Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 3

на тему: «Задача коммивояжёра. Алгоритм роя частиц»

Студент ИУ7-13М _____ Шемякин А. А.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Строганов Ю. В.
(Подпись, дата) (И. О. Фамилия)

2022 г.

1 Теоретический раздел

1.1 Метод роя частиц

Метод роя частиц — метод численной оптимизации, для использования которого не требуется знать точного градиента оптимизируемой функции.

Основные идеи:

- Инерция. Склонность придерживаться старых способов, которые оказались успешными в прошлом. «Я всегда так делал и буду продолжать так делать».
- Влияние общества. Старание подрожать частицам добившемся успеха используя их подходы. «Если это сработало для них, то, возможно, это сработает и для меня».
- Влияние соседей. Соседи и близкие частицы имеют на нас более сильное влияние, чем отдаленные.

1.2 Задача коммивояжёра

Задача, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. Допустимый маршрут в незамкнутой задаче коммивояжера – это маршрут, в котором все n городов посещаются ровно один раз. Допустимый маршрут в замкнутой задаче коммивояжера – это маршрут, который начинается и заканчивается в том же городе и который проходит через другие (n – 1) городов ровно один раз. В задаче коммивояжера мы стараемся минимизировать общее расстояние. Предположим, что n городов в незамкнутой задаче коммивояжера перечислены в порядке

$$x_1 \to x_2 \to \cdots \to x_n \tag{1.1}$$

Тогда общее расстояние равняется

$$D_T = \sum_{i=1}^{n-1} D(x_i, x_{i+1}) \tag{1.2}$$

2 Практический раздел

2.1 Инициализация

Мы можем значительно увеличить шансы найти хорошее решение, если при инициализации будем использовать стратегию ближайшего соседа. Чтобы более грамотно инициализировать эволюционный алгоритм будем применять алгоритм жадной инициализации на основе ближайшего соседа только для одной частицы в популяции.

Параметры алгоритма

1. Количество узлов (городов)
2. Количество итераций
3. Количество частиц
4. Социальный коэффициент
5. Когнитивный коэффициент

2.2 Реализация

На рисунках 2.1 и 2.2 продемонстрирована работа приложения с разными входными параметрами.

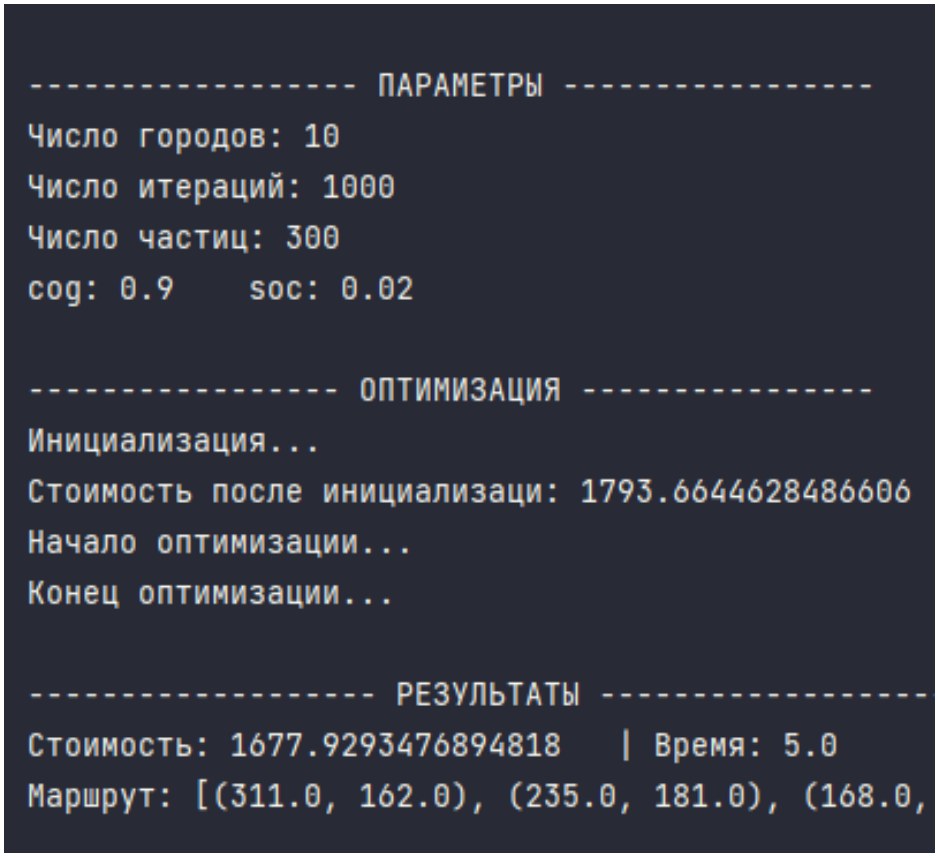

Рисунок 2.1 – Работа программы с разными параметрами

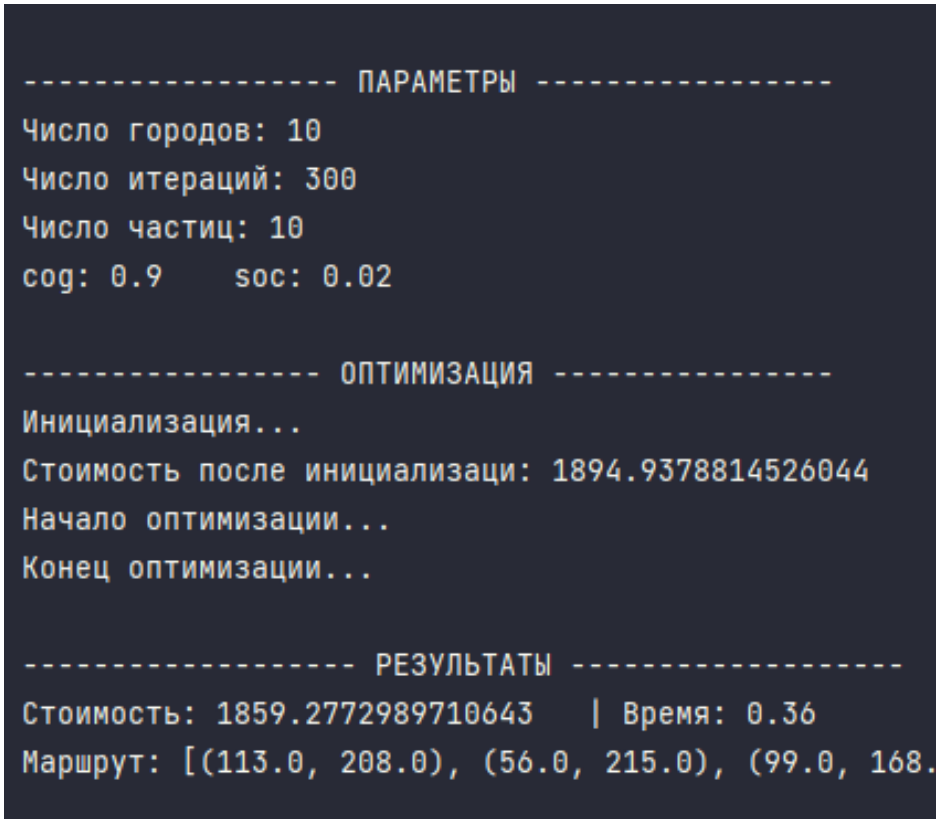

Рисунок 2.2 – Работа программы с разными параметрами

На рисунках 2.3 и 2.4 продемонстрирована визуализация маршрута с помощью библиотеки matplotlib. Координаты городов были взяты с реальной карты. Москва, Екатеринбург, Санкт-Петербург, Казань, Норильск, Магадан, Алматы, Новосибирск, Вологда, Омск.

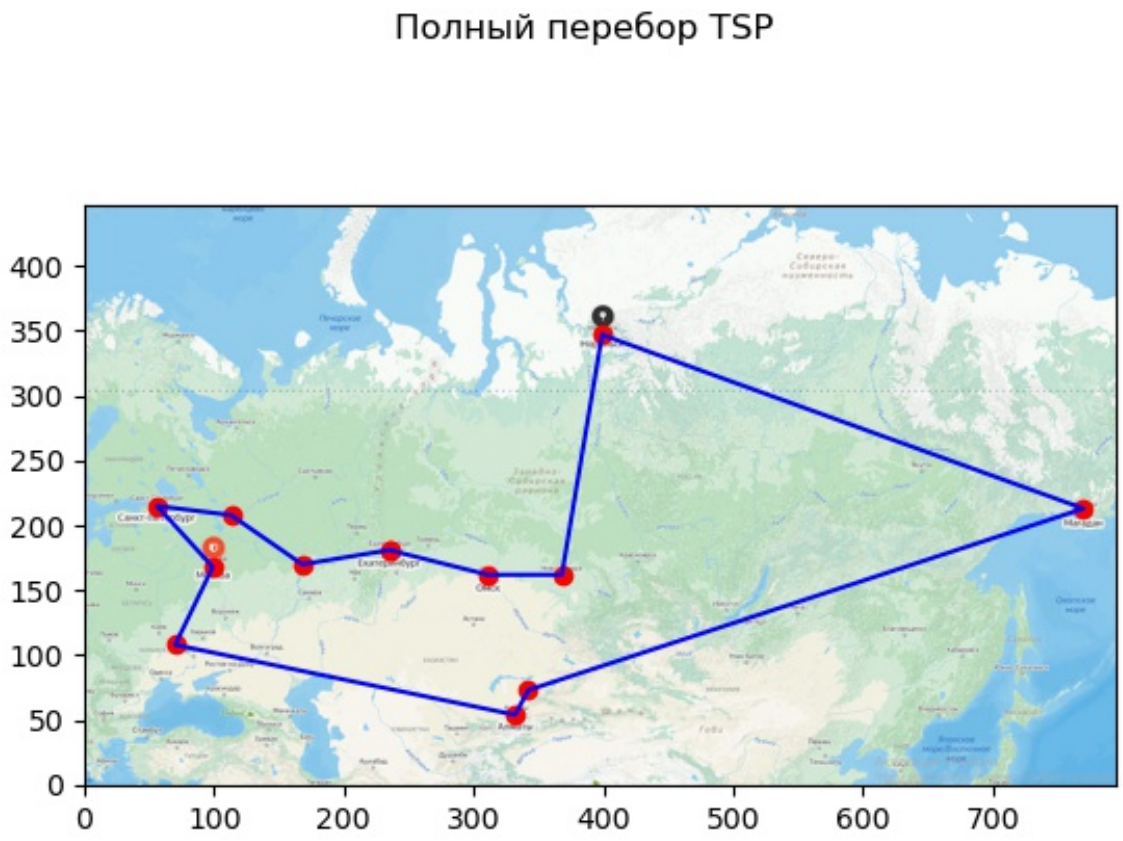

Рисунок 2.3 – Маршрут по 12 городам полным перебором

Рисунок 2.4 – Маршрут по 10 городам методом роя частиц

На рисунках 2.5 продемонстрирован график зависимости расстояния (стоимости) и количества итераций при использовании метода роя частиц (10 узлов, 300 частиц,).

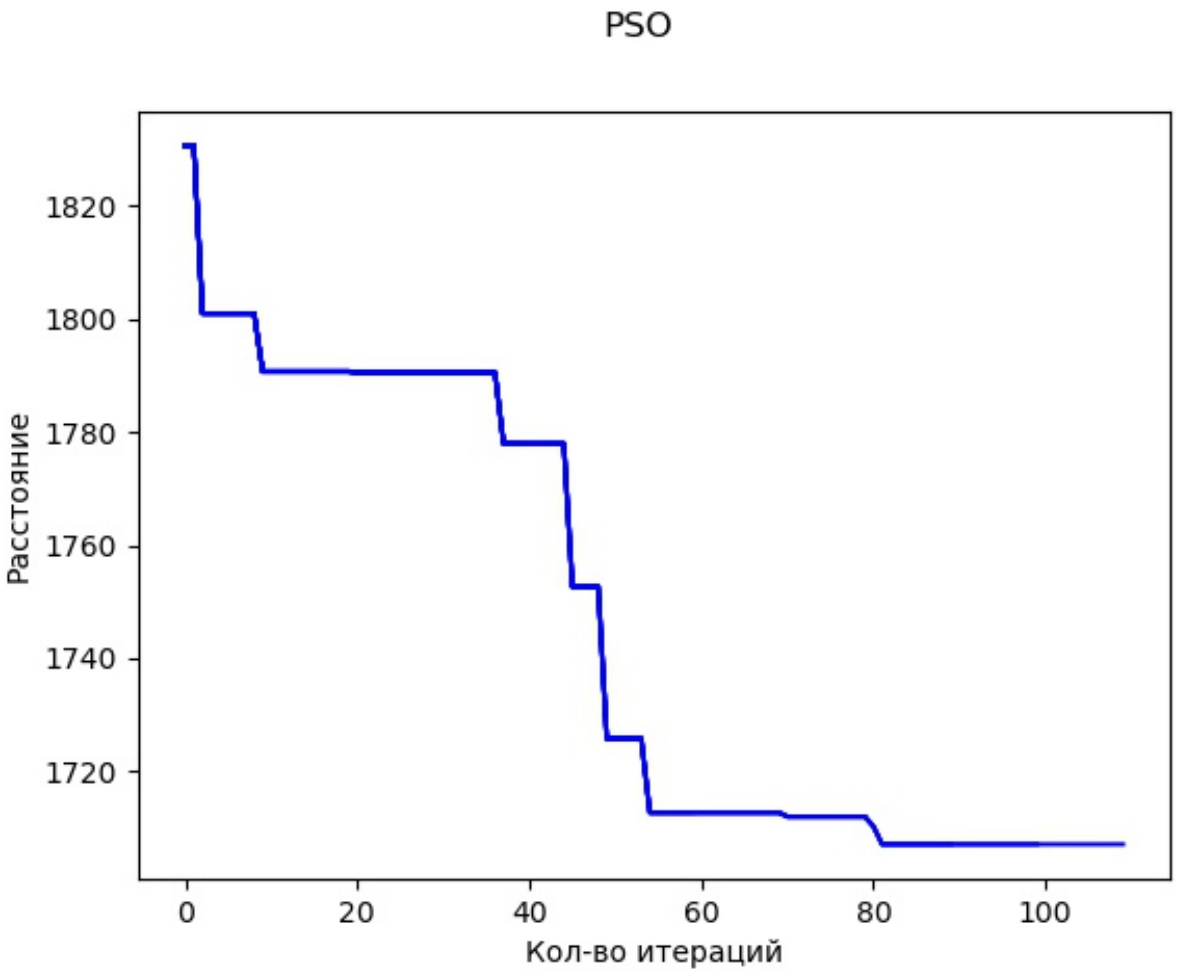

Рисунок 2.5 – График зависимости расстояния и кол-во итераций

2.3 Оценка качества работы

Для оценки качества работы использовались следующие параметры:

Кол-во узлов: 10

Кол-во итераций: 1000

Кол-во частиц: 300

Когнитивный коэффициент: 0.9

Социальный коэффициент: 0.02

Среднее время выполнения в секундах за 10 прогонов

Метод/Узлов	3	5	8	10	12
Полный перебор	0.0	0.0004	0.23	25.5	3905
Полный перебор	2.3	3.0	3.8	5.3	3.5

Средняя стоимость за 100 прогонов

Итераций/Узлов	5	8	10	12
1	478	2379	1829	2004
5	478	2379	1803	1997
10	478	2379	1791	1995
15	478	2379	1767	1984
20	478	2379	1759	1978
30	478	2379	1731	1956
50	478	2379	1713	1909
100	478	2379	1690	1830
200	478	2379	1681	1797
500	478	2379	1678	1787
1000	478	2379	1677	1779

3 Вывод

Решение полным перебором для 12 узлов происходит за 3905 сек, а используя метод роя частиц за 3.5 сек, скорость возрасла в 1115 раз. Например, полный перебор 13 узлов компьютер может считать неделю или даже больше, но МРЧ с этим быстро справится.

ПРИЛОЖЕНИЕ А

На листингах представлен исходный код программ на языке программирования Python.

Листинг А.1 – Исходный код pso

```
import random
import time

import matplotlib.pyplot as plt

from particle import Particle
from util import read_cities, visualize


class PSO:

    def __init__(self, iterations, particles_count, soc=1.0,
       cog=1.0, cities=None, verbose=True):
        self.cities = cities
        self.soc_best = None
        self.soc_cost_iter = []
        self.iterations = iterations
        self.particles_count = particles_count
        self.particles = []
        self.soc = soc
        self.cog = cog
        self.verbose = verbose

        if self.verbose:
            print()
            print("------------------ ПАРАМЕТРЫ -----------------")
            print("Число городов:", len(self.cities))
            print("Число итераций:", self.iterations)
            print("Число частиц:", self.particles_count)
            print("cog: {}\tsoc: {}".format(self.cog, self.soc))
            print()
            print("----------------- ОПТИМИЗАЦИЯ ----------------")
```

```
32              print("Инициализация...")
33          solutions = self.init_particles()
34          self.particles = [Particle(path=i) for i in solutions]
35
36      # возвращает рандомный маршрут по городам
37      def random_path(self):
38          return random.sample(self.cities, len(self.cities))
39
40      def init_particles(self):
41          random_population = [self.random_path() for _ in
                range(self.particles_count - 1)]
42          greedy_population = [self.greedy_path(0)]
43          return [*random_population, *greedy_population]
44
45      def greedy_path(self, start_index):
46          unvisited = self.cities[:]
47          del unvisited[start_index]
48          path = [self.cities[start_index]]
49          while len(unvisited):
50              index, nearest_city = min(enumerate(unvisited),
                    key=lambda item: item[1].distance(path[-1]))
51              path.append(nearest_city)
52              del unvisited[index]
53          return path
54
55      def run(self):
56          start = time.time()
57          self.soc_best = min(self.particles, key=lambda p:
                p.cog_best_cost)
58          print(f"Стоимость после инициализации:
                {self.soc_best.cog_best_cost}")
59
60          if self.verbose:
61              print("Начало оптимизации...")
62          plt.ion()
63          plt.draw()
64          for i in range(self.iterations):
65              self.soc_best = min(self.particles, key=lambda p:
                    p.cog_best_cost)
```

```
            # Чтобы посмотреть рилтайм изменения
            if i % 10 == 0:
                visualize(self, pso, i)
            self.soc_cost_iter.append(self.soc_best.cog_best_cost)

            for particle in self.particles:
                particle.clear_velocity()
                temp_velocity = []
                soc_best = self.soc_best.cog_best[:]
                new_path = particle.path[:]

                for i in range(len(self.cities)):
                    if new_path[i] != particle.cog_best[i]:
                        swap = (i,
                                particle.cog_best.index(new_path[i]),
                                self.cog)
                        temp_velocity.append(swap)
                        new_path[swap[0]], new_path[swap[1]] = \
                            new_path[swap[1]], new_path[swap[0]]

                for i in range(len(self.cities)):
                    if new_path[i] != soc_best[i]:
                        swap = (i, soc_best.index(new_path[i]),
                                self.soc)
                        temp_velocity.append(swap)
                        soc_best[swap[0]], soc_best[swap[1]] =
                            soc_best[swap[1]], soc_best[swap[0]]

                particle.velocity = temp_velocity

                for swap in temp_velocity:
                    if random.random() <= swap[2]:
                        new_path[swap[0]], new_path[swap[1]] = \
                            new_path[swap[1]], new_path[swap[0]]

                particle.path = new_path
                particle.update_costs_and_cog_best()
        if self.verbose:
            time.sleep(0.2)
```

```
101             print("Конец оптимизации...")
102             print()
103             print("------------------- РЕЗУЛЬТАТЫ
                    --------------------")
104         end = time.time() - start
105         print(f'Стоимость: {pso.soc_best.cog_best_cost}\t| Время:
                {round(end, 2)}')
106         print(f'Маршрут: {pso.soc_best.cog_best}')
107
108
109 if __name__ == "__main__":
110     sum = 0
111     cities_count = 10
112     cities = read_cities(cities_count)
113     i = 0
114     repeat = 1
115     while i < repeat:
116         pso = PSO(iterations=1000, particles_count=300, cog=0.9,
                soc=0.1, cities=cities, verbose=True)
117         pso.run()
118         x_list, y_list = [], []
119         for city in pso.soc_best.cog_best:
120             x_list.append(city.x)
121             y_list.append(city.y)
122         x_list.append(pso.soc_best.cog_best[0].x)
123         y_list.append(pso.soc_best.cog_best[0].y)
124
125         fig = plt.figure(1)
126         fig.suptitle('МРЧ задача Коммивояжёра')
127         plt.plot(x_list, y_list, 'r')
128         plt.plot(x_list, y_list)
129         plt.show()
130         plt.pause(20)
131         i = i + 1
132         sum = sum + pso.soc_best.cog_best_cost
```

Листинг А.2 – Исходный код util

```
import math

import matplotlib.pyplot as plt


class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # вычисляет гипотенузу треугольника с катетами X и Y (дистанцию)
    def distance(self, city):
        return math.hypot(self.x - city.x, self.y - city.y)

    def __repr__(self):
        return f"({self.x}, {self.y})"


def read_cities(size):
    cities = []
    with open(f'data/cities_{size}.data', 'r') as handle:
        lines = handle.readlines()
        for line in lines:
            x, y = map(float, line.split())
            cities.append(City(x, y))
    return cities


def path_cost(path):
    return sum([city.distance(path[index - 1]) for index, city in enumerate(path)])


def visualize_tsp(title, cities):
    fig = plt.figure()
    fig.suptitle(title)
```

```
36      x_list, y_list = [], []
37      for city in cities:
38          x_list.append(city.x)
39          y_list.append(city.y)
40      x_list.append(cities[0].x)
41      y_list.append(cities[0].y)
42      plt.plot(x_list, y_list, 'ro')
43      plt.plot(x_list, y_list, 'b')
44      datafile = 'img/russia.jpg'
45      img = plt.imread(datafile)
46      plt.imshow(img, zorder=0, extent=[0, 795, 0, 447])
47      plt.show()
48
49
50  def visualize(self, pso, i):
51      plt.figure(0)
52      plt.plot(pso.soc_cost_iter, 'b')
53      plt.ylabel('Расстояние')
54
55      plt.xlabel('Кол-во итераций')
56      fig = plt.figure(0)
57      fig.suptitle('PSO')
58      x_list, y_list = [], []
59      for city in self.soc_best.cog_best:
60          x_list.append(city.x)
61          y_list.append(city.y)
62      x_list.append(pso.soc_best.cog_best[0].x)
63      y_list.append(pso.soc_best.cog_best[0].y)
64      fig = plt.figure(1)
65      fig.clear()
66      fig.suptitle(f'PSO TSP (кол-во итераций {i})')
67      plt.xlabel('Позиция узла по X')
68      plt.ylabel('Позиция узла по Y')
69      plt.plot(x_list, y_list, 'b')
70      datafile = 'img/russia.jpg'
71      img = plt.imread(datafile)
72      plt.imshow(img, zorder=0, extent=[0, 795, 0, 447])
73      plt.draw()
74      plt.pause(.0001)
```

Листинг А.3 – Исходный код bruteXforce

```python
import itertools
import time

from util import read_cities, path_cost, visualize_tsp


class BruteForce:
    def __init__(self, cities):
        self.cities = cities

    def run(self):
        self.cities = min(itertools.permutations(self.cities),
           key=lambda path: path_cost(path))
        return path_cost(self.cities)


if __name__ == "__main__":

    brute = BruteForce(read_cities(12))
    start = time.time()
    brute.run()
    end = time.time() - start
    print(f'Стоимость: {path_cost(brute.cities)}\t| Время:
       {round(end, 12)}')

    visualize_tsp('Полный перебор TSP', brute.cities)
```

Листинг А.4 – Исходный код программы particle

```
from util import path_cost


class Particle:
    def __init__(self, path, cost=None):
        self.path = path
        self.cog_best = path
        self.current_cost = cost if cost else self.path_cost()
        self.cog_best_cost = cost if cost else self.path_cost()
        self.velocity = []

    def clear_velocity(self):
        self.velocity.clear()

    def update_costs_and_cog_best(self):
        self.current_cost = self.path_cost()
        if self.current_cost < self.cog_best_cost:
            self.cog_best = self.path
            self.cog_best_cost = self.current_cost

    def path_cost(self):
        return path_cost(self.path)
```