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

1 Транспортная задача 5

1.1 Транспортная задача в классической форме 5

1.2 Транспортная задача в сетевой форме 14

2 Нахождение оптимального решения транспортной задачи в сетевой форме 17

2.1 Постановка задачи и структура исходной информации 17

2.2 Обоснование выбора метода решения транспортной задачи в сетевой форме 19

2.3 Поиск решения с применением метода потенциалов 20

ЗАКЛЮЧЕНИЕ 26

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27

ПРИЛОЖЕНИЕ А Исходный код программы 28

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 c_{ij}, стоимость перевозки груза из пункта отправления с номером i к пункту назначения с номером j, а через x_{ij} обозначим объём перевозки груза в пунктах отправления. Запасы груза в пунктах обозначим через a_1, a_2, $\dots a_m$, потребности пунктов назначений обозначим через b_1, b_2, $\dots b_n$. Общую стоимость перевозки груза обозначим через формулы:

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

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

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \rightarrow min$$

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_{1n} x_{1n}	a_1
2	c_{21} x_{21}	c_{22} x_{22}	…	c_{2n} x_{2n}	a_2
…	…	…	…	…	…
m	c_{m1} x_{m1}	c_{m2} x_{m2}	…	c_{mn} x_{mn}	a_m
Потребности	b_1	b_2	…	b_n	

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

$$\begin{cases} x_{11} + x_{12} + \ldots + x_{1n} = a_1 \\ x_{21} + x_{22} + \ldots + x_{2n} = a_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{m1} + x_{m2} + \ldots + x_{mn} = a_m \end{cases} \qquad (2)$$

$$\begin{cases} x_{11} + x_{21} + \ldots + x_{m1} = b_1 \\ x_{12} + x_{22} + \ldots + x_{m2} = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{1n} + x_{2n} + \ldots + x_{mn} = b_n \end{cases} \qquad (3)$$

При выполнении условия $\sum_{i=1}^{n} b_i = \sum_{j=1}^{m} a_j$ транспортная задача называется закрытой, и можно приступить к решению задачи.

Если истинно выражение $\sum_{i=1}^{n} b_i \neq \sum_{j=1}^{m} a_j$, то задача является открытой. В этом случае задача приводится к закрытой с помощью ввода дополнительных пунктов.

Пример: при $\sum_{i=1}^{n} b_i > \sum_{j=1}^{m} a_j$ добавляется пункт отправления с номером m+1, с запасами, равными $a_{m+1} = \sum_{i=1}^{n} b_i - \sum_{j=1}^{m} a_j$ и со стоимостью перевозки груза, равной 0: $c_{1,n+1} = c_{2,n+1} = \ldots = c_{m,n+1} = 0$. Задача принимает вид:

Таблица 2 Пример приведения к закрытой ТЗ

Пункты назначений / Пункты отправлений	1	2	…	n	n+1	Запасы груза
1	c_{11} x_{11}	c_{12} x_{12}	…	c_{1n} x_{1n}	0 x_{1n+1}	a_1
2	c_{21} x_{21}	c_{22} x_{22}	…	c_{2n} x_{2n}	0 x_{2n+1}	a_2
…	…	…	…	…	…	…
m	c_{m1} x_{m1}	c_{m2} x_{m2}	…	c_{mn} x_{mn}	0 x_{mn+1}	a_m
Потребность в грузах	b_1	b_2	…	b_n	b_{n+1}	

Если $\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j$, 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 x_{ij} != 0 называются отмеченными, а клетки с перевозками $x_{ij} = 0$ называются не отмеченными. Для отмеченных клеток с помощью формулы $v_j - u_i = c_{ij}$ определяем значения потенциалов v_j, j =1,2,...n и u_i, i =1,2,...m.

Задача решается в два этапа:

В первом этапе находится первоначальное решение x_{ij}, 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 $b_1^1 = b_1 - a_1$.

Таблица 3 Метод северо-западного угла

Пункты назначений / Пункты отправлений	1	2	…	n	Запасы груза	
1	c_{11} x_{11}	c_{12} –	…	c_{1n} –	a_1	0
2	c_{21}	c_{22}	…	c_{2n}	a_2	
…	…	…	…	…	…	
m	c_{m1}	c_{m2}	…	c_{mn}	a_m	
Потребность в грузах	b_1	b_2	…	b_n		
	b_1^1					

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 $a_1^1 = a_1 - b_1$.

К первому пункту назначения из остальных пунктов отправлений груз не привозится.

Таблица 4 Метод северо-западного угла

Пункты назначений / Пункты отправлений	1	2	…	n	Запасы груза	
1	c_{11} x_{11}	c_{12}	…	c_{1n}	a_1	a_1^1
2	c_{21} _	c_{22}	…	c_{2n}	a_2	
…	…	…	…	…	…	
m	c_{m1} _	c_{m2}	…	c_{mn}	a_m	
Потребность в грузах	b_1	b_2	…	b_n		
	0					

Продолжая вычисления по 1-таблице, переходим к клетке (2,1). Пусть будет $x_{21} = min(a_1, b_1^1) = b_1^1$. Заполняя клетку вышеуказанным способом, получаем следующее:

Таблица 5 Метод северо-западного угла

Пункты назначений / Пункты отправлений	1	2	…	n	Запасы груза	
1	c_{11} x_{11}	c_{12} –	…	c_{1n} –	a_1	0
2	c_{21} x_{12}	c_{22}	…	c_{2n}	a_2	a_2^1
…	…	…	…	…	…	
m	c_{m1} –	c_{m2}	…	c_{mn}	a_m	
Потребность в грузах	b_1	b_2	…	b_n		
	b_1^1					
	0					

Продолжая вычисления таким образом до правого нижнего угла, определяем все значения x_{ij}, 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,

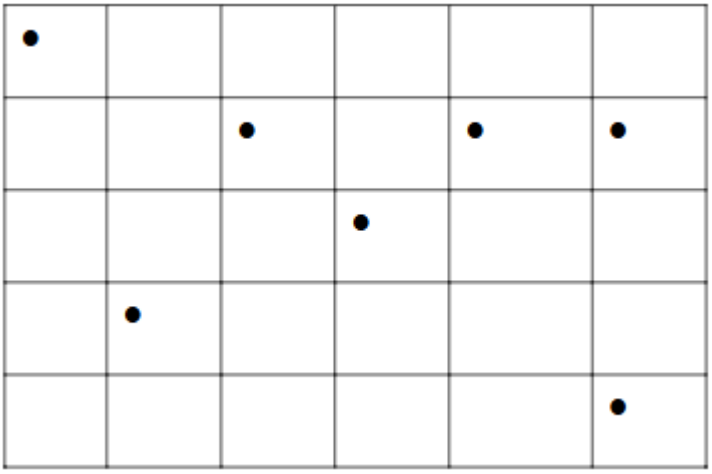


Рисунок 1 Метод потенциалов

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

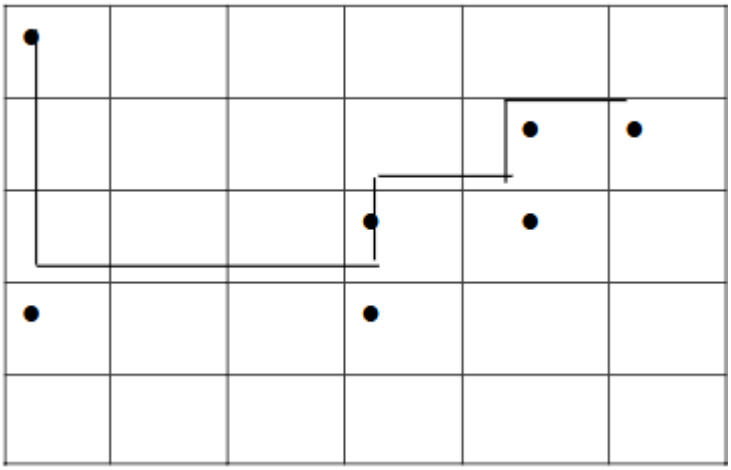


Рисунок 2 Метод потенциалов

Замкнутая цепь называется циклом. Например,

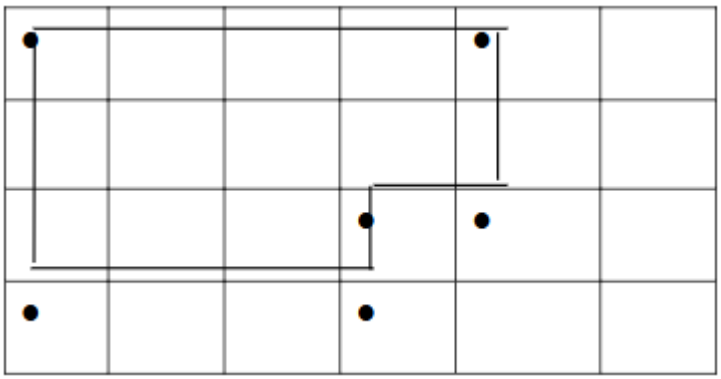


Рисунок 3 Метод потенциалов

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 $x_{ij} > 0$, то клетка (i,j) называется отмеченной.

Если в транспортной задаче для всех клеток находится план x_{ij}, $i=1,...,m;$ $j=1,..,n$, для которой удовлетворяется условие $v_j - u_i <= c_{ij}$ (4), а для отмеченных клеток удовлетворяется условие $v_j - u_i = c_{ij}$, то полученный план называется оптимальным. Множество чисел v_j, $j=1,2,...,n;$ u_i, $i=1,2,...,m$ называются потенциалами.

Метод потенциалов в транспортной задаче выполняется в следующем порядке:

1. Составляется система уравнений для отмеченных клеток удовлетворяющая следующим условиям $v_j - u_i = c_{ij}$, v_j, $j=1,2,...,n;$ u_i, $i=1,2,...,m$. 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 $v_j - u_i = c_{ij}$. Если это условие выполняется для всех клеток, то этот план будет оптимальным, и вычисляется значение функции $z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$;

3. Если условие $v_j - u_i <= c_{ij}$ не выполняется для некоторых клеток, то для этих клеток вычисляем

$$\delta_{ij} = v_j - u_i - c_{ij}$$

и находим

$$\delta_{i_0 j_0} = \max_{i,j} \delta_{ij}$$;

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 $\theta = min(x_{ij})$;
7. Из чисел x_{ij} в клетках со знаком «-» вычитываем θ, к числам x_{ij} в клетках со знаком «+» прибавляем θ;
8. Клетка с θ 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 $v_j - u_i <= c_{ij}$ для всех клеток.

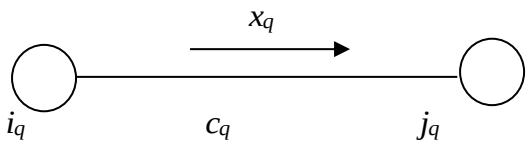
1.2 Транспортная задача в сетевой форме

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

a_i $\left(i = \overline{1, N}\right)$. Если a_i < 0, то в этом пункте продукция производится, если а, > 0, то продукция потребляется, а если a_i = 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:



где:

i_q - пункт, из которого груз вывозится;

j_q - пункт, в который груз завозится;

c_q - стоимость перевозки единицы груза на этом участке пути:

x_q - объем перевозки из пункта i_q в пункт j_q.

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 = ($x_1, x_2, \ldots x_s$), для которого:

$$\sum_{q=1}^{s} c_q \cdot x_q \to \min \qquad (4)$$

при условии

$$\sum_{j_q=1}^{N} x_q - \sum_{i_q=1}^{N} x_q = a_i \ , \quad i, j = \overline{1, N}; \quad x_q \geq 0, \quad q = \overline{1, s};$$

где

$\sum_{j_q} x_q$ - объем груза, ввозимого в пункт ***i***,

$\sum_{i_q} x_q$ - объем груза, вывозимого из ***i*** .

Необходимым и достаточным условием разрешимости данной задачи является

$$\sum_{i=1}^{N} a_i = 0; \qquad (5)$$

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

$$V_{jq} - V_{jq} = C_q, \quad X_q > 0, \qquad (6)$$

где V_{jq} и V_{jq} - потенциалы пунктов, ограничивающих один и тот же q-ьм участок пути, а C_q - стоимость единицы перевозимого по этому участку груза.

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

$$\left| V_{jq} - V_{iq} \right| \leq c_q \ , \qquad \forall_q = \overline{1, s} \ . \qquad (7)$$

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

$$\eta = \left| V_{jq} - V_{iq} \right| - c_q;$$

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.

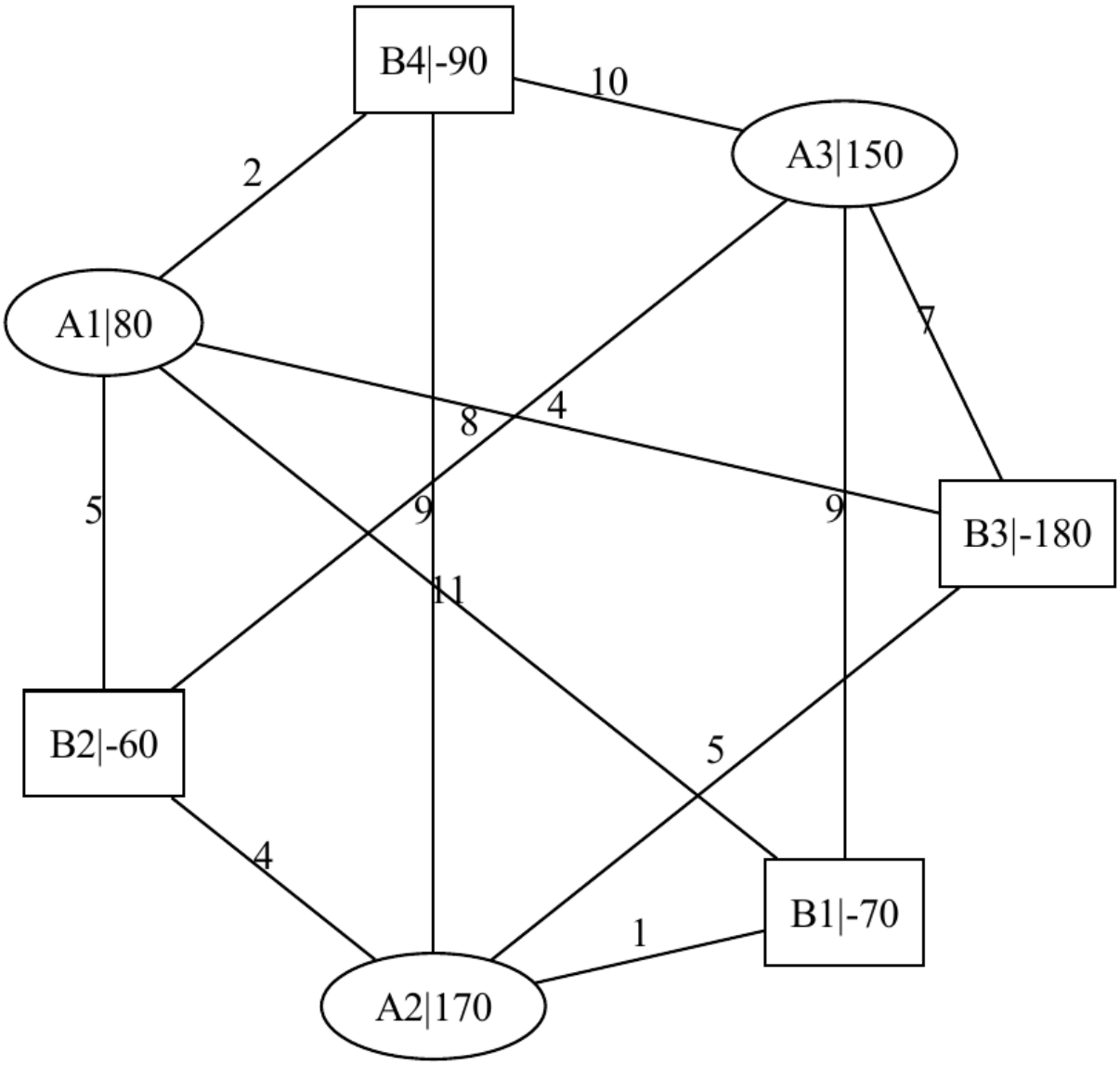


Рисунок 4 Постановка транспортной задачи в сетевой форме

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.

Метод потенциалов		Метод МОДИ
$u_i = v_j - \bar{c}_{ij};$	(8.10)	$u_i = \bar{c}_{ij} - v_j;$
$v_j = \bar{c}_{ij} + u_i;$	(8.11)	$v_j = \bar{c}_{ij} - u_i;$
$\bar{c}_{ij} = v_j - u_i;$	(8.12)	$\bar{c}_{ij} = v_j + u_i;$
$E_{ij} = c_{ij} - (v_j - u_i);$	(8.13)	$E_{ij} = c_{ij} - (v_j + u_i).$

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.

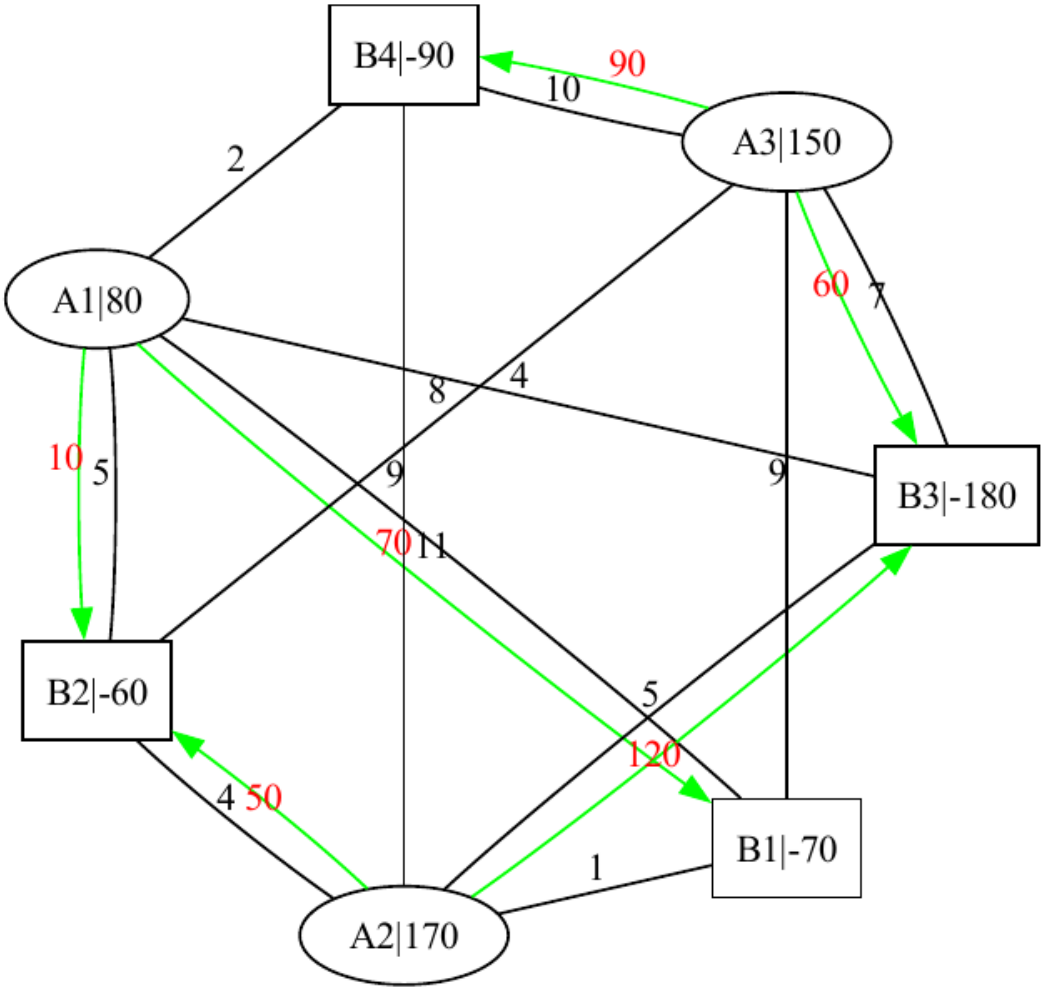


Рисунок 6 Начальное условие с опорным планом

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.

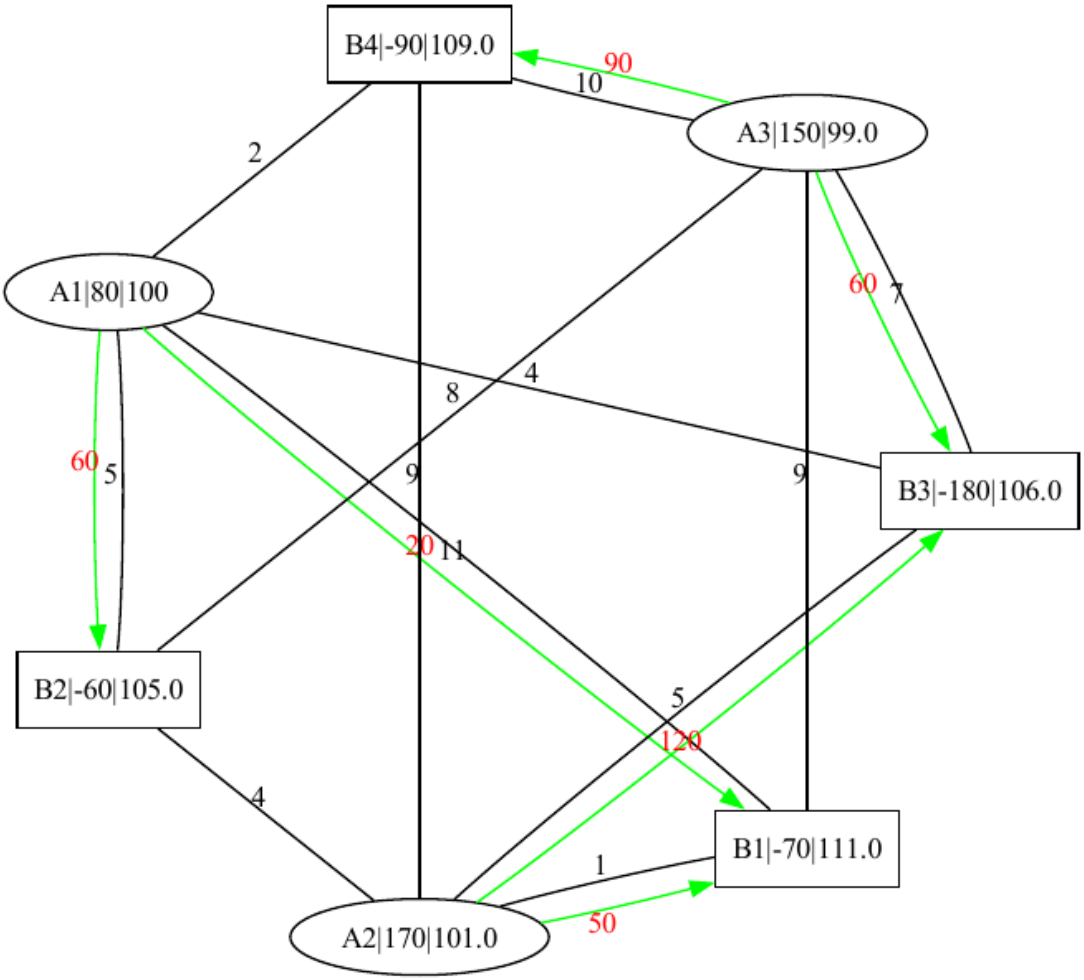


Рисунок 7 Решение после шага 1

2. 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.

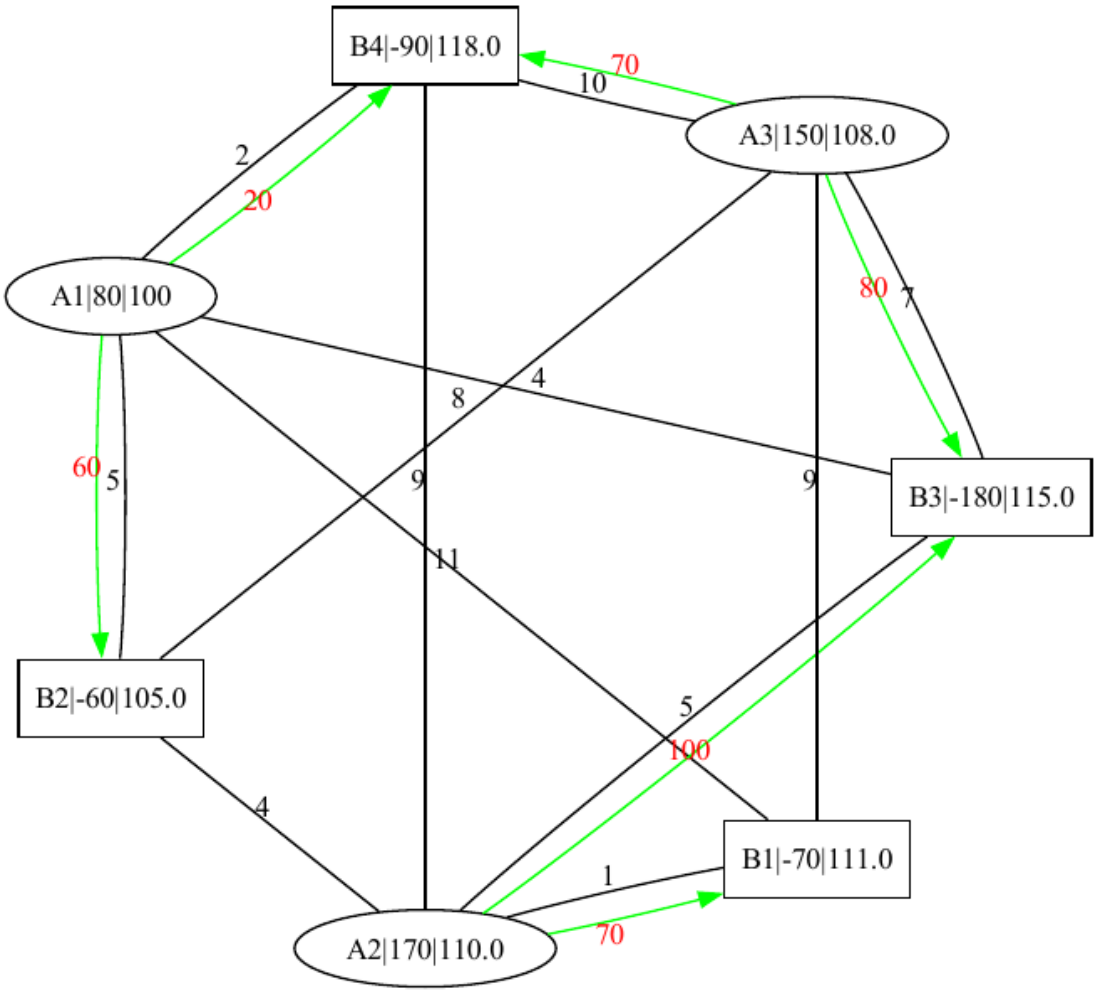


Рисунок 8 Решение после шага 2

3. 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.

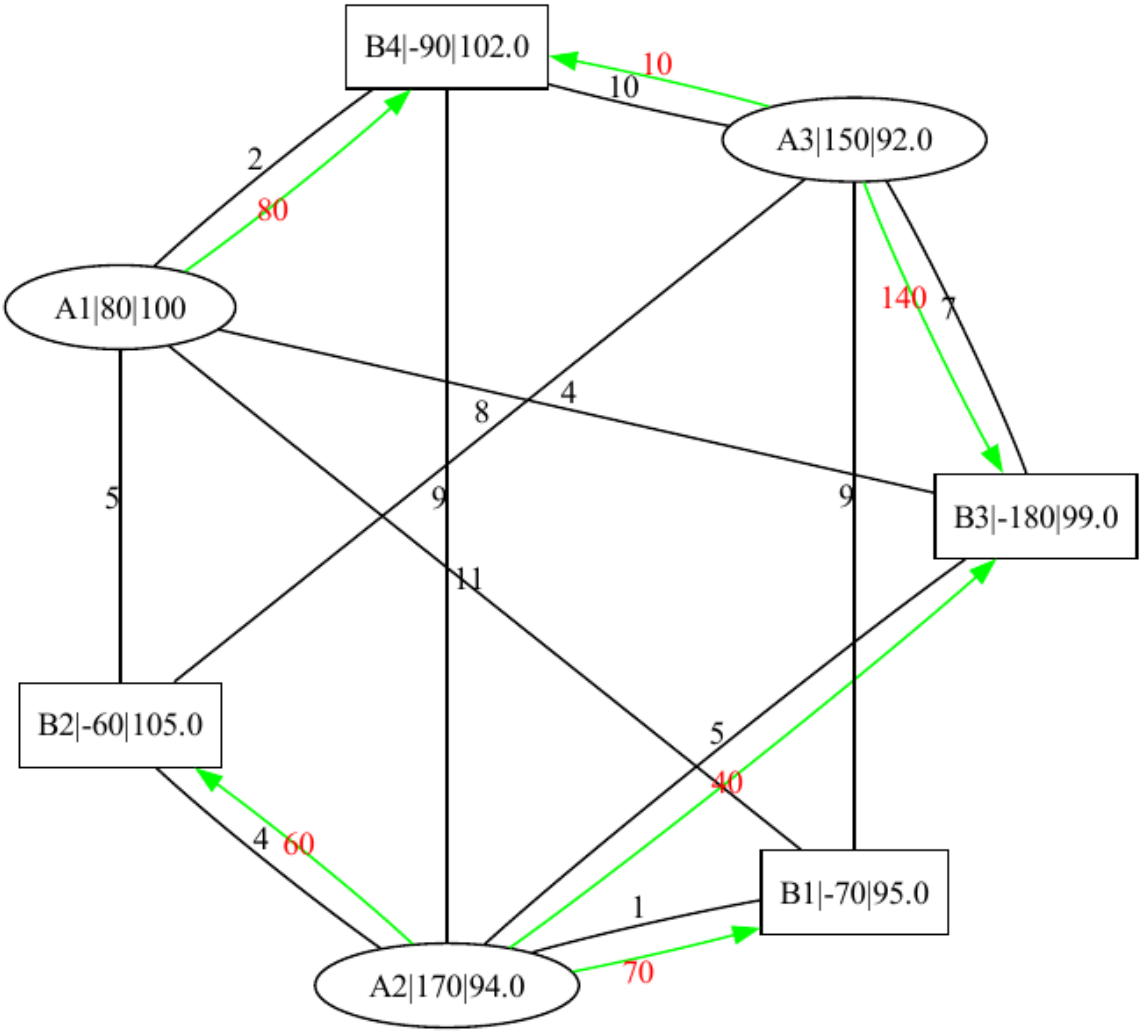


Рисунок 9 Решение после шага 3

4. 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

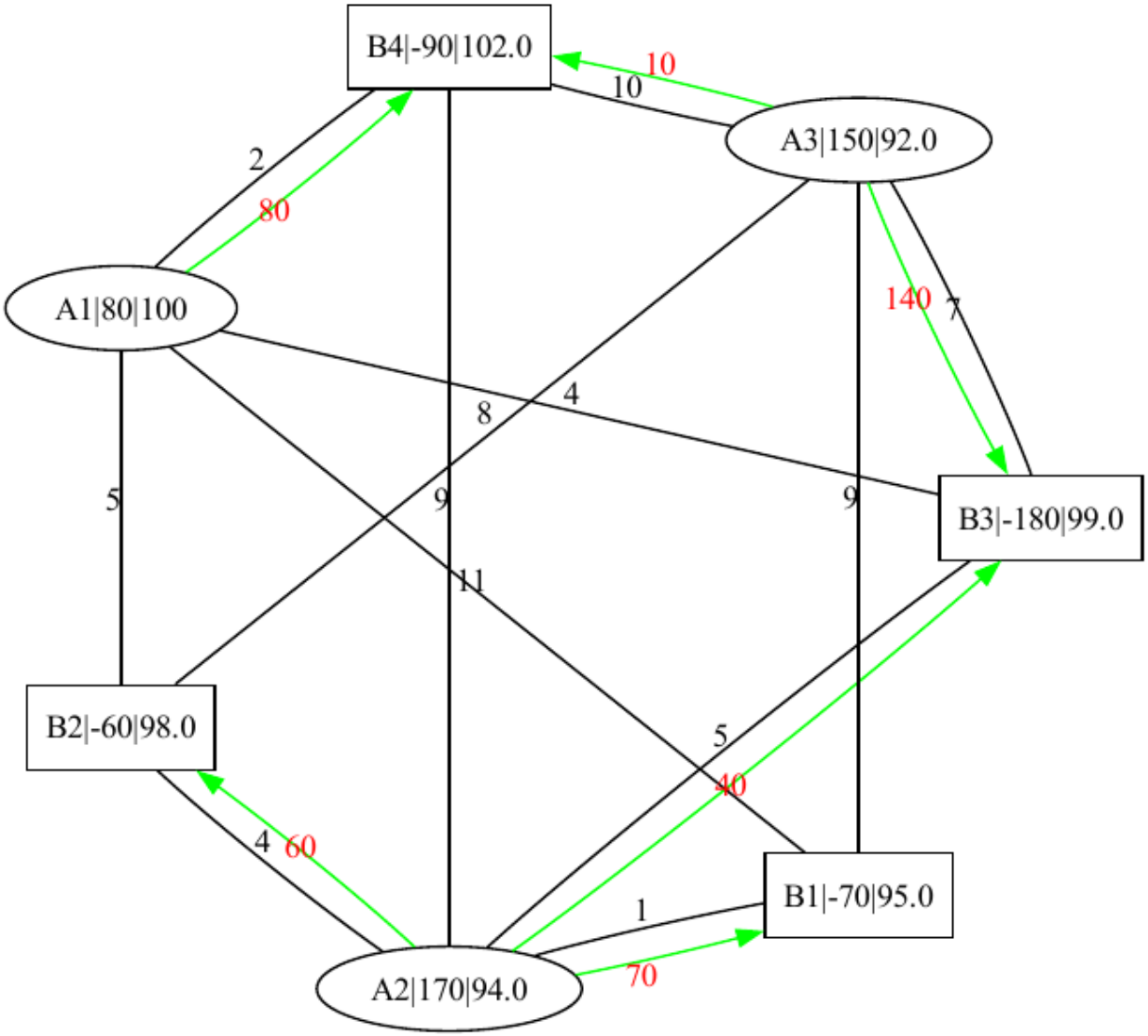


Рисунок 10 Итоговое решение ТЗ

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);
```