МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, к.ф.-м.н. |  |  |  | Н.А. Волкова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ПОНЯТИЕ ТРАНСПОРТНОЙ ЗАДАЧИ (ТЗ). СЕТЕВАЯ ПОСТАНОВКА ТЗ |
| по дисциплине: ПРИКЛАДНЫЕ МОДЕЛИ ОПТИМИЗАЦИИ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z1431 |  |  |  | М.Д. Быстров |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc188225428)

[1 Транспортная задача 5](#_Toc188225429)

[1.1 Транспортная задача в классической форме 5](#_Toc188225430)

[1.2 Транспортная задача в сетевой форме 14](#_Toc188225431)

[2 Нахождение оптимального решения транспортной задачи в сетевой форме 17](#_Toc188225432)

[2.1 Постановка задачи и структура исходной информации 17](#_Toc188225433)

[2.2 Обоснование выбора метода решения транспортной задачи в сетевой форме 19](#_Toc188225434)

[2.3 Поиск решения с применением метода потенциалов 20](#_Toc188225435)

[ЗАКЛЮЧЕНИЕ 26](#_Toc188225436)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc188225437)

[ПРИЛОЖЕНИЕ А Исходный код программы 28](#_Toc188225438)

# ВВЕДЕНИЕ

Для обеспечения наибольшей эффективности в производственной деятельности в современной производственной среде активно применяются алгоритмы оптимизации, расчеты по которым производятся с использованием вычислительных систем.

Одной из важнейших задач планирования производства является планирование доставки ресурсов в соответствии с потребностями потребителей и наличием ресурсов у поставщиков, а также транспортных расходов (расстояний от каждого поставщика к каждому потребителю). Эта задача может быть формализована как транспортная задача, которая является одной из задач линейного программирования.

Условия транспортной задачи могут быть заданы в различной форме. На практике оценка параметров транспортной работы производится в том числе с использованием картографических данных, которые позволяют определить расходы по транспортировке груза от каждого поставщика к каждому потребителю. Сетевая постановка транспортной задачи позволяет в более наглядной форме представить начальные условия, т.к. является сравнительно более удобной для восприятия формой определения условий транспортной задачи.

Целью данной курсовой работы является рассмотрение понятия транспортной задачи и постановки транспортной задачи в сетевой форме.

Рассмотрение понятия транспортной задачи будет произведено с помощью общих содержательной и формализованной постановок транспортной задачи.

Рассмотрение сетевой постановки транспортной задачи будет дано как описание особенностей постановки задачи в сетевой форме, а также обусловленных этими особенностями модификаций метода потенциалов для решения ТЗ.

Объектом исследования является транспортная задача с постановкой в сетевой форме.

Предметом исследования является применение метода потенциалов для решения транспортной задачи в сетевой постановке.

Транспортная задача является задачей линейного программирования. Основы линейного программирования заложил советский математик Л.В. Канторович в работе 1939 г. «Математические методы организации и планирования производства». Американский математик Джордж Бернард Данциг в 1949 году разработал эффективный метод решения задач линейного программирования – симплекс-метод.

Применительно к транспортной задаче существует модификация симплекс-метода – метод потенциалов, который позволяет за конечное число итераций получить оптимальный план, основываясь на опорном решении. В свою очередь, для нахождения опорного решения существуют такие методы, как метод северо-западного угла, метод наименьшей стоимости.

Для сетевой постановки транспортной задачи также применим метод потенциалов. Будут рассмотрены различия алгоритма при классической и сетевой постановке задачи.

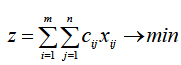
# 1 Транспортная задача

## 1.1 Транспортная задача в классической форме

Пусть имеются *m* пунктов отправления и *n* пунктов назначения груза. Обозначим через *сij,* стоимость перевозки груза из пункта отправления с номером *i* к пункту назначения с номером *j*, а через *xij* обозначим объём перевозки груза в пунктах отправления. Запасы груза в пунктах обозначим через *a1, a2, …am* , потребности пунктов назначений обозначим через *b1, b2, …bn*. Общую стоимость перевозки груза обозначим через формулы:



Необходимо уменьшить стоимость перевозки груза. Задача состоит в минимизации функции *z*:

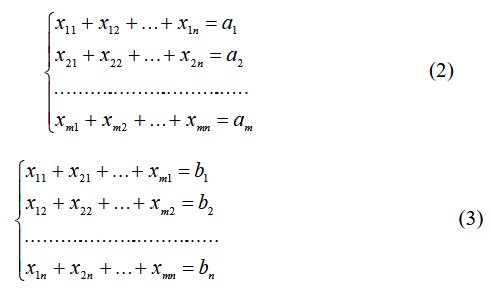


Задача может быть представлена с помощью таблицы:

*Таблица 1 Постановка ТЗ в классической форме*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *1* | *2* | *…* | *n* | Запасы |
| *1* | *c*11 *x*11 | *c*12 *x*12 | … | *c*1*n*  *x*1*n* | *a*1 |
| *2* | *c*21 *x*21 | *c*22 *x*22 | … | *c*2*n*  *x*2*n* | *a*2 |
| *…* | … | … | … | … | … |
| *m* | *cm*1  *xm*1 | *cm*2 *xm*2 | … | *cmn*  *xmn* | *am* |
| Потребности | *b*1 | *b*2 | … | *bn* |  |

Груз необходимо распределить между пунктами потребления, все грузы из пунктов отправления должны быть вывезены, все потребности пунктов назначения удовлетворены:

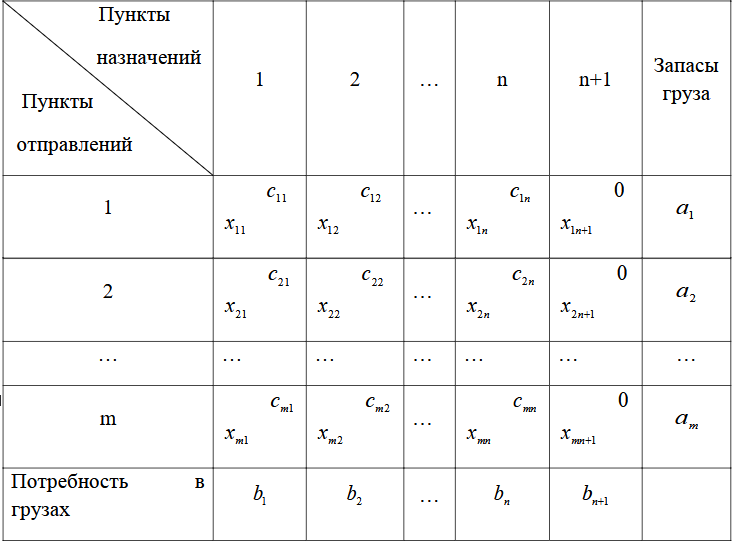


При выполнении условия  транспортная задача называется закрытой, и можно приступить к решению задачи.

Если истинно выражение , то задача является открытой. В этом случае задача приводится к закрытой с помощью ввода дополнительных пунктов.

Пример: при  добавляется пункт отправления с номером m+1, с запасами, равными  и со стоимостью перевозки груза, равной 0: . Задача принимает вид:

*Таблица 2 Пример приведения к закрытой ТЗ*



Если , аналогично добавляются дополнительные пункты отправлений с запасами груза, вследствие чего задача становится закрытой.

Для решения транспортной задачи может быть использован метод потенциалов.

Клетки с перевозками *xij* != 0 называются отмеченными, а клетки с перевозками *xij* = 0 называются не отмеченными. Для отмеченных клеток с помощью формулы *vj* -*ui* = *cij* определяем значения потенциалов *v j* , *j* =1,2,...*n* и *ui* , *i* =1,2,...*m*.

Задача решается в два этапа:

В первом этапе находится первоначальное решение *xij, i*=1*,*2*,*…*,m; j* =1*,*2*,*…*,n* , удовлетворяющее условиям (2)-(3). Имеются несколько способов для нахождения первоначального решения, например, метод северо-западного угла, метод минимального элемента и другие.

Метод северо-западного угла заключается в выборе клетки (1,1) и выборка объема поставки *x*11 = *min(a*1*,b*1*)* . Если *min(a*1*,b*1*)* = *a*1, то это означает, что все грузы из 1-го пункта отправления направлены к 1-пункту назначений, другим пунктам назначений из 1- пункта отправления груз не отправляется. Поэтому, к остальным клеткам в строке, где находится *a*1 вставляется знак «-». В 1- пункте назначения потребность в грузах будет *b*11 = *b*1 -*a*1.

*Таблица 3 Метод северо-западного угла*



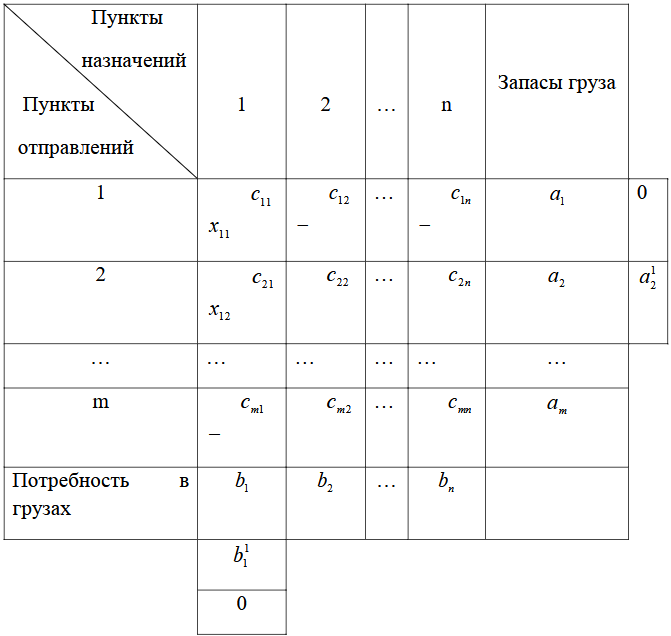
В ином случае, если *min(a*1*,b*1*)* =*b*1, то в 1- пункте назначения потребность в грузах будет удовлетворена, в 1-пункте отправления остаётся груз *a*11 = *a*1 -*b*1. К первому пункту назначения из остальных пунктов отправлений груз не привозится.

*Таблица 4 Метод северо-западного угла*



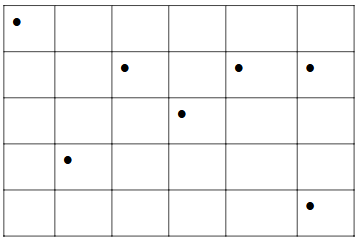
Продолжая вычисления по 1-таблице, переходим к клетке (2,1). Пусть будет *x*21 = *min*(*a*1*,b*11)=*b*11 . Заполняя клетку вышеуказанным способом, получаем следующее:

*Таблица 5 Метод северо-западного угла*



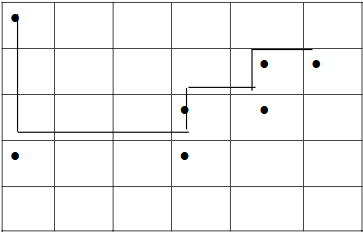
Продолжая вычисления таким образом до правого нижнего угла, определяем все значения *xij , i*=1*,*…*,m; j* =1*,*…*,n* . При этом должны выполняться условия (2)-(3).

На втором этапе находится оптимальное решение(план), удовлетворяющее условиям (1). Для нахождения оптимального плана имеется несколько способов, например метод потенциалов, метод распределений и т.д. Рассмотрим метод потенциалов. Для этого сначала ознакомимся с некоторыми понятиями. Произвольное множество точек в таблице называется набором. Например,



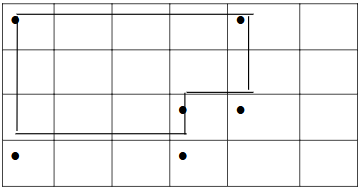
*Рисунок 1 Метод потенциалов*

Если в наборе число точек в каждой строке не превышает двух, то такой набор называется цепью. Например,



*Рисунок 2 Метод потенциалов*

Замкнутая цепь называется циклом. Например,



*Рисунок 3 Метод потенциалов*

Если в таблице набор из *n* количество точек не образуют цикл, при добавлении определенной точки набор *n*+1 точек образуют цикл, то первоначальный набор *n* точек называется ациклическим планом.

Если в транспортной задаче *xij* > 0, то клетка *(i,j)* называется отмеченной.

Если в транспортной задаче для всех клеток находится план *xij , i*=1*,*…*,m; j* =1*,*..*,n* , для которой удовлетворяется условие *vj* -*ui* <=*cij* (4) , а для отмеченных клеток удовлетворяется условие *v j* -*ui* = *cij* , то полученный план называется оптимальным. Множество чисел *v j ,* *j* =1*,*2*,...,n;* *ui ,* *i* =1*,*2*,...,m* называются потенциалами.

Метод потенциалов в транспортной задаче выполняется в следующем порядке:

1. Составляется система уравнений для отмеченных клеток удовлетворяющая следующим условиям *v j* -*ui* = *cij* , *v j ,* *j* =1*,*2*,...,n;* *ui ,* *i* =1*,*2*,...,m* . При этом число уравнений на одно меньше, чем число неизвестных. Поэтому система имеет бесконечное число решений. Найдя одно частное решение системы (приняв одно из неизвестных равным нулю), определим значение потенциалов;
2. Для неотмеченных клеток проверим условие *vj* -*ui* =*cij* . Если это условие выполняется для всех клеток, то этот план будет оптимальным, и вычисляется значение функции ;
3. Если условие *vj* -*ui* <= *cij* не выполняется для некоторых клеток, то для этих клеток вычисляем   
     
   и находим   
   ;
4. клетка *(i*0*,j*0 *)* добавляется в набор отмеченных клеток, и для этого набора составляется цикл;
5. начиная с клетки *(i*0*,j*0 *)*, по очереди вставляем знаки «+» и «-» к клеткам цикла. Вставка происходит, начиная со знака «+»;
6. для клеток со знаком «-» определяем   
   ;
7. Из чисел *xij* в клетках со знаком «-» вычитываем  , к числам *xij* в клетках со знаком «+» прибавляем ;
8. Клетка с  удаляется из числа отмеченных клеток.

В результате получаем новый план. Для нового плана повторяем операции (1)-(7). Вышеуказанные операции повторяются до тех пор, пока не выполняется условие *vj* -*ui* <=*cij* для всех клеток.

## 1.2 Транспортная задача в сетевой форме

Очень часто особенности исходной информации таковы, что возможно рассматривать их в сетевой постановке, и это позволяет иногда использовать более простые алгоритмы их решения. К числу таких задач относится и транспортная задача в сетевой постановке.

Пусть имеется ***N*** пунктов (производства, потребления, транзитной транспортировки грузов), связанных между собой некоторой транспортной сетью. В каждом пункте сети заданы числа

*ai* . Если *ai* < 0, то в этом пункте продукция производится, если а, > 0, то продукция потребляется, а если *ai* = 0, то данный пункт является транзитным, т.е. все, что в него привезено, должно быть вывезено.

Пусть транспортная сеть содержит *s* участков пути. Под участком пути понимается часть сети, соединяющая любые два ее пункта. Рассмотрим *q*-ый участок пути, на котором осуществляется перевозка:

*xq*

*iq cq jq*

где:

*iq* - пункт, из которого груз вывозится;

*jq* - пункт, в который груз завозится;

*cq* - стоимость перевозки единицы груза на этом участке пути:

*xq* - объем перевозки из пункта *iq*  в пункт *jq*.

Стрелка → показывает направление перевозки груза.

Требуется найти такой план перевозки груза, при котором из пунктов производства вывозится вся продукция, в пунктах потребления удовлетворяются их потребности, а суммарные затраты на перевозку грузов были бы минимальных, т.е. необходимо найти такой вектор X = (*x1, x2, … xs*), для которого:

 (4)

при условии

 , ; *, ;*

где

- объем груза, ввозимого в пункт ***i***,

 - объем груза, вывозимого из ***i*** .

Необходимым и достаточным условием разрешимости данной задачи является

; (5)

т.е. все, что произведено, должно быть потреблено.

Решение задачи осуществляется методом потенциалов. Опорный невырожденный план транспортной задачи должен содержать ровно *(N - 1)* положительную перевозку, не иметь замкнутых маршрутов и «висячих» пунктов.

Первоначальный опорный план строится по любому из существующих методов. Затем для каждого пункта сети находятся потенциалы по формуле

*Vjq - Vjq =Cq, Xq > 0*, (6)

где *Vjq* и *Vjq* - потенциалы пунктов, ограничивающих один и тот же *q*-ьм участок пути, a *Cq* - стоимость единицы перевозимого по этому участку груза.

Для оптимального плана транспортной задачи в сетевой постановке для всех участков пути должно выполняться условие:

 , . (7)

Если условие (7) не выполняется, то для тех участков пути, где оно не выполняется, рассчитывается невязка по формуле:

;

и на участке сети с наибольшей невязкой вводится *ε*-перевозка, определяется *ε*-маршрут, величина *ε*-перевозки, рассчитывается новый опорный план, который проверяется на оптимальность. Процедура повторяется до тех пор, пока не будет выполняться условие (7).

Преимущество сетевой постановки ТЗ относительно классической постановки заключается в том, что возможно простое расширение постановки до двухэтапной транспортной задачи, а также организации перевозок от поставщика к поставщику, когда это оказывается эффективнее, чем перевозки исключительно от поставщиков к потребителям. Эти улучшения достигаются всего лишь добавлением необходимых узлов и ребер в граф, представляющий собой постановку задачи. При этом изменений в методе поиска решения не требуется.

Также в сетевой форме при визуализации решения ТЗ является намного более простым для восприятия.

# 2 Нахождение оптимального решения транспортной задачи в сетевой форме

## 2.1 Постановка задачи и структура исходной информации

У поставщиков A1, A2, A3 есть некоторое количество груза, которое необходимо доставить в пункты потребления B1, B2, B3, B4. Затраты на перевозку одной единицы товара, запасы поставщиков и потребности потребителей отображены в графе на рисунке 4.

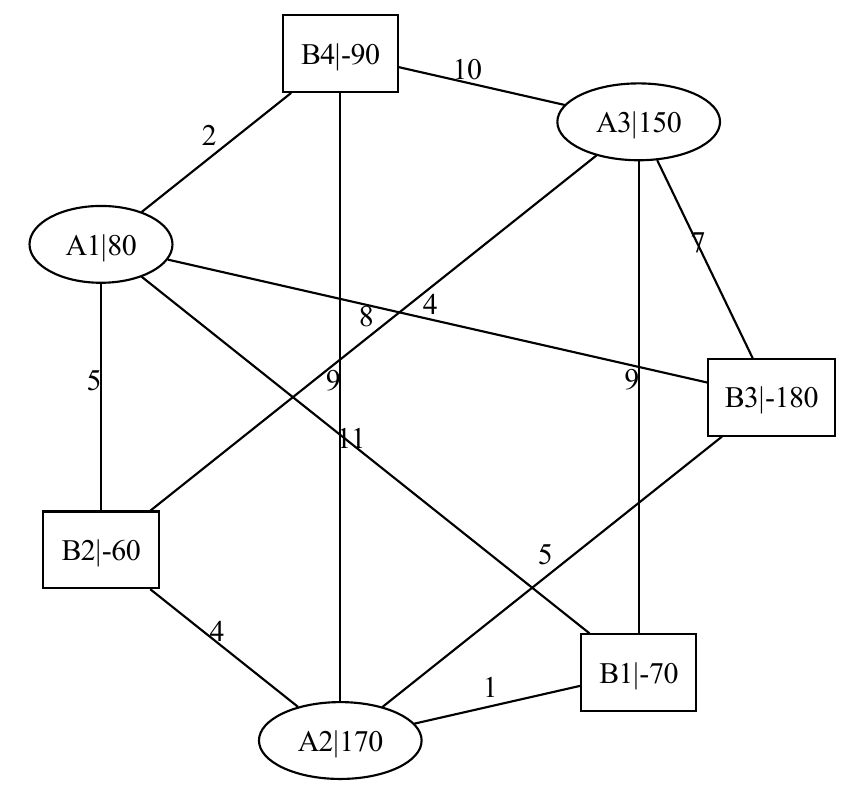


Рисунок 4 Постановка транспортной задачи в сетевой форме

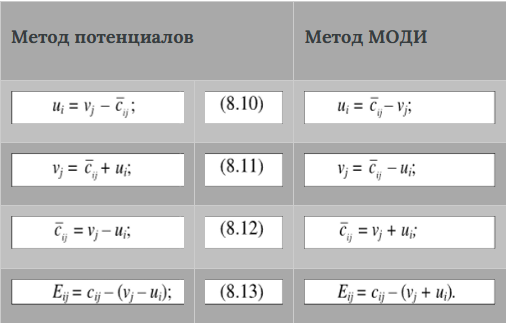
Необходимо найти такой план перевозок, при котором общие затраты на перевозку товара будут минимальны, а все потребности потребителей будут удовлетворены.

Поставщики изображены как узлы овальной формы, потребители – как узлы прямоугольной формы. Ребра между узлами показывают допустимость перевозки между поставщиками и потребителями. Числа, указанные в узлах – запасы и потребности каждого из пунктов. Если число отрицательное – оно отражает необходимость доставки товара в пункт потребления, если положительное – отражает наличие товара на складе поставщика.

## 2.2 Обоснование выбора метода решения транспортной задачи в сетевой форме

В ходе рассмотрения алгоритмов поиска решения транспортной задачи в сетевой форме было дано описание метода потенциалов для решения ТЗ в сетевой форме.

Для решения транспортной задачи в сетевой форме также может быть применен метод МОДИ (метод модифицированных распределений). Эти методы аналогичны, отличия в вычислениях отображены на рисунке 5.

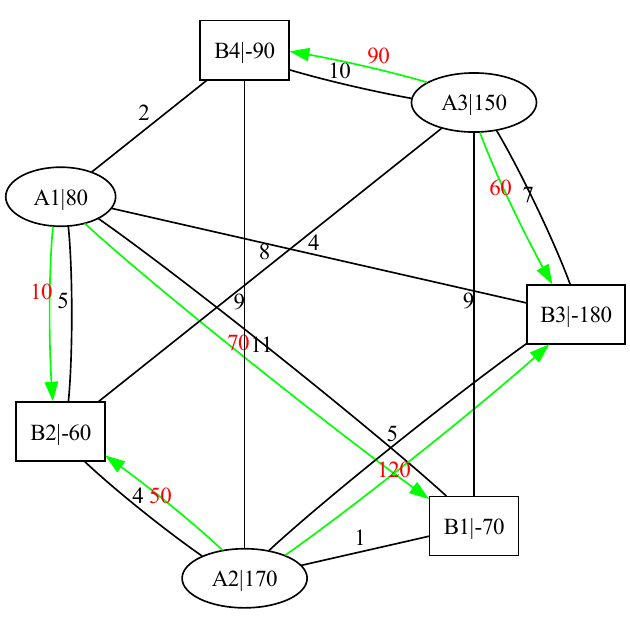


*Рисунок 5 Различия в методах потенциалов и МОДИ*

В ходе выполнения курсовой работы был сделан выбор в пользу метода потенциалов как более задокументированному при одинаковых временных и вычислительных затратах.

## 2.3 Поиск решения с применением метода потенциалов

Метод потенциалов позволяет перейти к оптимальному решению путем улучшения допустимого опорного плана. Для поиска этого опорного плана будет использоваться метод северо-западного угла. В соответствии с рассмотренным в гл. 1.1 алгоритмом найден опорный план, показанный на рисунке 6.



*Рисунок 6 Начальное условие с опорным планом*

Для отображения перевозок между пунктами на графе используются стрелки зеленого цвета. Для отображения объема перевозок рядом со стрелками указан объем перевозок (красный шрифт).

В соответствии с описанным в гл. 1.2 алгоритмом для поиска оптимального решения необходимо выполнить следующие действия:

1. Проверить текущий план на оптимальность путем расчета показателей незадействованных ребер, используя потенциалы узлов;
2. Если план не оптимален, ввести новую поставку для ребра без поставки с минимальным показателем, определив величину поставки как минимальную из противоположных поставок в замкнутом цикле поставок, при этом минимальное ребро удаляется;
3. Модифицировать ребра замкнутого цикла поставок путем прибавления величины нового ребра к попутным ребрам и вычитания той же величины из противоположных.
4. Повторить пункты 1-4, пока план не станет оптимальным.

Приведем пошаговое решение ТЗ созданной программой. Обозначения поставщиков и потребителей выполнены в виде их порядковых номеров. Порядковые номера начинаются с 0. Обозначения связей между поставщиками и потребителями представляют из себя пару порядковых номеров в круглых скобках, разделенных запятой. На рисунках изображены состояния перевозок после выполнения каждого шага. Второе число в каждом узле есть рассчитанный на пройденном шаге потенциал узла.

1. Шаг 1

Потенциалы поставщиков

{0: 100, 1: 101.0, 2: 99.0}

Потенциалы потребителей

{0: 111.0, 1: 105.0, 2: 106.0, 3: 109.0}

Оценки ребер без перемещений

{(0, 2): -2.0, (0, 3): -7.0, (1, 0): -9.0, (1, 3): 1.0, (2, 0): -3.0, (2, 1): 2.0}

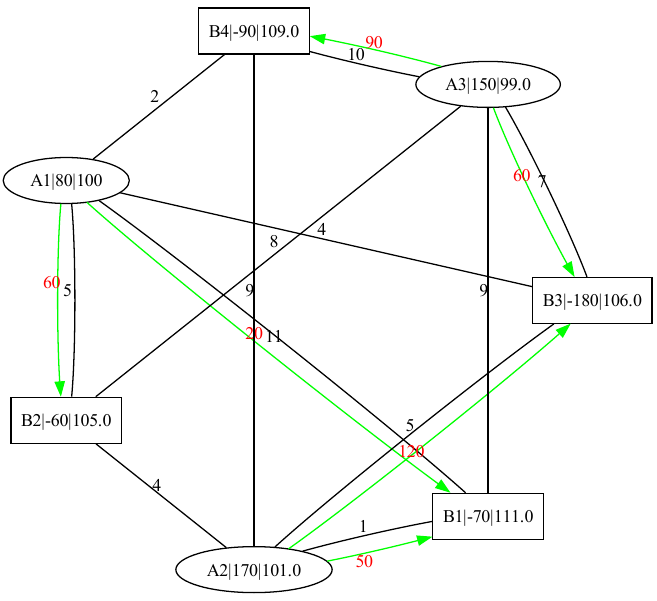
Найдена отрицательная хар-ка -9.0 у ребра (1, 0)

Найденный цикл, начиная с поставщика:

[0, 0, 1, 1]

Минимальная противоположная поставка 50.0 для ребра (1, 1)

Добавлено ребро A2-B1, удалено ребро A2-B2.



*Рисунок 7 Решение после шага 1*

1. Шаг 2

Потенциалы поставщиков

{0: 100, 1: 110.0, 2: 108.0}

Потенциалы потребителей

{0: 111.0, 1: 105.0, 2: 115.0, 3: 118.0}

Оценки ребер без перемещений

{(0, 2): -11.0, (0, 3): -16.0, (1, 1): -1.0, (1, 3): 1.0, (2, 0): 6.0, (2,

1): 5.0}

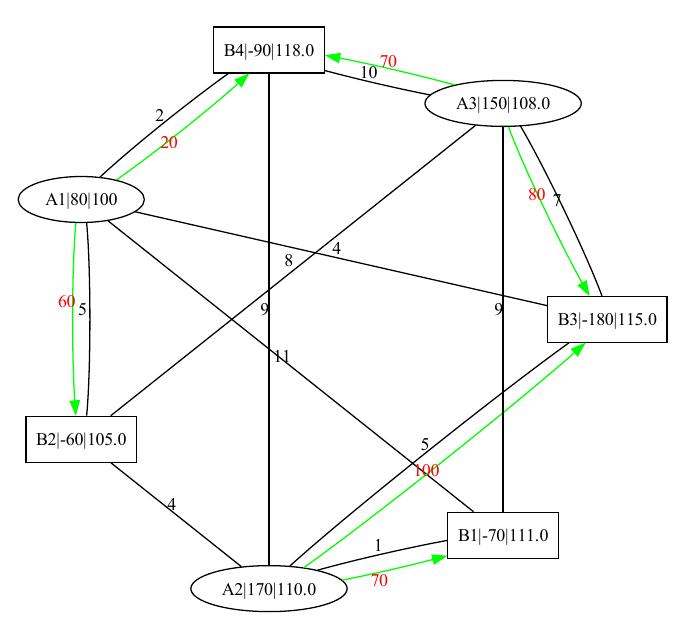
Найдена отрицательная хар-ка -16.0 у ребра (0, 3)

Найденный цикл, начиная с поставщика:

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

Минимальная противоположная поставка 20.0 для ребра (0, 0)

Добавлено ребро A1-B4, удалено ребро A1-B1.



*Рисунок 8 Решение после шага 2*

1. Шаг 3

Потенциалы поставщиков

{0: 100, 1: 94.0, 2: 92.0}

Потенциалы потребителей

{0: 95.0, 1: 105.0, 2: 99.0, 3: 102.0}

Оценки ребер без перемещений

{(0, 0): 6.0, (0, 2): 3.0, (1, 1): -7.0, (1, 3): 1.0, (2, 0): 6.0, (2, 1):

-5.0}

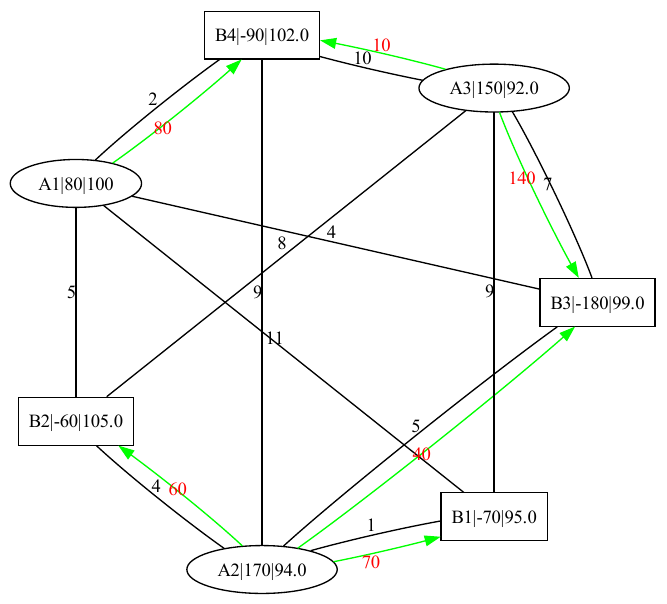
Найдена отрицательная хар-ка -7.0 у ребра (1, 1)

Найденный цикл, начиная с поставщика:

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

Минимальная противоположная поставка 60.0 для ребра (0, 1)

Добавлено ребро A2-B2, удалено ребро A1-B2.



*Рисунок 9 Решение после шага 3*

1. Шаг 4

Потенциалы поставщиков

{0: 100, 1: 94.0, 2: 92.0}

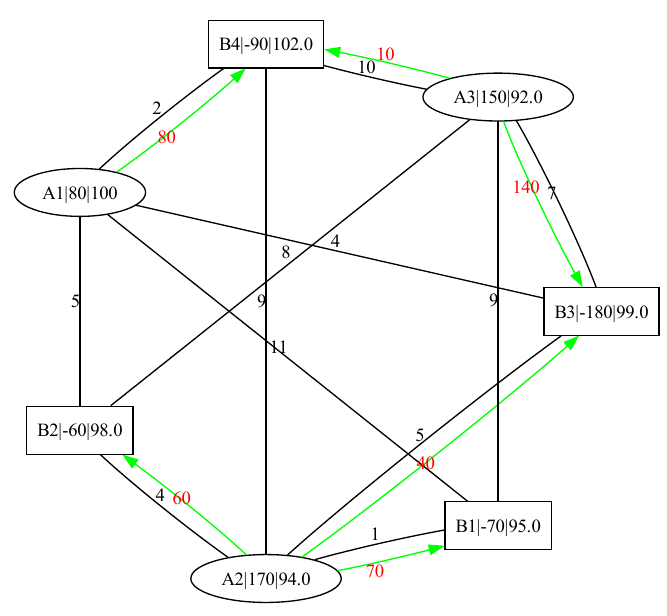
Потенциалы потребителей

{0: 95.0, 1: 98.0, 2: 99.0, 3: 102.0}

Оценки ребер без перемещений

{(0, 0): 6.0, (0, 1): 3.0, (0, 2): 3.0, (1, 3): 1.0, (2, 0): 6.0, (2, 1): 2.0}

Найдено оптимальное решение z=1750.0



*Рисунок 10 Итоговое решение ТЗ*

На рисунке 10 показано оптимальное решение транспортной задачи в сетевой форме. Минимизированное значение функции трудозатрат z = 1750.

# ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является программа для решения транспортной задачи в сетевой форме. Она предназначена для уменьшения затрат на доставку груза от пунктов производства до пунктов потребления.

Написанная программа позволяет визуализировать пошаговое решение транспортной задачи с помощью построения графов. Были определены и использованы при проектировании и реализации программы преимущества постановки транспортной задачи в сетевой форме, такие как двухэтапная транспортная задача или задача с перевозками между пунктами производства.

Программа написана на языке Python3, использует программное обеспечения для визуализации Graphviz, предназначена для запуска в ОС Windows.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рудик И.Д., Величко В.В. ПОНЯТИЕ, ВИДЫ И МЕТОДЫ РЕШЕНИЯ ТРАНСПОРТНОЙ ЗАДАЧИ // Международный студенческий научный вестник. – 2017. – № 4-4.

2. Лозгачёв И.А., Корепанов М.Ю. КЛАССИЧЕСКАЯ ТРАНСПОРТНАЯ ЗАДАЧА, РЕШЕННАЯ МЕТОДОМ ПОТЕНЦИАЛОВ // Международный студенческий научный вестник. – 2016. – № 3-1.

3. Бережная Е. В. Математические методы моделирования экономических систем: учеб. пособие. – 2-е изд., перераб. и доп. / Е. В. Бережная, В. И. Бережной. – М.: Финансы и статистика, 2006. – 432 с.

4. Большакова И.В. Линейное программирование: учебно-метод. пособие к контрольной работе для студ. эконом. факультета / И.В. Большакова, М.В. Кураленко. – Мн.: БНТУ. 2004. – 148 с.

5. Тюхтина А.А. Математические модели логистики. Транспортная задача: Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2016. – 66 с.

# ПРИЛОЖЕНИЕ А Исходный код программы

Программа написана на языке python3 с использованием программного решения визуализации графов Graphviz и python-библиотеки graphviz, которая предоставляет интерфейс для работы с ПО, установленным в системе.

Для запуска необходимо, чтобы Graphviz был установлен в системе, а все зависимости из import-директив были доступны интерпретатору.

Выполняться программа должна в ОС Windows.

import numpy as np;

import graphviz as gv;

import subprocess;

# метод северо-западного угла

def north\_west(paths, stocks, needs):

    moves=np.zeros(paths.shape);

    s = stocks.copy();

    n = needs.copy();

    excluded\_rows = set();

    excluded\_columns = set();

    while 1:

        rows = set(range(paths.shape[0])) - excluded\_rows;

        columns = set(range(paths.shape[1])) - excluded\_columns;

        if (len(rows) == 0 or len(columns) == 0): break;

        for i in rows:

            for j in columns:

                move = min(s[i], n[j]);

                moves[i][j] = move;

                if (move == s[i]):

                    excluded\_rows.add(i);

                    n[j] = n[j] - move;

                else:

                    excluded\_columns.add(j);

                    s[i] = s[i] - move;

                break;

            break;

    return moves;

# расчет потенциалов

# получить потенциалы

def get\_potentials(paths, moves):

    u = dict(); # потенциалы поставщика

    v = dict(); # потенциалы потребителя

    # кол-во u и v

    u\_num = paths.shape[0];

    v\_num = paths.shape[1];

    # для первого производителя (вершины)

    # берем потенциал = 100

    u[0] = 100;

    # расчет потенциалов перебором

    while (u.\_\_len\_\_() != u\_num

           or v.\_\_len\_\_() != v\_num):

        for prod in range(paths.shape[0]):

            for cons in range(paths.shape[1]):

                path = paths[prod][cons];

                move = moves[prod][cons];

                if (move != 0):

                    if prod in u and not(cons in v):

                        v[cons] = float(u[prod] + path);

                    elif cons in v and not(prod in u):

                        u[prod] = float(v[cons] - path);

    u = dict(sorted(u.items()));

    v = dict(sorted(v.items()));

    return [u,v];

def find\_edges\_scores(paths, moves, potentials):

    [u,v] = potentials;

    edges\_scores = {};

    # для ребер у которых нет стрелок считаем характеристику

    for prod in range(paths.shape[0]):

        for cons in range(paths.shape[1]):

            path = paths[prod][cons];

            move = moves[prod][cons];

            if (move > 0): continue;

            diff = u[prod] - v[cons];

            if diff < 0: diff = -1 \* diff;

            score = path - diff;

            edges\_scores[(prod, cons)] = float(score);

    min\_score = (min(edges\_scores.values())

        if len(edges\_scores.values()) > 0

        else 0);

    min\_edge = (-1, -1);

    for key, value in edges\_scores.items():

        if value == min\_score:

            min\_edge = key;

            break;

    return [edges\_scores, min\_edge, min\_score];

# поиск пути в графе

def find\_path(

    moves,

    source,

    dest = -1,

    source\_is\_cons = False,

    dest\_is\_cons = False,

    path = [],

    visited = set()):

    if (dest == -1): dest = source;

    localpath = path.copy();

    localpath.append(source);

    if (source\_is\_cons):

        for i in range(moves.shape[0]):

            if (moves[i][source] > 0

                and (i, source) not in visited):

                prod = i;

                # конец рекурсии - нашли конечную

                if (prod == dest

                    and dest\_is\_cons == False):

                    localpath.append(prod);

                    return localpath;

                localvisited = visited.copy();

                localvisited.add((prod, source));

                retpath = find\_path(

                    moves,

                    prod,

                    dest,

                    False,

                    dest\_is\_cons,

                    localpath,

                    localvisited);

                if (len(retpath)):

                    return retpath;

    else:

        for j in range(moves.shape[1]):

            if (moves[source][j] > 0

                and (source, j) not in visited):

                cons = j;

                # конец рекурсии - нашли конечную

                if (cons == dest

                    and dest\_is\_cons == True):

                    localpath.append(cons);

                    return localpath;

                localvisited = visited.copy();

                localvisited.add((source, j));

                retpath = find\_path(

                    moves,

                    cons,

                    dest,

                    True,

                    dest\_is\_cons,

                    localpath,

                    localvisited);

                if (len(retpath)):

                    return retpath;

    # пустой список - через эту вершину цикл найти не удалось

    return [];

# переместить ребро

def move\_edge(paths, moves, min\_edge):

    # новая матрица перемещений

    m = moves.copy();

    (min\_edge\_prod, min\_edge\_cons) = min\_edge;

    # поиск замкнутого цикла для добавляемого ребра

    cycle = find\_path(

        moves,

        min\_edge\_cons,

        min\_edge\_prod,

        True);

    print("Найденный цикл, начиная с поставщика:");

    print(cycle);

    min\_value = float("inf");

    new\_min\_edge = (-1,-1);

    # определяем минимальное противоположное ребро

    for i in range(len(cycle) - 1):

        # интересуют только стрелки из потребителей,

        # по направлению противоположные

        if (i % 2 == 0):

            (cons, prod) = (cycle[i], cycle[i+1]);

            if moves[prod][cons] < min\_value:

                new\_min\_edge = (prod, cons);

                min\_value = moves[prod][cons];

    if (new\_min\_edge == (-1,-1)): raise Exception("Can't found min move");

    print(f"Минимальная противоположная поставка {min\_value} для ребра {new\_min\_edge}");

    # определяем новое распределение поставок

    for i in range(len(cycle) - 1):

        # вычитаем из противоположных ребер

        # добавляем к попутным

        if (i % 2 == 0):

            (cons, prod) = (cycle[i], cycle[i+1]);

            m[prod][cons] = m[prod][cons] - min\_value;

        else:

            (prod, cons) = (cycle[i], cycle[i+1]);

            m[prod][cons] = m[prod][cons] + min\_value;

    # новое ребро - перемещение

    m[min\_edge\_prod][min\_edge\_cons] = min\_value;

    return m;

# решение транспортной задачи

def solve(paths, stocks, needs):

    stage = 1;

    # начальный граф

    draw\_graph(

        paths,

        stocks,

        needs,

        name = f"Начальный\_граф");

    # начальное допустимое опорное решение

    # методом северо-западного угла

    moves = north\_west(

        paths,

        stocks,

        needs);

    # начальный граф

    draw\_graph(

        paths,

        stocks,

        needs,

        moves,

        name = f"Начальный\_граф\_с\_опорным");

    while (1):

        # поиск потенциалов узлов

        [u, v] = get\_potentials(paths, moves);

        print("Потенциалы поставщиков");

        print(u);

        print("Потенциалы потребителей");

        print(v);

        # поиск оценок ребер без перемещений

        [edges\_scores,

        min\_edge,

        min\_score] = find\_edges\_scores(paths, moves, [u, v]);

        print("Оценки ребер без перемещений");

        print(edges\_scores);

        # если среди ребер без перемещений

        # есть отрицательные характеристики,

        # надо сдвинуть план

        if (min\_score < 0):

            print(f"Найдена отрицательная хар-ка {min\_score} у ребра {min\_edge}");

            moves = move\_edge(paths, moves, min\_edge);

        # отрисовка графа на текущем шаге

        draw\_graph(

            paths,

            stocks,

            needs,

            moves,

            [u, v],

            f"Шаг\_{stage}");

        # не нашлось отрицательных характеристик

        # план оптимален, выход

        if (min\_score >= 0):

            break;

        stage = stage + 1;

    # найдено оптимальное решение,

    # считаем минимизированное значение

    sum = 0;

    for i in range(paths.shape[0]):

        for j in range(paths.shape[1]):

            if (moves[i][j]):

                sum += moves[i][j] \* paths[i][j];

    print(f"Найдено оптимальное решение z={sum}");

# отрисовка графа

def draw\_graph(

    paths, # пути от поставщиков к потребителям

    stocks = np.array([]), # запасы

    needs = np.array([]), # потребности

    moves = np.array([]), # движения ресурсов

    potentials = [{},{}], # расчитанные потенциалы

    name = "graph"

    ):

    dot = gv.Digraph(engine="circo");

    nodes = set();

    [u,v] = potentials;

    for i in range(paths.shape[0]):

        for j in range(paths.shape[1]):

            from\_node = f"A{i+1}";

            to\_node = f"B{j+1}";

            if from\_node not in nodes:

                node = f"{from\_node}|{stocks[i]}";

                if i in u: node = f"{node}|{u[i]}";

                dot.node(from\_node, node);

                nodes.add(from\_node);

            if to\_node not in nodes:

                node = f"{to\_node}|{-needs[j]}";

                if j in v: node = f"{node}|{v[j]}";

                dot.node(to\_node, node, shape="rect");

                nodes.add(to\_node);

            dot.edge(

                from\_node,

                to\_node,

                label=f"{int(paths[i][j])}",

                dir="none");

    for i in range(moves.shape[0]):

        for j in range(moves.shape[1]):

            if (moves[i][j] != 0):

                from\_node = f"A{i+1}";

                to\_node = f"B{j+1}";

                dot.edge(

                    from\_node,

                    to\_node,

                    label=f"{int(moves[i][j])}",

                    color="green",

                    fontcolor="red");

    # dot.graph\_attr['ratio'] = "compress";

    dot.graph\_attr['size'] = "1920,1080";

    # print(dot.source);

    file = dot.render(f'output/{name}').replace('\\', '/');

    subprocess.run(["cmd", f"/c start {file}"]);

stocks = [80, 170, 150]; # запасы

needs = [70, 60, 180, 90]; # потребности

paths = np.array(

    #B1  B2  B3  B4

    [

    [11,  5,  4,  2], #A1

    [ 1,  4,  5,  9], #A2

    [ 9,  8,  7, 10], #A3

    ],

    dtype = float);

solve(paths, stocks, needs);