Санкт–Петербургский государственный университет

РУЩЕНКО Влада Владиславовна

Выпускная квалификационная работа

Разработка алгоритма упорядочивания наложенных контуров

Уровень образования: бакалавриат
Направление 02.03.02
«Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2017
«Программирование и информационные технологии»
Профиль «Автоматизация научных исследований»

Научный руководитель:
доцент, кафедра компьютерного моделирования
и многопроцессорных систем,
к.ф. - м.н. Корхов Владимир Владиславович

Рецензент:
ведущий инженер, ООО «Роберт Бош»,
Круглов Тимофей Владиславович

Санкт-Петербург
2021 г.

Содержание

Введение

В современном мире нейронные сети имеют обширную область практического применения. Одной из таких областей является промышленный дизайн. Несмотря на то, что нейронные сети могут справиться со многими различными задачами, часто результаты их работы нуждаются в анализе и дополнительной обработке специалистами прикладной области. Примером дополнительной обработки является преобразование точечных контуров, сгенерированных нейронными сетями, в векторизованный CAD-формат. Как правило, необходимо объединить несколько «одинаковых» контуров, отличающихся шумом порядка размера пикселя. Последовательное объединение двух контуров даёт два контурных витка, которые необходимо преобразовать в один. При этом простейшие алгоритмы поиска ближайших соседей могут пропустить часть точек (рис. 1), поэтому появилась необходимость в разработке алгоритма, упорядочивающего два наложенных контура в один без пропуска точек. В данной работе описывается несколько решений поставленной задачи.

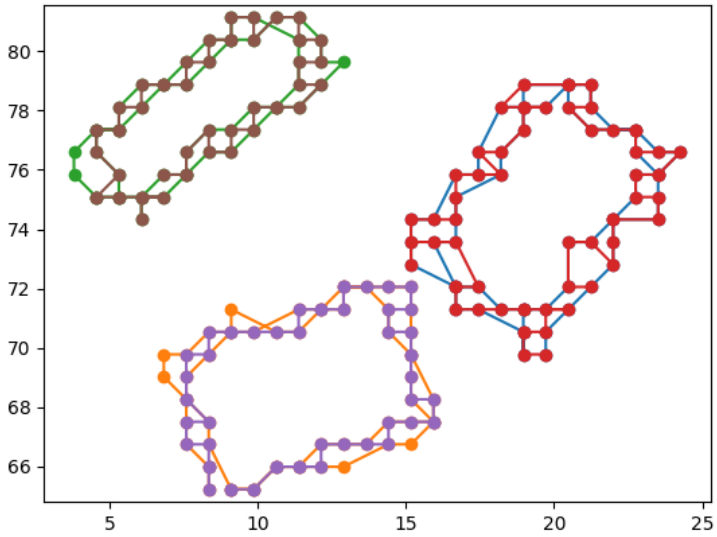

Рис. 1: Примеры корректной и некорректной работы поиска ближайшего соседа

На рис. 1 в крайней справа фигуре синим цветом обозначен исходный двойной контур, красным — одинарный. Здесь поиск ближайшего соседа

работает корректно, однако в нижней фигуре, где оранжевым цветом выделен исходный контур, а фиолетовым — одинарный, алгоритм пропускает 5 точек из исходного контура.

Постановка задачи

Целью данной работы является создание алгоритма, решающего задачу построения одинарного контура по заданному двойному, и его программной реализации. Входные данные представлены в виде набора $n \in [128, 512]$ точек, имеющих координаты (x_i, y_i), $i = 1, \ldots, n$ и описывающих двойной контур (см. рис. 2). На выходе программа должна выдавать упорядоченный набор из всех входных точек, представляющий из себя одинарный контур без самопересечений.

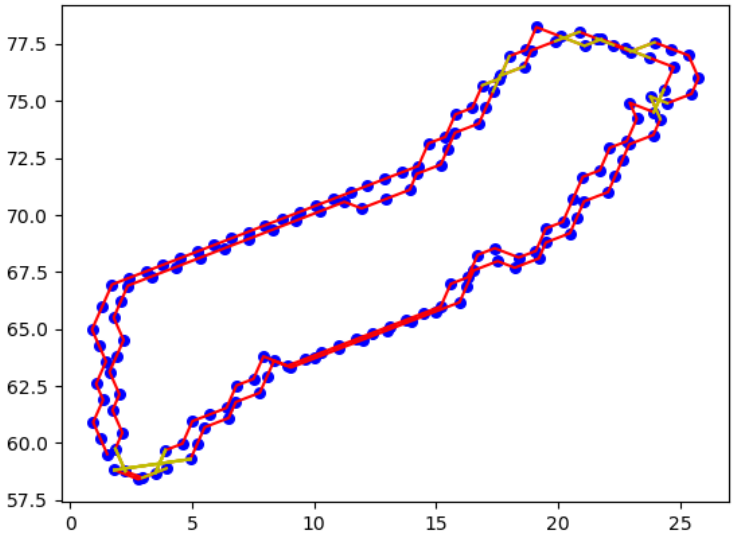

Рис. 2: Исходный двойной контур

Обзор литературы

Задачи упорядочивания или сортировки точек не имеют общего алгоритма решения, так как они сильно зависят от формулировки задачи. В документации языка Python[1] и других языков программирования описана встроенная функция `sorted()`, которая при сортировке входного двумерного массива возвращает данные, отсортированные по возрастанию или убыванию значений в столбцах или строках массива. Данная сортировка не подходит для решения задачи построения контура.

Популярные программные пакеты умеют строить контур внешней границы полигона, полученного по исходным данным. В пакете Shapely[2] контуром двойной петли будет являться внешний контур объединения полигонов, образованных данными петлями. Функция `findContours()` из библиотеки OpenCV[3] находит только внешние контуры объекта и контуры отверстий, если таковые имеются. Как в Shapely, так и в OpenCV функции умеют упорядочивать только внешнюю границу объекта, которая необязательно содержит все исходные данные.

Наиболее простой способ найти внешнюю границу точек, заданных в двумерном пространстве, — это найти их выпуклую оболочку[4] — такой наименьший выпуклый многоугольник, что каждая исходная точка лежит либо на его границе, либо в его внутренней области. Задача построения выпуклой оболочки была решена ещё в прошлом столетии несколькими методами. Описание наиболее популярных и понятных алгоритмов сканирования по Грэхему и обходу по Джарвису можно найти во многих пособиях по алгоритмам и структурам данных, например, в книге Томаса Кормена[4], часто используемой в учебных заведениях. Выпуклая оболочка не имеет самопересечений и представляет из себя одинарный контур, однако она не всегда включает в себя все исходные точки, поэтому алгоритмы её построения не решают поставленную задачу.

В упомянутом раннее алгоритме Грэхема сначала исходные точки сортируются в порядке возрастания полярного угла[4]. Сортировка по полярным координатам[5] — хороший способ упорядочивания всех исходных точек, который можно использовать для решения поставленной задачи. В

главах 1 и 2 будет показано, что данная сортировка позволяет построить контур, удовлетворяющий всем требованиям поставленной задачи, однако не самый оптимальный по сравнению с другими алгоритмами, описанными в главе 1.

Несмотря на то, что алгоритмы построения выпуклой оболочки строят контур не по всем исходным данным, их можно использовать для построения начального контура, в который далее будут добавляться пропущенные точки, что будет продемонстрировано в главе 1.

Поставленная задача похожа на задачу коммивояжёра (ЗК)[6] — обобщённую форму задачи нахождения наименьшего замкнутого контура, соединяющего несколько точек на плоскости.

Необходимо подчеркнуть, что задача коммивояжёра — NP-сложная и все эффективные, т. е. сокращающие полный перебор, методы её решения — эвристические, которые в большинстве случаев находят не самый оптимальный маршрут, а приближённое решение. Среди простейших способов решения задачи популярны жадные алгоритмы (алгоритм ближайшего соседа[7]), они не всегда дают оптимальные результаты, но замещают это своей простой реализацией.

В задаче коммивояжёра, как и в поставленной задаче, необходимо упорядочить все исходные данные без пропусков. Используя алгоритм 2-opt, впервые описанный в статье Круза[8] в 1958 году, для решения ЗК, можно получить контур без самопересечений. Но не существует такого алгоритма решения задачи коммивояжёра, который бы всегда давал на выходе одинарный контур, поэтому для решения данной задачи алгоритмами решения ЗК потребуется их модифицировать.

Таким образом, отсутствие алгоритма упорядочивания всех исходных данных так, чтобы они представляли из себя оптимальный одинарный контур, говорит о том, что необходимо изобрести собственные подходы для решения данной задачи.

Глава 1. Обзор алгоритмов

В данной главе сформулированы алгоритмы, решающие поставленную задачу, а также продемонстрированы результаты их применения для упорядочивания двойного контура, состоящего из 159 точек (см. рис. 2).

1.1 Алгоритм ближайшего соседа

Алгоритм ближайшего соседа — один из первых алгоритмов, использованный для приближённого решения задачи коммивояжёра. Его можно сформулировать следующим образом: точки последовательно включаются в маршрут, причём очередная включаемая точка должна быть ближайшей к последней выбранной точке среди всех остальных, ещё не включённых в маршрут. Результатом работы алгоритма является упорядоченный список входных точек.

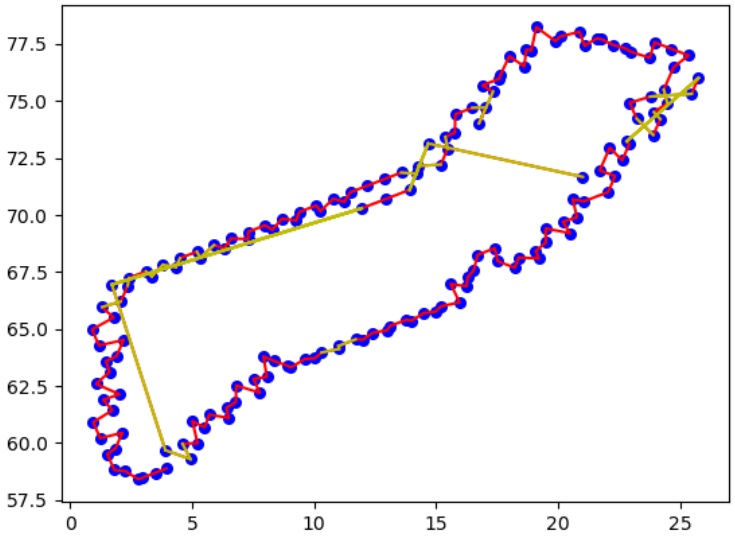

Рис. 3: Результат работы алгоритма ближайшего соседа

Алгоритм прост в реализации, быстро выполняется, но может выдавать неоптимальные решения. Поэтому его использование в рамках рассматриваемой задачи не гарантирует, что построенный им маршрут будет одинарным контуром. Этот недостаток алгоритма можно увидеть на

рис. 3, где жёлтыми линиями обозначены пересекающиеся рёбра построенного контура, имеющего красный цвет.

1.2 Алгоритм 2-opt

Основная идея алгоритма 2-opt заключается в том, чтобы выбрать маршрут, который пересекает сам себя, и изменить его так, чтобы он не пересекался.

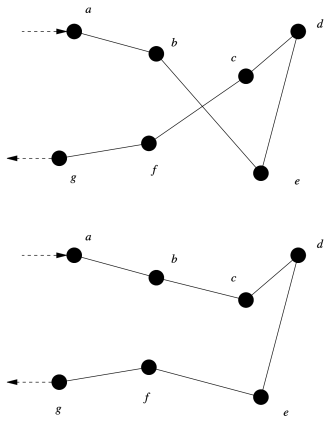

Рис. 4: Механизм обмена в алгоритме 2-opt

Эта задача решается при помощи механизма обмена (см. рис. 4), который состоит из следующих шагов:

1. На вход подаётся маршрут, индекс начала и конца петли.
2. В новый маршрут добавляются точки из старого маршрута до начала петли.
3. В новый маршрут добавляются точки петли в порядке, обратном старому маршруту.
4. После конца петли в новый маршрут добавляются точки в том же порядке, в котором они были в старом маршруте.

Данный механизм обмена был использован для нахождения контура по исходным данным. Пока уменьшалась длина пути, алгоритм искал пересечения в маршруте и удалял петли при помощи механизма обмена. В

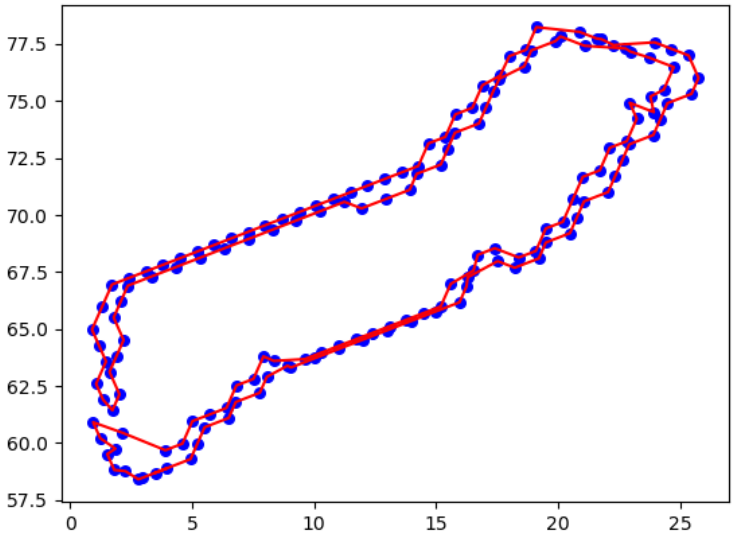

Рис. 5: Результат работы алгоритма 2-opt

результате был получен маршрут, представляющий из себя двойной контур (рис. 5). Следовательно, можно говорить, что эвристический алгоритм 2-opt решения ЗК не дал корректного результата для решаемой задачи построения одинарного контура из двойного на данном примере.

1.3 Алгоритм построения одинарного контура с пропуском точек

Для того чтобы получить одинарный контур, можно остановить алгоритм ближайшего соседа, когда маршрут будет замыкаться. В итоге улучшенный алгоритм будет отличаться от алгоритма ближайшего соседа только критерием остановки, т. е. в маршрут будут добавляться точки, пока ближайшая точка, не вошедшая в маршрут, не будет расположена дальше к рассматриваемой точке, чем первая точка маршрута.

В результате работы алгоритма получается одинарный контур, построенный не по всем входным точкам. На рис. 6 пропущенные данные обозначены зелёными точками.

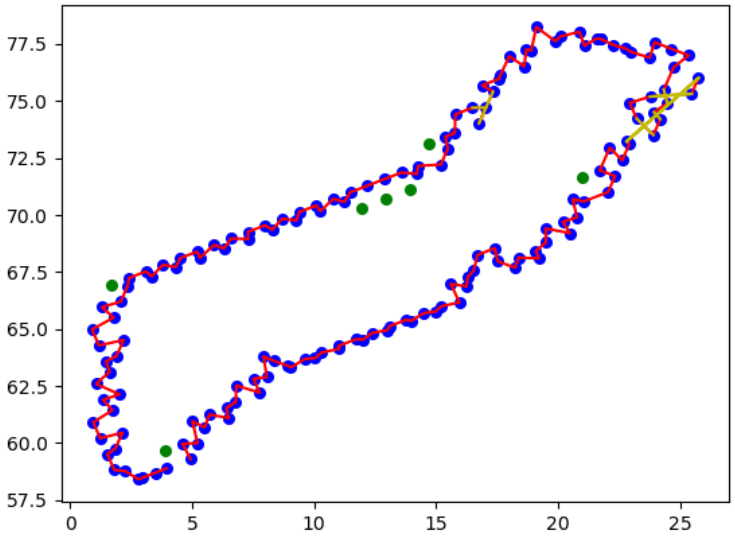

Рис. 6: Результат работы алгоритма ближайшего соседа с пропуском точек

1.4 Алгоритм устранения пропущенных точек

Для устранения пропусков после выполнения алгоритма построения одинарного контура, описанного в пункте 1.3, все пропущенные точки надо добавить в существующий контур оптимальным образом, так, чтобы разброс между точками контура был минимальным, значит, имеет смысл добавлять точки между двумя точками контура, сумма расстояний до которых минимальна.

Результаты использования данного метода продемонстрированы на рис. 7. Построенный контур не имеет пропущенных точек, однако у него сохранились самопересечения, которые были до добавления пропущенных точек (см. рис. 6). Способ их устранения будет описан в следующем пункте.

Стоит обратить внимание на то, что исходные контуры, имеющие большее количество точек, чем на представленном примере, после выполнения алгоритма из п. 1.3 имеют приблизительно такое же количество пропущенных точек, а после их устранения такое же небольшое количество петель, обычно не превышающее 10 (см. рис. 8). Следует отметить, что на данных, на которых большинство точек были пропущены после алгоритма из п. 1.3, после устранения пропусков было обнаружено большое количе-

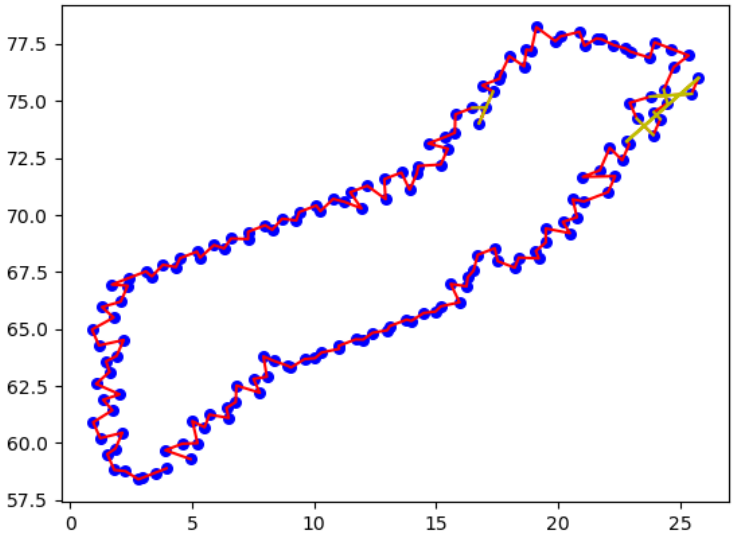

Рис. 7: Результат работы алгоритма ближайшего соседа без пропуска точек

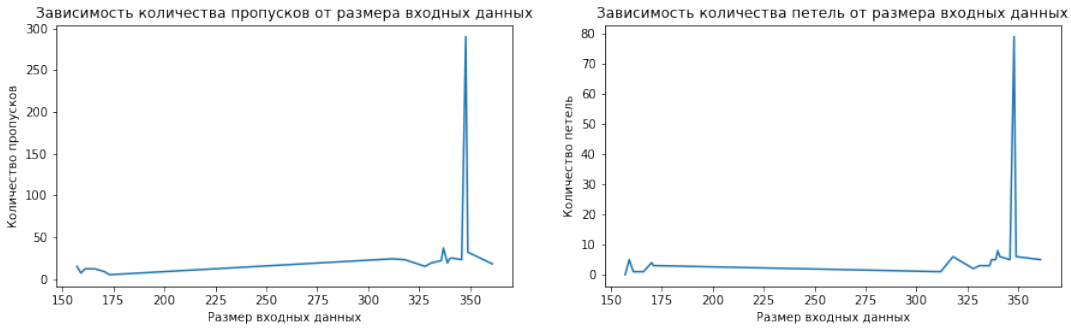

Рис. 8: График зависимости количества пропусков от размера входных данных – слева, график зависимости количества самопересечений от размера входных данных – справа

ство петель, так что описанный в данном пункте метод лучше использовать на данных с небольшим количеством пропусков.

1.5 Алгоритм устранения самопересечений

Для того чтобы алгоритм гарантировал отсутствие самопересечений в маршруте, в него можно добавить метод, который находит пересечения и устраняет их при помощи механизма обмена из алгоритма 2-opt, описанного в пункте 1.2. Найти пересечения двух отрезков можно наивным

способом[9]: найти точку пересечения прямых и проверить, принадлежит ли она обоим отрезкам.

При добавлении описанного выше способа устранения самопересечений в алгоритм построения одинарного контура с пропуском точек и с самопересечениями после устранения в нём пропусков получается алгоритм, решающий поставленную задачу (см. рис. 9).

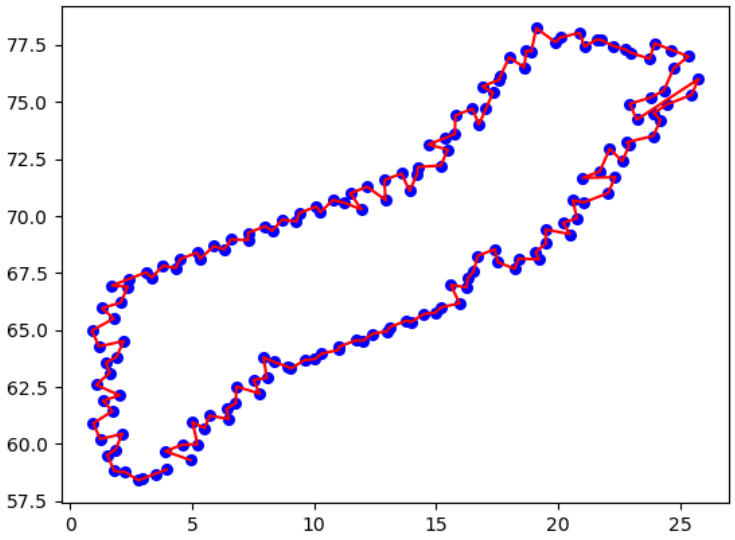

Рис. 9: Результат работы алгоритма ближайшего соседа без самопересечений

1.6 Алгоритм добавления точек из второго контура в первый

В условии поставленной задачи говорилось, что на вход подаются два последовательных контура. Каждый отдельный контур не является решением задачи, так как не содержит точки другого контура. При добавлении точек из второго контура в первый с помощью метода устранения пропущенных точек, описанного в п. 1.4, можно решить проблему пропуска точек (см. рис. 10). Можно заметить, что в построенном таким образом контуре присутствует самопересечение. Данная проблема решается описанным раннее методом устранения самопересечений (см. п. 1.5).

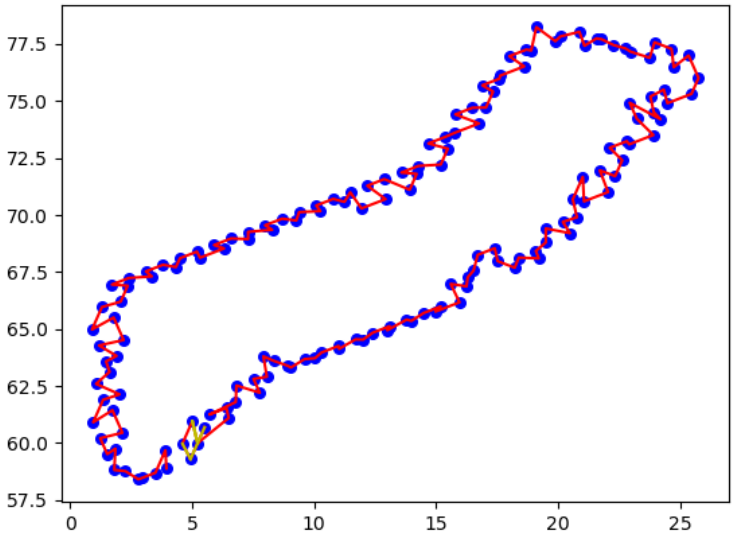

Рис. 10: Результат работы алгоритма добавления точек из второго контура в первый

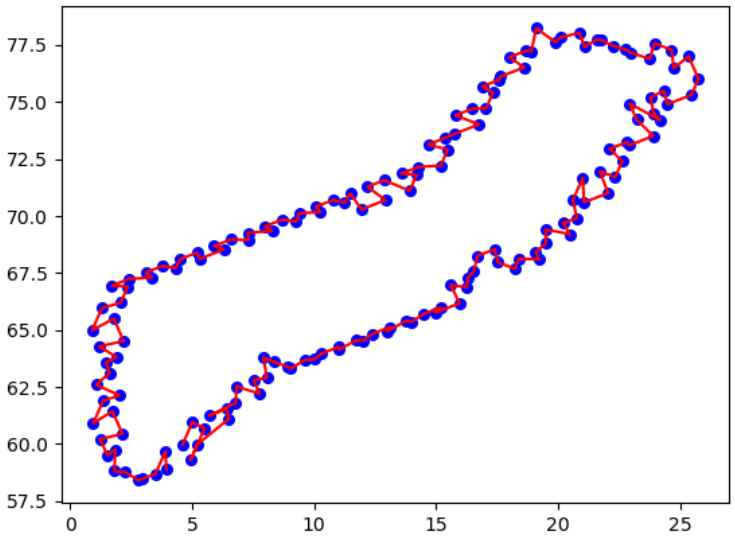

Рис. 11: Результат работы алгоритма добавления точек из второго контура в первый без самопересечений

Таким образом, при использовании метода устранения самопересечений после алгоритма добавления точек из второго контура в первый можно решить поставленную задачу, т. е. построить одинарный контур без пропусков и самопересечений (рис. 11).

1.7 Алгоритм сортировки точек по полярным координатам

Алгоритм сортировки по полярным координатам — единственный способ упорядочивания точек, не нуждающийся в дополнительных модификациях для решения поставленной задачи. Для сортировки точек по полярным координатам сначала следует перевести декартовы координаты исходных точек в пару полярных координат (см. 1) с центром, совпадающим с центром масс исходных данных.

$$\begin{aligned} \rho &= \sqrt{(x - x_0)^2 + (y - y_0)^2}, \\ \phi &= \arctan \frac{x - x_0}{y - y_0}, \end{aligned} \tag{1}$$

где (x,y) — декартовы координаты точки, (ρ, ϕ) — полярные координаты точки, (x_0, y_0) — центр полярных координат.

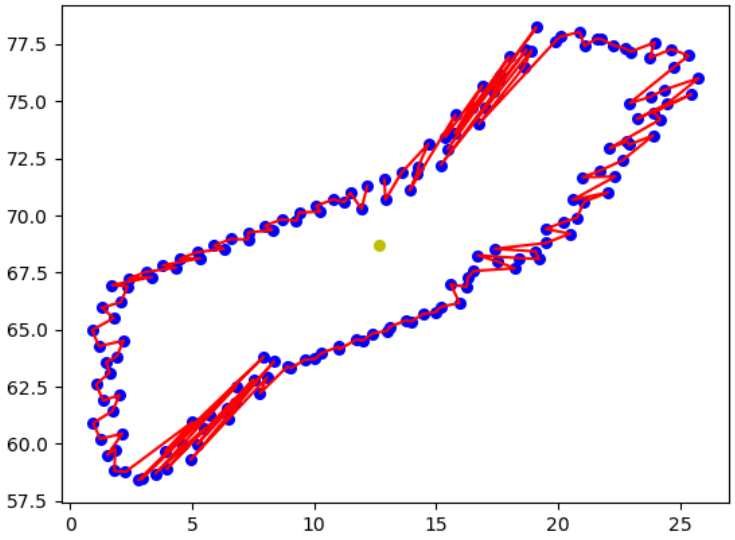

Рис. 12: Результат работы алгоритма сортировки точек по полярным координатам

После нахождения полярных координат точки сортируются по полярному углу, в случае, когда две точки имеют одинаковый полярный угол, они сортируются по возрастанию полярного радиуса.

При применении данного алгоритма был получен следующий контур

(см. рис. 12). Алгоритм позволяет обойти точки по часовой стрелке и получить упорядоченный контур без пропусков и самопересечений (значит он решает поставленную задачу), но при удалении от центроиды получаются большие шаги между соседними точками с близкими значениями полярных углов. Можно заметить, что длина полученного контура получилась значительно больше, чем у контуров, построенных алгоритмами на основе на метода ближайшего соседа (рис. 9,11).

1.8 Алгоритм Джарвиса

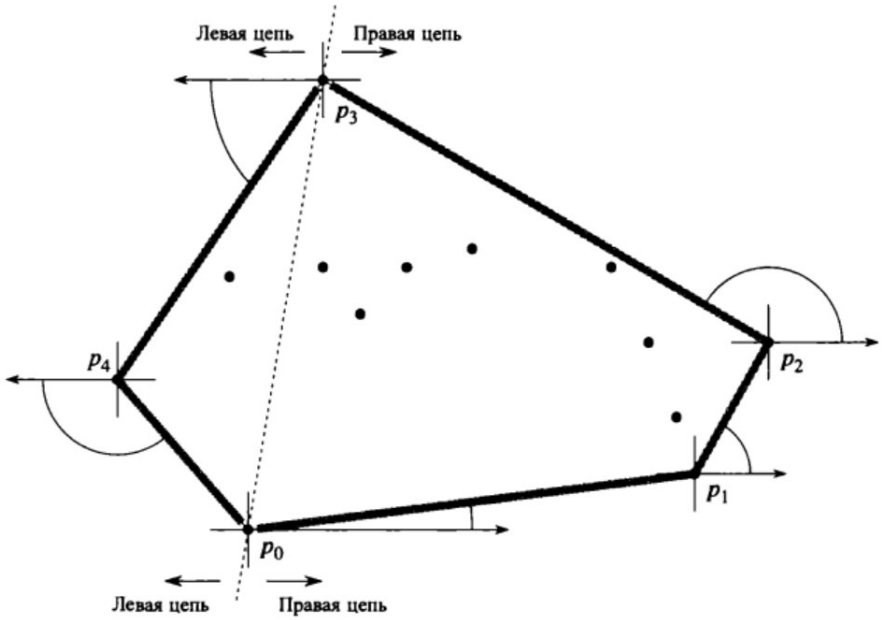

Рис. 13: Описание работы алгоритма Джарвиса

При построении выпуклой оболочки множества точек путём обхода по Джарвису используется метод, известный как упаковка пакета или упаковка подарка. Он состоит из следующих шагов:

1. В список добавляется вершина с наименьшим значением координаты x.
2. В список добавляется вершина, образующая с последней добавленной точкой и вектором, направленным по оси Ox, наименьший полярный угол.

3. Пункт 2 повторяется, пока в список не будет добавлена вершина с наибольшим значением координаты x.
4. В список добавляется вершина, образующая с последней добавленной точкой и вектором, направленным против оси Ox, наименьший полярный угол.
5. Пункт 4 повторяется, пока не будет добавлена вершина с наименьшим значением координаты x.

Работа данного алгоритма проиллюстрирована на рис. 13.

Алгоритм Джарвиса можно реализовать путём единообразного обхода вокруг выпуклой оболочки без отдельного построения правой и левой цепей. В данной реализации дополнительно накладывается требование, чтобы последовательность углов между сторонами оболочки и вектором, направленным по оси Ox, строго возрастала.

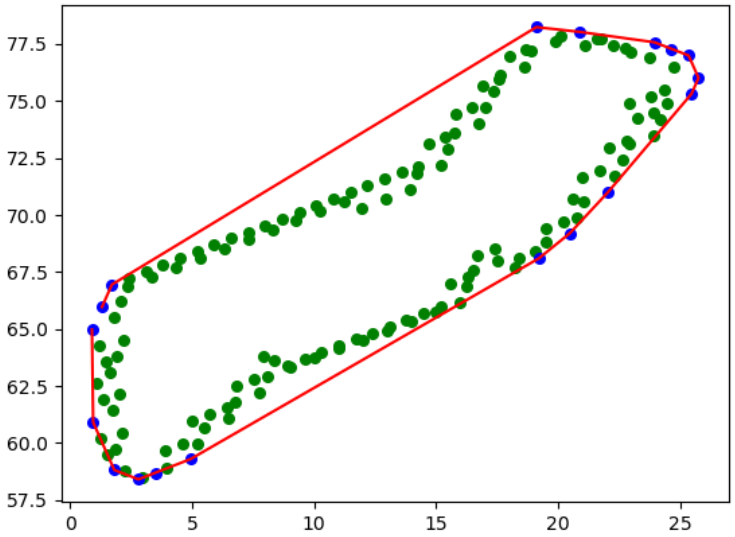

Рис. 14: Результат работы алгоритма Джарвиса построения выпуклой оболочки

При применении данного алгоритма была построена выпуклая оболочка — контур, в который были включены не все точки в связи с тем, что исходные контуры представляли из себя невыпуклые многоугольники и пересекались (см. рис. 14).

1.9 Алгоритм добавления точек в выпуклую оболочку

Для решения поставленной задачи, отталкиваясь от контура, построенного алгоритмом Джарвиса, необходимо добавить в контур пропущенные точки. На рис. 14 видно, что выпуклая оболочка была построена из небольшого количества исходных данных, поэтому пропусков в данном случае получилось гораздо больше, чем у контура, построенного улучшенным алгоритмом ближайшего соседа (см. рис. 6). Добавление большого числа пропущенных точек в выпуклую оболочку методом, описанным в п. 1.4, приводит к образованию большого количества петель (см. рис. 8), поэтому был сформулирован другой алгоритм устранения пропусков, подходящий для данного случая.

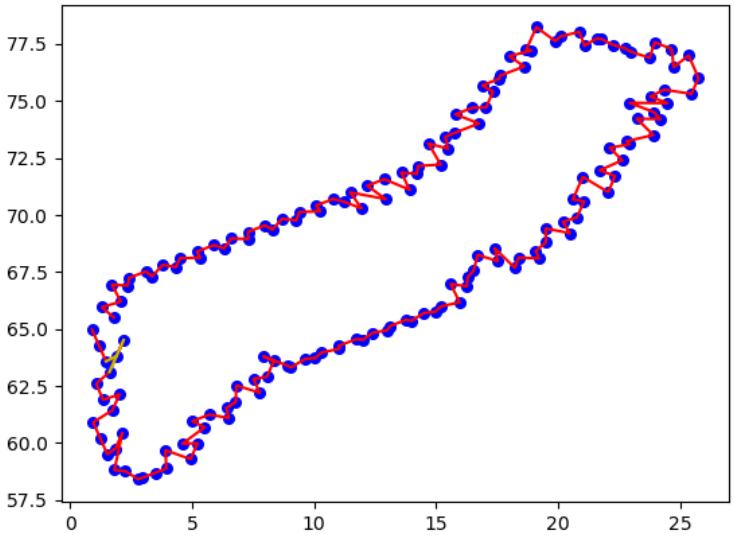

Рис. 15: Результат работы алгоритма, основанного на построении выпуклой оболочки

Алгоритм упорядочивания наложенных контуров, основанный на построении выпуклой оболочки, состоит в следующем:

1. По исходным данным строится выпуклая оболочка.
2. Для каждой точки, не лежащей в выпуклой оболочке, находится ближайшее ребро выпуклой оболочки и расстояние до него.

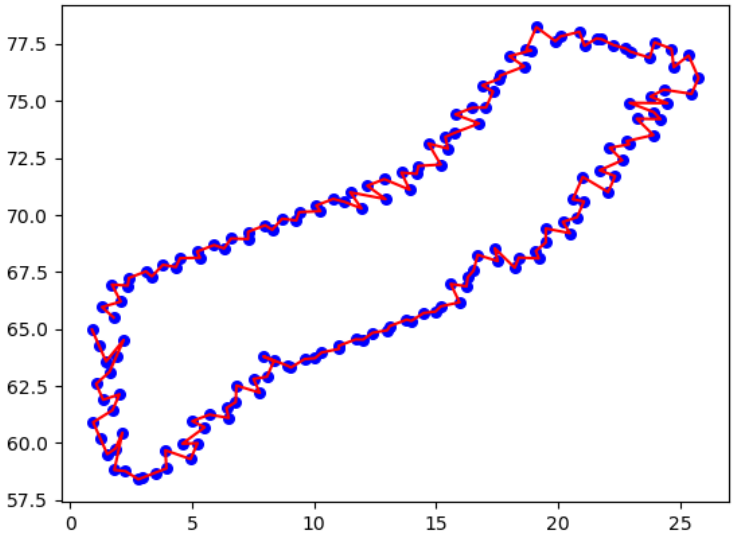

Рис. 16: Результат работы алгоритма, основанного на построении выпуклой оболочки, с устранением самопересечений

3. Создаётся список, куда добавляется первый элемент выпуклой оболочки.
4. Рассматривается первое ребро выпуклой оболочки.
5. Для каждой точки, расстояние от которой до рассматриваемого ребра наименьшее, находится её проекция на данное ребро.
6. Точки сортируются по возрастанию расстояния их проекции до последнего элемента списка.
7. Отсортированные точки добавляются в список.
8. В список добавляется следующий элемент выпуклой оболочки.
9. Рассматривается следующее ребро выпуклой оболочки.
10. Пункты 5-9 повторяются, пока в список не будут добавлены все исходные данные.

Результаты работы описанного выше алгоритма продемонстрированы на рис. 15. По ним можно заключить, что с помощью данного алгоритма можно построить одинарный контур, содержащий все исходные дан-

ные, однако он может иметь небольшое количество петель, которые можно устранить с помощью алгоритма, описанного в п. 1.5.

Подобным образом на основании алгоритма построения выпуклой оболочки можно создать алгоритм, решающий поставленную задачу. Контур (см. рис. 16), построенный при использовании данного алгоритма, удовлетворяет все требованиям задачи:

- Содержит все исходные точки.
- Не имеет самопересечений.
- Представляет из себя одинарный контур.

Глава 2. Обзор результатов

В данной главе описан программный код, решающий поставленную задачу, показаны результаты применения реализованных алгоритмов на 21-ом наборе данных, проведён сравнительный анализ лучших алгоритмов и на его основе сформулирован новый алгоритм.

2.1 Реализация

Алгоритмы из главы 1 были реализованы[10] на языке Python с использованием пакета для научных исследований NumPy[11] для ускорения работы вычислительных операций над массивами, библиотеки Matplotlib[12] для визуализации данных двумерной (2D) графикой и библиотекой Gmsh[13] для сохранения полученных контуров в формате .geo.

Все функции, реализующие алгоритмы упорядочивания наложенных контуров, расположены в файле `operations.py`.

В файле `sequence.py` реализован класс `Sequence`, описывающий упорядоченные данные, с методами `have_missed_data()`, `have_loops()`, `is_single_contour()`, `is_contour()`, нужными для оценки соответствия контура требованиям задачи, `get_contour_len()` — для получения длины построенного контура, `sorted(key)` — для сортировки данных алгоритмом, указанным в параметре `key` , и `show_contour()` — для визуализации контуров.

В папке `datasets` находятся примеры исходных данных.

В ноутбуке `analysis.ipynb` расположены результаты оценки разработанных алгоритмов и их сравнения после применения на данных из папки `datasets`.

Программа `main.py` считывает данные из файла, указанного в аргументах командной строки, строит одинарный контур при помощи лучшего алгоритма, описанного в п. 2.3, и сохраняет полученный контур в файл, указанный в аргументах командной строки. Для запуска программы необходимо прописать в командной строке следующее:

```
python main.py <input_file.dat> <output_file.geo_unrolled>
```

Где `<input_file.dat>` — имя файла с исходными данными, а `<output_file.geo_unrolled>` — имя файла, в который надо сохранить упорядоченный контур.

2.2 Оценка алгоритмов

Алгоритмы оценивались по соответствию построенных ими контуров требованиям поставленной задачи, т. е. по следующим критериям:

1. Отсутствие пропущенных точек.
2. Отсутствие самопересечений.
3. Построенный контур — одинарный.

Если построенный контур удовлетворял всем требованиям, то он решал поставленную задачу.

В табл. 1 представлены доли данных, на которых построенные контуры удовлетворили перечисленным требованиям. Следует отметить, что все алгоритмы, кроме алгоритма ближайшего соседа с пропуском точек и алгоритма построения выпуклой оболочки, построили контур из всех имеющихся данных.

Применение алгоритма устранения самопересечений после алгоритма ближайшего соседа без пропуска точек, алгоритма вставки второго контура в первый и алгоритма вставки точек в выпуклую оболочку позволило улучшить построенные контуры и во всех случаях достичь близкий к 100% результат. Поэтому данный алгоритм можно считать отличным способом решения проблемы с самопересечениями.

Практически все алгоритмы справились с требованием построения одинарного контура, за исключением алгоритма 2-opt, который на всех данных строил двойной контур, и алгоритма ближайшего соседа и производных от него алгоритмов, которые на одном наборе данных не справились с данным требованием из-за того, что при добавлении большого количества пропущенных точек, было получено большое количество петель, устранение которых и привело к образованию двойного контура.

Таким образом, с поставленной задачей смогли справиться на всех данных несколько алгоритмов:

- Алгоритм вставки второго контура в первый с устранением самопересечений.
- Алгоритм сортировки по полярным координатам.
- Алгоритм вставки точек в выпуклую оболочку с устранением самопересечений.

Таблица 1: Удовлетворение алгоритмов требованиям поставленной задачи

Алгоритмы	Без пропуска точек	Без петель	Одинарный контур	Решение задачи
Алгоритм ближайшего соседа	100% (21/21)	0% (0/21)	43% (9/21)	0% (0/21)
Алгоритм ближайшего соседа с пропуском точек	0% (0/21)	5% (1/21)	95% (20/21)	0% (0/21)
Алгоритм ближайшего соседа без пропуска точек	100% (21/21)	5% (1/21)	95% (20/21)	5% (1/21)
Алгоритм ближайшего соседа без петель	100% (21/21)	95% (20/21)	95% (20/21)	95% (20/21)
Алгоритм вставки второго контура в первый	100% (21/21)	5% (1/21)	100% (21/21)	5% (1/21)
Алгоритм вставки второго контура в первый без петель	100% (21/21)	100% (21/21)	100% (21/21)	100% (21/21)
Алгоритм сортировки по полярным координатам	100% (21/21)	100% (21/21)	100% (21/21)	100% (21/21)

Алгоритм Джарвиса построения выпуклой оболочки	0% (0/21)	100% (21/21)	100% (21/21)	0% (0/21)
Алгоритм вставки точек в выпуклую оболочку	100% (21/21)	5% (1/21)	100% (21/21)	5% (1/21)
Алгоритм вставки точек в выпуклую оболочку без петель	100% (21/21)	100% (21/21)	100% (21/21)	100% (21/21)
Алгоритм 2-opt	100% (21/21)	76% (16/21)	0% (0/21)	0% (0/21)
Объединение лучших алгоритмов	100% (21/21)	100% (21/21)	100% (21/21)	100% (21/21)

2.3 Сравнительный анализ алгоритмов

Среди алгоритмов, описанных в главе 1, были отобраны алгоритмы, справившиеся с поставленной задачей хотя бы в 95% случаев, и использованы для построения одинарного контура на различных входных данных.

Полученные в результате работы данных алгоритмов контуры сравнивались по длине. Длина вычислялась как сумма евклидовых расстояний между соседними точками. Чем меньше получилась длина, тем меньше был разброс между соседними точками, значит, лучше отработал алгоритм.

Таблица 2: Длины построенных контуров

	Улучшенный алгоритм ближайшего соседа	Вставка второго контура в первый	Сортировка по полярным координатам	Вставка точек в выпуклую оболочку	Объединение лучших алгоритмов
1	**107.61779**	114.34608	214.70337	110.31607	**107.61779**

2	**113.76290**	121.64087	354.84160	128.61215	**113.76290**
3	180.97136	**131.14641**	419.43639	141.19025	**131.14641**
4	**119.76365**	124.74927	258.07371	127.78836	**119.76365**
5	**121.46090**	129.12460	361.10041	130.95748	**121.46090**
6	**118.45502**	124.88444	250.91063	132.63265	**118.45502**
7	**117.78417**	125.16115	320.82950	133.47644	**117.78417**
8	**108.10310**	114.75650	160.16786	115.16257	**108.10310**
9	**118.61635**	122.46753	245.44413	129.45387	**118.61635**
10	**105.39820**	111.00424	210.66783	122.14973	**105.39820**
11	**129.56825**	135.63566	392.38708	144.29572	**129.56825**
12	122.83122	**120.27166**	225.69233	123.46143	122.83122
13	**119.83887**	124.88868	358.41321	135.55519	**119.83887**
14	**121.76140**	122.47075	223.93248	125.88969	**121.76140**
15	**120.47977**	125.50006	309.86489	138.59917	**120.47977**
16	**105.82502**	111.21777	151.62383	108.24094	**105.82502**
17	**109.15956**	114.41017	204.62595	127.10034	**109.15956**
18	**116.24701**	130.71500	206.47878	125.58276	**116.24701**
19	**121.61628**	127.88899	274.16515	136.29843	**121.61628**
20	**117.00994**	120.51214	210.88332	123.30736	**117.00994**
21	**117.07529**	122.36762	309.04598	125.06538	**117.07529**

Длины построенных контуров можно увидеть в табл. 2. По данным результатам можно сделать вывод, что улучшенный алгоритм ближайших соседей строит самый оптимальный контур (на 19 из 21 датасетов он показал минимальный результат), алгоритм вставки второго контура в первый и алгоритм вставки точек в выпуклую оболочку справились чуть хуже, а сортировка по полярным координатам построила контуры, сильно отличившиеся по длине в худшую сторону.

В связи с тем, что алгоритм ближайших соседей может допускать ошибки и решать задачу не на всех данных, был добавлен алгоритм, объ-

единяющий его и алгоритм вставки второго контура в первый и состоящий из следующих шагов:

1. Строится контур при помощи алгоритма ближайшего соседа с устранением петель.
2. Проверяется, удовлетворяет ли контур требованиям задачи. Если удовлетворяет, то алгоритм возвращает данный контур и завершает работу.
3. Иначе алгоритм возвращает новый контур, полученный при помощи алгоритма вставки второго контура в первый с устранением петель.

В данном алгоритме сначала используется улучшенный алгоритм ближайшего соседа, так как он строит контуры минимальной длины, однако из-за того, что он может допускать ошибки, проводится проверка построенного им контура, если он построен некорректно, то используется алгоритм, отлично показавший себя на всех данных, но строивший контуры немного длиннее, — алгоритм добавления точек из второго контура в первый.

Контур, построенный при помощи объединённого алгоритма, удовлетворил всем требованиям на всех датасетах (см. табл. 1) и почти везде (на 20 датасетах из 21) имел наименьшую длину (см. табл. 2) в сравнении с контурами, построенными лучшими алгоритмами, поэтому его можно считать самым оптимальным алгоритмом решения поставленной задачи.

Выводы

В данной работе была рассмотрена проблема упорядочивания наложенных контуров. Из различных алгоритмов упорядочивания точек были сформулированы алгоритмы, способные решить поставленную задачу. Была проведена оценка алгоритмов по мере удовлетворения результатов их работы требованиям задачи. По результатам оценки были отобраны лучшие алгоритмы:

1. Алгоритм ближайшего соседа с устранением пропусков точек и самопересечений.
2. Алгоритм добавления точек из второго контура в первый с устранением самопересечений.
3. Алгоритм сортировки точек по полярным координатам.
4. Алгоритм добавления точек в выпуклую оболочку с устранением самопересечений.

Было произведено сравнение данных алгоритмов по длине построенных контуров. Самый короткий контур в большинстве случаев получался при использовании улучшенного алгоритма ближайшего соседа. Алгоритм сортировки по полярным координатам всегда строил самый длинный контур, сильно превосходивший длины контуров, полученных другими алгоритмами.

После анализа результатов работы алгоритмов был сформулирован алгоритм, объединяющий улучшенный алгоритм ближайшего соседа и алгоритм добавления точек из второго контура в первый. Объединённый алгоритм построил контуры, имевшие наименьшие в сравнении с лучшими алгоритмами длины и удовлетворявшие всем требованиям поставленной задачи на всех наборах данных.

Заключение

В рамках выполнения выпускной квалификационной работы была поставлена задача упорядочивания наложенных контуров, были рассмотрены различные алгоритмы, позволяющие упорядочить точки.

В результате работы было сформулировано несколько алгоритмов, решающих поставленную задачу. На языке Python с использованием пакета Numpy был разработан программный код, реализующий данные алгоритмы. По результатам применения алгоритмов на 21-ом наборе данных была произведена оценка пригодности рассматриваемых алгоритмов по требованиям поставленной задачи. Среди алгоритмов, хорошо справившихся с задачей, был проведён сравнительный анализ, в результате которого был сформулирован лучший алгоритм, показавший отличные результаты при построении одинарного контура.

В связи с отсутствием работ по теме упорядочивания всех исходных данных в виде одинарного контура, данная работа может быть полезна для изучения указанной темы.

Список литературы

[1] Язык программирования Python [Электронный ресурс]: URL:https://python.org/ (дата обращения: 18.05.2021)

[2] Документация Shapely [Электронный ресурс]: URL:https://shapely.readthedocs.io/en/latest/index.html (дата обращения: 18.05.2021)

[3] Библиотека OpenCV [Электронный ресурс]: URL:https://opencv.org/ (дата обращения: 18.05.2021)

[4] Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ. М.: Вильямс, 2013. 1328 с.

[5] Follett B., Torrence E. The Student's Introduction to Mathematica// Cambridge University Press. 1999.

[6] Левитин А. В. Алгоритмы. Введение в разработку и анализ. М.: Вильямс, 2006. 576 с.

[7] Gutin G., Yeo A., Zverovich A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP // Discrete Applied Mathematics. 2002. No 117. P. 81–86.

[8] Croes G. A. A Method for Solving Traveling-Salesman Problems// Operations Research. 1958. Vol 6. No 6. P. 791–812.

[9] Шикин А. В., Боресков А. В. Компьютерная графика. Полигональные модели. М.: ДИАЛОГ-МИФИ, 2001. 464 с.

[10] Программный код [Электронный ресурс]: URL:https://github.com/erveve/sequence_contour (дата обращения: 31.05.2021)

[11] Библиотека Numpy [Электронный ресурс]: URL:https://numpy.org/ (дата обращения: 18.05.2021)

[12] Библиотека Matplotlib [Электронный ресурс]: URL:https://matplotlib.org/ (дата обращения: 18.05.2021)

[13] Gmsh [Электронный ресурс]: URL:http://gmsh.info (дата обращения: 31.05.2021)