ВЕСТНИК ПЕРМСКОГО УНИВЕРСИТЕТА
2016 Математика. Механика. Информатика Вып. 4(35)

УДК 519.176

Анализ точности и времени решения задачи коммивояжера с помощью "антижадного" алгоритма

А. А. Чусовлянкин[1], **В. В. Морозенко**[2]
[1]Национальный исследовательский университет "Высшая школа экономики" (Пермский филиал)
Россия, 614070, Пермь, ул. Студенческая, 38
lixich@mail.ru; 8-951-924-29-77

[2]Национальный исследовательский университет "Высшая школа экономики" (Пермский филиал)
Россия, 614070, Пермь, ул. Студенческая, 38
v.morozenko@mail.ru; 8-912-888-70-74

Предложен новый "антижадный" алгоритм для решения задачи коммивояжера, имеющий меньшую погрешность, чем известные приближенные полиномиальные алгоритмы. Идея "антижадного" алгоритма заключается в том, что из графа последовательно удаляются ребра наибольшей длины при одновременном соблюдении для оставшегося графа двух правил.
Во-первых, из каждой его вершины должно выходить, как минимум, два ребра.
Во-вторых, в нем не должно возникать циклов, состоящих менее чем из n ребер, где n – количество вершин в исходном графе. Проведено исследование работы трех алгоритмов: "антижадный", "жадный" и алгоритм Кристофидеса, – для евклидовых и неевклидовых графов. Получены статистические данные о погрешности и времени работы алгоритмов, которые демонстрируют значительное превосходство "антижадного" алгоритма в точности, в особенности для неевклидовых графов.

Ключевые слова: *задача коммивояжера; "антижадный" алгоритм; "жадный" алгоритм; алгоритм Кристофидеса; евклидовы графы; неевклидовы графы; погрешность.*

DOI: 10.17072/1993-0550-2016-4-68-75

Введение

Задача коммивояжера считается одной из наиболее известных задач комбинаторной оптимизации, возникающей в обширном классе приложений, включая распознавание траекторий и образов, построение оптимальных схем движения и др. [1]. В ней требуется найти самый короткий замкнутый маршрут, проходящий через все заданные точки только один раз. В теории графов задача сводится к поиску кратчайшего гамильтонова цикла в полном графе [2]. Задача относится к классу NP-полных задач [3]. Все известные точные алгоритмы для ее решения имеют большие временные затраты, поскольку выполняют полный перебор всех возможных вариантов. Поэтому разработка новых эффективных методов для решения задачи коммивояжера актуальна по сей день [4].

Предложен новый подход для решения задачи коммивояжера, а именно, "антижадный" алгоритм, который, в отличие от "жадного" алгоритма, исключает выбор длинных ребер при построении гамильтонова цикла [5]. Исследуемый алгоритм имеет более высокую точность в сравнении с "жадным" алгоритмом и алгоритмом Кристофидеса, что экспериментально продемонстрировано на евклидовых и неевклидовых графах с разным количеством вершин, но время работы "антижадного" алгоритма больше, чем у выбранных приближенных алгоритмов.

© Чусовлянкин А. А., Морозенко В. В., 2016

Объектами изучения данной работы являются приближенные алгоритмы для решения задачи коммивояжера: "антижадный", "жадный" и алгоритм Кристофидеса.

Предмет изучения – погрешность и время работы алгоритмов для разных типов графов.

Цель работы – предложить и исследовать новый подход к решению задачи коммивояжера на основе "антижадного" алгоритма.

1. Существующие алгоритмы

"Задача коммивояжера является NP-полной, поэтому алгоритмы решения задачи делятся на две группы: точные и приближенные. Все точные алгоритмы фактически представляют собой оптимизированный полный перебор вариантов.

В некоторых случаях эти алгоритмы достаточно быстро находят решения, но в общем случае приходится перебирать $n!$ циклов" [3].

Задачу коммивояжера можно решить различными методами. Все быстрые методы – это методы эвристические, основанные на сокращении полного перебора. Они имеют погрешность и чаще всего находят приближенный вариант решения, причем для полиномиальных алгоритмов погрешность может быть сколь угодно большой.

1.1. Приближенные алгоритмы

Выделяют следующую группу "быстрых" алгоритмов, дающих приближенный ответ, время работы которых полиномиально зависит от количества вершин в графе:

1. "Жадный" алгоритм (метод ближайшего соседа) – это алгоритм нахождения кратчайшего расстояния путем добавления самого короткого среди еще не выбранных ребер [6].

2. Алгоритм Кристофидеса (метод минимального остовного дерева – так называемый "деревянный" алгоритм). Идея данного алгоритма заключается в следующем: строится кратчайшее остовное дерево и дублируются все его ребра. В результате получается граф с вершинами только четной степени. В нем ищется эйлеров цикл, из которого затем удаляется повторное вхождение каждой вершины, кроме первой и последней (они должны совпадать).

Полученный цикл является результатом работы алгоритма [4, 7].

1.2. Точные алгоритмы

Иногда на практике применяют более точные методы, имеющие экспоненциальную сложность, но все они работают намного дольше:

1. Полный перебор – это метод решения задачи путем перебора всех гамильтоновых циклов, число которых в n-вершинном графе равно $n!$

2. Метод ветвей и границ (метод Литтла). В основе данного метода лежит идея последовательного разбиения множества допустимых решений на подмножества. Заведомо не содержащие оптимальных решений подмножества отбрасываются. В худшем случае алгоритму придется осуществлять полный перебор [3].

2."Антижадный" алгоритм

Идея "антижадного" алгоритма заключается в том, что из графа последовательно удаляются ребра наибольшей длины при одновременном соблюдении двух правил: в текущем графе, во-первых, из каждой вершины должно выходить, как минимум, два ребра; во-вторых, не должно образовываться циклов менее, чем из n ребер.

Первоначально создается квадратная матрица размера $n \times n$ – матрица начального состояния ребер, где "0" означает исключение (удаление) ребра, "2" – добавление (фиксирование) ребра, а первоначально все ребра имеют состояние "1".

Далее последовательно производится поиск самого длинного ребра в графе из оставшихся ребер. Рассматриваются два варианта: либо удаление этого ребра, либо его фиксирование (т.е. добавление ребра в искомый гамильтонов цикл).

Удаление данного ребра предпочтительнее, так как оно является самым длинным в текущем графе. Если удаление невозможно, то рассматривается вариант фиксирования ребра. Если же фиксирование ребра тоже невозможно, то это означает, что предыдущее действие – удаление либо фиксирование – выполнено неверно. В этом случае производится откат к предыдущему состоянию текущего графа, где был сохранен альтернативный вариант действия.

Альтернативный вариант для отката сохраняется на этапе выбора между фиксированием ребра и его удалением. Если возможно и удаление, и фиксирование ребра, то выполня-

ется удаление ребра, а его фиксирование сохраняется как альтернативный вариант.

Поиск максимального ребра производится среди ребер, которые имеют начальное состояние "1". Поэтому, если ребро было зафиксировано или удалено, то в набор ребер, среди которых производится поиск максимального ребра, данное ребро не входит.

Таким образом, поиск максимального ребра будет производиться до тех пор, пока все ребра не будут иметь состояние "0" ("удаленное") или "2" ("фиксированное") [5].

Псевдокод данного алгоритма для неориентированного графа представлен ниже:-

```
public class Edge
{
 int[] vertices;                //вершины ребра
 int weight;  //вес ребра
 int state;    //состояние ребра

 Edge(int _vertex1, int _vertex2, int _weight, int _state)
 {
  this.vertices = new int[2] {_vertex1, _vertex2};
  this.weight = _weight;
  this.state = _state;
 }
}
Antigreedy(int size, int[][] weightMatrix)
{
 List<Edge> currentGraph = new List<Edge>();
 for (int i=0; i<weightMatrix.Length; i++)
  for (int j=0; j<weightMatrix[i].Length; j++)
   if (j != i)
        currentGraph.Add(new Edge(i, j, weightMatrix[i][j], 1)); //все
ребра = "1"
   List<List<Edge>> Log = new List<List<Edge>>();
   Log.Add(currentGraph); //журнал откатов

 while (Log.Last().Where(x => x.state == 1). Count() > 0) //поиск ребра с
макс. весом и состоянием "1"
 {
  currentGraph = Log.Last();
  var edge = Log.Last().Where(x => x.state == 1). OrderByDescending(x =>
x.weight).First();

  List<Edge> deleteEdge = new List<Edge>();
  deleteEdge = currentGraph.Clone();
  deleteEdge.Find(edge.vertices).state = 0;

  List<Edge> fixEdge = new List<Edge>();
  fixEdge = currentGraph.Clone();
  fixEdge.Find(edge.vertices).state = 2;

  Processing(size, ref deleteEdge);
  Processing(size, ref fixEdge);
  if (CheckGraph(fixEdge, size))
        Log.Add(fixEdge); //для отката
  if (CheckGraph(deleteEdge, size))
        Log.Add(deleteEdge); //для отката

  Log.Remove(currentGraph); //текущий граф пройден, поэтому удаляет-
ся из журнала откатов
 }
 int HamiltonianCycle = 0;
 foreach (var edge in Log.Last())
  if (edge.state == 2)
        HamiltonianCycle += edge.weight;
 return HamiltonianCycle;
}
```

Обработка графа (Processing) состоит из трех этапов:

1. Удаление возможности появления мелких циклов (RemovingSmallCycles).
2. Удаление ненужных ребер в тех случаях, когда среди инцидентных ей ребер уже есть два фиксированных ребра.
3. Фиксирование ребер, если у конкретной вершины осталось всего два инцидентных ей ребра.

Псевдокоды данных функций расположены ниже:

```
public static void Processing(int size, ref List<Edge> graph)
{
 do
 {
  List<Edge> copyGraph = new List<Edge>();
  copyGraph = graph.Clone();
  for (int i=0; i<size; i++)по всем вершинам
   {
        if ((graph.Where(x => x.vertices.Contains(i) && x.state ==
2).Count() == 2) && (graph.Where(x => x.vertices.Contains(i) && x.state ==
1).Count() > 0)) //если у вершины есть 2 зафиксированных ребра и
ребра с начальным состоянием, то последние необходимо удалить
        graph.FindAll(x => x.vertices.Contains(i) && x.state !=
2).ForEach(x => x.state = 0);
        else
        if (graph.Where(x => x.vertices.Contains(i) && x.state !=
0).Count() == 2)//если у вершины осталось только 2 ребра, то их необ-
ходимо зафиксировать
        graph.FindAll(x => x.vertices.Contains(i) && x.state !=
0).ForEach(x => x.state = 2);
   }
  if (graph.Where(x => x.state == 2).Count() < size - 1) //если фиксирован-
ных ребер меньше, чем количество вершин - 1, тогда необходимо
удалить возможности появления мини-циклов
   foreach (var edge in graph.Where(x => x.state == 2))
   {
    bool[] visited = new bool[size];
    for (int i = 0; i < size; i++)
        visited[i] = false;
    visited[edge.vertices[0]] = true;
    visited[edge.vertices[1]] = true;
    RemovingSmallCycles(edge.vertices[0], edge.vertices[1], visited, size,
ref graph);
   }

} while (graph != copyGraph); //до тех пор пока состояния ребер не ста-
билизируются

public static void RemovingSmallCycles(int nStart, int nEnd, bool[] visited,
int size, ref List<Edge> graph)
{
 int i = 0;
 do
 {
  if ((!visited[i]) && (graph.Count(x => x.state == 2 &&
x.vertices.Contains(nEnd) && x.vertices.Contains(i)) != 0))
  {
   visited[i] = true;
   graph.Find(x => x.vertices.Contains(i) &&
x.vertices.Contains(nStart)).state = 0;
   nEnd = i; //к следующей вершине
   i = -1;
  }
  i++;
 } while (i < size);
}
public static bool CheckGraph(List<Edge> graph, int size)
{
 bool result = true;
```

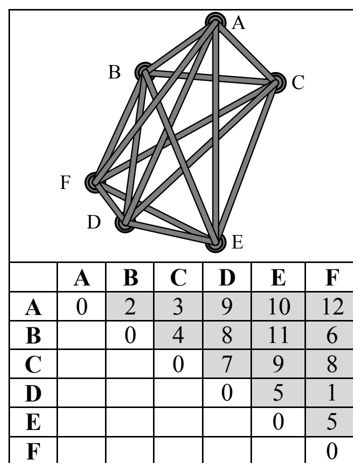
```
for (int i=0; i<size; i++)
  if ((graph.Where(x => x.vertices.Contains(i) && x.state == 2).Count() > 2)
|| (graph.Where(x => x.vertices.Contains(i) && x.state != 0).Count() < 2))
 result = false;
 return result;
}
```

Проверка графа (CheckGraph) заключается в том, чтобы убедиться, что в текущем графе у каждой вершины осталось, как минимум, два ребра с начальным или фиксированным состоянием, а также у каждой вершины осталось не больше двух инцидентных ей фиксированных ребер. И, если количество фиксированных ребер равно числу вершин в графе, то нужно убедиться, что построен гамильтонов цикл, а не набор нескольких коротких циклов. В противном случае выполняется откат к предыдущему состоянию графа.

3. Пример работы алгоритма

Рассмотрим пример работы "антижадного" алгоритма с конкретным графом с 6 вершинами. Сам граф и его матрица расстояний изображены на рис. 1.

В матрице расстояний элементы ниже главной диагонали не показаны, поскольку исходный граф является неориентированным и его матрица расстояний симметрична.



	A	B	C	D	E	F
A	0	2	3	9	10	12
B		0	4	8	11	6
C			0	7	9	8
D				0	5	1
E					0	5
F						0

Рис. 1. *Исходный граф и его матрица расстояний*

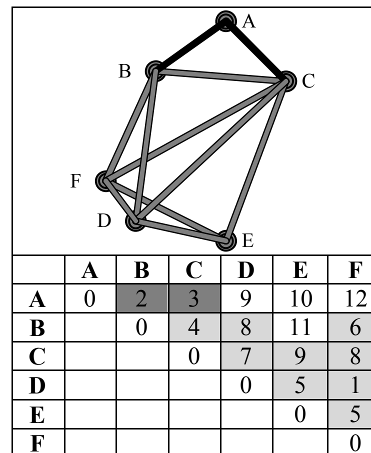
В матрице расстояний в начальный момент все ребра считаются находящимися в начальном состоянии. В процессе работы алгоритма состояние ребра может измениться на "удаленное" или "фиксированное".

Разные состояния ребер будут выделяться разными цветами в матрице расстояний, как это показано на рис. 2.

	Ребро
	"Удаленное"
	Начальное состояние
	"Фиксированное"

Рис. 2. *Условные обозначения для разных состояний ребер*

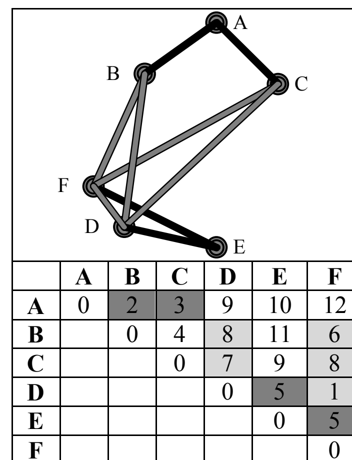
Сначала алгоритм удалит последо-вательно самые длинные ребра AF, BF, AE, AD. Их статус изменится на "удаленные". Поскольку у вершины A осталось два инцидентных ей ребра AB и AC с начальным статусом, то эти ребра получают статус "фиксированных" и добавляются в искомый гамильтонов цикл (рис. 3).



	A	B	C	D	E	F
A	0	2	3	9	10	12
B		0	4	8	11	6
C			0	7	9	8
D				0	5	1
E					0	5
F						0

Рис. 3. *Фиксирование ребер AB, BC*

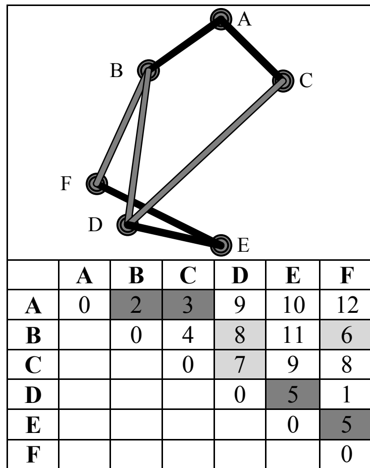
Ребро BC образовывает мини-цикл с ребрами AB и AC, поэтому оно удаляется. Продолжая удалять самые длинные ребра с начальным статусом, алгоритм удалит ребро CE, изменив его статус на "удаленное". После чего в текущем графе остается всего два ребра ED и EF, инцидентных вершине E. Оба этих ребра получают статус "фиксированные" и добавляются в искомый гамильтонов цикл (рис. 4).



	A	B	C	D	E	F
A	0	2	3	9	10	12
B		0	4	8	11	6
C			0	7	9	8
D				0	5	1
E					0	5
F						0

Рис. 4. *Фиксирование ребер ED, EF*

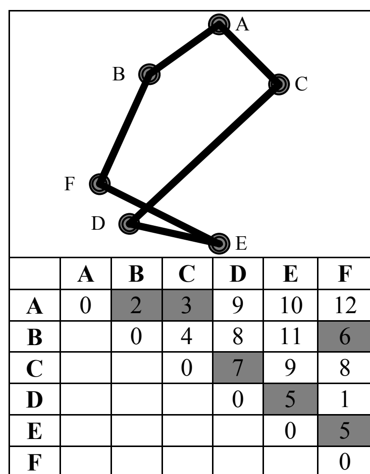
Ребро FD удаляется, так как оно образует цикл с "фиксированными" ребрами ED и EF. Также продолжается и удаление максимальных ребер, в данном случае удаляется ребро FC (рис. 5).



	A	B	C	D	E	F
A	0	2	3	9	10	12
B		0	4	8	11	6
C			0	7	9	8
D				0	5	1
E					0	5
F						0

Рис. 5. *Удаление ребер FD, FC*

Поскольку из вершин F и C выходят два ребра, то ребра BF и CD фиксируются. Так как вершине D инциденты уже два "фиксированных" ребра CD и DE, то инцидентное ей ребро DB удаляется (рис. 6).



	A	B	C	D	E	F
A	0	2	3	9	10	12
B		0	4	8	11	6
C			0	7	9	8
D				0	5	1
E					0	5
F						0

Рис. 6. *Удаление ребра DB и фиксирование ребер BF, CD*

В данном примере "антижадный" алгоритм нашел гамильтонов цикл длины 28, изображенный на рис. 6. Однако кратчайший цикл для этого графа имеет длину 26 и изображен на рис. 7.

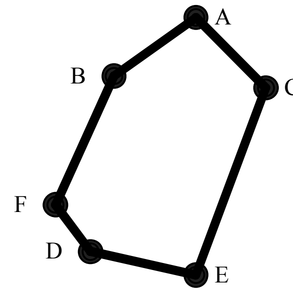


Рис. 7. *Кратчайший гамильтонов цикл*

Важно отметить, что "жадный" алгоритм для данного графа выдает ответ 29, а алгоритм Кристофидеса – 30.

Таким образом "антижадный" алгоритм демонстрирует лучшую точность по сравнению с конкурирующими алгоритмами.

4. Эффективность алгоритма

Для оценки качества работы "антижадного" алгоритма проведено исследование его времени работы и погрешности. Также реализованы "жадный" алгоритм, алгоритм Кристофидеса и метод ветвей и границ для получения верного ответа к тестовым заданиям. Генерация неориентированных графов производилась с помощью встроенного датчика случайных чисел в среде разработке Visual Studio 2013 на языке программирования C# (ПК с тактовой частотой 1.7 GHz). Для каждого фиксированного количества вершин [4, 18] было сгенерировано 100 тыс. случайных графов, общее количество тестируемых графов – 3 млн.

4.1. Погрешность

Погрешность "антижадного" алгоритма оказалась зависящей от типа графа.

4.1.1. Евклидовы графы

Генерация евклидовых графов производилась путем случайного выбора длин ребер из заданного диапазона [a, $2a$], где a – произвольное натуральное число от 2 до 1000, что гарантирует выполнение неравенства треугольника для каждого треугольника в графе.

Произведено исследование погрешности приближенных алгоритмов, где для различного количества вершин особенно выигрышно смотрится "антижадный" алгоритм, средняя погрешность которого составляет 1 %, а максимально полученная погрешность – 23 %.

На рис. 8, 9 представлены графики зависимости средней и максимальной погрешностей от количества вершин.

Рис. 8. *Средняя погрешность алгоритмов для евклидовых графов*

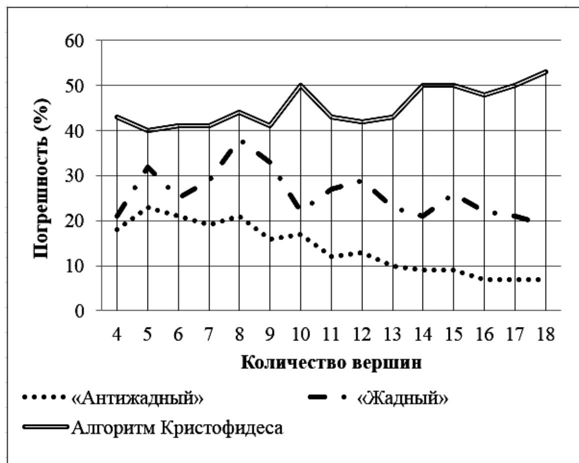


Рис. 9. *Максимальная погрешность алгоритмов для евклидовых графов*

4.1.2. Неевклидовы графы

Генерация неевклидовых графов производилась путем случайного выбора длин ребер из заданного диапазона [2, 1000], в котором проверялось наличие хотя бы одного неевклидового треугольника. В результате такой генерации средняя доля неевклидовости графов составила 50 % [8].

На рис. 10, 11 представлены графики зависимости средней и максимальной погрешностей от количества вершин. Полученная средняя погрешность "антижадного" алгоритма для различного количества вершин не превышает 10 % (показатель стабилизируются при увеличении количества вершин), максимальная же полученная погрешность составляет 386 % при n=5, но при росте количества вершин уже при n=18 максимальная погрешность составляет 63 %.

Рис. 10. *Средняя погрешность алгоритмов для неевклидовых графов*

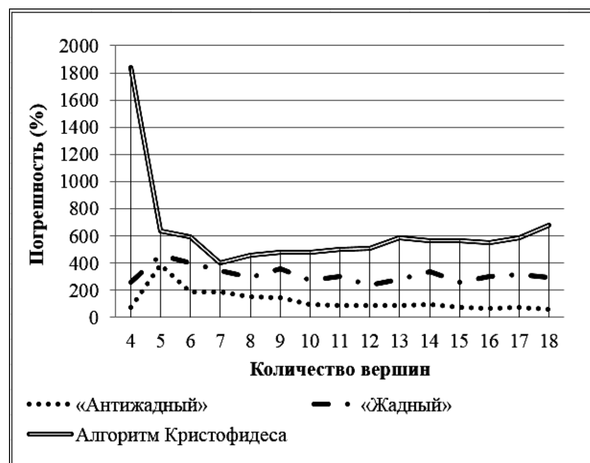


Рис. 11. *Максимальная погрешность алгоритмов для неевклидовых графов*

4.2. Время работы

В результате проведенного исследования также были получены данные о времени работы алгоритмов, которые сопоставимы как для евклидовых, так и для неевклидовых графов. Обнаружено, что "антижадный" алгоритм работает медленнее, чем известные приближенные алгоритмы (алгоритм Кристофидеса и "жадный" алгоритм), для которых среднее время работы на различных графах с указанными выше характеристиками не превышает 0,02 мс, а максимальное составляет 10 мс. Что же касается "антижадного" алгоритма, то среднее и максимальное время его работы при увеличении количества вершин возрастает значительно быстрее. Например, для графов с числом вершин $n = 18$ среднее

время работы составляет 20 мс, а максимальное – 40 с. Стоит отметить, что среднее время работы метода ветвей и границ для $n = 18$ все равно заметно выше и составляет 110 мс, максимальное же время хотя и больше, но лишь на 11 с и составляет 51 с. Более подробные данные о времени работы исследуемых алгоритмов представлены на рис. 12, 13.

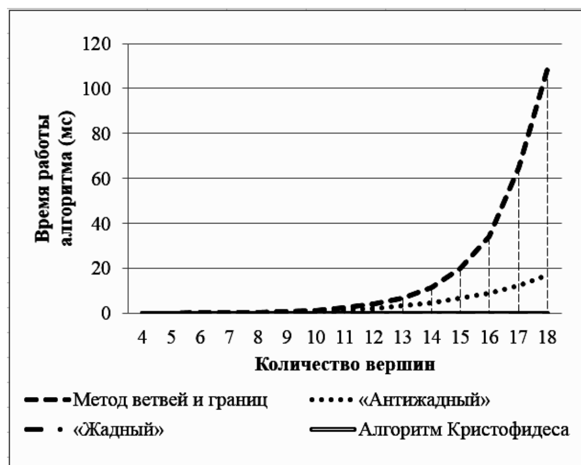


Рис. 12. *Среднее время работы алгоритмов*

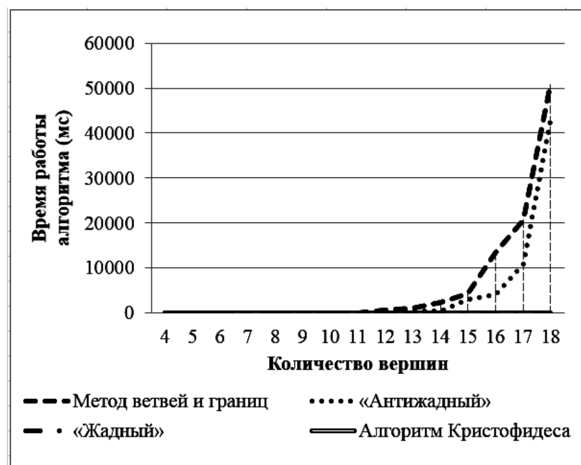


Рис. 13. *Максимальное время работы алгоритмов*

Заключение

Исследован новый метод решения задачи коммивояжера – "антижадный" алгоритм. Проведенное исследование погрешности данного метода в сравнении с другими алгоритмами свидетельствует о том, что главное преимущество данного алгоритма – низкая средняя относительная погрешность этого алгоритма по сравнению с известным "жадным" алгоритмом и алгоритмом Кристофидеса как для евклидовых, так и для неевклидовых графов. Причем, "антижадный" алгоритм особенно выигрышно выглядит при работе с неевклидовыми графами.

Отметим еще, что время работы данного алгоритма больше, чем у известных приближенных алгоритмов, но меньше, чем у метода ветвей и границ, который используется для точного решения задачи коммивояжера.

Список литературы

1. *Дулькейт В.И., Файзуллин Р.Т.* Приближенное решение задачи коммивояжера методом рекурсивного построения вспомогательной кривой // Вычислительные методы в дискретной математике. / Омск: Изд-во Омского гос. тех. ун-та, 2009. № 1 (3). С.72–78.
2. Задача коммивояжера // Институт математики им. С.Л. Соболева СО РАН. URL: http://math.nsc.ru/LBRT/k5/ORMMF/TSPr.pf свободный (дата обращения: 19.10.2016).
3. *Оре О.* Графы и их применение / пер. с англ., под ред. И.М. Яглома. М.: Мир, 1965. 174 с.
4. *Кузюрин Н.Н. Фомин С.А.* Сложность вычислений и анализ алгоритмов // DISCOP-AL. М.: МФТИ, 2007. 312 с. URL:http://discopal.ispras.ru/img_auth.php/f/f4/Book-advanced-algorithms.pdf, свободный. (дата обращения: 19.10.2016).
5. *Чусовлянкин А.А.* "Антижадный" алгоритм для решения задачи коммивояжера М.: МИЭМ НИУ ВШЭ, 2016. С. 10–11.
6. *Gutin G., Punnen A. P.* The Traveling Salesman Problem and Its Variations Springer // Combinatorial Optimization. Vol. 12. Boston: Kluwer Academic Publishers, 2002. P. 84–85.
7. *Stencek J.* Traveling salesman problem. JAMK University of Applied sciences: Bachelor's Thesis, 2013. P. 25, 31.
8. *Чусовлянкин А.А., Морозенко В.В.* О погрешности алгоритма Кристофидеса для случайных неевклидовых графов // Пермь: ПГНИУ, 2015. С. 338–342.

Accuracy and problem time analysis for solving the traveling salesman problem with the "anti-greedy" algorithm

A. A. Chusovliankin, V. V. Morozenko
Higher School of Economics (Perm branch); 38, Studencheskaya st., Perm, 614990, Russia
lixich@mail.ru; 8-951-924-29-77
v.morozenko@mail.ru; 8-912-888-70-74

In this paper a new "anti-greedy" algorithm for the traveling salesman problem is considered. The idea of the "anti-greedy" algorithm consists in consequent elimination of the longest edges from a graph according to two rules: each node of the graph should have at least two incident edges; the graph should not have cycles with less than n edges. This algorithm finds more accurate solutions in comparison with well-known polynomial algorithms, especially for non-Euclidean graphs.

Keywords: *traveling salesman problem;* "*anti-greedy*" *algorithm; Euclidean graph; non-Euclidean graph; accuracy.*