Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №6

по курсу «Анализ алгоритмов»

Муравьиный алгоритм

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Оглавление

Введение

В последние два десятилетия при оптимизации сложных систем исследователи все чаще применяют природные механизмы поиска наилучших решений. Это механизмы обеспечивают эффективную адаптацию флоры и фауны к окружающей среде на протяжении миллионов лет. Сегодня интенсивно разрабатывается научное направление Natural Computing — «Природные вычисления», объединяющее методы с природными механизмами принятия решений, а именно:

1. Genetic Algorithms — генетические алгоритмы;
2. Evolution Programming — эволюционное программирование;
3. Neural Network Computing — нейросетевые вычисления;
4. DNA Computing — ДНК-вычисления;
5. Cellular Automata — клеточные автоматы;
6. Ant Colony Algorithms — муравьиные алгоритмы.

Эти механизмы обеспечивают эффективную адаптацию флоры и фауны к окружающей среде на протяжении миллионов лет. Имитация самоорганизации муравьиной колонии составляет основу муравьиных алгоритмов оптимизации — нового перспективного метода природных вычислений. Колония муравьев может рассматриваться как многоагентная система, в которой каждый агент (муравей) функционирует автономно по очень простым правилам. В противовес почти примитивному поведению агентов, поведение всей системы получается на удивление разумным.

Муравьиные алгоритмы серьезно исследуются европейскими учеными с середины 90х годов. На сегодня уже получены хорошие результаты муравьиной оптимизации таких сложных комбинаторных задач, как: задачи коммивояжера, задачи оптимизации маршрутов грузовиков, задачи раскраски графа, квадратичной задачи о назначениях, оптимизации сетевых графиков, задачи календарного планирования и других. Особенно эффективны муравьиные алгоритмы при online-оптимизации процессов в распределенных нестационарных системах, например трафиков в телекоммуникационных сетях [1].

1. Аналитическая часть

В данном разделе будет описан муравьиный алгоритм на примере решения задачи коммивояжера.

1.1 Цель и задачи

Основной целью данной работы является изучение особенностей работы муравьиного алгоритма. Для того, чтобы добиться этой цели необходимо выполнить ряд задач:

1. примененить знания программирования для реализации муравьиного алгоритма;
2. получить практические навыки во время выполнения задания;
3. экспериментально подтвердить различия во временной эффективности работы муравьиного алгоритма при разных значениях коэффицента важности величины пути и коэффициента важности мощности феромона при помощи разработанного программного обеспечения на материале замеров процессорного времени;
4. описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1.2 Задача коммивояжера

Задача коммивояжера формулируется как задача поиска минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе с n вершинами. Вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояния или стоимости проезда. Эта задача является NP-трудной, и точный переборный алгоритм ее решения имеет факториальную сложность [2].

1.3 Описание муравьиного алгоритма

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений. Идея муравьиного алгоритма - моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь.

Моделирование поведения муравьёв связано с распределением феромона на тропе – ребре графа в задаче коммивояжёра. При этом вероятность включения ребра в маршрут

отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его рёбрах, следовательно, большее количество муравьёв будет включать его в синтез собственных маршрутов. Моделирование такого подхода, использующего только положительную обратную связь, приводит к преждевременной сходимости – большинство муравьёв двигается по локально оптимальному маршруту. Избежать этого можно, моделируя отрицательную обратную связь в виде испарения феромона.

С учётом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьёв при выборе пути.

1. Муравьи обладают «памятью». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещённых городов. Обозначим через $J_{i,k}$ список городов, которые необходимо посетить муравью k, находящемуся в городе i.

2. Муравьи обладают «зрением», которое определяет степень желания посетить город j, если муравей находится в городе i. Будем считать, что видимость обратно пропорциональна расстоянию между городами.

3. Муравьи обладают «обонянием», с помощью которого они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i,j) в момент времени t обозначим через $\tau_{ij}(t)$.

4. На основании предыдущих утверждений мы можем сформулировать вероятностно-пропорциональное правило, определяющее вероятность перехода k-ого муравья из города i в город j:

$$P_{ij,k}(t) = \begin{cases} \frac{(\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}}{\sum\limits_{l \in J(i,k)} (\tau_{il}(t))^{\alpha}(\eta_{il}(t))^{\beta}}, j \in J(i,k) \\ 0, j \notin J(i,k) \end{cases}, \tag{1.1}$$

   где $\tau_{ij}(t)$ – уровень феромона, $\eta_{ij}(t)$ – эвристическое расстояние, а α и β – константные параметры.

   Выбор города является вероятностным, в общую зону всех городов бросается случайное число, которое и определяет выбор муравья. При $\alpha = 0$ алгоритм вырождается до жадного алгоритма, по которому на каждом шаге будет выбираться ближайший город.

5. При прохождении ребра муравей оставляет на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть есть маршрут, пройденный муравьём k к моменту времени t, T – длина этого маршрута, $L_k(t)$ - цена текущего решения для k-ого муравья а Q – параметр, имеющий значение порядка цены оптимального решения. Тогда откладываемое количество феромона

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, (i,j) \in T_k(t) \\ 0, (i,j) \notin T_k(t) \end{cases}, \tag{1.2}$$

   а испаряемое количество феромона

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij,k}(t), \tag{1.3}$$

где m – количество муравьёв в колонии [3].

1.4 Вариации муравьиного алгоритма

Ниже приведены вариации муравьиного алгоритма.

1. **Элитарная муравьиная система**. Из общего числа муравьёв выделяются так называемые «элитные муравьи». По результатам каждой итерации алгоритма производится усиление лучших маршрутов путём прохода по данным маршрутам элитных муравьёв и, таким образом, увеличение количества феромона на данных маршрутах. В такой системе количество элитных муравьёв является дополнительным параметром, требующим определения. Так, для слишком большого числа элитных муравьёв алгоритм может «застрять» на локальных экстремумах.

2. **Max-Min муравьиная система**. Добавляются граничные условия на количество феромонов (τmax,τmin). Феромоны откладываются только на глобально лучших или лучших в итерации путях. Все рёбра инициализируются значением τmax.

3. **Ранговая муравьиная система(ASrank)**. Все решения ранжируются по степени их пригодности. Количество откладываемых феромонов для каждого решения взвешено так, что более подходящие решения получают больше феромонов, чем менее подходящие.

4. **Длительная ортогональная колония муравьёв (COAC)**. Механизм отложения феромонов COAC позволяет муравьям искать решения совместно и эффективно. Используя ортогональный метод, муравьи в выполнимой области могут исследовать их выбранные области быстро и эффективно, с расширенной способностью глобального поиска и точностью.

1.5 Вывод

В данном разделе были изучены различные вариации муравьиного алгоритма.

2. Конструкторская часть

В данном разделе в соответствии с описанием алгоритмов, приведенными в аналитической части работы, будет рассмотрена схема муравьиного алгоритма.

2.1 Разработка реализации

Основное действие происходит в функции run (рис. 2.1, 2.2, 2.3). На вход ей поступают следующие параметры:

1) time - количество поколений муравьев;

2) ants amount - количество муравьев;

3) elite_one - каждый i-ый(i = elite one) муравей будет элитным;

4) cities - количество городов;

5) eva(evaporation) - процент распыления феромона;

6) al(alpha) - параметр влияния расстояния;

7) be(beta) - параметр влияния феромона;

8) city_from - город появления муравьев;

9) Q, EQ - величина феромона обычного и элитного муравьев.

Они проверяются на корректность в функции run_check. Затем создаются и инициализируются матрицы дистанций, феромонов, массивы муравьев и начинается главный цикл функции - по поколениям. На этом этапе массив муравьев обновляет значения своих элементов до исходных значений (очищаются списки пройденных городов и пройденное расстояние). Затем начинается обход муравьев. Сначала добавляется родной город в список уже посещенных муравьем, чтобы он не мог 'случайно' заскочить домой, не обойдя все остальные города. Дальше начинается третий (по вложенности) цикл. Пока муравей не обошёл все города(can_move() - рис. 2.4), пусть двигается(make_move - 2.5). После того, как муравей пройдет все города (выход из цикла), вручную добавляется ему расстояние от текущего города (последнего, в котором он оказался) до родного (города рождения). Также добавляется его родной город в список посещенных. Это делается для того, чтобы если необходимо будет распечатать путь движения муравья, было видно, что он из дома вышел и домой же вернулся. Дальше 'распыляются' феромоны с помощью функции increase_pheromone. После чего сравнивается лучшее расстояние, что было (хранится в переменной best_way) с пройденным расстоянием муравья и с 0 (по умолчанию best_way равен -1). Если одно из этих условий выполняется, обновляется best_way и tabu_list (копию муравьиного массива пройденных городов). И последнее действие - уменьшение феромонов на карте. Условно считается, что каждый следующий муравей начинает движение,

когда вернулся прошлый, то есть одновременно на карте присутствует только 1 муравей. Тогда не выходя из цикла по муравьям, сразу после проверки на кратчайший путь уменьшаются феромоны по всей карте с помощью функции reduce_pheromone().

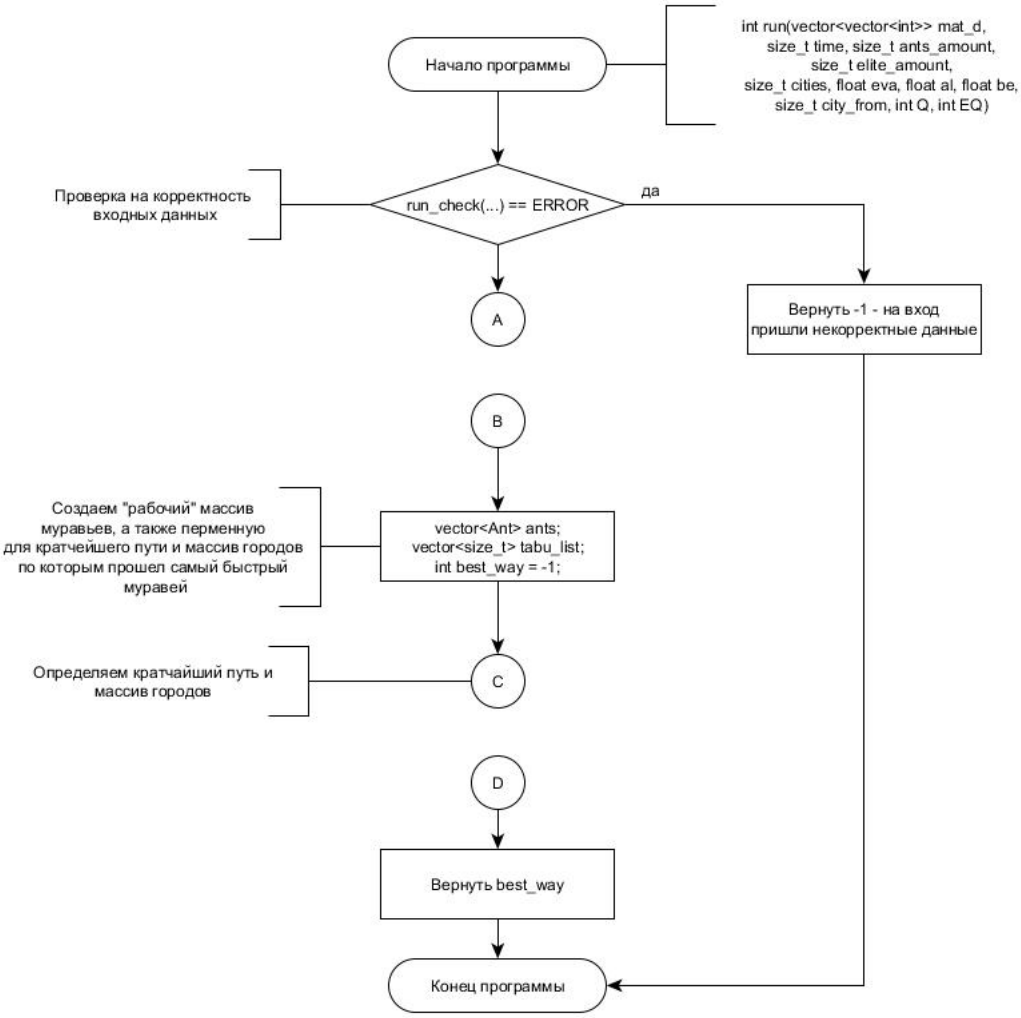

Рис. 2.1: Основная часть функции run.

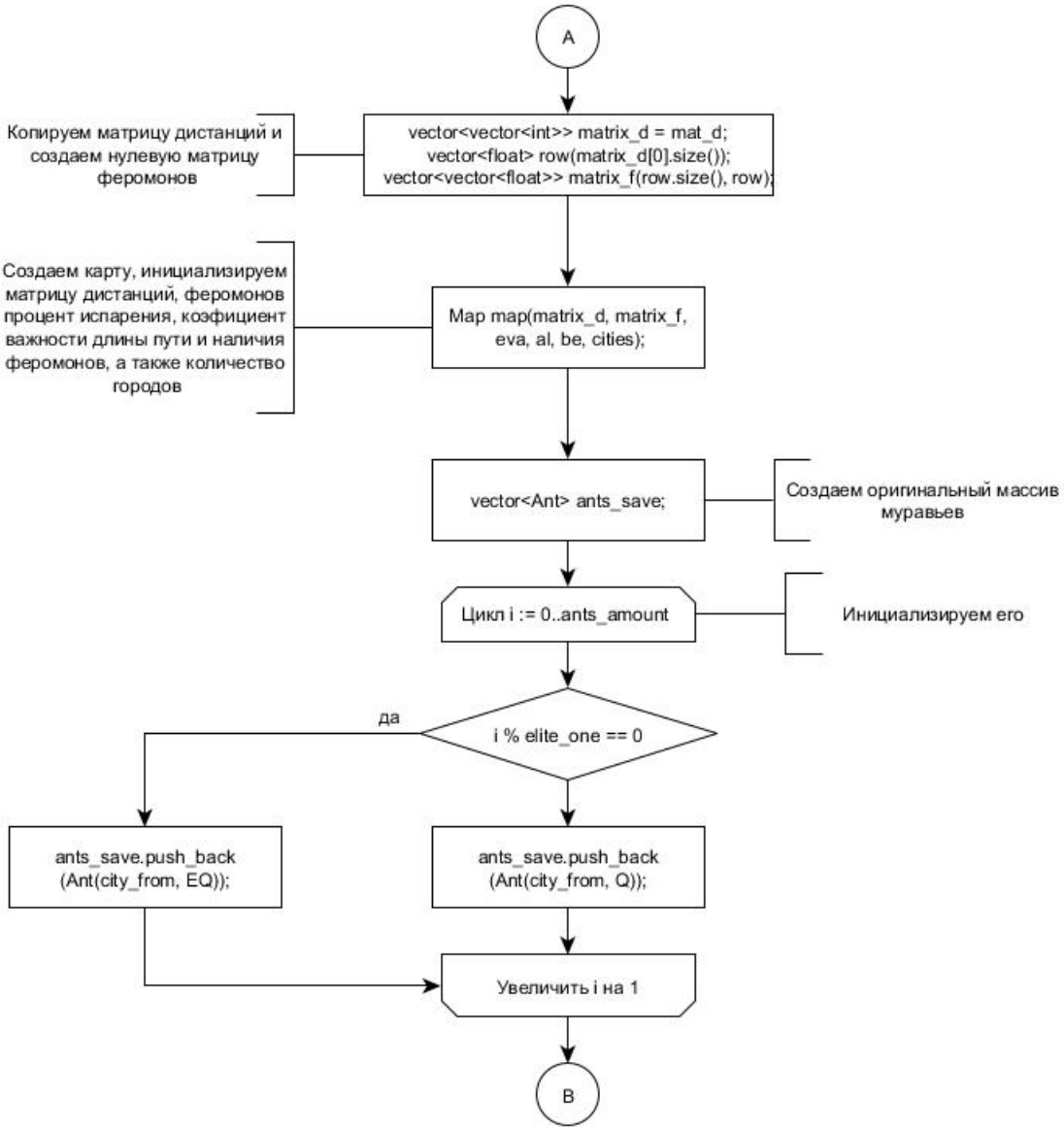

Рис. 2.2: Инициализация муравьев и матриц.

Функция can_move (рис. 2.4) принимает на вход муравья и возвращает false (и следовательно прекращает поход муравья), когда размер массива посещенных городов данного муравья становится больше или равен количеству существующих городов. Иначе вернет true и муравей продолжит двигаться.

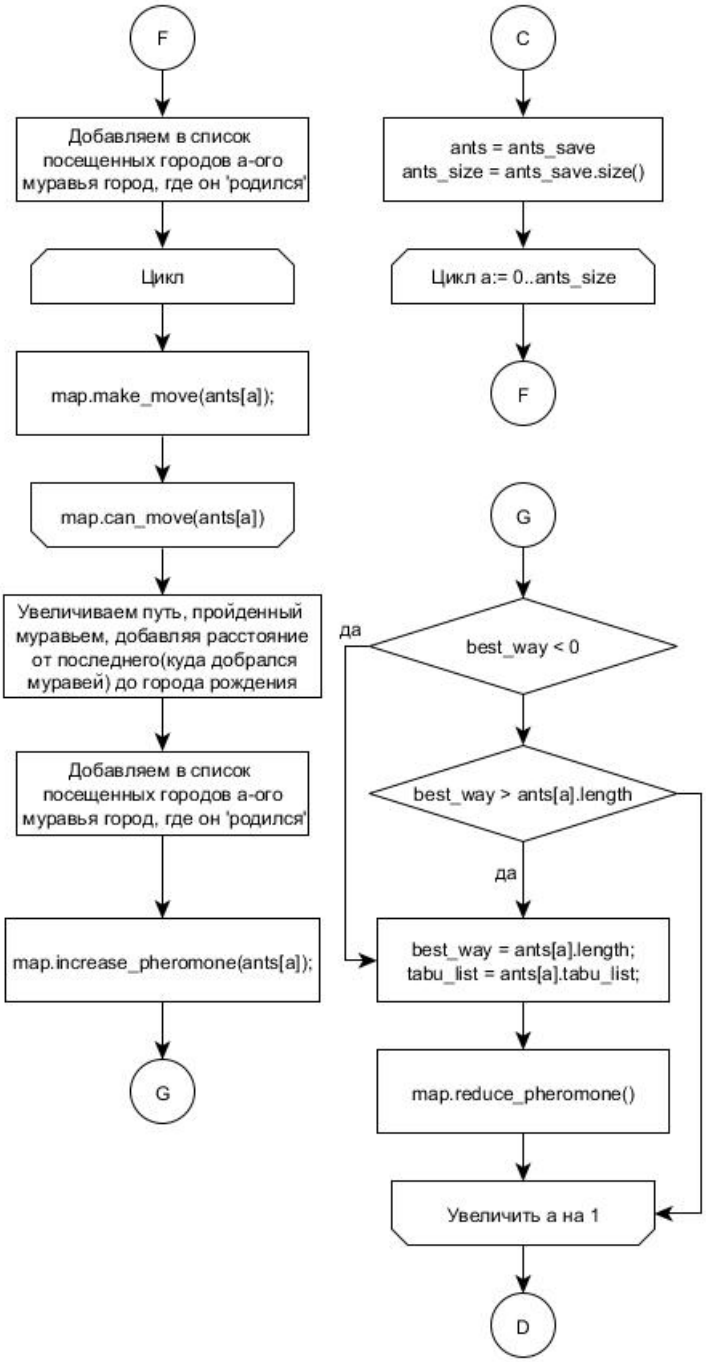

Рис. 2.3: Процесс поиска кратчайшего пути.

Функция make_move(рис. 2.5) класса Map принимает на вход муравья и ничего не возвращает. Вначале выполняется поиск (choose_city() - рис. 2.7) следующего города, куда отправится муравью. После чего определяется расстояние до выбранного города (по матрице дистанций) и дальше функция отдает управление самому муравью, а именно его одноимённой функции make_move.

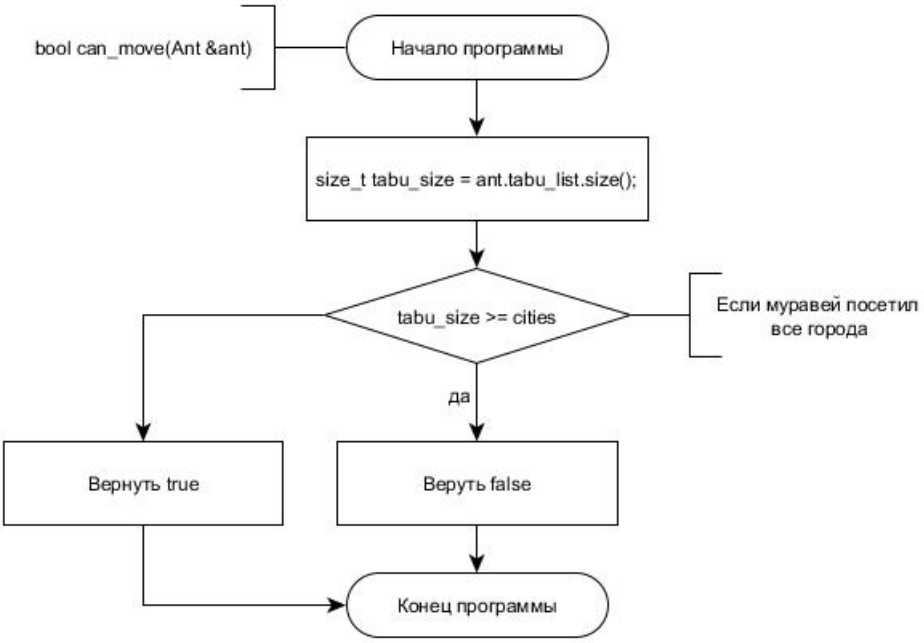

Рис. 2.4: Функция can_move.

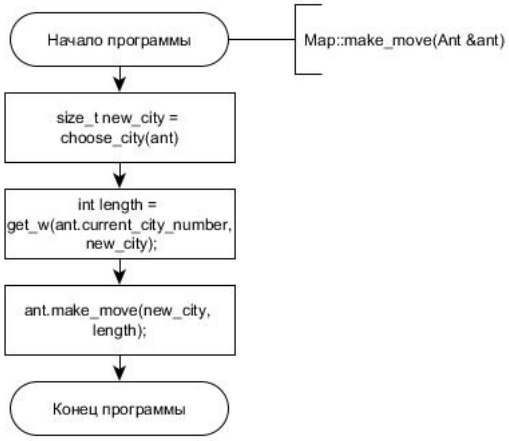

Рис. 2.5: Функция make_move класса Map.

Муравьиная функция make_move(рис. 2.6) принимает на вход два параметра: следующий город и расстояние до него и ничего не возвращает. Эта функция последовательно выполняет следующие 3 действия:

1) помещает номер указанного города в массив посещённых городов;

2) обновляет город пребывания муравья;

3) увеличивает пройденное расстояние муравья на указанное значение.

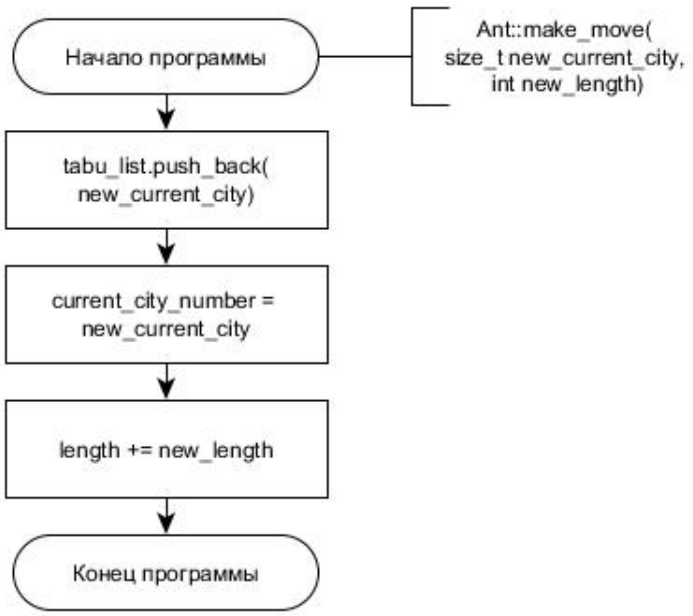

Рис. 2.6: Функция make_move класса Ant.

Функция choose_city (рис. 2.7), упомянутая в make_move для Map принимает на вход муравья, а возвращает номер города, который тот 'выбрал' для следующего посещения. Для этого функция получает массив вероятностей (элементом этого массива является пара вида: номер города, вероятность перехода (процент)) выбора каждого из городов. После чего с помощью генератора случайных чисел создаётся число от 0 до 100 и определяется, какому городу принадлежит отрезок, в котором находится данное число. Например, пусть вероятности попадания в города 1,2,3 равны соответсвенно 30, 60 и 10. Тогда отрезок с 0 по 30 'принадлежит' первому городу, с 31 по 90 - второму, с 90 по 100 - третьему. Если число, созданное генератором, равно 95, то муравей пойдет в третий город. Если число равно 40, то пойдет во второй город и т.д. Массив процентов определяется в функции get_p() - рис. 2.8, 2.9.

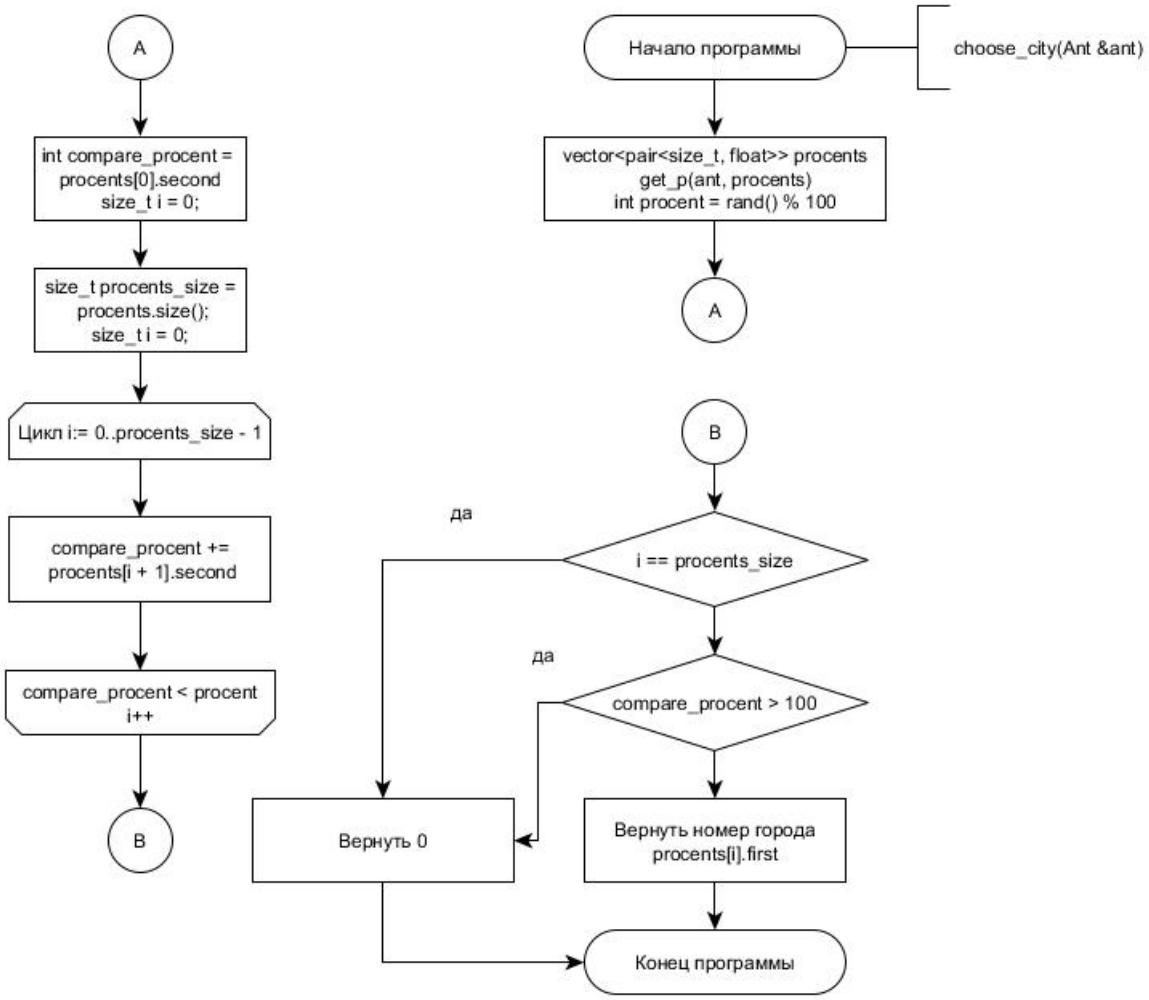

Рис. 2.7: Функция choose_city.

Функция get_p (рис. 2.8, 2.9) получает на вход муравья и ссылку на массив вероятностей. Результатом работы будет обновление этого массива. Первым делом массив вероятностей обнуляется. После чего начинается процесс его заполнения. Выполняется проход по всем городам и проверяется, был ли указанный муравей в i-ом городе. Если был, необходимо переключиться на следующий город, если нет, то вычисляется вероятность похода в этот город. Но перед этим уточняется, что i-ый город не является городом пребывания муравья, для этого достаточно проверить, что путь из текущего города в i-ый не равен нулю. Вероятность определяется по формуле **??**. Числитель этой формулы вынесен в отдельную функцию get_p_chisl. По определению вероятность - это отношение исходов некоторого события к общему количеству исходов. На данный момент есть только числитель формулы, т. е. было посчитано на самом деле количество исходов события перехода в i-ый город. Не хватает общего количества исходов. Для того, чтобы посчитать это количество, необходимо определить за пределами цикла переменную summ и при каждом вычислении количества исходов для определенного города будем прибавлять это значение к summ. При этом будем сохранять кол-во исходов для определенного города в массив вероятностей procents. После того, как будут обойдены все города, необходимо проверить, что переменная summ не ноль(чтобы избежать деления на ноль), а дальше надо пройтись по массиву procents и поделить каждое значение на summ.

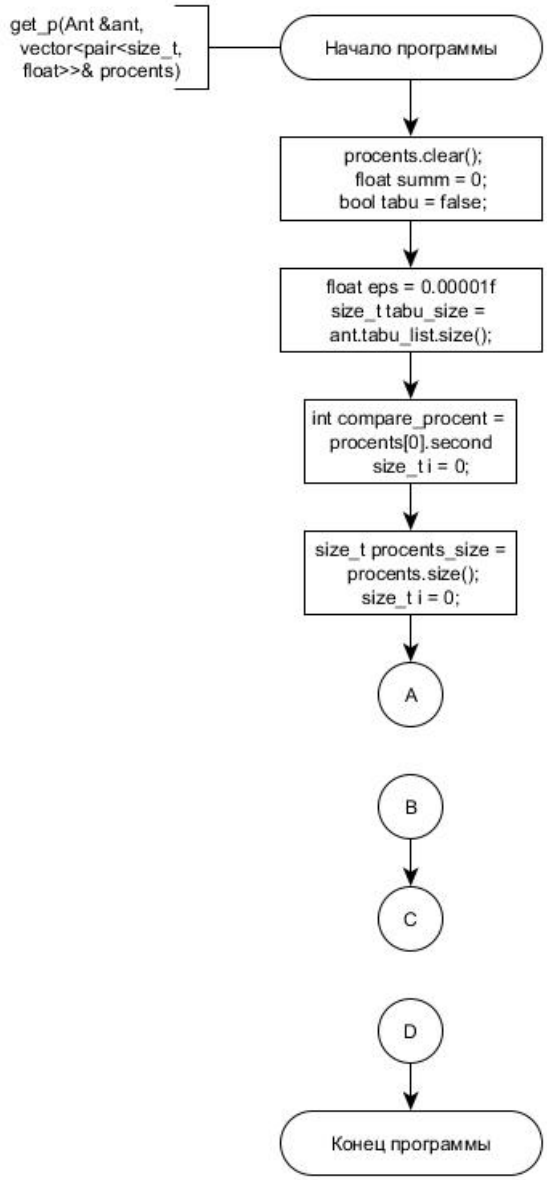

Рис. 2.8: Основная часть функции get_p.

2.2 Вывод

В данном разделе были рассмотрены принципы работы и схемы муравьиного алгоритма.

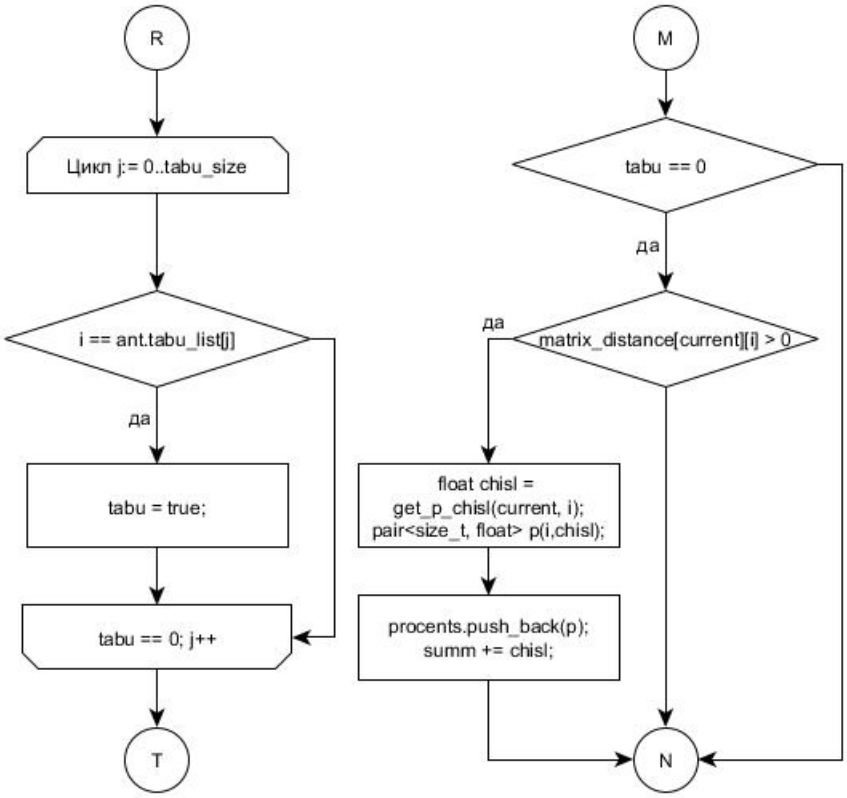

Рис. 2.9: Определение является ли город посещенным и занесение в массив процентов числителей вероятностей.

3. Технологическая часть

В данном разделе приведены требования к программному обеспечению, средствам реализации, а также листинги кода.

3.1 Средства реализации

Для реализации программы был использован язык программирования C++, так как он был подробно изучен в курсе объектно-ориентированного программирования в университете[6]. Для замера времени использовалась функция, приведенная на листинге[4]. Данная функция считает реальное процессорное время в тиках. Для ее работы была подключена библиотека time.h.

3.2 Реализация алгоритмов

Для реализации описанных в конструкторской части решений, было реализовано два класса: Map(листинг 3.1) и Ant(листинг 3.2).

Листинг 3.1: Класс Map

```
1  class Map {
2  public:
3    Map(vector<vector<int>> matrix_d,
4    vector<vector<float>> matrix_p,
5    float eva, float al, float be, size_t ci) :
6    matrix_distance(matrix_d), matrix_pheromone(matrix_p),
7    evaporation(eva), alpha(al), beta(be), cities(ci){;}
8
9    vector<vector<int>> matrix_distance;
10   vector<vector<float>> matrix_pheromone;
11
12   float evaporation;
13   float alpha, beta;
14   size_t cities;
15
16   bool can_move(Ant &ant) {
17     size_t tabu_size = ant.tabu_list.size();
18     if (tabu_size >= this->cities) {
19       return false;
20   }
21   return true;
22   }
23
24   void reduce_pheromone();
25
```

```
26    void increase_pheromone(Ant &ant);
27
28    void make_move(Ant &ant);
29
30  private:
31    float get_p_chisl(size_t from, size_t to);
32
33    int get_w(size_t from, size_t to);
34
35    float get_t(size_t from, size_t to);
36
37    size_t choose_city(Ant &ant);
38
39    void get_p(Ant &ant, vector<pair<size_t, float>>& procents);
40  };
```

Листинг 3.2: Класс Ant

```
1   class Ant {
2   public:
3     Ant(size_t ccn, int Q) {
4       current_city_number = ccn;
5       length = 0;
6       pheromone = Q;
7     }
8     vector<size_t> tabu_list;
9     size_t current_city_number;
10    int length;
11    float pheromone;
12
13    void make_move(size_t new_current_city, int new_length);
14  };
```

Листинг 3.3: Функция run - получить кратчайший путь

```
1   int run(vector<vector<int>> mat_d, size_t time, size_t ants_amount, size_t elite_one, size_t
        cities, float eva, float al, float be, size_t city_from, int Q, int EQ) {
2   if (run_check(mat_d,time, ants_amount, elite_one, cities,
3   eva, city_from) == ERROR) {
4     return ERROR;
5   }
6   vector<vector<int>> matrix_d = mat_d;
7   vector<float> row(matrix_d[0].size());
8   vector<vector<float>> matrix_f(row.size(), row);
9
10  Map map(matrix_d, matrix_f, eva, al, be, cities);
11  vector<Ant> ants_save;
12
13  for (size_t i = 0; i < ants_amount ; i++) {
14    if (i % elite_one == 0)
15      ants_save.push_back(Ant(city_from, EQ));
16    else
17      ants_save.push_back(Ant(city_from, Q));
18  }
```

```
19    vector<size_t> tabu_list;
20    vector<Ant> ants;
21    int best_way = -1;
22
23    size_t ants_size = ants_save.size();
24    for (size_t t = 0; t < time; t++) {
25      ants = ants_save;
26      for (size_t a = 0; a < ants_size; a++) {
27      ants[a].tabu_list.push_back(city_from);
28      while (map.can_move(ants[a])) map.make_move(ants[a]);
29
30      if (ants[a].tabu_list.back() < map.matrix_distance.size()) {
31      ants[a].length += map.matrix_distance[ants[a].tabu_list.back()][city_from];
32      ants[a].tabu_list.push_back(city_from);
33    }
34    else
35      std::cout << "\n strange error";
36
37    map.increase_pheromone(ants[a]);
38
39    if (best_way < 0 || best_way > ants[a].length) {
40      best_way = ants[a].length;
41      tabu_list = ants[a].tabu_list;
42  }
43  return best_way;
44    }
```

Листинг 3.4: Функция choose_city - Выбрать следующий город для муравья

```
1   size_t Map::choose_city(Ant &ant) {
2     vector<pair<size_t, float>> procents;
3     get_p(ant, procents);
4     int procent = rand() % 100;
5     int compare_procent = procents[0].second;
6
7     size_t procents_size = procents.size();
8     size_t i = 0;
9     for (;i < procents_size - 1 && compare_procent < procent;i++)
10      compare_procent += procents[i + 1].second;
11    if (i == procents_size) {
12      std::cout << "\n choose_city out of borders " <<
13      i << "/" << procents_size;
14      return 0;
15    }
16
17    if (compare_procent > 100) {
18      std::cout << "\n choose_city() error procenting " <<
19      compare_procent;
20      return 0;
21    }
22    return procents[i].first;
23  }
```

Листинг 3.5: Функция ant::make_move

```
1  void Ant::make_move(size_t new_current_city, int new_length) {
2    tabu_list.push_back(new_current_city);
3    current_city_number = new_current_city;
4    length += new_length;
5  }
```

3.3 Вывод

В данном разделе была рассмотрена конкретная реализация на языке C++ муравьиного алгоритма.

4. Экспериментальная часть

В данном разделе будет сравнительный анализ муравьиного алгоритма в зависимости от выбранных параметров на основе экспериментальных данных.

4.1 Сравненительный анализ

Для экспериментов использовалась матрица расстояний 10x10, изображенная в таблице 4.1. Количество повторов каждого эксперимента = 50. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

0	4	5	7	8	9	2	3	4	10
4	0	3	2	6	7	2	5	9	3
5	3	0	9	8	7	6	7	8	9
7	2	9	0	4	1	8	5	6	6
8	6	8	4	0	7	4	2	1	8
9	7	7	1	7	0	8	3	2	1
2	2	6	8	4	8	0	9	9	9
3	5	7	5	2	3	9	0	9	9
4	9	8	6	1	2	9	9	0	9
10	3	9	6	8	1	9	9	9	0

Таблица 4.1: Матрица расстояний между 10 городами.

Было проведено несколько опытов. Первый проводился с 50 муравьями, коэффициентом распыления 0.1, без элитных муравьев и в 200 поколений. Замерялось, как влияют коэффициенты "жадности"(α) и "стадности"(β)). По умолчанию $\alpha = 0$, а $\beta = 1$. Затем α увеличивается на 0.05 до 1, а β понижается до 0 с тем же шагом 0.05. Результаты занесены в таблицу 4.1. Она состоит из 4-ёх столбцов: корректность(не при всех коэффициентах получался правильный ответ), α , β и скорость работы. Как видно из рисунков 4.1 и 4.2 наиболее быстрым и в то же время правильным решением является использование коэффициентов $\alpha = 0.7$, $\beta = 0.3$, хотя разница по сравнению с другими значениями не столь велика и при других значениях прочих коэффициентов - количества обычных и элитных муравьев или поколений, а также коэффициента испарения - можно получить другие результаты.

Корректность (от 0 до 1)	α (от 0 до 1)	β (от 0 до 1)	время работы (в тиках)
0.93	0.00	1.00	552793632
0.90	0.05	0.95	671431366
0.93	0.10	0.90	677701845
1.00	0.15	0.85	688931058
1.00	0.20	0.80	672681069
1.00	0.25	0.75	665585706
0.93	0.30	0.70	678160444
1.00	0.35	0.65	685878101
1.00	0.40	0.60	772171866
0.97	0.45	0.55	650128055
0.97	0.50	0.50	641031277
1.00	0.55	0.45	664468673
1.00	0.60	0.40	643548411
1.00	0.65	0.35	645030182
1.00	0.70	0.30	641664003
1.00	0.75	0.25	647894697
1.00	0.80	0.20	654792064
1.00	0.85	0.15	670107572
0.93	0.90	0.10	675516409
1.00	0.95	0.05	663381617
1.00	1.00	0.00	718245238

Таблица 4.2: Зависимость времени и корректности решения от коэффициентов α и β.

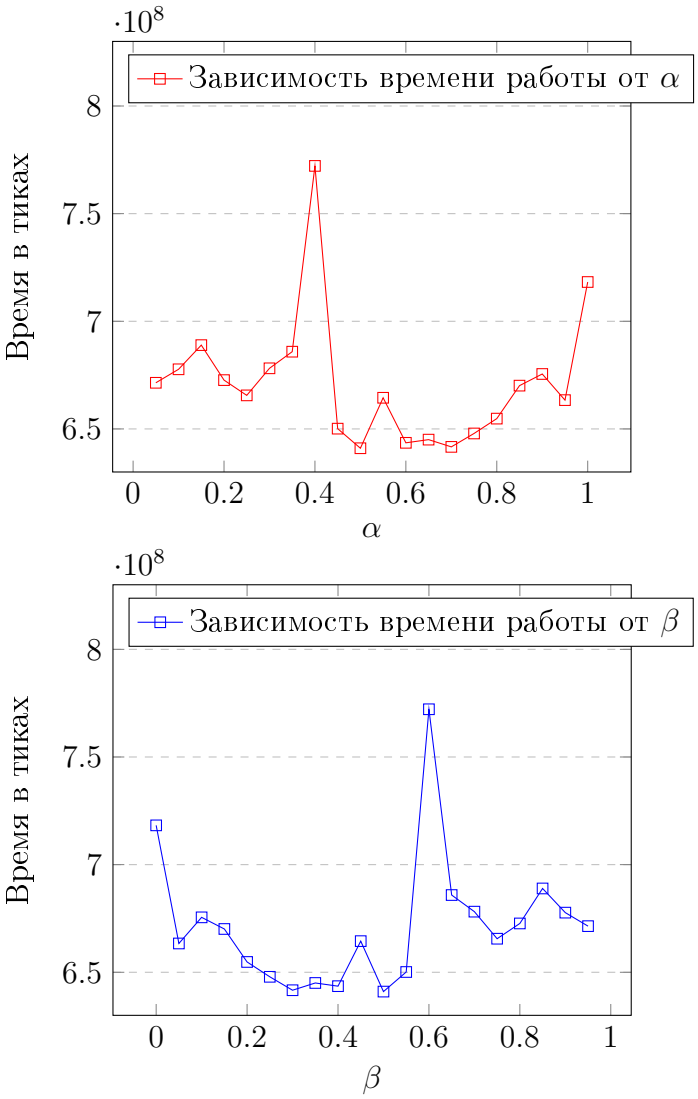

Рис. 4.1: График зависимости времени работы от α

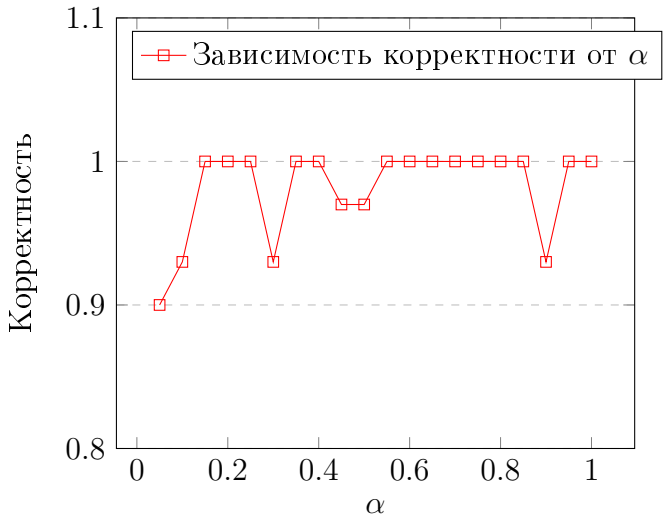

Рис. 4.2: График зависимости корректности ответа от α.

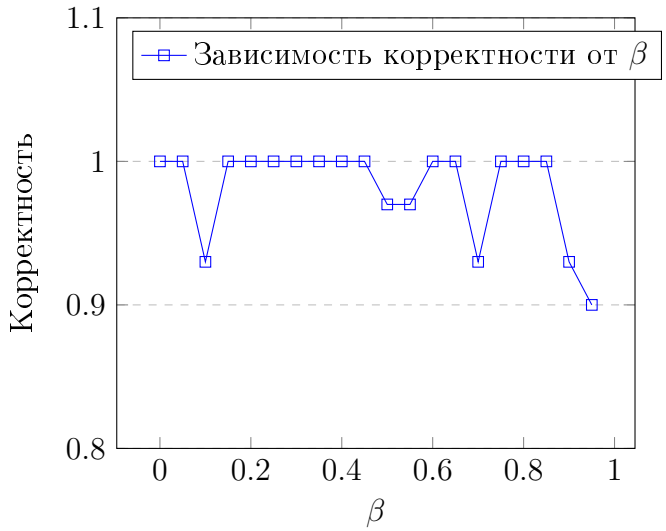

Рис. 4.3: График зависимости корректности ответа от β.

Теперь необходимо проанализировать зависимость от количества муравьев и количества поколений. α и β теперь будут константами. Количество муравьев варьируется от 10 до 80 с шагом сначала 10, потом 20, 40 и т.д., количество поколений от 20 до 650 с шагом сначала 10, потом 20, 40 и т.д. Результаты замеров сведены в таблицу 4.1. Она также состоит из 4 столбцов. Первый столбец вновь означает корректность, а четвертый - время работы. А второй и третий - 'количество муравьев' и 'количество замеров' соответственно. Рисунок 4.4 подтверждает формулу **??** расчёта трудоёмкости муравьиного алгоритма. Более интересен рисунок 4.5, который показывает, что при 20-и поколениях нет ни одного правильного ответа, при поколениях от 30 до 50 правильные ответы появляются только если взять более 40 муравьев. От 90 до 330 правильный ответ выводится уже при 20 муравьях, а если поколений 650, то и при 10 муравьях ответ будет правильный. Напомню, что обход происходит по 10 городам.

Корректность (от 0 до 1)	Количество муравьев (в штуках)	Количество поколений (в штуках)	время работы (в тиках)
0.93	10	20	11123431
0.93	20	20	23200249
0.93	40	20	45064059
0.93	80	20	89106562
0.93	10	30	15277990
0.93	20	30	30652063
1.00	40	30	62648153
1.00	80	30	132719349
0.93	10	50	27005527
0.93	20	50	52959829
1.00	40	50	109892963
1.00	80	50	211573758
0.93	10	90	47740420
1.00	20	90	94700006
1.00	40	90	199694544
1.00	80	90	387066569
0.93	10	170	90954856
1.00	20	170	181294108
1.00	40	170	363646870
1.00	80	170	729324525
0.93	10	330	174772204
1.00	20	330	344393887
1.00	40	330	712391169
1.00	80	330	1457121092
1.00	10	650	348804549
1.00	20	650	719097981
1.00	40	650	1431310052
1.00	80	650	2878805475

Таблица 4.3: Зависимость времени и корректности решения от количества муравьев и поколений.

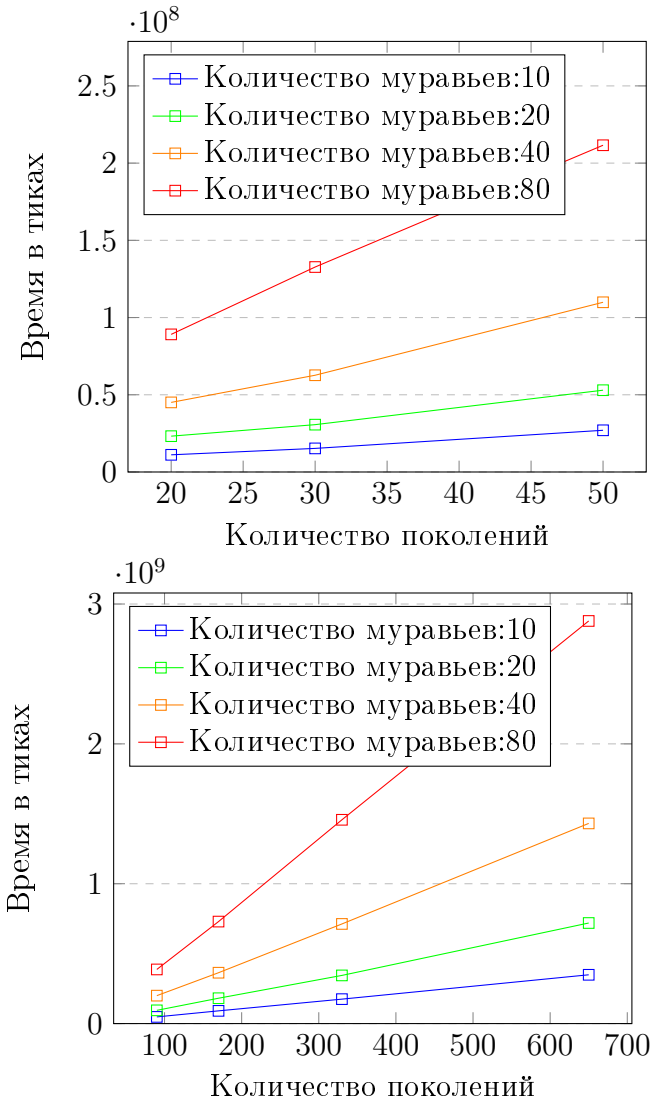

Рис. 4.4: График зависимости времени работы алгоритма от количества муравьев и поколений.

Теперь необходимо проверить, как сильно изменится скорость и корректность, если начать менять выделяемое муравьем количество феромонов и добавить элитных с повышенным количеством. Количество выделяемого феромона у обычного муравья будет варьироваться от 1 до 5 с шагом 2. У элитного всегда 10. Количество элитных изменяется от 50 до 1. Суммарное количество муравьев всегда равно 50. Процент испарения 0.1. Результаты работы приведены в таблице 4.1, где колчиество феромонов и количество элитных муравьев это 2 и 3 столбец. 1 и 4 как всегда корректность и затраченное время. Если посмотреть на рисунок 4.6, то можно заметить, что лучшие результаты достигаются, когда обычный муравей выделяет мало феромона(по сравнению с элитными) и при этом среди 50 муравьев 3 или 7 являются элитными(примерно 7 и 14 процентов от общего количества). Когда элитных муравьев слишком много или обычные муравьи выделяют феромона почти как элитные, то все преимущество наличия элитных муравьев исчезает, поскольку все муравьи примерно одинаковы. На рисунке 4.7 продемонстрирована зависимость корректности ответа от количества элитных муравьев и феромонов.

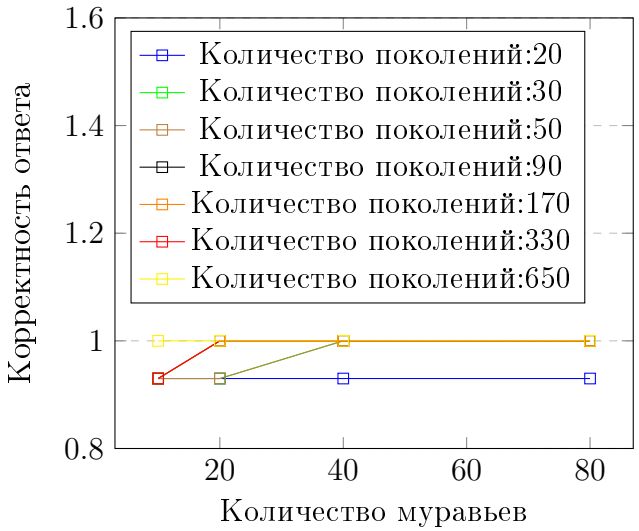

Рис. 4.5: График зависимости корректности работы алгоритма от количества муравьев и поколений.

Корректность (от 0 до 1)	Количество феромона, выделяемое обычным муравьем(в штуках)	Количество элитных муравьев(в штуках)	скорость работы (в тиках)
1.00	1	1	559773514
1.00	3	1	549665664
1.00	5	1	599716769
1.00	1	3	544924989
1.00	3	3	573101663
0.93	5	3	579858919
1.00	1	7	539835456
1.00	3	7	579842625
0.90	5	7	574067559
0.93	1	16	579406567
0.93	3	16	576663013
0.90	5	16	574714519
0.90	1	50	572369419
0.90	3	50	572640859
1.00	5	50	576265339

Таблица 4.4: Зависимость скорости и корректности решения от количества элитных муравьев и выделяемого феромона обычным муравьем.

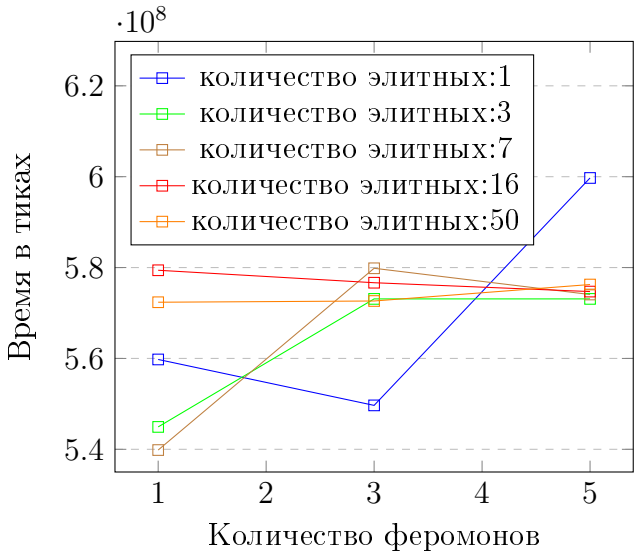

Рис. 4.6: График зависимости времени работы алгоритма от количества элитынх муравьев и феромонов.

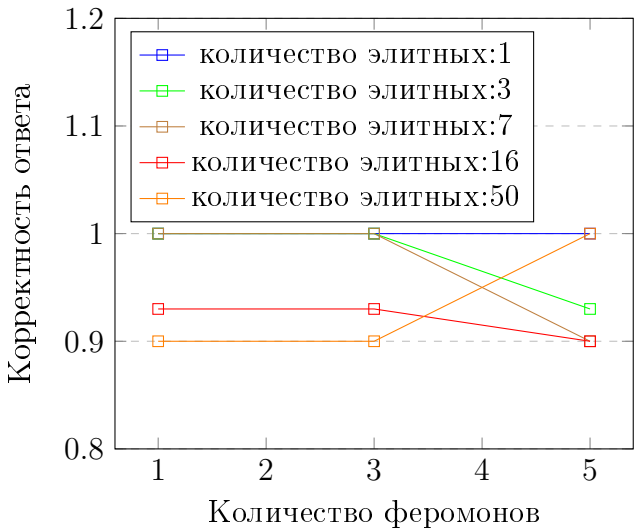

Рис. 4.7: График зависимости корректности работы от количества элитных муравьев и феромонов обычного муравья.

Теперь необходимо измерить влияние процента испарения феромона. Поскольку оно привязано к количеству феромона, надо варьировать β от 2 до 6, при этом $\alpha = 1$. Процент испарения изменяется от 0 до 0.9 с шагом 0.1. Элитных муравьев необходимо исключить, количество феромонов, переносимое одним муравьем, установить в единицу. Чтобы увеличить количесво некорректных ответов, надо задать малое количество муравьев и поколений - 30 и 20 соответственно. Полученные результаты занесены в таблицу 4.1, которая состоит из следующих стобцов: корректность, β (можно рассматривать как отношение β к α), коэффициент испарения, время работы. Из рисунка 4.8 видно, что наиболее хорошим значением коэффициента испарения является 0.1, но при нем алгоритм работает неправильно, когда β=2. Наиболее общий результат наблюдается при коэффициенте испарения, равным 0.4, все три замера дают почти идентичную скорость работы, причем для β=2 и β=6 наибольшую.

корректность (от 0 до 1)	β	коэффициент испарения (от 0 до 1)	скорость работы (в тиках)
0.93	2	0.00	32618353
1.00	4	0.00	33330212
1.00	6	0.00	32412299
0.93	2	0.10	32272986
1.00	4	0.10	31939012
0.93	6	0.10	32491167
1.00	2	0.20	32825864
1.00	4	0.20	32374316
0.90	6	0.20	31968868
0.93	2	0.30	31947558
1.00	4	0.30	32982935
0.90	6	0.30	32648815
1.00	2	0.40	32345549
1.00	4	0.40	32227624
1.00	6	0.40	32410232
1.00	2	0.50	32434753
0.93	4	0.50	33008954
1.00	6	0.50	32432297
0.93	2	0.60	33403780
0.93	4	0.60	32512934
1.00	6	0.60	32638150
0.93	2	0.70	32806587
1.00	4	0.70	33044899
0.93	6	0.70	33249484
1.00	2	0.80	33009915
0.93	4	0.80	33375704
0.93	6	0.80	33231976
1.00	2	0.90	33298340
0.93	4	0.90	33041315
1.00	6	0.90	33014037

Таблица 4.5: Зависимость скорости и корректности решения от испарения и влияния количества феромонов

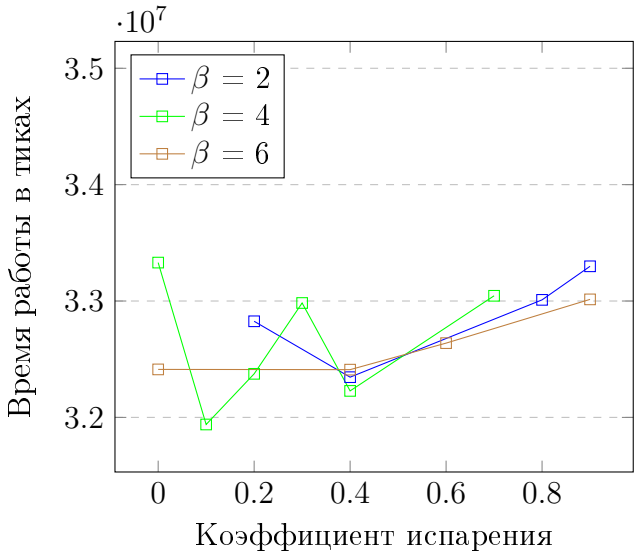

Рис. 4.8: График зависимости скорости работы от коэффициента испарения с учетом, что ответ верен.

4.2 Вывод

Скорость муравьиного алгоритма колеблется в зависимости от выбранных параметров. Наиболее быстрое решение удалось достичь при $\alpha = 0.7$, $\beta = 0.3$. Оптимальное количество поколений для 10 городоа составляет 30, количество муравьев - 40. Введение элитных муравьев ускоряет процесс, однако лишь в том случае когда процент элитных составляет 7-14 процентов от общего числа. Опытным путем было определено, что оптимальным соотношением количества феромонов обычного муравья к количеству феромонов элитного муравья является 0.1. Так, у обычного например 1, а у элитного 10. Коэффициент испарения - 0.4.

Заключение

Таким образом, в ходе работы был изучен и реализован муравьиный алгоритм. Были введены элитные муравьи для достижения более хороших результатов. Таким образом, были определены наиболее оптимальные параметры для быстрого и правильного решения задачи о коммивояжера на матрице из 10 городов.

Литература

[1] Штовба С.Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. – 2003. – №4

[2] Шутова Ю.О., Мартынова Ю.А. ИССЛЕДОВАНИЕ ВЛИЯНИЯ РЕГУЛИРУЕМЫХ ПАРАМЕТРОВ МУРАВЬИНОГО АЛГОРИТМА НА СХОДИМОСТЬ. Томский политехнический университет, 634050, Россия, г. Томск, пр. Ленина, 30, 2014. С. 281-282.

[3] Чураков Михаил, Якушев Андрей Муравьиные алгоритмы. 2006. С. 9- 11.

[4] [Электронный ресурс] Документация по функции замера времени. https://proginfo.ru/time/

[5] И.В. Ломовской. Курс лекций по языку программирования С, 2017.

[6] https://cppreference.com/ [Электронный ресурс]