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: Исследование возможностей применения муравьиных алгоритмов для решения задач о назначениях в группе мобильных роботов

приказ университета о направлении на практику
от «27» августа 2020г. № 3269-с

Отчет представлен к рассмотрению:
Студент группы КРБО-02-17 ____________________ Константинов М.В.

(подпись) *(расшифровка подписи)*

«29» 03 2021г.

Отчет утвержден.
Допущен к защите:

Руководитель практики от кафедры ____________________ С.В. Манько

(подпись) *(расшифровка подписи)*

«2» 03 2021г.

Руководитель практики от Университета ____________________ А.А. Сухоленцева

(подпись) *(расшифровка подписи)*

«29» 03 2021г.

Москва 2020

ОТЧЕТ
по прохождению учебной практики
студента 4 курса учебной группы КРБО-02-17
института кибернетики

Константинова Максима Вячеславовича

1. Практику проходил с 01.09.2020 по 20.12.2020 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: было изучено использование муравьиных алгоритмов для решения задачи о назначениях, разработано ПО для исследования муравьиных алгоритмов в задаче о назначениях.

Предложения по совершенствованию организации и прохождения практики: предложений нет.

Студент ____________ (Константинов М.В.)

«29» 03 2021г.

Заключение руководителя практики от профильной организации:
Приобрел следующие профессиональные навыки: разработка и отладка ПО, разработка и обоснование методики проведения экспериментальных исследований
Проявил себя как: трудолюбивый, квалифицированный и кропотливый работник, способный самостоятельно выполнять поставленные задачи

Проделанная работа полностью соответствует заданию и заслуживает оценки „отлично"

Руководитель практики от профильной организации

____
(наименование от профильной организации)

профессор, д.т.н. ____________ Манько С.В
(должность) (подпись) (фамилия и инициалы)

Отчет проверил:
Руководитель практики от Университета
____________ А.А. Сухоленцева
(подпись) (фамилия и инициалы)
29.03.2021г.

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 4 курса учебной группы КРБО-02-17

Константинова Максима Вячеславовича

Место и время практики: научная лаборатория кафедры проблем управления «Проектирование роботов и РТС», Г-210. С 01.09.2020 по 20.12.2020.

Должность на практике:____

1. ЦЕЛЕВАЯ УСТАНОВКА: Исследование возможностей применения муравьиных алгоритмов для решения задач о назначениях в группе мобильных роботов

2. СОДЕРЖАНИЕ ПРАКТИКИ:

2.1 Изучить: Особенности использования муравьиного алгоритма для решения задачи о назначениях, другие методы решения задачи о назначениях.

2.2 Практически выполнить: Разработку программных средств для решения задачи о назначениях на основе муравьиных алгоритмов в группе мобильных роботов; проведение модельных экспериментов

2.3 Ознакомиться: С методами построения маршрута для движения мобильного робота

3.ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ:____________________________

4. ОГРАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:___________

Заведующий кафедрой:

«__»__________20__г. ____________________ М.П. Романов

(подпись) (ФИО)

СОГЛАСОВАНО:

Руководитель практики от кафедры:

«23» 09 2020г. ____________________ С.В. Манько

(подпись) (ФИО)

Руководитель практики от Университета:

«24» 09 2020г. ____________________ А.А. Сухоленцева

(подпись) (ФИО)

Задание получил:

«23» 09 2020г. ____________________ М.В.Константинов

(подпись) (ФИО)

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ

студента 4 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**
1-5	**02.09.2020 — 04.10.2020**	Изучение особенностей алгоритма для решения задачи о назначениях, изучение способов построения пути	Выполнено
6-11	**05.10.2020 — 15.11.2020**	Разработка программного обеспечения для изучения муравьиных алгоритмов.	Выполнено
12-13	**16.11.2020 — 29.11.2020**	Проведение модельных экспериментов и анализ полученных данных	Выполнено
14-16	**30.11.2020 — 20.12.2020**	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

Проведенные инструктажи:

Охрана труда: «23» 08 2020г.

Инструктирующий ____________ (Манько С.В, профессор, д.т.н.)
Подпись Расшифровка, должность

Инструктируемый ____________ (Константинов И.В)
Подпись Расшифровка

Техника безопасности: «23» 08 2020г.

Инструктирующий ____________ (Манько С.В, профессор, д.т.н.)
Подпись Расшифровка, должность

Инструктируемый ____________ (Константинов И.В)
Подпись Расшифровка

Пожарная безопасность: «23» 09 2020г.

Инструктирующий ____________ (Манько С.В, профессор, д.т.н.)
Подпись Расшифровка, должность

Инструктируемый ____________ (Константинов И.В)
Подпись Расшифровка

С правилами внутреннего распорядка ознакомлен: «23» 08 2020г.

____________ (Константинов И.В)
Подпись Расшифровка

Оглавление

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.

Муравьиный алгоритм (Ant Colony Optimosation – ACO, Ant Systems – AS), является одним из алгоритмов, который может и эффективно используется для решения многих задач оптимизации. Одной из таких является задача о назначениях.

В рамках практической работы будут изучены статьи об уже ранее проведённых работах по изучению применения муравьиного алгоритма в задачах о назначениях. Будут разработаны программные средства для изучения и симуляции работы алгоритма, а также подведены результаты.

1. Муравьиный алгоритм как способ решения задачи о назначениях

1.1. Общее описание алгоритма

Муравьиный алгоритм [1, 2, 3, 4] был разработан в 1991 годы и 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. При выборе точки дальнейшего перемещения, кроме списка запретов, агент руководствуется «привлекательностью» доступных путей на основании их длины и уровню феромонового следа, оставленного на нём. Количество феромона на определённом пути не постоянно, а всё время обновляется, как уменьшаясь со временем, так и увеличиваясь из-за агентов, «проходивших» ранее по этому 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.

τij – уровень феромона на пути, связывающем вершины i и j.

dij – «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 α = 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 β = 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:

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

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

где:

Tk – множество вершин, пройденных во время пути.

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

Lk – 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:

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

Формула 1.3. Расчёт количества феромона, обновляемого с течением времени

где ρ (rho) – 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.

1.2. Задача о назначениях

В научном сообществе на протяжении многих лет с момента создания муравьиного алгоритма ведутся активные исследования и создаются многочисленные модификации. Муравьиный алгоритм показал себя эффективным во многих задачах. Одна из таких – задача о назначениях [5].

Задача о наилучшем распределении некоторого числа работ между таким же числом исполнителей. При ее решении ищут оптимальное назначение из условия максимума общей производительности, которая равна сумме производительности исполнителей.

Таким образом данную задачу можно представить в виде двудольного графа (Рисунок 1), одна половина которого – исполнители (на рисунке - агенты, поскольку далее будет рассматриваться применение для мобильных роботов), а другая – задачи. Каждая вершина графа исполнителей соединена с каждой вершиной – задачей. Рёбра, соединяющие их – и есть производительность / стоимость для определённого исполнителя в конкретной задаче.

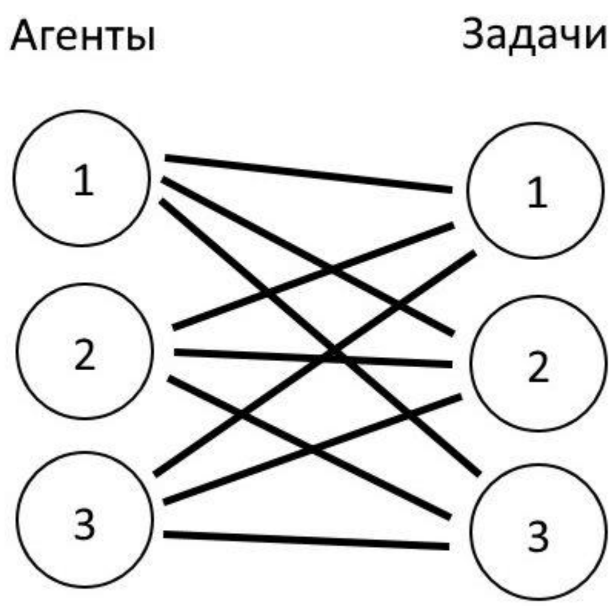


Рисунок 1. Двудольный граф для решения задачи о назначениях

Стоимость на рёбрах в таком графе удобно представить в виде квадратной матрицы или таблицы N*N, где N – количество агентов/задач (Таблица 1).

Таблица 1. Стоимость назначений агентов

Агенты\Задачи	1	2	3
1	a11	a12	a13
2	a21	a22	a23
3	a31	a32	a33

Производительность исполнителей в различных случаях может исчисляться в, например, времени исполнения, длине маршрута следования, энергозатратам и тому подобном. Обычно, как было сказано ранее, целью задачи о назначениях является нахождением минимальной суммы всех производительностей. В таком случае самым эффективным из алгоритмов является Венгерский алгоритм [6].

Однако не всегда в задаче главной целью является минимизация именно суммарной производительности. В некоторый случаях важнее найти, например, такое назначение, при котором самая большая стоимость из всех сочетаний была минимальна. Для большей ясности приведём такой пример: необходимо, чтобы группа роботов выполнила задачу за минимальное время. Таким образом, главным критерием является не суммарное время исполнения каждого, а верхний предел (т.е. минимизация каждой стоимости ниже предела уже маловажна). На первый взгляд это может показаться не таким важным различием, однако на самом деле отличия весьма значительны и Венгерский алгоритм в данном случае не даст оптимальный результат, что будет продемонстрировано далее. В таком случае хорошим способом решить данную задачу является применение муравьиного алгоритма, эффективность которого будет показана экспериментально в дальнейшей работе.

2. Применение муравьиного алгоритма в задаче о назначениях

2.1. Формулировка задачи

Как ранее было указано, для данной вариации задачи о назначениях главным критерием будет минимизация «верхнего предела» стоимости выполнения задач в найденных сочетаниях. Для простоты в качестве задач можно привести перемещение агента в указанную точку.

Задача: за наименьшее время установить произвольно расположенные агенты в заданные точки на заданном поле.

Исходные данные: местоположение агентов, целевых точек, а также препятствий.

Ожидаемый результат: матрица, содержащая оптимальное сочетание агентов и задач для решения задачи.

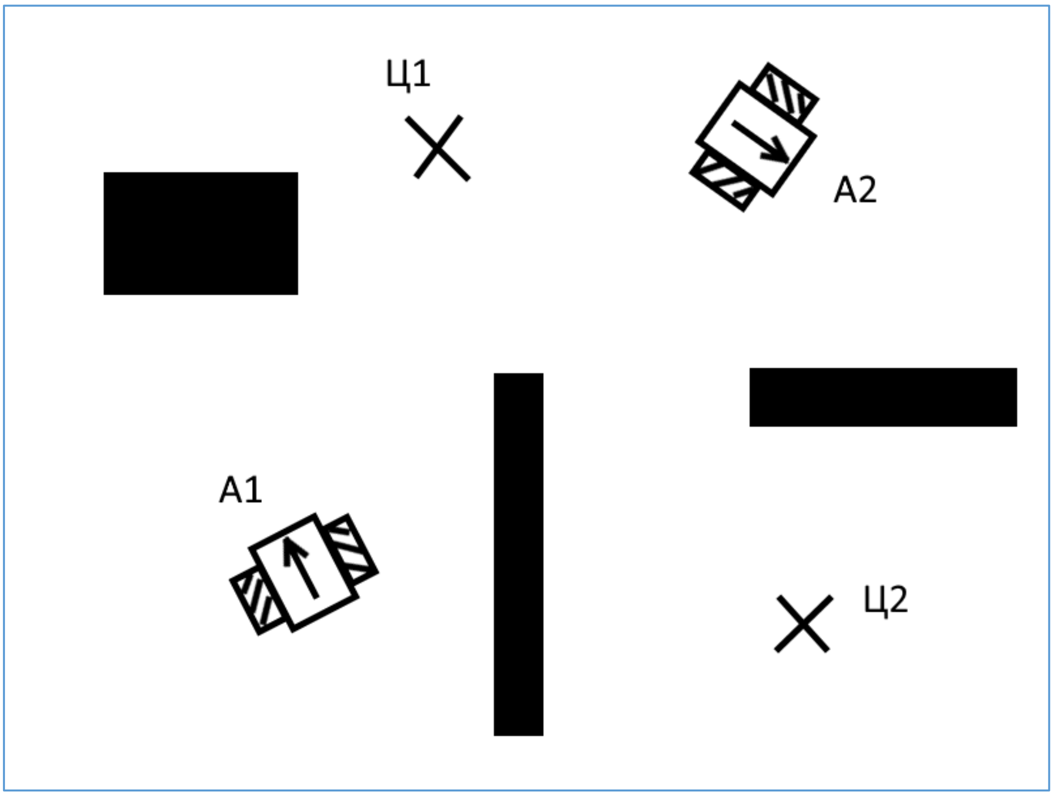


Рисунок 2. Схематическое изображение поставленной задачи

2.2. Общее описание программы для решения задачи

Для решения поставленной задачи была разработана программа на языке Python (Приложение 1). Её идея такова: симулировать централизованную систему, управляющий центр которой задаёт траектории и распределяет цели между агентами, реализуя тактический уровень управления, а сами агенты реализуют движение к заданной точке, т.е. имеют исполнительный уровень управления.

Траектории задаются следующим образом: при помощи алгоритма Дейкстры [7] просчитываются кратчайшие маршруты для всех вариантов сочетаний агент-задача. В результате этого будут высчитаны маршруты до всех целевых точек для каждого из агентов

Необходимо оптимизировать эти сочетания по времени, однако если считать скорость всех агентов постоянной, то время преодоления маршрута будет пропорционально длинам маршрутов. Таким образом длины найденных маршрутов и будут использованы в качестве стоимостей задач.

После этого при помощи муравьиного алгоритма происходит поиск оптимального распределения задач. Данные назначения передаются самим агентам, после чего они начинают движение к поставленным целям.

Поскольку во время движения возможно возникновение непредвиденных замедлений, например, для предотвращения столкновений агентов, тактический уровень управления продолжает свою работу постоянно или с некоторым периодом для коррекции назначений в реальном времени до тех пор, пока все агенты не закончат движение.

Структура такой системы изображена на Рисунке 3

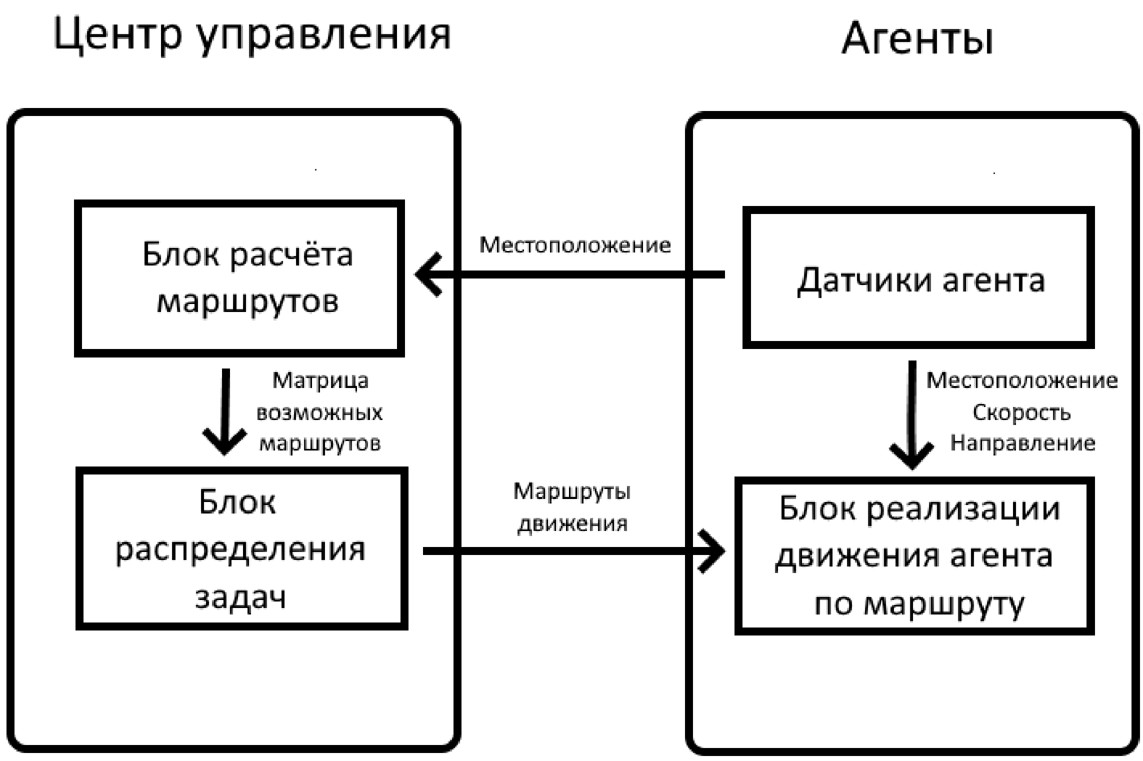


Рисунок 3. Общая структура описанной системы

Поскольку в разрабатываемой программе параллельное действие будет только симулироваться, возможно производить данные действия последовательно. Время вычислений траекторий будет фиксироваться, после чего вычисляется движение роботов за данное зафиксированное время, словно перемещение одновременно с вычислениями.

Блок-схема составленного алгоритма представлена на Рисунке 4.

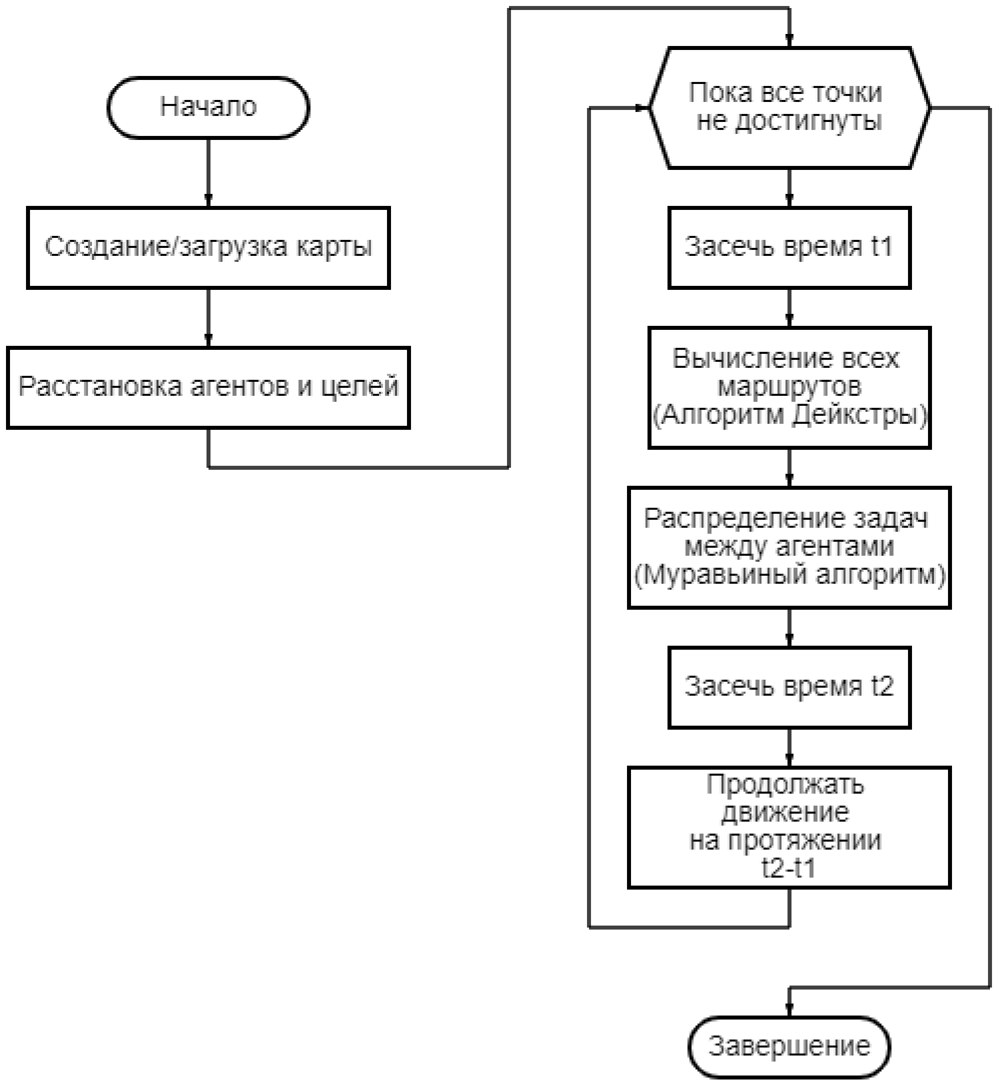


Рисунок 4. Блок-схема алгоритма разработанной программы

После запуска созданной программы появляется окно, в котором схематически изображается перемещение агентов (кружки) на ранее нарисованной карте. Оранжевым цветом изображены агенты, находящиеся в процессе следования к целевой точке, чёрные – закончившие движение (Рисунок 5 и 6). Во время работы в консоль производится отчёт по работе: номер цикла вычислений траектории, его длина и время, за которое алгоритм закончил вычисление (Рисунок 7). Для показа анимации использована открытая библиотека Pygame. Для сравнения муравьиного алгоритма с другими способами решения задачи о назначениях, в программе реализованы алгоритм полного перебора, жадный алгоритм (т.н. метод аукциона) и венгерский метод.

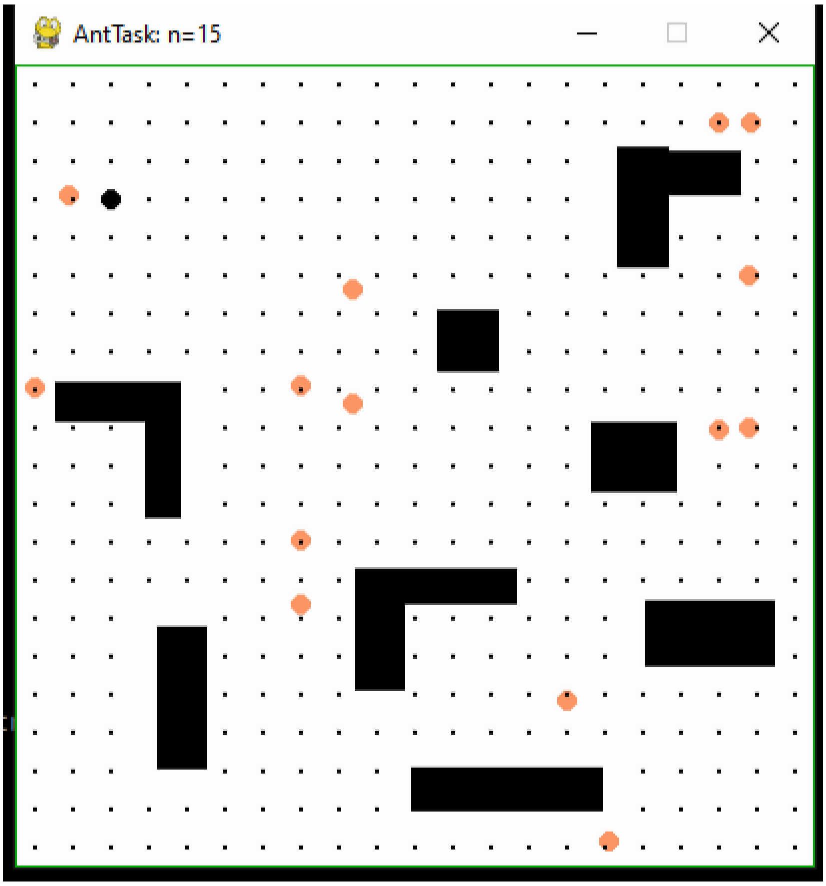


Рисунок 5. Окно симуляции во после запуска программы

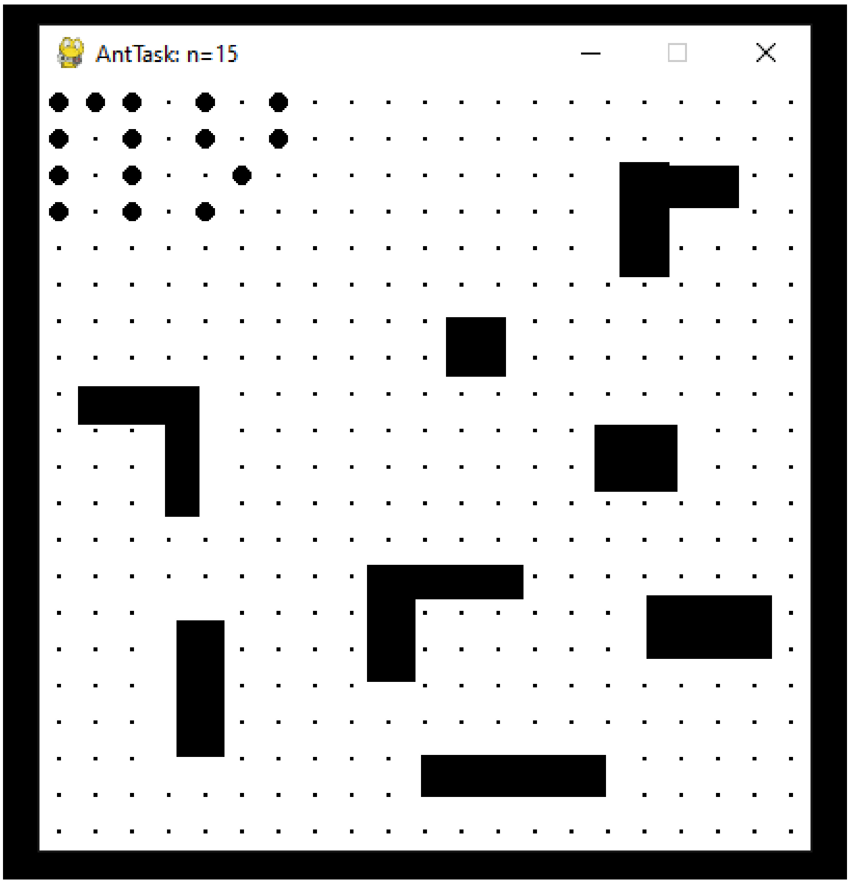
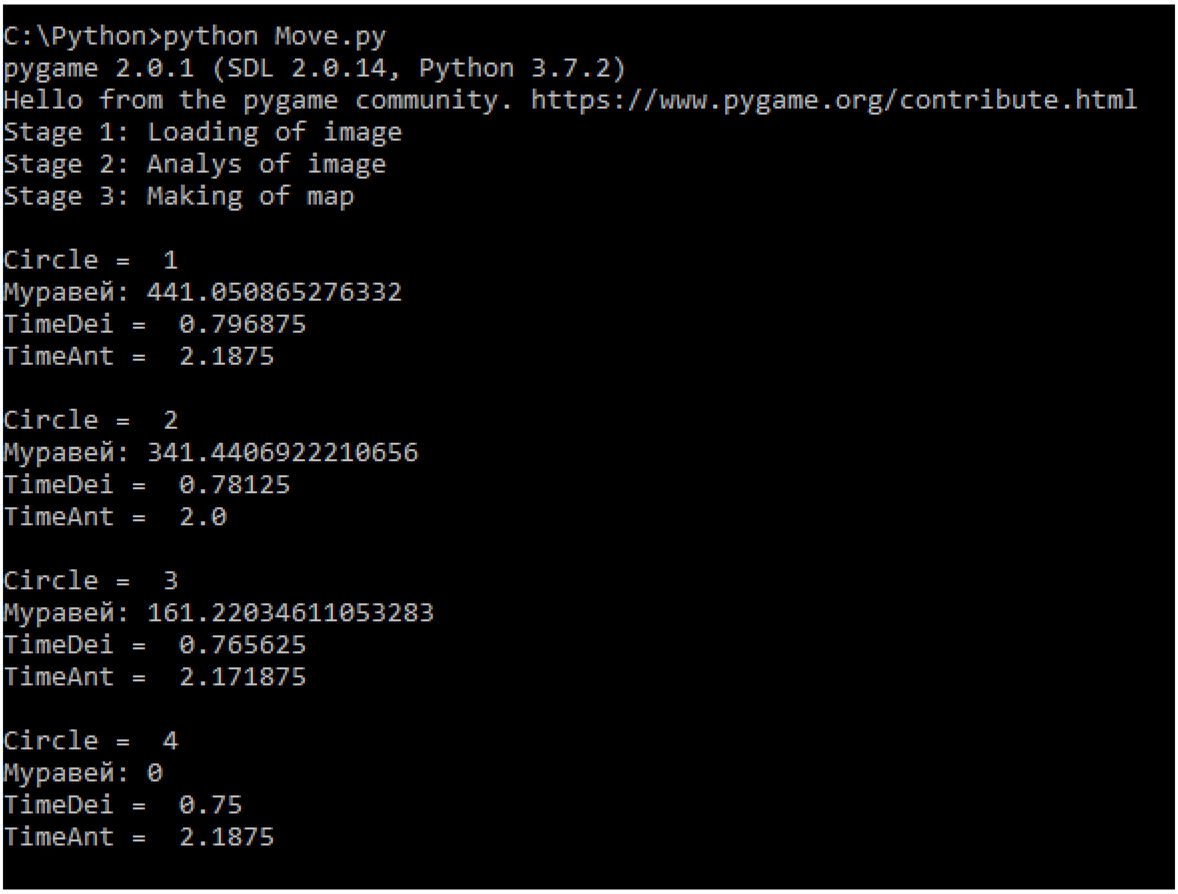


Рисунок 6. Окно симуляции после завершения работы программы



```
C:\Python>python Move.py
pygame 2.0.1 (SDL 2.0.14, Python 3.7.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
Stage 1: Loading of image
Stage 2: Analys of image
Stage 3: Making of map

Circle =  1
Муравей: 441.050865276332
TimeDei =  0.796875
TimeAnt =  2.1875

Circle =  2
Муравей: 341.4406922210656
TimeDei =  0.78125
TimeAnt =  2.0

Circle =  3
Муравей: 161.22034611053283
TimeDei =  0.765625
TimeAnt =  2.171875

Circle =  4
Муравей: 0
TimeDei =  0.75
TimeAnt =  2.1875
```

Рисунок 7. Консоль программы после окончания работы

2.3. Основная структура программы

Основные функции программы:

Функция OnTrail(agent,trail1,stage,S) – функция расчёта движения агентов

Функция DisplayAll() – функция вывода анимации в оконном режиме

Функция image_to_points(Blocks_picture) – функция считывания нарисованной карты.

Функция do_point_map_cage_im() – функция обработки карты для дискретизации и дальнейшей обработки при помощи алгоритма Дейкстры.

Функция DeikstrOnePoint(Now,NotBlock,MinRouteL,MinR) – функция используемая в реализации алгоритма Дейкстры.

Функция Deikstra(agent,end1) – функция вычисления маршрутов при помощи алгоритма Дейкстры.

Функция PutPher(Matches,l,deltaPher) – функция Муравьиного алгоритма. Расположение феромона на соответствующем ребре.

Функция UpdPher(deltaPher,pherTrail) – функция Муравьиного алгоритма. Обновление феромонов.

Функция probability(agent,goal,pherTrail) – функция Муравьиного алгоритма. Вычисление вероятности выбора ребра в зависимости от его стоимости и количества феромонов.

Функция ChoiseGoal(agent,NotTabu,pherTrail) – функция Муравьиного алгоритма. Создание единичого сочитания агент-задача.

Функция FindMatch(now,pherTrail) – функция Муравьиного алгоритма. Создание набора назначений агент-задача.

Функция Length(Trail) – функция нахождения «верхнего предела» стоимости текущих назначений.

Функция AntAdmin() – функция оптимизации назначения задач при помощи муравьиного алгоритма.

Функция makeExel(StepList,BestList) – функция сохранения данных в фотмате таблицы Excel.

<u>Дополнительные функции для проведения сравнительного анализа:</u>

Функция Greed() – реализация назначения при помощи жадного алгоритма

Функция GrossOne(step,NotBlock,Matches,min_l,min_Match) – элемент для реализации алгоритма перебора.

Функция Gross() – реализация назначения при помощи алгоритма полного перебора.

Функция Hungarian() – реализация назначения при помощи Венгерского метода.

2.4. Проведение экспериментального исследования алгоритма

2.4.1 Условия проводимого исследования

Настраиваемые параметры алгоритма в программе:

1. n_agent, количество агентов и целевых точек на карте
2. Tend, время окончания симуляции работы алгоритма
3. Alpha, коэффициент влияния феромона на вероятность выбора ребра.
4. Beta, коэффициент влияния длины ребра на вероятность его выбора.
5. Rho, коэффициент испарения феромонов на рёбрах графа.
6. Q, коэффициент увеличения феромонов.
7. pherMin, минимальное количество феромона на рёбрах графа.
8. pherMax максимальное количество феромона на рёбрах графа.
9. FPS – количество кадров в секунду при визуализации
10. speed – скорость агентов
11. near_range – шаг дискретизации карты
12. Rob_gabarite – габариты робота (влияет на допускаемое расстояние агента от препятствий)

Благодаря работе А.А. Кажарова и В.М. Курейчик [5], можно использовать найденные в ней оптимальные параметры Alpha, Beta и Rho:

Alpha = 1;

Beta = 2;

Rho = 0,2.

Таким образом для муравьиного алгоритма достаточно подобрать Q и установить пределы pherMin и pherMax. Таким образом были приняты следующие значения:

Q=200;

pherMin=1;

pherMax=5.

При данных значениях уровень феромонов достаточно увеличивается, чтобы выделить более оптимальные пути, однако не настолько, чтобы увести алгоритм в локальный минимум, а чтоб допустить дальнейший перебор неисследованных сочетаний.

Остальные параметры примем:

speed=30;

near_range=30;

Rob_gabarite=5.

2.4.2. Процесс нахождения оптимальных сочетаний

После вычисления всех маршрутов при помощи муравьиного алгоритма происходит назначение агентов. Стоит сказать, что поскольку алгоритм является вероятностным, то не имеет чёткого момента завершения обработки. Процесс нахождения стремится к оптимальному, однако нахождение глобально оптимального маршрута не гарантировано. В данном случае алгоритм был ограничен 400-ми циклами. Дальнейшее вычисление даёт слишком мало пользы в сравнении с временем вычислений, на который оно тратится. При уменьшении количества циклов пострадает получаемое качество оптимизации.

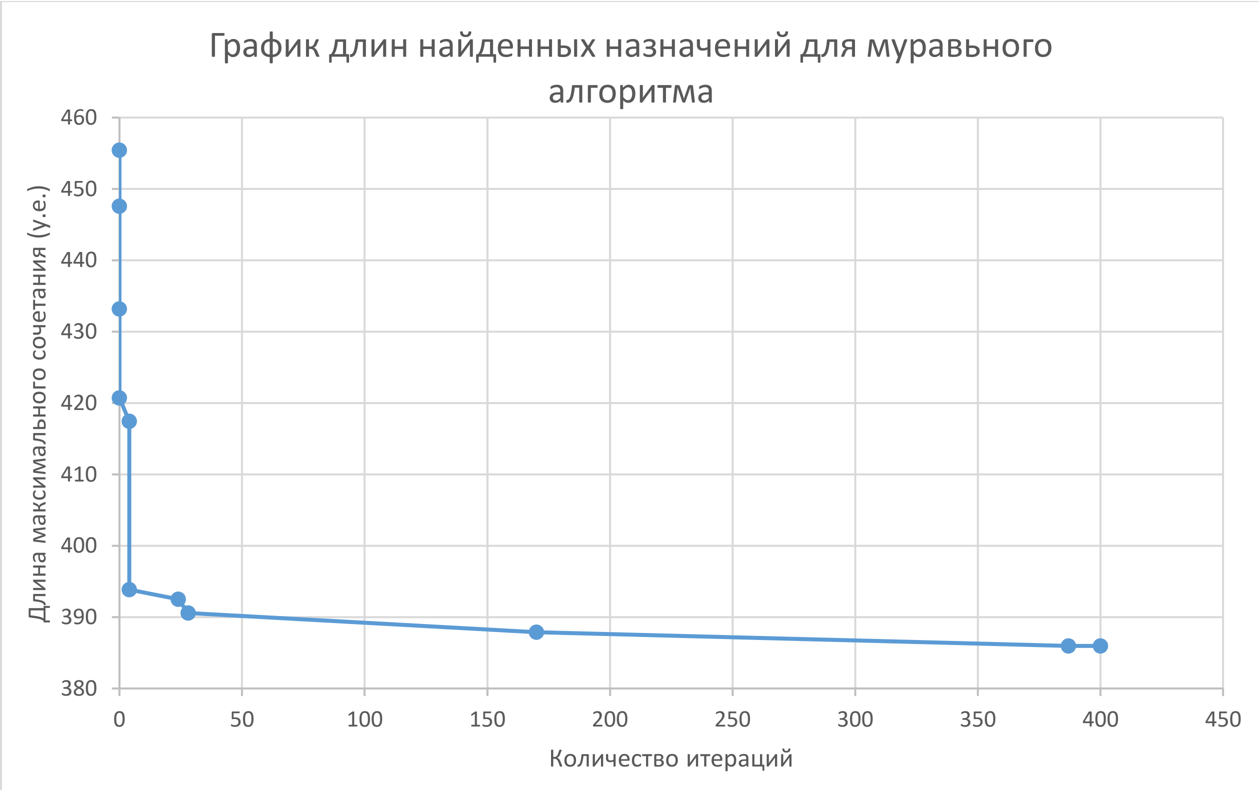


Рисунок 8. График процесса оптимизации назначений муравьиного алгоритма (n=15)

Во время перемещений постоянно происходит изменение назначений, что можно увидеть на Рисунках 9-11. Таким образом система адаптируется к изменению условий

```
Circle =  1
Муравей: 422.050865276332
Муравей:
0  :  8
1  :  14
2  :  1
3  :  6
4  :  2
5  :  5
6  :  3
7  :  7
8  :  13
9  :  4
10  :  12
11  :  11
12  :  9
13  :  0
14  :  10
TimeDei =  0.828125
TimeAnt =  2.046875
```

```
Circle =  2
Муравей: 314.5706345359768
Муравей:
0  :  5
1  :  14
2  :  1
3  :  3
4  :  12
5  :  6
6  :  0
7  :  8
8  :  4
9  :  11
10  :  2
11  :  7
12  :  9
13  :  10
14  :  13
TimeDei =  0.921875
TimeAnt =  2.0625
```

```
Circle =  3
Муравей: 107.48023074035522
Муравей:
0  :  5
1  :  14
2  :  1
3  :  3
4  :  12
5  :  6
6  :  2
7  :  8
8  :  4
9  :  7
10  :  9
11  :  11
12  :  0
13  :  10
14  :  13
TimeDei =  0.984375
TimeAnt =  2.25
```

Рисунки 9,10,11. Вывод найденных назначений в консоль после 1-ого, 2-ого и 3-его циклов вычислений соответственно

2.4.3. Сравнительное изучение характеристик алгоритма

Как было сказано, для сравнения эффективности алгоритма в программе реализованы также жадный алгоритм, алгоритм перебора и Венгерский метод решения задачи о назначениях. Изменяя количество агентов, исследуем скорость и точность алгоритма.

Ниже построены графики времени работы и найденных значений для каждого из алгоритмов при различном количестве агентов.

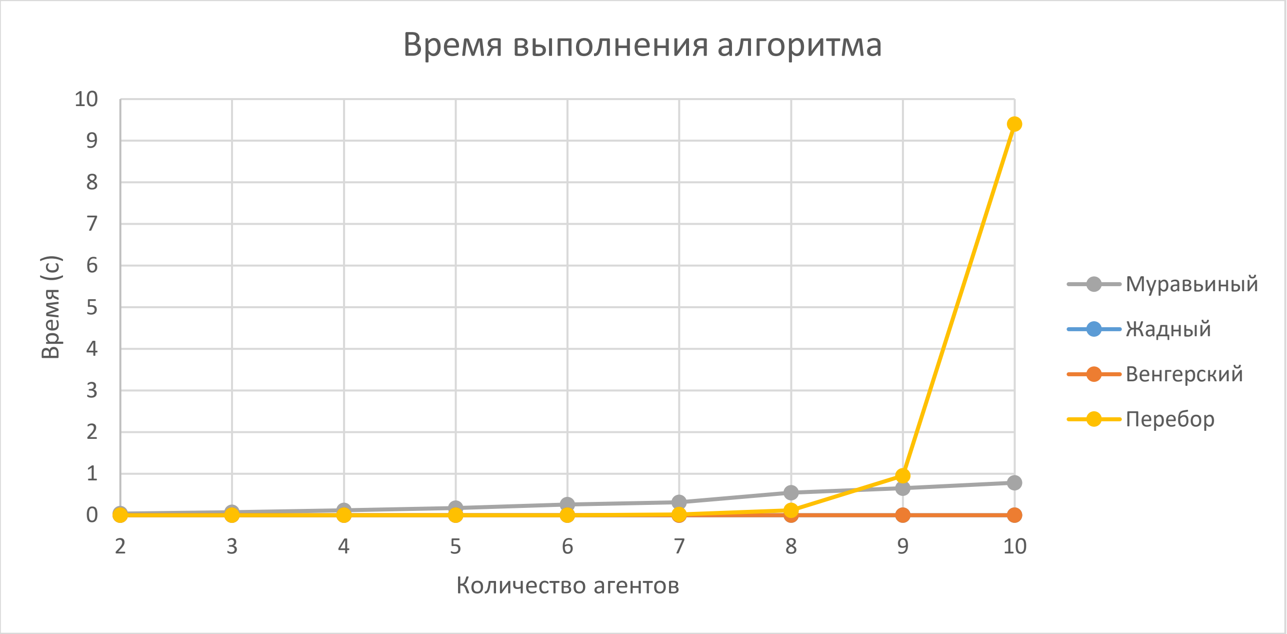


Рисунок 12. Графики времени выполнения алгоритмов при различном количестве агентов

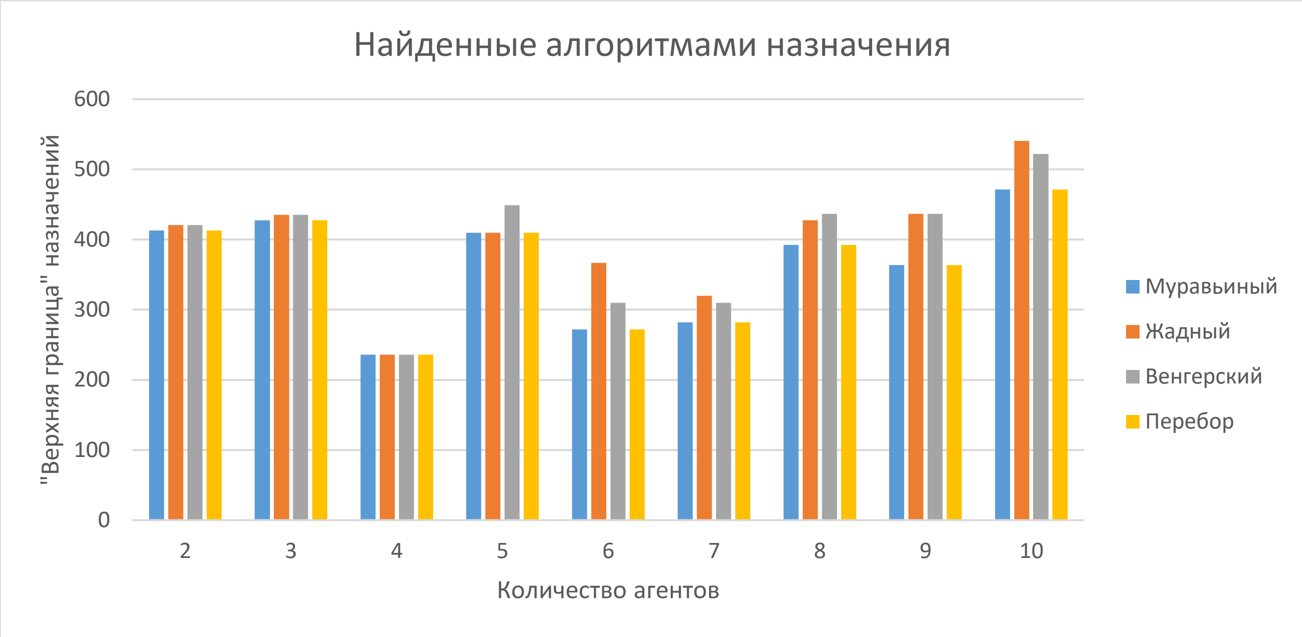


Рисунок 13. Графики найденных минимальных назначений для алгоритмов при различном количестве агентов

Как можно увидеть из графиков, муравьиный алгоритм приближается к глобально оптимальному назначению, а для малого количества (до 10 агентов) практически всегда его достигает. Таким образом, муравьиный алгоритм в данном параметре соизмерим с методом полного перебора. Однако у метода перебора есть главная проблема – его скорость. При увеличении количества агентов, время работы резко возрастает, поскольку сложность алгоритма равна (N!).

Жадный и венгерский алгоритм почти всегда не дают самого оптимального решения. Эта разница с оптимальным назначением будет сильнее проявляться при большем разбросе точек по карте. Их значительное преимущество – скорость. Но для данной задачи не настолько малое временя вычисления не является важным, а решающее значение играет точность распределения задач.

2.4.4. Оценка скорости работы созданной системы

Муравьиный алгоритм и алгоритм Дейкстры, хоть и обладают достаточно высокой скоростью, однако с увеличением количества обрабатываемых точек, время вычисления также заметно увеличивается. Таким образом при некотором количестве агентов система уже не будет иметь достаточную скорость, чтоб в процессе корректировать назначение маршрутов (Рисунок 14)

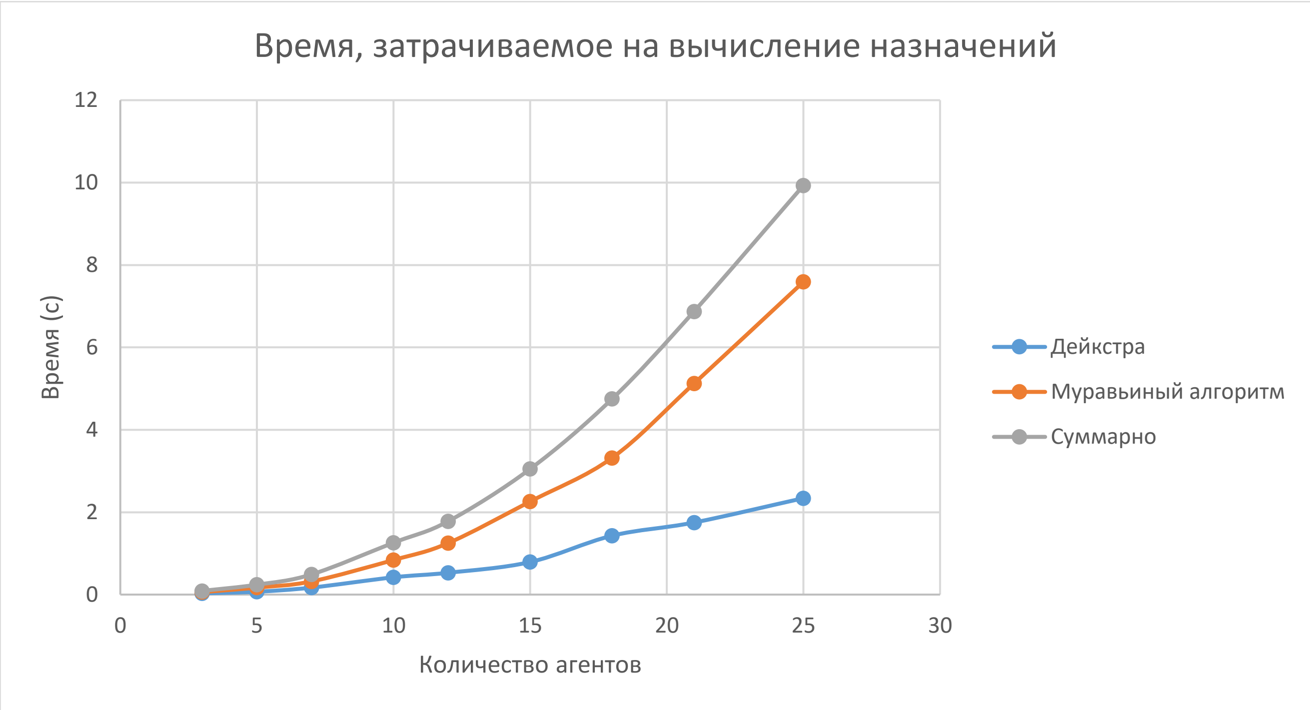


Рисунок 14. Графики времени вычислений оптимальных назначений в созданной системе при различном количестве агентов

Примем, что достаточная скорость обработки обеспечивается в том случае, когда на протяжении работы успевает произойти от 3 и более корректировок назначений. В данном случае максимальное агентов может быть равно 15.

В зависимости от скорости роботов и масштаба карты, а также производительности управляющего компьютера, ограничение на время обработки может накладываться различным образом и, следовательно, при различных ситуациях может быть допустимо различное количество агентов.

Для обработки большего количества целей возможно производить оценку стоимости путей до целей не при помощи Дейкстры, а евклидового расстояния. Таким образом количество запусков алгоритма снизится с N^2 до N, однако при этом значительно уменьшится качество оценки стоимости маршрута, поскольку в таком случае она происходит без учёта препятствий, напрямую. Также существует вариант некоторого ускорения работы за счёт использования Astar вместо алгоритма Дейксты или увеличения шага дискретизации.

Другим вариантом уменьшения длительности вычислений является уменьшение количества циклов (Tend) для муравьиного алгоритма, однако это также скажется на точности нахождения оптимального сочетания.

Заключение

В ходе работы разработана система, реализующая решение задачи о назначениях с нестандартным условием. Созданное программное обеспечение производит и визуализирует симуляцию использования решения задачи о назначениях в ходе движения мобильных роботов. Проведены исследования для системы с различными параметрами.

Разработанная система в ходе испытаний показала достаточную эффективность работы в сравнении с другими алгоритмами. Таким образом использование муравьиных алгоритмов для решения задачи о назначениях, подобных поставленной, является целесообразным.

Список источников

1. Marco Dorigo. Optimization, learning and natural algorithms. 1992.
2. Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. 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. А.А. Кажаров, В.М. Курейчик. Решение задачи о назначениях на основе муравьиных алгоритмов
6. В.П.Никулова, Е.В.Фирсова, Венгерский метод решения задачи о назначениях
7. Электронный источник. Объяснение алгоритма Дейкстры. (https://habr.com/ru/post/111361/)
8. Сообщество исследователей муравьиных алгоритмов в социальной сети Вконтакте (https://vk.com/ant_colony_optimization)

Приложение 1

Код разработанной программы.

```
import pygame
from pygame.locals import *
import sys
import random as rnd
import math
from PIL import Image, ImageDraw,ImageFont
import numpy as np
import time
import openpyxl
from munkres import Munkres

def OnTrail(agent,trail1,stage,S):
    next_point=trail1[stage]
    dist_among=math.hypot(agent_x[agent]-point_x[next_point],agent_y[agent]-
point_y[next_point])
    if(dist_among<S):
        agent_x[agent]=point_x[next_point]
        agent_y[agent]=point_y[next_point]
        if(stage==len(trail1)-1):
            endflag=1
            return endflag,stage
        else:
            return OnTrail(agent,trail1,stage+1,S-dist_among)
    else:
        alpha=math.atan2(point_y[next_point]-agent_y[agent],point_x[next_point]-
agent_x[agent])
        agent_x[agent]+=S*math.cos(alpha)
        agent_y[agent]+=S*math.sin(alpha)
        endflag=0
        return endflag,stage

def DisplayAll():
    for i in pygame.event.get():
            if i.type == pygame.QUIT:
                sys.exit()
    sc.fill(WHITE)
    sc.blit(bg,(0,0))
    for i in range(n_agent):
        if(endflag[i]==0):
            pygame.draw.circle(sc, ORANGE,(agent_x[i], agent_y[i]), Rob_gabarite)
        else:
```

```
        pygame.draw.circle(sc, BLACK,(agent_x[i], agent_y[i]), Rob_gabarite)
    for i in range(nCity):
      pygame.draw.circle(sc, BLACK,(point_x[i], point_y[i]), 1)
    pygame.display.update()

def image_to_points(Blocks_picture):
  img = Image.open(Blocks_picture)
  img = img.convert('1')
  first_arr = np.asarray(img, dtype='int')
  x_shape=first_arr.shape[1]
  y_shape=first_arr.shape[0]
  arr=[[0 for j in range(y_shape)] for i in range(x_shape)]
  arr2=[[0 for j in range(y_shape)] for i in range(x_shape)]
  for x_arr in range(x_shape):
    for y_arr in range(y_shape):
      arr[x_arr][y_arr]=(math.ceil((first_arr[y_arr][x_arr]/255)))*(-1)+1
      if (arr[x_arr][y_arr]==1):
        for x in range(Rob_gabarite+1):
          for y in range(x):
            if(x_arr+x-1>0 and y_arr+y-1<y_shape-1):
              arr2[x_arr+x-1][y_arr+y-1]=1
            if(x_arr+x-1>0 and y_arr-y-1>0):
              arr2[x_arr+x-1][y_arr-y-1]=1
            if(x_arr-x-1<x_shape-1 and y_arr-y-1>0):
              arr2[x_arr-x-1][y_arr-y-1]=1
            if(x_arr-x-1<x_shape-1 and y_arr+y-1<y_shape-1):
              arr2[x_arr-x-1][y_arr+y-1]=1
  return x_shape,y_shape,arr,arr2

def do_point_map_cage_im(): #Создания сетки на плоскости
  space=math.floor((((near_range**2)/2)**0.5)-2)
  x_len=math.floor(window_x/space)
  y_len=math.floor(window_y/space)
  nCity=x_len*y_len
  dist_among = [[0 for j in range(nCity)] for i in range(nCity)]
  schet=0
  for now_y in range(y_len):
    for now_x in range(x_len):
      if(arr2[math.ceil(now_x*space+space/2)][math.ceil(now_y*space+space/2)]==0):
        thisnear=[]
        point_x.append(math.ceil(now_x*space+space/2))
        point_y.append(math.ceil(now_y*space+space/2))
        now=schet
        for other in range(now):
```

```
        if ((((point_x[other]-point_x[now])**2 + (point_y[other]-
point_y[now])**2)**0.5)<near_range):
          dist=((point_x[other]-point_x[now])**2 + (point_y[other]-
point_y[now])**2)**0.5
          proof_space=math.ceil(dist/5)
          proof=1
          for n in range(proof_space):
            tx=math.ceil(point_x[now]+(point_x[other]-point_x[now])/proof_space*n)
            ty=math.ceil(point_y[now]+(point_y[other]-point_y[now])/proof_space*n)
            if(arr2[tx][ty]):
              proof=0
              break
          if(proof==1):
            thisnear.append(other)
            point_near[other].append(now)
            dist_among[now][other]=dist
            dist_among[other][now]=dist
      point_near.append(thisnear)
      schet=schet+1
  nCity=schet
  return nCity,dist_among

def DeikstrOnePoint(Now,NotBlock,MinRouteL,MinR):
  for Look in (point_near[Now]):
    if(dist_among[Now][Look]+MinRouteL[Now]<MinRouteL[Look]):
      MinRouteL[Look]=dist_among[Now][Look]+MinRouteL[Now]
      flag=0
      for i in (NotBlock):
        if(i==Look):
          flag=1
          break
      if (flag==0):
        NotBlock.append(Look)
      MinR[Look]=MinR[Now].copy()
      MinR[Look].append(Look)
  NotBlock.remove(Now)

def Deikstra(agent,end1):
  nearestPoint=0
  nearestLength=math.hypot(agent_x[agent]-point_x[nearestPoint],agent_y[agent]-
point_y[nearestPoint])
  for i in range(nCity):
    NewTry=math.hypot(agent_x[agent]-point_x[i],agent_y[agent]-point_y[i])
    if(nearestLength>NewTry):
      nearestLength=NewTry
```

```
      nearestPoint=i
  start1=nearestPoint
  NotBlocked=[]
  NotBlocked.append(start1)
  MinRouteLen=[]
  for i in range(nCity):
    MinRouteLen.append(100000000)
  MinRoute=[]
  for j in range(nCity):
    OnePoint=[]
    MinRoute.append(OnePoint)
  MinRoute[start1].append(start1)
  MinRouteLen[start1]=0
  DeikstrOnePoint(start1,NotBlocked,MinRouteLen,MinRoute)
  while(1):
    if(len(NotBlocked)==0):
      break
    min=NotBlocked[0]
    for k in (NotBlocked):
      if(MinRouteLen[min]>MinRouteLen[k]):
        min=k
    DeikstrOnePoint(min,NotBlocked,MinRouteLen,MinRoute)
  if(MinRouteLen[end1]==100000000):
    print("No route!")
  return MinRoute[end1],MinRouteLen[end1]

def PutPher(Matches,l,deltaPher): #Распределение феромонов после движения муравьёв
  for i in range(len(Matches)):
    l1=l
    if(l1<1):
      l1=1
    deltaPher[i][Matches[i]]+=Q/l1

def UpdPher(deltaPher,pherTrail): # Обновление феромонов
  for i in range(n_agent):
    for j in range (n_agent):
      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
      deltaPher[i][j]=0
```

```
def probability(agent,goal,pherTrail): #Расчёт весов определённого отрезка пути для расчёта вероятности прохода муравья по нему.
 l1=lenTrail[agent][goal]
 if(l1<1):
  l1=1
 p=(pherTrail[agent][goal]**Alpha)*((1/l1)**Beta)
 return p

def ChoiseGoal(agent,NotTabu,pherTrail): #Функция выбора следующей точки для перехода
 WhereList=[]
 ProbList=[]
 for cit in NotTabu:
  WhereList.append(cit)
  ProbList.append(probability(agent,cit,pherTrail))
 return rnd.choices(WhereList, weights=ProbList)[0]

def FindMatch(now,pherTrail): #Функция перехода муравья из одной точки в другую
 NotTabu=[]
 FreeAgent=[]
 NewMatches = [-1 for i in range(n_agent)]
 nextA=-1
 for i in range (n_agent):
  FreeAgent.append(i)
  NotTabu.append(i)
 while len(FreeAgent)>0:
  if(nextA==-1):
   nextA=now
  else:
   nextA=FreeAgent[rnd.randint(0,len(FreeAgent)-1)]
  Match=ChoiseGoal(nextA,NotTabu,pherTrail)
  NewMatches[nextA]=Match
  NotTabu.remove(Match)
  FreeAgent.remove(nextA)
 return NewMatches

def Length(Trail): #Расчёт верхней границы стоимости
 l=-1
 for i in range(len(Trail)):
  new_l=lenTrail[i][Trail[i]]
  if(new_l>l or l==-1):
   l=new_l
 return l
```

```
def AntAdmin():
  pherTrail = [[pherMin for j in range(n_agent)] for i in range(n_agent)]
  deltaPher = [[0 for j in range(n_agent)] for i in range(n_agent)]
  bestLen=-1
  bestTrail=[]
  BestList=[]
  StepList=[]
  bestTime=0
  for step in range(Tend):
    for Ant in range(n_agent):
      way=FindMatch(Ant,pherTrail)
      l=Length(way)
      if(bestLen==-1 or l<bestLen):
        bestLen=l
        bestTrail=way
        bestTime=step
        BestList.append(bestLen)
        StepList.append(step)
        #print("На шаге",step, " найден путь длиной",bestLen)
      PutPher(way,l,deltaPher)
    UpdPher(deltaPher,pherTrail)
  makeExel(StepList,BestList)
  return bestTrail,bestLen

def makeExel(StepList,BestList):
  wb = openpyxl.Workbook()
  wb.create_sheet(title = 'Ant', index = 0)
  sheet = wb['Ant']
  NowCell = sheet.cell(row = 1, column = 1)
  NowCell.value = "Step"
  NowCell = sheet.cell(row = 1, column = 2)
  NowCell.value = "FoundLength"
  for i in range (len(BestList)):
    NowCell = sheet.cell(row = i+2, column = 1)
    NowCell.value = str(StepList[i])
    NowCell = sheet.cell(row = i+2, column = 2)
    NowCell.value = str(BestList[i])
  wb.save('AntMatches.xlsx')

def Greed():
  NotBlock=[]
  Matches=[-1 for j in range(n_agent)]
  for i in range(n_agent):
    NotBlock.append(i)
  for i in range(n_agent):
```

```
    flag=0
    l_min=-1
    for j in NotBlock:
      if(flag==0 or lenTrail[i][j]<lenTrail[i][n_min]):
        n_min=j
        flag=1
    Matches[i]=n_min
    NotBlock.remove(n_min)
  print("Жадный: ",Length(Matches))
  return Matches

def GrossOne(step,NotBlock,Matches):
  global GrossMin_Match,GrossMin_l
  if(len(NotBlock)==0):
      if (Length(Matches)<GrossMin_l):
        GrossMin_Match=Matches
        GrossMin_l=Length(Matches)
  else:
    for i in NotBlock:
      m1=Matches.copy()
      m1.append(i)
      n1=NotBlock.copy()
      n1.remove(i)
      GrossOne(step,n1,m1)

def Gross():
  sys.setrecursionlimit(1000000)
  global GrossMin_Match,GrossMin_l
  GrossNotBlock=[]
  GrossMatches=[]
  GrossMin_Match=[]
  GrossMin_l=100000
  for i in range(n_agent):
    GrossNotBlock.append(i)
  GrossOne(0,GrossNotBlock,GrossMatches)
  print("Перебор: ",Length(GrossMin_Match))

def Hungarian():
  m = Munkres()
  indexes = m.compute(lenTrail)
  min_l = 0
  for row, column in indexes:
    value = lenTrail[row][column]
    if(min_l<value):
      min_l = value
```

```
    #print(f'({row}, {column}) -> {value}')
  print(f'Венгерский: {min_l}')


FPS = 30
delta_time=1/FPS

WHITE = (255, 255, 255)
ORANGE = (255, 150, 100)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
BLUE = (0,0, 255)
GREEN = (0, 170, 0)

n_agent=18
speed=30
near_range=30
Rob_gabarite=5

point_x=[]
point_y=[]
point_near=[]
agent_x=[]
agent_y=[]
stage=[]
trail=[]
lenTrail=[]
endflag=[]

Tend=400 #Время симуляции
Alpha=1 #Коэффициент альфа (порядка значимости феромона)
Beta=2 #Коэффициент бэта (порядка значимости длины пути)
Rho=0.2 #Коэффициент испарения феромона
Q=200 #Коэффициент увеличения феромона
pherMin=1 #Минимальное количество феромона на рёбрах графа
pherMax=5 #Максимальное количество феромона на рёбрах графа

#goals=[0,1,2,4,6,21,23,25,27,42,44,47,59,61,63]
goals=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
29,30,31,32,33,34,35,36,37,38,39,40]

print("Stage 1: Loading of image")
adress="C:\Python\space.png"
bg = pygame.image.load(adress)
print("Stage 2: Analysis of image")
```

```
window_x,window_y,arr,arr2=image_to_points(adress)
print("Stage 3: Making of map")
nCity,dist_among=do_point_map_cage_im()

for i in range(n_agent):
   while(1):
      new_x=rnd.randint(1,window_x-2)
      new_y=rnd.randint(1,window_y-2)
      if(arr2[math.ceil(new_x)][math.ceil(new_y)]==0):
         break
   agent_x.append(new_x)
   agent_y.append(new_y)
   stage.append(0)
   endflag.append(0)
   trail.append([])
   lenTrail.append([])

clock = pygame.time.Clock()
sc = pygame.display.set_mode((window_x, window_y))
name="AntTask: n="+str(n_agent)
pygame.display.set_caption(name)
t1=time.process_time()
time_end=0

for i in range(n_agent):
   for j in range(n_agent):
      t,lt=Deikstra(i,goals[j])
      trail[i].append(t)
      lenTrail[i].append(lt)

t2=time.process_time()
time_dif=t2-t1
time_end=time_dif
if (time_dif<1):
  time_dif=2
circle=1
while 1:
   print("\nCircle = ",circle)
   circle+=1
   t1=time.process_time()
   for i in range(n_agent):
      for j in range(n_agent):
         trail[i][j],lenTrail[i][j]=Deikstra(i,goals[j])
      stage[i]=0
      endflag[i]=0
```

```
t2=time.process_time()
Matches,BestLenAnt=AntAdmin()
print("Муравей:", BestLenAnt)
t3=time.process_time()
#print("Муравей: ")
#for i in range(n_agent):
 #print(i," : ",Matches[i])
#GreedMatches=Greed()
t4=time.process_time()
#Gross()
t5=time.process_time()
#Hungarian()
t6=time.process_time()
time_dif=t3-t1
now_time=0
time_end+=time_dif
if (time_dif<1.5):
 time_dif=1.5
print("TimeDei = ",t2-t1)
print("TimeAnt = ",t3-t2)
#print("TimeGreed = ",t4-t3)
#print("TimeGross = ",t5-t4)
#print("TimeHung = ",t6-t5)
while now_time<time_dif:
   for i in range(n_agent):
      if(endflag[i]==0):
         S_1=speed*delta_time
         endflag[i],stage[i]=OnTrail(i,trail[i][Matches[i]],stage[i],S_1)
   DisplayAll()
   pygame.draw.rect(sc,GREEN,(0, 0, window_x, window_x), 2)
   pygame.display.update()
   clock.tick(FPS)
   now_time+=delta_time
pygame.draw.rect(sc,RED,(0, 0, window_x, window_x), 2)
pygame.display.update()
```