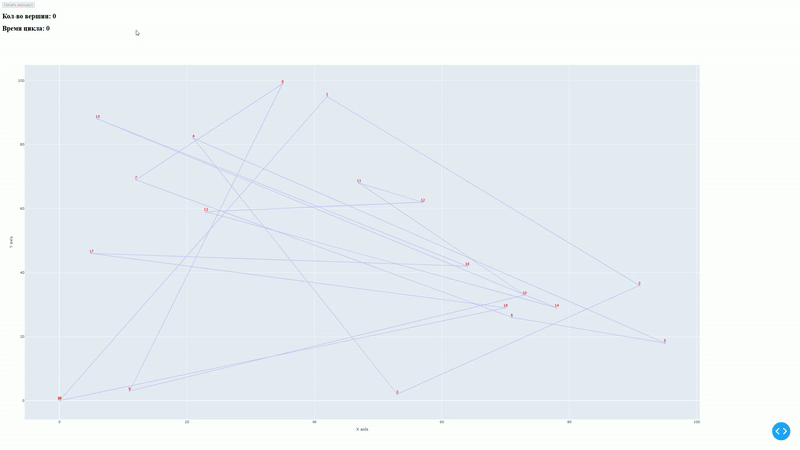
**Отчет**

Предполагалось, что маршрут должен пройти через все точки, поэтому основной целью в итоге был алгоритм для построения маршрута.

«Ниже должна быть gif’ка, демонстрирующая работу»

****

Содержание

[Постановка задачи 3](#_Toc186524833)

[Метод решения 5](#_Toc186524834)

[Сортировка точек 5](#_Toc186524835)

[Построение маршрута 6](#_Toc186524836)

[Тестовый файл №1 8](#_Toc186524837)

[Тестовый файл №2 10](#_Toc186524838)

[Тестовый файл №9 11](#_Toc186524839)

[Тестовый файл №11 12](#_Toc186524840)

# Постановка задачи

Требуется построить оптимальную по стоимости железную дорогу из доступных элементов

**Исходные данные**

Текстовый файл следующего формата:

Ключевые слова:

• **DATA**– начало блока по исходным

строительным элементам

• **ROUTE** – начало блока маршрутных точек

• **ORDER** – начало блока порядка элементов

• -- комментарий может быть либо в новой

либо в конце текущей строки

• / - окончание блока данных

• Данные в блоке пишутся с новой строки

**Ключевые слова. DATA**

• L1 – прямой блок длины 1

• L2 – прямой блок длины 2

• L3 – прямой блок длины 3

• L4 – прямой блок длины 4

• T4 – поворотный блок 𝜋/4 радиус кривизны 3

• T8 – поворотный блок 𝜋/8 радиус кривизны 3

• B1 – прямой блок длины 4 c поднятием в центре, нужен для организации пересечения путей

• После типа блока определено доступное количество элементов и цена за элемент

**Ключевые слова. ROUTE**

• Координаты точек, влияющие на оценку стоимости маршрута, в формате *x y v*

• 10 100 1000

• 10 24 -100

• Координата 0-0 обязательная, маршрут всегда начинается и заканчивается в точке 0-0

• Стоимость каждой точки маршрута учитывается следующим образом , где 𝑑𝑖𝑗

минимальное расстояние от точек соединения элементов пути до 𝑉𝑖

**Ключевые слова. ORDER**

• Указывается используемые элементы в порядке следования от стартовой точки

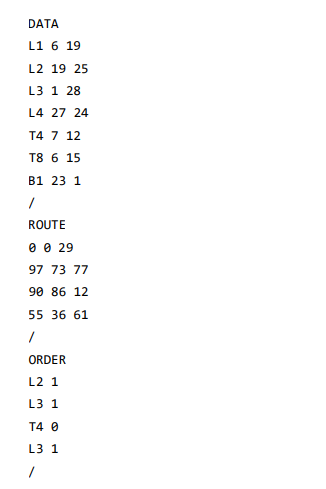
• Указываются направление поворота элемента относительно предыдущего

(1 – направо или прямо по движению, -1 – налево)

• Проверка на соответствие секции DATA

• Нумерация элементов с единицы

**Пример входного файла**



# Метод решения

## Сортировка точек

Порядок маршрутных точек оптимизируется алгоритмом ACO (Ant Colony Optimization).

Данный алгоритм изначально ищет путь оценивая только расстояния между точками, а у нас поворотные блоки имеют радиус поворота, поэтому он был немного изменен. К оценке расстояния теперь добавляется «штраф», который зависит от угла и расстояния между точками. Чем меньше расстояние и меньше угол (резкий поворот) – больше штраф, таким образом алгоритм избегает резких поворотов, выбирая более плавные маршруты даже с чуть большим расстоянием (ниже показано в сравнении).

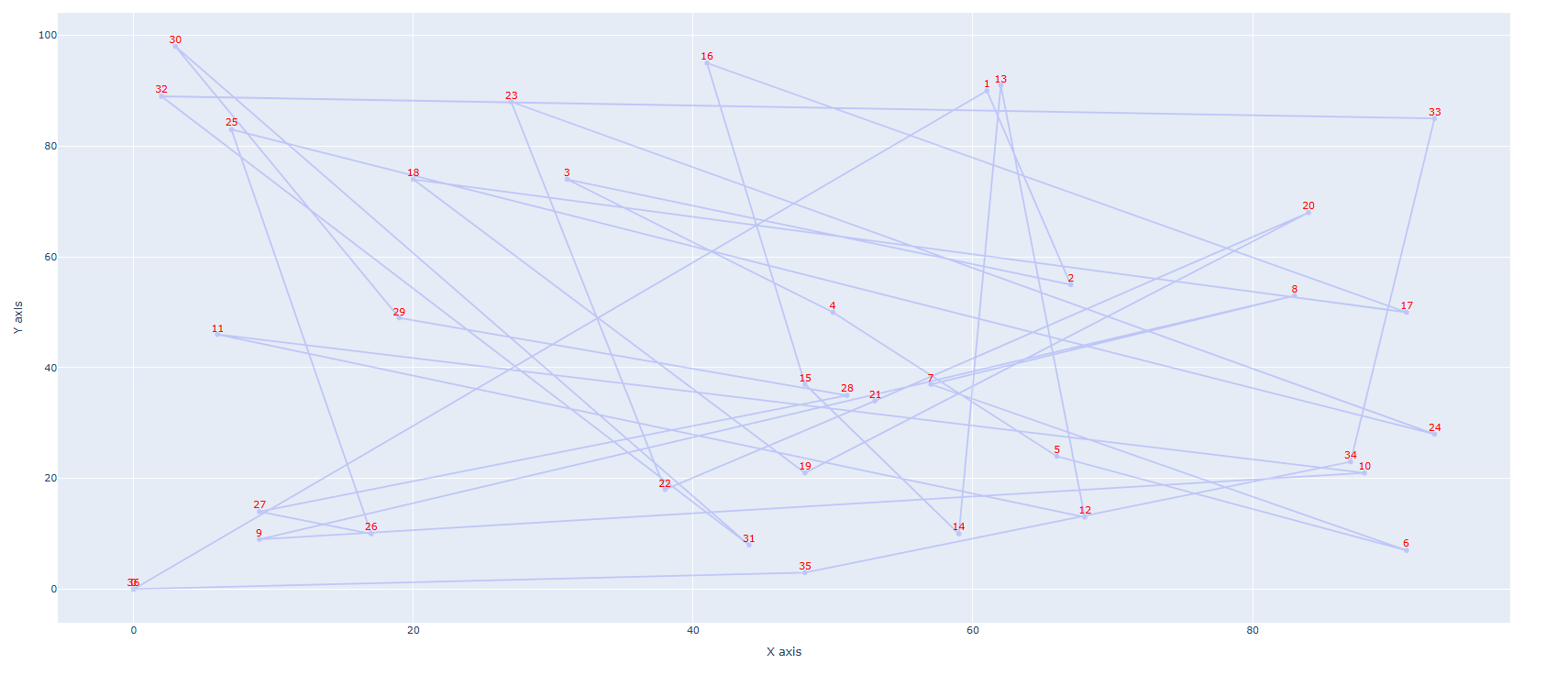
Пример:  


Рисунок 1 Файл №6. Исходный порядок точек

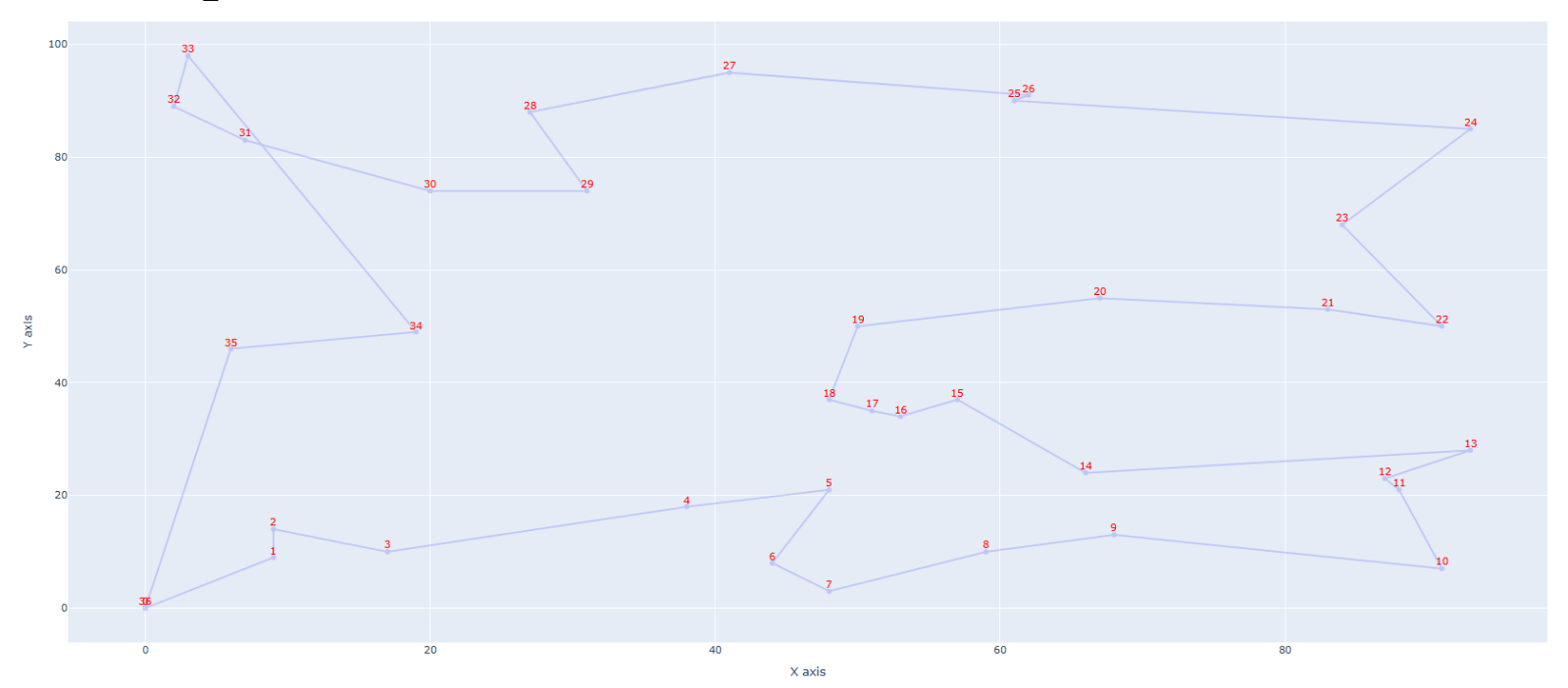


Рисунок 2 Файл №6. Оптимизированный ACO порядок точек

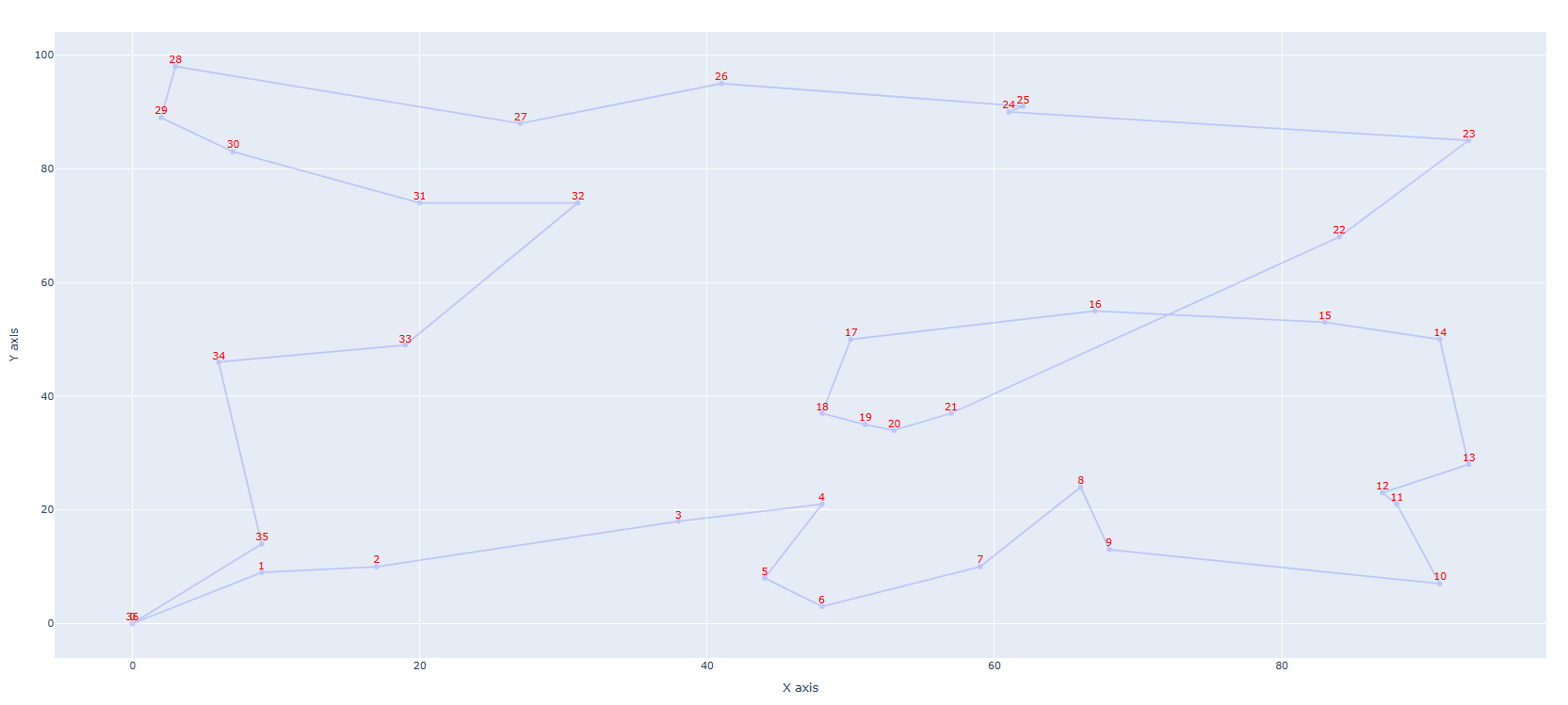


Рисунок 3 Файл №6. Оптимизированный измененным ACO порядок точек

Резких углов стало меньше.

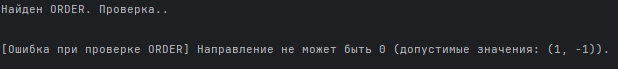
## Построение маршрута

Само построение маршрута происходит с помощью алгоритма A\*. Для оценки использовались: расстояние до целевой точки, стоимость блока и отклонение угла до целевой точки. Была попытка включить в оценку угол, с которым он достигнет в целевую точку в зависимости от расстояния до следующей точки после целевой (чтобы при близко расположенной следующей точке после целевой не нужно было разворачиваться), но это не дало желаемого результата.

Обработка пересечений это самое веселое =). Просто проверить пресечение линий — это не так сложно, но т.к. на каждой итерации из каждой точки строятся все доступные варианты следующего шага маршрута, то необходимо проверять новые линии на пересечение уже построенного маршрута и самопересечения нового. По мере построения маршрута это все вычисляется «в лоб» очень долго. Поэтому, для оптимизации процесса, используется пространственное индексирование с помощью сетки и проверка пересечений с использованием векторного произведения. Сетка строится по существующему маршруту и обновляется после достижения очередной целевой точки. В ячейках сетки хранятся отрезки, которые туда попадают. В итоге остается проверить пересечение нового сегмента пути с отрезками, попадающими в одну ячейку сетки. Это существенно сокращает количество проверок.

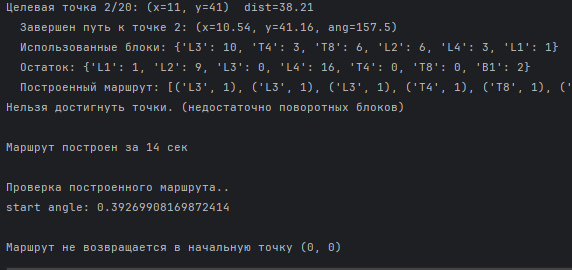
# Тестовый файл №1

Результаты проверки ORDER:

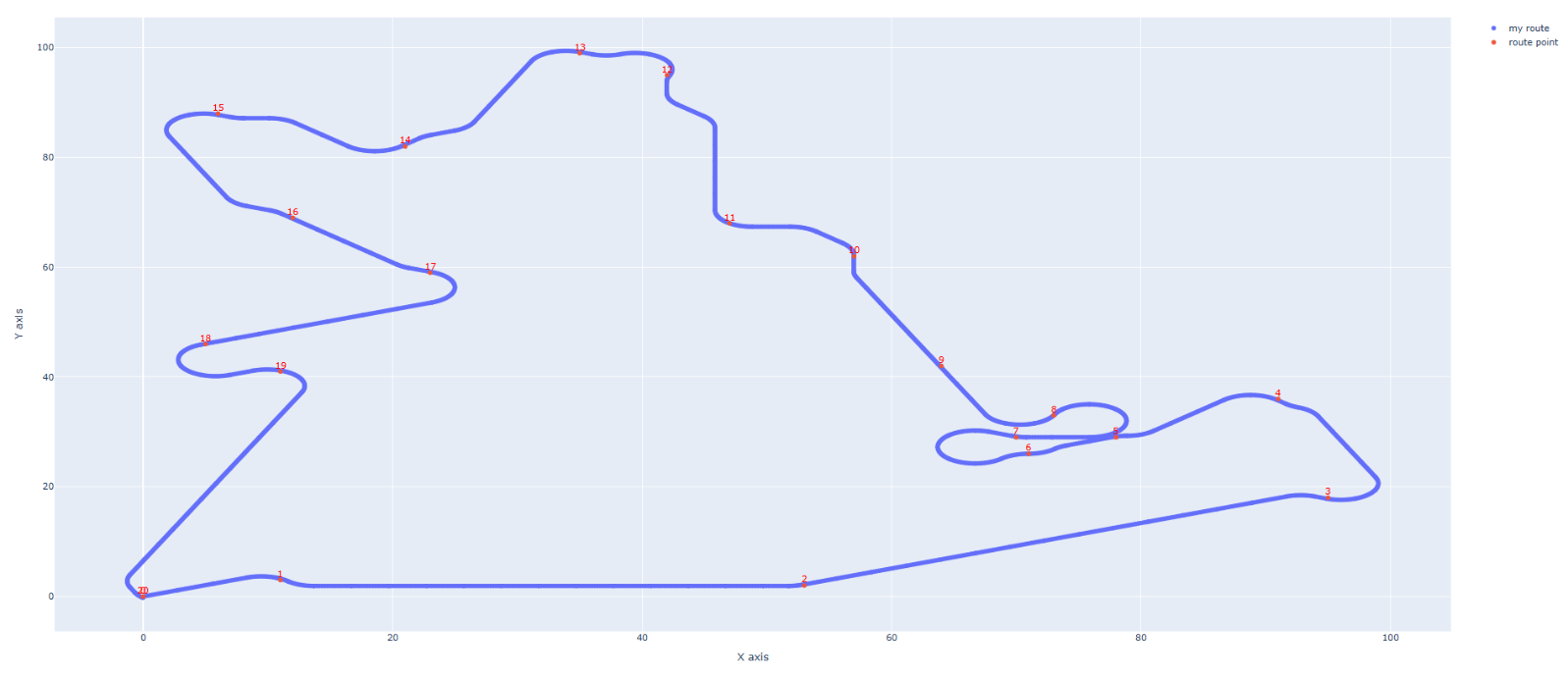


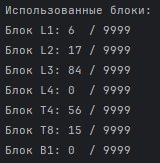
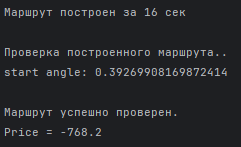
Попробуем сами построить маршрут.

Из доступных блоков маршрут построить нельзя.

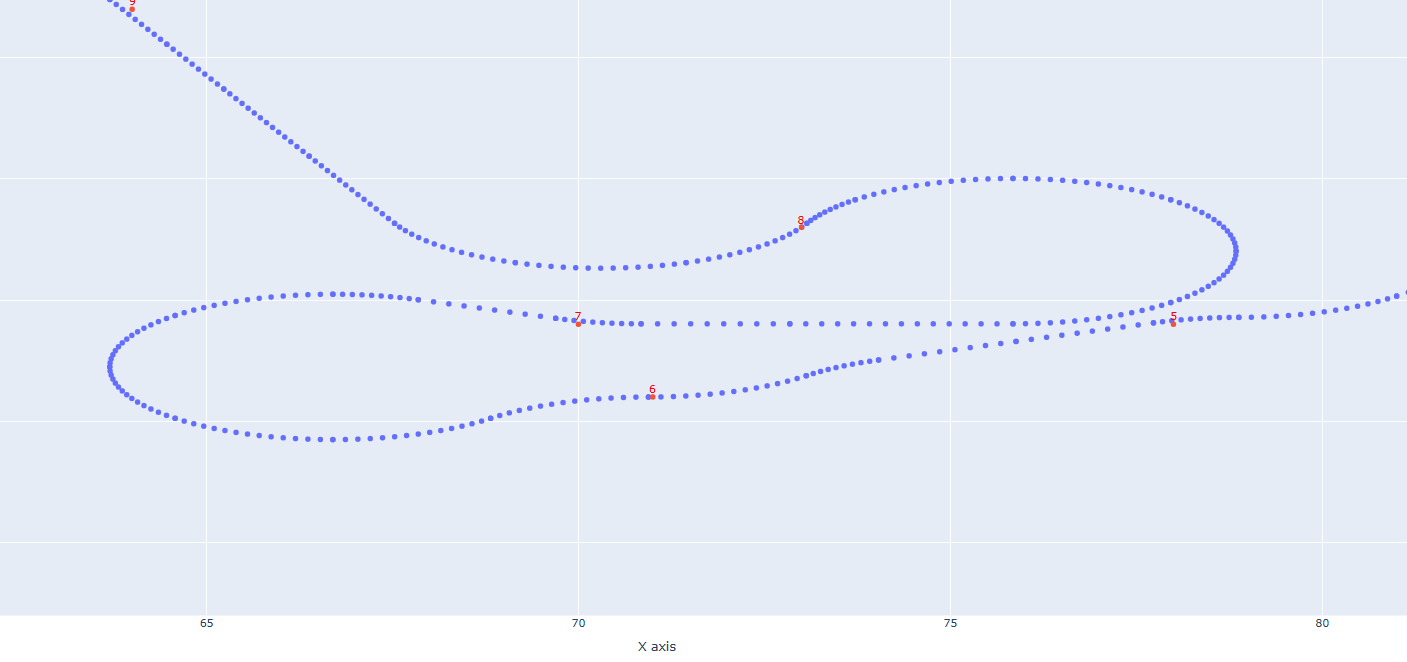


Но если убрать ограничение на количество блоков строится такой маршрут



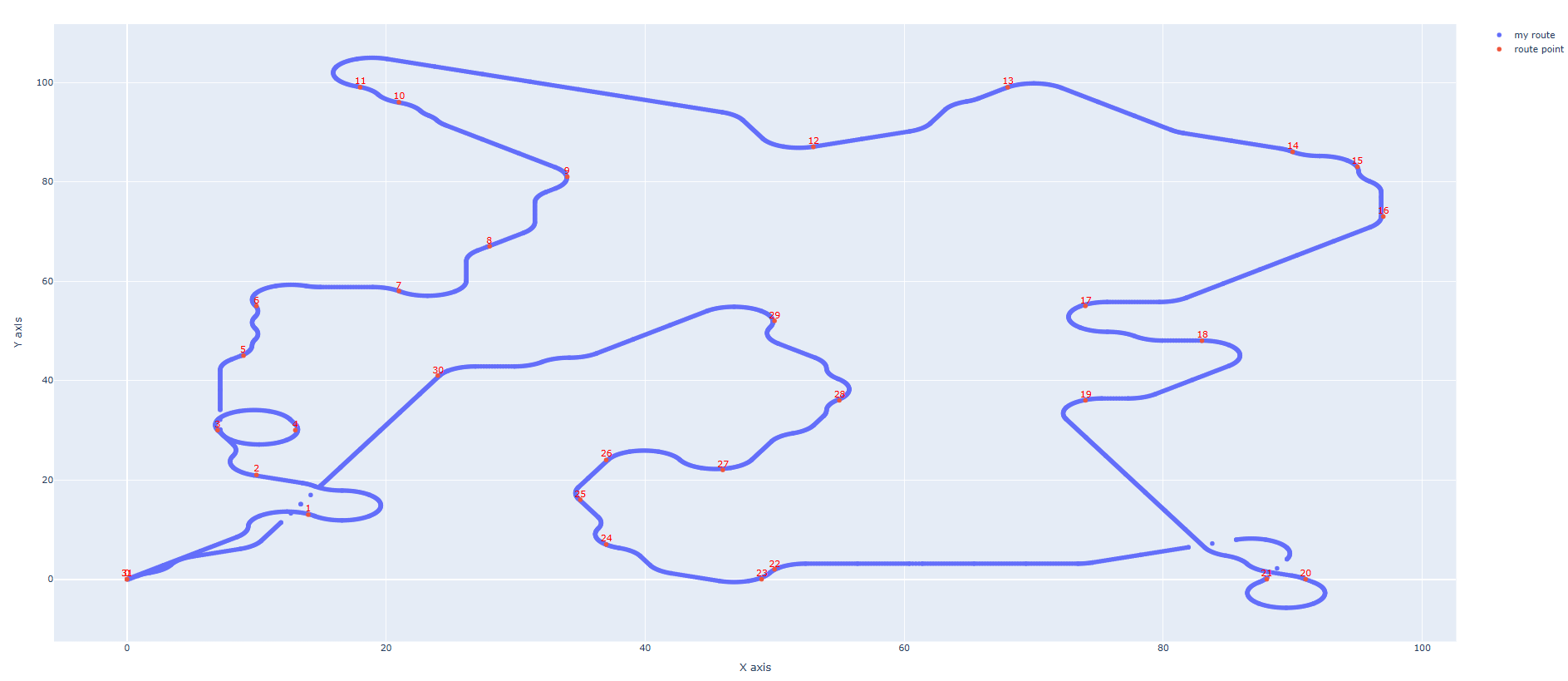


Пересечения нет если что.



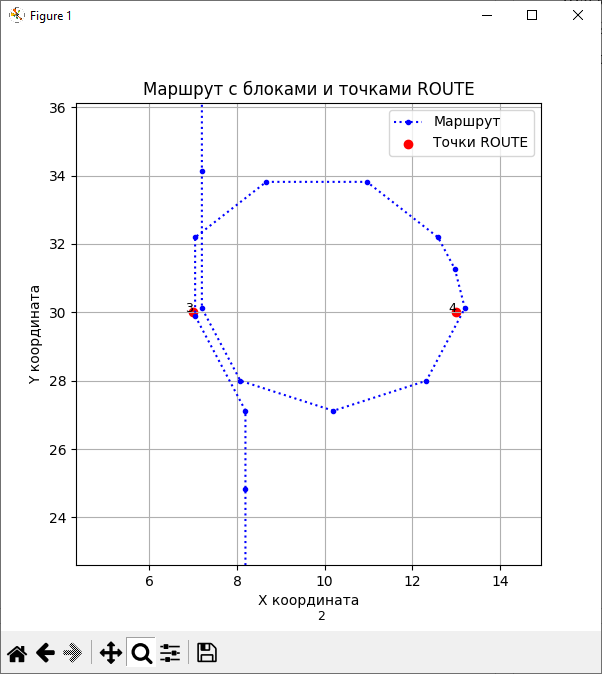
# Тестовый файл №2

Запускать с ограниченными элементами нет смысла, их в любом случае не хватит, поэтому сразу без лимита.

На графике видно пересечение около маршрутной точки 3 хоть там и стоит блок для пересечений. Это из-за количества точек при проверке пересечений. Т.к. там блок поворотный, а при проверке учитываются только начальная и конечная координаты. Увеличим кол-во точек – пересечение обнаружится, но будет дольше все считать.

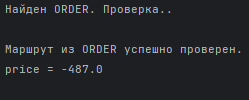
В начале/конце пути тоже самое.

Вот как это выглядит при расчетах:



# Тестовый файл №9

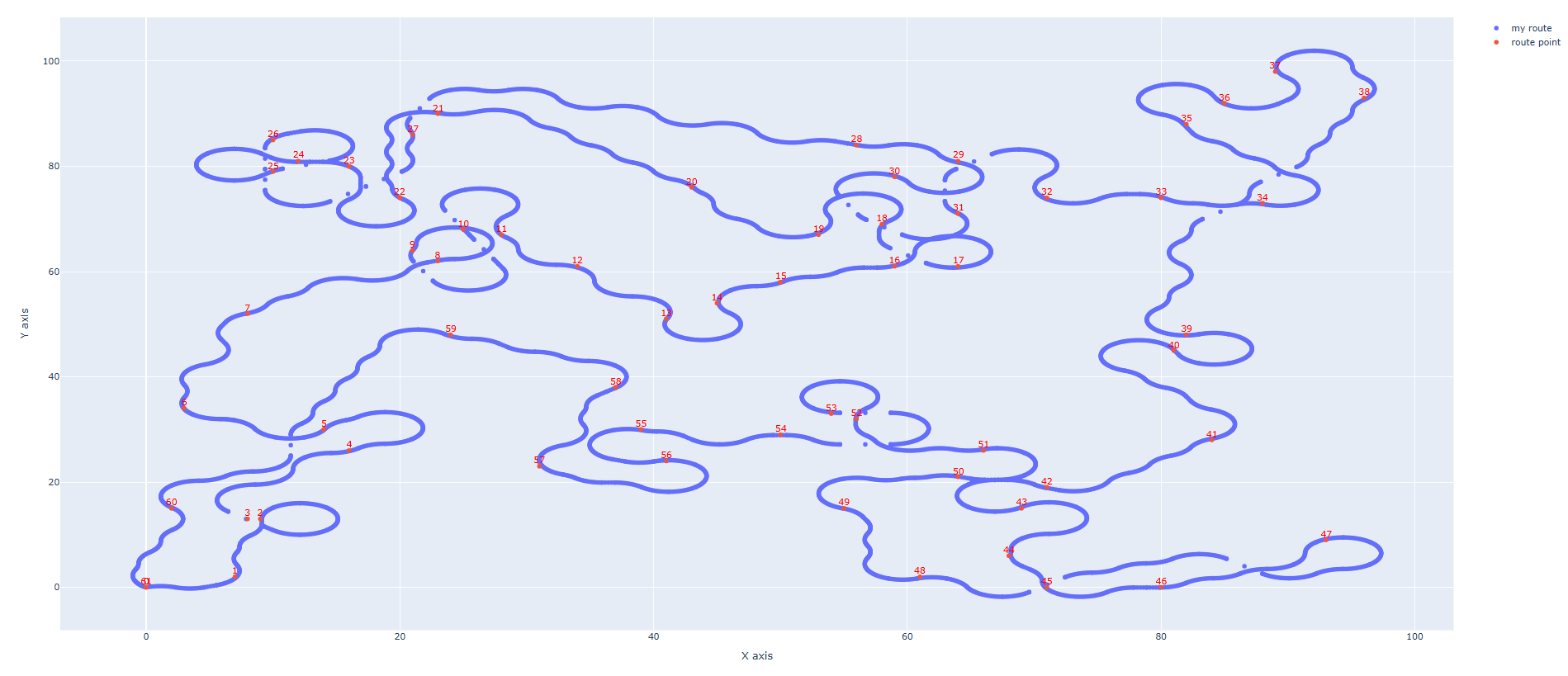
Результаты проверки ORDER:



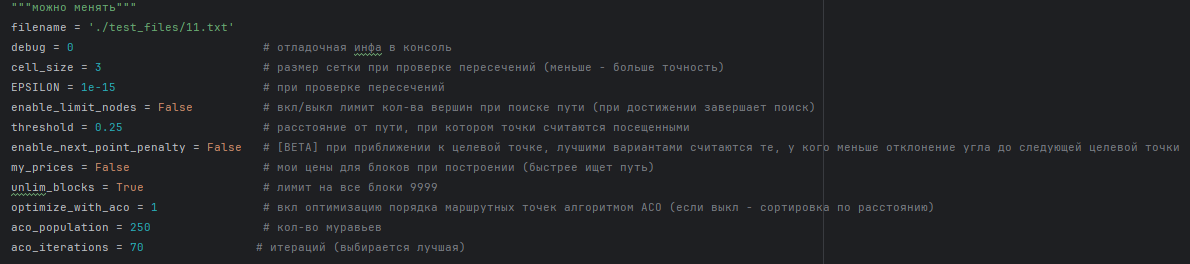
Тут рассказывать нечего

# Тестовый файл №11

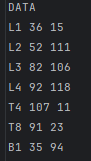
Построенный маршрут



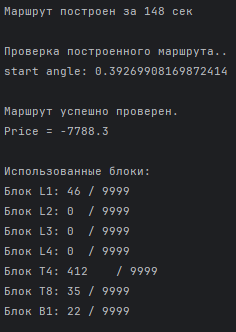
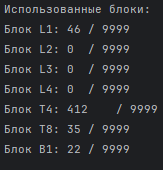
Параметры запуска



Данный маршрут получился таким волнистым из-за цены блоков.



Поворотные блоки сильно дешевле чем прямые, поэтому в основном они и использовались.

В общем близко расположенные точки это прям беда. Из-за них появляются петли, в которых куча потенциальных пересечений и других проблем.