# МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра алгебры, геометрии и дискретной математики**

Направление подготовки прикладная математика и информатика

Направленность (профиль) программы магистратуры: анализ данных

**ОТЧЕТ**

по учебной практике

на тему:

**«Изучение и реализация эффективно разрешимых случаев квадратичной задачи о назначениях»**

**Выполнил:**

студент группы 3823М1ПМад Сахаров Тихон Алексеевич



**Руководитель:**

Профессор кафедры АГДМ, доктор физико-математических наук

Малышев Дмитрий Сергеевич

Нижний Новгород

2023

**Содержание**

1. Введение…………………………………………….…………………….3
2. Постановка задачи………………………………….…………………….3
3. Цель работы……………………….……………………………………...4
4. Полиномиальные алгоритмы..….………………………………………..4
   1. Описание первого метода…………………………………………5
   2. Описание второго метода ………………………………………...5
   3. Описание третьего метода ...……………………………………...6
5. Алгоритмы динамического программирования………………………...7
   1. Последовательный………………………………………………....7
   2. Параеллельный……………………………………………………..7
6. Заключение………………………………………………………………..8
   1. Вывод…………………………………………………………….…8
7. Приложение………………………………………………………………10
8. Список литературы………………………………………………………12
9. **Введение**

Квадратичная задача о назначениях (КЗН) является базовой математической моделью в различных областях, таких как автоматизация проектирования схем расположения объектов, включая генеральные планы предприятий, цехов, радиоэлектронной аппаратуры.

Одним из прикладных применений квадратичной задачи о назначениях является оптимизация процесса распределения ресурсов в области транспортной логистики. Например, предположим, что у нас есть несколько складов и несколько магазинов, и мы хотим определить оптимальное соответствие между складами и магазинами для минимизации общих затрат на транспортировку товаров. Каждому складу можно назначить несколько магазинов, и каждое такое назначение имеет свою стоимость, которая зависит от расстояния и объема перевозимых товаров. Таким образом, мы можем сформулировать эту задачу как квадратичную задачу о назначениях, где мы стремимся минимизировать общую стоимость перевозок, выбирая оптимальное соответствие между складами и магазинами. Решение этой задачи позволит оптимизировать логистические процессы и снизить затраты на транспортировку товаров.

Для моделирования связей между объектами используется реберно-взвешенный граф без петель и кратных ребер. Вершины графа соответствуют объектам, а ребра указывают на наличие связей между ними, где веса ребер отражают удельные стоимости связей. Позиции для размещения объектов рассматриваются как узлы сети, а критерием оптимизации является минимальная суммарная стоимость связей между объектами.

В данной работе особое внимание уделяется разработке полиномиальных алгоритмов для специальных видов графов и сетей, что отличает ее от поиска сильной разрешимости КЗН для матричной постановки задачи. В частности, представлен эффективный алгоритм локальной оптимизации для случая двудольных графов. Показано, что если граф и сеть представляют изоморфные деревья, то задача КЗН разрешима за полиномиальное время с использованием методов динамического программирования.

1. **Постановка задачи**

Задан неориентированный взвешенный граф без петель и кратных ребер. Символ *W* означает, что граф взвешенный. Если веса одинаковы, то этот символ опускается. Каждому ребру приписан вес , т.е. удельная стоимость связи между размещаемыми объектами . Пусть – это сеть без петель и кратных дуг. Дуге приписан вес . Считаем, что . Размещением графа *LW* на сети *MW* будем называть взаимно однозначное отображение . Если , тогда *π* – это подстанов­ка. Необходимо разместить вершины графа *LW* в узлы сети *MW* по одной в каждый таким обра­зом, чтобы суммарная стоимость связей между вершинами графа LW была минимальной. Целе­вая функция имеет вид (1):

где – кратчайшее расстояние между узлами π(ni ) и π(nj ) в сети MW, в которые раз­ мещены вершины графа . Такую задачу размещения будем обозначать тройкой (LW, MW, F1). Отметим, что задача *NP­*-трудна, когда *M* – цепь, а *L* – произвольный граф. [2]

1. **Цель работы**

Изучение полиномиальных методов решения и методов динамического программирования для квадратичной задачи о назначениях, а также в инженерной оптимизации некоторых из них с целью улучшения эффективности и скорости решения задачи. В связи с этим, планируется провести анализ эффективности реализованных методов, а также разработать улучшенные версии алгоритмов с использованием современных подходов к оптимизации вычислений. Для сравнения будем использовать исследования Г.Г. Забудского и А.Ю. Лагздина, которые предложили использование динамического программирования для решения квадратичной задачи о назначениях на дереве, а также провели анализ параллельного алгоритма для этой задачи.

1. **Полиномиальные алгоритмы**

Полиномиальные алгоритмы для решения квадратичной задачи о конечных назначениях используются для оптимизации задачи, где требуется найти оптимальное назначение между конечными сущностями, минимизируя квадратичную функцию стоимости. Эти алгоритмы имеют вычислительную сложность, зависящую от размера входных данных, и позволяют найти решение за разумное время, что делает их эффективными в прикладных задачах.

Реализуем алгоритмы оптимизации сети:

4.1.Первый полиномиальный метод:

Сложность алгоритма оценивается как O(m3) операций, что объясняется операциями по построению кратчайших расстояний и упорядочением строк матрицы. Доказательство оптимальности алгоритма основано на свойствах минимума скалярного произведения векторов.

* Шаг 1: Строится матрица кратчайших расстояний между всеми парами узлов сети MW. Матрица имеет размер m строк на столбцов, исключая нулевые диагональные элементы.
* Шаг 2: Элементы строк полученной матрицы упорядочиваются по неубыванию, что приводит к формированию матрицы
* Шаг 3: Вершины графа SW упорядочиваются по невозрастанию стоимости связи с центром .
* Шаг 4: Находится минимальное покомпонентное произведение каждой строки матрицы K на вектор стоимостей связи с центром звезды. Выбирается строка, соответствующая узлу .
* Шаг 5: Присваиваются значения = .
  1. Второй полиномиальный метод.

Это алгоритм упорядочивания вершин графа LW и сети SW для последующего построения оптимального маршрута. На первом шаге вершины графа упорядочиваются по суммарной стоимости связей соседних вершин, на втором шаге сети упорядочиваются по расстояниям от центра, а на третьем шаге устанавливается соответствие между вершинами и центрами сети:

* Шаг 1. Упорядочиваем вершины графа LW по невозрастанию суммарной стоимости связей сосмежными вершинами. Если = , то пусть нумерация вершин N графа LW такова, что .
* Шаг 2. Узлы , сети SW упорядочиваем по неубыванию расстояний от цен тра v1, т.е. выполняются неравенства.
* Шаг 3. Полагаем .
  1. Третий метод:
* Шаг 1. Выделим в сети TW цепь максимальной длины, т.е.
* Шаг 2. Последовательно рассматриваем узлы выделенной цепи: полагаем . Далее, если . Для размещениявершин {3, 4, ..., + 1} применяется редуцированный вариант данного алгоритма для поддерева : выделяется цепь, соединяющая его корень и произвольный висячий узел, после чего размещение производится аналогично основному алгоритму.
* Заметим, что поскольку в TW выделена цепь максимальной длины, то .

1. **Алгоритмы динамического программирования**

Алгоритмы динамического программирования для квадратичной задачи о конечных назначениях используются для нахождения оптимального решения путем разбиения задачи на подзадачи и нахождения оптимального решения для каждой из них. Последующее комбинирование уже найденных оптимальных решений позволяет найти ответ для всей задачи. Эти алгоритмы обладают оптимальной подструктурой, что позволяет избежать вычисления одних и тех же значений несколько раз, что повышает эффективность решения задачи.

Задача: даны неориентированные реберно взвешенный граф и сеть , в узлы которой размещаются вершины графа . Предполагается, что . Необходимо найти взаимно однозначное отображение , для которого

(2)

где – вес ребра – кратчайшее расстояние между узлами

Считаем, что вершины графа *L* и узлы древовидной сети *M* пронумерованы числами

* 1. Последовательный алгоритм ДП
* Последовательно рассматриваются узлы сети *M* согласно их номерам и вычисляются значения функции *f* по формулам (4)–(6) в зависимости от степени узла. Значение *f(V)* – это оптимальное значение целевой функции задачи.
* Перебираются узлы сети *M* в обратном порядке и определяются вершины графа *L*, размещенные в узлы, – это вершины, на которых достигаются минимумы в соответствующих формулах (5) или (6).
  1. Параллельный алгоритм ДП

Пусть в параллельном алгоритме ДП работают r потоков. Один из них является главным, r−1 – подчиненными [27]. Обозначим подчиненные потоки как . Параллельный алгоритм состоит из двух этапов – прямого и обратного хода.

Прямой ход:

* Главный поток определяет множество узлов сети *M*, смежных с корнем. Для каждого , он передает узел vi и соответствующее ему поддерево с множеством узлов подчиненному потоку с номером
* Каждый подчиненный поток *i* применяет прямой ход алгоритма ДП к поддереву на множестве узлов и вычисляет соответствующие значения функции .
* После завершения работы подчиненных потоков главный поток вычисляет значение функции и, следовательно, находит оптимальное значение целевой функции .

Обратный ход:

* Главный поток определяет вершину графа *L*, размещенную в корень дерева *M*.
* Каждому из подчиненных потоков передается множество вершин, размещенных в соответствующие потокам поддеревья, и значения функции *f*.
* Потоки последовательно обрабатывают узлы поддеревьев, двигаясь от узла с наибольшим номером к наименьшему, и определяют вершины графа *L*, размещенные в данные узлы.

1. **Заключение**

Протестируем все методы на примере случайной входной матрицы сети MW (расстояния между узлами) размером

|  |  |
| --- | --- |
| Алгоритм | Рабочее время (с) |
| Полиномиальный алгоритм 1 | 0.0030300 |
| Полиномиальный алгоритм 2 | 0.0020008 |
| Полиномиальный алгоритм 3 | 19.80498 |
| Последовательный алгоритм динамического программирования | 0.000996 |
| Параллельный алгоритм динамического программирования | 0.0312 |

**Вывод**

В ходе учебной практики было изучено применение методов оптимизации для решения квадратичной задачи о назначениях. Было проведено сравнение полиномиальных методов и методов динамического программирования которые могут быть более эффективными для задач с большим объемом данных и сложными зависимостями между ними, так как позволяют избежать повторных вычислений и находить оптимальные решения в квадратичной задаче о конечных назначениях.

Наиболее оптимальным оказался последовательный алгоритм динамического программирования, использование механизма *concurrency* в языке python, не дало прироста в производительности.

1. **Приложение**

**Полиномиальные алгоритмы**

**# Алгоритм 1**

**class** **Base1**:

**def** **algorithm**(graph):

n = len(graph)

m = len(graph[**0**])

# Шаг 1: Выделяем в сети TW цепь максимальной длины

chain = []

visited = set()

**def** **dfs**(node, parent):

visited.add(node)

chain.append(node)

**for** neighbor **in** range(m):

**if** graph[node][neighbor] == **1** **and** neighbor != parent:

dfs(neighbor, node)

**for** i **in** range(n):

**if** i **not** **in** visited:

chain = []

dfs(i, -**1**)

**if** len(chain) > len(max\_chain):

max\_chain = chain

# Шаг 2: Рассматриваем узлы выделенной цепи

mapping = {max\_chain[**0**]: **1**, max\_chain[**1**]: **2**}

**for** i **in** range(**2**, len(max\_chain)):

**if** i == **2**:

mapping[max\_chain[i]] = **3**

**else**:

mapping[max\_chain[i]] = mapping[max\_chain[i-**2**]] + **2**

**return** mapping

**# Алгоритм 2**

**class** **Base2**:

**def** **algorithm**(graph):

n = len(graph)

m = len(graph[**0**])

vertices = list(range(n))

vertices.sort(key=**lambda** x: sum(graph[x]), reverse=True)

center = vertices[**0**]

nodes = list(range(m))

nodes.sort(key=**lambda** x: graph[center][x])

# Шаг 3: Присваиваем вершинам соответствующие узлы

mapping = {vertices[i]: nodes[i] **for** i **in** range(n)}

**return** mapping

**# Алгоритм 3**

**class** **FloydMarshall**:

**def** **floyd\_warshall**(graph):

n = len(graph)

dist = [[float('inf')] \* (n-**1**) **for** \_ **in** range(n)]

**for** i **in** range(n):

**for** j **in** range(n-**1**):

**if** i == j:

**continue**

dist[i][j] = graph[i][j]

**for** k **in** range(n -**1**):

**for** i **in** range(n - **1**):

**for** j **in** range(n - **1**):

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

**return** dist

**# Тесты**

**import** **time**

**import** **random**

# Algorithms

**from** **Base1** **import** Base1

**from** **Base2** **import** Base2

**from** **FloydMarshall** **import** FloydMarshall

**def** **test**():

# Пример входной матрицы сети MW (расстояния между узлами)

# Генерация случайной матрицы размером 100x100

n = **500**

graph = [[random.randint(**1**, **10**) **for** \_ **in** range(n)] **for** \_ **in** range(n)]

# Заполнение диагонали нулями (расстояние от узла до самого себя)

**for** i **in** range(n):

graph[i][i] = **0**

# Floyd-marshall test

start\_time = time.time()

shortest\_distances = FloydMarshall.floyd\_warshall(graph)

end\_time = time.time()

**print**("**\n**Floyd-marshall method working time: ", float(end\_time - start\_time))

# Second base test

start\_time = time.time()

shortest\_distances = Base2.algorithm(graph)

end\_time = time.time()

**print**("**\n**Second algo method working time: ", float(end\_time - start\_time))

# Перезаполняем

graph = [[random.randint(**1**, **10**) **for** \_ **in** range(n)] **for** \_ **in** range(n)]

# Заполнение диагонали нулями (расстояние от узла до самого себя)

**for** i **in** range(n):

graph[i][i] = **0**

# Third base test

start\_time = time.time()

shortest\_distances = Base1.algorithm(graph)

end\_time = time.time()

**print**("**\n**Third algo method working time: ", float(end\_time - start\_time))

test()

**# Алгоритмы динамического программирования**

# Последовательный алгоритм ДП

def dfs(node, parent, dp, mw, tree):

dp[node][1] = mw[node]

for child in tree[node]:

if child != parent:

dfs(child, node, dp, mw, tree)

dp[node][0] += max(dp[child][0], dp[child][1])

dp[node][1] += dp[child][0]

def min\_cost\_assignment(tree, MW):

sys.setrecursionlimit(len(MW)\*len(MW[0]))

n = len(tree)

dp = [[0, 0] for \_ in range(n)]

dfs(0, -1, dp, MW, tree)

return max(dp[0][0], dp[0][1])

# Параллельный алгоритм ДП

import concurrent.futures

import sys

def min\_cost\_assignment(tree, mw):

sys.setrecursionlimit(len(mw)\*\*4)

n = len(tree)

dp = [[0, 0] for \_ in range(n)]

def dfs(node, parent):

dp[node][1] = mw[node]

with concurrent.futures.ThreadPoolExecutor() as executor:

futures = [executor.submit(dfs, child, node) for child in tree[node] if child != parent]

concurrent.futures.wait(futures)

for future in futures:

child\_dp = future.result()

dp[node][0] += max(child\_dp[0], child\_dp[1])

dp[node][1] += child\_dp[0]

return dp[node]

return max(dfs(0, -1))

**Список литературы**

**Список литературы**

1. Забудский Г.Г., Лагздин А.Ю. "Динамическое программирование для решения квадратичной задачи о назначениях на дереве" // Автоматика и телемеханика, 2012, выпуск 2, с. 141–155. https://elibrary.ru/download/elibrary\_17437526\_19587991.pdf
2. Забудский Г.Г., Лагздин А.Ю. "Анализ эффективности параллельного алгоритма динамического программирования для квадратичной задачи о назначениях на дереве" // Тезисы докладов XIV Всероссийской конференции "Математическое программирование и приложения", Информационный бюллетень Ассоциации математического программирования, Екатеринбург, 2011, № 12, с. 176.

https://elibrary.ru/download/elibrary\_17437526\_69706246.pdf

1. Koopmans T.C., Beckman M. Assignment problems and the location of economic activities. Econometric. 1957. № 25. P. 53–76.https://elischolar.library.yale.edu/cgi/viewcontent.cgi?article=1220&context=cowles-discussion-paper-series
2. Панюков А.В., Пельцвергер Б.В., Шафир А.Ю. Оптимальное размещение точек ветвления транспортной сети на цифровой модели местности // Автоматика и телемеханика, 1990. № 9. С. 153-162. https://www.mathnet.ru/links/ 9d86820cdcdad92b4658aa7a97cfb5af/at5941.pdf
3. Демиденко, В. М. Эффективно разрешимые случаи квадратичной задачи о назначениях с обобщенно монотонными и неполными матрицами Анти-Монжа / В. М. Демиденко, А. Долгий // Кибернетика и системный анализ. – 2007. – Т. 43, № 1. – С. 135-150. – EDN CBHJUA.
4. Демиденко, В. М. Обобщение условий сильной разрешимости квадратичной задачи о назначениях с матрицами анти-Монжа и Теплица / В. М. Демиденко // Доклады Национальной академии наук Беларуси. – 2003. – Т. 47, № 2. – С. 15-18. – EDN SJBIAN.
5. Архипова, Д. А. Сравнение эффективности генетического алгоритма и метода ветвей и границ для решения задачи размещения вершин графа / Д. А. Архипова, А. А. Галяутдинов, В. А. Суздальцев // Современные материалы, техника и технология : Сборник научных статей 9-й Международной научно-практической конференции. В 2-х томах, Курск, 28 декабря 2019 года / Ответственный редактор А.А. Горохов. Том 1. – Курск: Юго-Западный государственный университет, 2019. – С. 33-37. – EDN SAKDBD.
6. Комбинаторно-алгебраические методы в прикладной математике : Межвузовский сборник. – Горький : Горьковский государственный университет им. Н.И. Лобачевского, 1986. – 207 с. – EDN ZDKPPV.
7. Безопасные информационные технологии : Сборник трудов Десятой международной научно-технической конференции, Москва, 03–04 декабря 2019 года. – Москва: Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), 2019. – 409 с. – ISBN 978-5-9906630-6-0. – EDN QYLXMG.
8. Смирнов, В. А. Оптимальное проектирование в машиностроении в примерах и задачах / В. А. Смирнов. – Старый Оскол : ООО «Тонкие наукоемкие технологии», 2021. – 312 с. – ISBN 978-5-94178-697-8. – EDN CUHBLF.
9. Метельский Н.Н. Методы локальной оптимизации для задачи размещения двудольных графов // Ж. вычисл. матем. и матем. физ. 1984. Т. 24. № 9. С. 1428–1432.