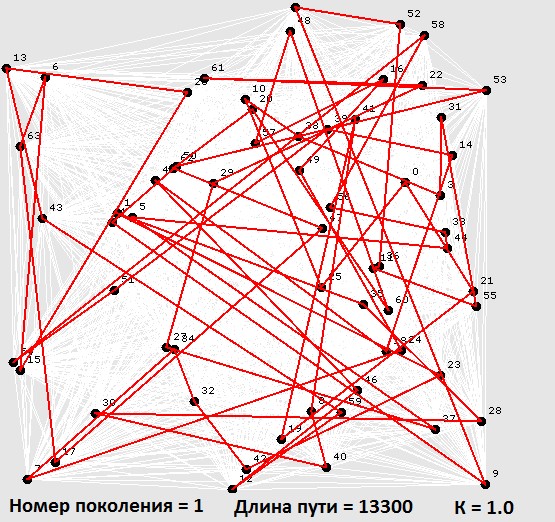
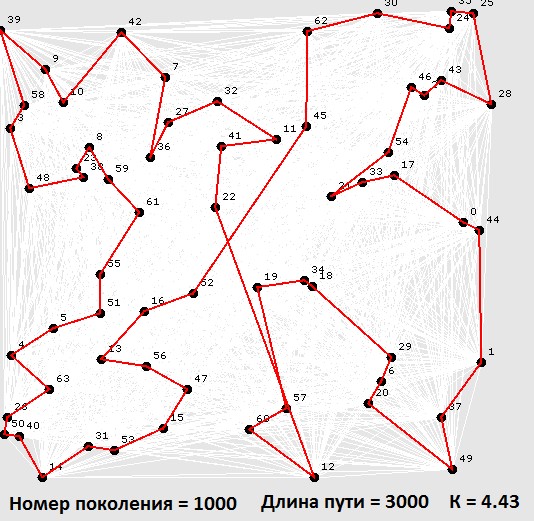


Рисунок 11 – Пример работы алгоритма



## Рисунок 12 – Пример работы алгоритма

3.1 Размер популяции

Необходимо исследовать влияние размера популяции на получаемый результат решения задачи. Другие параметры генетического алгоритма будут следующими:

1. Количество поколений: 1000;
2. Метод селекции: турнирный; 3. Скрещивание: поэлементное;

4. Вероятность мутации: 2%.

В качестве оценочных значений размеры популяций будут равны: 10, 100, 500, 1000, 5000, 10000.

После проведения всех испытаний получаем следующий график

(рисунок 13).

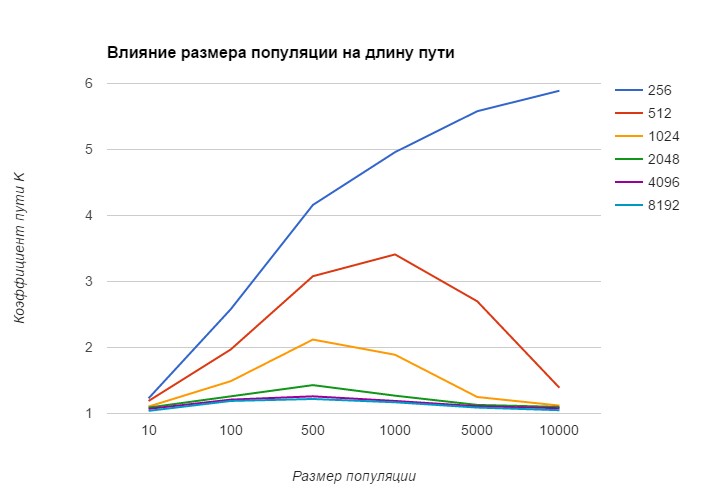


Рисунок 13 – Влияние размера популяции на длину пути

Очевидно, что для графов с количеством вершин более 500 – 1000, нет необходимости проводить вычисления с большим количеством особей, так как это увеличение не дает улучшения результата.

3.2 Количество поколений

Проведем исследование, в котором определим, как влияет количество поколений на результат работы алгоритма. Параметры алгоритма следующие:

1. Количество особей: 1000;
2. Метод селекции: турнирный;
3. Скрещивание: поэлементное;
4. Вероятность мутации: 2%.

Эксперименты проводятся с значением количества поколений: 10, 100, 1000, 10000.

Проведенные эксперименты предоставляют следующие результаты

(рисунок 14).

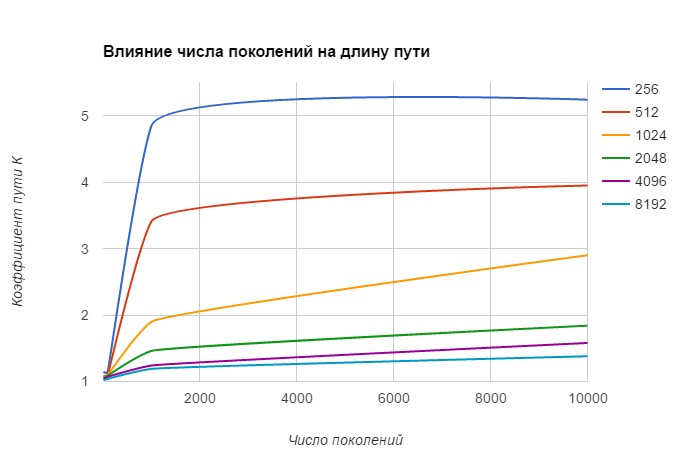


Рисунок 14 – Влияние числа поколений на длину пути

Постоянный рост коэффициента показывает, что увеличение числа поколений позитивно влияет на результат работы алгоритма и это естественно – с каждым поколением популяция изменяется и стремится к минимальному значению фитнесс-функции. Чем больше число поколений, тем точнее результат.

3.3 Метод селекции

С помощью селекции происходит отбор хромосом, которые затем будут участвовать в скрещивании. Необходимо определить наиболее эффективный алгоритм селекции для эффективных вычислений.

1. Количество особей: 1000;
2. Количество поколений: 1000;
3. Скрещивание: поэлементное;
4. Вероятность мутации: 2%.

В эксперименте будут исследоваться три типа селекции: турнирная, ранговая и «Рулетка».

Был получен график, представленный на рисунке 15.

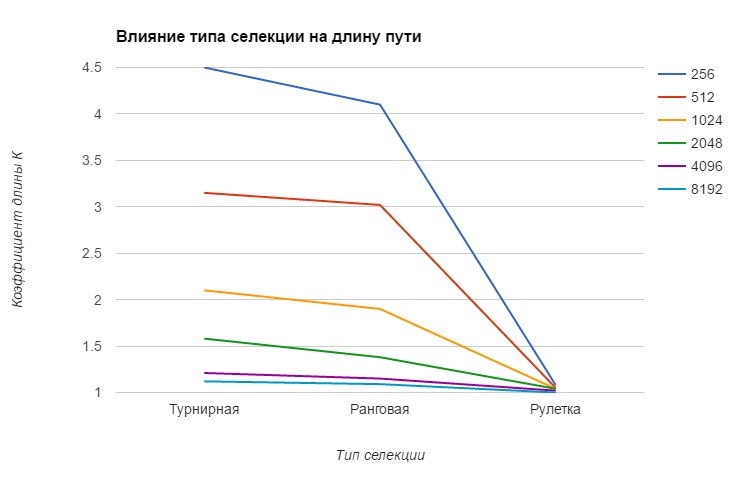


Рисунок 15 – Влияние типа селекции на длину пути

Можно увидеть, что метод рулетки наименее эффективен в задаче коммивояжера, так как имеет высокую сходимость, т.е. особи с малым значением фитнесс-функции вырождаются из популяции намного быстрее, чем при использовании турнирного и рангового методов.

3.4 Тип скрещивания

Основной частью эволюции является скрещивание. Необходимо провести эксперименты с разными видами скрещивания для определения наиболее эффективного метода.

1. Количество особей: 1000;
2. Количество поколений: 1000;
3. Метод селекции: турнирный;
4. Вероятность мутации: 2%.

Типы скрещивания, которые требуется изучить: одноточечный, двухточечный, поэлементный, одноэлементный.

Получаем результаты, представленные на рисунках 16 и 17.

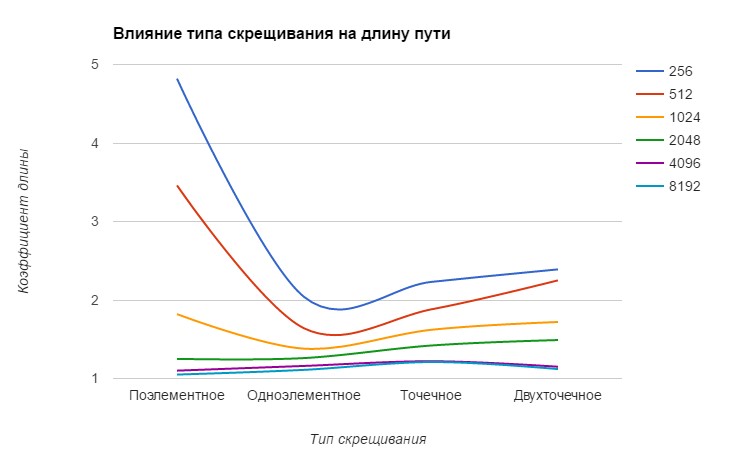


Рисунок 16 – Влияние типа скрещивания на длину пути

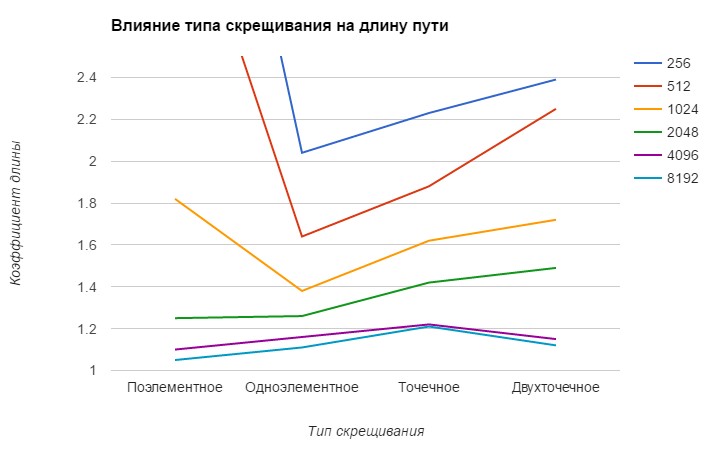


Рисунок 17 – Влияние типа скрещивания на длину пути в увеличенном масштабе

Поэлементный тип скрещивания наиболее эффективно влияет на результат работы алгоритма, когда вычисления проводятся над графами с небольшим количеством вершин (менее 2000). Точечный тип скрещивания наиболее эффективен в графах с большим количеством вершин (более 2000).

3.5 Мутация

Следующие эксперименты будут проведены с разными видами мутации: слабая, средняя, сильная. Так же необходимо провести испытание без мутации для того, чтобы картина была полная.

1. Количество особей: 1000;
2. Количество поколений: 1000;
3. Метод селекции: турнирный;
4. Метод скрещивания: поэлементный;

Получили результаты, представленные на рисунке 18.

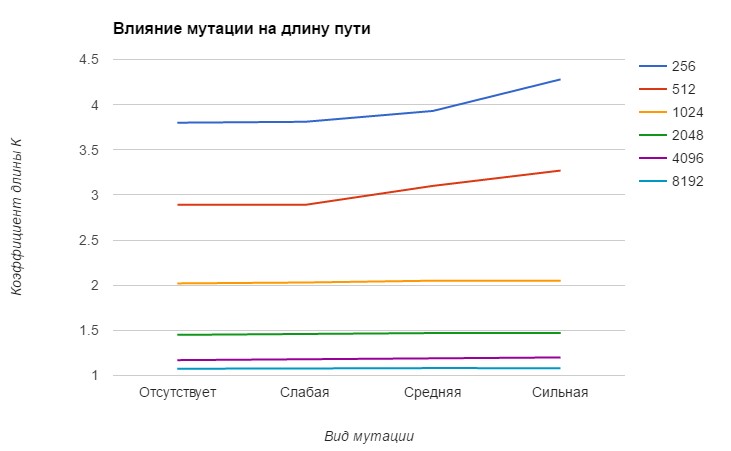


Рисунок 18 – Влияние мутации на длину пути

Очевидно, что на графах с малым количеством вершин мутация влияет на результат сильнее, чем на графы с большим количеством вершин (> 1000). В любом случае, мутация оказывает положительный эффект на результат работы алгоритма.

# ЗАКЛЮЧЕНИЕ

Для каждой задачи необходимо подбирать такие характеристики, при которых решение задачи будет удовлетворять условию.

Проведенные эксперименты показывают, что увеличение размера популяции не всегда оказывает позитивный эффект на результат работы алгоритма. Имеет асимптотическую линейную сложность O(n).

Увеличение количества поколений необходимо для эффективного решения задачи, так как популяция видоизменяется и стремится к тому значению фитнесс-функции, который был задан. Асимптотическая сложность O(n).

Турнирный метод селекции, очевидно, наиболее удачный выбор для решения задачи коммивояжера, кроме этого имеет наименьшую сложность среди методов селекции. Недостаток метода рулетки присутствует на любом графе – высокая сходимость.

Тип скрещивания для разных размеров графов необходимо выбирать исходя из экспериментов. Так, например, поэлементный тип скрещивания на малых графах намного эффективнее остальных, но на больших графах наиболее удачным выбором будет являться точечное скрещивание.

Мутация во всех испытаниях показывает позитивное влияние на результат.

Решение поставленной задачи считаю частично выполненной, для графов с большим количеством вершин необходима ЭВМ с большей вычислительной мощностью.

Полученные результаты исследования могут внести вклад в развитие знаний о генетических алгоритмах и помогают использовать эти методы

более эффективно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванов Д.В.Разработка метода маршрутизации для беспроводной ячеистой сети с учетом качества обслуживания. [Электронный ресурс] Статья, 2013. – С.34, URL: dlib.rsl.ru/01004597354, [Дата обращения: 1 февраля 2015].
2. Любимова Т.В[.](http://elibrary.ru/author_items.asp?authorid=757111) Решение комбинаторных задач методами эволюционных вычислений. [Электронный ресурс] Статья, 2015. – С.39, URL: elibrary.ru/item.asp?id=23484461, [Дата обращения: 18 марта 2015].
3. Макконнел Дж. Основы современных алгоритмов. // М.: Наука, 2012.

– С.368.

1. Миков А.И., Кособуцкая Е.В, Лапина О.Н Вычислимость и сложность алгоритмов // Краснодар, 2013. – С.52.
2. Мищенко В.А. Использование генетических алгоритмов в обучении нейронных сетей**.** [Электронный ресурс] Статья, 2015. – С.21, URL: scienceeducation.ru/ru/article/view?id=5138, [Дата обращения: 18 марта 2015].
3. Моров В.А. Применение генетического алгоритма к задачам оптимизации. [Электронный ресурс] Статья, 2006. – С.88, URL: people.amursu.ru, [Дата обращения: 3 марта 2016].
4. Никитин В.В. Логико-генетический метод оптимизации АСТПП авиадвигателестроения в условиях управления проектами «бережливого» производства. [Электронный ресурс] Статья, 2014. – С.30, URL: dlib.rsl.ru/01005381081, [Дата обращения: 12 марта 2015].
5. Носов В.А. Основы теории алгоритмов и анализа их сложности // М.:

Мир, 1992. – С.140.

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы // М.: Горячая линия, 2006. – С.383.
2. Сергиенко А. Б. Описание множества операторов для алгоритмов оптимизации. [Электронный ресурс] Статья, 2015. – С.28, URL: github.com/Harrix, [Дата обращения: 12 декабря 2015].
3. Харабет В.В. Подготовка специалистов социальной работы в контексте регулирования занятости молодёжи: региональный аспект.

[Электронный ресурс] Статья, 2012. – С.175, URL: elibrary.ru/item.asp?id=17872049, [Дата обращения: 7 марта 2015].

1. Alabsi F., Naoum R. Comparison of Selection Methods and Crossover

Operations using Steady State Genetic Based Intrusion Detection System. [Электронный ресурс] Статья, 2012. – C.6, URL: citeseerx.ist.psu.edu, [Дата обращения: 15 февраля 2016].

# ПРИЛОЖЕНИЕ А

## Класс программы – GeneticAlg

import java.util.Random;

public class GeneticAlg{

public long timeToselect = 0; public long timeTocross = 0; public long timeToMutate = 0; public long timeToFF = 0;

public static final double FULLNESS = 0.999d; public static final selectType SELECT = selectType.TOURNIR;

public static final crossType CROSS =

crossType.RECOMBINATION\_ONE\_POINT;

public static final boolean MUTATION = true; public static final long GENERATION = 10000L;

public static final int LENGTH = 64; //длина public static final int MASK = LENGTH - 1; public static final int SHIFT; static { int shiftForDivision = 0; int tmp = LENGTH; while (tmp > 1) { tmp >>= 1; shiftForDivision++;

}

SHIFT = shiftForDivision;

}

public enum selectType { TOURNIR {

public String toString() { return "TOURNIR";

}

}, ROULETTE { public String toString() { return "ROULETTE"; }

}, RANK { public String toString() { return "RANK";

}

}

}

public enum crossType {

RECOMBINATION\_ONE\_POINT {

public String toString() {

return "RECOMBINATION\_ONE\_POINT";

}

}, RECOMBINATION\_TWO\_POINT {

public String toString() {

return "RECOMBINATION\_TWO\_POINT";

}

}, RECOMBINATION\_ELEMENTWISE {

public String toString() {

return "RECOMBINATION\_ELEMENTWISE";

}

}, RECOMBINATION\_ONE\_ELEMENT {

public String toString() {

return "RECOMBINATION\_ONE\_ELEMENT";

}

}

}

private Fitness fitFunct;

private int indLength; //Длина генома в битах private int sizeOfArray;

private long genCount; //Количествово поколений private int indCount; //Количествово геномов в поколении private selectType selectType; //Тип Селекции private crossType crossType; //Тип Скрещивания private boolean usemutations; //Использовать ли мутацию private double mutationsProcent; //Вероятность мутации private long[][] ListParents; private long[][] ListOffsprings; private long[] actual; private long[] FunctionResult; private long currentGeneration = 0;

private Random random = new Random(System.currentTimeMillis());

public GeneticAlg(Fitness fitFunct) { this.fitFunct = fitFunct; this.indLength = fitFunct.getAr(); this.sizeOfArray = (int) Math.ceil((double) this.indLength / LENGTH); this.genCount = GENERATION;

this.indCount = (int) (1 + Math.log(1 / Math.pow(1 - FULLNESS, 1 /

indLength)) / Math.log(2)); this.selectType = SELECT; this.crossType = CROSS; this.usemutations = MUTATION;

this.mutationsProcent = indLength \* (1 - Math.pow((1 - 10 \* Math.pow((1 / 2), (indLength - 1))),(1 / indLength)));

}

// Main loop public long[] run() {

//Подготовка this.ListParents = new long[this.indCount][]; this.ListOffsprings = new long[this.indCount][]; this.FunctionResult = new long[this.indCount]; this.actual = new long[this.indCount]; for (int i = 0; i < this.indCount; i++) { this.actual[i] = -1;

}

//Генерируем первое поколение this.generateFirst();

while (this.currentGeneration < this.genCount) {

this.select(); this.cross(); if (this.usemutations) { this.mutations();

}

long[][] tmp = this.ListParents; this.ListParents = this.ListOffsprings; this.ListOffsprings = tmp;

this.currentGeneration++;

}

long bestFunctionResult = 0; long[] bestindividual = null; for (long[] individual : this.ListParents) { long FunctionResult = this.fitFunct.run(individual); if (bestFunctionResult <= FunctionResult) { bestindividual = individual; bestFunctionResult = FunctionResult;

}

}

return bestindividual;

}

//Генерация первого поколения private void generateFirst() { for (int i = 0; i < this.indCount; i++) { this.ListParents[i] = this.generateindividual();

}

}

//Генерация одного генома private long[] generateindividual() { long[] result = new long[this.sizeOfArray]; for (int i = 0; i < this.sizeOfArray; i++) { result[i] = this.random.nextLong();

} return result;

}

//Селекция - выбор геномов для селекции private void select(){

long old = System.currentTimeMillis(); //время

switch (selectType) { case ROULETTE:{

float[] wheels = new float[this.indCount]; wheels[0] = this.getFunctionResult(0); for (int i=1;i<this.indCount;i++){ wheels[i] = wheels[i-1] + this.getFunctionResult(i);

}

float all = wheels[this.indCount-1];

for (int i=0;i<this.indCount;i++){ float index = Math.abs(this.random.nextFloat())\*all;

int la = 0; int rr = indCount-1; int c = 0; while (la < rr){ c = (la+rr) >> 1; if (wheels[c] >= index){ rr = c; } else { la = c + 1;

}

}

this.ListOffsprings[i] = this.ListParents[l].clone();

} break;

}

case TOURNIR:{ for (int i=0;i<this.indCount;i++){ int first = random.nextInt(indCount); int second = random.nextInt(indCount);

long ffTime = System.currentTimeMillis(); //время

long frame1 = this.getFunctionResult(first); long frame2 = this.getFunctionResult(second);

this.timeToFF += (System.currentTimeMillis() - ffTime);

//время

this.ListOffsprings[i] = frame1 > frame2 ?

this.ListParents[first].clone() : this.ListParents[second].clone();

} break;

}

case RANK: { float[] ranked = new float[this.indCount]; float fitnessResultSum = 0;

for (int i = 0; i < this.indCount; i++) { float fitnessResult = this.getFunctionResult(i); ranked[i] = fitnessResult; fitnessResultSum += fitnessResult;

}

float[][] rankedMas = new float[this.indCount][10]; for (int i = 0; i < this.indCount; i++) { float maxFunction = 0; int maxFunctionIndividualNumber = 0; for (int j = 0; j < this.indCount; j++) { if (ranked[j] >= maxFunction) { maxFunction = ranked[j]; maxFunctionIndividualNumber = j;

}

}

rankedMas[i][0] = maxFunction; rankedMas[i][10] = maxFunctionIndividualNumber; if (i == 0) { rankedMas[i][9] = maxFunction / fitnessResultSum;

} else {

rankedMas[i][9] = rankedMas[i-1][9] + (maxFunction) /

fitnessResultSum;

}

ranked[maxFunctionIndividualNumber] = 0;

}

for (int j = 0; j < indCount; j++) { float rand = random.nextFloat(); int i = 0; float index = -1; while (i < this.indCount || index < 0) { if (rankedMas[i][9] > rand) index = i; i++;

}

this.ListOffsprings[j] = this.ListParents[(int)index].clone();

}

break;

} default:

throw new UnsupportedOperationException();

}

this.timeToselect += (System.currentTimeMillis() - old); // time

}

//Скрещивание всех геномов поколения private void cross() {

long old = System.currentTimeMillis(); //время

for (int i = 0; i < indCount / 2; i++) { int first = i << 1; int second = first | 1; cross(this.ListOffsprings[first], this.ListOffsprings[second]);

}

this.timeTocross += (System.currentTimeMillis() - old); //время

}

//Получение фитнес-функции private long getFunctionResult(int individualNumber) { if (this.actual[individualNumber] != this.currentGeneration) {

this.FunctionResult[individualNumber] =

this.fitFunct.run(this.ListParents[individualNumber]); this.actual[individualNumber] = this.currentGeneration;

}

return this.FunctionResult[individualNumber];

}

//Скрещивание двух геномов private void cross(long[] individualfirst, long[] individualsecond) { switch (crossType) {

case RECOMBINATION\_ONE\_POINT:{

int index = this.random.nextInt(this.indLength); int outerOffset = index >> SHIFT; int innerOffset = LENGTH - (index & MASK); long tmp = 0;

if (innerOffset < 63) { long mask = 1L << (innerOffset + 1) - 1;

long swap = (individualfirst[outerOffset] ^

individualsecond[outerOffset]) & mask;

individualfirst[outerOffset] ^= swap; individualsecond[outerOffset] ^= swap; outerOffset++;

}

for (int i=outerOffset;i<this.sizeOfArray;i++){

tmp = individualfirst[i]; individualfirst[i] = individualsecond[i]; individualsecond[i] = tmp;

}

break;

}

case RECOMBINATION\_TWO\_POINT:{

int first = this.random.nextInt(this.indLength); int second = this.random.nextInt(this.indLength); int startIndex = Math.min(first, second); int endIndex = Math.max(first, second); int startOuterOffset = startIndex >> SHIFT; int startInnerOffset = LENGTH - (startIndex & MASK); int endOuterOffset = endIndex >> SHIFT; int endInnerOffset = LENGTH - (endIndex & MASK); long tmp = 0;

if (startInnerOffset < LENGTH-1) { long mask = 1L << (startInnerOffset + 1) - 1; long swap = (individualfirst[startOuterOffset] ^

individualsecond[startOuterOffset]) & mask; individualfirst[startOuterOffset] ^= swap; individualsecond[startOuterOffset] ^= swap; startOuterOffset++;

}

for (int i=startOuterOffset;i<=endOuterOffset;i++){ tmp = individualfirst[i]; individualfirst[i] = individualsecond[i]; individualsecond[i] = tmp;

}

if (endInnerOffset > 0) { long mask = 1L << endInnerOffset - 1;

long swap = (individualfirst[endOuterOffset] ^

individualsecond[endOuterOffset]) & mask; individualfirst[endOuterOffset] ^= swap; individualsecond[endOuterOffset] ^= swap;

}

break;

}

case RECOMBINATION\_ELEMENTWISE:{ for (int outerOffset = 0; outerOffset < this.sizeOfArray;

outerOffset++) { long mask = this.random.nextLong();

long swap = (individualfirst[outerOffset] ^

individualsecond[outerOffset]) & mask;

individualfirst[outerOffset] ^= swap; individualsecond[outerOffset] ^= swap;

} break;

}

case RECOMBINATION\_ONE\_ELEMENT:{

int index = this.random.nextInt(this.indLength); int outerOffset = index >> SHIFT; int innerOffset = LENGTH - (index & MASK); long mask = 1L << innerOffset;

long swap = (individualfirst[outerOffset] ^

individualsecond[outerOffset]) & mask;

individualfirst[outerOffset] ^= swap; individualsecond[outerOffset] ^= swap; break;

} default:

throw new UnsupportedOperationException();

}

}

//Мутация private void mutations() {

long old = System.currentTimeMillis(); //время

for (long[] individual : this.ListOffsprings) { if (random.nextDouble() <= mutationsProcent) { mutate(individual);

}

}

this.timeToMutate += (System.currentTimeMillis() - old); //время

}

//Мутация одного генома private void mutate(long[] individual) { int index = this.random.nextInt(this.indLength); int outerOffset = index >> SHIFT; int innerOffset = (index & MASK); long mask = 1L << innerOffset; individual[outerOffset] ^= mask;

}

public long getgenCount() { return genCount;

}

public void setgenCount(long genCount) { this.genCount = genCount;

}

public int getindCount() { return indCount;

}

public void setindCount(int indCount) { this.indCount = indCount;

}

public selectType getselectType() { return selectType;

}

public void setselectType(selectType selectType) { this.selectType = selectType;

}

public crossType getcrossType() { return crossType;

}

public void setcrossType(crossType crossType) {

this.crossType = crossType;

}

public boolean getUsemutations() { return usemutations;

}

public void setUsemutations(boolean usemutations) { this.usemutations = usemutations;

}

public double getmutationsProcent() { return mutationsProcent;

}

public void setmutationsProcent(double mutationsProcent) { this.mutationsProcent = mutationsProcent;

}

public long[][] getListParents() { return ListParents;

}

public void setListParents(long[][] ListParents) { this.ListParents = ListParents;

}

public long[][] getListOffsprings() { return ListOffsprings;

}

public void setListOffsprings(long[][] ListOffsprings) { this.ListOffsprings = ListOffsprings;

}

public int getindLength() { return indLength;

}

}

# ПРИЛОЖЕНИЕ Б

## Класс программы – FittnessFunction

import java.io.BufferedWriter; import java.io.FileNotFoundException; import java.io.FileReader; import java.io.FileWriter; import java.io.IOException; import java.util.Random; import java.util.Scanner;

public class FitnessFunction {

public long prepareTime = 0; public long checkTime = 0; public long sortingTime = 0;

private static final int BIT\_TO\_INT = 8; private int pathLength = 0; private int[] path = null; private int[] seq = null; private int vertexCount; private int[][] matrix;

public FitnessFunction (String filename) throws FileNotFoundException{ super();

Scanner in = new Scanner(new FileReader(filename)); this.vertexCount = in.nextInt();

this.matrix = new int[this.vertexCount][this.vertexCount]; for(int i=0;i<this.vertexCount;i++){ for(int j=0;j<this.vertexCount;j++){ this.matrix[i][j] = in.nextInt();

} } in.close(); this.pathLength = vertexCount; this.path = new int[this.pathLength]; this.seq = new int[this.pathLength];

}

@Override public long run(long[] genom) {

long old = System.currentTimeMillis(); //время

int offset=0; int vertexNumber=0; int index = 0; long tmp = 0;

for (int i=0;i<this.pathLength/BIT\_TO\_INT;i++){ offset = i<<3; tmp = genom[i]; for (int j=0;j<BIT\_TO\_INT;j++){ vertexNumber = (int)(tmp & 255); index = offset+j; this.path[index] = vertexNumber; this.seq[index] = index; tmp >>= 8;

}

}

this.prepareTime += (System.currentTimeMillis()-old); //время old = System.currentTimeMillis(); //время

qsort(this.path,this.seq,0,this.pathLength-1);

this.sortingTime += (System.currentTimeMillis()-old); //время old = System.currentTimeMillis(); //время

long pathLength = this.checkPath(this.seq);

this.checkTime += (System.currentTimeMillis()-old); //время

return (Long.MAX\_VALUE-pathLength);

}

private void quicksort(int[] ToSort, int[] ToMix,int l, int r){ int i = l; int j = r; int tmp = 0; int pivot = arrayToSort[(l+r)>>1];

while (i <= j) { while (arrayToSort[i] < pivot) {i+=1;} while (arrayToSort[j] > pivot) {j-=1;} if (i <= j) { tmp = arrayToSort[i]; arrayToSort[i] = arrayToSort[j]; arrayToSort[j] = tmp; tmp = arrayToMix[i]; arrayToMix[i] = arrayToMix[j]; arrayToMix[j] = tmp; i+=1; j-=1;

} } if (l < j){ qsort(arrayToSort, arrayToMix, l, j);

} if (i < r){ qsort(arrayToSort, arrayToMix, i, r);

}

}

public long checkPath(int[] path){

long result = 0; int pathLength = path.length; int predVertex = path[0]; int nextVertex = 0;

for (int i=1;i<pathLength;i++){ nextVertex = path[i]; result += this.matrix[predVertex][nextVertex]; predVertex = nextVertex;

}

return result;

}

public static void generateRandomFile(String filename,int n) throws

IOException{

Random random = new Random();

BufferedWriter out = new BufferedWriter(new FileWriter(filename)); out.write(n+"\n");

int[][] matrix = new int[n][n];

for(int i=0;i<n-1;i++){ for(int j=i+1;j<n;j++){ matrix[i][j] = random.nextInt(256); matrix[j][i] = matrix[i][j];

}

}

for(int i=0;i<n;i++){ for(int j=0;j<n;j++){ out.write(matrix[i][j]+" ");

} out.write("\n");

}

out.flush(); out.close();

}

public int[] getPath() { return path;

}

public int[] getSeq() { return seq;

}

}

# ПРИЛОЖЕНИЕ В

## Класс программы – Main

import java.io.FileWriter; import java.io.IOException;

public class Main {

public static void main(String[] args) throws IOException{ int num = 128;

num = 256;

FitnessFunction.generateRandomFile("matrix2.txt", num);

FitnessFunction ff = new FitnessFunction("matrix2.txt"); GeneticAlg ge = new GeneticAlg(ff); ge.setindCount(1000); ge.setgenCount(1000); ge.setselectType(GeneticAlg.selectType.TOURNIR);

ge.setcrossType(GeneticAlg.crossType.RECOMBINATION\_ELEMENTWISE);

ge.setUsemutations(true); ge.setmutationsProcent(0.02d);

long time = System.currentTimeMillis(); long[] better = ge.run(); long bestPath = Long.MAX\_VALUE-ff.run(better); long medianPath = 128 \* num;

long timeToFF = ge.timeToFF; long timeToSelection = ge.timeToselect; long timeToCrossing = ge.timeTocross; long timeToMutate = ge.timeToMutate;

System.out.println("Время работы:\t"+(System.currentTimeMillis()time)/1000 + " сек");

System.out.println("Фитнес функция:\t"+timeToFF/1000 + " сек");

System.out.println(" Подготовка:\t"+ff.prepareTime/1000 + " сек");

System.out.println(" Сортировка:\t"+ff.sortingTime/1000 + " сек");

System.out.println(" Проверка: \t"+ff.checkTime/1000 + " сек");

System.out.println("Селекция:\t"+(timeToSelection-timeToFF)/1000 +

" сек");

System.out.println("Скрещивание:\t"+timeToCrossing/1000 + " сек");

System.out.println("Мутация: \t"+timeToMutate/1000 + " сек");

System.out.println("Длина среднего пути: \t" + medianPath);

System.out.println("Длина лучшего пути: \t" + bestPath);

System.out.println("Коэффициент средний/лучший: \t" +

(float)medianPath/bestPath); for (int i = 0; i < ff.getSeq().length; i++) {

System.out.print(ff.getSeq()[i] + " ");

} System.out.println();

for (int i = 0; i < ff.getPath().length; i++) {

System.out.print(ff.getPath()[i] + " ");

} System.out.println();

System.out.println("Время работы:\t"+(System.currentTimeMillis()time)/1000 + " сек");

try(FileWriter writer = new FileWriter("history.txt", true))

{

writer.write(Integer.toString(num)); writer.write(" "); writer.write(Integer.toString(ge.getIndCount())); writer.write(" "); writer.write(Long.toString(ge.getGenCount())); writer.write(" "); writer.write(ge.getselType().toString()); writer.write(" "); writer.write(ge.getcrossType().toString()); writer.write(" "); writer.write(Boolean.toString(ge.getusemutations())); writer.write("

"); writer.write(Double.toString(ge.getmutationProcent()));

writer.write(" "); writer.write(Long.toString((System.currentTimeMillis()-

time)/1000)); writer.write(" "); writer.write(Long.toString(medianPath)); writer.write(" "); writer.write(Long.toString(bestPath)); writer.write(" "); writer.write(Float.toString((float)medianPath/bestPath));

writer.write(" ");

writer.append('\n'); writer.flush();

}

catch(IOException ex){

System.out.println(ex.getMessage());

}

}

private static void printLongInBin(long l, int last){ if (last>0){ int p = (int)(l & 1); printLongInBin(l>>1,--last);

System.out.print(p);

}

}

}