Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №4

Название: Алгоритмы поиска в графе

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ П.К Хетагуров

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение

В данной лабораторной работе будут рассмотренны и проанализированы два алгоритма поиска решения в задаче коммивояжера. Проведено тестирование и анализ рассмотренных алгоритмов.

Задача коммивояжера заключается в поиске кратчайшего гамильтонова цикла в графе. Задача является NP-полной.

1 Аналитическая часть

В данном разделе будут поставлены цели и задачи работы, будут рассмотренны основные теоритические сведения связанные с алгоритмами.

1.1 Цель и задачи работы

Цель работы:

Изучение муравьинного алгоритма.

Задачи работы:

1. описать алгоритм полного перебора и муравьиный алгоритм;
2. реализовать описанные алгоритмы;
3. провести эксперименты по замеру времени работы разработанных алгоритмов;
4. провести сравнения алгоритмов по затраченному времени.

1.2 Алгоритм полного перебора

Полный перебор подразумевает рассмотрение всех возможных решений задачи и выбор из них подходящего. Время вычислений напрямую зависят от количества возможных решений. Так как задача коммивояжера является NP-полной, то, как и на любой NP-полной задаче, поиск решения методом полного перебора быстро становится трансвычислительным. Любая NP-полная задача может быть решена полным перебором.

1.3 Муравьиный алгоритм

Идея муравьиного алгоритма - моделирование поведения реальных муравьев при поиске пути от источника пищи к муравейнику. [1]

Пройдя по ребру (i, j) графа, муравей помечает его определенным количеством феромона, по формуле (1), где Q - параметр, имеющий значение порядка длины оптимального пути, $T_k(t)$ - путь, пройденный муравьем k к моменту времени t, а $L_k(t)$ - длина этого пути.

$$\Delta\tau_{ij,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, & (i,j) \in T_k(t) \\ 0, & (i,j) \notin T_k(t) \end{cases} \tag{1}$$

Через еденицу времени, после движения всех муравьев в следующую точку своего пути, происходит испарение феромона по правилу (2), где p - коэффицент испарения.

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}(t); \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \tag{2}$$

Правила поведения муравьев:

1. муравьи имеют собственную "память у муравья уже есть список посещенных городов;

2. муравьи обладают "зрением"η_{ij} - видимость есть эвристическое желание посетить город j, если муравей находится в городе i, расчитанное по формуле (3).

$$\eta_{ij} = \frac{1}{D_{ij}} \tag{3}$$

3. муравьи обладают "обонянием они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьев.

Правило, определяющее вероятность перехода k-ого муравья из города i в город j, описано в формуле (4).

$$\begin{cases} P_{ij,k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in J_{ik}} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}}, & j \in J_{i,k}; \\ P_{ij,k}(t) = 0, & j \notin J_{i,k} \end{cases} \tag{4}$$

Где α, β - параметры, задающие веса следа феромона.

1.4 Вывод

В данной части были поставлены задачи и цель работы, рассмотрены алгоритм полного перебора и муравьиный алгоритм.

2 Конструкторская часть

В данном разделе будут рассмотренны схемы алгоритмов, требования к функциональности ПО.

2.1 Требования к ПО

ПО должно иметь два режима работы, выбираемых из меню

1. Режим демонстрации. В этом должена осуществляться демонстрация работы на ней всех реализованных алгоритмов.

2. Режим тестирования. В этом режиме должны проводится замеры времени выполнения реализованных алгоритмов.

2.2 Схема алгоритма полного перебора

На рисунке 1 изображена схема алгоритма полного перебора.

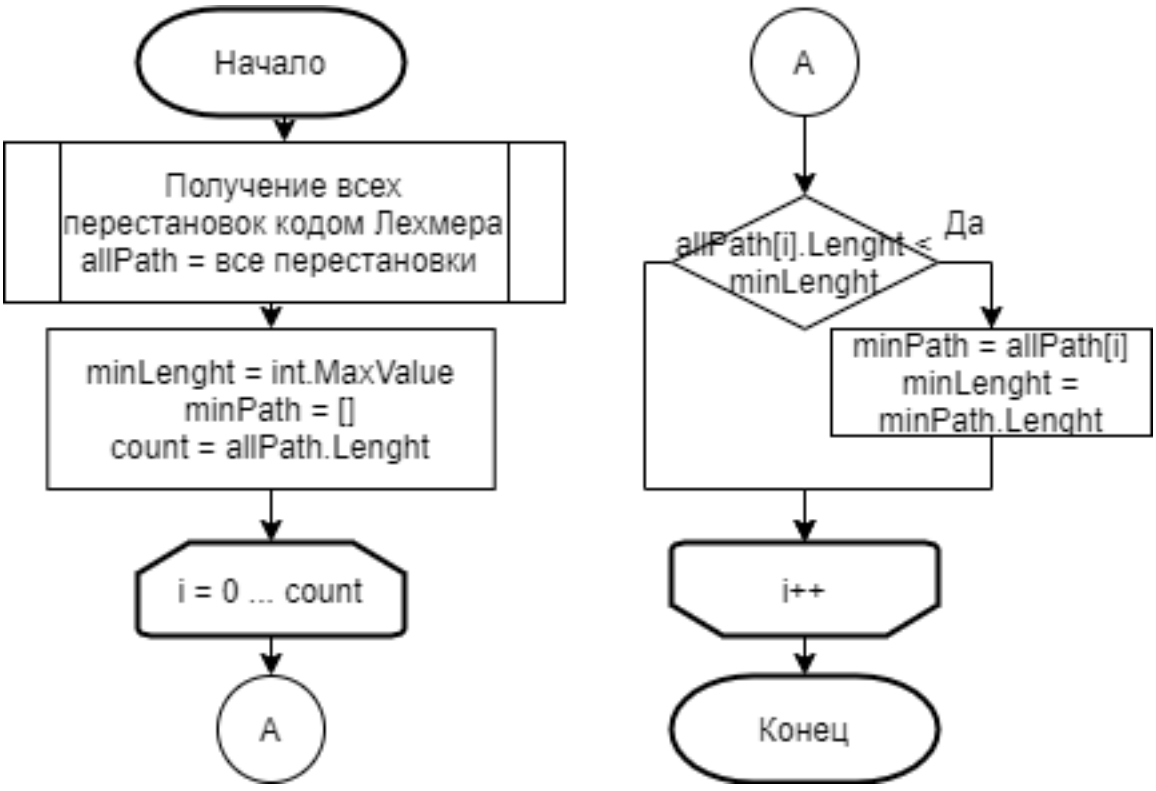


Рисунок 1 – Схема алгоритма полного перебора

2.3 Схема муравьиного алгоритма

На рисунке 2 изображена схема муравьиного алгоритма.

1. Ввод матрицы расстояний D.
2. Инициализация параметров алгоритма — α, β, e, Q.
3. Инициализация ребер — присвоение видимости η_{ij} и начальной концентрации феромона.
4. Размещение муравьев в случайно выбранные города без совпадений.
5. Выбор начального кратчайшего маршрута и определение L*
6. Цикл по времени жизни колонии t=1,t_{max}.
   7. Цикл по всем муравьям k=1,m
      8. Построить маршрут $T_k(t)$по правилу (7.3.1) и рассчитать длину $L_k(t)$.
   9. конец цикла по муравьям.
   10. Проверка всех $L_k(t)$ на лучшее решение по сравнению с L*.
   11. Если да, то обновить L* и T*.
   12. Цикл по всем ребрам графа.
      13. Обновить следы феромона на ребре по правилам (7.3.2) и (7.3.3).
   14. конец цикла по ребрам.
15. конец цикла по времени.
16. Вывести кратчайший маршрут T* и его длину L*.

Рисунок 2 – Схема муравьиного алгоритма

2.4 Вывод

В данном разделе были рассмотрены схемы алгоритмов и обозначены требования к ПО.

3 Технологическая часть

Ниже будут представлены средства реализации и листинги реализованной программы.

3.1 Средства реализации

Выбранный язык программирования - C#, так как требований по конкретнему языку не выдвигалось и он предоставляет удобные средства распараллеливания.[2] Среда разработки - Visual Studio.[3]

Технические характеристики машины, на которой проводились тесты:

- Windows 10 x64;
- 8 ГБ оперативной памяти;
- CPU: AMD FX(tm)-6350 Six-Core Processor 3.90GHz;
- 6 логических ядер.

3.2 Реализации алгоритмов

Ниже представлены листинги реализаций алгоритмов. На листинге 1 представлен алгоритм полного перебора

Листинг 1 – Алгоритм полного перебора

```
1  static class BruteForce
2    {
3        public static Path GetMinPath(Graph graph)
4        {
5            Path shortest = new Path(graph);
6            bool isStart = true;
7            foreach (Path cur in GetAllRoutes(graph))
8            {
9                if (isStart)
10               {
11                   shortest = cur;
12                   isStart = false;
13               }
14               if (shortest.N > cur.N)
15               {
16                   shortest = cur;
17               }
18           }
19           return shortest;
20       }
```

```
21
22          private static List<Path> GetAllRoutes(Graph graph)
23          {
24              int count = graph.count;
25              List<Path> routes = new List<Path>();
26              List<int> data = new List<int>();
27              int[] factorial = new int[count];
28              int allSwap = 1;
29              for (int i = 0; i < count; i++)
30              {
31                  data.Add(i);
32                  allSwap *= i + 1;
33                  factorial[i] = allSwap;
34              }
35
36              for (int i = 0; i < allSwap; i++)
37              {
38                  Path curRes = new Path(graph);
39                  List<int> source = new List<int>(data);
40                  for (int j = 0; j < count; j++)
41                  {
42                      int p = i / factorial[count - 1 - j] % source.Count;
43                      curRes.AddTown(source[p]);
44                      source.RemoveAt(p);
45                  }
46                  curRes.AddTown(curRes.FirstTown());
47                  routes.Add(curRes);
48              }
49              return routes;
50          }
```

На листинге 2 представлен муравьиный алгоритм.

Листинг 2 – Муравьиный алгоритм

```
1  static class AntColonyOptimization
2      {
3          public static Path GetRoute(Graph graph, int maxTime, double alpha, double beta
               , double Q, double pho)
4          {
5              Random r = new Random();
6
7              Path shortest = new Path(graph);
```

```
8               shortest.distance = int.MaxValue;
9
10              int count = graph.count;
11              double[,] pher = InitPheromone(0.1, count);
12
13              for (int time = 0; time < maxTime; time++)
14              {
15                  List<Ant> ants = InitAnts(graph);
16                  for (int i = 0; i < count - 1; i++)
17                  {
18                      double[,] deltaPher = InitPheromone(0, count);
19                      foreach (Ant ant in ants)
20                      {
21                          int curTown = ant.LastVisited();
22
23                          double sum = 0;
24                          for (int town = 0; town < count; town++)
25                          {
26                              if (!ant.IsVisited(town))
27                              {
28                                  double tau = pher[curTown, town];
29                                  double eta = 1.0 / graph[curTown, town];
30                                  sum += Math.Pow(tau, alpha) * Math.Pow(eta, beta);
31                              }
32                          }
33
34                          double[] probability = new double[count];
35                          for (int checkTown = 0; checkTown < count; checkTown++)
36                          {
37                              double chance = 0;
38                              if (!ant.IsVisited(checkTown))
39                              {
40                                  double tau = pher[curTown, checkTown];
41                                  double eta = 1.0 / graph[curTown, checkTown];
42                                  chance = Math.Pow(tau, alpha) * Math.Pow(eta, beta) /
                                        sum;
43                              }
44                              probability[checkTown] = chance;
45                          }
46                          int newTown = GetPosByProb(probability);
47                          ant.VisitTown(newTown);
```

```
48                      deltaPher[curTown, newTown] += Q / ant.GetDistance();
49                  }
50                  for (int k = 0; k < count; k++)
51                  {
52                      for (int t = 0; t < count; t++)
53                      {
54                          pher[k, t] = (1 - pho) * pher[k, t] + deltaPher[k, t];
55                          if (pher[k, t] < 0.1)
56                          {
57                              pher[k, t] = 0.1;
58                          }
59                      }
60                  }
61              }
62              foreach (Ant ant in ants)
63              {
64                  ant.VisitTown(ant.Start);
65
66                  if (ant.GetDistance() < shortest.N)
67                  {
68                      shortest = ant.GetPath();
69                  }
70              }
71          }
72          return shortest;
73      }
74      static int GetPosByProb(double[] probability)
75      {
76          double[] comulativeProb = new double[probability.Length];
77          comulativeProb[0] = probability[0];
78          for (int i = 1; i < probability.Length; i++)
79          {
80              comulativeProb[i] = probability[i] + comulativeProb[i - 1];
81          }
82
83          Random random = new Random((int)DateTime.UtcNow.Ticks);
84          int res = 0;
85          double choose = random.NextDouble();
86          for (int i = 0; i < comulativeProb.Length; i++)
87          {
88              if (choose <= comulativeProb[i] && probability[i] != 0)
```

```
89                  {
90                      res = i;
91                      break;
92                  }
93              }
94              return res;
95
96          }
97
98          private static List<Ant> InitAnts(Graph graph)
99          {
100             List<Ant> ants = new List<Ant>();
101             for (int i = 0; i < graph.count; i++)
102             {
103                 ants.Add(new Ant(graph));
104                 ants[i].VisitTown(i);
105             }
106             return ants;
107         }
108
109         private static double[,] InitPheromone(double num, int size)
110         {
111             double[,] phen = new double[size, size];
112             for (int i = 0; i < size; i++)
113             {
114                 for (int j = 0; j < size; j++)
115                 {
116                     phen[i, j] = num;
117                 }
118             }
119             return phen;
120         }
121     }
```

3.3 Вывод

В данном разделе были описаны программные и аппаратные средства реализации, были представлены листинги реализаций алгоритмов.

4 Экспериментальная часть

В данной главе будет представлен пример работы программы, результат экспериментов по замеру времени и произведен сравнительный анализ алгоритмов по затрачиваемому времени.

4.1 Пример работы программы

Пример работы программы представлен на рисунке 3

```
Полный перебор нашел путь: 0 1 3 2 4 0
Длина пути: 8

Муравьи нашли путь: 0 1 3 2 4 0
Длина пути: 8
```

Рисунок 3 – Пример работы программы

4.2 Сравнительный анализ алгоритмов по времени

Были проведены замеры времени работы алгоритмов на количестве вершин от 2 до 10 с шагом 1 и от 10 до 100 с шагом 5. Результаты представлены на рисунке 4.

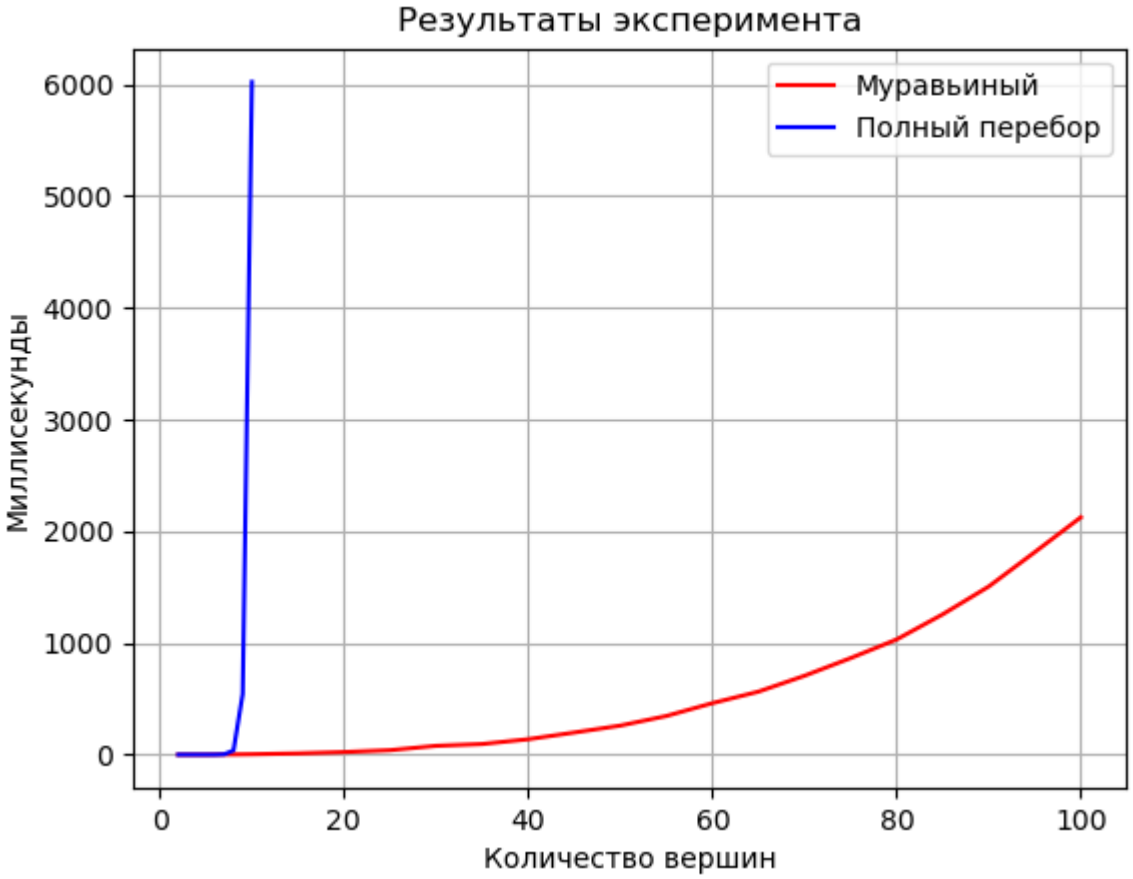


Рисунок 4 – Результаты эксперимента

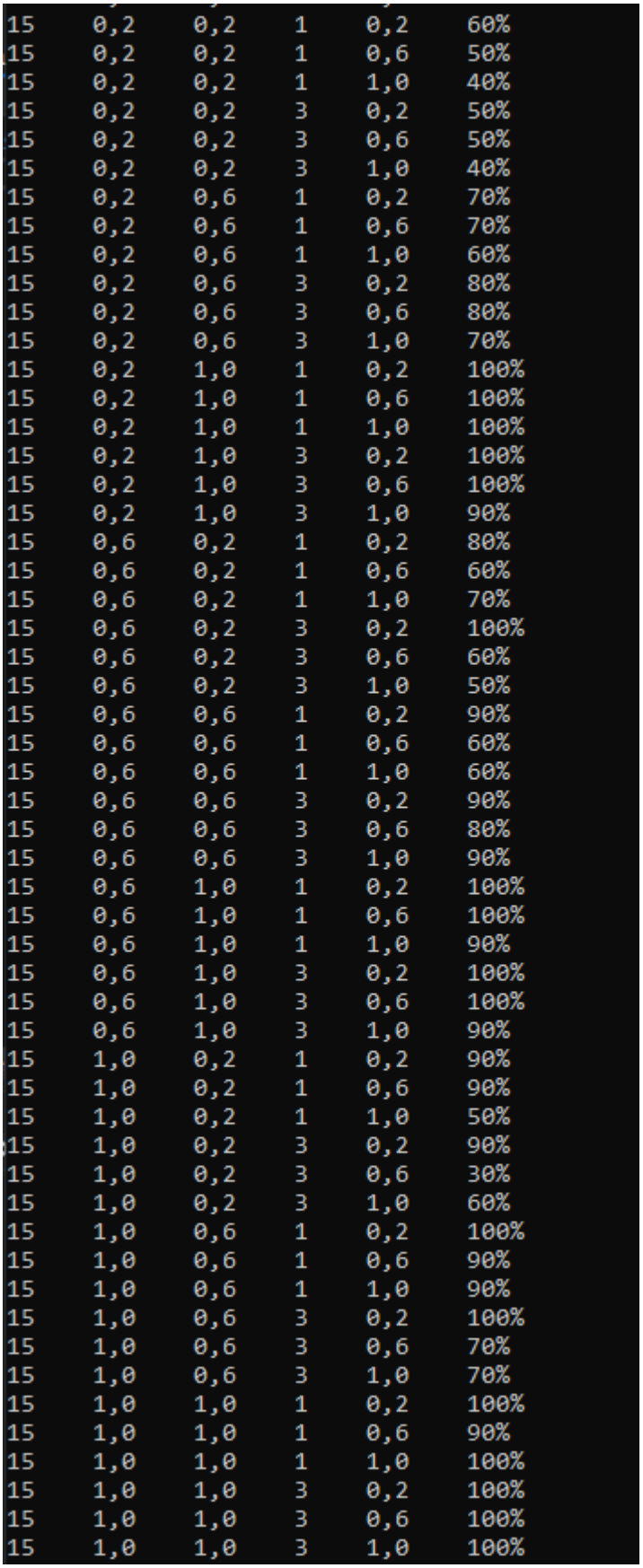
Как видно из графиков, муравьиный алгоритм значительно быстрее алгоритма полного перебора. Алгоритм полного перебора уже при 10 вершинах демонстрирует резкий рост времени выполнения, что делает дальнейшее его тестирование слишком затратным по времени. Муравьиный же алгоритм даже при 100 вершинах демонстрирует меньшее время работы, чем алгоритм полного перебора при 10 вершинах.

4.3 Параметризация муравьиного алгоритма

На рисунках 5- 7 изображен процент успешного решения задачи в серии из 10 задач в зависимости от варьируемых параметров.

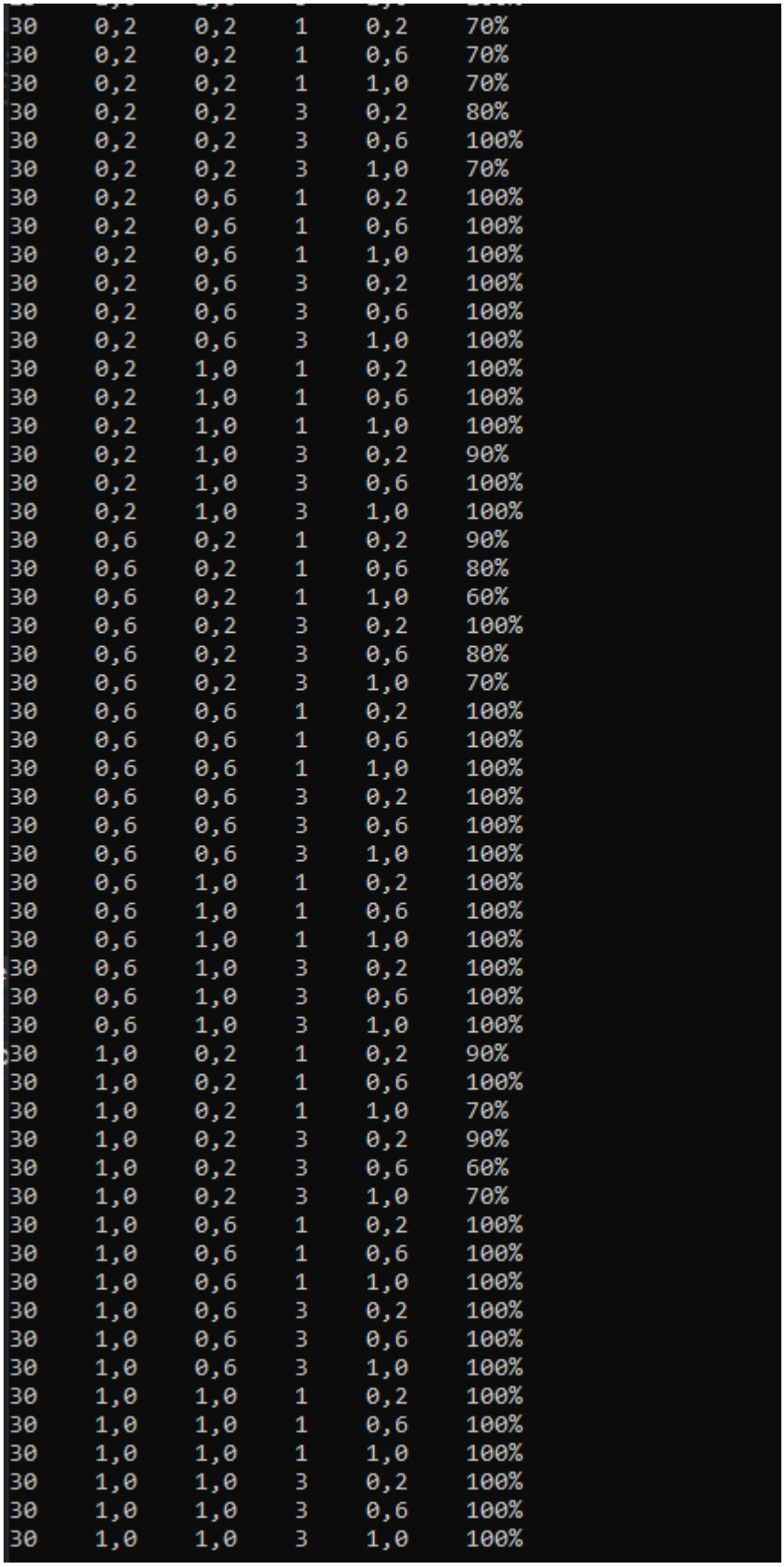
TimeOfLive	alpha	beta	Q	pho	%
1	0,2	0,2	1	0,2	0%
1	0,2	0,2	1	0,6	10%
1	0,2	0,2	1	1,0	0%
1	0,2	0,2	3	0,2	10%
1	0,2	0,2	3	0,6	10%
1	0,2	0,2	3	1,0	0%
1	0,2	0,6	1	0,2	20%
1	0,2	0,6	1	0,6	10%
1	0,2	0,6	1	1,0	10%
1	0,2	0,6	3	0,2	0%
1	0,2	0,6	3	0,6	0%
1	0,2	0,6	3	1,0	30%
1	0,2	1,0	1	0,2	30%
1	0,2	1,0	1	0,6	0%
1	0,2	1,0	1	1,0	0%
1	0,2	1,0	3	0,2	10%
1	0,2	1,0	3	0,6	20%
1	0,2	1,0	3	1,0	0%
1	0,6	0,2	1	0,2	10%
1	0,6	0,2	1	0,6	20%
1	0,6	0,2	1	1,0	0%
1	0,6	0,2	3	0,2	0%
1	0,6	0,2	3	0,6	10%
1	0,6	0,2	3	1,0	10%
1	0,6	0,6	1	0,2	10%
1	0,6	0,6	1	0,6	20%
1	0,6	0,6	1	1,0	10%
1	0,6	0,6	3	0,2	20%
1	0,6	0,6	3	0,6	0%
1	0,6	0,6	3	1,0	20%
1	0,6	1,0	1	0,2	0%
1	0,6	1,0	1	0,6	20%
1	0,6	1,0	1	1,0	10%
1	0,6	1,0	3	0,2	30%
1	0,6	1,0	3	0,6	30%
1	0,6	1,0	3	1,0	0%
1	1,0	0,2	1	0,2	10%
1	1,0	0,2	1	0,6	0%
1	1,0	0,2	1	1,0	0%
1	1,0	0,2	3	0,2	0%
1	1,0	0,2	3	0,6	0%
1	1,0	0,2	3	1,0	30%
1	1,0	0,6	1	0,2	30%
1	1,0	0,6	1	0,6	10%
1	1,0	0,6	1	1,0	10%
1	1,0	0,6	3	0,2	10%
1	1,0	0,6	3	0,6	20%
1	1,0	0,6	3	1,0	10%
1	1,0	1,0	1	0,2	30%
1	1,0	1,0	1	0,6	20%
1	1,0	1,0	1	1,0	10%
1	1,0	1,0	3	0,2	10%
1	1,0	1,0	3	0,6	50%
1	1,0	1,0	3	1,0	0%

Рисунок 5 – Результаты эксперимента при TimeOfLive = 1



15	0,2	0,2	1	0,2	60%
15	0,2	0,2	1	0,6	50%
15	0,2	0,2	1	1,0	40%
15	0,2	0,2	3	0,2	50%
15	0,2	0,2	3	0,6	50%
15	0,2	0,2	3	1,0	40%
15	0,2	0,6	1	0,2	70%
15	0,2	0,6	1	0,6	70%
15	0,2	0,6	1	1,0	60%
15	0,2	0,6	3	0,2	80%
15	0,2	0,6	3	0,6	80%
15	0,2	0,6	3	1,0	70%
15	0,2	1,0	1	0,2	100%
15	0,2	1,0	1	0,6	100%
15	0,2	1,0	1	1,0	100%
15	0,2	1,0	3	0,2	100%
15	0,2	1,0	3	0,6	100%
15	0,2	1,0	3	1,0	90%
15	0,6	0,2	1	0,2	80%
15	0,6	0,2	1	0,6	60%
15	0,6	0,2	1	1,0	70%
15	0,6	0,2	3	0,2	100%
15	0,6	0,2	3	0,6	60%
15	0,6	0,2	3	1,0	50%
15	0,6	0,6	1	0,2	90%
15	0,6	0,6	1	0,6	60%
15	0,6	0,6	1	1,0	60%
15	0,6	0,6	3	0,2	90%
15	0,6	0,6	3	0,6	80%
15	0,6	0,6	3	1,0	90%
15	0,6	1,0	1	0,2	100%
15	0,6	1,0	1	0,6	100%
15	0,6	1,0	1	1,0	90%
15	0,6	1,0	3	0,2	100%
15	0,6	1,0	3	0,6	100%
15	0,6	1,0	3	1,0	90%
15	1,0	0,2	1	0,2	90%
15	1,0	0,2	1	0,6	90%
15	1,0	0,2	1	1,0	50%
15	1,0	0,2	3	0,2	90%
15	1,0	0,2	3	0,6	30%
15	1,0	0,2	3	1,0	60%
15	1,0	0,6	1	0,2	100%
15	1,0	0,6	1	0,6	90%
15	1,0	0,6	1	1,0	90%
15	1,0	0,6	3	0,2	100%
15	1,0	0,6	3	0,6	70%
15	1,0	0,6	3	1,0	70%
15	1,0	1,0	1	0,2	100%
15	1,0	1,0	1	0,6	90%
15	1,0	1,0	1	1,0	100%
15	1,0	1,0	3	0,2	100%
15	1,0	1,0	3	0,6	100%
15	1,0	1,0	3	1,0	100%

Рисунок 6 – Результаты эксперимента при TimeOfLive = 15



30	0,2	0,2	1	0,2	70%
30	0,2	0,2	1	0,6	70%
30	0,2	0,2	1	1,0	70%
30	0,2	0,2	3	0,2	80%
30	0,2	0,2	3	0,6	100%
30	0,2	0,2	3	1,0	70%
30	0,2	0,6	1	0,2	100%
30	0,2	0,6	1	0,6	100%
30	0,2	0,6	1	1,0	100%
30	0,2	0,6	3	0,2	100%
30	0,2	0,6	3	0,6	100%
30	0,2	0,6	3	1,0	100%
30	0,2	1,0	1	0,2	100%
30	0,2	1,0	1	0,6	100%
30	0,2	1,0	1	1,0	100%
30	0,2	1,0	3	0,2	90%
30	0,2	1,0	3	0,6	100%
30	0,2	1,0	3	1,0	100%
30	0,6	0,2	1	0,2	90%
30	0,6	0,2	1	0,6	80%
30	0,6	0,2	1	1,0	60%
30	0,6	0,2	3	0,2	100%
30	0,6	0,2	3	0,6	80%
30	0,6	0,2	3	1,0	70%
30	0,6	0,6	1	0,2	100%
30	0,6	0,6	1	0,6	100%
30	0,6	0,6	1	1,0	100%
30	0,6	0,6	3	0,2	100%
30	0,6	0,6	3	0,6	100%
30	0,6	0,6	3	1,0	100%
30	0,6	1,0	1	0,2	100%
30	0,6	1,0	1	0,6	100%
30	0,6	1,0	1	1,0	100%
30	0,6	1,0	3	0,2	100%
30	0,6	1,0	3	0,6	100%
30	0,6	1,0	3	1,0	100%
30	1,0	0,2	1	0,2	90%
30	1,0	0,2	1	0,6	100%
30	1,0	0,2	1	1,0	70%
30	1,0	0,2	3	0,2	90%
30	1,0	0,2	3	0,6	60%
30	1,0	0,2	3	1,0	70%
30	1,0	0,6	1	0,2	100%
30	1,0	0,6	1	0,6	100%
30	1,0	0,6	1	1,0	100%
30	1,0	0,6	3	0,2	100%
30	1,0	0,6	3	0,6	100%
30	1,0	0,6	3	1,0	100%
30	1,0	1,0	1	0,2	100%
30	1,0	1,0	1	0,6	100%
30	1,0	1,0	1	1,0	100%
30	1,0	1,0	3	0,2	100%
30	1,0	1,0	3	0,6	100%
30	1,0	1,0	3	1,0	100%

Рисунок 7 – Результаты эксперимента при TimeOfLive = 30

Как видно из результатов, наибольшее влияние на точность алгоритма оказывает TimeOfLive. При равном TimeOfLive, варьирование alpha и beta оказывает второе по величине влияние, phi - третье и Q - четвертое.

4.4 Вывод

В данной части работы экспериментально было подтверждено значительное превосходство муравьиного алгоритма в скорости работы над алгоритмом полного перебора, была проведена

Заключение

В данной лабораторной работе были разработаны параллельные алгоритмы Винограда. Они были реализованы, а также были проведены эксперименты по замеру времени работы реализованных алгоритмов и проведены сравнения алгоритмов по результатам эксперимента. Цель работы достигнута, все задачи выполнены.

Список литературы

[1] Муравьиный алгоритм [Электронный ресурс]. Режим доступа: (дата обращения - 01.12.2020) Свободный. URL: http://www.machinelearning.ru/wiki/index.php?title=Муравьиные_алгоритмы

[2] API.NET. Пространство имен System.Threading [Электронный ресурс]. Режим доступа: (дата обращения - 01.12.2020) Свободный. URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.threading?view=netcore-3.1

[3] Visual Studio [Электронный ресурс]. Режим доступа: (дата обращения - 01.12.2020) Свободный. URL: https://visualstudio.microsoft.com/ru/