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»

Лабораторная работа № 6
по дисциплине «Анализ Алгоритмов»

Тема Методы решения задачи коммивояжёра

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-52Б

Преподаватели Волкова Л.Л., Строганов Д.В.

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача коммивояжёра – одна из самых популярных и старинных задач комбинаторной оптимизации. Впервые задача была исследована в 18-м веке ирландским математиком Уильямом Роуан Гамильтоном и британским математиком Томасом Пенингтон Киркманом в их книге [2] "Graph Theory".

Целью данной работы является рассмотрение методов решения задачи коммивояжёра.

Для достижения этой цели требуется решить следующие задачи:

— постановка задачи коммивояжёра;

— рассмотрение методов полного перебора и муравьиной колонии;

— разработка рассмотренных алгоритмов решения задачи;

— реализация разработанных алгоритмов;

— проведение параметризации для алгоритма на основе метода муравьиной колонии.

1 Аналитическая часть

1.1 Постановка задачи

Пусть дан неориентированный полносвязный граф $G_n = (V_n, E_n)$ с $n = |V_n|$ узлов и $m = |E_n| = C_n^2$ рёбер. Ребро с концами i и j обозначается как ij или как (i, j).

Пусть также даётся функция $f : E_n- > \mathbf{R}$ разметки графа. Каждая значение функции называется длиной ребра (i, j) и пусть длиной цикла называется сумма длин всех рёбер входящих в него.

Гамильтоновым циклом в неориентированном графе называется цикл графа, проходящий через каждую его вершину [1].

Тогда задача поиска гамильтонова цикла с минимальной длинной называется симметричной задачей коммивояжёра [3].

1.2 Методы решения

В данной работе будут рассмотрены 2 метода решения задачи коммивояжёра: метод полного перебора и метод муравьиной колонии

1.2.1 Метод полного перебора

Под методом полного перебора подразумевается решение задачи, при котором формируются все возможные гамильтоновы циклы, просчитывается их длины и выбирается тот, у которого эта длина минимальна. Метод полного перебора в результате даёт оптимальное решение, тем не менее с увеличением числа узлов в графе сложность метода существенно возрастает, так как для графа из n узлов существует $\frac{(n-1)!}{2}$ [3] гамильтоновых циклов. Таким образом фактическая трудоёмкость алгоритма полного перебора возрастает не медленнее чем по факториальной зависимости.

1.2.2 Метод муравьиной колонии

Метод муравьиной колонии – один из стохастических методов, применяющихся для решения задач комбинаторной оптимизации, в частности для решения задачи коммивояжёра.

Метод основан на способе обмена информацией у муравьев, при котором они передают информацию о привлекательности путей с помощью меток(феромонов).

Не считая этап инициализации метод состоит из основного цикла, который делится на 2 части [4].

Конструирование путей муравьёв

На данном этапе каждый из m муравьёв инициализируется с циклом с единственным начальным узлом c_p и на каждом шаге стохастически выбирает узел из тех, которые он не посещал, и добавляет в свой цикл.

Выбор узла на очередном этапе цикла производится с вероятностью, вычисляемой по формуле (1.1) [4].

$$p(e^i_j) = \frac{\tau^{\alpha}_{ij} * f(e^i_j)^{\beta}}{\sum \tau^{\alpha}_{ij} * f(e^i_j)^{\beta}}, \tag{1.1}$$

где

— i – номер последнего добавленного узла в пути;

— j – номер рассматриваемого узла из не посещённых муравьём узлов;

— e^i_j – ребро между i-м и j-м узлами;

— τ_{ij} – количество феромона на ребре между i-м и j-м узлами;

— $f(e^i_j)$ – функция выражающая "привлекательность"ребра между i-м и j-м узлами для муравья. В случае задачи коммивояжёра эта величина обратна пропорциональная длине ребра и выражается формулой $\frac{1}{L(e^i_j)}$;

— α и $\beta \in (0, 1)$ – параметры определяющие влияние феромонов и длины пути на вероятность выбора ребра. При $\alpha = 0$ выбор муравья полностью жадный и определяется длиной ребра, а при $\beta = 0$ выбор муравья полностью стайный и определяется только количеством феромона на ребре.

В результате этапа каждый муравей формирует гамильтонов цикл по графу. Каждый из сформированных циклов сравнивается с лучшим на текущий момент.

Обновление феромонов

На данном этапе формируются положительные и отрицательные обратные связи. Каждый муравей проходя по своему пути распыляет на нём феромоны, при этом чем длиннее путь, тем меньше распылённое количество феромонов. Распылённое число феромонов для k-го муравья рассчитывается по формуле (1.2) [5].

$$\triangle\tau^k_{ij} = \begin{cases} \frac{Q}{L_k}, & \text{если ребро (i, j) есть в цикле муравья,} \\ 0, & \text{иначе,} \end{cases} \tag{1.2}$$

где L_k – длина цикла k-го муравья, а Q – константа.

Для того, чтобы количество феромона не было решающим фактором после нескольких циклов в модели есть испарение феромона, которое в конце цикла уменьшает концентрацию феромона на каждом ребре. Таким образом длина рёбер продолжает влиять на решение и одновременно наиболее длинные пути становятся менее привлекательными для муравья. Испарение феромона рассчитывается по формуле (1.3) [5].

$$\tau_{ij} = (1-p)\tau_{ij}, \tag{1.3}$$

где p – коэффициент испарения.

Таким образом феромон изменяется по формуле (1.4)

$$\tau_{ij} = (1-p)\tau_{ij} + \sum_{k=0}^{m} \triangle\tau_{ij}^{k}. \tag{1.4}$$

Элитарные муравьи

Элитарные муравьи – модификация муравьиного алгоритма. Элитарные муравьи на каждой итерации идут по лучшему найденному пути и распыляют по нему феромоны, таким образом увеличиваю привлекательность его частей для других муравьёв [5]. С данной модификацией расчёт феромона на каждой итерации происходит по формуле (1.5).

$$\tau_{ij} = (1-p)\tau_{ij} + \sum_{k=0}^{m} \triangle\tau_{ij}^{k} + me * \frac{Q}{L_b}, \tag{1.5}$$

где me – количество элитных муравьёв, L_b – длина лучшего цикла.

Вывод

В результаты аналитического раздела была рассмотрена формальная постановка задачи коммивояжёра, а также рассмотрены методы её решения.

2 Конструкторская часть

2.1 Требования к программному обеспечению

К разрабатываемой программе предъявлен ряд требований:

Входные данные: Взвешенный неориентированный граф, заданный матрицей стоимостей.

Выходные данные: Оптимальный гамильтонов цикл, в случае метода полного перебора, субоптимальный гамильтонов цикл в случае муравьиного алгоритма.

2.2 Алгоритм полного перебора

2.2.1 Метод генерации перестановок

Основной частью алгоритма полного перебора является алгоритм генерации перестановок, так как гамильтонов путь, как и гамильтонов путь, являются перестановками множества узлов графа. Существуют множество методов генерации перестановок [6] как рекурсивных, так и итеративных. В данной работе будет использоваться нерекурсивный метод Хипа (Heap's method) разработанный в [7].

2.2.2 Алгоритм

На рисунке 2.1 представлен алгоритм полного перебора для решения задачи коммивояжёра.

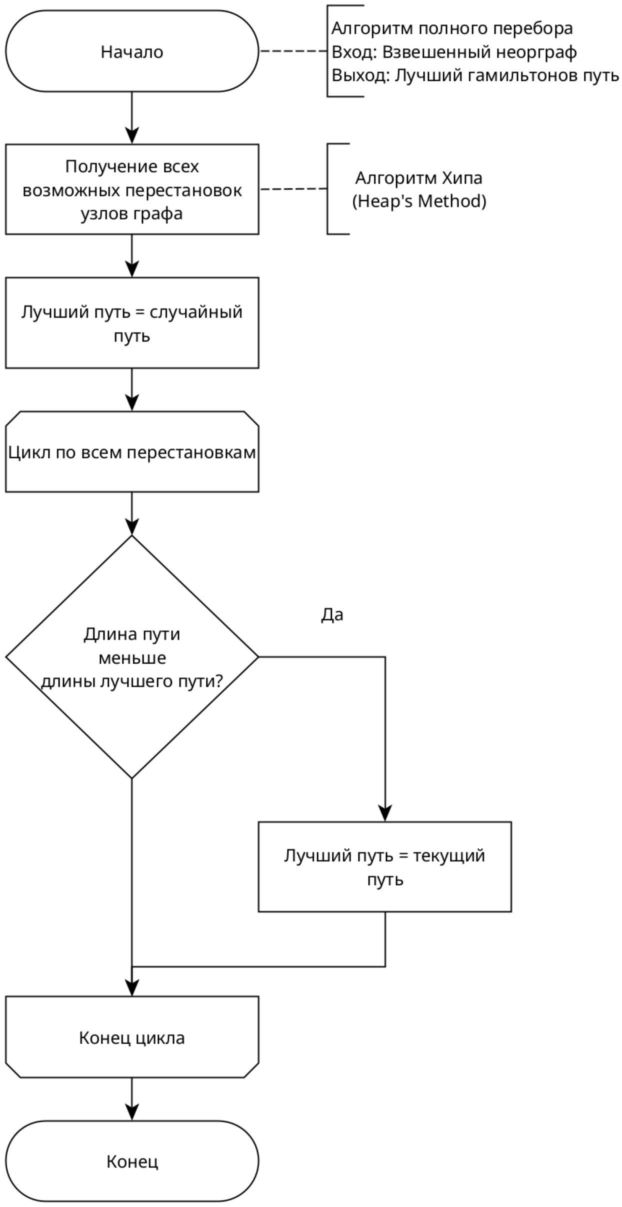

Рисунок 2.1 — Схема алгоритма полного перебора

2.3 Муравьиный алгоритм

На рисунках 2.2 и 2.3 представлен алгоритм муравьиного алгоритма для неорграфа с элитными муравьями.

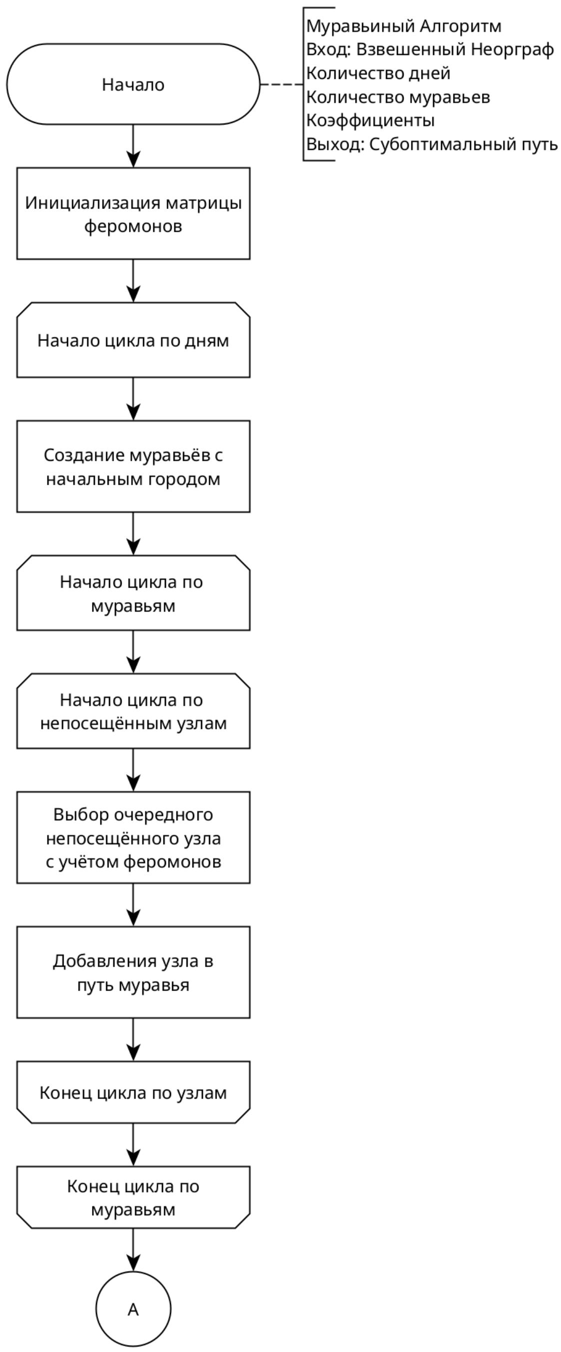

Рисунок 2.2 — Схема муравьиного алгоритма (начало)

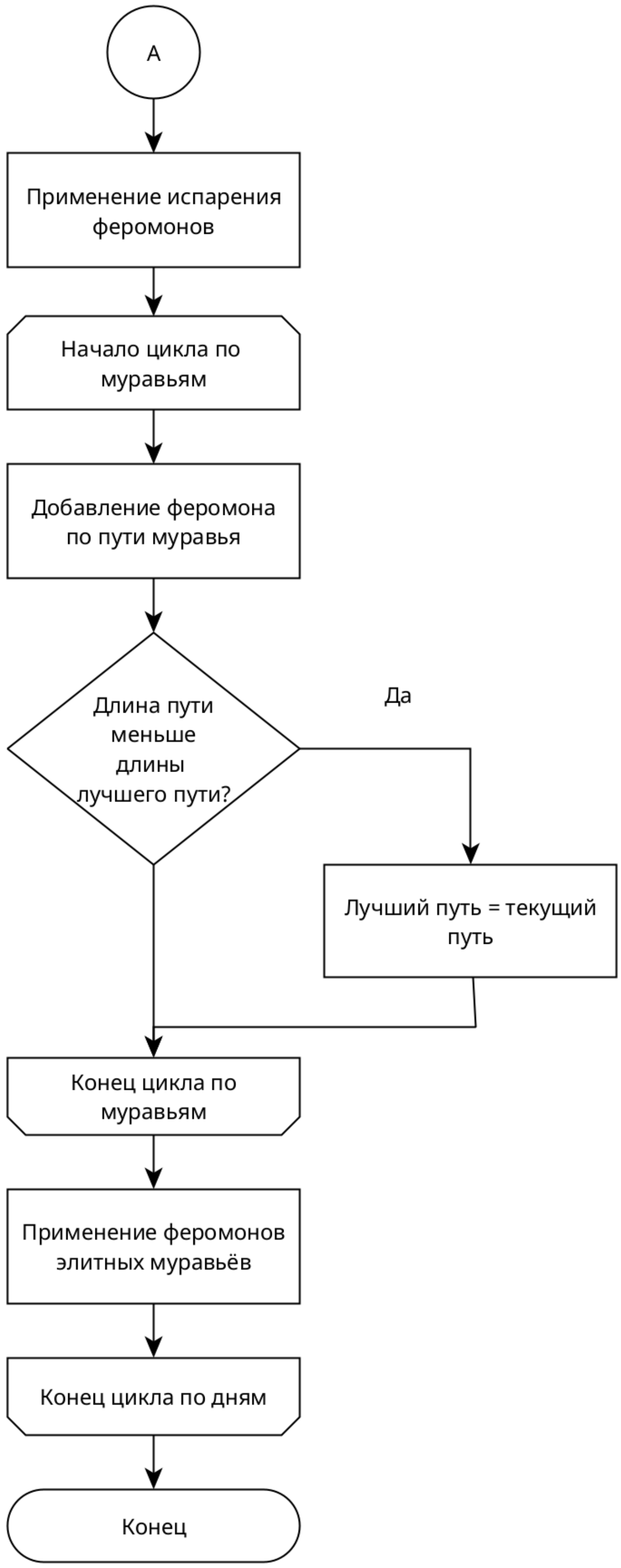

Рисунок 2.3 — Схема муравьиного алгоритма (конец)

2.4 Вывод

В результате конструкторской части были определены требования к ПО, а также разработаны схемы алгоритмов полного перебора и муравьиного алгоритма.

3 Технологическая часть

3.1 Средства разработки

В качестве языка программирования был выбран go [8], так как данный язык обладает достаточными средствами для реализации алгоритмов, а также позволяет реализовать конкурентно отдельные части алгоритма с помощью горутин [9].

3.2 Реализация алгоритмов

В листинге (3.1) представлен алгоритма полного перебора, при этом часть генерации перестановок и расчёта длины пути выполняются конкурентно в горутинах, передавая данные через канал [9].

Листинг 3.1 — Алгоритм полного перебора

```
func HeapAlgo(arr []int) <-chan []int {
  ch := make(chan []int)
  go func() {
    c := make([]int, len(arr))
    for i := range c {
      c[i] = 0
    }
    arrCopy := make([]int, len(arr))

    copy(arrCopy, arr)
    ch <- arrCopy

    i := 1
    for i < len(arr) {
      if c[i] < i {
        if i%2 == 0 {
          arr[0], arr[i] = arr[i], arr[0]
        } else {
          arr[c[i]], arr[i] = arr[i], arr[c[i]]
        }
        copy(arrCopy, arr)
        ch <- arrCopy
        c[i] += 1
        i = 1
      } else {
        c[i] = 0
        i++
```

```
      }
    }
    close(ch)
  }()
  return ch
}

type FullSearch struct{}

func NewFullSearch() *FullSearch {
  return &FullSearch{}
}

func (f *FullSearch) Run(gr *graph.WeightedUndirectedGraph) (*graph.
   WeightedCycle, error) {
  bestCycle := gr.GetRandomHamiltonian()
  ch := HeapAlgo(gr.GetNodes())
  for arr := range ch {
    path := graph.NewWeightedCycle(gr)
    for _, node := range arr {
      err := path.AddNode(node)
      if err != nil {
        return nil, err
      }
    }
    if path.CalculateWeight() < bestCycle.CalculateWeight() {
      bestCycle = path
    }
  }
  return bestCycle, nil
}
```

В листинге (3.2) представлена реализация муравьиного алгоритма, а на (3.3) реализация построения пути отдельным муравьём.

Листинг 3.2 — Муравьиный алгоритм

```
type ElitistAntAlgorithm struct {
  distanceCoeff     float64
  pheromoneCoeff    float64
  evaporationCoeff  float64
  initPheromone     float64
  pheromonePerAnt   float64
```

```
  antsCount       int
  eliteAntsCount  int
  daysCount       int
}

func NewElitistAntAlgorithm(distanceCoeff, pheromoneCoeff,
    evaporationCoeff, initPheromone, pheromonePerAnt float64, antsCount,
    eliteAntsCount, daysCount int) *ElitistAntAlgorithm {
  return &ElitistAntAlgorithm{
    distanceCoeff:    distanceCoeff,
    pheromoneCoeff:   pheromoneCoeff,
    evaporationCoeff: evaporationCoeff,
    initPheromone:    initPheromone,
    pheromonePerAnt:  pheromonePerAnt,

    antsCount:      antsCount,
    eliteAntsCount: eliteAntsCount,
    daysCount:      daysCount,
  }
}

func (a *ElitistAntAlgorithm) Run(gr *graph.WeightedUndirectedGraph) (*
    graph.WeightedCycle, error) {
  bestCycle := gr.GetRandomHamiltonian()

  phgr := NewGraphWithPheromon(gr, a.initPheromone)
  for day := 0; day < a.daysCount; day++ {
    ants := make([]*Ant, a.antsCount)
    initNode := gr.GetNodes()[0]
    for i := 0; i < a.antsCount; i++ {
      ants[i] = NewAnt(phgr, a.pheromonePerAnt, a.distanceCoeff, a.
          pheromoneCoeff, initNode)
    }

    for _, ant := range ants {
      if err := ant.Go(); err != nil {
        return nil, err
      }

      if ant.GetPath().CalculateWeight() < bestCycle.CalculateWeight() {
```

```
        bestCycle = ant.GetPath()
      }
    }

    phgr.EvaporatePheromone(a.evaporationCoeff)

    for _, ant := range ants {
      phgr.ApplyPheromon(ant.GetPath(), ant.pheromone)
    }

    for i := 0; i < a.eliteAntsCount; i++ {
      phgr.ApplyPheromon(bestCycle, a.pheromonePerAnt)
    }
  }
  return bestCycle, nil
}
```

Листинг 3.3 — Алгоритм построения пути муравьём

```
func (a *Ant) chooseNextNode() int {
  lastNode, _ := a.path.LastNode()

  sumDesire := 0.
  for _, node := range a.unvisited {
    sumDesire += a.desireFunc(lastNode, node)
  }

  probabilities := make(map[int]float64, len(a.unvisited))
  for i := range a.unvisited {
    probabilities[i] = a.desireFunc(lastNode, a.unvisited[i]) / sumDesire
  }
  randVal := rand.Float64()
  sumProb := 0.
  for i, prob := range probabilities {
    sumProb += prob
    if sumProb >= randVal {
      node := a.unvisited[i]
      a.unvisited = append(a.unvisited[:i], a.unvisited[i+1:]...)
      return node
    }
  }
  return -1
}
```

```
func (a *Ant) Go() error {
  for len(a.unvisited) > 0 {
    nextNode := a.chooseNextNode()
    if nextNode == -1 {
      return fmt.Errorf("no valid next node found")
    }
    a.path.AddNode(nextNode)
  }
  return nil
}
```

3.3 Оценка сложности алгоритмов

Для алгоритма полного перебора оценка сложности составляет $O(n! * n) = O(n!)$, так как сложность алгоритма генерации перестановок $O(n!)$ [7], а сложность поиска цены пути составляет $O(n)$.

У муравьиного алгоритма оценка сложности рассчитывается следующим образом

$$O(d * (m * (n^2) + (n^2) + m * n + n)) = O(dmn^2), \tag{3.1}$$

где

— d – количество дней;

— m – число муравьёв;

— n – число узлов.

3.4 Тестирование

Тестирование алгоритма полного перебора возможно провести с проверкой точного значения, однако для муравьиного алгоритма можно проверить лишь то, что результирующий цикл не содержит все узлы, то есть является гамильтоновым циклом.

Тест 1

Входные данные:

$$\begin{bmatrix} 0 & 10 & 15 & 20 & 25 \\ 10 & 0 & 30 & 35 & 40 \\ 15 & 30 & 0 & 45 & 50 \\ 20 & 35 & 45 & 0 & 55 \\ 25 & 40 & 50 & 55 & 0 \end{bmatrix}$$

Ожидаемое значение: [0 1 2 3 4], 100.0

Выходные данные полного перебора: [0 1 2 3 4], 100.0

Выходные данные муравьиного алгоритма: [0 4 3 2 1] 100.0

Тест 2

Входные данные:

$$\begin{bmatrix} 0.00 & 12.34 & 23.45 & 34.56 & 45.67 & 56.78 & 67.89 & 78.90 & 89.01 & 90.12 \\ 12.34 & 0.00 & 13.14 & 25.35 & 36.46 & 47.57 & 58.68 & 69.79 & 80.90 & 91.01 \\ 23.45 & 13.14 & 0.00 & 14.25 & 26.36 & 38.47 & 50.58 & 62.69 & 74.80 & 86.91 \\ 34.56 & 25.35 & 14.25 & 0.00 & 16.37 & 28.48 & 40.59 & 52.70 & 64.81 & 76.92 \\ 45.67 & 36.46 & 26.36 & 16.37 & 0.00 & 18.49 & 30.60 & 42.71 & 54.82 & 66.93 \\ 56.78 & 47.57 & 38.47 & 28.48 & 18.49 & 0.00 & 12.61 & 24.72 & 36.83 & 48.94 \\ 67.89 & 58.68 & 50.58 & 40.59 & 30.60 & 12.61 & 0.00 & 12.83 & 24.94 & 37.05 \\ 78.90 & 69.79 & 62.69 & 52.70 & 42.71 & 24.72 & 12.83 & 0.00 & 13.05 & 25.16 \\ 89.01 & 80.90 & 74.80 & 64.81 & 54.82 & 36.83 & 24.94 & 13.05 & 0.00 & 13.27 \\ 90.12 & 91.01 & 86.91 & 76.92 & 66.93 & 48.94 & 37.05 & 25.16 & 13.27 & 0.00 \end{bmatrix}$$

Ожидаемое значение: [0 1 2 3 4 5 6 7 8 9] 216.47

Выходные данные полного перебора: [0 1 2 3 4 5 6 7 8 9] 216.47

Выходные данные муравьиного алгоритма: [0 1 2 3 4 6 7 8 9 5] 231.57

Вывод

В ходе технологической части работы были разработаны муравьиный алгоритм и алгоритм полного перебора для задачи коммивояжёра, а также проведена оценка их временной сложности. Все тесты успешно пройдены.

4 Исследовательская часть

4.1 Параметризация

Параметризация проводится по 3-м параметрам муравьиного алгоритма: α – коэффициент влияние феромонов на выбор муравья, p – коэффициент испарения феромонов, дни – количество дней в муравьином алгоритме.

4.1.1 Класс данных

В качестве класса данных для параметризации используем графы, построенные на городах Африки, при этом всего будет 3 графа. В качестве весов графа использовались расстояния между этими городами по прямой в км.

Граф 1

1) Каир (Египет);
2) Лагос (Нигерия);
3) Найроби (Кения);
4) Кейптаун (Южноафриканская Республика);
5) Аккра (Гана);
6) Дакар (Сенегал);
7) Абиджан (Кот-д'Ивуар);
8) Дар-эс-Салам (Танзания);
9) Аддис-Абеба (Эфиопия);
10) Хартум (Судан).

Граф 2

1) Бондио (Габон);
2) Нджамена (Чад);
3) Конакри (Гвинея);
4) Луанда (Ангола);
5) Мансу (Сьерра-Леоне);
6) Джамбул (Казахстан);
7) Махаджанга (Мадагаскар);
8) Габороне (Ботсвана);
9) Виндхук (Намибия);
10) Ямусукро (Кот-д'Ивуар).

Граф 3

1) Тунис (Тунис);
2) Кигали (Руанда);
3) Момбаса (Кения);
4) Лусака (Замбия);
5) Джуба (Южный Судан);
6) Мапуту (Мозамбик);
7) Бамако (Мали);
8) Либревиль (Габон);
9) Кигали (Руанда);
10) Антананариву (Мадагаскар).

4.1.2 Результаты параметризации

В результате параметризации оказалось, что самыми лучшими параметрами по заданному классу данных оказались при $\alpha = 0.3$, $p = 0.3$ и количеством дней равным 200. При этом данные параметры дают лучший результат как по средним параметрам, так и по минимальным значениям. Результаты параметризации для данных параметра представлены в таблице (4.1), а вся таблица параметризации представлена в приложении А.

Таблица 4.1 — Результаты параметризации муравьиного алгоритма

Параметры			Граф 1			Граф 2			Граф 3		
α	p	Дни	min	max	avg	min	max	avg	min	max	avg
0.3	0.3	200	250.90	250.90	250.90	225.50	233.50	226.30	258.60	261.60	258.90

4.2 Вывод

В результате исследовательской части была проведена параметризация по заданному классу данных и выявлены лучшие параметры.

ЗАКЛЮЧЕНИЕ

Цель – рассмотрение методов решения задачи коммивояжёра – была выполнена. При этом в ходе работы были выполнены следующие задачи:

— сформулирована задача коммивояжёра;
— рассмотрены методы полного перебора и муравьиной колонии;
— разработаны рассмотренные алгоритмы решения задачи;
— реализованы разработанные алгоритмы на языке Go;
— проведена параметризация для алгоритма на основе метода муравьиной колонии.

В результате работы были проанализированы сложности алгоритмов муравьиной колонии и полного перебора, проведена параметризация муравьиного алгоритма и выявлены лучшие параметры для класса данных, основанных на городах Африки, а именно: $\alpha = 0.3$, $p = 0.3$ и количество дней – 200.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. М. О. Асанов, В. А. Баранский, В. В. Расин. Дискретная математика: графы, матроиды, алгоритмы : Учебное пособие. - 3-е изд. - СПб.: Лань, 2020. - 364 с.

2. Donald Davendra. Traveling Salesman Problem, Theory and Applications. [Текст] / Donald Davendra. — Croatia: IntechOpen, 2010 — 338 с.

3. Michael Junger, Gerhard Reinelt, Giovanni Rinaldi. The Traveling Salesman Problem. - Institut fur Informatik UNIVERSITAT ZU KOLN, 1994. - 112 с.

4. Marco Dorigo, Thomas Stützle. Ant Colony Optimization: Overview and Recent Advances // Handbook of Metaheuristics. - Switzerland: Springer International Publishin, 2019. - С. 311-351.

5. Marco Dorigo, Alberto Colorni. Ant Colony Optimization: Overview and Recent Advances // IEEE Transactions on Systems, Man, and Cybernetics–Part B. - Institute of Electrical and Electronics Engineers, 1996. - С. 1-13.

6. Robert Sedgevick. Permutations generation methods // Computing Survey. - 1977

7. Robert Sedgevick. Permutations generation methods // Princeton University. - 2002

8. GO // GO programming language URL: https://go.dev/ (дата обращения: 24.10.2024).

9. The Go Memory Model // GO programming language URL: https://go.dev/ref/mem (дата обращения: 24.10.2024).

Приложение А