ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

Кафедра математического и программного обеспечения информационных систем

РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ ПОЧТАЛЬОНА ДЛЯ ОРИЕНТИРОВАННЫХ ГРАФОВ

Выпускная квалификационная работа бакалавра

заочной формы обучения
направление подготовки: 02.03.02 «Фундаментальная информатика и информационные технологии»
профиль подготовки: «Супервычисления»
4 курса группы 07001360
Шамрицкий Борис Евгеньевич

Научный руководитель
Доцент Васильев П.В.

БЕЛГОРОД 2017

Содержание

Введение

Современный период развития цивилизованного общества по праву называют этапом информатизации.

Характерной чертой этого периода является тот факт, что доминирующим видом деятельности в сфере общественного производства становится сбор, обработка, хранение, передача и использование информации, осуществляемые на базе современных информационных технологий.

Всё чаще даёт о себе знать проблема низкой производительности каких-либо расчётов. Вот и транспортная задача не стала исключением. С возрастанием количества точек для развоза грузов переборные алгоритмы хотя и продолжают выдавать оптимальные результаты расчёта, но делают это слишком медленно.

Поэтому, перед нами встаёт задача улучшить, насколько это возможно, расчёты маршрутов автотранспорта. В рамках задачи исследованы точные, эвристические и гибридные алгоритмы для решения задачи почтальона.

Объект исследования данной работы - методы решения задач транспортного типа.

Предметом исследования являются конкретные алгоритмы и их комбинации, позволяющие получать решение задачи почтальона с необходимой точностью за удовлетворяющее нас время.

Целью работы стала реализация и исследование алгоритмов решения задачи почтальона для ориентированных графов.

В первом разделе описана задача почтальона, и вкратце методы ее решения, как точные, так и эвристические.

Во втором разделе подробно рассмотрены методы решения:

алгоритм полного перебора, метод ветвей и границ, эвристические алгоритмы, алгоритм ближайшего соседа, поведенческие (роевые)

алгоритмы, метод муравьиной колонии, алгоритм искусственной иммунной системы, алгоритм интеллектуальных капель, алгоритм имитации отжига, генетический алгоритм.

В третьем разделе проводится сравнительный анализ приведенных алгоритмов.

Четвертый раздел описывает гибридные системы решения задачи почтальона, далее приводиться схема интегрированного поиска.

В пятом разделе представлена схема интегрированного поиска.

И в заключительной шестом разделе реализуется муравьиный алгоритм который был выбран для выполнения работы.

Данная выпускная квалификационная работа включает в себя теоретическую часть, практическую часть, шесть таблиц, пятнадцать рисунков и десять формул.

1. Предметная область

Задача о почтальоне является одной из классических задач комбинаторики: задано n городов, известна стоимость проезда между всеми городами, требуется найти маршрут минимальной стоимости, проходящий через каждый город.

Существует несколько частных случаев задачи почтальона:

- геометрическая задача (планарная или евклидова), когда в расчетной матрице отражается информация о расстоянии между точками на плоскости;
- треугольная задача, когда для данных расчетной матрицы выполняется неравенство треугольника;
- симметричная и асимметричная задачи;
- задача с несколькими почтальонами – определение нескольких маршрутов в графе, суммарная стоимость которых будет минимальной;
- полностью определенная.

Чтобы гарантировать существование решения и упростить задачу, обычно, считают входной граф полностью связным. В случае если между некоторыми городами не существует сообщения, полноты графа можно достичь путем ввода рёбер с максимальной длиной. Если оптимальный маршрут для данного графа существует, введенные ребра никогда не попадут в решение.

Итак, в рамках данной работы я исследую задачу с помощью муравьиного алгоритма.

Методы решения

Все методы решения задач можно разделить на две группы: точные и эвристические.

К точным методам относятся алгоритм полного перебора и метод ветвей и границ. На поиск решения требуются большие временные затраты, но результаты, полученные данными методами, имеют 100% точность.

Отдельно, среди методов выделяют приближенные методы, использующие случайный поиск. Такие алгоритмы называют эвристическими. Данный класс алгоритмов позволяет получить решения, «близкие» к оптимальным. Зачастую этого бывает достаточно, ведь для большинства практических задач необходимо получить приближенное решение.

Эвристические методы будем разбивать на подгруппы: жадные методы, эволюционные и поведенческие (роевые) алгоритмы. Роевые алгоритмы, в свою очередь, можно разделить на методы, инспирированные живой и неживой природой.

Данная классификация позволяет:

- оценить риски получения неудовлетворительного результата у каждой группы алгоритмов;
- помогает выработать требования для гибридных методов;
- скомбинировать методы в оптимальную по времени и/или производительности систему.

2. Алгоритмы решения задачи почтальона

2.1 Точные методы

На поиск решения тратится большое количество времени, но результаты, полученные данными методами, имеют 100% точность, как было сказано выше, поэтому на этапах тестирования и сравнения для задач с небольшим количеством городов результат работы точных алгоритмов будем считать эталонным. В процессе разработки и исследования маршруты, вычисленные остальными алгоритмами, будут сравниваться с эталонным решением.

Алгоритм полного перебора

Данный алгоритм принадлежит к классу методов поиска решения исчерпыванием всевозможных вариантов.

Суть алгоритма в полном последовательном переборе всех вариантов решения.

Сложность метода зависит от количества всех возможных решений задачи. Например, при переборе всех путей для графа из n вершин потребуется анализ (n-1)! вариантов. Если n очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

Метод ветвей и границ

Метод ветвей и границ является широко распространённым, точным, не переборным алгоритмом решения задачи почтальона.

Суть метода заключается в направленном частичном переборе допустимых решений с отсевом подмножеств, заведомо не содержащих оптимальных решений.

Условные обозначения:

- C – начальная матрица;
- (k, l) – ребро ветвления;
- h – стоимость очередного тура; L – лист с наименьшей стоимостью h;
- T – тур.

Алгоритм метода ветвей и границ представлен на рисунке 2.1.1, в виде блок-схемы.

Рис. 2.1.1 Блок-схема метода ветвей и границ

Рассмотрим подробнее каждый этап алгоритма.

Приведение матрицы C, вычисление h.

В каждой строке матрицы C среди элементов, не равных -1 (-1 обозначает вычеркнутые элементы матрицы), находится минимум. Найденные минимумы вычитаются из всех элементов строки, соответственно, и суммируются в переменную h.

Производим аналогичные действия второй раз, но минимумы находим и вычитаем уже из столбцов матрицы C. Добавляем их к h.

Строим корневой узел дерева решений, который включает в себя приведенную матрицу стоимостей C и начальную стоимость пути h.

Выбор ребра ветвления (k, l).

Для каждого элемента C[i, j], равного 0, находим сумму минимальных значений в i–й строке и j–м столбце, исключая сам элемент C[i, j]. В качестве ребра ветвления (k, l) выбираем ребро, дающее максимальное значение вычисленной суммы минимумов.

Построение новых ветвей

Вычисление узла Y и построение ветви LY.

Переносим в лист Y значения из узла L – матрицу стоимостей C, стоимость маршрута h и найденный тур T.

Изменяем стоимость h узла Y, добавив к значению h сумму минимумов, вычисленную для ребра ветвления (k, l). Элемент C[k, l] вычеркивается из матрицы стоимостей C.

Добавляем в тур T ребро HE (k, l) – это значит, что данное ребро не может быть использовано в конечном маршруте и отбрасывается как неоптимальное.

Вычисление узла X и построение ветви LX.

Из матрицы C вычеркиваем k–ю строку и l–й столбец и элемент C[l, k].

Приводим полученную матрицу к стандартному виду путем нахождения и вычитания минимумов из строк и столбцов. Все найденные минимумы прибавляются к значению h. Добавляем в тур T ребро (k, l).

Нахождение листа с наименьшей стоимостью.

Поиск листа в дереве решений с минимальным значением h.

Присваиваем h(min) = h.

В литературе даны две оценки временной сложности алгоритма - $O(n*\log_2 n)$ для случая без возвратов к оборванным ветвям и $O(n^5*\log_2 n)$ в случае возврата.

2.2 Эвристические алгоритмы

К ним относятся алгоритмы решения, не имеющие строгого обоснования, но, тем не менее, дающие приемлемое решение задачи в большинстве практически значимых случаях.

Алгоритм ближайшего соседа.

Данный алгоритм следует отнести к простейшим эвристическим методам решения транспортной задачи.

На каждом шаге выбирается ребро наименьшей стоимости из множества непомеченных рёбер, не нарушающих корректности решения.

Алгоритм метода представлен на рисунке 2.2.1, в виде блок-схемы.

Временная сложность метода составляет O(n-1), так как он рассматривает лишь один из возможных вариантов. Данный метод можно использовать для инициализации начального решения.

Рис. 2.2.1 Блок-схема алгоритма ближайшего соседа

Поведенческие (роевые) алгоритмы

Роевой интеллект описывает коллективное поведение самоорганизующейся децентрализованной системы. Концепция роевого интеллекта рассматривается в теории искусственного интеллекта, как метод оптимизации.

Впервые, данный термин был введен Херардо Бени и Ван Цзином в 1989 году. Система роевого интеллекта состоит из множества агентов, локально взаимодействующих с окружающей средой и подобными особями. Сами агенты обычно довольно просты, но все вместе, локально взаимодействуя, создают, так называемый, роевой интеллект.

Разделим роевые алгоритмы на две категории: методы, инспирированные живой и неживой природой.

К первой подгруппе можно отнести алгоритм муравьиной колонии и метод имитации искусственной иммунной системы. Отличительной особенностью подгруппы является то, что система способна к обучению, обладает памятью и решает задачи поиска и классификации.

Метод муравьиной колонии

Одним из эвристических методов искусственного интеллекта является алгоритм муравьиной колонии**.** Данный алгоритм был предложен Марко Дориго и имитирует движение колонии муравьев в природе.

Обладая коллективным разумом, муравьи способны легко адаптироваться к изменяющимся условиям, как показано на рисунке 2.2.2 и быстро находить оптимальный маршрут до заданной точки (например, к источнику пищи).

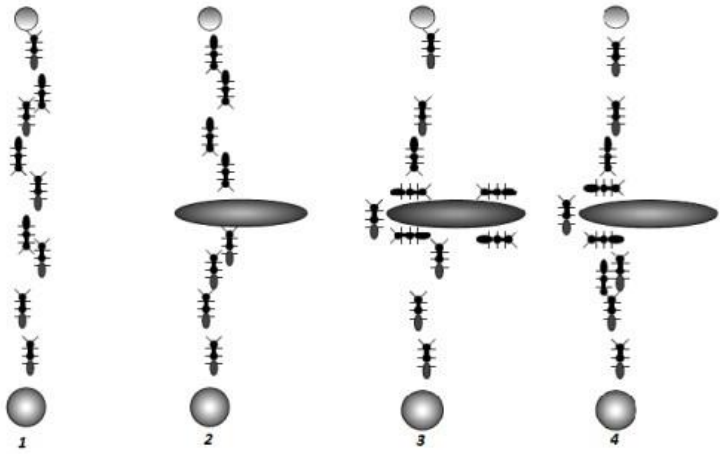

Рис. 2.2.2. Муравьи прокладывают маршрут. Адаптация к изменениям

Идея алгоритма муравьиной колонии – моделирование сценария поведения поиска маршрутов муравьями в природе.

Любой муравьиный алгоритм, независимо от модификаций и дополнений, представим в виде рисунка 2.2.3.

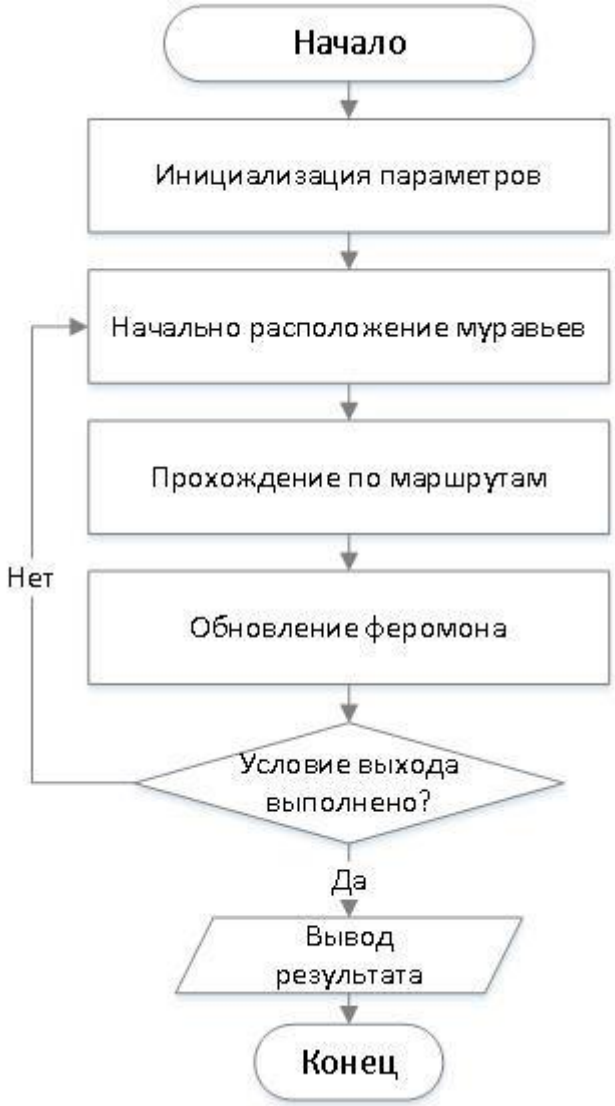

Рис. 2.2.3 Блок-схема метода муравьиной колонии

Подробнее рассмотрим каждый шаг алгоритма.

Инициализация начальных параметров.

Перед началом работы алгоритма необходимо определить количество особей (агентов) в роевой системе, интенсивность, с которой будут

испаряться феромоны, а так же степень зависимости решения от значения феромона и таблицы стоимостей.

Все параметры определяются опытным путем.

Расположение муравьев.

Существует множество различных способов начальной инициализации колонии. Способ расположения напрямую зависит от численности роя и количества городов в системе.

Я представлю несколько способов расположения колонии:

• «Покрытие» – стандартное размещение муравьев. Количество агентов в системе совпадает с количеством городов, и каждый агент располагается в собственной начальной вершине графа.

• «Рандомное» распределение – муравьи в вершинах графа располагаются случайным образом. Численность колонии и количество вершин могут не совпадать.

• «Центрирование» или «фокус» – вся колония находится в одной вершине, на каждой итерации поиск маршрута начинается из данной вершины. Численность колонии может быть любой.

• «Мигрирующая колония» – после каждой итерации вся колония муравьев перемещается в любую, случайно выбранную вершину графа. Численность колонии может быть любой.

Прохождение по маршрутам

Выбор города основывается на следующих матрицах - матрице расстояний D и таблице феромонов T. Феромоны – это некоторое вещество, которое «откладывают» муравьи, помечая пройденный маршрут между городами. В рамках поставленной задачи почтальона, феромоны – это некоторые веса, которыми помечаются ребра графа.

Независимо от способа покрытия, каждый муравей строит свой маршрут, начинающийся из его личной начальной вершины, основывая свой выбор на следующих свойствах:

Муравьи обладают «*памятью*». Так как каждая вершина должна попасть в маршрут только один раз, то у каждого муравья список разрешенных городов после каждой итерации уменьшается. Пройденные города стираются из памяти.

Обозначим через $J_{i,k}$ список вершин, которые муравью *k,* находящемуся в городе *i,* еще необходимо посетить.

Муравьи обладают *«зрением».* Видимость - это некоторый коэффициент, определяющий силу желания посетить город j, если муравей находится в городе i.

В рамках задачи будем считать, что видимость обратно пропорциональна расстоянию между городами $\eta_{i,j} = 1/ d_{i,j}$.

Муравьи обладают *«обонянием»*, т.е. они могут улавливать след феромона, в момент времени t на ребре (i, j) вес феромона составляет $\tau_{i,j}(t)$.

Направление движения муравья определяет случайное число, которое отправляет его из города i в город j с бóльшей вероятностью, если функция 2.2.1, представленная ниже, принимает бóльшее значение.

$$\begin{cases} P_{ij,k}(t) = \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} * \left[\eta_{ij}\right]^{\beta}}{\sum_{l \in J_{i,k}} \left[\tau_{il}(t)\right]^{\alpha} * \left[\eta_{il}\right]^{\beta}}, j \in J_{i,k} \\ P_{ij,k}(t) = 0, j \notin J_{i,k}; \end{cases} \quad (2.2.1)$$

где $\tau_{i,j}$ – уровень феромона, $\eta_{i,j}$ – расстояние, α, β – параметры зависимости принятия решения от количества феромона и значений матрицы расстояний.

При α = 0 алгоритм вырождается и становится «жадным», т.е. выбор будет сделан в пользу вершины, расположенной на самом минимальном расстоянии к текущей. При β = 0 выбор основывается только на опыте других муравьев – будет выбрано направление, имеющее наибольшее значение феромона, что может привести к раннему зацикливанию на локальном минимуме. Поэтому, экспериментальным путем необходимо найти компромисс между параметрическими величинами.

Пример работы метода муравьиной колонии с различными вариациями параметров для задачи с 50 городами представлен ниже в таблице 2.2.1.

Таблица 2.2.1 Время работы алгоритма муравьиной колонии

Параметры	α = 1 и β = 1		α = 2 и β = 1		α = 5 и β = 1		α = 0 и β = 1		α = 1 и β = 2		α = 1 и β = 5		α = 1 и β = 0	
Количество итераций	10	100	10	100	10	100	10	100	10	100	10	100	10	100
Лучшее решение	6501	4126	3657	3506	4321	4249	10097	9857	4015	3569	**3509**	**3471**	15551	12750
Худшее решение	7384	4996	4675	3814	4909	4337	11188	10481	4464	3762	**3582**	**3493**	15578	13402
Средний результат	6943	4561	4166	3660	4615	4293	10643	10169	4240	3666	**3546**	**3482**	15565	13076

Заметим, что выбор города является вероятностным, правило (1) лишь определяет значения соответствующих вероятностей переходов из города i в другие города. Когда все вероятности посчитаны, случайным образом выбирается зона, соответствующая некоторому городу, который и попадает в маршрут.

Правило (1) не изменяется в ходе алгоритма, но у двух разных муравьёв значение вероятности перехода будут отличаться, т.к. они имеют разный список разрешённых городов.

*Обновление феромона***.**

Пусть $T_k(t)$ – маршрут, который проходит муравей k к моменту времени t, $L_k(t)$ – длина пройденного маршрута, а Q – параметр, имеющий значение

порядка длины оптимального пути. Тогда количество феромона, откладываемого на данном ребре, может быть задано в виде функции 2.2.2.

$$\Delta\tau_{ij,k}=\begin{cases}\dfrac{Q}{L_k},(i,j)\in T_k(t)\\ 0,(i,j)\notin T_k(t)\end{cases} \qquad (2.2.2)$$

где Q – параметр, имеющий значение порядка цены оптимального решения, то есть Q/L_k – феромон, откладываемый k-ым муравьём, использующим ребро (i, j).

Веса феромонов, которыми помечены ребра графа, изменяются – увеличиваются и уменьшаются. При частом прохождении муравьев по какому-либо ребру значение весов увеличивается. Если же ребро невостребованное, значение веса со временем уменьшается – феромон «испаряется». Скорость процесса уменьшения веса зависит от параметра p и требует отдельного рассмотрения. Как представлено в формуле 2.2.3.

$$\tau_{ij}(t+1)=(1-p)\tau_{ij}(t)+\Delta\tau_{ij} \qquad (2.2.3)$$

Где $p \in [0,1]$ – интенсивность испарения, $L_k(t)$ – цена текущего решения для k-ого муравья.

Если скорость будет слишком велика, решение быстро вырождается.

Критерий останови

Критерием остановки может быть время жизни колонии – количество совершаемых итераций или достижение установленного значения минимума для искомого решения.

Временная сложность этого алгоритма зависит от времени жизни колонии t (число итераций), количества вершин графа **n** и числа муравьев **m**, и определяется как $O(t*n^2*m)$.

Алгоритм искусственной иммунной системы

В природе иммунная система защищает организм животного от инородных вирусов и инфекций. Работа иммунной системы выстроена следующим образом. Лимфоциты передвигаются по кровеносным сосудам в поисках антигенов. Найденные антигены уничтожаются, система развивается, учится находить новые антигены и продолжает поиск.

Для решения задачи почтальона с помощью иммунной системы сопоставим биологические объекты и процессы с их математическими аналогами.

Вершины графа – антигены и помеченные антигены.

Ребра графа – всевозможные ходы кровеносной системы.

Агенты роевой системы – лимфоциты.

Процесс выбора агентом следующей вершины основывается на матрице расстояний и на предыдущих неудачных попытках.

Вероятность выбора вершины j из i обратно пропорциональна расстоянию до этой вершины. Агент, запущенный для прохождения пути повторно, содержит свой предыдущий маршрут, что позволяет изменять решение.

Алгоритм метода представлен ниже на рисунке 2.2.4, в виде блок-схемы.

Рис. 2.2.3 Блок-схема алгоритма имитации иммунной системы

Если стоимость нового маршрута превышает известный агенту минимум – агент прекращает движение по данному маршруту и начинает сначала.

Процесс построения маршрута, с учетом прошлых ошибок, выстроен следующим образом:

Изменение маршрута на данном ребре будет совершено с вероятностью 50%. Если принято решение изменить маршрут, то вершина, принадлежащая предыдущему маршруту, в данный момент не принимает участия в процессе выбора следующей вершины.

Критерием остановки может служить как количество итераций, так и сходимость решения к одному значению.

Алгоритм интеллектуальных капель

Алгоритм относится к подгруппе методов, инспирированных неживой природой.

Данный метод имитирует совместное передвижение капель воды (рек) в природе, которые можно ассоциировать с гигантскими массивами агентов роевой системы.

Каждая капля в отдельности и весь рой в целом движутся по пути наименьшего сопротивления, и для подобного роя русло, по которому передвигается река, является самым оптимальным по стоимости маршрутом. Агенты данной системы, перенося частички земли, способны перестраивать русло и адаптировать маршрут к изменениям.

Таким образом, концепцию передвижения капель воды в природе можно применить для решения задачи почтальона.

Ассоциируем природные явления их математическим аналогам.

Ребра графа – все возможные маршруты передвижения.

Вершины графа – точки смены направления.

Таблица стоимости – таблица, на основании которой ребрам графа присваиваются некоторые веса.

Вес на ребре – количество почвы на данном участке. Данное значение напрямую зависит от стоимости ребра.

Агенты роевой системы – капли, проходящие по ребрам.

Каждый агент обладает скоростью и способен переносить с собой некоторое количество почвы.

Алгоритм поиска оптимального маршрута представлен на рисунке 2.2.4, виде блок-схемы.

.

Рис. 2.2.4 Блок-схема алгоритма интеллектуальных капель

Рассмотрим каждый шаг подробнее:

Инициализация начальных данных.

На данном этапе определяются максимальное количество итераций или иное условие останова, задается количество городов и значения матрицы стоимости, определяется число агентов роевой системы и их расположение.

Определяется количество почвы, изначально находящейся на ребрах графа.

Задаются начальные значения параметров скорости vel_0 и начальный уровень почвы $soil_0$. Количество переносимой почвы задаётся с помощью уравнения с параметрами a_v, b_v,c_v, а так же задаются коэффициенты для уравнения обновления почвы a_s, b_s, c_s. Параметры p_0 и p_n определяют соотношение между изменением количества запасённой земли на данном участке и количество переносимой земли каждым агентом роя. Сумма данных параметров должна быть равна 1.

Для получения решения использовались различные наборы параметров. В реализации метода была использована следующая комбинация параметров, представленная ниже в таблице 2.2.2.

Таблица 2.2.2 Входные параметры для алгоритма капель

Параметр	$soil_0$	vel_0	a_v	b_v	c_v	a_s	b_s	c_s	p_0	p_n
Значение	1000	200	1	0,1	1	1	0,1	1	0.9	0,1

В список посещенных агентом городов заносится первый случайно выбранный город.

Выбор следующей вершины.

Рассматриваются всевозможные направления движения. В маршрут включается то ребро, с которым минимальна разница относительно

предыдущего ребра по количеству почвы. Обновляется список городов, посещенных агентом.

Изменение скорости передвижения агента

При перемещении скорость со временем изменяется по формуле …:

$$vel_{t+1} = vel_t + a_v / (b_v + c_v * soil^2(i,j)) \quad (2.2.4)$$

где vel_t – скорость на предыдущем участке, a_v, b_v,c_v – параметры, определяющие количество переносимой почвы, soil(i,j) – количество почвы на участке.

Обновление количества почвы на ребрах

Каждый агент способен переносить с собой некоторое количество почвы с одного ребра на другое. Имеется зависимость этого количества от скорости элемента и времени его движения (2):

$$\Delta soil_{i,j} = a_s / (b_s + c_s * time(i,j,vel_{t+1}) \quad (2.2.5)$$

где $\Delta soil_{i,j}$ - переносимая почва, a_s, b_s, c_s – параметры, определяющие уравнение обновления почвы, $time(i,j,vel_{t+1})$ - эвклидово расстояние между точками, деленное на скорость движения капли (1).

Новое количество почвы на ребре (i, j) вычисляется по формуле (3):

$$soil_{i+1,j+1} = p_0 * soil_{i,j} + p_n * \Delta soil_{i,j} \quad (2.2.6)$$

где p_0, p_n – коэффициенты, определяющие процесс изменения количества почвы.

Вычисляется лучшее время, которое потребовалось для преодоления данного участка пути. Обозначим его T.

После того, как в маршрут будут включены все вершины графа, вычисляется лучшее время, необходимое для обхода графа.

Алгоритм имитации отжига

За основу взят процесс кристаллизации вещества.

Каждый металл обладает своей кристаллической решеткой, которая описывает геометрическое положение атомов вещества. Совокупность позиций атомов в кристаллической решетке будем называть состоянием системы. Каждому состоянию соответствует определенный уровень энергии.

Цель отжига – привести систему в состояние с наименьшей энергией. Чем ниже уровень энергии, тем «лучше» кристаллическая решетка, и тем прочнее металл.

Преобразуем данную концепцию для решения задачи почтальона:

- Ребра графа – узлы кристаллической решетки.
- Маршрут по всем ребрам – состояние системы.
- Стоимость маршрута – уровень энергии в системе.
- Количество итераций – $(T_0 – T_{min})/\Delta T$, Где T_0 – начальная температура, T_{min} – нижний порог, ΔT – уменьшение температуры на каждом шаге.

Инициализируем начальные параметры.

Задаем температуру, шаг изменения температуры и начальное, произвольное состояние системы – маршрут для задачи почтальона.

Построение кандидата.

Данный процесс заключается в случайном изменении состояния системы – изменении последовательности городов в маршруте. Могут поменять свое положение 2 и более городов, выбор города осуществляется случайно.

Оценка решения.

Задается значение, определяющее, насколько стоимость маршрута кандидата может отклониться от имеющегося лучшего маршрута в худшую сторону.

Кандидат, удовлетворяющий данным условиям, становится текущим решением. Лучшее решение сохранится.

Алгоритм метода представлен на рисунке 2.2.5, в виде блок-схемы.

Рис. 2.2.5 Блок-схема алгоритма имитации отжига

Временная сложность алгоритма - $O(n^2 * \log_2(n))$, где n – количество городов.

2.3 Эволюционное моделирование

В области эволюционного моделирования выделяют раздел «эволюционные алгоритмы». Такие алгоритмы используются при комбинаторной оптимизации, в частности, при решении классических NP-полных задач.

В общем виде, идею эволюции понимают как преобразование комбинации блоков. Установлено, что гены чаще всего объединяются в кластеры. Благодаря этому, целые семейства могут быть сразу включены или выключены из эволюции.

Генетический алгоритм

Алгоритм основан на генетических процессах биологических организмов: биологические популяции развиваются в течение нескольких поколений, подчиняясь законам естественного отбора и по принципу "выживает наиболее приспособленный".

Для нашей задачи мы будем ассоциировать цепочку городов с цепочкой генов, т.е. хромосом.

На первом этапе параметрически задаем мощность популяции и инициализируем начальное поколение, определяем вероятность возникновения мутации и условия останова.

«Эволюционный процесс» продолжается несколько жизненных циклов (поколений). Критерием остановки может быть:

нахождение глобального решения;

исчерпание числа поколений, отпущенных на эволюцию;

исчерпание отпущенного на эволюцию времени.

Решение генетического алгоритма представлено на рисунке 2.2.6, в виде блок-схемы.

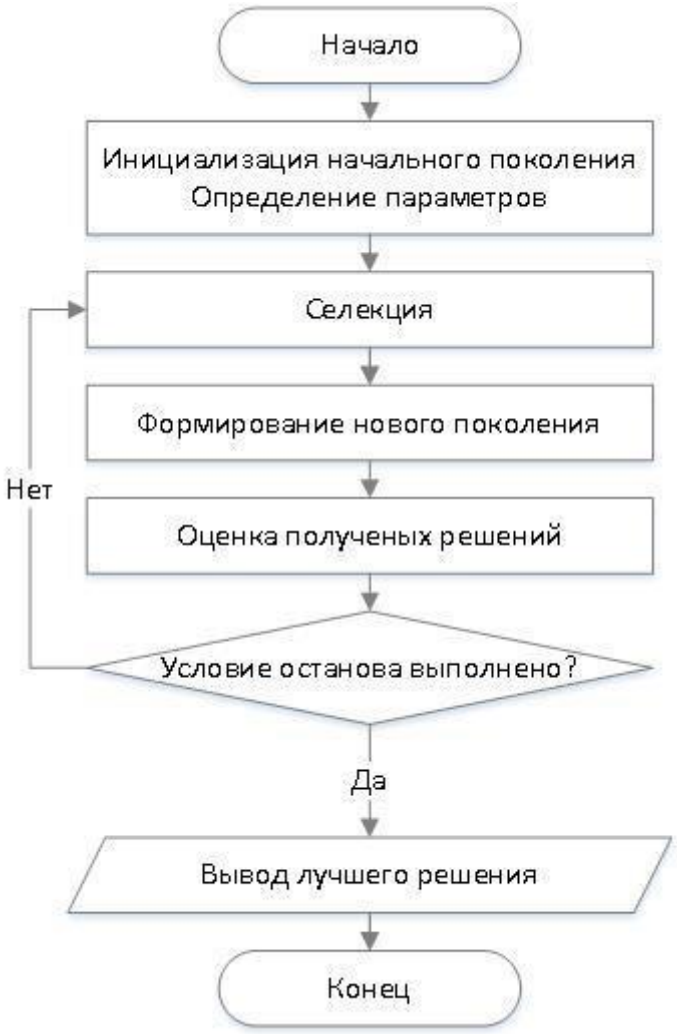

Рис. 2.2.6 Блок-схема генетического алгоритма

Селекция - это выбор определенной доли популяции, которая останется «в живых» на данном этапе эволюции. Селекция необходима, так как множество потомков и мутантов имеют сниженную жизнеспособность, и их необходимо отсеять в процессе естественного отбора. В результате селекции необходимо выделить особей, которые будут участвовать в дальнейшем скрещивании и формировании нового поколения.

Пример матрицы представлен в виде таблицы 2.3.1.

Таблица 2.3.1

–	**1**	**2**	**3**
4	**–**	**5**	**6**
2	**4**	**–**	**6**
8	**1**	**3**	**–**

Маршруты и их стоимости

- 1 – 2 – 3 – 4 (1 + 5 + 6 + 8 =20)
- 1 – 3 – 4 – 2 (2 + 6 + 1 + 4 = 13)
- 1 – 4 – 2 – 3 (3 + 1 + 5 + 2 = 10)

Последняя особь в данном примере является самой сильной.

Теперь высчитываем вероятность, с которой каждая особь может стать родителем.

P(1) = 1/20 / (1/20+1/13+1/10) = 0,05 / (0,05+0,08+,0,1) = 0,22 (22%)

P(2) = 1/13 / 0,23 = 0,35 (35%)

P(3) = 1/10 / 0,23 = 0,43 (43%)

Делим числовую прямую пропорционально процентным долям и выбираем особей, которые будут дальше участвовать в процессе поиска решения.

Формирование нового поколения происходит с помощью операторов Кроссинговера и мутации.

Кроссинговер (так же кроссовер или скрещивание) — это операция, порождающая из двух хромосом одну или несколько новых.

Выделяют одноточечный и многоточечный кроссинговер.

Кроссинговер в генетическом алгоритме реализуется не так, как в природе, так как маршрут не должен проходить через 1 город дважды, и это стоит учитывать. Хромосомы разрезаются в случайной точке и обмениваются частями без повторений, с дальнейшим сдвигом и добавлением недостающих генов.

Пример: (1, 2, 3, 4, 5) и (1, 4, 3, 5, 2) разрезаем между третьим и четвертым генами. Берем первую часть первой хромосомы (1, 2, 3, ?, ?), по порядку просматриваем гены из второй части второй хромосомы и, если такого гена в дочерней хромосоме не найдено, вставляем на соответствующее место (1, 2, 3, 5, ?). Если в дочерней хромосоме заполнены не все пустые места, проходим по первой части второй родительской хромосомы, и заполняем пустые места по порядку недостающими генами, получаем дочернюю хромосому (1, 2, 3, 5, 4). Аналогично получаем вторую хромосому (1, 4, 3, 5, 2), которая совпадает с одной из родительских.

Мутация — это преобразование хромосомы, случайно изменяющее одну или несколько позиций генов. В нашем случае, два случайно выбранных гена будут меняться местами.

Пример: (1, 2, 3, 4, 5) мутирует в (1, 2, 3, 5, 4).

Оператор мутации напоминает процесс построения нового маршрута в методе имитации отжига.

Теоретическая временная сложность алгоритма составляет $T*M*2*N^2$, где T - число поколений, M - размер популяции, N^2 - временная сложность декодирования хромосомы, N - число цепей.

3. Сравнительный анализ алгоритмов

Стоит заметить, что роевые алгоритмы имеют много общего с генетическими.

Сила генетических алгоритмов (ГА), прежде всего, в параллельной природе поиска решений. В данных алгоритмах реализован псевдослучайный поиск решения, который сохраняет кратные решения и дает приемлемые результаты. Генетические операторы позволяют даже слабым, в принципе, не перспективным решениям оставаться в числе потенциальных решений и генерировать из них приемлемые.

Использование генетических операторов обеспечивает успех поиска хорошего решения. Все генетические алгоритмы используют некоторую форму рекомбинации, позволяющую порождать новые решения, которые сохраняют хорошие качества родителей и с большой вероятностью показывают хорошие характеристики. Кроссинговер является основным оператором генетического алгоритма, в то время как мутация используется намного реже.

Роевые алгоритмы (РА) не имеет генетических операторов, подобных кроссинговеру и мутации. Здесь потенциальные решения-частицы взаимодействуют и изменяют свои скорости. Частицы имеют память, что очень важно для алгоритма.

Механизмы передачи информации в РА и ГА совершенно различны:

- в ГА хромосомы обмениваются информацией друг с другом. Поэтому вся популяция движется как единая группа в область оптимума;
- в РА только глобальная (локальная) лучшая позиция передается другим частицам. Это единственный механизм передачи информации.

Роевые алгоритмы способны выдавать удовлетворительное решение для задач с одним явным глобальным экстремумом уже через несколько

итераций. Если же функция обладает несколькими локальными экстремумами, то необходимо достаточное количество времени для определения глобального.

Сравним все методы, представленные в данной работе, по следующим критериям.

Точные алгоритмы – в 100% случаев выдают точное решение.

Эвристические – в методе принятия решения присутствует некоторая случайность.

- Перечислим, что может учитываться при построении нового маршрута: опыт предыдущих попыток;
- матрица стоимости (используется в процессе построения); матрица дополнительных коэффициентов;
- заведомо плохое решение;
- процесс построения маршрута включает элемент случайности;
- представленный метод требует инициализации начального решения;
- матрица стоимости используется только для оценки полученного решения;
- алгоритм подразумевает возможность распараллеливания.

Ниже представлены сравнительный анализ исследуемых методов, в виде рисунка 3.1, по перечисленным выше признакам таблицы 2.3.1 и результаты оценки временной сложности работы алгоритмов в таблице 3.2

	Точный	Эвристический	Предыдущий опыт	Матрица стоимостей	Дополнительные коэффициенты	Случайный выбор	Включает плохое решение	Требует начальное решение	Выход из локальных экстремумов	Матрица стоимости используется только для оценки решения	Возможность распараллеливания	Количество параметров
Полный перебор	v		v	v			v		v	v		0
Метод ветвей и границ	v		v	v	v	v	v		v	v		0
Ближайшего соседа		v		v		v				\		0
Рекурсивный перебор		v		v		v				\		1
Генетические алгоритмы		v	v	v		v	v	v	v	v	\	1
Метод имитации отжига		v		\		v		v	v	v	v	4
Искусственная иммунная система		v	v	\			v			v	v	3
Муравьиной колонии (МК)		v	v	v	v	v	v			v	v	4
Алгоритм интеллектуальных капель		v		\	v	v	v			v	v	8

Рис. 3.1 сравнительный анализ исследуемых методов.

V – алгоритм полностью соответствует данному признаку, / - частичное соответствие алгоритма признаку.

Ниже представлена таблица 3.1 с оценкой сложности алгоритмов.

Таблица 3.1 Оценка сложности алгоритмов

Алгоритм	Временная сложность	
Полный перебор	$O(n!)$	
Метод ветвей и границ	Луч.	$O(n^3 * \log_2 n)$
	Худ.	$O(n^5 * \log_2 n)$
Ближайшего соседа	$O(n)$	
Рекурсивный перебор	$2^{O(n)}$	
Муравьиной колонии (МК)	$O(t * m * n^2)$	
Искусственная иммунная система	$O(t * m * n^2)$	
Алгоритм интеллектуальных капель	$(m * n^2)$	
Метод имитации отжига	$O(n^2 * \log_2 n)$	
Генетические алгоритмы	$O(t * m * n^2)$	

где n – количество городов, m – численность популяции, t – количество совершенных итераций.

Изучив две представленные выше таблицы, можно сделать некоторые выводы.

Например, в худшем случае генетические и муравьиные алгоритмы тратят на получение решения одинаковое количество операций.

Алгоритмы, использующие дополнительную таблицу коэффициентов, хуже выходят из локальных минимумов.

Случайные алгоритмы, не учитывающие опыт предыдущих решений, отличаются скоростью генерации решений, но обладают некоторой неточностью. В некоторых случаях, быстрый алгоритм даже может дать самое худшее решение из существующих.

Метод рекурсивного перебора, использующий алгоритм объединения городов в кластеры, показывает хорошие результаты, если города изначально разбиты на тесные подгруппы, расположенные далеко друг от друга. Иначе, результат проваливается в глобальный минимум и не выходит из него.Так же, были проведены эксперименты и получены решения для матриц. Сравнивались длины найденных маршрутов за 10 (или менее, при быстрой сходимости) итераций алгоритма. Для матрицы 5 х 5 решение, полученное с помощью метода ветвей и границ, являлось эталонным. В результате найден гамильтонов цикл, включающий следующие ребра: (1,5), (5,3), (3,2), (2,4), (4,1). Длина маршрута равна 1570. Далее представлена таблица 3.2, в которой производится сравнение длин маршрутов, полученных различными методами.

Таблица 3.2 Сравнение длин маршрутов, полученных различными методами

Алгоритм	Найденная длина маршрута (для эвристик за 10 итераций)		
	5 городов	25 городов	50 городов
Полный перебор	-	-	-
Метод ветвей и границ	**1570**	-	-
Ближайшего соседа	2063	6510	14490
Рекурсивный перебор	2112	7320	12030
Муравьиной колонии (МК)	**1570**	5914	**9831**
Искусственная иммунная система	1629	6940	10312
Алгоритм интеллектуальных капель	1644	5914	**9831**
Метод имитации отжига	1644	**5794**	10893
Генетические алгоритмы	1805	5843	9911

4. Гибридные системы

На данный момент применение разобранных ранее эвристических алгоритмов (эволюционных, популяционных и случайного поиска) не может гарантировать успех при каждой попытке найти решение. Но в настоящее время существуют методы, способные объединять алгоритмы в более продуктивные системы.

Создание гибридной (интегрированной) системы алгоритмов, включающей в себя как разные, так и одинаковые методы, но с различными параметрами, позволит компенсировать недостатки одного алгоритма преимуществами другого. Следовательно, в настоящее время одним из путей повышения эффективности поиска решений является разработка гибридных алгоритмов.

Существует множество различных способов гибридизации алгоритмов. В рамках данной работы будем опираться на одноуровневую классификацию Ванга, которая включает три разновидности гибридных алгоритмов:

• вложенные алгоритмы – внедрение методов друг в друга;

• препроцессор/постпроцессор – последовательная комбинация алгоритмов;

• коалгоритмы – параллельное выполнение методов с постоянным обменом информацией.

Рассмотрим их подробнее:

- Во вложенной гибридизации выделим два уровня вложенности: высокоуровневое и низкоуровневое.

Высокоуровневая гибридизация – процесс внедрения одного алгоритма в другой на поверхностном уровне, когда связь между алгоритмами очень слаба, и один алгоритм без труда можно выделить из другого.

Низкоуровневая гибридизация подразумевает под собой сращивание алгоритмов в одно целое, при этом каждый отдельный алгоритм очень сложно выделить из гибрида.

Общий алгоритм действия для создания гибридов методом низкоуровневого вложения отсутствует, но основная идея - модификация операторов поиска с помощью инструментов нового алгоритма.

Интересные решения можно получить при взаимодействии эволюционного алгоритма с популяционными методами. Пример, генетический алгоритм и метод имитации отжига. Оператор селекции достраивается частью алгоритма имитации отжига. Это повышает шансы вхождения в новую популяцию нового индивида, который не позволит решению провалиться в глобальный экстремум.

- Гибридизацию способом препроцессор/постпроцессор так же разделим на две подкатегории: фиксированные последовательности алгоритмов и гибриды альтернативной адаптации.

При фиксированной последовательности порядок следования методов известен заранее. Это самый элементарный, но, тем не менее, действенный метод формирования новых алгоритмов.

Основная идея – на начальных этапах обеспечить широкий обзор всевозможных решений, а на последующих этапах сужать области поиска.

Метод быстро позволяет отыскать окрестности решения, но тратит неоправданно большое количество итераций на отыскание глобального экстремума.

Удалось сформулировать единый сценарий действий для создания гибридов с помощью гибридизации данного класса:

- инициализация поколения;
- поиск решений с помощью популяционного алгоритма;

- проверка и выявление лучших решений с помощью дополнительного алгоритма;
- проверка выполнения условия останова: да – пункт 5, нет – переход к пункту 2;
- вывод данных, к пункту 6.
- Выход.

Адаптирующиеся гибриды используют более двух алгоритмов, при этом очередность выполнения определяется по ходу решения конкретной задачи.

Например, популяционный алгоритм + Имитации отжига + алгоритм локального поиска.

- Ярким примером коалгоритмической гибридизации являются алгоритмы с несколькими популяциями.

Многопопуляционный генетический или многопопуляционный гибридный муравьиный алгоритмы, в которых две или более популяции развиваются параллельно. Изредка особи одной группы могут внедриться ко второй, вследствие чего происходит обмен информацией, и вторая группа, возможно, изменит ход эволюции.

Опираясь на классификацию по признакам, можно сформулировать комбинации алгоритмов, компенсирующие недостатки друг друга.

4.1Модификация муравьиного алгоритма

Данная модификация будет отнесена к классу высокоуровневой вложенной гибридизации. Модификация алгоритма муравьиной колонии заключается во введении «элитных» муравьёв. Их основное назначение – усиление лучших маршрутов за счет выделения бóльшего количества феромонов (4.1).

$$\Delta\tau_e = e * \frac{Q}{L^*} \quad (4.1)$$

Вложенность достигается следующим способом.

Муравьиный алгоритм отрабатывает одну итерацию. Находит комплект оптимальных решений на данный момент. Затем происходит внедрение элитных муравьев, которые делают свой выбор, основываясь на цене наилучшего маршрута.

Стоить заметить, что в данной системе количество элитных муравьёв является дополнительным параметром, требующим определения, так как для слишком большого числа элитных муравьев алгоритм может "застрять" на локальных экстремумах.

5. Схема интегрированного поиска

Основная идея интегрированного поиска - объединение алгоритмов в многоуровневую модель для повышения эффективности процесса получения решения.

Гибридная модель будет относиться к группе препроцессор/постпроцессор, являться адаптирующейся системой и включать 3 уровня, как показано на рисунке 5.1.

Рис. 5.1 Блок-схема интегрированного метода

На первом, предварительном уровне, будут генерироваться одно или несколько множеств для получения первоначальной популяции решений. В данном блоке возможно использование «жадных» алгоритмов.

Далее получаем семейство решений с помощью роевых алгоритмов. Главная цель этого уровня – обеспечение широкого обзора всевозможных решений. На начальных этапах нежелательно применение алгоритмов, использующих матрицу дополнительных коэффициентов, так как данная матрица усиливает локальные экстремумы, поэтому воспользуемся методом имитации отжига.

На заключительном этапе необходимо сузить области поиска путем отбрасывания заведомо неэффективных решений.

Воспользуемся генетическим алгоритмом и его оператором селекции.

После выполнения сужения области поиска проверяем выполнение условия останова и пересчитываем входные параметры, если требуется повторение алгоритма.

Предлагаются следующие варианты комбинаций алгоритмов:

• Метод ближайшего соседа + имитация отжига + оператор селекции из ГА. Временная сложность – $O(n^2 * \log_2 n)$.

• Одна итерация муравьиного алгоритмы с поиском по матрице стоимостей + муравьиный алгоритм с уравновешенным процессом поиска (по феромонам и стоимости) + оператор мутации из ГА. Временная сложность составляет $O(t * m * n^2)$.

• Вырожденный алгоритм интеллектуальных капель (без переноса почвы, для построения начальных маршрутов) + кроссинговер генетического алгоритма + алгоритм интеллектуальных капель с усиленным коэффициентом переноса почвы. Временная сложность – $O(t * m * n^2)$.

Далее на рисунке 5.2 представлены результаты работы стандартного муравьиного алгоритма (параметры $\alpha = 1$ и $\beta = 1$) и 10 итераций интегрированного подхода (последняя комбинация методов). Маршрут для 100 городов был найден с помощью 52 агентов и оператора мутации.

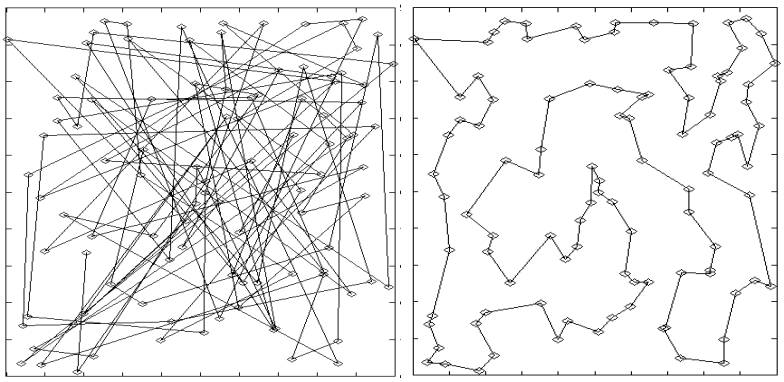

Рис. 5.2 Алгоритм муравьиной колонии и интегрированный подход, 10 итераций

Слабые места алгоритмов были компенсированы сильными сторонами других, что привело к повышению качества полученного решения по сравнения с алгоритмом, взятым за основу (в нашем случае с алгоритмом муравьиной колонии).

Вычислительная сложность последней комбинации методов аналогична сложности метода муравьиной колонии.

6. Реализация муравьиного алгоритма

6.1 Входные параметры

Для того чтобы построить подходящий муравьиный алгоритм для решения какой-либо задачи, нужно:

- Представить задачу в виде набора компонент и переходов или набором неориентированных взвешенных графов, на которых муравьи могут строить решения
- Определить значение следа феромона
- Определить эвристику поведения муравья, когда строим решение
- Если возможно, то реализовать эффективный локальный поиск
- Выбрать специфический ACO алгоритм и применить для решения задачи 6. Настроить параметр ACO алгоритма

Также определяющими являются:

- Количество муравьёв
- Баланс между изучением и использованием
- Сочетание с жадными эвристиками или локальным поиском
- Момент, когда обновляется феромон

6.2 Муравьиный алгоритм для задачи коммивояжёра в псевдокоде

1. Ввод матрицы расстояний D
2. Инициализация параметров алгоритма – α,β,e,Q
3. Инициализация рёбер – присвоение видимости ηij и начальной концентрации феромона
4. Размещение муравьёв в случайно выбранные города без совпадений
5. Выбор начального кратчайшего маршрута и определение L^*

//Основной цикл

6. Цикл по времени жизни колонии t=1,tmax

7. Цикл по всем муравьям k=1,m

8. Построить маршрут Tk (t) по правилу (1) и рассчитать длину Lk (t)

9. конец цикла по муравьям

10. Проверка всех Lk (t) на лучшее решение по сравнению с L*

11. В случае если решение Lk (t) лучше, обновить L* и T*

12. Цикл по всем рёбрам графа

13. Обновить следы феромона на ребре по правилам (2) и (3)

14. конец цикла по рёбрам

15. конец цикла по времени

16. Вывести кратчайший маршрут T* и его длину L*

Сложность данного алгоритма, как несложно заметить, зависит от времени жизни колонии ($t_{\max}$), количества городов (n) и количества муравьёв в колонии (m).

Ниже представлена диаграмма классов, включающая генетический и муравьиный алгоритмы в виде рисунка 6.2.1.

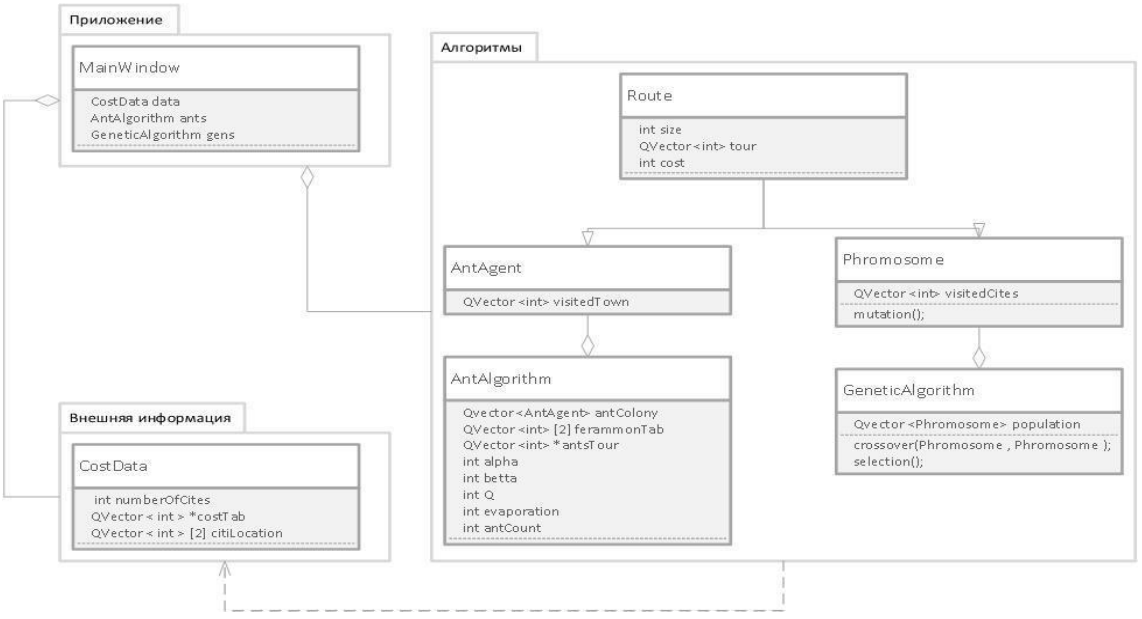

Рис. 6.2.1 Диаграмма классов

Визуализация программы

На рисунке 6.2.2 мы видим строку из среды программирования Visual Studio 2015, в которую вносятся исходные данные программы вычисления задачи почтальона.

```
ogram

t double Inf = 13000000;
к:0
ic void Main(string[] args)

//исходная строка с данными заданного формата
string Input = "0; Иванов Иван; Белгород Победы 81; 0; 0; 2 # 1; Коз
//string input2 = "0; Иванов Иван; Белгород Победы 81; 0; 0; 2 # 1;
//вызов метода getroute для получения маршрута
Console.Write(GetRoute(Input));
//double[] C = GetCoords("белгород ул. проспект славы 50");
//Console.Write(C[0] + " " + C[1] + "\n");
Console.ReadLine();

а:1
ic string GetRoute(string input)
```

Рис. 6.2.2 Строка ввода данных

На рисунке 6.2.3 представлен запуск программы, производящийся после ввода данных из среды программирования Visual Studio, или же после запуска .exe файла.

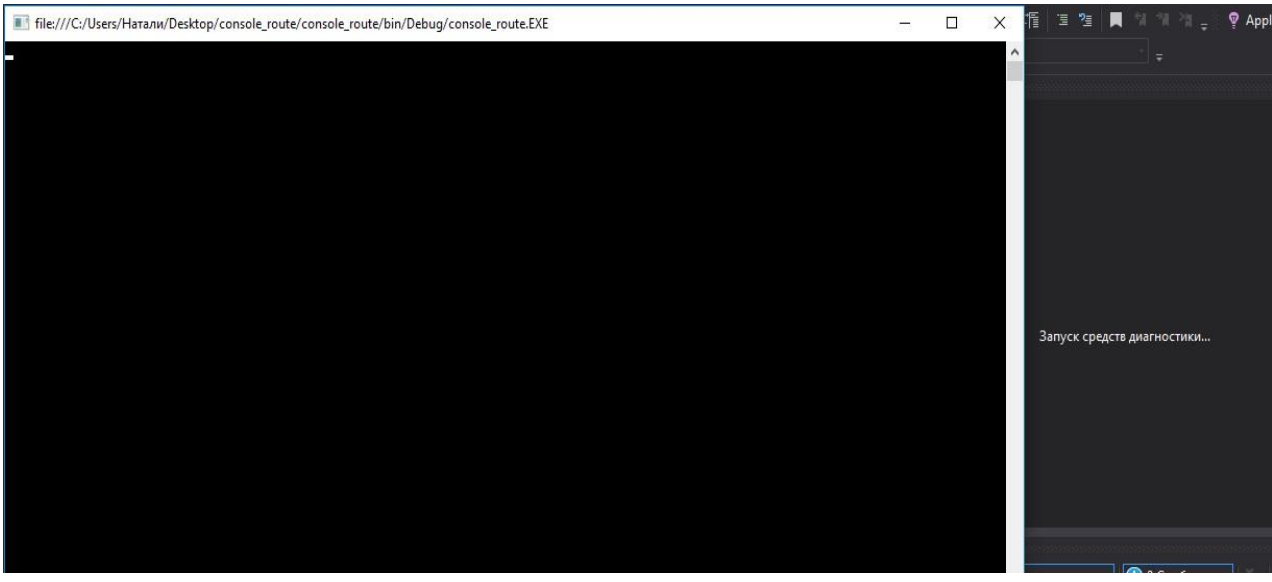

Рис. 6.2.3 Запуск программы

Просматривая рисунок 6.2.4 мы видим результат работы программы для решения задачи почтальона с оптимальным решением.

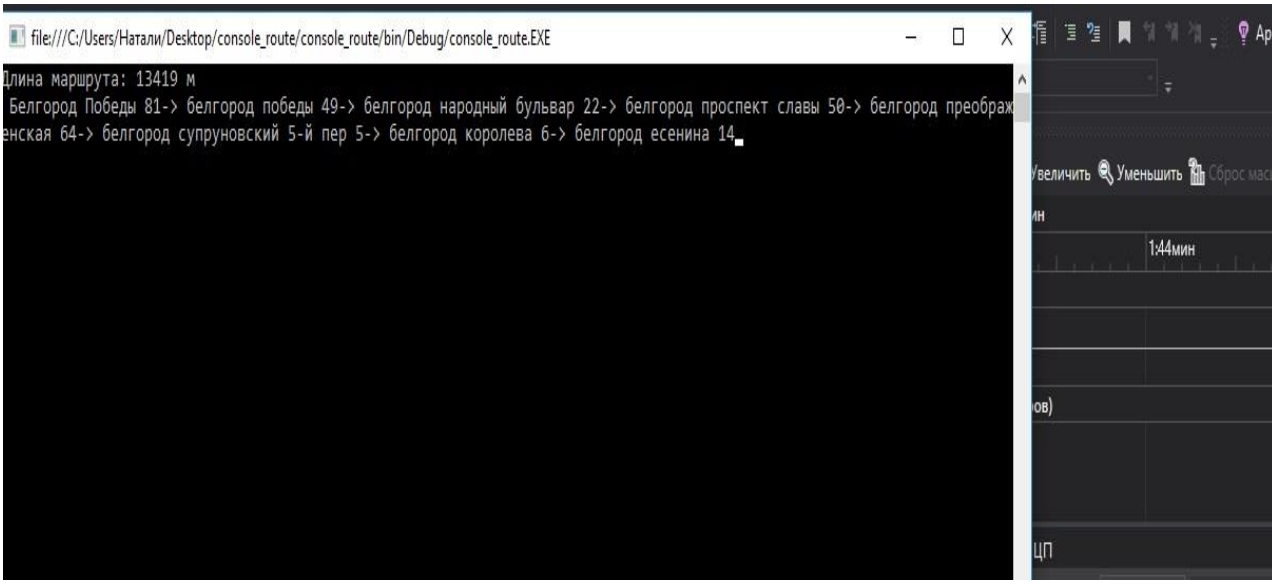

Рис. 6.2.4 Результат полученный в ходе работы программы

Программа имеет фиксированный набор данных, изменение координат возможно из Visual Studio.

Заключение

В результате исследовательской работы был проведен обзор и анализ существующих алгоритмов решения задачи почтальона.

Составлена классификация отобранных методов по отдельным признакам.

Изучен процесс гибридизации и предпринята попытка разработки универсального интегрированного подхода для эффективного синтеза новых оптимизационных методов.

Предложенный муравьиный алгоритм имеет преимущество относительно стандартных методов решения, так как способен адаптироваться к изменениям, варьировать точность в зависимости от требований и выдавать решение, приближенное к точному. Время, затраченное на получение результата, не превышает времени работы алгоритмов, являющихся составными частями интегрированного метода.

Была выполнена работа по реализации и исследованию алгоритмов решения задачи почтальона для ориентированных графов.

В первом разделе была изучена задача почтальона, и вкратце методы ее решения, как точные, так и эвристические.

Во втором разделе были подробно рассмотрены методы решения:

алгоритм полного перебора, метод ветвей и границ, эвристические алгоритмы, алгоритм ближайшего соседа, поведенческие (роевые) алгоритмы, метод муравьиной колонии, алгоритм искусственной иммунной системы, алгоритм интеллектуальных капель, алгоритм имитации отжига, генетический алгоритм.

В третьем разделе проводится сравнительный анализ приведенных алгоритмов.

Четвертый раздел описывает гибридные системы решения задачи почтальона, далее приводиться схема интегрированного поиска.

Данная выпускная квалификационная работа включает в себя теоретическую часть, практическую часть, шесть таблиц, пятнадцать рисунков и десять формул.

Алгоритмы, исследованные в рамках работы, реализованы в среде Visual Studio 2015.

Список литературы

1. Dorigo M., Maniezzo V., Colorni A. The Ant System: Optimization by a colony of cooperating objects // IEEE Trans. on Systems, Man, and Cybernetics. – 1996. – Part B. – N 26(1) – pp. 29 – 41.

2. Zaporozhets, D.U.; Zaruba, D.V.; Kureichik, V.V. Representation of solutions in genetic VLSI placement algorithms // Design & Test Symposium (EWDTS), East-West. 1 - 4, 2014 3. Беспалько В. П. Образование и обучение с помощью компьютеров / В.
П. Беспалько. – М. : Высш. шк., 1998. – 135 с.

4. Борознов В. О. Дополнение метода ветвей и границ для решения задачи коммивояжера // Вестн. Ростов. гос. ун-та путей сообщения. – 2007. – № 1. – С. 160-163. 5. Борознов В. О. Исследования генетических методов решения задачи коммивояжёра / В. О. Борознов, О. Г. Ведерникова // Вестн. Ростов. гос. ун-та путей собщения. – 2004. – № 1. – С. 42-45.

6. Боронихина Е. А. Эвристический метод решения задачи коммивояжера // Новые информационные технологии в исследовании сложных структур. Материалы девятой Российской конференции с международным участием. Томск: Изд-во НТЛ, 2014.

7. Боронихиной Е. А. «Муравьиный алгоритм для решения задачи коммивояжера» - Материалы II Всероссийской молодежной научной конференции с международным участием «Математическое и программное обеспечение информационных, технических и экономических систем». Томск, 2014 г.

8. Боронихина Е. А., Сибирякова В.А., Сравнение методов решения задачи коммивояжера – Инфомационные технологии и математическое моделирование И74 (ИТММ-2014): Материалы XIII Международной научнопрактической конференции имени А. Ф. Терпугова (20-22 ноября 2014г.) . – Томск: издательство Том. Ун-та, 2014. – С 18-21.

9. Боронихина Е. А. Сравнение методов решения задачи коммивояжера – Материалы 53-й Международной научной студенческой конференции МНСК-2015: Информационные технологии / Новосиб. гос. ун-т. Новосибирск, 2015. С 82.

10. Боронихина Е. А. Интегрированный подход к нахождению решений задачи коммивояжера, с использованием бионспирированных методов - материалы XXXVII студенческой международной заочной научно-практической конференции Новосибирск 2015, С 67-71

11. Боронихиной Е. А. Точные и эвристические методы решения задачи – материалы XIX Всероссийской научно-практической конференции «Научное творчество молодежи. Математика. Информатика», Томск 2015. С 94-97.

12. Боронихиной Е. А. Точные и эвристические методы решения задачи коммивояжера» - Материалы III Всероссийской молодежной научной конференции «Математическое и программное обеспечение информационных, технических и экономических систем». Томск, 2015 г. С 203-205

13. Боронихина Е. А. Интегрированный подход к нахождению решений задачи коммивояжера, с использованием методов, инспирированных природными системами – Сборник тезисов докладов участников XIII Всероссийского молодежного форума проблем культурного наследия, экологии и безопасности жизнедеятельности «Юнеко – 2015». – М., 2015. – С 290-291.

14. Боронихина Е. А. Интегрированный подход к нахождению решения задачи коммивояжера – Сборник материалов всероссийской молодежной школы семинара «Актуальные проблемы информационных технологий, электроники и радиотехники – 2015» Том 2 – Таганрог: Изд-во НОЦ ЗИС КТ Южного федерального университета, 2015. – С 229-233.

15. Боронихина Е. А. Решение задачи коммивояжера с помощью интегрированного метода Сборник материалов III Международной научнопрактической конференции – Кемерово, январь 2016 г. – С 278-280.

16. Боронихиной Е. А. Исследование гибридных эвристических методов решения задачи коммивояжера - Материалы IV Всероссийской молодежной научной конференции «Математическое и программное обеспечение вычислительных машин и компьютерных сетей.». Томск, 2016г.

17. Гладков Л. А., Курейчик В. М., Курейчик В. В. Генетические алгоритмы / Л. А. Гладков. – Ростов-на-Дону: ООО «Ростиздат», 2004г.

18. Запорожец Д.Ю., Заруба Д.В., Лежебоков А.А. Об одном способе кодирования решения для задачи размещения. Известия ЮФУ. Технические науки. 2012. № 11 (136). С. 183-188.

19. Интернет ресурс http://chelyabinsk.ru/text/eco/525308.html - Челябинск.ру, время обращения 15.10.2015

20. Капра Ф. Паутина жизни. Новое научное понимание живых систем. Перевод с английского – М.: ИД “Гелиос”, 2002.

21. Клаг У., Каммингс М. Основы генетики. – М.: Техносфера, 2007.

22. Курейчик В.В. Эволюционные методы решения оптимизационных задач. Таганрог, 1999, ТРТУ.

23. МакКоннелл Дж. Основы современных алгоритмов // Дж. МакКоннелл. – М.: Техносфера, 2004. – 368 с.

24. Петухова А. А. Формирование умений студентов в ознакомительном чтении с использованием компьютерной обучающей программы / А. А. Петухов. – Таганрог : ТРТУ, 2001. – 217 с.

25. Рапопорт Г.Н., Герц А.Г. Искусственный и биологические интеллекты. Общность структуры, эволюция и процессы познания.- М.: Комкнига, 2005.

26. Ридли М. Геном: автобиография вида в 23 главах // Перевод с английского –М.:Эксмо, 2008.

27. Романовский И. В. Алгоритмы решения экстремальных задач / И. В.

Романовский. – М. : Наука, 1977. – 352 с.

28. Рутковская Д. Пилиньски М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. – М.: Горячая линия – Телеком, 207. 452с.

29. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. -М.: Эдиториал УРСС, 2002. -352с.

30. Хакен Г. Тайны природы. Синергетика: учение о взаимодействии. – Москва - Ижевск: Институт компьютерных исследований, 2003. 31. Хедрик Ф. Генетики популяции. – М.: Техносфера, 2003. – 592с.

ПРИЛОЖЕНИЕ

Листинг программного кода

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Newtonsoft.Json;
using System.Globalization;
using System.Net;
using System.IO;
using Newtonsoft.Json.Linq;

namespace console_route
{
    class Program
    {
        const double Inf = 13000000;
        static void Main(string[] args)
        {
            string Input = "0; Иванов Иван; Белгород Победы 81;
0; 0; 2 # 1; Козлов Николай; белгород победы 49; 0; 0; 1 # 2;
Сидоров Михаил; белгород проспект славы 50; 0; 0; 5 #3; Петров
Петр; белгород народный бульвар 22; 0; 0; 2 #4; Сергеев Сергей;
белгород есенина 14; 0; 0; 15 # 5; Тимофеев Тимофей; белгород
преображенская 64; 0; 0; 10 # 6; Васильев Василий; белгород
супруновский 5-й пер 5; 0; 0; 7 # 7; Павлов Павел; белгород
королева 6; 0; 0; 4";
            Console.Write(GetRoute(Input));

            Console.ReadLine();
        }
        static string GetRoute(string input)
        {
            string output = "";
            string[] Address;
            double[] Latitude;
            double[] Longitude;
            byte[] Priority;
            int[] ID;
            string[] Name;
            StringToData(input, out Address, out Latitude, out
Longitude, out Priority, out ID, out Name);
```

```
            double[,] R = GetMatrixDistances(Latitude, Longitude,
Latitude.Length);
            double[,] M = R;
            int[] D = ExhaustiveSearch(M);
            int N = D.Length;
            Console.WriteLine("Длина маршрута: " +
CountDistance(D, R) + " м");
           CountDistance(D, M) + " м/пр");
            int k = 0;
            for (int i = 0; i < N; i++)
            {
                output += Address[k] + "->";
                k = D[k];
            }
            return output.Remove(output.Length - 2);
        }
        static int[] GreedyAlgorithm(double[,] MatrixDistances)
        {
            int N = (int)Math.Sqrt(MatrixDistances.Length);
            int[] D = new int[N];
            for (int j = 0; j < N; j++) D[j] = -1;
            bool[] Used = new bool[N];
            double min;
            int min_j = 0;
            int i = 0;
            for (int k = 0; k < N; k++)
            {
                min = Inf;
                for (int j = 1; j < N; j++)
                    if (MatrixDistances[i, j] < min && !Used[j])
{ min = MatrixDistances[i, j]; min_j = j; }
                Used[i] = true;
                Used[min_j] = true;
                D[i] = min_j;
                i = min_j;
            }
            return D;
        }
        static int[] ExhaustiveSearch(double[,] MatrixDistances)
        {
            int N = (int)Math.Sqrt(MatrixDistances.Length);
            int[] D = new int[N];
            for (int i = 0; i < N; i++) D[i] = i;
            int[] min_D = new int[N];
```

```
            double min_R = Inf;
            List<int[]> P = new List<int[]>();
            P = GetPermutations(D);
            int M = P.Count();
            for (int i = 0; i < M; i++)
                if (IsCorrect(P[i]))
                {
                    double R = CountDistance(P[i],
MatrixDistances);
                    if (R < min_R)
                    {
                        min_R = R;
                        P[i].CopyTo(min_D, 0);
                    }
                }
            return min_D;
        }
        static int[] AntColonyOptimization(double[,]
MatrixDistances)
        {
            const double A = 1.5;
            const double B = 3.5;
            const double p = 0.2;
            const double E = 100;
            const double F = 2.5;
            double Q = 100;
            int N = (int)Math.Sqrt(MatrixDistances.Length);
            int[] D = new int[N];
            double[,] u = new double[N, N];
            Random Rnd = new Random();
            Random Rnd_p = new Random();
            bool[] Used = new bool[N];
            double[] P = new double[N];
            double[,] T = new double[N, N];
            int steps = 0;
            int i;
            steps++;
            for (i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                {
                    T[i, j] = Rnd.Next(0, F > 1 ? (int)F : 1);
                    u[i, j] = 1 / MatrixDistances[i, j];
                }
            double Lo = 0, Ln = 0;
```

```
            bool flag = true;
            do             {
                i = 0;
                for (int k = 0; k < N; k++)
                {
                    Used[k] = false;
                    D[k] = -1;
                    P[k] = 0;
                }
                Used[i] = true;
                for (int z = 0; z < N - 1; z++)
                {
                    for (int k = 0; k < N; k++) P[k] = 0;
                    double Ps = 0;
                    for (int k = 0; k < N; k++)
                        if (!Used[k] && k != i) Ps +=
Math.Pow(u[i, k], B) * Math.Pow(T[i, k], A);
                    for (int j = 0; j < N; j++)
                    {
                        if (i == j || Used[j]) continue;
                        P[j] = (Math.Pow(u[i, j], B) *
Math.Pow(T[i, j], A)) / Ps;
                    }
                    double x = Rnd_p.NextDouble();
                    double px = 0;
                    int t = 0;
                    while (px <= x)
                    {
                        px += P[t];
                        t++;
                    }
                    if (t > 0) t--;
                    Used[t] = true;
                    D[i] = t;
                    i = t;

                }
                Lo = Ln;
                Ln = CountDistance(D, MatrixDistances);
                double dT;
                if (flag) { flag = false; Q = F * Ln; }
                dT = Q / Ln;
                int y = 0;
                for (int k = 0; k < N - 1; k++)
```

```
                {
                    T[y, D[y]] = (1 - p) * T[y, D[y]] + dT;
                    y = D[y];
                }
            }
            while (Math.Abs(Ln - Lo) > E);
            return D;
        }
        static bool IsCorrect(int[] D)
        {
            int N = D.Length;
            bool[] Used = new bool[N];
            int k = 0, i = 0;
            for (int m = 0; m < N; m++)
                if (!Used[D[i]]) { Used[D[i]] = true; k++; i =
D[i]; }
            return k == N;
        }

        static double[,] ReduceMatrix(double[,] MatrixDistances)
        {
            int N = (int)Math.Sqrt(MatrixDistances.Length);
            double[,] M = Copy2DArray(MatrixDistances);
            for (int i = 0; i < N; i++)
            {
                double min_i = Inf;
                for (int j = 0; j < N; j++)
                {
                    if (i == j) continue;
                    if (M[i, j] < min_i) min_i = M[i, j];
                }
                for (int j = 0; j < N; j++)
                {
                    if (i == j) continue;
                    M[i, j] -= min_i;
                }
            }
            for (int j = 0; j < N; j++)
            {
                double min_j = Inf;
                for (int i = 0; i < N; i++)
                {
                    if (i == j) continue;
                    if (M[i, j] < min_j) min_j = M[i, j];
```

```
                }
                for (int i = 0; i < N; i++)
                {
                    if (i == j) continue;
                    M[i, j] -= min_j;
                }
            }
            return M;
        }
        static double[,] SetPriority(double[,] Distances, byte[]
Priority)
        {
            int N = (int)Math.Sqrt(Distances.Length);
            double[,] M = Copy2DArray(Distances);
            for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                    M[i, j] /= PriorityFunction(Priority[j]);
            return M;
        }
        static byte PriorityFunction(byte priority)        {
            return priority;
        }
        static void StringToData(string input, out string[]
Address, out double[] Latitude,
            out double[] Longitude, out byte[] Priority, out
int[] ID, out string[] Name)
        {
            string[] s = input.Split('#');
            int N = s.Length;
            Address = new string[N];
            Latitude = new double[N];
            Longitude = new double[N];
            Priority = new byte[N];
            ID = new int[N];
            Name = new string[N];
            bool flag = false;
            for (int i = 0; i < N; i++)
            {
                s[i] = s[i].Trim();
                try { ID[i] = int.Parse(s[i].Split(';')[0]); }
                catch { ID[i] = i; }
                Name[i] = s[i].Split(';')[1];
                Address[i] = s[i].Split(';')[2];
                try
```

```
                {
                    if (Latitude[i] == 0 || Longitude[i] == 0)
                    {
                        double[] C = GetCoords(Address[i]);
                        Latitude[i] = C[0];
                        Longitude[i] = C[1];
                    }
                    else
                    {
                        Latitude[i] =
double.Parse(s[i].Split(';')[3]);
                        Longitude[i] =
double.Parse(s[i].Split(';')[4]);
                    }
                }
                catch
                {
                    double[] C = GetCoords(Address[i]);
                    if (i != 0 && C[0] == 50 && C[1] == 60) {
C[0] = Latitude[i - 1]; C[1] = Longitude[i - 1]; }
                    Latitude[i] = C[0];
                    Longitude[i] = C[1];
                }
                try
                {
                    if (!flag && Priority[i] == 0) flag = true;
                    Priority[i] = byte.Parse(s[i].Split(';')[5]);
                }
                catch { Priority[i] = 2; }
            }
            if (flag)
                for (int i = 0; i < N; i++) Priority[i]++;

        }
        static bool IsComplete(int[] Route)
        {
            int N = Route.Length;
            for (int i = 0; i < N; i++)
                if (Route[i] == -1) return false;
            return true;
        }
        static double[,] GetMatrixDistances(double[] Latitude,
double[] Longitude, int Length)
        {
```

```
            double[,] M = new double[Length, Length];
            for (int i = 0; i < Length; i++)
                for (int j = 0; j < Length; j++)
                    if (i != j)
                        M[i, j] = CountDistance(Latitude[i],
Longitude[i], Latitude[j], Longitude[j]);
                    else M[i, j] = Inf;
            return M;
        }
        static double CountDistance(double latitude1, double
longitude1, double latitude2, double longitude2)
        {
            double d;
            try
            {
                var url =
String.Format(CultureInfo.InvariantCulture,
"http://maps.googleapis.com/maps/api/directions/json?origin={0},{
1}&destination={2},{3}&sensor=false&mode=driving&key=AIzaSyCGraxC
eHZ6fy0k_sKsOchhYyTpwsDXjS0", latitude1, longitude1, latitude2,
longitude2);
                var request = WebRequest.Create(url);
                var response = request.GetResponse();
                var responseString = new
StreamReader(response.GetResponseStream()).ReadToEnd();
                var result = JObject.Parse(responseString);
                d =
(double)result["routes"][0]["legs"][0]["distance"]["value"];
                }
            catch
            {
                try
                {
                    var url =
String.Format(CultureInfo.InvariantCulture,
"http://osrm.logexpert.ru:5000/viaroute?z=14&loc={0},{1}&loc={2},
{3}&geometry=false", latitude1, longitude1, latitude2,
longitude2);
                    var request = WebRequest.Create(url);
                    var response = request.GetResponse();
                    var responseString = new
StreamReader(response.GetResponseStream()).ReadToEnd();
                    var result = JObject.Parse(responseString);
```

```
                    d =
(double)result["route_summary"]["total_distance"];
                }
                catch                {
                    latitude1 *= Math.PI / 180;
                    latitude2 *= Math.PI / 180;
                    longitude1 *= Math.PI / 180;
                    longitude2 *= Math.PI / 180;
                    double a = Math.Pow(Math.Cos(latitude2) *
Math.Sin(longitude2 - longitude1), 2);
                    double b = Math.Pow(Math.Cos(latitude1) *
Math.Sin(latitude2) - Math.Sin(latitude1) * Math.Cos(latitude2) *
Math.Cos(longitude2 - longitude1), 2);
                    double c = Math.Sin(latitude1) *
Math.Sin(latitude2) + Math.Cos(latitude1) * Math.Cos(latitude2) *
Math.Cos(longitude2 - longitude1);
                    d = Math.Atan2(Math.Sqrt(a + b), c);
                    const int R = 6372795;
                    d *= R;
                }
            }
            return d;
        }
        static double[] GetCoords(string address)
        {
            double[] C = new double[2];
            try             {
                String.Format(CultureInfo.InvariantCulture,
"http://openstreetmap.ru/api/search?q={0}", address);
                var url =
String.Format(CultureInfo.InvariantCulture, "https://geocode-
maps.yandex.ru/1.x/?format=json&geocode={0}", address);
                var request = WebRequest.Create(url);
                var response = request.GetResponse();
                var responseString = new
StreamReader(response.GetResponseStream()).ReadToEnd();
                var result = JObject.Parse(responseString);
double.Parse(result["response"]["featureMember"].ToString().Split
(' ')[1]);
                string x =
result["response"]["GeoObjectCollection"]["featureMember"][0]["Ge
oObject"]["Point"]["pos"].ToString().Split(' ')[1];
                C[0] =
double.Parse(result["response"]["GeoObjectCollection"]["featureMe
```

```
mber"][0]["GeoObject"]["Point"]["pos"].ToString().Split('
')[1].Replace('.', ','));
                C[1] =
double.Parse(result["response"]["GeoObjectCollection"]["featureMe
mber"][0]["GeoObject"]["Point"]["pos"].ToString().Split('
')[0].Replace('.', ','));
            }
            catch
            {
                try
                {
                    var url =
String.Format(CultureInfo.InvariantCulture,
"http://openstreetmap.ru/api/search?q='{0}'", address);
                    var request = WebRequest.Create(url);
                    var response = request.GetResponse();
                    var responseString = new
StreamReader(response.GetResponseStream()).ReadToEnd();
                    var result = JObject.Parse(responseString);
                    C[0] = (double)result["matches"][0]["lat"];
                    C[1] = (double)result["matches"][0]["lon"];
                }
                catch
                {
                    C[0] = 50;
                    C[1] = 60;
                }

            }
            return C;
        }
        private static int[] Swap(int[] D, int a, int b)
        {
            if (a == b) return D;
            int[] P = new int[D.Length];
            D.CopyTo(P, 0);
            int tmp = P[a];
            P[a] = P[b];
            P[b] = tmp;
            return P;
        }

        public static List<int[]> GetPermutations(int[] P)
        {
```

```
            List<int[]> A = new List<int[]>();
            int x = P.Length - 1;
            GeneratePermutation(A, P, 0, x);
            return A;
        }

        private static void GeneratePermutation(List<int[]> A,
int[] P, int k, int m)
        {
            if (k == m)
            {
                A.Add(P);
            }
            else
                for (int i = k; i <= m; i++)
                {
                    P = Swap(P, k, i);
                    GeneratePermutation(A, P, k + 1, m);
                    P = Swap(P, k, i);
                }
        }
  static double CountDistance(int[] D, double[,] MatrixDistances)
        {
            int N = D.Length;
            int k = 0;
            double R = 0;
            for (int i = 0; i < N - 1; i++)
            {
                R += MatrixDistances[k, D[k]];
                k = D[k];
            }
            return R;
        }
        static double[,] Copy2DArray(double[,] Array)
        {
            int N = (int)Math.Sqrt(Array.Length);
            double[,] X = new double[N, N];
            for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                    X[i, j] = Array[i, j];
            return X;
        }
    }
}
```