

Санкт-Петербургский государственный университет

Попова Светлана Константиновна

Выпускная квалификационная работа

**Задача китайского почтальона в приложении
к организации работы муниципальной техники**

Направление 01.03.02

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель:
доцент, Кафедра математического
моделирования энергетических
систем,
кандидат физ.-мат. наук,
Лежнина Елена Александровна

Рецензент:
доцент, Кафедра технологии
программирования,
кандидат физ.-мат. наук,
Раевская Анастасия Павловна

Санкт-Петербург

2021

Содержание	
Введение	3
Обзор литературы	4
Постановка задачи	8
Задача китайского почтальона	9
Алгоритм	10
Реализация	14
Заключение	17
Список литературы	18
Приложение	23

Введение

Рациональное управление работой муниципальной техники может включать оптимизацию посещаемых маршрутов. Сокращение длин путей, проходимых техникой, позволяет уменьшать время работы транспортных средств, что может продлевать их срок службы, а также экономить топливо, что будет способствовать сохранению расходов из бюджета.

В работе муниципальной техники также важно рассматривать временные окна для запрета на парковку автомобилей. Они указываются на дорожных знаках, запрещающих парковку вдоль дороги на определённый период времени.

Целью данной работы является разработка программы для нахождения маршрутов для организации работы муниципальной техники и рекомендации временных окон для запрета на парковку. Для этого можно поставить следующие задачи: рассмотреть задачу китайского почтальона в приложении к организации работы муниципальной техники, изучить подходящий для этого алгоритм, реализовать его и рекомендовать временные окна.

Обзор литературы

Эвристические алгоритмы широко используются в различных задачах с высокой сложностью. Такие алгоритмы позволяют найти хорошее решение, но оно не всегда является лучшим, поэтому постоянно предлагаются различные методы для улучшения таких алгоритмов.

В 1989 году Меламед И. И., Сергеев С. И., Сигал И. Х. рассмотрели способы построения и оценки эффективности эвристических алгоритмов для задачи коммивояжера [1]. В 1985 году в статье Grefenstette J., Gopal, R. Rosmaita, B., Van Gucht D. были представлены некоторые подходы к применению генетического алгоритма к задаче коммивояжера [2].

M. Dorigo, U. Schnepf в 1991 году представили метод машинного обучения на основе генетических алгоритмов и неконтролируемого обучения с подкреплением для генерации и организации поведения робота [3].

В 2005 году Chan F. T. S., Chung S. H., Chan P. L. Y. представили адаптивный генетический алгоритм для задачи распределённого планирования в многопрофильной и многопродуктовой среде [4]. Также они изучали тестовые задачи для исследования оптимальных условий потоковой передачи. В 2009 году Chung S. H., Chan F. T. S., Chan H. K. предложили мо-

дифицированный подход с использованием генетического алгоритма для проблемы распределённого планирования с учётом технического обслуживания, чтобы минимизировать время выполнения работ [5]. В том же году Chan F. T. S., Wong, T. C., Chan L. Y. применили генетический алгоритм для определения условий потоковой передачи в задаче планирования работы магазина [6].

Tamer F. Ambelmaguidy и Maged M. Dessouky в 2006 году предложили новый подход к использованию генетического алгоритма в интегрированной задаче распределения запасов [7].

Dong Won Cho, Young Hae Lee, Tae Youn Lee, Mitsuo Gen в 2013 году представили адаптивный генетический алгоритм, позволяющий получить качественное решение для задачи управления запасами, зависящей от времени [8].

В 2013 году Rakesh Kumar, Girdhar Gopal, Rajesh Kumar изучили гибридизацию генетического алгоритма и локального поиска и факторы, влияющие на производительность гибридного алгоритма [9]. J. L. Deneubourg, J. M. Pasteels, J. C. Verhaeghe в 1983 году описали математическую модель поведения муравьёв во время поиска пищи и пришли к тому, что степень случайности поведения может регулироваться в соответствии с количеством и распределением пищи [10].

Alberto Colorni, Marco Dorigo, Vittorio Maniezzo в 1991 исследовали поведение муравьёв, чтобы применить к решению задач и оптимизации, таким образом они представили распределённую среду решения и предложили её для поиска решения задачи коммивояжёра. В том же году они исследовали свойства муравьиного алгоритма [11].

М. Dorigo в 1992 году рассмотрел возможность улучшения моделирования нелинейного поведения загрязнения воздуха, предложив алгоритм с использованием оптимизации муравьиного алгоритма. Такой метод привел к снижению ошибки прогнозирования и к получению более надёжных моделей [12].

В 1996 году М. Dorigo, V. Maniezzo, А. Colorni предложили муравьиный алгоритм как новый подход к стохастической комбинаторной оптимизации. Такую модель они охарактеризовали положительной обратной связью, распределёнными вычислениями и использованием конструктивной жадной эвристики. Этот метод М. Dorigo, V. Maniezzo, А. Colorni применили к задаче коммивояжёра и для демонстрации надёжности к другим задачам оптимизации: асимметричной задаче коммивояжёра, квадратичным назначениям и планированию работы магазина [13].

В 1997 году М. Dorigo, L.M. Gambardella представили му-

равьиный алгоритм, распределённый алгоритм, применённый к задаче коммивояжера. Также они сравнили этот алгоритм с другими популярными алгоритмами для симметричных и асимметричных задач коммивояжера [14].

В 1999 году Vittorio Maniezzo, Alberto Colomi применили муравьиный алгоритм к задаче квадратичного присваивания, их исследования показали, что система муравьёв конкурентоспособна на всех задачах тестирования [15].

В 2011 году Enxiu Chen и Xiyu Liu представили мультиколониальный муравьиный алгоритм, который позволяет избегать стагнаций. В результате они установили, что этот алгоритм является точным для задачи коммивояжера [16].

Xinye Chen, Ping Zhang, Guanglong Du, Fang Li в 2018 году рассмотрели проблему множественного коммивояжера с одним или несколькими депо. Они предложили меметический алгоритм, основанный на оптимизации муравьиного алгоритма и продемонстрировали сравнение с разновидностями генетических алгоритмов, в которых предложенный ими меметический алгоритм превзошёл их [17].

Постановка задачи

Проблема организации работы муниципальной техники может быть сформулирована как задача китайского почтальона. То есть, уборочной машине необходимо проехать по всем дорогам, также как китайскому почтальону обойти все улицы [18], [19].

Во время работы муниципального транспорта необходимо, чтобы на проезжей части не стояли автомобили, поэтому встречаются дорожные знаки, запрещающие парковку в определённый период времени. То есть, данная задача содержит временные окна.

Дан граф $G = (V, E)$, где $V = \{1, 2, \dots, n\}$ - множество вершин, E - множество рёбер. Граф задан матрицей расстояний (матрицей весов рёбер) $D = \{d_{ij}\}$, $i = 1, \dots, n$, $j = 1, \dots, n$. Необходимо найти маршрут, при котором будут посещены все рёбра графа. Маршрут должен начинаться и заканчиваться в одной точке. Каждое ребро должно быть пройдено в двух разных направлениях. Также необходимо на основе полученного маршрута составить временные окна для запрета на парковку транспортных средств.

Задача китайского почтальона

Задача китайского почтальона была сформулирована китайским математиком Kwan Meigu Guan. Почтальон должен обойти все улицы города, и необходимо найти маршрут, который бы минимизировал пройденное почтальоном расстояние. Для графа $G = (V, E)$, где $V = \{1, 2, \dots, n\}$ - множество вершин, E - множество рёбер, должен быть построен маршрут, содержащий каждое ребро, по крайней мере, один раз, начинающийся и заканчивающийся в одной точке.

Существуют различные варианты задачи китайского почтальона [20]. Некоторые из них описаны далее.

- 1** Задача сельского китайского почтальона с ветром. В этой задаче стоимость прохождения рёбер зависит от направления. Необходимо найти маршрут, который содержит все требуемые рёбра.
- 2** Задача китайского почтальона для ориентированного графа. Рёбра графа имеют направления и нужно найти путь, содержащий все рёбра.
- 3** k -задача китайского почтальона [21]. Здесь, вместо одного почтальона, k почтальонов должны посетить все рёбра.

Алгоритм

Задача китайского почтальона относится к задачам высокой сложности, поэтому для неё можно эффективно использовать эвристические алгоритмы.

Эвристический алгоритм – это алгоритм, содержащий практический метод, не всегда позволяющий найти точное решение, но дающий приемлемый результат, и способный ускорять получение решения. К подобным алгоритмам относится муравьиный алгоритм. Он способен адаптироваться к изменениям, варьировать точность в зависимости от требований и выдавать решение, приближенное к точному.

В живой природе муравьям удаётся находить кратчайшие пути от их колонии до источника пищи за счёт оставляемых ими феромонов. Проходя по маршруту, муравей оставляет за собой след из некоторого количества феромона. Этот муравей движется случайным образом, а следующий муравей может обнаружить след, оставленный предыдущим муравьём и с большей вероятностью выбрать этот путь, усиливая его своим феромоном. Со временем феромоны испаряются, поэтому чем более длинный маршрут, тем меньшее количество феромона на нём остаётся.

Теперь рассмотрим муравьиный алгоритм для задачи ки-

тайского почтальона. За основу муравьиного алгоритма взята работа М. Dorigo, V. Maniezzo, A. Colorni [13].

k -ый муравей выбирает ребро (i, j) с вероятностью

$$p_{ij}(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{j=1}^n [\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}, \quad (1)$$

Видимость ребра (i, j) определяется по следующей формуле

$$\eta_{ij} = \frac{1}{d_{ij}}, \quad (2)$$

где d_{ij} - вес ребра.

τ_{ij} - интенсивность феромона на ребре (i, j) в момент времени t после одной итерации алгоритма, определяется следующей формулой

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}, \quad (3)$$

где ρ - коэффициент, который представляет собой испарение $(1 - \rho)$ в промежуток времени от t до $t+n$,

$$\Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij}^k, \quad (4)$$

τ_{ij}^k - количество феромона, откладываемого k -ым муравьём на ребре (i, j) , которое высчитывается по следующей формуле

$$\Delta\tau_{ij}^k = \frac{Q}{L^k}, \quad (5)$$

где Q - константа, L^k - длина пути k -го муравья.

Интенсивность следа $\tau_{ij}(t)$ даёт информацию о том, сколько муравьев в прошлом выбрали одно и то же ребро. Видимость

η_{ij} говорит о том, что чем меньше вес ребра, тем оно более привлекательно.

Параметры α , β определяют относительную важность феромона по сравнению с видимостью. При $\alpha = 0$ выбор будет сделан в пользу самого короткого ребра. А при $\beta = 0$ выбор будет основываться только на опыте других муравьёв.

Перед началом алгоритма можно задать L^* , равной бесконечности. В алгоритме есть n итераций, в каждой итерации m муравьёв. Каждый муравей проходит по графу, пока не обойдёт все рёбра. После этого длина маршрута муравья сравнивается с L^* , если она лучше, то записывается в L^* . Таким образом можно найти хороший маршрут.

Далее мы можем обобщить муравьиный алгоритм для задачи китайского почтальона.

- 1 Задается матрица расстояний D
- 2 Инициализируются параметры α , β , Q
- 3 Инициализируются видимость η_{ij} и начальная концентрация феромона
- 3 Муравьи размещаются в вершинах
- 4 Определяется L^*
- 5 Цикл по итерациям $t = (1, n)$

5.1 Цикл по муравьям $k = (1, m)$

5.1.1 Построить маршрут муравья $T_k(t)$ и рассчитать его длину

$$L_k(t)$$

5.1.2 Если $L_k(t) < L^*$, то $L^* = L_k(t)$, $T^* = T_k(t)$

5.1.3 Цикл по всем рёбрам графа

5.1.3.1 Обновление феромонов

5 Вывод T^* , L^*

Временная сложность муравьиного алгоритма зависит от числа итераций, количества вершин графа и количества муравьёв.

Реализация

Программная реализация была написана на языке программирования Python в среде разработки PyCharm.

Для работы алгоритма задаются параметры α и β , коэффициент испарения, подаётся матрица расстояний, определяется количество муравьев и итераций алгоритма.

Функция `run()` класса `AntColony()` запускает муравьиный алгоритм. Функция `SpreadPher()` обновляет феромон. В функции `GenPathDist()` определяется длина пути муравья. Функция `GenAllPaths()` объединяет все маршруты муравьёв. В функции `GenPath()` строится маршрут одного муравья с использованием цикла, пока все рёбра не пройдены. С помощью функции `PickMove()`, основывающейся на вероятностях, выбирается вершина для передвижения муравья. На выходе программы получается маршрут, содержащий все рёбра. На основе полученного маршрута находятся временные окна для запрета на парковку.

Для примера использования алгоритма был взят участок Старого Петергофа. Данные о расстоянии были взяты из Яндекс.карт. Был построен маршрут, составляющий примерно 63 километра: [(0, 15), (15, 16), (16, 13), (13, 18), (18, 17), (17, 16), (16, 15), (15, 14), (14, 13), (13, 16), (16, 15), (15, 14), (14, 13), (13, 18), (18, 17), (17, 22), (22, 21), (21, 20), (20, 19), (19, 11), (11, 12),

(12, 8), (8, 9), (9, 10), (10, 11), (11, 12), (12, 8), (8, 7), (7, 2), (2, 4), (4, 3), (3, 5), (5, 14), (14, 13), (13, 12), (12, 11), (11, 19), (19, 18), (18, 17), (17, 16), (16, 15), (15, 14), (14, 13), (13, 16), (16, 17), (17, 18), (18, 19), (19, 20), (20, 21), (21, 18), (18, 13), (13, 16), (16, 17), (17, 18), (18, 21), (21, 20), (20, 19), (19, 11), (11, 19), (19, 20), (20, 21), (21, 22), (22, 17), (17, 16), (16, 15), (15, 14), (14, 13), (13, 18), (18, 19), (19, 11), (11, 10), (10, 9), (9, 8), (8, 12), (12, 11), (11, 19), (19, 11), (11, 10), (10, 9), (9, 8), (8, 12), (12, 11), (11, 19), (19, 11), (11, 12), (12, 8), (8, 7), (7, 2), (2, 1), (1, 0), (0, 15), (15, 14), (14, 5), (5, 6), (6, 4), (4, 2), (2, 1), (1, 3), (3, 4), (4, 6), (6, 5), (5, 3), (3, 1), (1, 0), (0, 15), (15, 16), (16, 17), (17, 22), (22, 21), (21, 18), (18, 13), (13, 14), (14, 15), (15, 16), (16, 13), (13, 14), (14, 5), (5, 3), (3, 1), (1, 2), (2, 7), (7, 8), (8, 12), (12, 13), (13, 16), (16, 15), (15, 0), (0, 1), (1, 0)]

Рекомендуемые временные окна для каждого ребра: (0, 15): ('06:00', '07:00'), (15, 16): ('06:00', '07:00'), (16, 13): ('06:00', '07:00'), (13, 18): ('06:00', '07:00'), (18, 17): ('06:00', '07:00'), (17, 16): ('06:00', '07:00'), (16, 15): ('06:00', '07:00'), (15, 14): ('06:00', '07:00'), (14, 13): ('06:00', '07:00'), (13, 16): ('06:00', '07:00'), (17, 22): ('06:00', '07:00'), (22, 21): ('06:00', '07:00'), (21, 20): ('06:00', '07:00'), (20, 19): ('06:00', '07:00'), (19, 11): ('06:00', '07:00'), (11, 12): ('06:00', '07:00'), (12, 8): ('06:00', '07:00'), (8, 9): ('06:00',

'07:00'), (9, 10): ('06:00', '07:00'), (10, 11): ('06:00', '07:00'), (8, 7): ('06:00', '07:00'), (7, 2): ('06:00', '07:00'), (2, 4): ('06:00', '07:00'), (4, 3): ('06:00', '07:00'), (3, 5): ('06:00', '07:00'), (5, 14): ('06:00', '07:00'), (13, 12): ('06:00', '07:00'), (12, 11): ('06:00', '07:00'), (11, 19): ('06:00', '07:00'), (19, 18): ('06:00', '07:00'), (16, 17): ('07:00', '08:00'), (17, 18): ('07:00', '08:00'), (18, 19): ('07:00', '08:00'), (19, 20): ('07:00', '08:00'), (20, 21): ('07:00', '08:00'), (21, 18): ('07:00', '08:00'), (18, 13): ('07:00', '08:00'), (18, 21): ('07:00', '08:00'), (21, 22): ('07:00', '08:00'), (22, 17): ('07:00', '08:00'), (11, 10): ('07:00', '08:00'), (10, 9): ('07:00', '08:00'), (9, 8): ('07:00', '08:00'), (8, 12): ('07:00', '08:00'), (2, 1): ('08:00', '09:00'), (1, 0): ('08:00', '09:00'), (14, 5): ('08:00', '09:00'), (5, 6): ('08:00', '09:00'), (6, 4): ('08:00', '09:00'), (4, 2): ('08:00', '09:00'), (1, 3): ('08:00', '09:00'), (3, 4): ('08:00', '09:00'), (4, 6): ('08:00', '09:00'), (6, 5): ('08:00', '09:00'), (5, 3): ('08:00', '09:00'), (3, 1): ('08:00', '09:00'), (13, 14): ('08:00', '09:00'), (14, 15): ('08:00', '09:00'), (1, 2): ('09:00', '10:00'), (2, 7): ('09:00', '10:00'), (7, 8): ('09:00', '10:00'), (12, 13): ('09:00', '10:00'), (15, 0): ('09:00', '10:00'), (0, 1): ('09:00', '10:00').

Заключение

В данной работе проблема организации муниципальной техники рассматривается как задача китайского почтальона. В ходе исследования были изучены задача китайского почтальона и применимый для неё алгоритм. Был реализован муравьиный алгоритм на языке программирования Python и найден маршрут, включающий все улицы, и были рекомендованы временные окна. В дальнейшем, исследование может быть продолжено рассмотрением оптимизаций муравьиного алгоритма для поиска наиболее хорошего решения и ускорения работы программы.

Список литературы

- [1] Меламед И. И., Сергеев С. И., Сигал И. Х. Задача коммивояжера. Приближенные алгоритмы // Автомат. и телемех. 1989. №11. 3-26. Autom. Remote Control. 50:11. 1459-1479.
- [2] Grefenstette J., Gopal R., Rosinaita B. and van Gucht D. Genetic Algorithms for the Traveling Salesman Problem // Proc. Int. Conf. Genetic Algorithms and Their Applications. 1985. 160-168.
- [3] Dorigo M., Schnepf U. Organisation of robot behavior through genetic learning processes // Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments Advanced Robotics. 1991.
- [4] Chang F.T.S., Chung S.H., Chang P.L.Y. An adaptive genetic algorithm with dominated genes for distributed scheduling problems // Expert Systems With Applications. 2005. 29(2):364-371.
- [5] Chung S. H., Chan F. T. S., Chan H. K. A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling // Engineering applications of artificial intelligence. 2009. v. 22, no. 7, p. 1005-1014.

- [6] Chan F. T. S., Wong T. C., Chan L. Y., The application of genetic algorithms to lot streaming in a job-shop scheduling problems // International Journal of Production Research. 2009. Vol. 47 Issue 12, p3387-3412, 26p.
- [7] Tamer F. Ambelmagui, Maged M. Dessouky. A genetic algorithm approach to the integrated inventory-distribution problem // International Journal of Production Research. 2006. Vol. 44 Issue 21, p4445-4464, 20p.
- [8] Dong Won Cho, Young Hae Lee, Tae Youn Lee, Mitsuo Gen. An adaptive genetic algorithm for the time dependent inventory routing problem// Journal of Intelligent Manufacturing. 2014. 25(5):1025-1042.
- [9] Rakesh Kumar, Girdhar Gopal, Rajesh Kumar. Hibridization in genetic algorithms // International Journal of Advanced Research in Computer Science and Software Engineering. 2013. Vol-3, Issue-4, pp 403-409.
- [10] DeneubourgJ. L., PasteelsJ., VerhaegheJ. C. Probablistic behaviour in ants: A strategy of errors? // Journal of theoretical biology. 1983. 105, 2, page (259-271).
- [11] Alberto Coloni, Marco Dorigo, Vittorio Maniezzo, Distributed optimization by ant colonies // Proceedings of the First

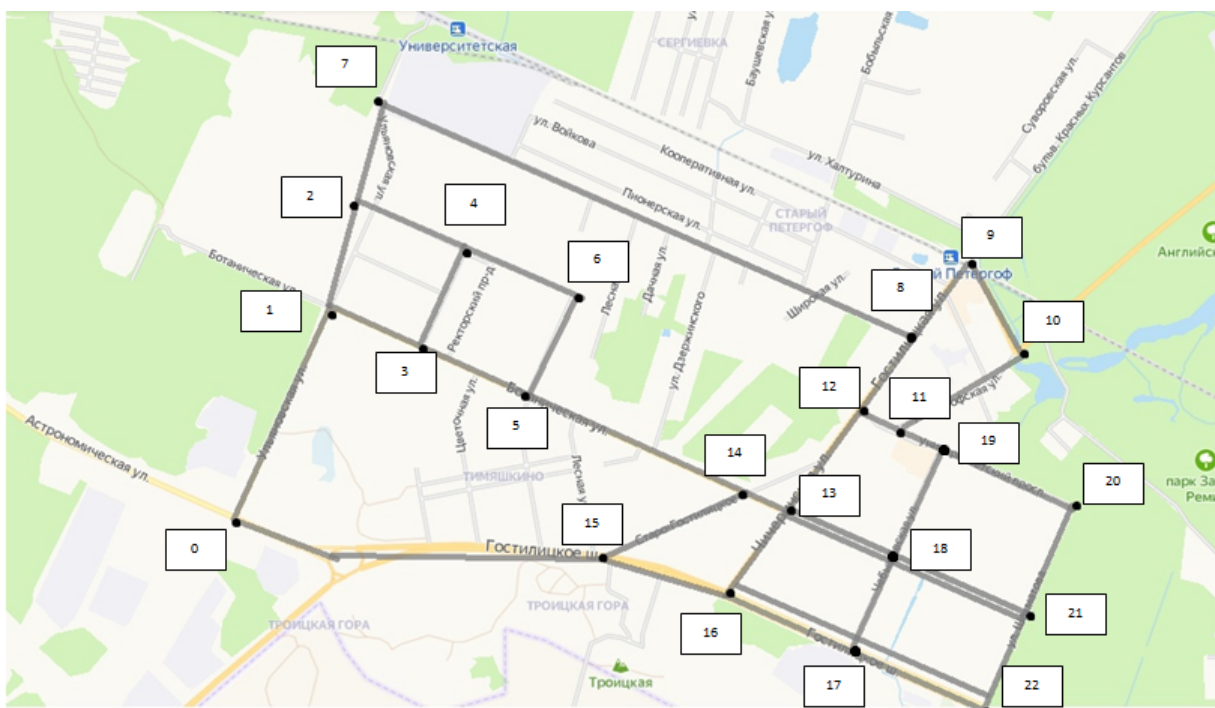
- European Conference on Artificial Life. Elsevier Publishing. 1991. 134–142.
- [12] Dorigo M. Optimization, learning and natural algorithms // *Elettronica e Informazione*, Politecnico di Milano, Italy. 1992.
- [13] Dorigo M., Maniezzo V., Coloni A. Ant system: optimization by a colony of cooperating agents // *Transactions on Systems, Man, and Cybernetics*. 1996. 26(1):29-41.
- [14] Dorigo M., Gambardella L. M. Ant colony system: a cooperative learning approach to the traveling salesman problems // *Transactions on Evolutionary Computation*. 1997. 1(1):53-66.
- [15] Maniezzo V., Coloni A. The ant system applied to the quadratic assignment problem // *Transactions on Knowledge and Data Engineering*. 1999. 11(5):769-778.
- [16] Enxiu Chen, Xiyu Liu. Multi-colony ant algorithm, In *Ant Colony Optimization-Methods and Applications* // Pengeditanoleh Avi Ostfeld. InTech. 2011.
- [17] Xinye Chen, Ping Zhang, Guanglong Du, Fang Li. Ant colony optimization based memetic algorithm to solve bi-objective multiple traveling salesmen problem for multy-robot systems // *Access*. 2018. 6:21745-21757.

- [18] Задача китайского почтальона.
<https://xlinux.nist.gov/dads/HTML/chinesePostman.html>
- [19] Задача китайского почтальона
http://scask.ru/j_book_graph.php?id=93
- [20] Gordenko M. K., Avdoshin S. M. Variants of Chinese Postman Problems and a Way of Solving through Transformation into Vehicle Routing Problems // Proceedings of the Institute for System Programming of RAS. 2018. 30(3):221-232
- [21] İbrahim Zeki Akyurt, Timur Keskindurk, Çağatay Kalkanlı. Using Genetic Algorithm For Winter Maintenance Operations: Multi Depot K-Chinese Postman Problem. Emerging Markets Journal. 2015.
- [22] Дроздов А.А., Баженов Р.И. Нахождение оптимального пути с помощью муравьиного алгоритма в задаче коммивояжера // Постулат. 2018.
- [23] Dorigo M., Maniezzo V., Colomi A. Distributed Optimization by Ant Colonies // European Conference on Artificial Life. Elsevier Publishing. 1991. 134–142.
- [24] Dorigo M., Gambardella, L.M. Ant Colony System: A cooperative learning approach to the Traveling Salesman

Problem // Transactions on Evolutionary Computation. 1(1):53
- 66.

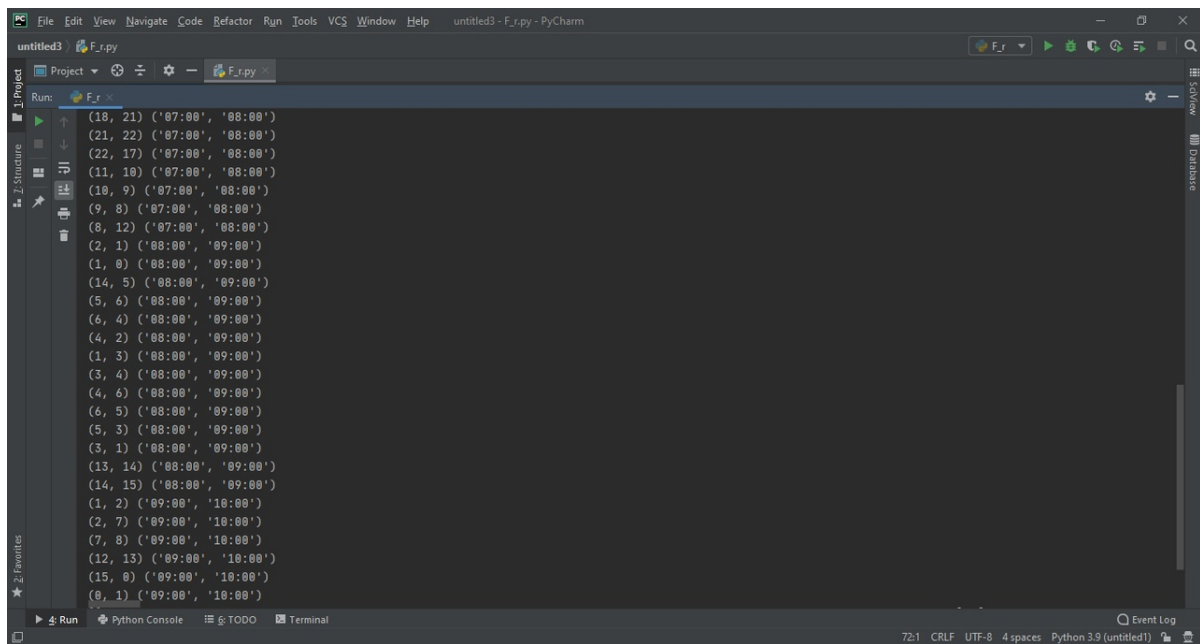
Приложение

Приложение 1. Участок Старого Петергофа



Приложение 2. Результаты программы

```
C:\Users\acer\PycharmProjects\untitled1\venv\Scripts\python.exe C:/Users/acer/PycharmProjects/untitled3/F_r.py
[[ (0, 15), (15, 16), (16, 13), (13, 18), (18, 17), (17, 16), (16, 15), (15, 14), (14, 13), (13, 16), (16, 15), (15, 14), (14, 13), (13, 18), (18, 17), (17, 16), 62560.0
(0, 15) ('06:00', '07:00')
(15, 16) ('06:00', '07:00')
(16, 13) ('06:00', '07:00')
(13, 18) ('06:00', '07:00')
(18, 17) ('06:00', '07:00')
(17, 16) ('06:00', '07:00')
(16, 15) ('06:00', '07:00')
(15, 14) ('06:00', '07:00')
(14, 13) ('06:00', '07:00')
(13, 16) ('06:00', '07:00')
(17, 22) ('06:00', '07:00')
(22, 21) ('06:00', '07:00')
(21, 20) ('06:00', '07:00')
(20, 19) ('06:00', '07:00')
(19, 11) ('06:00', '07:00')
(11, 12) ('06:00', '07:00')
(12, 8) ('06:00', '07:00')
(8, 9) ('06:00', '07:00')
(9, 10) ('06:00', '07:00')
(10, 11) ('06:00', '07:00')
(8, 7) ('06:00', '07:00')
(7, 2) ('06:00', '07:00')
(2, 4) ('06:00', '07:00')
(4, 3) ('06:00', '07:00')
(4, 3) ('07:00', '08:00')
(3, 5) ('07:00', '08:00')
(5, 14) ('07:00', '08:00')
(13, 12) ('07:00', '08:00')
(12, 11) ('07:00', '08:00')
(11, 19) ('07:00', '08:00')
(19, 18) ('07:00', '08:00')
(16, 17) ('07:00', '08:00')
(17, 18) ('07:00', '08:00')
(18, 19) ('07:00', '08:00')
(19, 20) ('07:00', '08:00')
(20, 21) ('07:00', '08:00')
(21, 18) ('07:00', '08:00')
(18, 13) ('07:00', '08:00')
(18, 21) ('07:00', '08:00')
(21, 22) ('07:00', '08:00')
(22, 17) ('07:00', '08:00')
(11, 10) ('07:00', '08:00')
(10, 9) ('07:00', '08:00')
(9, 8) ('07:00', '08:00')
(8, 12) ('07:00', '08:00')
(2, 1) ('08:00', '09:00')
(1, 0) ('08:00', '09:00')
(14, 5) ('08:00', '09:00')
(5, 6) ('08:00', '09:00')
(6, 4) ('08:00', '09:00')
(4, 2) ('08:00', '09:00')
(4, 2) ('08:00', '09:00')
```

```
untitled3 - F.r.py - PyCharm
Project: F.r.py
Run: F.r.py
(18, 21) ('07:00', '08:00')
(21, 22) ('07:00', '08:00')
(22, 17) ('07:00', '08:00')
(11, 10) ('07:00', '08:00')
(10, 9) ('07:00', '08:00')
(9, 8) ('07:00', '08:00')
(0, 12) ('07:00', '08:00')
(2, 1) ('08:00', '09:00')
(1, 0) ('08:00', '09:00')
(14, 5) ('08:00', '09:00')
(5, 6) ('08:00', '09:00')
(4, 4) ('08:00', '09:00')
(4, 2) ('08:00', '09:00')
(1, 3) ('08:00', '09:00')
(3, 4) ('08:00', '09:00')
(4, 6) ('08:00', '09:00')
(6, 5) ('08:00', '09:00')
(5, 3) ('08:00', '09:00')
(3, 1) ('08:00', '09:00')
(13, 14) ('08:00', '09:00')
(14, 15) ('08:00', '09:00')
(1, 2) ('09:00', '10:00')
(2, 7) ('09:00', '10:00')
(7, 8) ('09:00', '10:00')
(12, 13) ('09:00', '10:00')
(15, 0) ('09:00', '10:00')
(0, 1) ('09:00', '10:00')
```

Приложение 3. Код программы

```
import random as rn
import numpy as np
from numpy.random import choice as np_choice
import math
import time

class AntColony(object):

    def __init__(self, distances, n_ants, n_best,
n_iterations, decay, E_all, alpha=1, beta=1):

        self.distances = distances
        self.pheromone = np.ones(self.distances.shape)
/ len(distances)
```

```

self.all_inds = range(len(distances))

self.n_ants = n_ants

self.n_best = n_best

self.n_iterations = n_iterations

self.decay = decay

self.alpha = alpha

self.beta = beta

self.E_all = E_all

# Основная функция
def run(self):

    shortest_path = None

    all_time_shortest_path = ("placeholder",
np.inf)

    for i in range((self.n_iterations)-1):

        all_paths = self.gen_all_paths()

        self.spread_pher(all_paths, self.n_best,
shortest_path=shortest_path)

        shortest_path = min(all_paths, key=lambda
x: x[1])

        if shortest_path[1] <
all_time_shortest_path[1]:

```

```

        all_time_shortest_path = shortest_path
        self.pheromone = self.pheromone *
self.decay
        return all_time_shortest_path

# Обновление феромона
def spread_pher(self, all_paths, n_best,
shortest_path):
        sorted_paths = sorted(all_paths, key=lambda
x: x[1])
        for path, dist in sorted_paths[:n_best]:
            for move in path:
                self.pheromone[move] += 1.0 /
self.distances[move]

# Длина пути муравья
def gen_path_dist(self, path):
        total_dist = 0
        for ele in path:
            total_dist += self.distances[ele]
        return total_dist

```

```

# Все маршруты муравьёв
def gen_all_paths(self):
    all_paths = []
    for i in range(self.n_ants):
        path = self.gen_path(0)
        sh_path = path
        all_paths.append((path,
self.gen_path_dist(path)))
    return all_paths

# Маршрут одного муравья
def gen_path(self, start):
    path = []
    visited = set()
    prev = start
    E = set()
    visited_prev = prev
    while E != self.E_all:
        move = self.pick_move(self.pheromone[prev],
self.distances[prev], visited_prev)
        ed = str(prev) + str(move)
        if ed not in E:

```

```

        E.add(ed)

        path.append((prev, move))

        visited_prev = prev

        prev = move

        visited.add(move)

    path.append((prev, start))

# возвращение в начальную точку
    return path

# Выбор следующего города
def pick_move(self, pheromone, dist, v):
    pheromone = np.copy(pheromone)
    pheromone[v] = pheromone[v]/12
    row = pheromone ** self.alpha * ((1.0 / dist)
** self.beta)

    norm_row = row / row.sum()

    move = np_choice(self.all_inds, 1, p=norm_row)

    return move

distances = np.array([[np.inf, 750, np.inf, np.inf,

```

np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, 1800,
np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf],
[750, np.inf, 410, 370, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, 410, np.inf, np.inf, 390, np.inf,
np.inf, 430, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, 370, np.inf, np.inf, 390, 430,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, 390, 390, np.inf, np.inf,
440, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, 430, np.inf, np.inf,
370, np.inf, np.inf, np.inf, np.inf, np.inf,

np.inf, np.inf, 1000, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf,
np.inf, np.inf, np.inf, 440, 370, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, 430, np.inf, np.inf, np.inf,
np.inf, np.inf, 2000, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, 2000, np.inf, 320, np.inf, np.inf,
340, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, 320, np.inf, 360, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, 360, np.inf, 530,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,

np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, 530, np.inf,
160, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, 170, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, 340, np.inf, np.inf, 160,
np.inf, 480, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
480, np.inf, 150, np.inf, 390, np.inf,
400, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, 1000,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, 150, np.inf, 530, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[1800, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, 530, np.inf, 440, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,

np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, 390, np.inf, 440, np.inf, 530,
np.inf, np.inf, np.inf, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, 530, np.inf,
350, np.inf, np.inf, np.inf, 510],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, 400, np.inf, np.inf, np.inf, 350,
np.inf, 480, np.inf, 500, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, 170,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
480, np.inf, 510, np.inf, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, 510, np.inf, 420, np.inf],
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,

```
500, np.inf, 420, np.inf, 310],  
[np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,  
np.inf, np.inf, np.inf, np.inf, np.inf, np.inf,  
np.inf, np.inf, np.inf, np.inf, np.inf, 510,  
np.inf, np.inf, np.inf, 310, np.inf]])
```

```
n_ants = 230
```

```
n_iterations = 8
```

```
decay = 0.9
```

```
alpha = 1
```

```
beta = 1
```

```
n_best = 1
```

```
E_all = set()
```

```
for i in range(len(distances[:, 1])):
```

```
    for j in range(len(distances[1, :])):
```

```
        if distances[i][j] != np.inf:
```

```
            e = str(i) + str(j)
```

```
            e_inv = str(j)+str(i)
```

```
            if e not in E_all:
```

```
                E_all.add(e)
```

```
                E_all.add(e_inv)
```

```

Ant = AntColony(distances, n_ants, n_best, n_iterations, de
short_path = Ant.run()

print(short_path) # вывод маршрута
print(short_path[1]) # длина маршрута
sh_path = short_path[0]

t_o = 360 # минуты

T_all = [["" for j in range(len(distances[:, 1]))] for i in
t = t_o

velocity = 300 # м/мин (18км/ч)

# Построение временных окон
for p in range(len(sh_path)):
    edge = sh_path[p]
    i = edge[0]
    j = edge[1]
    a = math.floor(t/60)*3600
    n = time.strftime("%H:%M", time.gmtime(a))
    t_e = distances[i][j]/velocity
    t += t_e
    b = math.ceil(t/60)*3600
    m = time.strftime("%H:%M", time.gmtime(b))
    if T_all[i][j] == "":
        T_all[i][j] = (n, m)

```

```
print(T_all) # вывод временных окон  
print(t) # время
```