



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт кибернетики

(наименование института, филиала)

Кафедра проблем управления

(наименование кафедры)

ОТЧЁТ ПО ПРАКТИКЕ ППУ И ОПД

(указать тип практики)

Тема практики: Разработка и исследование муравьиных алгоритмов для
решения задач планирования целенаправленных перемещений автономных
роботов

приказ университета о направлении на практику

от «10» 02 2020г. № 759-с

Отчет представлен к

рассмотрению:

Студент группы КРБО-02-17

(подпись)

Константинов М.В.

(расшифровка подписи)

«22» 02 2021 г.

Отчет утвержден.

Допущен к защите:

Руководитель практики

от кафедры

(подпись)

С.В. Манько

(расшифровка подписи)

«22» 02 2021 г.

Руководитель практики

от Университета

(подпись)

А.А. Сухоленцева

(расшифровка подписи)

« » 2021 г.

Проверен работоспособность оценки «отлично»

Москва 2020



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт кибернетики

(наименование института, филиала)

Кафедра проблем управления

(наименование кафедры)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ ПО ППУ И ОПД

(указать вид практики)

Студента 3 курса учебной группы КРБО-02-17

Константинова Максима Вячеславовича

Место и время практики: Межкафедральная специализированная учебно-научная лаборатория "Интеллектуальные автономные и мультиагентные робототехнические системы"; Лаборатория для самостоятельных занятий. С 10.02.2020 по 30.05.2020.

Должность на практике:

1. ЦЕЛЕВАЯ УСТАНОВКА: Разработка и исследование муравьиных алгоритмов для решения задач планирования целенаправленных перемещений автономных роботов

2. СОДЕРЖАНИЕ ПРАКТИКИ:

2.1 Изучить: Особенности построения и возможности применения для решения задач планирования целенаправленных перемещений автономных роботов

2.2 Практически выполнить: Разработка программных средств для решения задач планирования целенаправленных перемещений автономных роботов; проведение модельных экспериментов

2.3 Ознакомиться: с особенностями использования муравьиных алгоритмов в задачах оптимизации

3. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ:

4. ОГРАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:

Заведующий кафедрой:

«__» ____ 20__ г.

(подпись)

М.П. Романов
(ФИО)

СОГЛАСОВАНО:

Руководитель практики от кафедры:

«17» 02 2020 г.

(подпись)

С.В. Манько
(ФИО)

Руководитель практики от Университета:

«__» ____ 20__ г.

(подпись)

А.А. Сухоленцева
(ФИО)

Задание получил:

«17» 02 2020 г.

(подпись)

М.В. Константинов
(ФИО)

Проведенные инструктажи:

Охрана труда:

«17» 02 2020г.

Инструктирующий

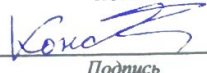


Подпись

(МАНЬКО С.В., профессор, д.т.н)

Расшифровка, должность

Инструктируемый



Подпись

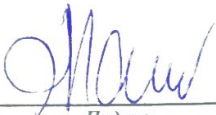
(КОНСТАНТИНОВ М.В.)

Расшифровка

Техника безопасности:

«17» 02 2020г.

Инструктирующий

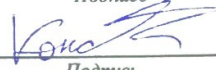


Подпись

(МАНЬКО С.В., профессор, д.т.н)

Расшифровка, должность

Инструктируемый



Подпись

(КОНСТАНТИНОВ М.В.)

Расшифровка

Пожарная безопасность:

«17» 02 2020г.

Инструктирующий



Подпись

(МАНЬКО С.В., профессор, д.т.н)

Расшифровка, должность

Инструктируемый



Подпись

(КОНСТАНТИНОВ М.В.)

Расшифровка

С правилами внутреннего распорядка ознакомлен: «17» 02 2020г.



Подпись

(КОНСТАНТИНОВ М.В.)

Расшифровка



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

**РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ
ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

студента 3 курса группы КРБО-02-17 очной формы обучения,
обучающегося по направлению подготовки «Мехатроника и
робототехника», профиль «Автономные роботы»

Неделя	Сроки выполнения	Этап	Отметка о выполнении
1-6	10.02.2020 — 22.03.2020	Изучение алгоритмов для решения задач планирования перемещений автономных роботов	Выполнено
7-8	23.03.2020 — 05.04.2020	Изучение современных исследований по использованию муравьиных алгоритмов в задачах оптимизации	Выполнено
9-13	06.04.2020 — 10.05.2020	Разработка программного обеспечения для изучения муравьиных алгоритмов. Проведение модельных экспериментов	Выполнено
14-16	11.05.2020 — 31.05.2020	Оформление общего отчёта по практике	Выполнено

Согласовано:

Заведующий кафедрой

М.П. Романов

Руководитель практики
от кафедры

С.В. Манько

Руководитель практики
от Университета

А.А. Сухоленцева

Обучающийся

М.В. Константинов

ОТЧЕТ
по прохождению учебной практики
студента 3 курса учебной группы КРБО-02-17
института кибернетики

Константинова Максима Вячеславовича

1. Практику проходил с 10.02.2020 по 30.05.2020 в межкафедральной специализированной учебно-научной лаборатории "Интеллектуальные автономные и мультиагентные робототехнические системы".
2. Задание на практику выполнил в полном объеме.

Подробное содержание выполненной на практике работы и достигнутые результаты: были изучены муравьиные алгоритмы как метод решения различных задач оптимизации, разработано ПО для исследования муравьиных алгоритмов в задаче построения оптимального маршрута.

Предложения по совершенствованию организации и прохождения практики: предложений нет.

Студент Константинов (Константинов М.В.)

«22» 02 2021 г.

Заключение руководителя практики от профильной организации:

Приобрел следующие профессиональные навыки: разработка и отладка ПО

Сбор и анализ необходимой научно-исследовательской литературы, обработка экспериментальных данных

Проявил себя как: инициатор, инициатор, инициатор и

инициатор, инициатор, инициатор

инициатор, инициатор, инициатор

инициатор, инициатор, инициатор

Руководитель практики от профильной организации

профессор, Д.Т.Н.
(должность)

(наименование от профильной организации)

(подпись)

Маниско С.В.
(фамилия и инициалы)

Отчет проверил:

Руководитель практики от Университета

(подпись)

(фамилия и инициалы)

Оглавление

Введение	8
1. Муравьиные алгоритмы как алгоритмы оптимизации	
1.1. Появление алгоритма.....	9
1.2. Общее описание алгоритма	11
1.3. Псевдокод реализации алгоритма	15
1.4. Модификации алгоритма	16
1.5. Практическое использование муравьиного алгоритма	19
2. Экспериментальное исследование муравьиного алгоритма	
2.1. Использование алгоритма для нахождения кратчайшего пути на местности.....	20
2.2. Общее описание программы	24
2.3. Основная структура программы	27
2.4. Проведение экспериментального исследования алгоритма	
2.4.1 Условия проводимого исследования	30
2.4.2. Процесс нахождения оптимального пути	32
2.4.3. Нахождение оптимальных значений α и β	34
2.4.4. Нахождение оптимальных значений Q , ρ и количества муравьёв.....	36
2.5. Использование модификаций алгоритма	
2.5.1. Модификация элитных муравьёв	48
2.5.2. Модификация ASrank	49
2.5.3. Метод локального поиска	52
2.6. Сравнение муравьиного алгоритма с другими алгоритмами	
2.6.1. Алгоритм полного перебора.	53
2.6.2. Генетический алгоритм	54
3. Целесообразность использования муравьиных алгоритмов	
3.1. Преимущества	57
3.2. Недостатки	57
Заключение	59

Список источников.....	60
Приложение 1.....	63

Введение

Оптимизация процесса всегда является одной из важнейших частей в любой области, чтобы снизить количество затрачиваемого времени и ресурсов. В последнее время с развитием техники появилась возможность замены человека автоматическим управлением, что позволяет реализовать всевозможные алгоритмы для оптимизации и ускорения различных процессов.

Одним из довольно перспективных алгоритмов, который разносторонне исследуется и уже на данный момент широко используется - это муравьиный алгоритм (Ant Colony Optimisation – ACO, Ant Systems – AS), который был создан на примере поведения муравьёв в природе.

В рамках практической работы будут изучены статьи об уже ранее проведённых работах с использованием ACO для решения различных задач, а также работ по усовершенствованию алгоритма. Также будет проведено исследование работы ACO на примере задачи поиска кратчайшего маршрута. В итоге будут выявлены преимущества и недостатки использования ACO и подведён вывод.

1. Муравьиные алгоритмы как алгоритмы оптимизации

1.1. Появление алгоритма

В природе муравьи, существуя в виде колоний и взаимодействуя между собой образуют так называемый роевой интеллект. Каждый из них сам по себе не видит общей картины и не может принимать какое-либо осознанное решение, а лишь опирается на механизмы, развитые эволюцией и заложенные в него с его появления. По большей части общение внутри колонии производится неявно, при помощи феромонов (такой вид взаимодействия называется стигмергией). И, хоть по одиночке муравьи довольно незначительны, вместе, при помощи коллективной работы большого количества особей, удаётся достигать немыслимых результатов.

Способ установления кратчайшего пути до источника пищи у муравьёв довольно прост: муравей, проходя от муравейника до пищи и обратно оставляет за собой след феромона, который увлекает других особей и позволяет им добраться до пропитания. Чем короче путь, который был найден, тем больше особей за определённый промежуток времени сможет его пройти, и, следовательно, тем сильнее будет феромоновый след и тем сильнее он будет увлекать других особей. А из этого следует, что со временем, рано или поздно, будет найден и зафиксирован самый короткий путь без возможности его случайного изменения на менее оптимальный.

Также этот способ быстро адаптируется к динамической среде, реагируя на изменение расстояний, а также прерывание или появление новых путей. Увидеть это можно на примере поведения муравьёв при поведении препятствия на дороге. В изначальный момент пути с обеих сторон от препятствия равнозначны, однако короткий путь будет быстрее проходиться муравьями, из чего следует, что на этом пути феромоновый след будет сильнее, а со временем более длинный путь вовсе перестанет использоваться в пользу найденного короткого.

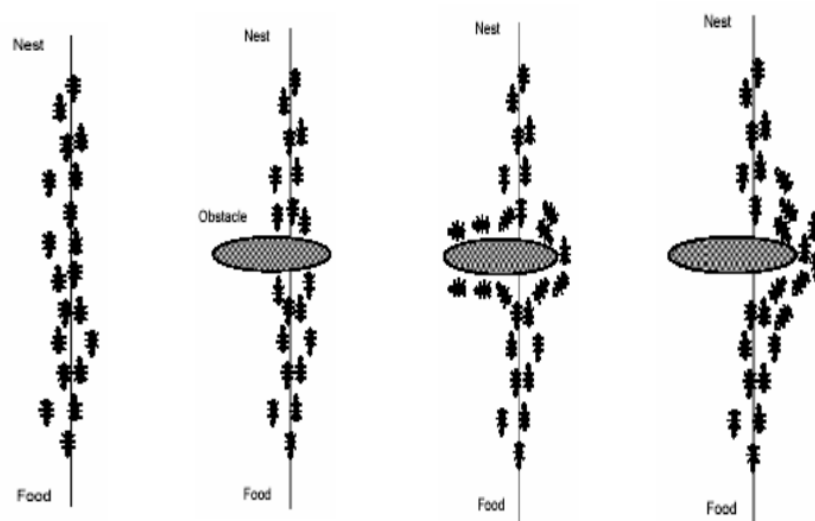


Рисунок 1.1. Иллюстрация приспособления муравьёв к появлению препятствия на пути

Предложение о заимствовании этого природного механизма для использования в технике было сделано Марко Дориго в 1991 году [1, 2, 3, 4]. С тех пор началось активное развитие алгоритма, и на сегодняшний день уже получены хорошие результаты для решения таких сложных комбинаторных задач, как задача коммивояжера, задача раскраски графа, задача оптимизации сетевых графиков, задача календарного планирования и многие другие. И хоть уже создано большое количество модификаций алгоритма, основной принцип остаётся один.

1.2. Общее описание алгоритма

Муравьиный алгоритм моделирует многоагентную систему, каждый агент которой выполняет роль «муравья», функционируя по простым правилам, совершая простые вычисления и имея минимальный запас памяти, который необходим только для запоминания пройденных точек. Из пройденных точек оставляется так называемый Список запретов (tabu list), который пополняется посещёнными точками во время перемещения и обнуляется с каждым новым циклом итерации. Этот список крайне необходим для того, чтобы «муравей» не «сделал круг» и не пошёл по ранее учтённой во время движения точки, а дальше развивал свой путь. Важность наличия такой памяти можно увидеть на примере поведения муравьёв в природе.

В природе муравей, не имея возможности запоминать пройденный путь и не в состоянии сделать какое-либо другое решение кроме как пойти по феромоновому следу, нередко увлекается своим же недавно оставленным следом и начинает проходить по кругу снова и снова, с каждым кругом закрепляя свой феромоновый след и увлекая в него других особей, что почти всегда приводит к тому, что они двигаются по кругу до своей гибели, подчиняясь природным механизмам и не в состоянии сойти с него. Такое явление получило название «Муравьиная спираль смерти».

В алгоритме, разработанном на основании поведения муравьёв, такая возможность исключается, поскольку агенту программно запрещён повторный переход в точки, находящиеся в списке запретов.

При выборе точки дальнейшего перемещения, кроме списка запретов, агент руководствуется «привлекательностью» доступных путей на основании их длины и уровню феромонового следа, оставленного на нём. В отличие от расстояния, количество феромона на определённом пути не постоянно, а всё время обновляется, как уменьшаясь со временем (как и в природе, «испаряясь»), так и увеличиваясь из-за других агентов, проходящих по этому-же пути и оставивших за собой такой же феромоновый след.

Для определения вероятности прохождения агента, находящегося в точке i по определённому пути, связывающего вершину i с соседствующей вершиной j используется следующая формула:

$$P_{ij}(t) = \frac{\tau_{ij}(t)^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}{\sum_{j \in \text{allowed nodes}} \tau_{ij}(t)^\alpha \left(\frac{1}{d_{ij}}\right)^\beta}$$

Формула 1.1. Расчёт вероятности выбора определённого ребра среди остальных

где:

t – время.

τ_{ij} – уровень феромона на пути, связывающем вершины i и j .

d_{ij} – «вес» связи, расстояние между вершинами i и j .

α и β – настраиваемые коэффициенты. Как можно увидеть, в зависимости от параметра α зависит то, насколько сильно влияет количество феромона на вероятность выбора определённой связи. А параметр β влияет на зависимость вероятности выбора связи от расстояния между вершинами.

То есть, при $\alpha = 0$ проложенные феромоновые дорожки практически бесполезны и выбор вершины зависит только от расстояния до неё, что практически лишает алгоритм заложенного в него смысла и превращает его в подобие «жадного» алгоритма.

При $\beta = 0$ выбор пути наоборот, производится только на основании феромонов, отчего резко возрастает вероятность прихода к субоптимальному пути.

Необходимо нахождение некоторого компромиссного решения и баланса между величинами этих коэффициентов, что достижимо опытным путём.

Количество оставляемого одним агентом феромона на пути между соседствующими точками i и j определяется по следующей формуле:

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

Формула 1.2. Расчёт количества феромона, оставляемого муравьём на пройденном пути

где:

T_k – множество вершин, пройденных во время пути.

Q – регулируемый параметр, имеющий значение порядка длины оптимального пути.

L_k – длина маршрута муравья.

Таким образом, количество феромона, оставляемого на пути одним агентом обратно пропорционально длине пройденного маршрута, что способствует отметанию менее оптимальных путей в пользу кратчайшего, поскольку на нём агентом, прошедшим этим путём, оставляется большее количество феромона.

На самом пути количество феромона, связывающем две соседние вершины в графе описывается следующей формулой:

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \sum_{\substack{k \in Colony \text{ that} \\ \text{used edge } (i, j)}} \frac{Q}{L_k}$$

Формула 1.3. Расчёт количества феромона, обновляемого с течением времени

где ρ (ρ ho) – настраиваемый коэффициент испарения феромона в пределах от 0 до 1.

В результате настраиваемыми параметрами данного алгоритма являются четыре переменные: α , β , ρ и Q , которые определяются опытным путём.

1.3. Псевдокод реализации алгоритма

1. Инициализация настраиваемых параметров α , β , ρ и Q
2. Составление матрицы расстояний между всеми вершинами графа
3. Размещение на рёбрах графа начальной концентрации феромона
4. Размещение «муравьёв» на графе
5. Цикл по времени симуляции колонии (от 1 до t_{\max})
 - 5.1) Цикл перебора всех муравьёв (от 1 до n)
 - 5.1.1) Составление маршрута для «муравья», руководствуясь формулой 1.1.
 - 5.1.2) Если путь более короткий, чем найденные ранее, он запоминается как новый оптимальный путь.
 - 5.1.3) Вычисление количества феромонов, размещаемого «муравьём» на рёбра графа по формуле 1.2
 - 5.2) Обновление феромонов на рёбрах графа по формуле 1.3.
6. Завершение цикла
7. Вывод лучшего из найденных маршрутов

1.4. Модификации алгоритма

Классический алгоритм при первых реализациях давал многообещающие результаты, но они были не самыми лучшими среди уже существовавших. Однако из-за простоты алгоритма появилось множество возможностей для различных доработок и модификаций для улучшения быстродействия и результата [5, 6]. Самые ранние и наиболее известные модификации муравьиного алгоритма следующие:

А) Elitist Ant System (1992) [1, 2, 3]. В данной модификации, кроме обычных муравьёв, вводятся также «элитные муравьи», которые постоянно используют для передвижения только самый кратчайший из ранее найденных путей и оставляют большее количество феромонов. В большинстве случаев использование кратчайшего пути большим количеством муравьёв позволяет с большей вероятностью обнаружить более короткие пути. Как показал опыт использования алгоритма, до какой-то степени количество элитных муравьёв способствует более быстрому нахождению оптимального пути, однако также с увеличением их количества повышается вероятность нахождения «локального минимума» пути, который не является оптимальным. Следовательно, количество таких муравьёв также необходимо в настройке и определяется, как и другие параметры, опытным путём.

Б) Ant-Q (1995) [7]. Алгоритм хранит Q-таблицу, сопоставляющую каждому из ребер величину, определяющую «полезность» перехода по этому ребру. Эта таблица изменяется в процессе работы алгоритма – то есть обучения системы. Значение полезности перехода по ребру вычисляется исходя из значений полезностей перехода по следующим ребрам в результате предварительного определения возможных следующих состояний. После каждой итерации полезности обновляются исходя из длин путей, в состав которых были включены соответствующие ребра.

В) Ant Colony System (1996) [8]. Главные отличия от классического алгоритма следующие:

- Уровень феромонов на ребрах обновляется не только в конце очередной итерации, но и при каждом переходе муравьев из узла в узел.

- В конце итерации уровень феромонов повышается только на кратчайшем из найденных путей.

- Алгоритм использует измененное правило перехода: либо, с определенной долей вероятности, муравей безусловно выбирает лучшее – в соответствие с длиной и уровнем феромонов – ребро, либо производит выбор так же, как и в классическом алгоритме.

Г) Max-min Ant System (1996) [9, 10]. Особенность его работы следующая: на всех рёбра вводится ограничение на минимальное и максимальное количество феромонов. Изначально на рёбрах находится значение феромона, равное максимальному значению. Феромон откладывается только на самом лучшем из найденных маршрутов. Ограничение, введённое на количество феромонов, защищает алгоритм от преждевременной сходимости к субоптимальным решениям. При помощи этой модификации достигается, с одной стороны, более тщательное исследование области поиска, с другой – его ускорение.

Д) ASrank (Ранговая муравьиная система) (1997) [11]. В данной модификации каждому муравью в зависимости от длины пройденных путей присуждается определённый ранг. Таким образом муравьи, которые находят более оптимальный путь получают больший ранг и оставляют феромоновый след сильнее. Кроме того, для более тщательного исследования окрестностей уже найденных удачных решений, алгоритм используется вместе с модификацией элитных муравьёв.

А также, ANTS (1998) [12], Best-worst AS (2000) [13, 14], Population based ACO (2002) [15], Beam ACO (2004) [16], Hyper Cube ACO (2004) [17] и многие другие.

Кроме этого существует множество возможностей гибридного использования муравьиного алгоритма вместе с другими методами, что позволило объединять положительные стороны различных алгоритмов и добиваться ещё более эффективной работы алгоритма [18, 19].

Однако не каждая модификация может подходить для решения определённых задач. В различных ситуациях данные изменения могут как улучшить, так и ухудшить результат. Также некоторые модификации созданы для решения только конкретных задач. Например, ранее упомянутый ANTS, предназначен для решения только квадратичной задачи о назначениях (QAP), или модификация, предназначенная для решения задачи отбора функций (Feature selection) в которой вместо использования коэффициентов α и β , возможно применить экспоненциальную или логарифмическую функцию [20].

В последнее время, с развитием технологий, стала возможна параллельная обработка при помощи многоядерных процессоров или графических видеокарт, что дало ещё больше простора для развития муравьиных алгоритмов, на этот раз с параллельно протекающими процессами, которые имеют ещё большую эффективность работы [21,22,23,24].

До сих пор изучение муравьиных алгоритмов активно продолжается, и возникают всё новые, более эффективные модификации.

Значимость исследований в области муравьиных алгоритмов может быть продемонстрирована тем фактом, что статьи и работы на эту тему регулярно появляются в ходе IEEE Swarm Intelligence Symposium series и многих других научных конференций, а также Biannual International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS), посвящённые исключительно муравьиным алгоритмам, как и некоторые специальные журналы.

Кроме того, существует много неофициальных статей, форумов и сообществ, в которых проводится обсуждение использования муравьиных алгоритмов и возможности их применения [32,33].

1.5. Практическое использование муравьиного алгоритма

Муравьиные алгоритмы применимы для решения многих задач оптимизации [5]. Многочисленные компьютерные эксперименты свидетельствуют, что муравьиные алгоритмы поддерживают хороший баланс между точностью и скоростью работы. Ряд экспериментов показывает, что эффективность муравьиных алгоритмов возрастает с ростом размерности решаемых задач оптимизации. Лучше всего алгоритм показывает результаты при решении задач в нестационарных системах с изменением различных параметров. Как пример: расчёт телекоммуникационных и компьютерных сетей.

На данный момент муравьиные алгоритмы были использованы для решения огромного спектра задач, таких как задача коммивояжера, секвенирование ДНК [25], планирование [26], стыковка белок-лиганд [27], балансировка конвейера [28], последовательное упорядочивание [29], маршрутизация с коммутацией пакетов [30], сворачивание белка 2D-NP [31] и многих других.

2. Экспериментальное исследование муравьиного алгоритма

2.1. Использование алгоритма для нахождения кратчайшего пути на местности

Были проведены тесты для выполнения задачи, построенной по изначальному принципу муравьиной колонии – нахождение кратчайшего пути на местности. Для реализации использования муравьиного алгоритма местность была представлена при помощи графа, вершины которого распределены случайно. Использование алгоритма в задаче нахождения кратчайшего пути на взвешенном графе, вполне возможно. Однако при проведении собственных исследований (на созданной программе, которая является переделанной версией той, что будет использована ниже) и сравнением его работы с алгоритмом Дейкстры, были выявлены многие недостатки такого алгоритма для данной задачи:

1. Медлительность и неэффективность.

В данной задаче возникает слишком много факторов, сильно тормозящих нахождение оптимального пути или хотя бы приблизительного к нему. Один из них – то, что в алгоритм основан на случайном (точнее вероятностном) выборе, отчего при значительном количестве вершин и значительной разветвлённости графа многие попытки нахождения финишной точки попадают в тупик или преграждают себе путь, занеся все соседствующие вершины ранее в таблицу. По этой причине для нахождения точки цели проводится слишком много излишних вычислений и циклов поиска, которые не приносят результата.

Поэтому после сравнения алгоритма Дейкстры (Рисунок 2.1) и муравьиного алгоритма (Рисунок 2.2) в данной задаче, можно увидеть, что первый справился на несколько порядков быстрее и точнее.

Алгоритм Дейкстры:

Длина найденного маршрута 526,646

Время нахождения 0,00116 сек

Муравьиный алгоритм:

Длина найденного маршрута 598,805

Время нахождения 8,3669 сек

```
Deijkstra
Time:0.0011570453643798828
Best route:[0, 92, 56, 88, 35, 8, 82, 64, 93, 18, 1]
Length of best route:526.6466372463077
```

Рисунок 2.1. Отчёт программы о нахождении кратчайшего пути с помощью алгоритма Дейкстры

```
Start ACO
На шаг 0 ( 0.0048 сек) найден путь длиной 1196.6798641759674
На шаг 2 ( 0.1467 сек) найден путь длиной 1146.4233640209745
На шаг 3 ( 0.2292 сек) найден путь длиной 1106.6362833156195
На шаг 4 ( 0.317 сек) найден путь длиной 942.171457447633
На шаг 5 ( 0.4078 сек) найден путь длиной 706.1305520954267
На шаг 138 ( 3.2249 сек) найден путь длиной 621.0945365481555
На шаг 374 ( 8.3669 сек) найден путь длиной 598.805185452502
len 598.805185452502
step 374
time 8.3669
Alpha = 1
Beta = 1
```

Рисунок 2.2. Отчёт программы о нахождении кратчайшего пути с помощью классического муравьиного алгоритма

2. Практическая бесполезность составляющей вероятности β : предпочтение более коротких рёбер графа в сравнении с оптимальными.

Поскольку в данном алгоритме на каждой вершине высчитывается вероятность перехода по выходящим рёбрам на основании длины ребра и находящихся на ребре феромонов, при первом прохождении, когда количество феромонов ещё равно, «муравьи» с куда большей вероятности выберут кратчайшее ребро, хоть на самом деле оптимальное может быть другим (Рисунок 2.3. Зелёная стрелка – оптимальный путь. Красный – построенный алгоритмом).

Но со временем, поскольку «муравьи» продолжают выбирать кратчайшее ребро, на него будет откладываться большее количество феромонов, что ещё сильнее уменьшит вероятность выбора оптимального маршрута после ранее найденного.

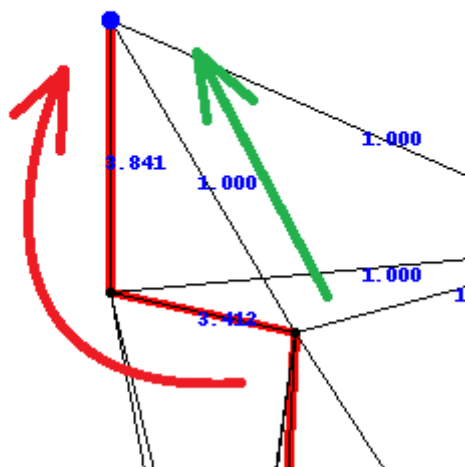


Рисунок 2.3. Элемент выведенной карты найденного муравьиным алгоритмом маршрута

Т.е. в данной задаче учёт длины рёбер совсем не нужен при выборе рёбер из вершины графа.

Следовательно, данную проблему возможно частично устранить, установив параметр $\beta=0$ и встроив в алгоритм элемент локального поиска. Однако в данном случае алгоритм становится практически случайным на

ранних стадиях и слишком закрепляется на более поздних, поскольку выбор ребра становится зависим только от количества феромонов. По этим причинам алгоритм в целом становится неэффективным.

Также есть вариант решения этой проблемы добавлением составляющей направления в расчёт вероятности перехода из вершины, однако это также добавит другие проблемы, которые бы потребовали дальнейшего решения.

3. Проблема настройки параметров алгоритма.

Некоторые оптимальные параметры (а точнее Q и Rho) сильно зависят от приблизительного расстояния и количества вариантов перехода. Однако поскольку изначально неизвестна сложность карты, тяжело установить те настройки, которые бы быстрее всего справились с обработкой и нахождением наилучшего пути. Правда, возможно встроить в алгоритм элемент, чтобы изменять данные параметры в процессе работы, однако на данную тему до сих пор проводятся исследования.

В результате, за учётом данных факторов, использование муравьиных алгоритмов для решения задачи нахождения кратчайшего пути между точками показало себя малоэффективным по скоростным и точностным характеристикам. Следовательно, для такой задачи желательно использовать другие, более подходящие для этого, алгоритмы.

2.2. Общее описание программы

Куда более полезным данный алгоритм смог проявить себя в других задачах (См. пункт 1.5). Одной из них является задача коммивояжёра или построения маршрута между множеством точек. Для проведения исследований алгоритма создадим программу на языке программирования Python (Приложение 1), в которой симулируется работа данного алгоритма для решения задачи построения оптимального маршрута среди точек, схожей с задачей коммивояжёра, с единственным отличием: точки «старта» и точки «цели» заданы изначально, отчего задача имеет алгоритмическую сложность $(n-2)!$ возможных маршрута вместо $(n-1)!$.

Главной целью исследований является изучение зависимости работы алгоритма от задаваемых параметров, поскольку, как ранее было сказано, скорость работы алгоритма сильно зависит от этих настроек, а также сравнение работы алгоритма с другими методами решения задачи.

В программу для сравнения скорости и точности встроены алгоритм перебора и генетический алгоритм. Также встроены три модификации муравьиного алгоритма: добавление в алгоритм «элитных муравьёв», ASrank и добавление в алгоритм гибридного элемента локального поиска.

Общая последовательность работы в программе следующая:

1. Ввод необходимых параметров (Alpha, Beta, Q, Rho, nAnt, nCity и др.)
2. Запуск генерации и вывода карты из случайно расположенных точек (Рисунок 2.4). Выделенные синим точки обозначают точки муравейника и точки цели (возможно после сменить на карте точки «муравейника» и «цели»)
3. Запуск муравьиного алгоритма поиска и/или другого алгоритма. Параллельно, для анализа опционально возможно выводить историю нахождения лучших маршрутов (Рисунок 2.5), а также наглядно фиксировать лучший найденный маршрут на изображении карты на

момент нахождения нового лучшего пути (Рисунок 2.6). Для последующего анализа результатов все данные заносятся в базу.

4. Анализ полученных данных. В программу для облегчения анализа добавлен блок, позволяющий сохранить полученные данные в таблице Excel.

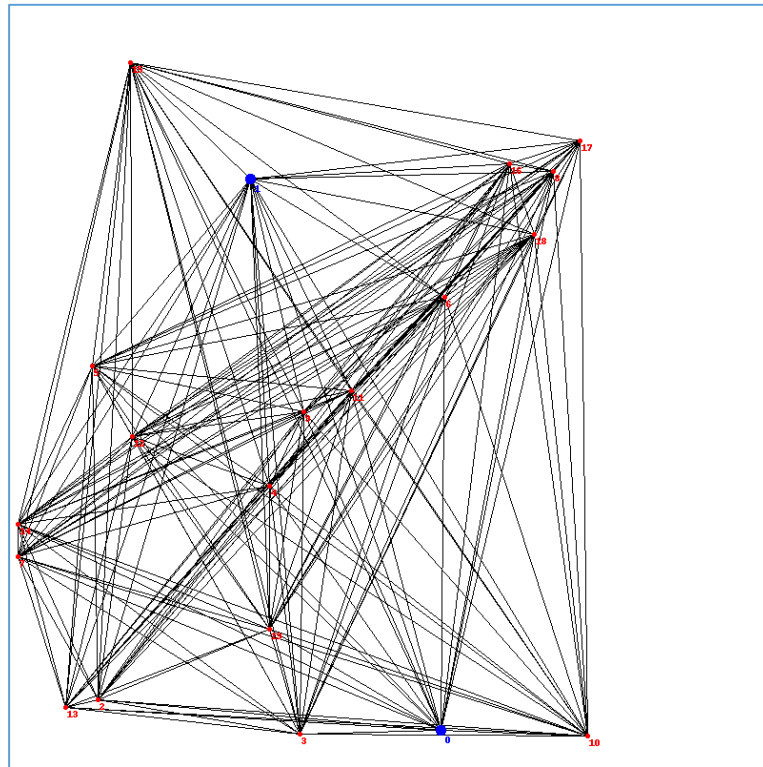


Рисунок 2.4. Вывод карты в программе для исследования муравьиного алгоритма

```

➤ На шаг 1 ( 0.0006 сек) найден путь длиной 3180.620641506154
На шаг 1 ( 0.04 сек) найден путь длиной 2908.1728015320805
На шаг 1 ( 0.0745 сек) найден путь длиной 2808.2837448032165
На шаг 1 ( 0.1098 сек) найден путь длиной 2103.9737587800378
На шаг 1 ( 0.1459 сек) найден путь длиной 1821.8224201110202
На шаг 1 ( 0.1812 сек) найден путь длиной 1767.3565019662974
На шаг 2 ( 0.2202 сек) найден путь длиной 1766.3917322531506
На шаг 2 ( 0.2538 сек) найден путь длиной 1753.1314137332613
На шаг 2 ( 0.2879 сек) найден путь длиной 1588.208746978046
На шаг 3 ( 0.3299 сек) найден путь длиной 1457.6368123728448
На шаг 5 ( 0.3837 сек) найден путь длиной 1373.536298210548
На шаг 9 ( 0.4486 сек) найден путь длиной 1360.634752961586
На шаг 12 ( 0.5108 сек) найден путь длиной 1351.397480021546
На шаг 19 ( 0.5947 сек) найден путь длиной 1351.0589458922846
На шаг 29 ( 0.7205 сек) найден путь длиной 1324.036404531017
На шаг 35 ( 0.8022 сек) найден путь длиной 1300.5649854696494
На шаг 52 ( 0.9916 сек) найден путь длиной 1296.0370389236639
На шаг 115 ( 1.556 сек) найден путь длиной 1296.0370389236637
len 1296.0370389236637
step 115
time 1.556
1 / 1

```

Рисунок 2.5. Вывод истории нахождения оптимального пути в программе для исследования муравьиного алгоритма

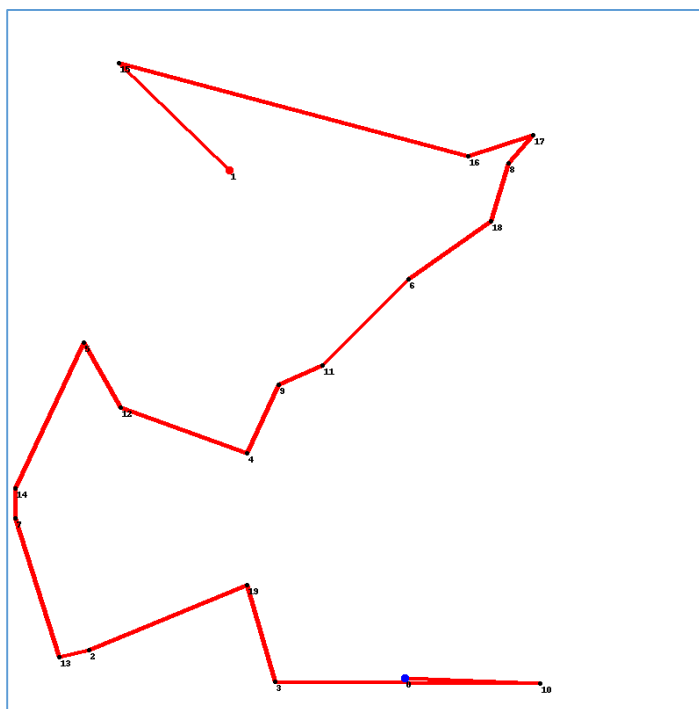


Рисунок 2.6. Промежуточная отрисовка последнего найденного маршрута в программе для исследования муравьиного алгоритма

2.3. Основная структура программы

Обработка карты:

Функция *do_point_map()* – функция генерации карты. Случайно распределяются точки по полю заданного размера и ближайшие из них соединяются и объявляются соседними.

Функция *FirstDraw()* – функция отображения изначальной карты.

Функция *ProgressDraw(time,bestLen,Trail,FoldName)* – функция фиксации и отображения карты с количеством феромонов и лучшим найденным путём.

Классический муравьиный алгоритм:

Функция *probability(i,j)* – функция нахождения вероятности перехода из одной точки в другую.

Функция *UpdPher()* – функция обновления количества феромонов.

Функция *Move(now,tabu)* – функция выбора точки перехода.

Функция *March(Fr,To)* – функция построения маршрута из изначальной точки в конечную.

Функция *Length(Trail)* – функция подсчёта длины маршрута.

Функция *PutPher(Trail,l)* – функция добавления муравьями феромона на пути.

Модификации муравьиного алгоритма:

Функция *RankModSort(FirstList)* – функция модификации ASrank для присвоения рангов муравьям.

Функция *RankModPher(TopList, LenList)* – функция модификации ASrank для распределения феромонов в соответствии с рангом.

Функция LocalFind(route) – функция гибридного элемента локального поиска

Дополнительные алгоритмы для сравнения с муравьиным алгоритмом:

Функция Gross(tabuNow,now) – функция для реализации алгоритма полного перебора

Функция Together(route1, route2) – функция генетического алгоритма для «скрещивания» двух маршрутов.

Функция Mutation1(route) – функция генетического алгоритма для внесения «мутации» в один из маршрутов поколения. Тип мутации 1.

Функция Mutation2(route) – функция генетического алгоритма для внесения «мутации» в один из маршрутов поколения. Тип мутации 2.

Функция Mutation3(route) – функция генетического алгоритма для внесения «мутации» в один из маршрутов поколения. Тип мутации 3.

Функция RandGen() – функция генетического алгоритма для создания случайного первого поколения.

Функция Find2Best(genlen) – функция генетического алгоритма для нахождения 2х кратчайших маршрутов из текущего поколения для дальнейшего скрещивания.

Функция FindBest(genlen) – функция генетического алгоритма для нахождения одного оптимального маршрута после скрещивания.

Функция Selection(r1,r2) – функция генетического алгоритма для проведения отбора из лучших особей.

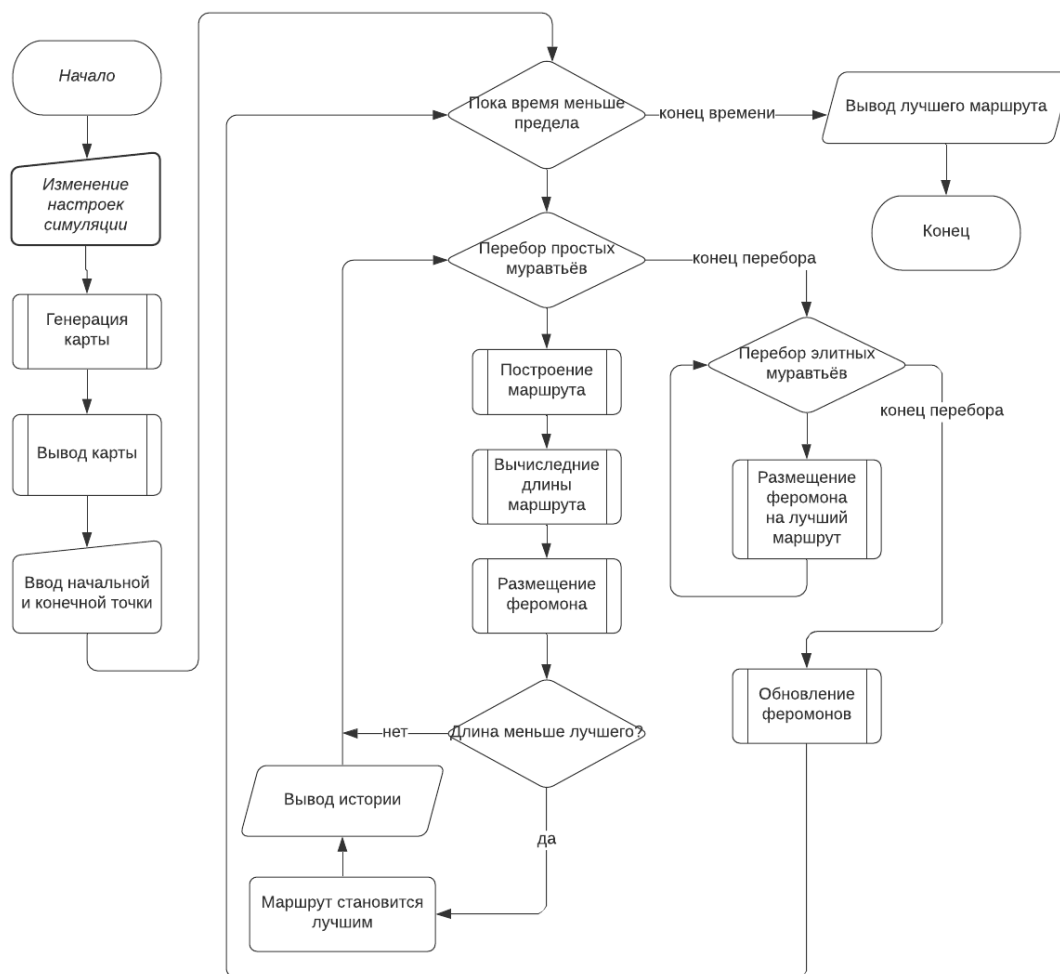


Рисунок 2.6. Блок-схема основного алгоритма программы

2.4. Проведение экспериментального исследования алгоритма

2.4.1 Условия проводимого исследования

Настраиваемые параметры алгоритма в программе:

1. nCity, количество точек на карте
2. nAnt, количество муравьёв в симуляции
3. Tmax, время окончания симуляции работы алгоритма
4. Alpha, коэффициент влияния феромона на вероятность выбора ребра.
5. Beta, коэффициент влияния длины ребра на вероятность его выбора.
6. Rho, коэффициент испарения феромонов на рёбрах графа.
7. Q, коэффициент увеличения феромонов.
8. window_x, ширина поля.
9. window_y, высота поля.
10. pherMin, минимальное количество феромона на рёбрах графа.
11. pherMax максимальное количество феромона на рёбрах графа.
12. nElite, количество «элитных» муравьёв.
13. ASrank, подключение модификации ранговой системы
14. LocalMin, подключение элемента локального поиска.

Опции для проведения исследования:

1. Par1Name, Par2Name, Par3Name, название 1,2 и 3 параметра соответственно, перебираемых в проводимых опытах.
2. Rng1max, Rng2max, Rng3max, количество различных значений переменных для перебора 1,2 и 3 параметра соответственно.
3. Par1Mas, Par2Mas, Par3Mas, перечни значений переменных для перебора 1,2 и 3 параметра соответственно.

4. MaxCikl, количество повторений опыта с каждой комбинацией параметров

Опции отображения вывода:

1. ScrCoef, масштабирование отображения карты.
2. ShowProcessMaps, фиксирование новых кратчайших найденных маршрутов в процессе работы алгоритма
3. ShowProgressString, вывод истории нахождения более коротких маршрутов в процессе работы алгоритма
4. AllLines, отображение всех линий с количеством феромонов на сохраняемых картах

В данной работе будут исследованы влияние изменения параметров $nCity$, $nAnt$, α , β , ρ , Q , $nElite$. Влияние будет оцениваться как по скорости нахождения оптимального решения, так и по его точности.

Стоит заметить, что поскольку метод нахождения пути не теоретический, а экспериментальный и основан на случайных вероятностях, от чего зависимости времени и расстояния могут не в полной мере отражать реальную зависимость результата от параметра и всегда остаётся возможность более быстрой или медленной сходимости, или получения результата хуже или лучше, чем вероятнее всего должен был получиться. Однако основные зависимости должны сохраниться, что позволяет наблюдать закономерности в результатах экспериментов. Для более точного результата проводится несколько тестов и вычисляется среднее значение.

2.4.2. Процесс нахождения оптимального пути

Для начала симулируем работу программы чтобы проследить, как изменяется найденный кратчайший маршрут с течением времени.

Параметры симуляции:

nCity=100 #Количество вершин

nAnt=30 #Количество простых муравьёв

Tend=20 #Время симуляции

Alpha=2 #Коэффициент альфа (порядка значимости феромона)

Beta=3 #Коэффициент бэта (порядка значимости длины пути)

Rho=0.3 #Коэффициент испарения феромона

Q=800 #Коэффициент увеличения феромона

```
На шаг 1 ( 0.0243 сек) найден путь длиной 4590.3560601178415
На шаг 1 ( 0.0895 сек) найден путь длиной 4184.425951681079
На шаг 1 ( 0.2748 сек) найден путь длиной 4114.484720484158
На шаг 2 ( 0.6872 сек) найден путь длиной 3966.782703822931
На шаг 2 ( 0.7739 сек) найден путь длиной 3589.2879802524126
На шаг 2 ( 0.8378 сек) найден путь длиной 3569.1012162340376
На шаг 2 ( 1.139 сек) найден путь длиной 3565.9356434966176
На шаг 4 ( 1.8048 сек) найден путь длиной 3486.432939328109
На шаг 6 ( 2.9265 сек) найден путь длиной 3458.4895569215264
На шаг 6 ( 3.1605 сек) найден путь длиной 3334.42397816288
На шаг 18 ( 8.6365 сек) найден путь длиной 3311.8515457096305
На шаг 37 ( 17.6034 сек) найден путь длиной 3234.614629477514
len 3234.614629477514
step 37
time 17.6034
1 / 1
```

Рисунок 2.7. Вывод истории нахождения кратчайшего пути в симуляции

Таблица 2.1. История нахождения кратчайшего пути в симуляции

Best	Step	Time
8843,7	1	0,0235
8322,6	1	0,0922
7896,5	1	0,1591
7725,4	2	0,6202
7541	2	0,7093
7194,1	2	0,7764
7158,3	2	0,8943
6818,4	2	0,9832
6232	2	1,1227

5501,8	3	1,5122
5321,4	5	2,3205
5158,4	5	2,6193
4954,7	7	3,3916
4930,9	8	3,9029
4923,8	8	4,0854
4781,9	8	4,2885
4765,1	9	4,6335
4633,1	12	5,9572
4396,9	12	6,2504
4353,6	13	6,5751
4255,7	15	7,5862
4239,2	20	9,9619

Построим график нахождения оптимального маршрута во времени

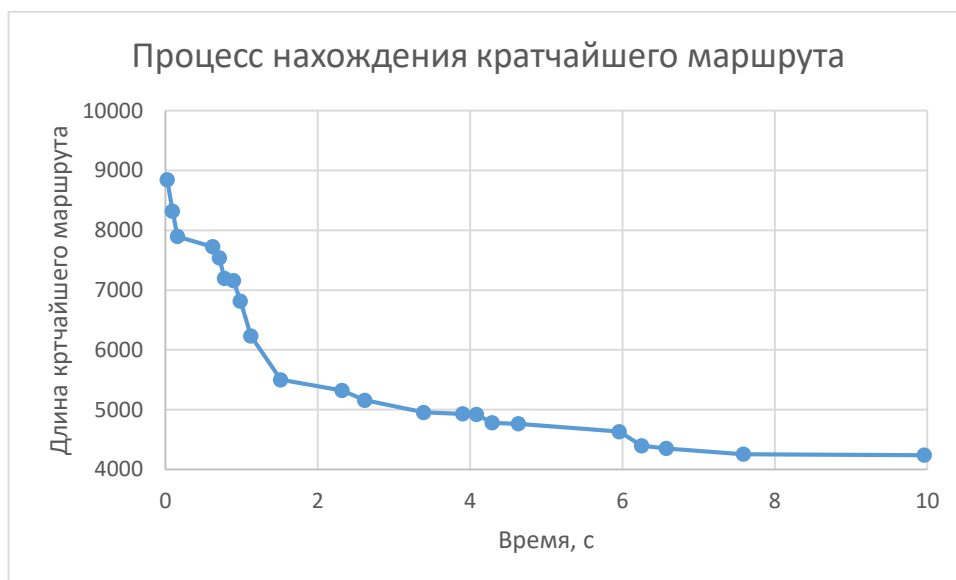


Рисунок 2.8. График изменения длины кратчайшего маршрута

По графику можно увидеть, как только алгоритм начинает свою работу, он быстро приближается к лучшему значению постепенно замедляясь, и вскоре уже разница между новыми найденными путями становится равной единицам.

2.4.3. Нахождение оптимальных значений α и β

Настраиваемые параметры α и β являются практически определяющими для этого алгоритма, поскольку именно от их соотношения в первую очередь зависит построение маршрута и работоспособность алгоритма в целом. Рассматривать эти коэффициенты по отдельности нельзя, поскольку в формуле вероятности выбора ребра графа они находятся в произведении, а, следовательно, только их совокупное взаимодействие определяет результат формулы.

В таблицах ниже жёлтым и зелёным цветом будут отмечены наилучшие значения найденного пути при определённых комбинациях параметров. Цвета ячеек скопированы на приложенные в пару таблицы времени нахождения этих кратчайших путей.

Параметры симуляции:

nCity=30 #Количество вершин;

nAnt=20 #Количество простых муравьёв;

Rho=0.25 #Коэффициент испарения феромона;

Q=800 #Коэффициент увеличения феромона;

Максимальное время эксперимента: 2 секунды.

Таблица 2.2. Кратчайшие найденные маршруты для различных значений α и β

	Beta=0	Beta=0.5	Beta=1	Beta=2	Beta=3	Beta=4	Beta=5
Alpha=0	4176,4	3601,9	2973,1	2202,7	1891,1	1732,5	1687,8
Alpha=0.5	4143,9	3318,3	2571,5	1962,7	1741,8	1661,3	1663,7
Alpha=1	4211,9	2748,8	2071	1751,5	1666,4	1653,3	1639,6
Alpha=2	4071,7	1886,1	1732	1678,3	1642,3	1644	1674,2
Alpha=3	4356,9	2041	1771,7	1712,1	1693,6	1705,8	1687,3
Alpha=4	4506,9	2488,9	1859,1	1738	1716,3	1701,6	1715,9

Таблица 2.3. Время нахождения кратчайших маршрутов для различных значений α и β

	Beta=0	Beta=0.5	Beta=1	Beta=2	Beta=3	Beta=4	Beta=5
Alpha=0	1,4142	1,2555	1,0558	1,04	1,1362	1,2832	1,5323
Alpha=0.5	1,0725	0,8246	1,2844	1,1841	1,406	0,9276	0,631
Alpha=1	0,8668	1,3202	1,0351	0,7497	0,8321	1,0001	1,0368

Alpha=2	0,5649	1,6091	1,2672	1,0785	0,6486	0,2347	0,6846
Alpha=3	0,5625	1,3017	1,0311	0,9181	0,1256	0,2137	1,1286
Alpha=4	0,1558	0,9194	0,125	0,4978	0,1673	0,1447	0,3464

Для уточнения результатов имеет провести схожий опыт с большим количеством вершин. Для уменьшения времени расчётов отбросим бесперспективные коэффициенты и рассмотрим $\alpha=1-3$ и $\beta=1-5$

nCity=50 #Количество вершин;

nAnt=20 #Количество простых муравьёв;

Rho=0.25 #Коэффициент испарения феромона;

Q=800 #Коэффициент увеличения феромона;

Максимальное время эксперимента: 2 секунды.

Таблица 2.4. Кратчайшие найденные маршруты для различных значений α и β

	Beta=1	Beta=2	Beta=3	Beta=4	Beta=5
Alpha=1	4339,6	2528,7	2322,2	2293,7	2242,3
Alpha=2	2530,2	2330,3	2221	2314,1	2264,9
Alpha=3	2430,6	2343	2272	2292,3	2225,7

Таблица 2.5. Время нахождения кратчайших маршрутов для различных значений α и β

	Beta=1	Beta=2	Beta=3	Beta=4	Beta=5
Alpha=1	2,5528	2,6859	1,8415	1,4856	2,0273
Alpha=2	3,1529	2,3476	2,152	1,0175	0,6264
Alpha=3	1,7946	2,4217	1,3833	2,0157	0,9263

В результате самые оптимальные параметры для муравьиного алгоритма в данной системе $\alpha=2$ (+-1) и $\beta=4$ (+-1). При этих параметрах наилучшая сходимость к минимальной длине пути

Также стоит рассмотреть следующие случаи:

$\alpha=0$, $\beta=0$:

В данном случае составляющие формулы вероятности прохождения по определённому ребру практически всегда будут равны 1. Таким образом муравьиный алгоритм становится просто построением случайных маршрутов.

$\alpha>0$, $\beta=0$:

Поскольку составляющая вероятности, отвечающая за длину ребра, в любом случае будет равна 1, то будут учитываться только то, какое количество феромонов было ранее распределено. Но поскольку изначально, когда феромоны ещё не распределены, выбор рёбер случаен, маршрут закрепляется в уже ранее найденном положении и не приходит к оптимальному. И, как можно увидеть, чем больше значение α , тем сильнее алгоритм «застревает» на ранее проложенном неоптимальном маршруте. Исходя из этого составляющая β необходима.

$$\alpha=0, \beta>0:$$

В данном случае выбор ребра будет основываться исключительно на их длине. В результате, хоть и выбирается кратчайшие рёбра, глобально оптимальный маршрут найти будет сложнее и, соответственно, дольше. По этой причине и необходима составляющая α .

2.4.4. Нахождение оптимальных значений Q , ρ и количества муравьёв.

Для нахождения оптимальных значений, проведём эксперименты при различных значениях Q , ρ и n_{Ant} , подставив уже ранее найденные значения α и β .

Есть основания предполагать, что при большом количестве муравьёв феромоновая дорожка будет слишком быстро закрепляться, отчего не будет возможности найти более оптимальные пути, а значит в таком случае стоит использовать значение коэффициента Q меньше. При небольшом количестве же муравьёв феромоны на рёбрах феромоны не будут успевать обновляться с должной скоростью, а, следовательно, нужен будет больший коэффициент Q .

Масштаб карты 400*400

Alpha=2

Beta=4

Максимальное время эксперимента: 1 секунда

nCity=50

Rho=0.1

Таблица 2.6. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.1$

Rho=0.1	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	2885,7	2826,4	2811,7	2774,1	2728,4	2699,1	2771,9
Q=500	2720,5	2731,8	2722	2647,9	2757,1	2739,1	2658,1
Q=1000	2716,6	2708,6	2613	2653,7	2695,5	2683,1	2698,3
Q=2000	2737,7	2648,9	2703,6	2643,5	2666,1	2670,9	2726,9
Q=4000	2749,3	2698,6	2646,9	2667,6	2699,5	2703,1	2708,4
Q=8000	2681,5	2659,1	2650,3	2655,7	2773,3	2695,6	2720,6
Q=16000	2624,9	2682,3	2682,4	2708,8	2768,6	2823,4	2853,8

Таблица 2.7. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.1$

Rho=0.1	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	0,3798	0,5814	0,7673	0,7698	0,7825	0,8955	1,2585
Q=500	0,3517	0,4641	0,5665	0,5721	0,8456	0,6662	1,3214
Q=1000	0,353	0,6176	0,3998	0,6802	0,7018	0,8972	1,324
Q=2000	0,327	0,7821	0,7277	0,3649	0,5999	1,0554	1,3243
Q=4000	0,5106	0,3967	0,6145	0,6713	0,8947	0,8142	1,4054
Q=8000	0,8628	0,4344	0,6247	0,8707	0,7993	0,7915	1,266
Q=16000	0,5649	0,5952	0,5868	0,2179	0,366	0,6262	1,2065

Rho=0.25

Таблица 2.8. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.25$

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	2968,4	2985,1	2944,4	2876,9	2749,6	2724,2	2715,1
Q=500	2856,9	2772,2	2724,8	2678,8	2694,6	2683,1	2673,7
Q=1000	2772,6	2732,3	2744,5	2689,4	2705,7	2669,7	2665,9
Q=2000	2752,5	2731,1	2793,4	2703,2	2716	2647,4	2680,1
Q=4000	2788,6	2687,2	2717	2624,6	2652,3	2697,7	2720,4
Q=8000	2670,1	2707,9	2729,1	2613,6	2668,3	2714,1	2796,1
Q=16000	2743,4	2700	2708,3	2666,9	2727,3	2844,1	2871,7

Таблица 2.9. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.25$

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	0,4391	0,5721	0,8401	0,5984	0,7853	1,0324	1,2896
Q=500	0,5235	0,7716	0,5597	0,473	0,8465	0,9208	1,1334
Q=1000	0,4233	0,6014	0,4889	0,6487	0,6464	0,9637	1,2124
Q=2000	0,557	0,4208	0,3319	0,3496	0,4882	0,8886	1,3013
Q=4000	0,1576	0,3991	0,4835	0,6124	0,6854	0,7348	0,9718
Q=8000	0,342	0,5483	0,8193	0,7601	0,4222	0,7208	1,1512
Q=16000	0,5304	0,4372	0,3181	0,6589	0,5855	0,5297	1,125

Rho=0.5

Таблица 2.10. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при Rho=0.5

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	3009,6	3043	2867,3	2891,6	2755,3	2691,9	2726,4
Q=500	2957,2	2923,1	2808,8	2721,6	2688,7	2671	2739,4
Q=1000	2884,3	2787,2	2682	2743	2709,7	2670,7	2670,5
Q=2000	2765,7	2765,6	2685,9	2732,5	2673,9	2653,7	2653,1
Q=4000	2742,6	2692,5	2660,6	2695,8	2664,3	2653,1	2795,7
Q=8000	2849,1	2743,3	2787,2	2577	2718,9	2730,5	2839,4
Q=16000	2646,1	2664,3	2747,5	2628,7	2714,7	2799,2	2875,3

Таблица 2.11. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при Rho=0.5

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	0,5192	0,4671	0,4583	0,8817	0,7691	0,977	1,2397
Q=500	0,3917	0,4849	0,5187	0,6831	0,6973	0,8623	1,3793
Q=1000	0,29	0,4241	0,6648	0,7917	0,7967	0,9475	1,3045
Q=2000	0,5741	0,4462	0,7357	0,597	0,7598	0,7829	1,4334
Q=4000	0,3143	0,2592	0,3686	0,3854	0,7051	0,9887	0,9903
Q=8000	0,1477	0,2302	0,3578	0,5473	0,4373	1,0086	1,1444
Q=16000	0,2139	0,497	0,4776	0,5057	0,6699	0,989	1,0771

Rho=0.75

Таблица 2.12. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при Rho=0.75

Rho=0.75	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	2965,7	3004	3012,1	2810,2	2841,6	2725,7	2753,5
Q=500	3040,1	3059,9	2831,2	2733,5	2750,6	2652,4	2713,1
Q=1000	2969,3	2832,8	2708,2	2762,1	2635,7	2675,4	2745,7
Q=2000	2752,1	2725	2711,5	2752,6	2662,2	2595,7	2737,4
Q=4000	2710,6	2653,6	2753,6	2660,1	2679,4	2649,1	2721,2
Q=8000	2740,4	2712,4	2705,2	2660,6	2678,7	2736,9	2849,3

Q=16000	2783,5	2761,9	2661,8	2669,5	2655,8	2742,7	2899,3
---------	--------	--------	--------	--------	--------	--------	--------

Таблица 2.13. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.75$

Rho=0.75	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	0,321	0,2533	0,4298	0,6915	0,7859	0,9099	1,2583
Q=500	0,5404	0,3192	0,7324	0,6653	0,8411	1,123	1,4123
Q=1000	0,7438	0,5763	0,5089	0,5618	0,5718	0,9933	1,3644
Q=2000	0,7705	0,4917	0,5448	0,5401	0,6112	0,8482	1,0298
Q=4000	0,4241	0,406	0,2022	0,7132	0,6039	0,8058	1,1651
Q=8000	0,5126	0,539	0,2003	0,6659	0,8142	0,9945	1,34
Q=16000	0,0614	0,1786	0,3588	0,5437	0,5541	0,7866	1,2912

Rho=0.9

Таблица 2.14. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.9$

Rho=0.9	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	2969,2	3035,2	2966,7	2989,3	2858,7	2802,1	2779,3
Q=500	2999,4	2971,1	2890,1	2784,5	2716,3	2676	2686,6
Q=1000	2901,6	2907,8	2792,1	2697	2747,7	2663,5	2695,2
Q=2000	2820	2703,7	2788	2642,6	2712	2683,1	2695,9
Q=4000	2696,1	2727,3	2678,2	2702,4	2717,7	2709	2714,9
Q=8000	2773,6	2725,7	2756,7	2619,5	2674,5	2694,8	2727,4
Q=16000	2714,8	2766	2734,6	2672	2724,4	2772,8	2833,4

Таблица 2.15. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.9$

Rho=0.9	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=200	0,4748	0,2085	0,1896	0,6389	0,5137	0,8502	0,9828
Q=500	0,5142	0,3184	0,5188	0,4138	0,8603	0,8828	1,2621
Q=1000	0,2103	0,8084	0,7783	0,6806	0,8887	0,8701	1,4936
Q=2000	0,6921	0,504	0,2457	0,4683	0,7936	1,0254	1,4712
Q=4000	0,4358	0,3094	0,6993	0,484	0,479	0,6914	1,2783
Q=8000	0,5782	0,1408	0,4561	0,4898	0,6067	0,632	1,3558
Q=16000	0,5303	0,1602	0,4634	0,7108	0,7808	0,8841	1,1399

Как можно увидеть, в таблице оптимальные значения занимают полосу «по диагонали», что подтверждает ранее выдвинутое предположение. При том чем больше коэффициент испарения Rho , тем сильнее сдвигается эта полоса. Оно и неудивительно, поскольку при большей скорости испарения необходимо

больше количество феромонов, оставляемых муравьёв или большее количество муравьёв, которые бы оставили феромон. Однако при большом коэффициенте испарения также алгоритм ведёт себя менее предсказуемо, поскольку слишком быстро исчезают оставляемые на рёбрах следы, переставая влиять на выбор через 1-2 итерации.

Также можно заметить, что при меньшем количестве задействованных муравьёв выше и скорость нахождения решения, близкого к оптимальному. Чем меньше муравьёв, тем больше итераций будет выполнено за положенное время. И, следовательно, тем больше ранее отложенных «поколений» феромонов будут влиять на последующие выборы рёбер графа.

Примем оптимальным $Rho=0,25-0,5$. Таким образом феромон будет не так быстро накапливаться на неоптимальных рёбрах, но и достаточно будет оставаться следов, чтобы воздействовать на последующие итерации.

Стоит сказать, что при коэффициенте ρ равном нулю с каждым ходом феромон будет только накапливаться на оптимальном пути, что при изменении местности не даст найти новый оптимальный маршрут, заставляя муравьёв передвигаться по оставшейся части от старого пути и искать обходной, но скорее всего менее выгодный маршрут.

При коэффициенте ρ равном 1 феромон наоборот будет с каждой единицей времени полностью испаряться, что не даёт феромоновой дорожке оптимального пути образоваться.

Для определения зависимости значений оптимальных коэффициентов от количества вершин графа проведём схожие эксперименты для $nCity=25$ и $nCity=75$.

$nCity=25$

$Rho=0.25$

Таблица 2.16. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.25$. $nCity=25$

$Rho=0.25$	$nAnts=5$	$nAnts=10$	$nAnts=15$	$nAnts=25$	$nAnts=50$	$nAnts=100$
$Q=200$	1742,9	1688,2	1670,9	1677,1	1682	1676,8

Q=500	1669	1690	1681,1	1689,2	1689,4	1684,7
Q=1000	1699,7	1702,4	1701,8	1685,7	1693,7	1686,9
Q=2000	1711,4	1691,6	1708,4	1680,7	1699,9	1704,8
Q=4000	1727,2	1703,4	1701	1700	1701,4	1708,3
Q=8000	1716,8	1696,2	1695,9	1690,8	1686,9	1714,4
Q=12000	1706,7	1691,1	1696,5	1715,4	1706,7	1720,5
Q=16000	1686	1686,1	1709,2	1702,3	1733,2	1715,4

Таблица 2.17. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.25$. $nCity=25$

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100
Q=200	0,1107	0,3094	0,3079	0,1648	0,1674	0,2176
Q=500	0,2002	0,1417	0,0959	0,1042	0,1665	0,2311
Q=1000	0,1713	0,106	0,0586	0,0718	0,1444	0,231
Q=2000	0,1602	0,1905	0,1062	0,0869	0,2345	0,214
Q=4000	0,1475	0,1412	0,2051	0,2157	0,277	0,235
Q=8000	0,2612	0,1727	0,1657	0,1747	0,1425	0,1464
Q=12000	0,2248	0,2195	0,195	0,3052	0,2078	0,2621
Q=16000	0,2252	0,2362	0,1385	0,1328	0,3297	0,3265

Rho=0.5

Таблица 2.18. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.5$. $nCity=25$

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100
Q=200	1772,5	1754,5	1710,2	1673	1669,7	1667,5
Q=500	1743,3	1671,5	1671,1	1683,7	1687,5	1684,1
Q=1000	1675,4	1698,1	1671,6	1695,6	1687,1	1683,4
Q=2000	1693,9	1707,6	1702,9	1699,6	1699,1	1701
Q=4000	1681,1	1699,7	1711,3	1695	1700,7	1702,3
Q=8000	1734,3	1701,8	1700,7	1697,1	1700,3	1715,4
Q=12000	1728,3	1700,7	1697	1695,7	1702,4	1722,6
Q=16000	1720,4	1699,5	1696,4	1696,2	1697	1697,9

Таблица 2.19. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.5$. $nCity=25$

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100
Q=200	0,1464	0,1302	0,2054	0,24	0,217	0,2234
Q=500	0,2997	0,2169	0,1781	0,1184	0,1365	0,2362
Q=1000	0,1722	0,2051	0,0792	0,1388	0,0878	0,1974
Q=2000	0,2681	0,1793	0,0893	0,0693	0,1622	0,425
Q=4000	0,2439	0,1773	0,0558	0,1634	0,1908	0,3143
Q=8000	0,1636	0,1472	0,1522	0,2692	0,3191	0,3615
Q=12000	0,1504	0,1976	0,1833	0,1216	0,2058	0,1396

Q=16000	0,1971	0,1859	0,1669	0,27	0,2009	0,2934
---------	--------	--------	--------	------	--------	--------

nCity=75

Rho=0.25

Таблица 2.20. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при Rho=0.25. nCity=75

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	4176,8	3751,3	3344,5	3195,8	3150,4	3158,1	3262,1
Q=1000	3573,4	3126,1	3157,2	3075,4	3067,6	3141,2	3118,1
Q=2000	2998,2	3051,3	2956,8	3016,7	3071,7	3126,1	3054,6
Q=4000	3048,8	3022,8	2995,8	2982,3	2985,1	3072,1	3112,2
Q=8000	3079,3	3026	3029,3	3053,5	3131,5	3176,5	3301,3
Q=12000	3070,5	3048,7	3115,4	3149,7	3176,3	3207,4	3395,7
Q=16000	3006,2	3055,9	3127,6	3162,1	3186,6	3360,5	3322,4
Q=24000	3098,7	3151,5	3183,9	3208,6	3335	3473,6	3488,4

Таблица 2.21. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при Rho=0.25. nCity=75

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	0,396	1,5326	1,4186	2,5599	1,9487	2,943	2,6771
Q=1000	2,0907	1,7795	1,86	2,4239	2,1616	2,74	3,6048
Q=2000	2,1346	1,9834	1,5634	1,873	1,989	2,277	3,3493
Q=4000	1,7031	1,3739	1,8015	2,0242	1,615	1,4108	3,2391
Q=8000	1,2612	1,6193	1,9819	1,6426	1,3365	1,447	2,8415
Q=12000	1,9958	1,7814	1,3708	1,2015	1,2453	1,4459	3,092
Q=16000	1,7597	1,5931	0,865	1,0749	0,9075	1,3364	2,8969
Q=24000	2,0288	1,3784	0,73	1,0001	1,6229	2,0945	3,1238

nCity=75

Rho=0.5

Таблица 2.22. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при Rho=0.5. nCity=75

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	4222,1	4109,3	3939,7	3596,1	3436,9	3167,4	3175,2
Q=1000	4096,9	3606	3406,5	3166,6	3141,8	3025,8	3111,3
Q=2000	3478,5	3090,7	3014,9	3041	3063,1	3093,2	3053,7
Q=4000	3024,7	3042,9	2993,3	2990,4	3041	3082,1	3060,4
Q=8000	3133,7	3040	3047	3023,4	3055,4	3148,4	3264,5
Q=12000	3089,2	2965,7	2999,5	3059,4	3119,7	3259,8	3365,7

Q=16000	3182,4	3052,6	3041,7	3082,7	3115,9	3243,5	3408,2
Q=24000	2930,9	3024,9	3096,6	3184,4	3232,3	3433,7	3587,9

Таблица 2.23. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.5$. $nCity=75$

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	1,3009	1,8545	1,1628	2,1211	2,3111	2,628	3,7759
Q=1000	1,112	1,2798	2,3947	2,2337	1,8514	2,0991	3,7152
Q=2000	1,3386	1,8355	2,0133	1,9963	2,059	2,2406	3,451
Q=4000	1,5156	1,2845	1,9357	1,7702	1,7869	2,3332	3,5942
Q=8000	1,2451	1,7867	1,6965	1,4456	1,5094	1,6548	2,9463
Q=12000	1,1626	1,2156	1,3961	2,0134	1,4381	1,6714	3,1264
Q=16000	1,3644	1,9348	1,9441	2,213	1,4626	2,0165	3,2031
Q=24000	1,5059	2,1077	1,6945	2,1231	1,4079	1,8015	3,406

Для сравнения вынесем средние ячейки Q в полосе оптимальных значений при одинаковом количестве муравьёв.

Таблица 2.24. Оптимальные значения Q для различного количества вершин и коэффициента испарения Rho

nAnts=25	Q	
nCity	Rho=0.25	Rho=0.5
25	1000	2000
50	2000	4000
75	4000	8000

Как можно заметить, при увеличении количества городов увеличивается также и оптимальное значение Q . Также, что неудивительно, при увеличении количества городов также увеличивается время нахождения оптимального маршрута.

Следовательно, не рекомендуется для двух случаев с различным, значительно отличающемся количеством городов, использовать одинаковые настройки. Поскольку в полученных значениях прослеживается ясная закономерность, предлагается приблизительно устанавливать настройки на основании уже ранее пройденных экспериментов.

Проведём схожие эксперименты, но теперь вместо количества городов будем изменять масштаб карты (в предыдущих экспериментах масштаб был 400*400).

Масштаб карты 200*200

Alpha=2

Beta=4

Максимальное время эксперимента: 1 секунда

nCity=50

Rho=0.25

Таблица 2.25. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.25$. Масштаб карты 200*200

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=125	1252,4	1183	1184	1133,6	1104,6	1124,4	1099,7
Q=250	1195,6	1125,8	1099	1075,1	1100,5	1096	1094,9
Q=500	1107	1101,4	1108,5	1083	1082,6	1106,9	1086,5
Q=1000	1109,3	1112	1080	1122,8	1092,1	1095,6	1110,8
Q=2000	1173,6	1077,3	1083	1098,2	1113,9	1116,2	1136,9
Q=4000	1085,3	1097,1	1093,3	1094,2	1105,3	1142,2	1142,7
Q=8000	1090,2	1101,4	1145,6	1140,9	1148,1	1166,1	1225,8
Q=12000	1084,9	1140	1169,3	1150,6	1139,3	1204,9	1208,7

Таблица 2.26. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.25$. Масштаб карты 200*200

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=125	0,2243	0,6215	0,5195	0,895	0,9732	0,8128	1,1895
Q=250	0,6858	0,6021	0,7981	0,9683	0,7619	0,8595	1,2232
Q=500	0,5947	0,5694	0,4632	0,6737	0,9742	0,8494	1,2474
Q=1000	0,3631	0,4106	0,719	0,6428	0,7958	0,9086	1,0719
Q=2000	0,2746	0,7537	0,8077	0,5001	0,6153	0,704	1,2061
Q=4000	0,3398	0,6175	0,6793	0,3411	0,7057	0,7091	1,2
Q=8000	0,4294	0,6839	0,2417	0,5545	0,5497	0,5928	1,0725
Q=12000	0,6763	0,479	0,4448	0,3978	0,7557	0,8341	0,7361

Rho=0.5

Таблица 2.27. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.5$. Масштаб карты 200*200

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
---------	---------	----------	----------	----------	----------	-----------	-----------

Q=125	1267,5	1233,5	1196,6	1117,1	1091	1104,3	1105,2
Q=250	1237,3	1161,3	1131,6	1079,8	1082,8	1075,6	1077,2
Q=500	1180,7	1092,3	1092,8	1083,1	1059,7	1066,4	1070
Q=1000	1077,9	1090,8	1070,4	1078,9	1069,5	1092,3	1102,6
Q=2000	1116,5	1135,6	1083,9	1081,9	1081,5	1116,9	1121,5
Q=4000	1128,4	1101,5	1102,7	1096,2	1122,8	1154,3	1159,7
Q=8000	1142,3	1089,3	1104,6	1133,6	1151,2	1152,7	1181,5
Q=12000	1131,4	1106,2	1102,5	1155	1173,1	1187,2	1234,4

Таблица 2.28. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.5$. Масштаб карты 200*200

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=125	0,378	0,4666	0,8034	0,5218	0,8631	0,9965	1,1626
Q=250	0,2902	0,4524	0,4456	0,8986	0,8714	0,9614	1,2706
Q=500	0,6381	0,6232	0,5119	0,8125	0,9661	0,9917	1,1579
Q=1000	0,6924	0,686	0,477	0,7141	0,8084	0,6958	1,1045
Q=2000	0,692	0,4112	0,3956	0,6432	0,6321	0,6318	1,125
Q=4000	0,5878	0,7352	0,7084	0,7856	0,3928	0,8047	1,0097
Q=8000	0,5294	0,6556	0,85	0,5433	0,6327	0,7119	1,032
Q=12000	0,7039	0,4325	0,2132	0,6008	0,871	0,9001	0,852

Масштаб карты 800*800

Alpha=2

Beta=4

Максимальное время эксперимента: 1,5 секунды

nCity=50

Rho=0.25

Таблица 2.29. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.25$. Масштаб карты 800*800

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	5297,5	4954,5	4836	4696,7	4705,1	4646,5	4623,6
Q=1000	5074,8	4725,8	4664,8	4704,6	4693,3	4713	4568,2
Q=2000	4891,6	4697,8	4661,3	4635,5	4575,4	4672,4	4629,6
Q=4000	4711,8	4612,1	4706,8	4687,8	4591,1	4656,1	4627,1
Q=8000	4677,7	4848,3	4690,8	4659	4610,7	4617	4715,6
Q=12000	4742,1	4683,1	4609,8	4662,4	4645,1	4794,1	4861,2
Q=16000	4816,5	4684,9	4575,9	4679,1	4690,8	4775,6	4916,5
Q=24000	4769	4677,1	4670,6	4602,9	4697,8	4759,5	4988,9

Таблица 2.30. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.25$. Масштаб карты 800*800

Rho=0.25	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	1,0868	0,8387	1,3458	1,1117	0,9989	0,95	1,1807
Q=1000	0,404	1,2786	0,9286	1,1202	0,8134	1,0672	1,1928
Q=2000	0,8857	1,0491	0,6837	0,9051	0,9718	0,8975	1,1244
Q=4000	0,4904	0,4495	0,5979	0,6084	0,6942	1,1693	1,1378
Q=8000	0,5817	0,201	0,7781	0,6359	0,8065	0,708	1,1766
Q=12000	0,0955	0,6357	0,5706	0,662	0,6792	1,051	1,1775
Q=16000	0,468	1,2916	1,0306	0,6656	0,8525	0,9328	1,0448
Q=24000	0,9731	0,6157	0,1994	0,4865	0,7192	1,2259	1,0969

Rho=0.5

Таблица 2.31. Кратчайшие найденные маршруты для различных значений количества муравьёв и Q при $Rho=0.5$. Масштаб карты 800*800

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	5193	5284,6	5265,4	4901,2	4688,4	4792,5	4627,9
Q=1000	5104	4950,3	4762,9	4693,9	4645,6	4596,9	4607,2
Q=2000	5036,7	4702,9	4754,7	4707,2	4654,9	4592,3	4603,4
Q=4000	4644,9	4704,9	4740,5	4568,5	4624,5	4601	4681,4
Q=8000	4729,3	4811,8	4762,4	4645,9	4690,9	4627,7	4798,6
Q=12000	4864,7	4752	4732,2	4608,8	4612,5	4720,9	4858,4
Q=16000	4857,6	4602,5	4640	4654,6	4682,6	4816,4	4925,1
Q=24000	4768,5	4611,8	4692,1	4611,7	4713,6	4909,4	5037,4

Таблица 2.32. Время нахождения кратчайших маршрутов для различных значений количества муравьёв и Q при $Rho=0.5$. Масштаб карты 800*800

Rho=0.5	nAnts=5	nAnts=10	nAnts=15	nAnts=25	nAnts=50	nAnts=100	nAnts=200
Q=500	0,4884	1,1653	0,7154	1,1262	0,9338	1,2413	1,253
Q=1000	0,6175	0,6815	0,8536	1,0292	0,9991	1,2792	0,8813
Q=2000	0,3657	0,8095	0,9229	0,7002	0,9836	1,1194	1,1795
Q=4000	0,9298	1,0796	0,3712	0,6446	0,4932	0,8898	1,146
Q=8000	0,5627	0,3767	0,5469	0,7306	0,6259	0,7335	1,1327
Q=12000	0,9979	0,5363	0,8511	0,5294	0,8522	0,9272	1,3416
Q=16000	0,6392	0,5029	0,9535	0,8783	1,0563	0,9382	0,9992
Q=24000	0,3842	0,7181	0,8532	0,8478	0,9973	0,9398	0,8258

Таблица 2.33. Оптимальные значения Q для различного масштаба карты и коэффициента испарения феромона Rho

nAnts=25	Q
----------	---

Масштаб	Rho=0.25	Rho=0.5
200*200	500	1000
400*400	2000	4000
800*800	8000	8000

Как можно заметить, оптимальные значения Q при различном масштабе карты изменяются практически пропорционально площади карты. При рассмотрении формулы распределения феромонов можно заметить, что поскольку масштаб изменяется в x раз, а, следовательно, и средняя длина рёбер в графе также будет изменена в x раз, то одинаковое количество феромонов будет отложено на ребро только при условии, если Q также будет изменена в x раз. Таким образом при изменении масштаба легко можно вычислить коэффициент Q при дальнейшем увеличении масштаба на местности.

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

Формула 2.1. Добавление феромона одним муравьём в классическом алгоритме

Для того, чтобы избавиться от подобной зависимости от масштаба местности предлагаю добавить в формулу выше множитель, равный площади местности.

$$\Delta\tau(t) = \begin{cases} \frac{Q \cdot S}{L_k(t)}, (i, j) \in T_k(t) \\ 0, (i, j) \notin T_k(t) \end{cases}$$

Формула 2.2. Предложенная формула добавления феромона одним муравьём

Таким образом, вполне возможно, изменение средних длин рёбер будет компенсировано, и для равного количества точек оптимальное значение Q будет оставаться близко к оптимальному при различном масштабе карты.

2.5. Использование модификаций алгоритма

Как ранее было сказано, было разработано множество модификаций данного алгоритма, которые могли бы ускорить алгоритм, увеличить его точности и решить некоторые из его проблем.

В программе реализованы три модификации алгоритма: «элитные муравьи», ASrank (ранговая система муравьёв) и гибрид с нахождением локального минимума.

2.5.1. Модификация элитных муравьёв.

При данной модификации кроме обычных муравьёв вводятся также и особые, элитные, которые используют только тот, что является кратчайшим из всех ранее найденных. На самом деле для симуляции прохода муравьями пути необходимо просто добавить на наилучший маршрут дополнительных феромонов.

Теоретически эта модификация должна поспособствовать сходимости, поскольку ранее оптимальный маршрут быстрее закрепляется, отчего муравьи с меньшей вероятностью будут проходить по менее оптимальным и быстрее система найдёт другие, более короткие, пути.

То, как быстро дополнительно закрепляется наилучший путь в данной модификации зависит от количества элитных муравьёв (n_{Elite}). Проведём эксперименты по нахождению оптимального числа элитных муравьёв с ранее найденными оптимальными значениями Alpha, Beta, Q, Rho.

$n_{Ant}=25$ #Количество простых муравьёв

$Alpha=2$ #Коэффициент альфа (порядка значимости феромона)

$Beta=4$ #Коэффициент бэта (порядка значимости длины пути)

$Rho=0.25$ #Коэффициент испарения феромона

$Q=8000$ #Коэффициент увеличения феромона

$n_{City}=50$ #Количество вершин

Максимальное время эксперимента: 1 секунда.

Таблица 2.34. Кратчайшие найденные маршруты для различного количества элитных муравьёв

nElite=0	nElite=1	nElite=2	nElite=3	nElite=4	nElite=5	nElite=7	nElite=10
2478,6	2474,2	2456,2	2439,2	2447,3	2469,6	2473,8	2485,9

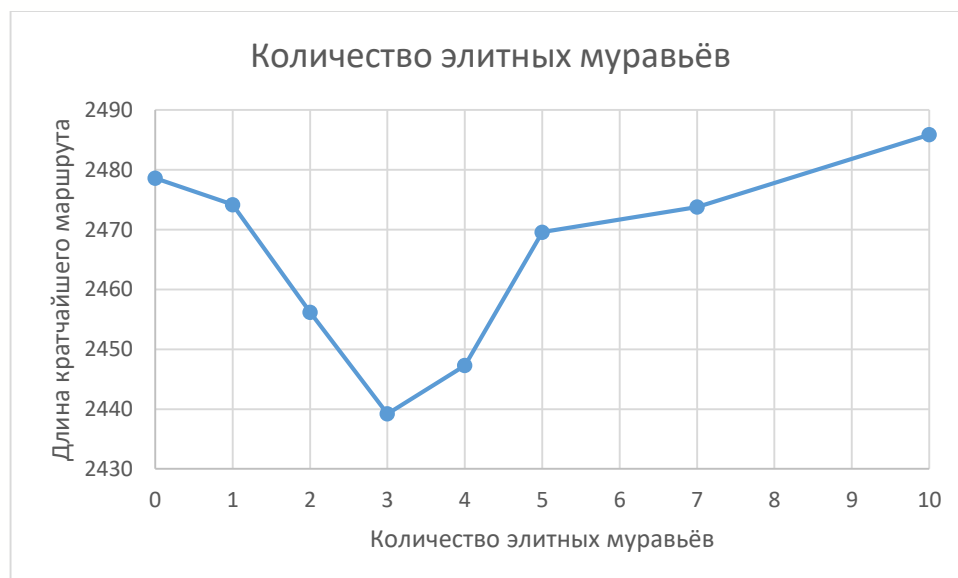


Рисунок 2.9. График средних найденных наилучших маршрутов в зависимости от количества элитных муравьёв

Таким образом при nElite=3 скорость схождения маршрута к наиболее оптимальному, в сравнении с классическим алгоритмом, увеличивается больше всего.

2.5.2. Модификация ASrank

При данной модификации всем муравьям, которые проложили маршрут в текущей итерации присваивается определённый ранг в зависимости от длины маршрута по сравнению с другими муравьями.

Все пройденные пути добавляются в список и сортируются от лучшего к худшему. В зависимости от очередности в данном отсортированном списке, на рёбра будет класться различное количество феромонов, которое можно вычислить по формуле

$$\Delta\tau(t) = \begin{cases} \frac{Q}{L_k(t)} * \left(1 - \frac{n}{n_{Ant}}\right), & (i, j) \in T_k(t) \\ 0, & (i, j) \notin T_k(t) \end{cases},$$

где n_{Ant} – количество муравьёв, и n – порядок пути от 0 до $n_{Ant}-1$ в отсортированном списке.

Таким образом для наилучшего пути феромоны будут отложены в полной мере, для второго по мере оптимальности, только $\frac{n_{Ant}-1}{n_{Ant}}$ часть, следующему $\frac{n_{Ant}-2}{n_{Ant}}$ и так далее, вплоть до последнего, который сможет отложить только $\frac{1}{n_{Ant}}$ часть от тех феромонов, которые бы мог отложить ранее. Таким образом, даже при малом различии длин оптимальных маршрутов на менее оптимальный маршрут будет откладываться значительно меньше феромона, чем на тот, что более оптимален.

Поскольку при данном алгоритме значительно изменяется принцип распределения феромонов, необходимо заново найти оптимальное значение Q .

Параметры эксперимента:

$n_{Ant}=25$ #Количество простых муравьёв

$\alpha=2$ #Коэффициент альфа (порядка значимости феромона)

$\beta=4$ #Коэффициент бэта (порядка значимости длины пути)

$\rho=0.3$ #Коэффициент испарения феромона

$n_{City}=50$ #Количество вершин

Максимальное время эксперимента: 1 секунда.

Таблица 2.35. Кратчайшие найденные маршруты для различного значения Q в ранговой модификации

$Q=67$	$Q=125$	$Q=250$	$Q=500$	$Q=1000$	$Q=2000$	$Q=4000$	$Q=8000$	$Q=16000$	$Q=32000$
2825,7	2837,9	2792,5	2784,9	2802,2	2775,7	2764,1	2770,2	2766,7	2814,7

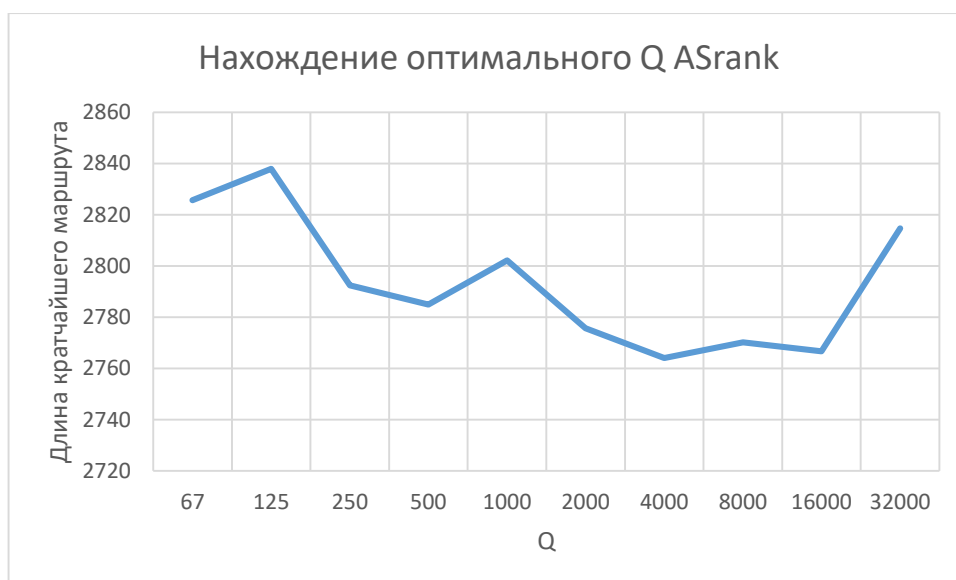


Рисунок 2.10. График средних найденных наилучших маршрутов в зависимости от Q

Таким образом $Q=4000$ для данных параметров можно считать оптимальным для модификации.

Сравним с классическим алгоритмом.



Рисунок 2.11. График сравнения нахождения кратчайшего маршрута классического алгоритма и модификации ASrank

К сожалению, реализация данной модификации не оправдала ожиданий, однако, вполне возможно, при иной реализации или другом использовании данная модификация также сможет улучшить характеристики классического алгоритма.

2.5.3. Метод локального поиска

Иногда для улучшения скорости и/или точности работы алгоритма можно совместить несколько методов в едином алгоритме. В данном случае был создан гибридный алгоритм, использующий методы локального поиска (А если конкретно, алгоритм «проходился» по маршруту, пробуя по одной замене связей, в попытках найти наилучшую из них).

Построим графики нахождения минимального значения для классического и гибридного алгоритма.

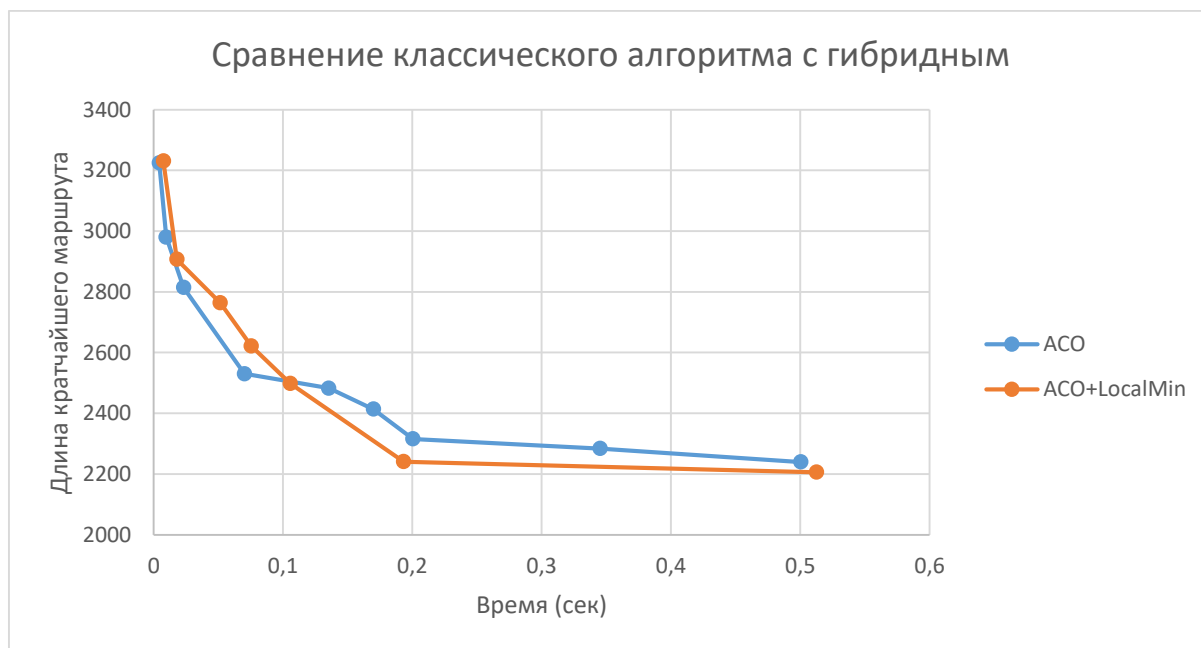


Рисунок 2.11. График сравнения нахождения кратчайшего маршрута классического алгоритма и алгоритма с гибридным использованием метода локального поиска

Как можно увидеть, скорость работы и точность алгоритма возросла по сравнению с классическим алгоритмом.

2.6. Сравнение муравьиного алгоритма с другими алгоритмами

2.6.1. Алгоритм полного перебора.

Данный алгоритм является одним из самых простых и универсальных, однако при этом самым медленным среди используемых.

Проведём сравнение нахождения оптимального маршрута при различном количестве вершин в графе для муравьиного алгоритма и метода перебора.

Таблица 2.36. Время нахождения кратчайшего маршрута для алгоритма полного перебора и Муравьиного алгоритма

nCity	5	7	9	11
Перебор	0.00016 с	0.00351 с	0.10269 с	9.260 с
ACO	0.0005 с	0.0009 с	0.0011 с	0.0019 с

Как можно увидеть, чем больше вершин в созданном графе, тем значительней разрыв по времени между муравьиным алгоритмом и методом полного перебора. И, хоть при малом количестве вершин метод перебора куда лучше в точности и скорости, при большом количестве вершин время вычислений гораздо больше, чем у муравьиного алгоритма.

Существует также и более быстрые методы. Например, метод ветвей и границ, являющийся модификацией полного перебора. Однако в данной работе он реализован не был, поскольку основная задача этого пункта – показать достоинства эмпирических алгоритмов, поскольку, хоть у метода перебора или метода ветвей и границ практически гарантированное нахождение наилучшего маршрута, время их вычислений резко растёт с количеством вершин. Поэтому при достаточно большом количестве вершин становится выгоднее использовать эмпирические алгоритмы. Такие как муравьиный алгоритм, генетический алгоритм, метод имитации отжига и другие. Хоть есть вероятность, что глобально оптимальный маршрут найден не будет или будет обнаружен через слишком большой промежуток времени, однако относительно быстро данные алгоритмы могут найти маршрут, длина которого близка к оптимальному,

использование которого может быть гораздо выгоднее по затраченному времени, чем продолжение вычислений и поиска самого оптимального пути.

Таким образом точные методы целесообразно использовать только для малого количества вершин в комбинаторной задаче. При большом количестве вершин куда лучше справляются с задачей эмпирические алгоритмы.

2.6.2. Генетический алгоритм

Генетический алгоритм, наряду с муравьиным алгоритмом, является одним из самых используемых алгоритмов для решения комбинаторных задач. Основной принцип работы генетического алгоритма таков: Создаётся поколение, каждое из которых могло получить мутацию, т.е. изменение в одном из участков генома (в данном случае вместо генома используется маршруты). После этого отбираются два лучших генома из созданных и проводится их скрещивание (создание на основании двух геномов третьего, обладающего признаками геномов обоих родителей). Мутация в данном эксперименте будет проводиться при помощи случайных изменений, применяемых в методах локального минимума.

Сравним графики нахождения оптимального маршрута при помощи модификаций Муравьиного алгоритма и генетического алгоритма.

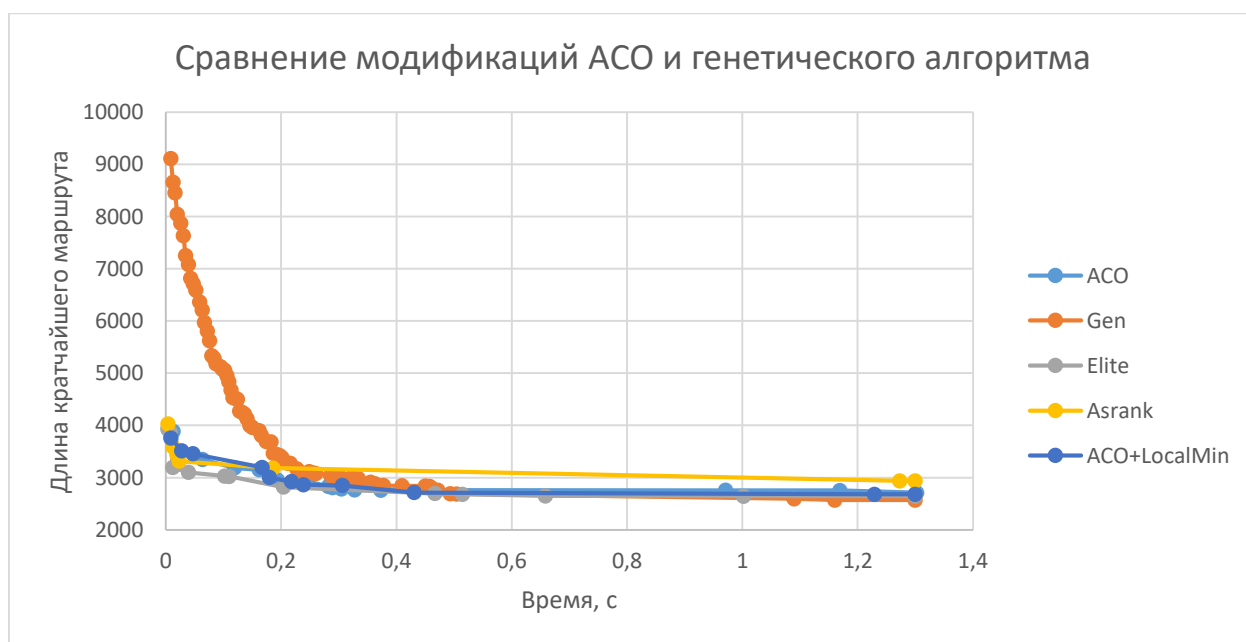


Рисунок 2.12. График сравнения нахождения кратчайшего маршрута при помощи генетического алгоритма и различных модификаций муравьиного алгоритма

Как можно увидеть, на начальных этапах генетический алгоритм значительно медленнее муравьиного алгоритма, оттого, что его изначальные значения генерируются случайно. Однако возможно улучшить скорость, задав эти значения при помощи, например, «жадного» алгоритма или того же муравьиного алгоритма.

Для сравнения точности приблизим нижнюю часть графика.

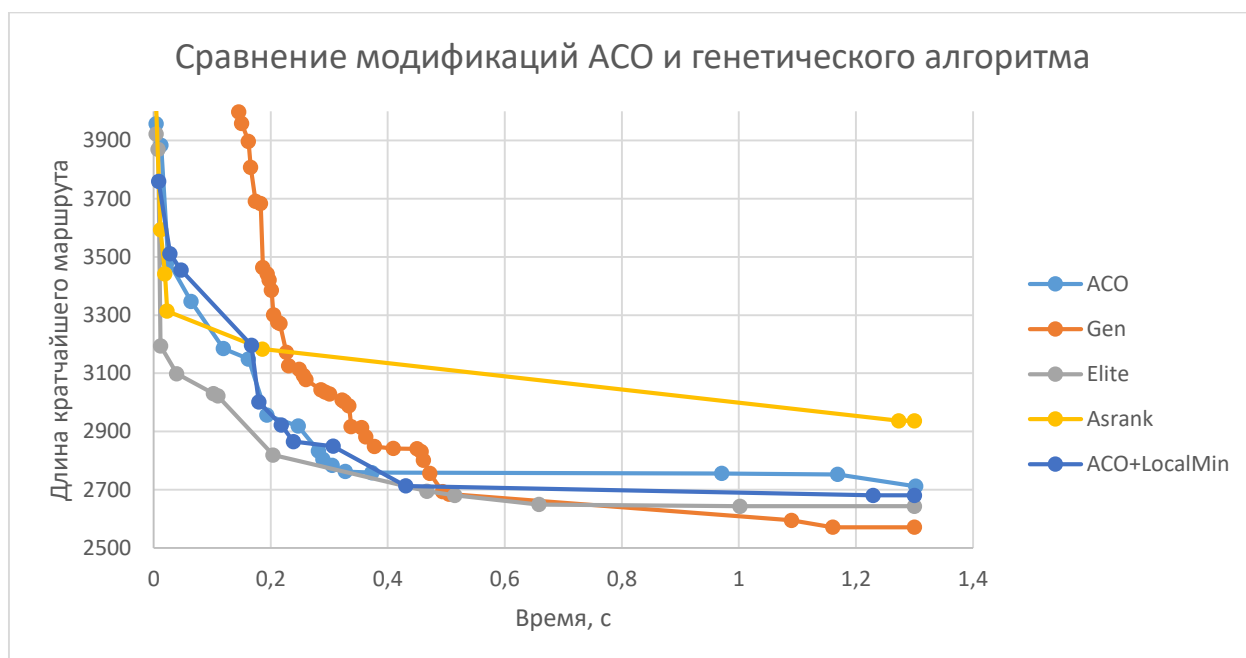


Рисунок 2.13. График сравнения нахождения кратчайшего маршрута при помощи генетического алгоритма и различных модификаций муравьиного алгоритма

Как можно увидеть, генетический алгоритм за заданный промежуток в данном случае дал более точное значение, чем муравьиные алгоритмы, хоть изначально и нашёл близкие к оптимальным маршруты значительно медленнее муравьиных алгоритмов. Однако стоит заметить, что у генетических алгоритмов также есть значительный недостаток – существенная вероятность попадания в локальный минимум. Алгоритм может найти субоптимальный

маршрут, после чего, какие бы ни происходили мутации, из него не выйдет, поскольку при мутации проводятся незначительные изменения генома, а самый оптимальный маршрут может глобально отличаться от найденного. В то время муравьиные алгоритмы подвержены этому в гораздо меньшей степени и через некоторый промежуток времени, неизвестно, короткий или значительный, но получит маршрут, являющийся самым оптимальным из возможных.

3. Целесообразность использования муравьиных алгоритмов

3.1. Преимущества

- Для решения задач, подобных задаче коммивояжера, сравнительно эффективны и имеют больше скорость сходимости чем многие другие (как было показано на примере генетического алгоритма)
- Применимы к множеству различных типов задач: сетевая маршрутизация, машинное обучение, планирование и многое другое.
- Могут адаптироваться к динамической обстановке и к изменению расстояний и местности
- В отличии от, например, генетических алгоритмов, опираются на память о всей колонии и меньше подвержены неоптимальным начальным решениям.
- Сходимость алгоритма гарантирована (хотя иногда для нахождения глобально оптимального значения может потребоваться значительное количество времени)

3.2. Недостатки

- Затруднение теоретического анализа:
 - 1) метод является экспериментальным, а не теоретическим и применим только во время самой работы роботов или её симуляции
 - 2) начальное распределение и передвижение во время использования алгоритма определено случайно на основании вероятностей и может изменяться при повторных экспериментах
- Как ранее было сказано, хоть нахождение оптимального маршрута гарантировано, однако время схождения не определено и может быть довольно значительным.
- Необходима предварительная настройка параметров для обеспечения эффективной работы, что нередко может быть сделано только после экспериментальных исследований или во время работы алгоритма

- Нередко необходимо использование модификаций или дополнительных алгоритмов для улучшения работы. Как пример, ранее представленный метод локального поиска.

Заключение

Алгоритм, хоть и является довольно простым в исполнении, способен показывать впечатляющие результаты в решении многих задачах, которые могут быть интерпретированы в виде графа. И, несмотря на некоторые недостатки, обладает рядом весомых преимуществ над другими алгоритмами оптимизации. Развитие муравьиных алгоритмов и использование их или их модификаций в качестве способа оптимизации является довольно перспективным направлением.

Применение его в различных задачах оптимизации, в том числе и для планирования маршрута, например, передвижения мобильных роботов, полностью оправдано и способно показать достойные результаты на фоне других существующих алгоритмов.

Список источников

1. Marco Dorigo. Optimization, learning and natural algorithms. 1992.
2. Marco Dorigo, Vittorio Maniezzo, and Alberto Colomi. Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). 1996
3. Bonabeau E., Dorigo M., Theraulaz G.. Swarm Intelligence: From Natural to Artificial Systems. 1999. New York: Oxford University Press.
4. С.Д. Штовба. Муравьиные алгоритмы: теория и применение, 2005 г.
5. Aleem Akhtar. Evolution of Ant Colony Optimisation Algorithm – A Brief Literature Review. 2019 г.
6. В.М. Курейчик, А.А. Кажаров. О некоторых модификациях муравьиного алгоритма.
7. Luca M Gambardella and Marco Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. Pages 252–260. 1995.
8. M Dorigo and Luca Maria Gambardella. Ant colonies for the traveling salesman problem. in BioSystems, 1997
9. Thomas Stutzle and Holger H Hoos. Improving the ant system: A detailed report on the max–min ant system. FG Intellektik, FB Informatik, TU Darmstadt, Germany, Tech. Rep. 1996.
10. Thomas Stutzle and Holger Hoos. Max-min ant system and local search for the traveling salesman problem. In Evolutionary Computation. Pages 309–314. 1997.
11. Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. A new rank based version of the ant system. a computational study. 1997.
12. Vittorio Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS journal on computing. Pages 358–369, 1999.
13. Oscar Cordón García, In˜aki Fern˜andez de Viana, and Francisco Herrera Triguero. Analysis of the best-worst ant system and its variants on the tsp. Mathware & soft computing. 2002.

- 14.Oscar Cordon, Inaki Fernandez de Viana, Francisco Herrera, and Llanos Moreno. A new aco model integrating evolutionary computation concepts: The best-worst ant system. 2000.
- 15.Michael Guntsch and Martin Middendorf. A population based approach for aco. In *Workshops on Applications of Evolutionary Computation*. Springer. Pages 72–81. 2002
- 16.Christian Blum. Theoretical and practical aspects of ant colony optimization. 2004
- 17.Christian Blum, Andrea Roli, and Marco Dorigo. Hc–aco: The hyper-cube framework for ant colony optimization. Pages 399–403. 2001.
- 18.Sabrina Oliveira, Mohamed Saifullah Hussin, Andrea Roli, Marco Dorigo, and Thomas Stützle. Analysis of the population-based ant colony optimization algorithm for the tsp and the qap. In *Evolutionary Computation (CEC)*. Pages 1734–1741. 2017.
- 19.Thomas Stutzle and Ruben Ruiz. Iterated greedy. *Handbook of Heuristics*. Pages 547–577. 2018.
- 20.Neha Jain Jasvinder, Pal Singh. *Modification of Ant Algorithm for Feature Selection*. 2006. LNCT, Bhopal, India.
- 21.Ginés D Guerrero, Joser M Cecilia, Antonio Llanes, Joser M García, Martyn Amos, and Manuel Ujaldorn. Comparative evaluation of platforms for parallel ant colony optimization. *The Journal of Supercomputing*. 2014.
- 22.Joser M Cecilia, Joser M García, Andy Nisbet, Martyn Amos, and Manuel Ujaldorn. Enhancing data parallelism for ant colony optimization on gpus. *Journal of Parallel and Distributed Computing*, 2013.
- 23.Laurence Dawson and Iain Stewart. Improving ant colony optimization performance on the gpu using cuda. In *Evolutionary Computation (CEC)*, 2013
- 24.Audrey DelerVacq, Pierre Delisle, Marc Gravel, and Michaël Krajecki. Parallel ant colony optimization on graphics processing units. *Journal of Parallel and Distributed Computing*, 2013.

25. Christian Blum, Mateu Yarbar Valle`s, and Maria J Blesa. An ant colony optimization algorithm for dna sequencing by hybridization. 2008.
26. Christian Blum. Beam-aco hybridizing ant colony optimization with beam search: An application to open shop scheduling. 2005
27. Oliver Korb, Thomas StuEtzle, and Thomas E Exner. An ant colony optimization approach to flexible protein–ligand docking. 2007
28. Christian Blum. Beam-aco for simple assembly line balancing. 2008
29. Luca Maria Gambardella and Marco Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. 2000
30. Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. 1998
31. Alena Shmygelska and Holger H Hoos. An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. 2005
32. Статья «Муравьиные алгоритмы» на портале Хабр, Электронный ресурс (<https://habr.com/ru/post/105302/>)
33. Сообщество исследователей муравьиных алгоритмов в социальной сети Вконтакте (https://vk.com/ant_colony_optimization)

Приложение 1

Код разработанной программы.

```
import random
import numpy as np
from PIL import Image, ImageDraw, ImageFont
import os
import math
from shutil import rmtree
import time
import openpyxl

def do_point_map(): #Создания случайных точек на плоскости
    for now in range(nCity):
        point_x.append(random.randint(1, window_x-2))
        point_y.append(random.randint(1, window_y-2))
        for other in range(now):
            dist=((point_x[other]-point_x[now])**2 + (point_y[other]-
point_y[now])**2)**0.5
            lenTrail[now][other]=dist
            lenTrail[other][now]=dist

def FirstDraw(): # Рисование ранее построенного графа
    image = Image.new("RGB", (window_x*ScrCoef,window_y*ScrCoef),
(255,255,255))
    draw = ImageDraw.Draw(image)
    for now in range(nCity):
        for other in range(now):

draw.line([(ScrCoef*point_x[other],ScrCoef*point_y[other]),(ScrCoef*point_x[now]
,ScrCoef*point_y[now])], fill = "black", width = 0)
        for now in range(nCity):
            if(now==start or now==end):
                draw.ellipse([(ScrCoef*point_x[now]-5,ScrCoef*point_y[now]-
5),(ScrCoef*point_x[now]+5,ScrCoef*point_y[now]+5)], fill = 'blue', outline = 'blue')
                draw.text((ScrCoef*point_x[now]+5,ScrCoef*point_y[now]+5),
str(now),fill='blue')
            else:
                draw.ellipse([(ScrCoef*point_x[now]-2,ScrCoef*point_y[now]-
2),(ScrCoef*point_x[now]+2,ScrCoef*point_y[now]+2)], fill = 'red', outline = 'red')
                draw.text((ScrCoef*point_x[now]+2,ScrCoef*point_y[now]+2),
str(now),fill='red')
        del draw
    image.save("Map.png", "PNG")
```

```

def ProgressDraw(time,bestLen,Trail,FoldName): #Рисование карты -
промежуточно найденного лучшего пути
    image = Image.new("RGB", (ScrCoef*window_x,ScrCoef*window_y),
(255,255,255))
    draw = ImageDraw.Draw(image)
    if(AllLines==1):
        for now in range(nCity):
            for other in range(now):

draw.line([(ScrCoef*point_x[now],ScrCoef*point_y[now]),(ScrCoef*point_x[other],
ScrCoef*point_y[other])], fill = "black", width = 2)

        for now in range(nCity):
            for other in range(now):
                draw.text((ScrCoef*(point_x[other]+point_x[now])/2-
2,ScrCoef*(point_y[other]+point_y[now])/2-2), str('%0.3f' %
pherTrail[now][other]),fill='blue')

        for i in range(len(Trail)-1):

draw.line([(ScrCoef*point_x[Trail[i]],ScrCoef*point_y[Trail[i]]),(ScrCoef*point_x[
Trail[i+1]],ScrCoef*point_y[Trail[i+1]])], fill = "red", width = 5)
        for now in range(nCity):
            draw.ellipse([(ScrCoef*point_x[now]-2,ScrCoef*point_y[now]-
2),(ScrCoef*point_x[now]+2,ScrCoef*point_y[now]+2)], fill = 'black', outline
='black')
            draw.ellipse([(ScrCoef*point_x[start]-4,ScrCoef*point_y[start]-
4),(ScrCoef*point_x[start]+4,ScrCoef*point_y[start]+4)], fill = 'blue', outline = 'blue')
            draw.ellipse([(ScrCoef*point_x[end]-4,ScrCoef*point_y[end]-
4),(ScrCoef*point_x[end]+4,ScrCoef*point_y[end]+4)], fill = 'red', outline = 'red')
        for now in range(nCity):
            draw.text((ScrCoef*point_x[now]+2,ScrCoef*point_y[now]+2),
str(now),fill='black')

    del draw
    name=FoldName+"/"+ "time"+str(time)+"best"+str(bestLen)+".png"
    image.save(name, "PNG")

def probability(i,j): #Расчёт весов определённого отрезка пути для расчёта
вероятности прохода муравья по нему.
    p=(pherTrail[i][j]**Alpha)*((1/lenTrail[i][j])**Beta)
    return p

```

```

def UpdPher(): # Обновление феромонов
    for i in range(nCity):
        for j in range (i):
            if(lenTrail[i][j]!=0):
                pherTrail[i][j]=pherTrail[i][j]*(1-Rho)+deltaPher[i][j]
            if(pherTrail[i][j]<pherMin):
                pherTrail[i][j]=pherMin
            if(pherTrail[i][j]>pherMax):
                pherTrail[i][j]=pherMax
            pherTrail[j][i]=pherTrail[i][j]
            deltaPher[i][j]=0
            deltaPher[j][i]=0

def Move(now,tabu,to): #Функция выбора следующей точки для перехода
    WhereList=[]
    ProbList=[]
    if(len(tabu)==nCity-1):
        return to
    for cit in range(nCity):
        flag=0
        for i in range(len(tabu)):
            if(tabu[i]==cit):
                flag=1
                break
        if(len(tabu)<nCity-1 and cit!=to and flag==0):
            WhereList.append(cit)
            ProbList.append(probability(now,cit))
    return random.choices(WhereList, weights=ProbList)[0]

def March(Fr,To): #Функция нахождения маршрута муравья из одной точки в
    другую
    now=Fr
    tabu=[]
    tabu.append(now)
    while 1:
        next=Move(now,tabu,To)
        now=next
        tabu.append(now)
        if (now==To):
            return tabu

def Length(Trail): #Расчёт длины пути на графе
    l=0
    for i in range(len(Trail)-1):
        l+=lenTrail[Trail[i]][Trail[i+1]]

```

```

return l

def PutPher(Trail,l): #Распределение феромонов после движения муравьёв
    for sch in range(len(Trail)-1):
        deltaPher[Trail[sch]][Trail[sch+1]]+=Q/l
        deltaPher[Trail[sch+1]][Trail[sch]]=deltaPher[Trail[sch]][Trail[sch+1]]

def Gross(tabuNow,now): # Функция полного перебора
    if (len(tabuNow)==nCity-1):
        tabuCopy=tabuNow.copy()
        tabuCopy.append(end)
        l=Length(tabuCopy)
        if(len(GrossLen)==0):
            GrossLen.append(round(l,1))
            NowTime=time.time()
            GrossTime.append(round(NowTime-StartTime,4))
            return
        if(l<GrossLen[len(GrossLen)-1]):
            GrossLen.append(round(l,1))
            NowTime=time.time()
            GrossTime.append(round(NowTime-StartTime,4))
            return
    for i in range(nCity):
        flag=0
        if (i==end):
            flag=1
        for j in range(len(tabuNow)):
            if (i==tabuNow[j]):
                flag=1
        if(flag==0):
            tabuCopy=tabuNow.copy()
            tabuCopy.append(i)
            Gross(tabuCopy,i)

def Together(route1,route2): # Скрещивание для Генетического алгоритма
    newroute=[]
    newrouteTail=[]
    for i in range(len(route1)):
        if (i%2==0):
            for First in range(len(route1)):
                flag=0
                for Finding1 in range(len(newroute)):
                    if(newroute[Finding1]==route1[First]):
                        flag=1
                        break

```

```

    if(flag==0):
        for Finding2 in range(len(newrouteTail)):
            if(newrouteTail[Finding2]==route1[First]):
                flag=1
                break
        if(flag==0):
            newroute.append(route1[First])
            break
    else:
        for Second in range(len(route2)):
            flag=0
            for Finding1 in range(len(newroute)):
                if(newroute[Finding1]==route2[len(route2)-Second-1]):
                    flag=1
                    break
            if(flag==0):
                for Finding2 in range(len(newrouteTail)):
                    if(newrouteTail[Finding2]==route2[len(route2)-Second-1]):
                        flag=1
                        break
                if(flag==0):
                    newrouteTail.append(route2[len(route2)-Second-1])
                    break
        for i in range(len(newrouteTail)):
            newroute.append(newrouteTail[len(newrouteTail)-i-1])
    return newroute

def Mutation1(route): #Вариант 1 мутации генома генетического алгоритма
    newroute=route.copy()
    cit1=random.randint(1,len(route)-2)
    cit2=random.randint(1,len(route)-2)
    newroute[cit1]=route[cit2]
    newroute[cit2]=route[cit1]
    return newroute

def Mutation2(route): #Вариант 2 мутации генома генетического алгоритма
    newroute=route.copy()
    cit1=random.randint(1,len(route)-2)
    cit2=random.randint(1,len(route)-2)
    delta=cit2-cit1
    if (cit1>cit2):
        for i in range(cit1-cit2+1):
            newroute[cit2+i]=route[cit1-i]
    else:
        for i in range(cit2-cit1+1):

```

```

    newroute[cit1+i]=route[cit2-i]
return newroute

```

```

def Mutation3(route): #Вариант 3 мутации генома генетического алгоритма
    newroute=route.copy()
    cit1=random.randint(1,len(route)-2)
    cit2=random.randint(1,len(route)-2)
    delta=cit2-cit1
    if (cit1>cit2):
        for i in range(cit1-cit2):
            newroute[cit2+i+1]=route[cit2+i]
            newroute[cit2]=route[cit1]
    else:
        for i in range(cit2-cit1):
            newroute[cit1+i+1]=route[cit1+i]
            newroute[cit1]=route[cit2]
    return newroute

```

```

def RandGen(): #Случайная генерация первого поколения для генетического
алгоритма
    newroute=[]
    newroute.append(start)
    inside=[]
    for i in range(nCity):
        if(i!=start and i!=end):
            inside.append(i)
    random.shuffle(inside)
    for j in range(len(inside)):
        newroute.append(inside[j])
    newroute.append(end)
    return newroute

```

```

def Find2Best(genlen): #Нахождение двух лучших геномов среди поколения
генетического алгоритма
    top1=1000000000
    top2=1000000000
    ntop1=0
    ntop2=0
    flagN=0
    for i in range (len(genlen)):
        if (genlen[i]<top1):
            if(flagN==1):
                ntop2=ntop1
                top2=top1
            top1=genlen[i]

```



```

    ntop1=i
elif (genlen[i]<top2):
    flagN=1
    top2=genlen[i]
    ntop2=i
return ntop1,ntop2

```

def FindBest(genlen): #Нахождение одного лучшего генома среди поколения генетического алгоритма

```

    top1=1000000000
    ntop1=0
    for i in range (len(genlen)):
        if (genlen[i]<top1):
            top1=genlen[i]
            ntop1=i
    return ntop1

```

def Selection(r1,r2): #Скрещивание двух лучших геномов поколения генетического алгоритма

```

    child1=Together(r1,r2)
    child2=Together(r2,r1)
    toplist=[r1,r2,child1,child2]
    toplen=[Length(r1),Length(r2),Length(child1),Length(child2)]
    return toplist[FindBest(tolen)]

```

def RankModSort(FirstList): #Сортировка списка маршрутов для модификации ASrank

```

    TopList=[]
    LenList=[]
    for i in range (len(FirstList)-1):
        flag=0
        l=Length(FirstList[i])
        for j in range(len(TopList)-1):
            if(l<LenList[j]):
                flag=1
                TopList.insert(j,FirstList[i])
                LenList.insert(j,l)
                break
        if(flag==0):
            TopList.append(FirstList[i])
            LenList.append(l)
    return TopList, LenList

```

def RankModPher(TopList, LenList): #Распределение феромонов для модификации ASrank

```

for i in range(len(TopList)-1):
    for sch in range(len(TopList[i])-1):
        deltaPher[TopList[i][sch]][TopList[i][sch+1]]+=(Q/l)*(1-i/len(TopList))

deltaPher[TopList[i][sch+1]][TopList[i][sch]]=deltaPher[TopList[i][sch]][TopList[i][sch+1]]

def LocalFind(route): #Гибридный элемент локального поиска на маршруте
    best=Length(route)
    outroute=route.copy()
    for i in range(2,len(route)-2):
        for j in range(i-10,i):
            if(j<1):
                break
            newroute=route.copy()
            for k in range(i-j):
                newroute[j+k+1]=route[j+k]
            newroute[j]=route[i]
            if (Length(newroute)<best):
                best=Length(newroute)
                outroute=newroute
    return outroute

point_x=[]
point_y=[]
point_near=[]

window_x=400 #ширина поля
window_y=400 #высота поля
ScrCoef=2 #Масштабирование при выводе карт

start=0 #Точка муравейника
end=1 #Точка цели

nCity=50 #Количество вершин

lenTrail = [[0 for j in range(nCity)] for i in range(nCity)]
do_point_map()
FirstDraw() #Сохранение рисунка карты

nAnt=25 #Количество простых муравьёв
Alpha=2 #Коэффициент альфа (порядка значимости феромона)
Beta=4 #Коэффициент бэта (порядка значимости длины пути)
Rho=0.3 #Коэффициент испарения феромона
Q=2000 #Коэффициент увеличения феромона

```

```

pherMin=1 #Минимальное количество феромона на рёбрах графа
pherMax=30 #Максимальное количество феромона на рёбрах графа

# Модификации
nElite=0 #Количество элитных муравьёв
ASrank=0 #Модификация ASrank. 1 - активно, 0 - неактивно
LocalMin=0 #Добавление элемента локального поиска. 1 - активно, 0 -
неактивно

# Имена и циклы
dir_name="Tests"

# Первая переменная цикла
Par1Name="Q"
Rng1max = 1
Par1Mas=[500,1000,2000,4000,8000,12000,16000] # массив значений 1ой
переменной

# Вторая переменная цикла
Par2Name="Rho"
Rng2max = 1
Par2Mas=[0.3] # массив значений 2ой переменной

# Третья переменная цикла
Par3Name="nElite"
Rng3max = 1
Par3Mas=[0,1,2,3,4,5,7,10]

MaxCikl=9

ShowProcessMaps=0 # Отображение промежуточных маршрутов на картах: 0 -
нет, 1 - да
ShowProgressString=0 # Отображение строк с нахождением новых оптимальных
путей: 0 - нет, 1 - да
AllLines=0 # Отображение на промежуточных картах всех рёбер и феромонов: 0
- нет, 1 - да

#Цикл для исследования АСО с различными параметрами
if (os.path.exists(dir_name)):
    rmtree(dir_name)
if (ShowProcessMaps==1):
    os.mkdir(dir_name)

```

```

BestList=[]
TimerList=[]
StepList=[]
MedianList=[]
MidList=[]
TimeMidList=[]

for Par1 in range(Rng1max):
    #Q=Par1Mas[Par1]

    BestListPar1=[]
    TimerListPar1=[]
    StepListPar1=[]
    MedianListPar1=[]
    MidListPar1=[]
    TimeMidListPar1=[]

    if(ShowProcessMaps==1):
        dir_name1=dir_name+"/"+Par1Name+"_"+str(Par1)
        os.mkdir(dir_name1)
    for Par2 in range(Rng2max):
        #Rho=Par2Mas[Par2]

        BestListPar2=[]
        TimerListPar2=[]
        StepListPar2=[]
        MedianListPar2=[]
        MidListPar2=[]
        TimeMidListPar2=[]

        if(ShowProcessMaps==1):
            dir_name2=dir_name1+"/"+Par2Name+"_"+str(Par2)
            os.mkdir(dir_name2)

    for Par3 in range(Rng3max):
        #Q=Par3Mas[Par3]

        BestListPar3=[]
        TimerListPar3=[]
        StepListPar3=[]

        if(ShowProcessMaps==1):
            dir_name3=dir_name2+"/"+Par3Name+"_"+str(Par3)
            os.mkdir(dir_name3)
        for Cikli in range(MaxCikli):

```

```

BestListCikl=[]
TimerListCikl=[]
StepListCikl=[]
RealStartTime = time.time()

if(ShowProcessMaps==1):
    dir_name4=dir_name3+"/"+Cikl+"_"+str(Cikl+1)
    os.mkdir(dir_name4)
pherTrail = [[pherMin for j in range(nCity)] for i in range(nCity)]
deltaPher = [[0 for j in range(nCity)] for i in range(nCity)]
bestLen=-1
bestTrail=[]
bestTime=0
#Tend//nAnt+(nAnt//10)*70
step=0
while(time.time()-RealStartTime<1.5):
    step+=1

    if(ASrank==1):
        StepRoutes=[]
        StepLens=[]

    for Ant in range(nAnt):
        if(step%2==0):
            way=March(start,end)
        else:
            way=March(end,start)
        if(LocalMin==1):
            way=LocalFind(way)
        l=Length(way)
        if(bestLen== -1 or l<bestLen):
            bestLen=l
            bestTrail=way
            bestTime=step

    RealNowTime = time.time()
    BestListCikl.append(bestLen)
    TimerListCikl.append(RealNowTime-RealStartTime)
    StepListCikl.append(step)

    if(ShowProgressString==1):
        print("На шаге",step," ("",round(RealNowTime-RealStartTime,4),"сек)
найден путь длиной",bestLen)
    if(ShowProcessMaps==1):
        ProgressDraw(step,round(bestLen,1),bestTrail,dir_name4)

```

```

        if(ASrank==1):
            StepRoutes.append(way)
    if(ASrank==1):
        StepRoutes,StepLens=RankModSort(StepRoutes)
        RankModPher(StepRoutes,StepLens)
    else:
        PutPher(way,1)

    for EliteAnt in range(nElite):
        PutPher(bestTrail,bestLen)
    UpdPher()
    if(ShowProgressString==1):
        print("len",bestLen)
        print("step",bestTime)
        print("time",round(RealNowTime-RealStartTime,4))
        #print(Par1Name,Par1Mas[Par1])
        #print(Par2Name,Par2Mas[Par2])
        #print(Par3Name,Par3Mas[Par3])
        #print("Cikl = "+str(Cikl))

print(Par1*Rng2max*Rng3max*MaxCikl+Par2*Rng3max*MaxCikl+Par3*MaxCikl
+Cikl+1,"/",Rng1max*Rng2max*Rng3max*MaxCikl)

    BestListPar3.append(BestListCikl) #Собираем в список с каждого цикла
    TimerListPar3.append(TimerListCikl)
    StepListPar3.append(StepListCikl)

copyBest=BestListPar3.copy()
copyBest.sort()
Median=copyBest[MaxCikl//2]
for k in range (MaxCikl):
    if(BestListPar3[k]==Median):
        MedianNum=k
        break

sum=0
for k in range (MaxCikl):
    sum+=BestListPar3[k][len(BestListPar3[k])-1]
Mid=sum/MaxCikl
MidListPar2.append(Mid)

Timesum=0
for k in range (MaxCikl):
    Timesum+=TimerListPar3[k][len(BestListPar3[k])-1]

```

```
TimeMid=Timesum/MaxCikl
TimeMidListPar2.append(TimeMid)
```

```
MedianListPar2.append(MedianNum) #Собираем в список разных параметров
3
BestListPar2.append(BestListPar3)
TimerListPar2.append(TimerListPar3)
StepListPar2.append(StepListPar3)
MidListPar1.append(MidListPar2)
TimeMidListPar1.append(TimeMidListPar2)
MedianListPar1.append(MedianListPar2) #Собираем в список разных
параметров 2
BestListPar1.append(BestListPar2)
TimerListPar1.append(TimerListPar2)
StepListPar1.append(StepListPar2)
MidList.append(MidListPar1)
TimeMidList.append(TimeMidListPar1)
MedianList.append(MedianListPar1) #Собираем в список разных параметров 1
BestList.append(BestListPar1)
TimerList.append(TimerListPar1)
StepList.append(StepListPar1)
```

```
#Полный перебор
StartTime = time.time()
tabu=[]
GrossTime=[]
GrossLen=[]
tabu.append(start)
Gross(tabu,start)
NowTime=time.time()
AllTime=NowTime-StartTime
print(AllTime)
print(GrossLen[len(GrossLen)-1])
```

```
#Генетический алгоритм
nGen=50 #Кол-во особей в поколении
Epoch=1000 #Кол-во поколений
RealStartTime = time.time()
nowgen=[]
nowgenlen=[]
GenLen=[]
GenTime=[]
```

```

for i in range (nGen):
    nowroute=RandGen()
    nowgen.append(nowroute)
    nowgenlen.append(Length(nowroute))
top1,top2=Find2Best(nowgenlen)

for step in range(Epoch):

    Leader=Selection(nowgen[top1],nowgen[top2])

    for i in range(nGen-1):
        mark=random.randint(1,3)
        if(mark==1):
            nowgen[i]=Mutation1(Leader)
        elif(mark==2):
            nowgen[i]=Mutation2(Leader)
        else:
            nowgen[i]=Mutation3(Leader)
        nowgenlen[i]=Length(nowgen[i])
    nowgen[nGen-1]=Leader
    nowgenlen[nGen-1]=Length(Leader)
    top1,top2=Find2Best(nowgenlen)
    if(len(GenLen)==0):
        GenLen.append(nowgenlen[top1])
        RealNowTime = time.time()
        GenTime.append(RealNowTime-RealStartTime)
    elif(GenLen[len(GenLen)-1]>nowgenlen[top1]):
        GenLen.append(nowgenlen[top1])
        RealNowTime = time.time()
        GenTime.append(RealNowTime-RealStartTime)
    print(GenLen[len(GenLen)-1])
    print(GenTime[len(GenTime)-1])

```

#ACO. Сохранение истории нахождения оптимального маршрута в Excel.

```

wb = openpyxl.Workbook()
wb.create_sheet(title = 'History', index = 0)
sheet = wb['History']

```

```

for i in range (len(BestList)):
    for j in range (len(BestList[i])):
        for k in range (len(BestList[i][j])):

```

```

NowColumn=i*len(BestList[i])*len(BestList[i][j])*5+j*len(BestList[i][j])*5+k*5+1
    NowCell = sheet.cell(row = 1, column = NowColumn)
    NowCell.value = Par1Name+str(Par1Mas[i])

```



```

NowCell = sheet.cell(row = 2, column = NowColumn)
NowCell.value = Par2Name+str(Par2Mas[j])
NowCell = sheet.cell(row = 3, column = NowColumn)
NowCell.value = Par3Name+str(Par3Mas[k])
NowCell = sheet.cell(row = 4, column = NowColumn)

b1=round(BestList[i][j][k][MedianList[i][j][k]][len(BestList[i][j][k][MedianList[i][j][k]])-1],1)
NowCell.value = "Best="+str(b1)

NowColumn=i*len(BestList[i])*len(BestList[i][j])*5+j*len(BestList[i][j])*5+k*5+2
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Best"

for Sch1 in range (len(BestList[i][j][k][MedianList[i][j][k]])):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =round(BestList[i][j][k][MedianList[i][j][k]][Sch1],1)

NowColumn=i*len(BestList[i])*len(BestList[i][j])*5+j*len(BestList[i][j])*5+k*5+3
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Step"
for Sch1 in range (len(StepList[i][j][k][MedianList[i][j][k]])):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =round(StepList[i][j][k][MedianList[i][j][k]][Sch1])

NowColumn=i*len(BestList[i])*len(BestList[i][j])*5+j*len(BestList[i][j])*5+k*5+4
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Time"
for Sch1 in range (len(TimerList[i][j][k][MedianList[i][j][k]])):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =round(TimerList[i][j][k][MedianList[i][j][k]][Sch1],4)

wb.save('AntHistory.xlsx')

#ACO. Сохранение средних длин найденных оптимальных маршрутов.
wb = openpyxl.Workbook()
wb.create_sheet(title = 'TimeBest', index = 0)
sheet2 = wb['TimeBest']
wb.create_sheet(title = 'Best', index = 0)
sheet = wb['Best']

```

```

for i in range (Rng2max):
    NowColumn=i*(Rng3max+2)+1
    NowCell = sheet.cell(row = 1, column = NowColumn)
    NowCell.value = Par2Name+"="+str(Par2Mas[i])
    for sch in range (Rng1max):
        NowCell = sheet.cell(row = sch+2, column = NowColumn)
        NowCell.value = Par1Name+"="+str(Par1Mas[sch])
    for j in range (Rng3max):
        NowColumn=i*(Rng3max+2)+j+2
        NowCell = sheet.cell(row = 1, column = NowColumn)
        NowCell.value = Par3Name+"="+str(Par3Mas[j])
    for k in range (Rng1max):
        NowCell = sheet.cell(row = k+2, column = NowColumn)
        NowCell.value = round(MidList[k][i][j],1)

for i in range (Rng2max):
    NowColumn=i*(Rng3max+2)+1
    NowCell = sheet2.cell(row = 1, column = NowColumn)
    NowCell.value = Par2Name+"="+str(Par2Mas[i])
    for sch in range (Rng1max):
        NowCell = sheet2.cell(row = sch+2, column = NowColumn)
        NowCell.value = Par1Name+"="+str(Par1Mas[sch])
    for j in range (Rng3max):
        NowColumn=i*(Rng3max+2)+j+2
        NowCell = sheet2.cell(row = 1, column = NowColumn)
        NowCell.value = Par3Name+"="+str(Par3Mas[j])
    for k in range (Rng1max):
        NowCell = sheet2.cell(row = k+2, column = NowColumn)
        NowCell.value = round(TimeMidList[k][i][j],4)

wb.save('AntSheet.xlsx')

```

#Метод перебора. Сохранение истории нахождения оптимального маршрута в Excel.

```

Grwb = openpyxl.Workbook()
Grwb.create_sheet(title = 'Первый лист', index = 0)
sheet = Grwb['Первый лист']

```

```

NowColumn=1
NowCell = sheet.cell(row = 1, column = NowColumn)
b1=GrossLen[len(GrossLen)-1]
NowCell.value = "Best="+str(b1)
NowCell = sheet.cell(row = 2, column = NowColumn)
NowCell.value = "Time="+str(AllTime)

```

```

NowColumn=2
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Best"
for Sch1 in range (len(GrossLen)):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =GrossLen[Sch1]
NowColumn=3
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Time"
for Sch1 in range (len(GrossTime)):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =GrossTime[Sch1]

```

```

Grwb.save('Gross.xlsx')

```

#Генетический алгоритм. Сохранение истории нахождения оптимального маршрута в Excel

```

Genwb = openpyxl.Workbook()
Genwb.create_sheet(title = 'Первый лист', index = 0)
sheet = Genwb['Первый лист']

```

```

NowColumn=1
NowCell = sheet.cell(row = 1, column = NowColumn)
b1=GenLen[len(GenLen)-1]
NowCell.value = "Best="+str(b1)
NowColumn=2
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Best"
for Sch1 in range (len(GenLen)):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =GenLen[Sch1]
NowColumn=3
NowCell = sheet.cell(row = 1, column = NowColumn)
NowCell.value = "Time"
for Sch1 in range (len(GenTime)):
    NowCell = sheet.cell(row = Sch1+2, column = NowColumn)
    NowCell.value =GenTime[Sch1]

```

```

Genwb.save('Gen.xlsx')

```