

Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет ИТМО

Эволюционные вычисления

Весна

2023

Лабораторная работа №4

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА

Цель работы

Целью данной работы является получение навыков разработки эволюционных алгоритмов для решения комбинаторных задач на примере задачи коммивояжёра.

Краткие теоретические сведения

Большое множество задач оптимизации имеют комбинаторный или дискретный характер. Одним из наиболее известных примеров является задача коммивояжёра, которая формулируется следующим образом. Существует n городов $V = \{v_i\}$ с заданными расстояниями между ними: $E = \{e_{ij}\} 1 \leq i, j \leq n$. Необходимо построить маршрут, проходящий через все города с возвратом в исходный город, с наименьшей итоговой пройденной длиной. Предполагается, что каждый город может быть посещён только один раз (за исключением исходного).

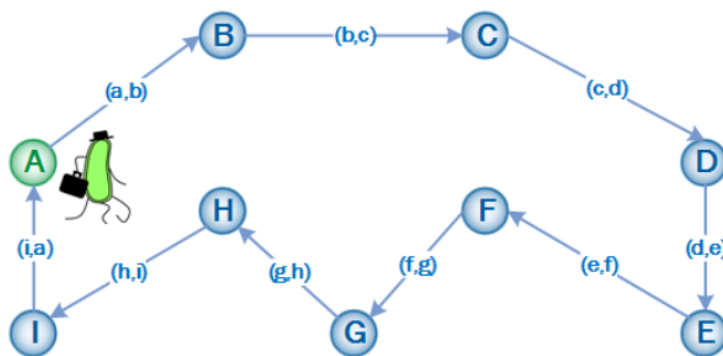


Рисунок 3.1 – Пример представления задачи коммивояжёра

Таким образом, решением задачи является перестановка городов – упорядоченный список длиной n , а результатом является полученная дистанция данного маршрута. В рамках разрабатываемого эволюционного алгоритма вид решения может быть изменён с целью более эффективных реализаций внутренних функций, а функция оценки качества решений может быть также преобразована для улучшения отличия между решениями.

ГА удобен для решения комбинаторных задач в силу того, что индивиды популяции легко представимы в комбинаторном виде, а также в силу комбинаторной природы операций мутации и кроссовера.

При решении комбинаторных задач сложность задачи резко (экспоненциально) возрастает с увеличением размерности задач. Также изменение одной переменной решения влияет и на другие. Таким образом, при решении задачи эволюционными алгоритмами следует учитывать специфику задачи, а также возможные ограничения. Например, если решение должно составлять комбинацию всех заданных уникальных объектов по одному разу, то реализация мутации в виде замены одной переменной на любой другой объект является недопустимой (рисунок 3.2).

изменение
[ABCDE] \longrightarrow [ABCDA] ✗

Рисунок 3.2 – Пример невалидной мутации для комбинаторного представления

Для реализации оператора мутации для комбинаторного представления можно выделить несколько наиболее очевидных стратегий. Предположим, что наши решения представляют из себя комбинации 9 чисел от 1 до 9. На рисунке 3.3 представлена структура решения (хромосомы). В терминологии эволюционных алгоритмов можно выделить следующие понятия. Структура хромосомы определяется определённым и упорядоченным набором аллелей. Аллель представляет собой индекс или позицию для генов. Ген представляет собой размещенный объект в позиции аллели. В данном примере, геном является любое из чисел от 1 до 9, а аллелью является индекс элемента массива.



Рисунок 3.3 – Структура хромосомы

Первый очевидный тип мутации – перестановка (**swap mutation**), где случайно выбираются две аллели по которым происходит перестановка их генов.

Вставка (**insert mutation**) – случайно выбираются два гена, затем второй помещается в позиции за первым геном.

Перемешивание (**scramble mutation**) – случайно выбирается диапазон аллелей, в рамках которого все гены перемешиваются.

Инверсия (**inversion mutation**) – случайно выбираются две аллели, между которыми все гены меняют свой порядок на противоположный.

Примеры стратегий мутации представлены на рисунке 3.4. Первые три варианта мутации представляют из себя небольшие изменения и больше подходят для локального поиска, в то время как инверсия представляет более существенное изменение.

Реализация оператора кроссовера также имеет свою специфику при работе с комбинаторными задачами. Как и в случае мутации, генерируемые кроссовером новые решения должны быть валидны – содержать все элементы в единственном числе. Рассмотрим наиболее известную стратегию кроссовера – упорядоченный кроссовер (**order crossover**). Пример для построения одного потомка на основе двух родительских решений представлен на рисунке 3.5.

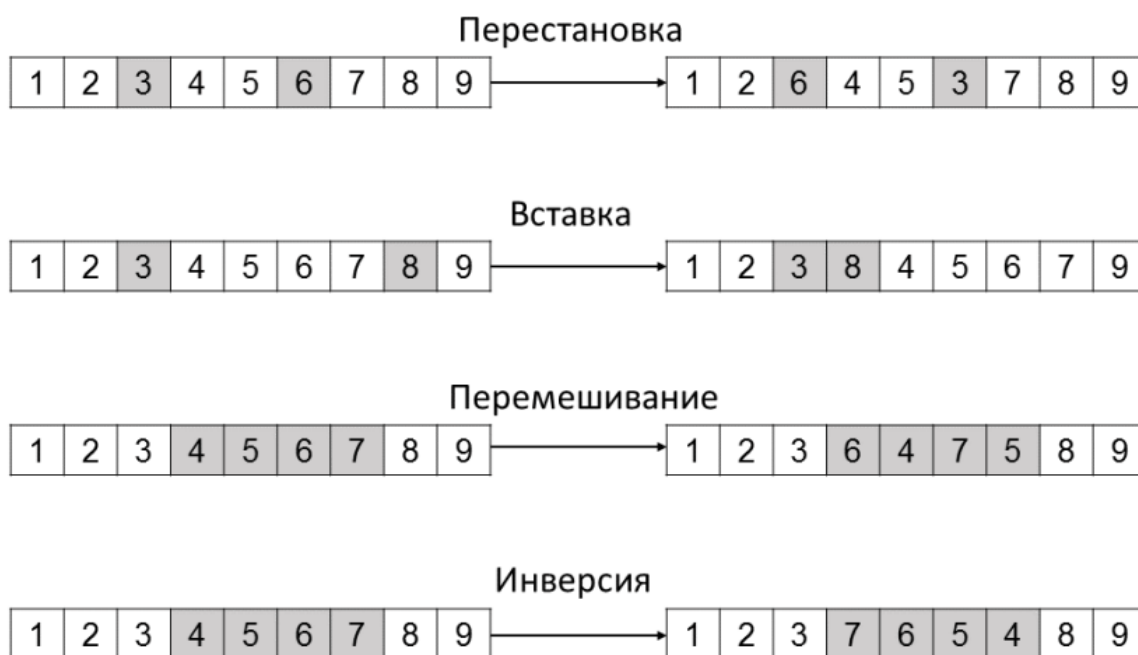


Рисунок 3.4 – Примеры мутации для комбинаторного представления

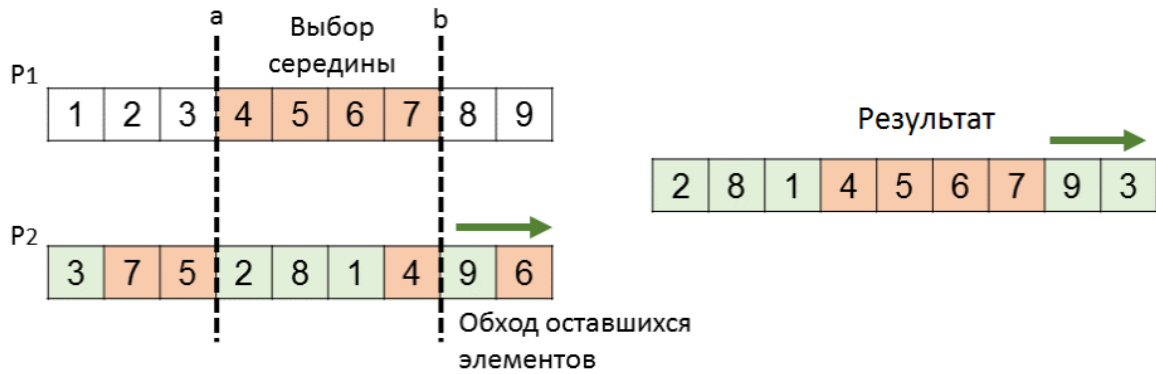


Рисунок 3.5 – Пример реализации упорядоченного кроссовера

Первый шаг алгоритма – выбор двух точек a и b . Потомок целиком наследует середину от первого родителя P_1 . Второй шаг алгоритма – обход всех незадействованных генов в порядке второго родителя P_2 , начиная от позиции b . Такая реализация является наиболее очевидной и простой. Однако существует и множество других стратегий [2], которые встречаются при решении различных задач.

<http://www.math.uwaterloo.ca/tsp/vlsi/index.html>

Ход работы

В данной лабораторной работе решается задача коммивояжёра. Данные и описание тестовых задач доступны по адресу:

<http://www.math.uwaterloo.ca/tsp/vlsi/index.html>

Для начала возьмём задачу XQF131, в которой необходимо построить оптимальный маршрут по 131 городам. В файле с данными представлены 3 столбца, где первый столбец – индексы городов, второй столбец – координата

по оси X , третий столбец – координата по оси Y . Формат данных представлен ниже в листинге 3.1.

Индекс	X	Y
1	0	13
2	0	26
3	0	27
4	0	39
5	2	0
6	5	13
7	5	19
8	5	25
9	5	31
...		

Листинг 3.1 – Формат входных данных для задачи коммивояжёра

В описании проблемы также находится и необходимый результат – оптимальный маршрут, пример которого представлен на рисунке 3.6.

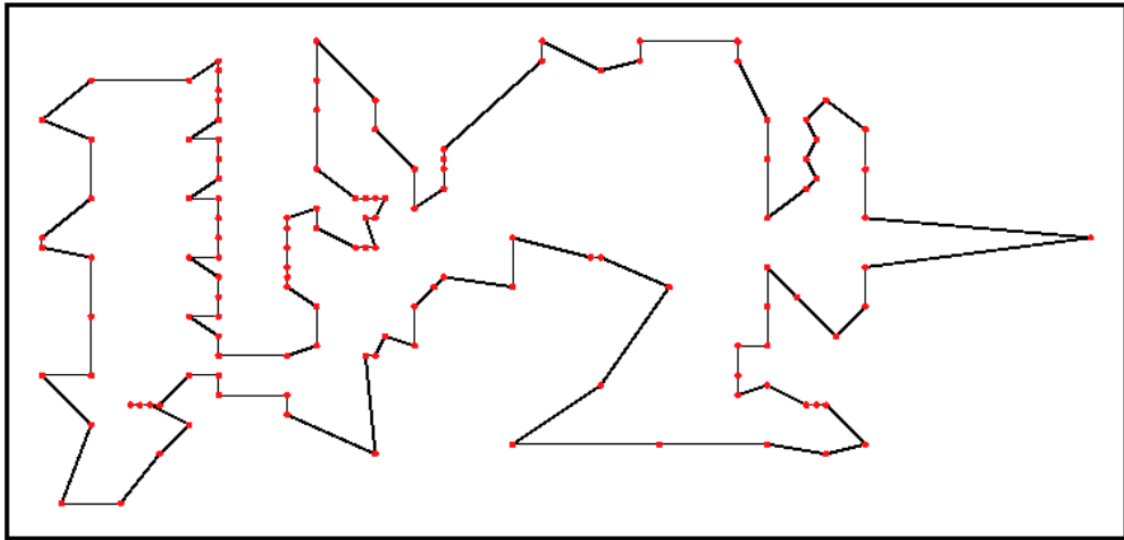


Рисунок 3.6 – Оптимальный маршрут для задачи XQF131 с длиной 564

https://gitlab.com/itmo_ec_labs/lab3

Необходимо загрузить шаблон проекта по ссылке:

https://gitlab.com/itmo_ec_labs/lab3

Шаблон также основан на фреймворке Watchmaker. Проект содержит следующие классы:

- TspAlg.java – основной класс проекта;
- TspCrossover.java – реализация оператора кроссовера;
- TspFactory.java – реализация оператора инициализации решений;
- TspFitnessFunction.java – реализация фитнес-функции;
- TspMutation.java – реализация оператора мутации;
- TspSolution.java – реализация представления решений.

Основным отличием от лабораторной работы №2 является то, что теперь необходимо реализовать представление решений в классе TspSolution.java и фитнес-функцию в классе TspFitnessFunction.java. Важно отметить, что в классе фитнес-функции, функция *isNatural()* теперь возвращает значение *false*. Это необходимо для переключения алгоритма в режим решения задачи минимизации. Для реализации фитнес-функции необходимо считать файл с входными данными проблемы (например, xqf131.tsp). Рекомендуется проводить считывание входного файла при инициализации объекта с фитнес-функцией, передав ей на вход имя проблемы или путь к файлу. При реализации представления решений рекомендуется расширить стандартную функцию *toString()* для использования при выводе прогресса эволюции алгоритма.

При реализации инициализации, кроссовера и мутации требуется ввести все необходимые параметры и настроить их для достижения лучшей производительности алгоритма. Также можно варьировать **размер популяции**, **количество итераций**, стратегию селекции (например, TournamentSelection), схему алгоритма (например, GenerationalEvolutionEngine).

Для проведения экспериментальных исследований необходимо загрузить с источника тестовых проблем несколько экземпляров задач с разной размерностью (количеством городов). Для каждой задачи необходимо настроить параметры алгоритма и провести минимум по 10 запусков. В таблице 3.1 требуется ввести усреднённые значения по результатам серий запусков.

Таблица 3.1 Результаты решения тестовых экземпляров задачи коммивояжёра

Проблема	Размер	Параметры popsize и gens	Длина маршрута	Количество итераций до сходимости	Оптимальный маршрут
XQF131	131	10; 10	600	8	564
...					

Количество итераций до сходимости может быть вычислено путём хранения лучшего решения в процессе эволюции и значения итерации, на которой это решение было получено. В отчёте необходимо описать как было реализовано представление решений, операторы мутации и кроссовера, в том числе введенные параметры и их значения при проведении экспериментов.

Вопросы

1. Можно ли определить, что полученное решение является глобальным оптимумом?
2. Можно ли допускать невалидные решения (с повторением городов). Если да, то как обрабатывать такие решение и как это повлияет на производительность алгоритма?
3. Как изменится задача, если убрать условие необходимости возврата в исходную точку маршрута?

Литература

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – 2013.
2. Eiben A. E., Smith J. Introduction to Evolutionary Computing. – 2015.
3. Тестовые экземпляры проблем для задачи коммивояжёра, <http://www.math.uwaterloo.ca/tsp/data/index.html>.

