

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
кафедра Информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе на тему

ФРЕЙМВОРК ДЛЯ РЕШЕНИЯ ОПТИМИЗАЦИОННЫХ ЗАДАЧ НА
ПРИМЕРЕ ЗАДАЧИ КОММИВОЯЖЕРА

Студент: гр. 753504 Милинкевич М.И.

Руководитель: ассистент кафедры информатики
Леченко А.В.

Минск 2019

СОДЕРЖАНИЕ

Введение.....	3
1 Задача Коммивояжера. Общее описание.....	4
1.1 Постановка задачи.....	4
1.2 Алгоритмическая сложность.....	4
2 Реализованные методы решения задачи Коммивояжера.....	5
2.1 Полный перебор.....	5
2.2 Метод ближайшего соседа	6
2.3 Метод минимального оствового дерева.....	7
2.4 Метод ветвей и границ.....	8
2.5 Генетический алгоритм.....	8
3 Реализованные методы оптимизации задачи Коммивояжера.....	11
3.1 Локальная оптимизация.....	11
3.2 Метод имитации отжига.....	12
4 Описание фреймворка.....	14
4.1 Формат файлов тестов и решений.....	14
4.2 Использование фреймворка.....	15
4.2.1 Многопоточный режим.....	16
4.2.2 Отображение найденного пути с помощью boost/python.....	17
5 Анализ полученных результатов.....	20
Заключение	22
Список использованных источников	23

Введение

Задача коммивояжера является одной из наиболее интенсивно изучаемых задач вычислительной математики. Данная курсовая работа посвящена сравнению различных алгоритмов нахождения кратчайшего маршрута для задачи коммивояжера. В качестве основного языка для реализации различных решений был выбран C++.

1 ЗАДАЧА КОММИВОЯЖЕРА. ОБЩЕЕ ОПИСАНИЕ

1.2 Постановка задачи

Задача коммивояжера может иметь различные формулировки. Например, есть множество городов и путешественник. Путешественник должен посетить каждый из городов ровно один раз и вернуться в первоначальный город, при этом он должен свести к минимуму суммарное пройденное расстояние. Другими словами, необходимо найти Гамильтонов цикл минимального веса во взвешенном полном (не всегда) графе.

Задача коммивояжера является NP-полной.

1.2 Алгоритмическая сложность

Полный перебор состоит в том, чтобы оценить все возможные маршруты и вернуть лучший. Так как есть $(n - 1)!$ возможных маршрутов, то эта стратегия занимает $O(N!)$.

2 РЕАЛИЗОВАННЫЕ МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЕРА

2.1 Полный перебор

В рамках данной курсовой работы был реализован полный перебор с отсечением по времени.

Порядок решения задачи с помощью полного перебора без отсечения по времени:

- 1) Рассматриваем вершину с индексом ноль в качестве начальной и конечной точки.
- 2) Генерируем $(N - 1)!$ перестановку городов.
- 3) Рассчитываем стоимость для каждой перестановки и поддерживаем минимальную стоимость пути.
- 4) Полученная перестановка с минимальной стоимостью — ответ.

Отличие полного перебора с отсечением по времени от описанного выше заключается в том, что первый будет рассматривать только первые k перестановок, которые пройдут по времени и выбирать лучший маршрут из них. Это говорит о том, что на больших данных решение будет давать приближённый ответ.

На рисунке 2.1 фиолетовым цветом изображен оптимальный маршрут, а зеленым — найденный полным перебором за 50 секунд.

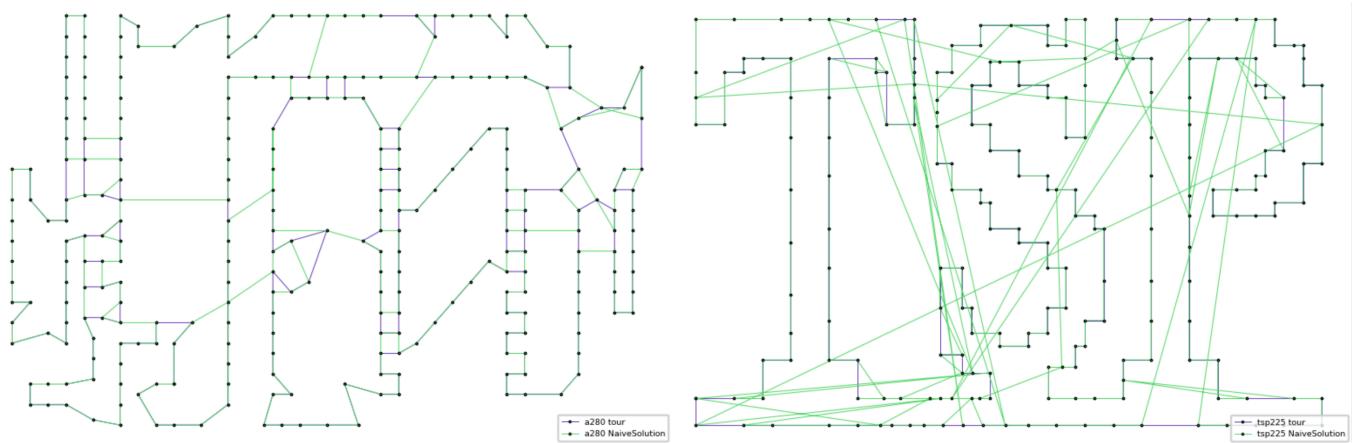


Рисунок 2.1 Решение задачи коммивояжера с помощью полного перебора с отсечением по времени

2.2 Метод ближайшего соседа

Суть метода ближайшего соседа заключается в том, что вершины обхода графа последовательно включаются в маршрут, причем каждая очередная включаемая вершина должна быть ближайшей к последней выбранной среди всех остальных, ещё не включенных в состав маршрута.

Метод ближайшего соседа является “жадным” алгоритмом и может выдавать неоптимальные маршруты.

Порядок решения задачи с помощью метода ближайшего соседа:

- 1) Инициализируем все вершины как не посещенные.
- 2) Выбираем произвольную вершину, устанавливаем ее в качестве текущей вершины u . Отмечаем посещение данной вершины.
- 3) Находим ребро минимальной длины, соединяющее текущую вершину u и непосещенную вершину v .
- 4) Устанавливаем v в качестве текущей вершины u . Отмечаем посещение вершины v .
- 5) Если все вершины посещены, то завершаем алгоритм. В противном случае переходим к шагу 3.

Последовательность посещенных вершин является маршрутом для задачи коммивояжера. Алгоритмическая сложность данного метода составляет $O(N^2)$.

Пример работы метода ближайшего соседа можно видеть на рисунке 2.2

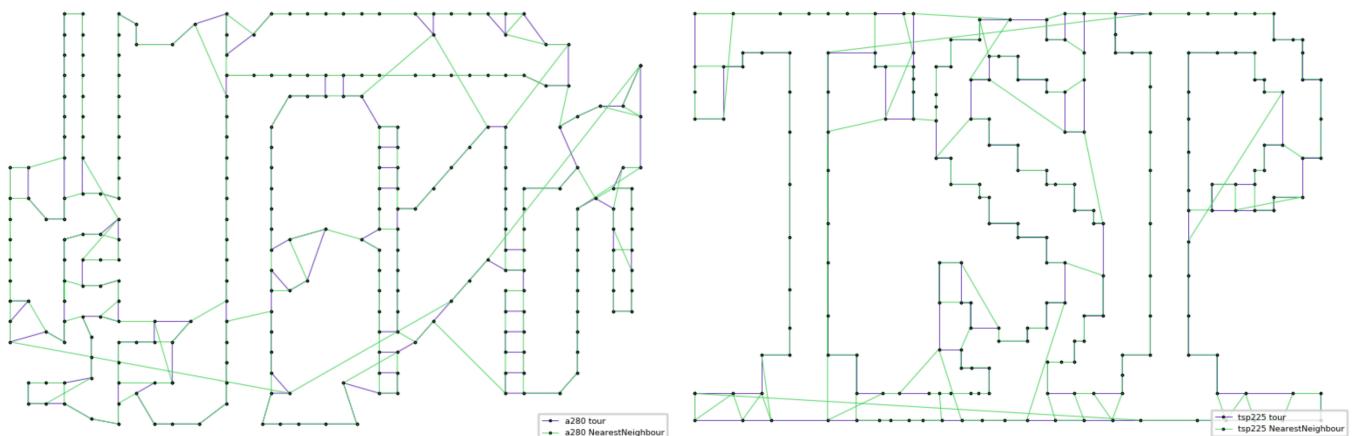


Рисунок 2.2 Решение задачи коммивояжера с помощью метода ближайшего соседа

2.3 Метод минимального оствовного дерева

Метод минимального оствовного дерева, в отличие от описанных ранее методов, имеет гарантированную точность решения $r = 2$. То есть данный алгоритм в худшем случае найдет решение в два раза большее, чем оптимальное.

Суть метода заключается в следующем:

Для полного взвешенного графа задачи коммивояжера построим оствовное дерево A_{mst} минимального веса. Заметим, что $A_{mst}(I) \leq Opt(I)$ для любого примера, так как удаление ребра из оптимального решения задачи коммивояжера дает оствовное дерево. Удвоим каждое ребро в полученном оствовном дереве. Получим эйлеров граф, каждая вершина имеет четную степень. Построим произвольный эйлеров цикл.

Перестроим его так, чтобы получился гамильтонов цикл. Начиная с произвольной вершины, двигаемся вдоль эйлерова цикла и помечаем вершины. Если очередная вершина уже помечена, то пропускаем ее и двигаемся дальше, пока не найдем непомеченную вершину или не вернемся в первую вершину. Цепочку дуг для помеченных вершин заменяем прямой дугой в непомеченную или первую вершину.

В данной курсовой работе был реализован алгоритм Борувки для поиска минимального оствовного дерева в графе, а для поиска оптимального маршрута был использован поиск в глубину.

На рисунке 2.3 фиолетовым цветом изображен оптимальный маршрут, а зеленым — найденный методом минимального оствовного дерева.

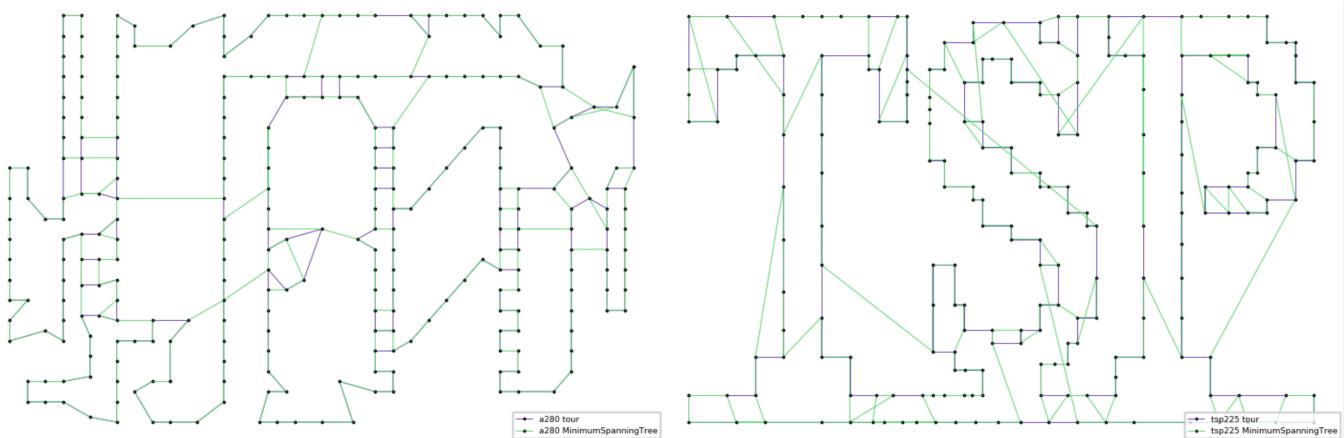


Рисунок 2.3 Решение задачи коммивояжера с помощью метода минимального оствовного дерева

2.4 Метод ветвей и границ

Метод ветвей и границ является развитием метода полного перебора, однако в отличие от него, он отсекает подмножества допустимых решений, которые заведомо не содержат оптимальный маршрут.

Для задачи коммивояжера данный метод представляет из себя рекурсивный алгоритм, в котором перебираются возможные ветви решения и отсекаются те, которые не удовлетворяют следующему неравенству:

$$\text{current_weight} + \Omega > \text{best_weight} \quad (2.1)$$

В формуле 2.1 current_weight — это текущая стоимость пути для ветви решения, best_weight — наилучшая найденная стоимость пути, а Ω — нижняя оценка оптимального решения, которая вычисляется эвристически. Хорошой эвристикой является сумма весов минимального ребра по всем вершинам, не входящим в текущий путь.

На рисунке 2.4 фиолетовым цветом изображен оптимальный маршрут, а зеленым — найденный методом ветвей и границ.

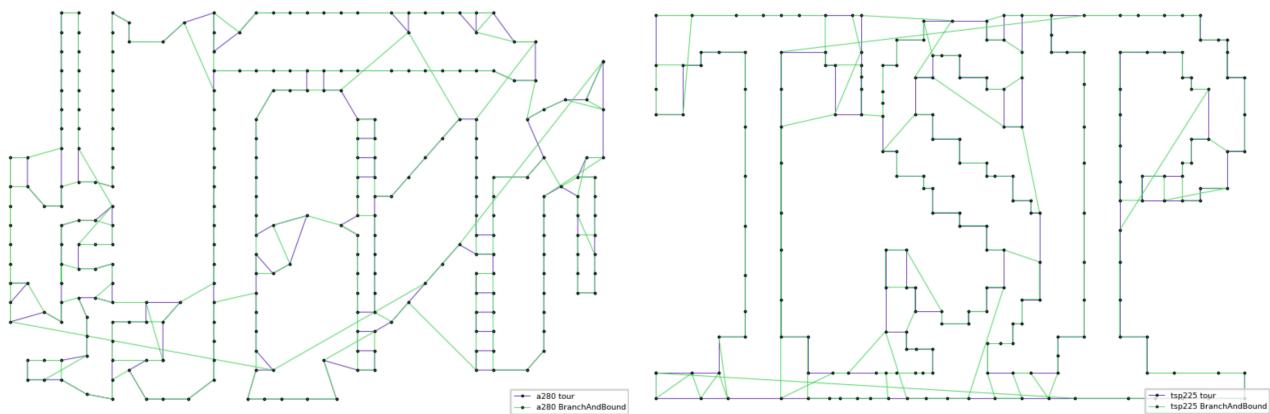


Рисунок 2.4 Решение задачи коммивояжера с помощью метода ветвей и границ

2.5 Генетический алгоритм

Стандартный генетический алгоритм начинает свою работу с формирования начальной популяции — конечного набора допустимых решений задачи. Эти решения могут быть выбраны случайным образом или получены с помощью конструктивных алгоритмов. Выбор начальной популяции не имеет значения для сходимости алгоритма, однако формирование "хорошей" начальной популяции (например, из множества локальных оптимумов) может заметно сократить время достижения глобального оптимума.

На каждом шаге эволюции с помощью вероятностного оператора селекции выбираются два решения, родители s_1, s_2 . Оператор скрещивания по этим решениям строит новое решение s , которое затем подвергается небольшим случайным модификациям — мутациям. Затем решение добавляется в популяцию, а решение с наименьшим значением функции приспособленности удаляется из популяции.

Порядок решения задачи с помощью генетического алгоритма:

1. Выбрать начальную популяцию.
2. Повторять, пока не выполнен критерий остановки.
 - 2.1. Выбрать родителей из популяции.
 - 2.2. Построить новое решение по решениям .
 - 2.3. Модифицировать .
 - 2.4. Обновить популяцию.
3. Предъявить наилучшее решение в популяции.

На рисунке 2.5 фиолетовым цветом изображен оптимальный маршрут, а зеленым — найденный генетическим алгоритмом.

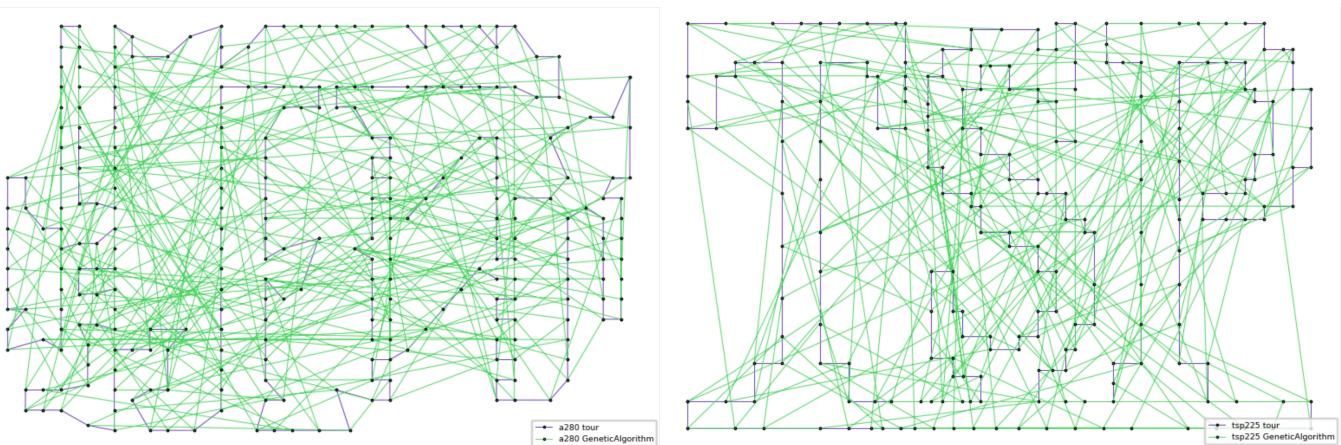


Рисунок 2.5 Решение задачи коммивояжера с помощью генетического алгоритма

Генетический алгоритм сам по себе работает хорошо только если у него достаточно времени для того, чтобы найти оптимальное решение. Графики сходимости для генетического алгоритма (на разных тестовых примерах), реализованного в рамках данной курсовой работы можно видеть на рисунках 2.6-2.8.

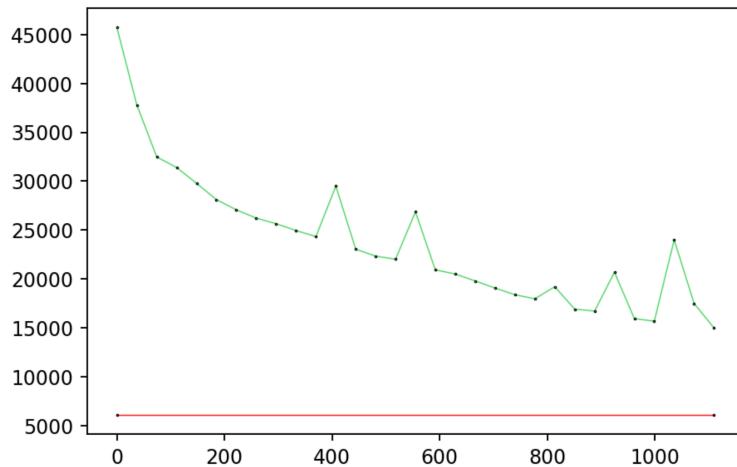


Рисунок 2.6 График сходимости генетического алгоритма (ch130)

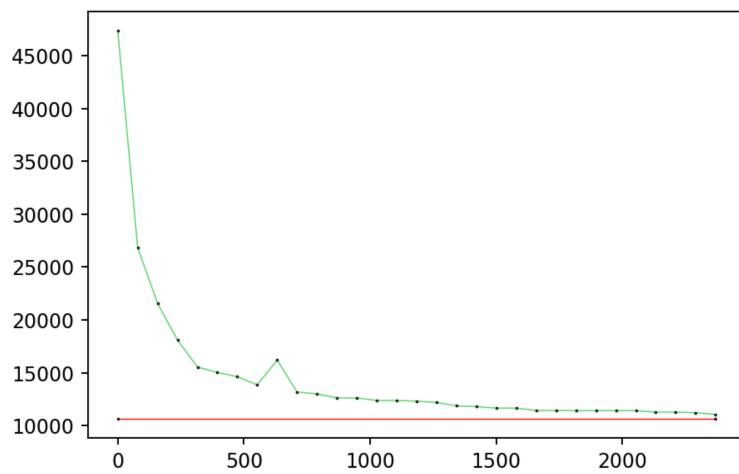


Рисунок 2.7 График сходимости генетического алгоритма (att48)

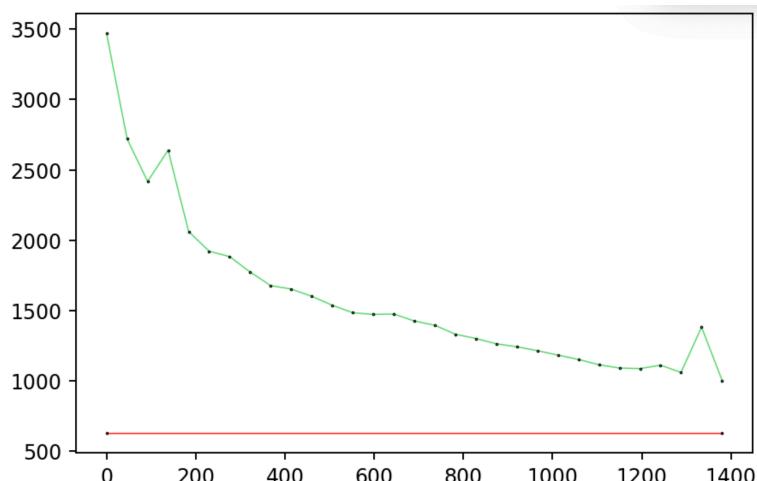


Рисунок 2.8 График сходимости генетического алгоритма (eil101)

3 РЕАЛИЗОВАННЫЕ МЕТОДЫ ОПТИМИЗАЦИИ ЗАДАЧИ КОММИВОЯЖЕРА

3.1 Локальная оптимизация

Пусть у нас есть множество решений задачи и целевая функция, минимум которой требуется найти на множестве решений. В задаче коммивояжера в качестве целевой функции, значение которой необходимо минимизировать выступает стоимость пути.

Локальная оптимизация начинает свою работу с некоторого начального решения, которое может быть выбрано случайно или найдено с помощью какого-либо алгоритма. На каждом шаге алгоритм изменяет текущее решение на соседнее, значение целевой функции которого меньше.

В качестве функции перехода от старого решения к новому с меньшим значением целевой функции может служить следующий алгоритм:

1) Рассматриваем некоторое начальное решение (путь) в качестве множества точек.

2) Перебираем левую и правую границу.

3) Смотрим, выполняется ли следующее соотношение:

$Distance(prev, r) + Distance(l, next) < Distance(prev, l) + Distance(r, next)$, где $prev$ — это предыдущая вершина, а $next$ — следующая за правой границей вершина.

Если данное соотношение выполняется, то выполняем реверсию отрезка пути с l по r и обновляем значение целевой функции.

4) Если минимум был найден, прекращаем алгоритм, иначе переходим к пункту 2.

На рисунках 3.1-3.2 изображены пути, найденные различными методами решения с использованием локальной оптимизации.

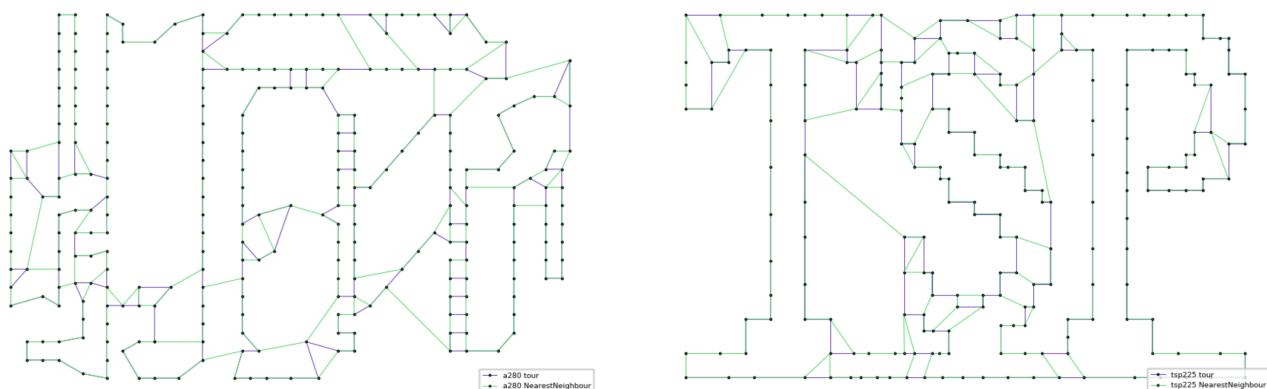


Рисунок 3.1 Решение задачи коммивояжера методом ближайшего соседа с локальной оптимизацией

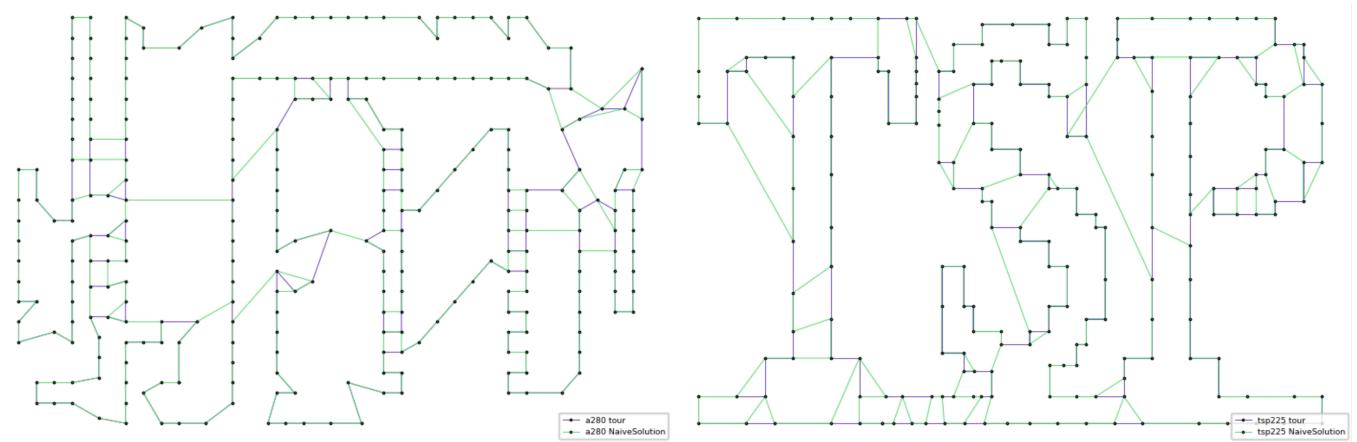


Рисунок 3.2 Решение задачи коммивояжера с помощью полного перебора и локальной оптимизацией

3.2 Метод имитации отжига

Метод имитации отжига является общим алгоритмическим методом для решения различных задач глобальной оптимизации, особенно дискретной и комбинаторной оптимизации.

Схема работы алгоритма выглядит следующим образом:

- 1) Положить $k = 0, T_k = 100$
- 2) Выбрать случайный элемент $s(0)$ из области допустимых решений (из множества S).
- 3) Снизить температуру по правилу: $T_{k+1} = \alpha T_k$
- 4) Построить новый элемент $s(k + 1) = g(s(k))$ по некоторому случайному алгоритму. Обычно предполагается, что этот алгоритм работает так, что каждое последующее приближение должно отличаться не очень сильно.
- 5) Вычисляем $dh = h(s(k + 1)) - h(s(k))$.
- 6) Если $dh < 0$, т.е. найденное приближение лучше чем было, то принять $s(k + 1)$.
- 7) Если $dh \geq 0$, тогда принять решение $s(k + 1)$ с вероятностью: $P(dh) = e^{\frac{-dh}{T_k}}$. Если решение не принимается, то положить $s(k + 1) = s(k)$.
- 8) Положить $k = k + 1$. Перейти к шагу 3.

На рисунках 3.3-3.5 изображены пути, найденные различными методами решения с использованием метода имитации отжига для оптимизации.

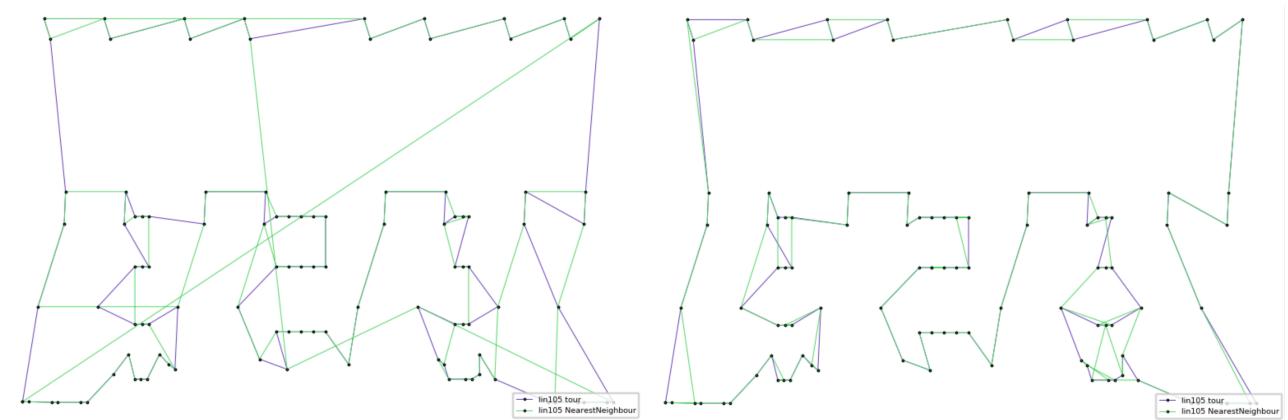


Рисунок 3.3 Решение задачи коммивояжера с помощью метода ближайшего соседа и оптимизации методом отжига

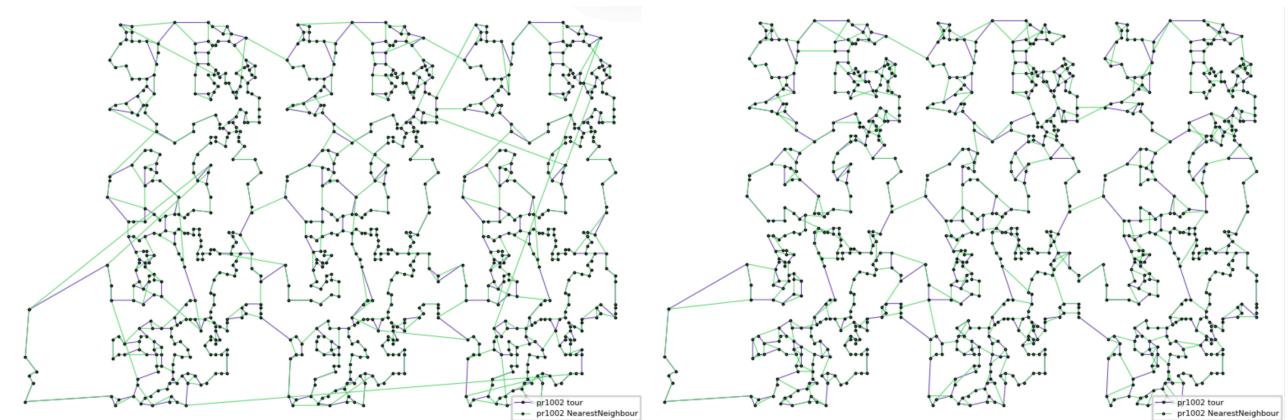


Рисунок 3.4 Решение задачи коммивояжера с помощью метода ближайшего соседа и оптимизации методом отжига

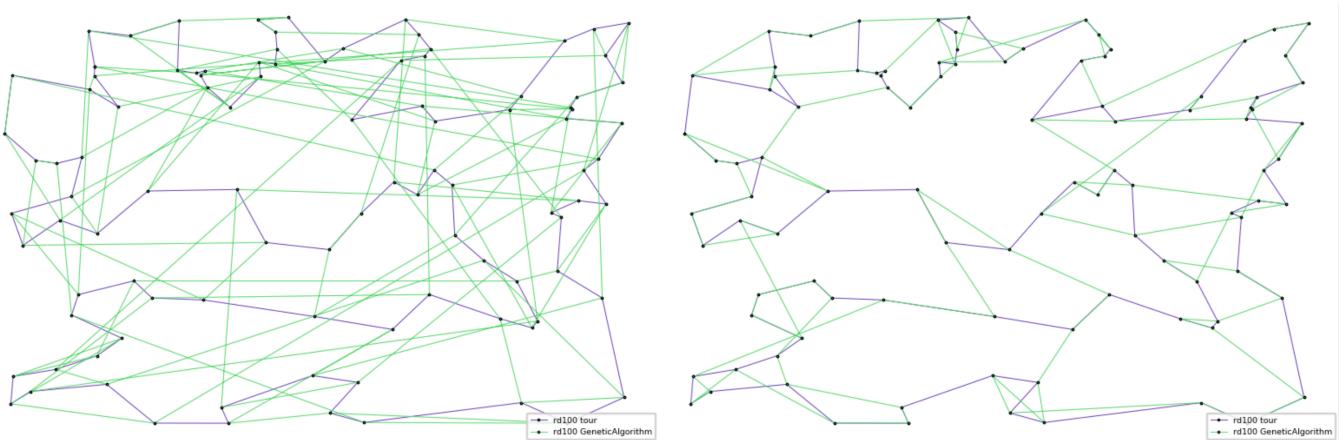


Рисунок 3.5 Решение задачи коммивояжера с помощью генетического метода и оптимизации методом отжига

4 ОПИСАНИЕ ФРЕЙМВОРКА

4.1 Формат файлов тестов и решений

В рамках данной курсовой работы были реализованы все решение и оптимизации, описанные ранее в разделах 2 и 3. Все классы решений и оптимизаций находятся в папке “algo/“.

Для того, чтобы тестовый пример можно было запустить на выполнение, необходимо, чтобы формат файла теста был следующим:

```
1 NAME : test name
2 COMMENT : additional info about test
3 TYPE : TSP
4 DIMENSION : vertex amount (type: int)
5 EDGE_WEIGHT_TYPE : one from EUC_2D / GEO / ATT / EXPLICIT / CEIL_2D
6 EDGE_WEIGHT_FORMAT: FULL_MATRIX/UPPER_DIAG_ROW/LOWER_DIAG_ROW/UPPER_ROW
7 NODE_COORD_SECTION/EDGE_WEIGHT_SECTION
8 ...here coords or matrix..
```

Рисунок 4.1 Формат теста для задачи Коммивояжера

Также имя тестового файла должно быть в формате ‘*testname.tsp*’. В зависимости от того, какой *EDGE_WEIGHT_TYPE* указан в teste, будет выбрана соответствующая функция для вычисления расстояния между вершинами (если не была дана матрица, т.е. был указан *EXPLICIT*). Более подробно о том, как вычисляется расстояние можно найти в источнике [2] (стр.6). В проекте реализация данных функций находится в файле “algo/Distance.cpp”.

Каждое найденное фреймворком решение записывается в файл с названием имеющим формат: <testname>_<solution_name>_<version>.tour и сохраняются в папку “results/“. Формат файла решения имеет схожий с тестом формат, который показан на рисунке 4.2

Инкремент версии происходит для каждого решения с каждым его запуском. Формат файла, содержащего решение:

```
1 NAME : test name
2 TYPE: TOUR
3 DIMENSION : vertex amount (type: int)
4 WEIGHT: best cost found by solution
5 TOUR_SECTION
6 ...here vertex indices..
```

Рисунок 4.2 Формат файла, содеоржащего решение задачи Коммивояжера

4.2 Использование фреймворка

Фреймворк, по сути, является консольным приложением и поэтому все возможные режимы работы необходимо указывать через командную строку. В качестве решения для корректной обработки параметров командной строки была использована сторонняя библиотека *cxxopts*, которая находится в свободном доступе и содержит инструкцию по установке. Ссылку на исходные файлы и документацию к данной библиотеке можно найти в источнике [6].

У фреймворка есть несколько параметров командной строки, которые необходимо возможно для корректной работы. Все они описаны в таблице 4.1.

Параметр	Возможные значения	Описание
--mode	run-solution, list-optimizers, list-solutions	режимы фреймворка
--solution-name	NaiveSolution, NearestNeighbour, MinimumSpanningTree, BranchAndBound, GeneticAlgorithm или их сокращения NS, NN, MST, BAB, GA	название решения default = all
--solution-deadline	1000	дедлайн для решения в миллисекундах, default = 3000. для решений с отсечением по времени
--thread-count	8	количество потоков для многопоточного режима, default = 1
--testname	a280, att48, ali535 и т.д. из папки datasets/	название теста default = all
--optimizer-name	LocalSearch, SimulatedAnnealing	название оптимизатора
--optimizer-deadline	3000	дедлайн в миллисекундах default = 3000
--comment		комментарий к решению
--save-convergence		для метода отжига и генетического метода возможность отслеживать сходимость метода

Таблица 4.1 Параметры командной строки для использования фреймворка

Таким образом, если программа была запущена с помощью набора команд, указанного на рисунке 4.3, то программа запустит на выполнение генетический алгоритм с оптимизацией методом имитации отжига; дедлайн для решения будет равен трем секундам, а для оптимизации — десяти секундам; решение запустится на teste с названием *st70*.

```

1 | /tsp --mode run-solution
2 | --solution-name GeneticAlgorithm
3 | --test-name st70
4 | --optimizer-name SimulatedAnnealing
5 | --optimizer-deadline 10000

```

Рисунок 4.3 Пример набора команд для запуска фреймворка

4.2.1 Многопоточный режим

Многопоточный режим в последней версиии фреймворка доступен для полного перебора и генетического алгоритма.

Многопоточный режим реализован с помощью класса *ThreadPool*, в котором есть метод *enqueue*, который в качестве параметра принимает лямбда-функцию — задачу для потока.

На рисунках 4.4-4.5 слева решение, которое было найдено генетическим алгоритмом без многопоточного режима, а справа — с использованием многопоточного режима. На рисунке 4.6 все то же, но уже для метода полного перебора решения задачи коммивояжера.

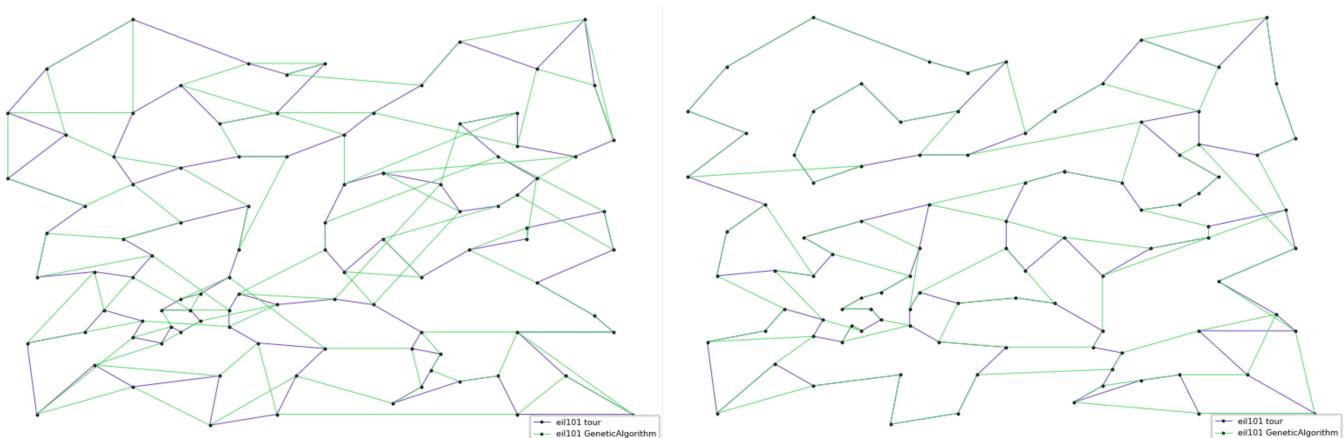


Рисунок 4.4 Многопоточный режим для генетического метода

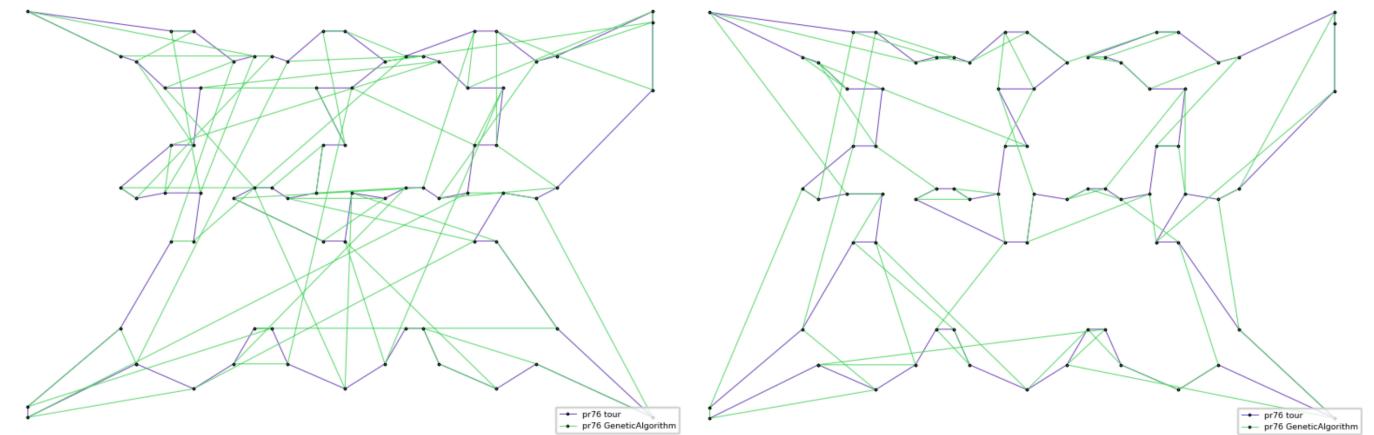


Рисунок 4.5 Многопоточный режим для генетического метода

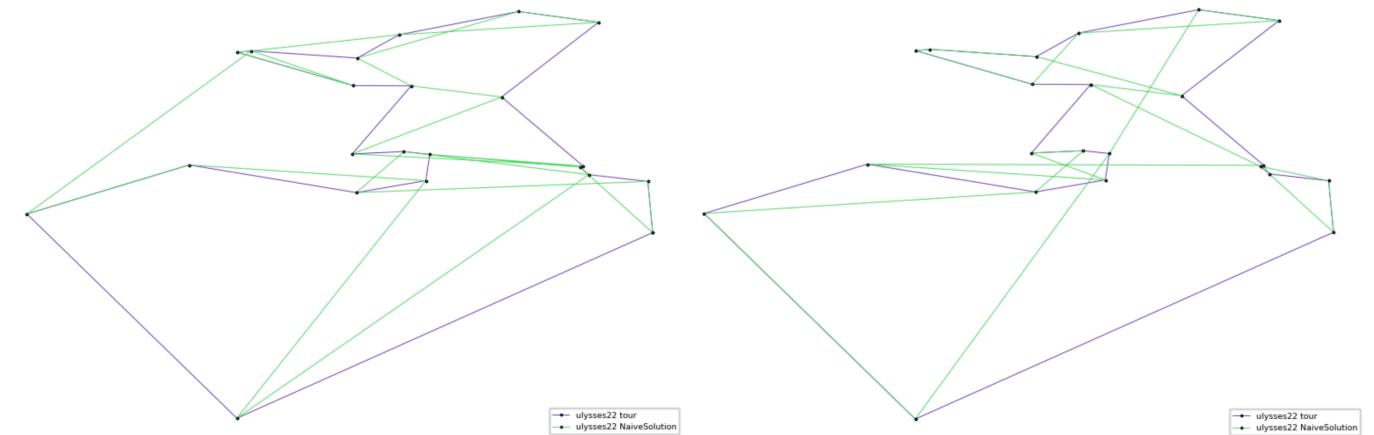


Рисунок 4.6 Многопоточный режим для полного перебора

4.2.2 Отображение найденного пути с помощью boost/python

Если на выполнение был запущен один тест, то результирующий путь выводится на экран в виде графа. Данная возможность была реализована с помощью сторонней библиотеки *boost/python* для C++. Данная библиотека позволяет выполнять команды языка *python* с помощью функции PyRun_SimpleString. Таким образом, выполняя команды, указанные на рисунке 4.7 выполняется код на языке *python*, который сохраняет обрисованные с помощью *matplotlib* график в файл с *fig.png*, который лежит в корневой директории. Затем с помощью функций библиотеки *gtkmm* данный файл отображается в новом окне. На рисунках 4.8-4.9 показан результат работы генетического метода с использованием локальной оптимизации для тестов

ali535 и *gr96*. В данном окне также отображается найденный и оптимальный вес для данного теста.

```
Py_Initialize();
PyRun_SimpleString( s: "import signal");
PyRun_SimpleString( s: "signal.signal(signal.SIGINT, signal.SIG_DFL)");

PyRun_SimpleString( s: "import matplotlib\n"
                     "matplotlib.use('GTK3Agg')");
PyRun_SimpleString( s: "import matplotlib.pyplot as plt");
PyRun_SimpleString( s: "fig, ax = plt.subplots(dpi=170)");
if (results.size() == 1) {
    auto tour = results[0].tour;
    std::string new_x, new_y;

    auto test = tour.GetTest();
    for (int i = 0; i <= tour.path.size(); i++) {
        new_x += std::to_string( val: test.GetPoint( index: tour.path[i % tour.path.size()].x ) + "," );
        new_y += std::to_string( val: test.GetPoint( index: tour.path[i % tour.path.size()].y ) + "," );
    }

    std::string plot_cmd = "plt.plot([" + new_x + "] + ", [" + new_y + "], "
                           "linewidth=0.4, marker='.', markersize=1, "
                           "color=\"#673ab7\",markeredgecolor=\"black\")";
    PyRun_SimpleString( s: plot_cmd.c_str());
    PyRun_SimpleString( s: "plt.savefig('fig.png')");
}
```

Рисунок 4.7 Сохранение графа результирующего пути в файл

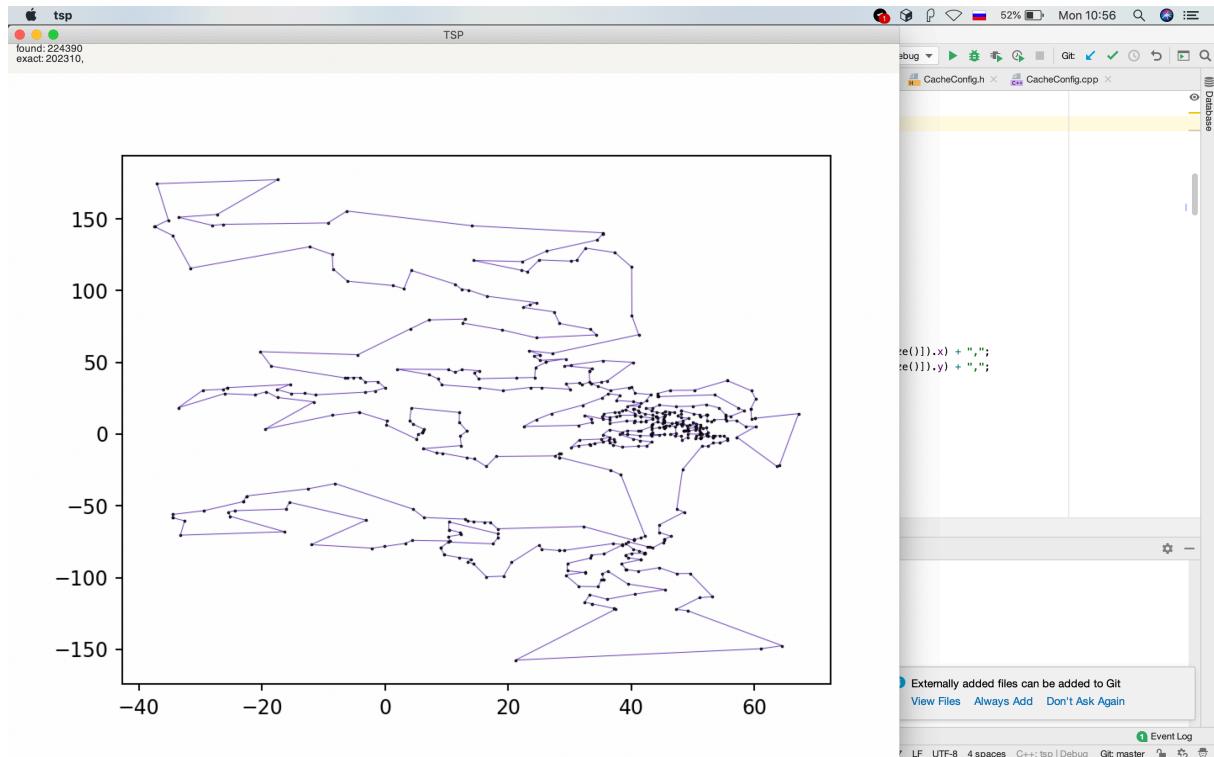


Рисунок 4.8 Результирующий путь для теста *ali535*

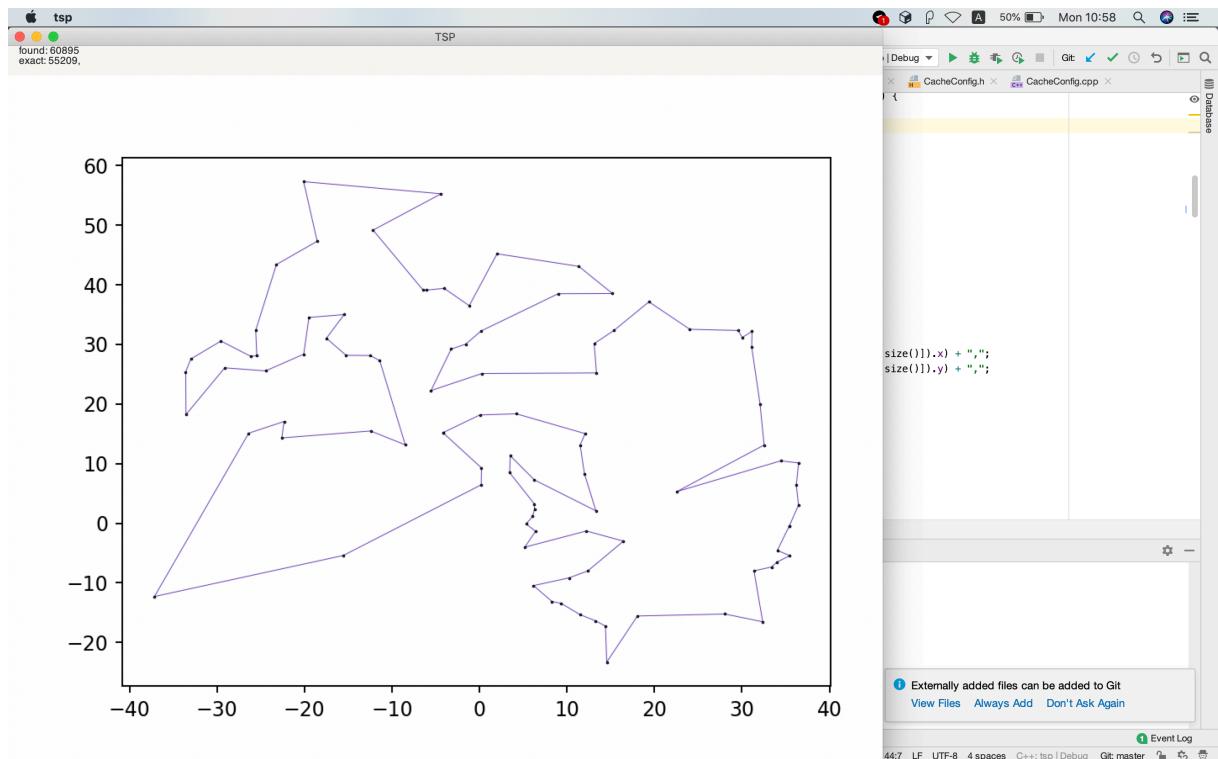


Рисунок 4.9 Результатирующий путь для теста *gr96*

5 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В рамках данной курсовой был произведен сравнительный анализ для различных методов решений задачи коммивояжера.

Так как все тестовые примеры имеют правильные ответы или отрезки, в которые ответ на данный тест входит, то стало возможным посчитать коэффициент, который показывает насколько близкое решение к верному находит каждый из реализованных алгоритмов. Данный коэффициент вычисляется по формуле 5.1.

$$c = (found - tour)/tour \quad (5.1)$$

В формуле 5.1 *found* — это решение, которое было найдено тем или иным методом, *tour* — это оптимальный ответ.

Так как результаты решений без применения оптимизаций и с ними могут сильно разнится, то отдельно сравниваются пары (решение, оптимизация). Таким образом, в таблице 5.1 содержится “рейтинг” решений и оптимизаций.

В таблице 5.1 также показано минимальное, среднее и максимальное время работы для каждой из комбинаций “метод + оптимизация”.

	solution	optimizer	score	test_run	time_ms_min	time_ms_max	time_ms_avg
0	NearestNeighbour	SimulatedAnnealing	0.154334	fl417, si175, si1032, kroE100, u159, si535, rl...	3028.0	20168.0	10344.777778
1	GeneticAlgorithm	SimulatedAnnealing	0.494020	si1032, kroE100, u159, si535, att48, rat99, rd...	14051.0	61619.0	30091.700000
2	NaiveSolution	SimulatedAnnealing	0.714925	fl417, si175, kroE100, u159, si535, brg180, ra...	2500.0	10009.0	7301.400000
3	NearestNeighbour	LocalSearch	5.775072	fl417, si175, si1032, kroE100, u159, si535, gi...	5000.0	1250150.0	20098.432432
4	NearestNeighbour		5.895183	fl417, si175, si1032, kroE100, u159, si535, gi...	0.0	1246060.0	14945.846847

	solution	optimizer	score	test_run	time_ms_min	time_ms_max	time_ms_avg
5	MinimumSpanningTree	LocalSearch	42.720179	f1417, si175, si1032, kroE100, u159, si535, gi...	5000.0	146814.0	10459.972973
6	NaiveSolution	LocalSearch	47.041753	f1417, si175, si1032, kroE100, u159, si535, gi...	8000.0	8154.0	8039.684685
7	MinimumSpanningTree	SimulatedAnnealing	48.353020	f1417, si175, si1032, kroE100, u159, si535, gi...	5000.0	146475.0	10435.234234
8	MinimumSpanningTree		53.771919	f1417, si175, si1032, kroE100, u159, si535, gi...	0.0	154276.0	5805.171171
9	NaiveSolution		68.784407	f1417, si175, si1032, kroE100, u159, si535, gi...	3000.0	30001.0	3439.216216
10	BranchAndBound	SimulatedAnnealing	839.045712	f1417, si175, si1032, kroE100, u159, si535, gi...	5010.0	41683.0	10329.000000
11	GeneticAlgorithm	LocalSearch	867.813422	f1417, si175, si1032, kroE100, u159, si535, gi...	10002.0	19762.0	10425.018018
12	BranchAndBound	LocalSearch	877.197034	f1417, si175, si1032, kroE100, u159, si535, gi...	5012.0	44529.0	11986.711712
13	BranchAndBound		906.055804	f1417, si175, si1032, kroE100, u159, si535, gi...	8.0	35038.0	5188.018018
14	GeneticAlgorithm		912.351336	f1417, si175, si1032, kroE100, u159, si535, gi...	3004.0	22300.0	4116.297297

ЗАКЛЮЧЕНИЕ

Задача коммивояжера, пожалуй, самая известная задача комбинаторной оптимизации, которая интенсивно изучается исследователями. В рамках данной курсовой работы были изучены и реализованы различные методы решения и оптимизации задачи коммивояжера. Также был проведен сравнительный анализ качества и времени работы изученных алгоритмов. Удалось выяснить, что многие методы решений в совокупности с локальной оптимизацией или оптимизацией метода отжига способны найти точный ответ даже на тестах с довольно большим количеством вершин.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] The Traveling Salesman Problem [Электронный ресурс]. – Режим доступа : <https://www.cs.d.uoc.gr/~hy583/papers/ch11.pdf>
- [2] TSPLIB 95 [Электронный ресурс]. – Режим доступа : <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf>
- [3] Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation [Электронный ресурс]. – Режим доступа : <http://user.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf>
- [4] Travelling Salesman Problem [Электронный ресурс]. – Режим доступа : <http://www.permutationcity.co.uk/projects/mutants/tsp.html>
- [5] Wikipedia [Электронный ресурс]. – Режим доступа : https://en.wikipedia.org/wiki/Travelling_salesman_problem
- [6] Github [Электронный ресурс]. – Режим доступа : <https://github.com/jarro2783/cxxopts>