Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

РЕФЕРАТ
на тему

АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА

Выполнил: Н.Р. Ровдо
Проверила: О.И. Костюкова

МИНСК 2018

СОДЕРЖАНИЕ

Введение

Задача коммивояжёра — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются *критерий выгодности маршрута* (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город *только один раз* — в таком случае выбор осуществляется среди циклов. Существует *несколько частных случаев* общей постановки задачи, в частности, геометрическая задача коммивояжёра (также называемая планарной или евклидовой, когда матрица расстояний отражает расстояния между точками на плоскости), метрическая задача коммивояжёра (когда на матрице стоимостей выполняется неравенство треугольника), симметричная и асимметричная задачи коммивояжёра. Также существует обобщение задачи, так называемая *обобщённая задача коммивояжёра*.

Математическая модель этой задачи отображает также ситуацию совершенно иного характера. Имеется n сортов мороженого, которое изготавливается на одном и том же оборудовании. Пусть c_{ij} означает затраты времени на очистку и подготовку оборудования, когда сорт j изготавливается после сорта i. Предполагается, что заданная последовательность производства повторяется каждый день, т.е. оборудование после последнего сорта мороженого опять настраивается на производство первого сорта. Требуется найти такую последовательность производства, при которой затраты на переналадку были бы минимальными.

Алгоритмы, приведённые в реферате – *эвристические*. Они *не гарантируют* оптимальное решение.

Обозначения и сокращения

Задача о назначениях: имеется некоторое число работ и некоторое число исполнителей. Любой исполнитель может быть назначен на выполнение любой (но только одной) работы, но с неодинаковыми затратами. Нужно распределить работы так, чтобы выполнить работы с минимальными затратами.

Задача о назначениях является *частным случаем транспортной задачи*, которая является частным случаем задачи нахождения потока минимальной стоимости, а она, в свою очередь, является *частным случаем задачи линейного программирования*. Любую из этих задач можно решить симплекс-методом, но каждая специализация имеет свой *более эффективный алгоритм*, опирающийся на особенности структуры задачи.

Венгерский алгоритм — алгоритм, разработанный для решения линейной задачи о назначениях.

Метод ветвей и границ

Метод ветвей и границ — общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является *вариацией полного перебора* с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Метод ветвей и границ впервые предложили в *1960 году* Аилсой Ленд и Элисон Дойг для решения задач целочисленного программирования.

Процедура ветвления состоит *в разбиении множества допустимых значений* переменной x на подобласти (подмножества) меньших размеров. Процедуру нужно *рекурсивно* применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или *деревом ветвей и границ*. Узлами этого дерева являются *построенные подобласти* (подмножества множества значений переменной x).

Метод задания маршрутов *(через поиск минимумов)*

Мы имеем матрицу весов или расстояний для задачи.

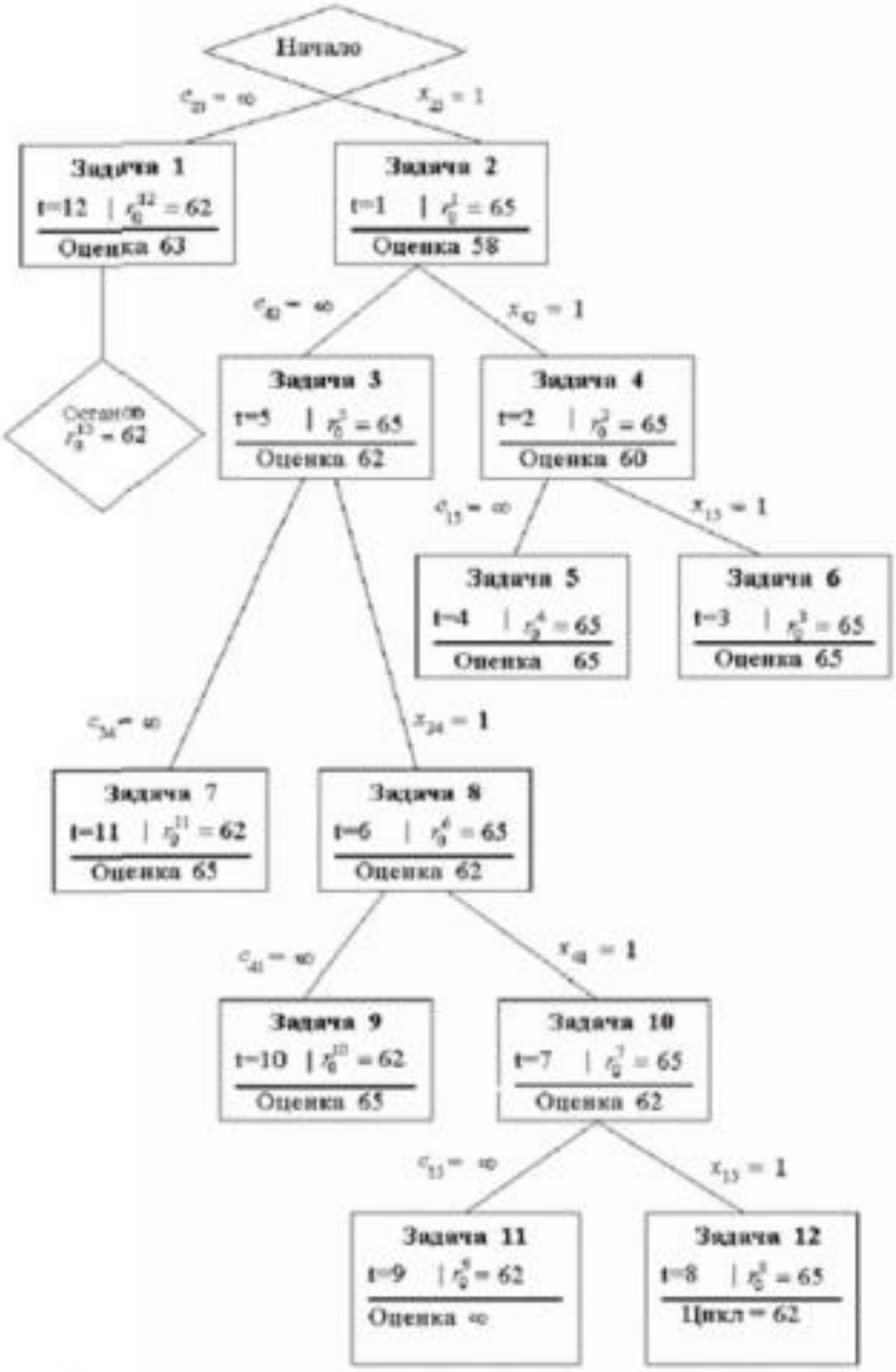
1) Находим любой маршрут коммивояжёра;
2) Находим его стоимость;
3) Запоминаем её. Варианты, которые больше не рассматриваем в последующем, потому что они *очевидно* не оптимальные;
4) Если найдём вариант лучше то, его запомним и будем сравнивать на его основе;
5) В матрице стоимостей;
6) В матрице берём минимальный по строкам;
7) Вычитаем из каждой строки минимумы;
8) Считаем сумму и запоминаем;
9) И тоже самое по столбцам;
10) Эти две суммы мы складываем;
11) Сумма показывает, что длина текущего плана или любого иного плана не будет меньше этой величины;
12) Берём матрицу после вычитания минимумов для строк и столбцов;
13) Для каждого нулевого элемента находим минимум по строкам и столбцам его координат;
14) Выбираем максимальную такую сумму;

15) Определяем индекс максимального для этого элемента;
16) Далее ветвим на два подмножества;
17) Одно **без** с этого маршрута;
18) Другое **вместе** с этим маршрутом;
19) Рисуем матрицу с нулем и вместо него поставим бесконечность;
20) Снова ищем минимум по колонкам и по столбцам;
21) Их сумму добавляем к минимуму, найденному в первый раз;
22) Это будет левая граница множества;
23) Правая граница – наш найденный план;
24) То же самое делаем для включённого множества;
25) Убираем строки и столбцы координат максимального индекса;
26) В обратные координаты ставим бесконечность;
27) добавляем слева границу потом начинаем снова считать нули;
28) Это будет правая граница второй ветви для возможных решений;
29) Левая граница – наша максимальная сумма;
30) Продолжаем данный алгоритм, рекурсивно не забывая о родительских ограничениях на добавление/удаление дуг;
31) Добавляем элементы так чтобы не забывать преобразовать матрицу;
32) Необходимо проверять решение на циклы и удалять их из матрицы выставляя бесконечность в соответствующую ячейку;

Пример

Решение задачи для таблицы весов

inf	31	15	19	8	55
19	inf	22	31	7	35
25	43	inf	53	57	16
5	50	49	inf	39	9
24	24	33	5	inf	14
34	26	6	3	36	inf



Оптимальный цикл: r_0=62
(задача 12)

$x_{13}=x_{32}=x_{23}=x_{34}=x_{41}=1$

Метод разрыва/исключения подциклов *(через задачу о назначениях)*

1) Загружаем задачу в задаче о назначениях;
2) Из задачи о назначениях получаем матрицу получаем маршрут;
3) Если есть цикл меняем дуги цикла бесконечности ветвим;
4) На две задачи о назначениях;
5) Продолжаем пока не найдем план меньше;
6) Когда планов не останется в очереди задач, приостановить поиск;

```
from numpy import inf, copy
from Naznach import *
from functools import reduce

def inputValues(file_name):
    matrix = []
    with open(file_name, 'r') as fin:
        for line in fin:
            matrix.append(list(map(float, line.split())))
    return matrix

class Comivoyager(object):
  def __init__(self, c):
    self.c = c # matrix of weights
    self.n = len(c[0]) # count of cities
    self.r = reduce(lambda s, x: s + x, [c[i][i+1] for i in range(self.n -
1)]) + c[self.n - 1][0] # record
    self.plan = list(range(self.n)).append(0)

  def calculate_plan(self, res):
    return reduce(lambda s, x: s + x, [self.c[i][res[i]] for i in
range(self.n)])

  def dfs(self, v):
    self.used[v] = True
    self.components[self.comp_index].append((v,
self.assigment_problem_res[v]))
    if not self.used[self.assigment_problem_res[v]]:
      self.dfs(self.assigment_problem_res[v])

  def create_components(self):
    self.used = [False for _ in range(self.n)]
    self.components = []
    self.comp_index = 0
    for i in range(self.n):
      if not self.used[i]:
        self.components.append([])
        self.dfs(i)
        self.comp_index += 1

  def branch_and_bound(self):
    self.assigment_problem_res = MatchSolve(copy(self.c)).solve()
    r = self.calculate_plan(self.assigment_problem_res)
    if r >= self.r:
      return
    self.create_components()

    if len(self.components) == 1:
```

```
      self.plan = [c[0] for c in self.components[0]]
      self.plan.append(0)
      self.r = r
    else:
      for component in self.components:
        for i, j in component:
          cij = self.c[i][j]
          self.c[i][j] = inf
          self.branch_and_bound()
          self.c[i][j] = cij

  def solve(self):
    self.branch_and_bound()
    return self

c = inputValues('11ex')
res = Comivoyager(c).solve()
print(res.plan)
print(res.r)
```

Рекурсивный метод ветвей и границ методом задачи о назначениях

Пример

Пусть имеется $n = 5$ городов, связанных системой дорог. Обозначим через $d_{ij} > 0$ длину пути из города *i* в город *j*. Числа $d_{ij}, i = \overline{1,5},\ j = \overline{1,5},$ приведены в таблице

∞	2	1	10	6
4	∞	3	1	3
2	5	∞	8	4
6	7	13	∞	3
10	2	4	6	∞

Требуется найти маршрут, обладающий следующими свойствами:

A) маршрут заканчивается в том городе, с которого он начался,

B) маршрут должен включать все города и ни один город (кроме начального) не может быть включен в маршрут дважды,

C) маршрут имеет минимально возможную длину.

Решение. Выберем любой маршрут, удовлетворяющий свойствам A) и B). Пусть это будет маршрут

$$1 \to 2 \to 3 \to 4 \to 5 \to 1. \quad (11.1)$$

Длина данного маршрута равна $r^0 = 2+3+8+3+10 = 26$. Запоминаем маршрут. Число $r^0 = 26$ берем в качестве рекорда. В список задач о назначениях включает задачу №1, с матрицей с матрицей стоимостей, приведенной в таблице.

Итерация 1.

Решим задачу о назначениях №1. Получим ответ: оптимальное назначение

$$1 \to 3, \quad 3 \to 1, \quad 2 \to 4, \quad 4 \to 5, \quad 5 \to 2,$$

его стоимость равна $1+2+1+3+2 = 9 < r^0 = 29.$ Также получим последнюю приведенную матрицу стоимостей:

$$C^1 = \begin{pmatrix} \infty & 1 & 0 & 9 & 6 \\ 3 & \infty & 2 & 0 & 2 \\ 0 & 3 & \infty & 6 & 2 \\ 3 & 4 & 10 & \infty & 0 \\ 8 & 0 & 2 & 4 & \infty \end{pmatrix}$$

Дуги (1,3), (3,1), (2,4), (4,5), (5,2), соответствующие оптимальному назначению, образуют два контура:

$$1 \to 3 \to 1 \text{ и } 2 \to 4 \to 5 \to 2.$$

Поэтому решение задачи о назначениях не может быть использовано в качестве решения исходной задачи коммивояжера.

Из контуров выберем контур с минимальным количеством дуг. В данном примере это контур $1 \to 3 \to 1$, состоящий из двух дуг (1,3) и (3,1). Каждой дуге выбранного контура поставим в соответствие задачу о назначениях по правилу: дуге (i, j) из выделенного контура соответствует задача о назначениях с матрицей стоимостей, полученной из матрицы C^1 заменой коэффициента d_{ij} на $d_{ij} = \infty$. С рассматриваемом примере мы получаем две задачи:

задача №2 с матрицей стоимостей

$$C^2 = \begin{pmatrix} \infty & 1 & \infty & 9 & 6 \\ 3 & \infty & 2 & 0 & 2 \\ 0 & 3 & \infty & 6 & 2 \\ 3 & 4 & 10 & \infty & 0 \\ 8 & 0 & 2 & 4 & \infty \end{pmatrix}$$

и задача №3 с матрицей стоимостей

$$C^3 = \begin{pmatrix} \infty & 1 & 0 & 9 & 6 \\ 3 & \infty & 2 & 0 & 2 \\ \infty & 3 & \infty & 6 & 2 \\ 3 & 4 & 10 & \infty & 0 \\ 8 & 0 & 2 & 4 & \infty \end{pmatrix}$$

Задачу №1 вычеркиваем из списка задач, полученные задачи №2 и №3 включаем в список. Переходим к следующей итерации.

Итерация 2.

Из списка задач о назначениях выберем задачу №2 и решим ее. Получим ответ: оптимальное назначение

$$3 \to 1,\ 1 \to 2,\ 2 \to 4,\ 4 \to 5,\ 5 \to 3,$$

его стоимость равна $2+1+3+4+2=12<r^0=29$. Дуги (3,1), (1,2), (2,4), (4,5), (5,3), соответствующие оптимальному назначению, образуют один контур. Этому контуру соответствует допустимый маршрут

$$3 \to 1 \to 2 \to 4 \to 5 \to 3 \quad (11.3)$$

длина которого равна $12<r^0=29$. Поэтому меняем рекорд, положив $r^0=12$, и запоминаем найденный допустимый маршрут. Задачу №2 вычеркиваем из списка и идем на следующую итерацию.

Итерация 3.

Из списка задач о назначениях выбираем задачу №3 с матрицей стоимостей C^3. В результате решения этой задачи получаем оптимальное назначение

$$1 \to 3,\ 3 \to 5,\ 5 \to 2,\ \ 2 \to 4,\ \ 4 \to 1,$$

стоимость которого равна $1+4+2+1+6=14.$ Поскольку $14 > r^0 = 12$, то вычеркивает задачу №3 из списка и идем на следующую итерацию.

Итерация 4.

Список задач о назначениях пуст. Исходная задача коммивояжера решена: оптимальным является маршрут, соответствующий рекорду $r^0 = 12.$

Результат выполнения программой:

C:\Users\Harwister\AppData\Local\Programs\Python\Python36\python.exe G:/Labs/bsuir-labs/7cem/System_Analysis_And_Operations_Research/lab11/lab_11.py

[0, 1, 3, 4, 2, 0]

12.0

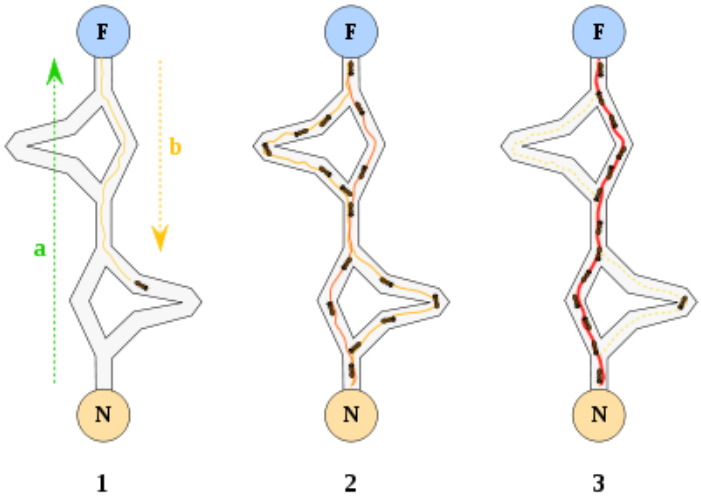
Отметим два существующих отличия данного варианта алгоритма ветвей и границ от предыдущего. в данном варианте нижняя оценка для выбранной задачи, но не отыскивается и оптимальные решение

Муравьиный алгоритм *(вероятностно-временной подход)*

Муравьи используют окружающую среду как средство общения. Они обмениваются информацией косвенным путём, через феромоны, в ходе их «работы». Обмен информации имеет локальный характер, только те муравьи, которые находятся в непосредственной близости, где остались феромоны — могут узнать о них. Такая система называется стигмергия и справедлива для многих социальных животных (была изучена для случая строительства столбов в гнёздах термитов). Данный механизм решения проблемы очень сложен и является хорошим примером самоорганизации системы. Такая система базируется на положительной (другие муравьи укрепляют феромонную тропу) и отрицательной (испарение феромонной тропы) обратной связи. Теоретически, если количество феромонов будет оставаться неизменным с течением времени по всем маршрутам, то невозможно будет выбрать путь. Однако из-за обратной связи, небольшие колебания приведут к усилению одного из маршрутов, и система стабилизируется к кратчайшему пути.

1) Оригинальная идея исходит от наблюдения за муравьями в процессе поиска кратчайшего пути от колонии до источника питания.
2) Первый муравей находит источник пищи (F) любым способом (a), а затем возвращается к гнезду (N), оставив за собой тропу из феромонов (b).
3) Затем муравьи выбирают один из четырёх возможных путей, затем укрепляют его и делают привлекательным.
4) Муравьи выбирают кратчайший маршрут, так как феромоны с более длинных путей быстрее испаряются.
5) Среди экспериментов по выбору между двумя путями неравной длины, ведущих от колонии к источнику питания, биологи заметили, что, как правило, муравьи используют кратчайший маршрут. Модель такого поведения заключается в следующем:
6) Муравей проходит случайным образом от колонии
7) Если он находит источник пищи, то возвращается в гнездо, оставляя за собой след из феромона
8) Эти феромоны привлекают других муравьёв, находящихся вблизи, которые вероятнее всего пойдут по этому маршруту
9) Вернувшись в гнездо, они укрепят феромонную тропу
10) Если существует 2 маршрута, то по более короткому, за то же время, успеют пройти больше муравьёв, чем по длинному

11) Короткий маршрут станет более привлекательным
12) Длинные пути, в конечном итоге, исчезнут из-за испарения феромонов



Пример использования феромонов

```
procedure ACO_MetaHeuristic
  while(not_termination)
     generateSolutions()
     daemonActions()
     pheromoneUpdate()
  end while
end procedure
```

Псевдокод алгоритма

Генетический алгоритм *(эволюционно вероятностный)*

Генетический алгоритм является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и кроссинговер. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов, где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях генетического алгоритма (ГА) предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет, насколько хорошо фенотип, им описываемый, решает поставленную задачу.

Алгоритм

1) Начальная популяция
2) Скрещивание или мутация
3) Селекция
4) Формирование нового поколения
5) Завершить при достижении результата

```
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <iostream>
#include <numeric>

int main()
{
    srand((unsigned int)time(NULL));
    const size_t N = 1000;
    int a[N] = { 0 };
```

```
    for ( ; ; )
    {
        //мутация в случайную сторону каждого элемента:
        for (size_t i = 0; i < N; ++i)
            a[i] += ((rand() % 2 == 1) ? 1 : -1);
        //теперь выбираем лучших, отсортировав по возрастанию
        std::sort(a, a + N);
        //и тогда лучшие окажутся во второй половине массива.
        //скопируем лучших в первую половину, куда они оставили потомство,
а первые умерли:
        std::copy(a + N / 2, a + N, a);
        //теперь посмотрим на среднее состояние популяции. Как видим, оно
всё лучше и лучше.
        std::cout << std::accumulate(a, a + N, 0) / N << std::endl;
    }
}
```

Сравнение алгоритмов

Плюсы для всех – сокращение полного перебора

Минусы для всех – нет гарантии точности, только приближённое решение.

	Плюсы	**Минусы**
Метод разрыва/исключения подциклов	• Быстрый • Простой • Прогрессирующий результат	• Требует качественного решения задачи о назначении
Метод задания маршрутов	• Простой • Итеративен	• Сложный
Муравьиный алгоритм	• Простой • Надёжный • Прогрессирующий результат	• Ресурсозатратный
Генетический алгоритм	• Прогрессирующий результат	• Ресурсозатратный • Сложный • Плохо масштабируется • Ненадёжный

Вывод

Для поиска маршрутов приемлемой длины точные методы следует комбинировать с эвристическими.