МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

WEB-дизайн и мультимедиа
(направленность (профиль)/ специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему «Сравнительный анализ методов решения транспортной задачи»

Студент В.Д. Амоналишоев
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., доцент, Н. А. Сосина
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема бакалаврской работы: «Сравнительный анализ методов решения транспортной задачи».

Бакалаврская работа посвящена сравнению методов решения транспортной задачи с дополнительными условиями с использованием библиотек Python.

В ходе выполнения исследований по бакалаврской работе на языке Python было написано несколько вариаций программного обеспечения для решения транспортной задачи с дополнительными условиями. В каждом варианте используется различные программные методы решения транспортных задач – cvxopt, scipy.optimize и pulp. По итогам вычислительных экспериментов проведен сравнительный анализ данных методов.

Бакалаврская работа состоит из введения, трёх глав, заключения и списка литературы. В введении описывается актуальность проводимого исследования, дается краткая характеристика проделанной работы.

В первой главе анализируются проблемы линейного программирования.

Во второй главе описывается решение задач линейного программирования с использованием Python.

В третьей главе описываются программные реализации решения транспортной задачи с дополнительными условиями с использованием различных методов. Приводится обсуждение результатов вычислительных экспериментов.

В заключении представлены выводы по проделанной работе.

В работе присутствуют 28 рисунков, 20 формул. Список литературы состоит из 20 литературных источников. Общий объем выпускной квалификационной работы составляет 40 страницы.

ABSTRACT

The present graduation work is devoted to comparative analysis of the methods for solving a transport problem.

The research deals with comparing the methods for solving the transport problem with additional conditions by using Python libraries.

While conducting the research, several versions of software were written in Python language to solve a transport problem with additional conditions. Each version uses different software methods for solving the transport problems, for example, *cvxopt*, *scipy.optimize* and *pulp*. Based on the results of the computational experiments, a comparative analysis of the methods in question has been carried out.

The graduation work consists of an introduction, 3 chapters, a conclusion, 28 figures, 20 formulas and the list of 20 references.

The introduction explains the relevance of the research carried out and provides a brief description of the work done.

The first chapter analyzes the problems of linear programming.

The second chapter dwells on solving the linear programming problems by using Python.

The third chapter gives full coverage to software implementation to solve the transport problem with additional conditions by applying various methods. It also discusses the results of the computational experiments.

Содержание

Введение

В настоящее время площадь России по официальным данным составляет порядка 17,13 миллионов квадратных километров. При этом количество населенных пунктов по данным общероссийского классификатора территорий муниципальных образований равно 155649. С учетом данных фактов, продумывание логистики доставки товаров превращается в большую исследовательскую задачу, требующую использования компьютерных мощностей для ее решения.

С точки зрения математики логистическую задачу можно свести к решению оптимизационной задачи линейного программирования (к транспортной задаче) [1, 3]. От успеха решения транспортной задачи, например, зависит количество издержек транспортной компании при доставке заказов. Крупные интернет-магазины также вынуждены периодически решать транспортные задачи для своевременного обеспечения своих клиентов заказами.

Одновременно с этим сложность транспортных задач постоянно увеличивается, и для их решения привлекаются компьютерные мощности [5]. Поэтому увеличивается актуальность в разработке программных комплексов для решения задач линейного программирования. При создании программных комплексов часто используется компонентный подход, когда вместо написания своей реализации алгоритма, применяются свободно распространяемые библиотеки. С одной стороны это упрощает разработку программного обеспечения, но с другой – у каждой реализации есть скрытые особенности, определить которые можно только в условиях вычислительных экспериментов.

В настоящее время существует множество библиотек для языка Python, в которых реализованы методы, позволяющие решать задачи линейного программирования [10, 12, 14]. Из их числа наиболее известные свободно распространяемые библиотеки с открытым исходным кодом –

cvxopt, SciPy и PuLp. В этих библиотеках для решения задач линейного программирования используются различные реализации симплекс-метода.

Из-за различий в реализациях сложно оценить, как поведут себя реализованные в этих библиотеках методы при решении одной и той же транспортной задачи с дополнительными условиями. Поэтому актуальной проблемой является разработка программного обеспечения для сравнительного анализ методов решения транспортной задачи.

Цель работы – разработка программного обеспечения для сравнения различных программных методов решения транспортных задач с дополнительными условиями.

Для достижения поставленной цели в работе решаются следующие задачи:

1. Анализ современных аспектов линейного программирования.
2. Исследование подходов по решению задач линейного программирования с использованием Python.
3. Разработка программного обеспечения для сравнения различных программных методов решения транспортных задач с дополнительными условиями.

1 Анализ задач линейного программирования

1.1 Современные аспекты линейного программирования

Линейное программирование — математическая дисциплина, посвящённая теории и методам решения экстремальных задач на множествах *n*-мерного векторного пространства, задаваемых системами линейных уравнений и неравенств [4].

Линейное программирование является частным случаем выпуклого программирования, которое в свою очередь является частным случаем математического программирования. Одновременно оно — основа нескольких методов решения задач целочисленного и нелинейного программирования. Одним из обобщений линейного программирования является дробно-линейное программирование [6].

Многие свойства задач линейного программирования можно интерпретировать также как свойства многогранников и, таким образом, геометрически формулировать и доказывать их.

Общей (стандартной) задачей линейного программирования называется задача нахождения минимума линейной целевой функции (линейной формы) вида (1):

$$f(x) = \sum_{j=1}^{n} c_j x_j = c_1 x_1 + c_2 x_2 + ... + c_n x_n \quad (1)$$

Задача, в которой фигурируют ограничения в форме неравенств, называется основной задачей линейного программирования (ОЗЛП) (2).

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \ (i = 1, 2, ..., m) \\ &x_j \geq 0. \ (j = 1, 2, ..., n) \end{aligned} \quad (2)$$

Задача линейного программирования будет иметь канонический вид, если в основной задаче вместо первой системы неравенств имеет место система уравнений с ограничениями в форме равенства (3):

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, (i = 1, 2, ..., m) \quad (3)$$

Основную задачу можно свести к канонической путём введения дополнительных переменных [7].

Задачи линейного программирования наиболее общего вида (задачи со смешанными ограничениями: равенствами и неравенствами, наличием переменных, свободных от ограничений) могут быть приведены к эквивалентным (имеющим то же множество решений) заменами переменных и заменой равенств на пару неравенств [8].

Легко заметить, что задачу нахождения максимума можно заменить задачей нахождения минимума, взяв коэффициенты с обратным знаком.

Рассмотрим примеры задач линейного программирования.

Рассмотрим задачу о «Максимальном паросочетании» в двудольном графе: есть несколько юношей и девушек, причём для каждых юноши и девушки известно, симпатичны ли они друг другу. Нужно поженить максимальное число пар со взаимной симпатией.

Введём переменные x_{ij}, которые соответствуют паре из *i*-го юноши и *j*-й девушки и удовлетворяют ограничениям (4):

$$\begin{aligned} &0 \le x_{ij} \le 1, \\ &x_{1j} + x_{2j} + ... + x_{nj} \le 1, \\ &x_{i1} + x_{i2} + ... + x_{im} \le 1, \end{aligned} \quad (4)$$

с целевой функцией $f = c_{11}x_{11} + c_{12}x_{12} + c_{nm}x_{nm}$, где c_{ij} равны 1 или 0 в зависимости от того, симпатичны ли i-й юноша и j-я девушка друг другу. Можно показать, что среди оптимальных решений этой задачи найдётся целочисленное. Переменные, равные 1, будут соответствовать парам, которые следует поженить.

Рассмотрим задачу о «Максимальном потоке». Пусть имеется граф (с ориентированными рёбрами), в котором для каждого ребра указана его пропускная способность. И заданы две вершины: сток и исток. Нужно

указать для каждого ребра, сколько через него будет протекать жидкости (не больше его пропускной способности) так, чтобы максимизировать суммарный поток из истока в сток (жидкость не может появляться или исчезать во всех вершинах, кроме истока и стока, соответственно).

Возьмём в качестве переменных x_i – количество жидкости, протекающей через i-е ребро. Тогда можно записать условие (5).

$$0 \le x_i \le c_i, \tag{5}$$

где c_i – пропускная способность i-го ребра. Эти неравенства надо дополнить равенством количества втекающей и вытекающей жидкости для каждой вершины, кроме стока и истока. В качестве функции f естественно взять разность между количеством вытекающей и втекающей жидкости в истоке.

Обобщение предыдущей задачи – максимальный поток минимальной стоимости. В этой задаче даны стоимости для каждого ребра и нужно среди максимальных потоков выбрать поток с минимальной стоимостью. Эта задача сводится к двум задачам линейного программирования: сначала нужно решить задачу о максимальном потоке, а потом добавить к этой задаче ограничение $f(x) \ge m,$ где m – величина максимального потока, и решить задачу с новой функцией $f(x)$ – стоимостью потока.

Рассмотрим задачу «Игра с нулевой суммой». Есть матрица *A* размера $n \times m$. Первый игрок выбирает число от 1 до *n*, второй — от 1 до *m*. Затем они сверяют числа и первый игрок получает a_{ij} очков, а второй $(-a_{ij})$ очков (i – число, выбранное первым игроком, j – вторым). Нужно найти оптимальную стратегию первого игрока.

Пусть в оптимальной стратегии, например, первого игрока число i нужно выбирать с вероятностью p_i. Тогда оптимальная стратегия является решением следующей задачи линейного программирования (6):

$$
\begin{aligned}
&0 \le p_i \le c_i, \\
&p_1 + ... + p_n = 1 \quad , \\
&a_{1i} p_1 + a_{2i} p_2 + ... + a_{ni} p_n \ge c \quad (i = 1,...,m)
\end{aligned}
\tag{6}
$$

в которой нужно максимизировать функцию $f(p_1,..., p_n, c) = c$. Значение c в оптимальном решении будет математическим ожиданием выигрыша первого игрока в наихудшем случае.

1.2 Методы решения задач линейного программирования

Наиболее известным и широко применяемым на практике для решения общей задачи линейного программирования (ЛП) является симплекс-метод. Несмотря на то, что симплекс-метод является достаточно эффективным алгоритмом, показавшим хорошие результаты при решении прикладных задач ЛП, он является алгоритмом с экспоненциальной сложностью. Причина этого состоит в комбинаторном характере симплекс-метода, последовательно перебирающего вершины многогранника допустимых решений при поиске оптимального решения [9, 16].

Первый полиномиальный алгоритм, метод эллипсоидов, был предложен в 1979 году советским математиком Л. Хачияном, разрешившим таким образом проблему, долгое время остававшуюся нерешённой. Метод эллипсоидов имеет совершенно другую, нежели симплекс-метод, некомбинаторную природу. Однако в вычислительном плане этот метод оказался неперспективным. Тем не менее, сам факт полиномиальной сложности задач привёл к созданию целого класса эффективных алгоритмов линейного программирования — методов внутренней точки, первым из которых был алгоритм Н. Кармаркара, предложенный в 1984 году [11, 13]. Алгоритмы этого типа используют непрерывную трактовку задачи линейного программирования, когда вместо перебора вершин многогранника решений задачи линейного программирования осуществляется поиск вдоль

траекторий в пространстве переменных задачи, не проходящих через вершины многогранника. Метод внутренних точек, который, в отличие от симплекс-метода, обходит точки из внутренней части области допустимых значений, использует методы логарифмических барьерных функций нелинейного программирования, разработанные в 1960-х годах Фиако (Fiacco) и МакКормиком (McCormick) [15, 17].

Выводы по главе

В ходе анализа источников литературы проведен обзор аспектов линейного программирования. Описана математическая постановка задач «максимального паросочетания», «максимального потока» и «игры с нулевой суммой». Установлено что при программном решении оптимизационных задач линейного программирования в подавляющем большинстве случаев используется симплекс-метод.

2 Решение задач линейного программирования с использованием Python

2.1 Постановка транспортной задачи

Рассмотрим постановку транспортной задачи.

Имеется некоторое количество m поставщиков, при этом поставщики обозначаются, как $A_1, A_2, ..., A_m$. В наличии поставщиков имеет определенные запасы товара, количество которого можно выразить в виде $a_1, a_2, ..., a_m$. Эти товары нужно переправить n потребителям $B_1, B_2, ..., B_n$, потребность в товаре можно выразить в виде $b_1, b_2, ..., b_n$.

При этом известны текущая стоимость перевозки груза, которая выражена в виде матрицы C, компоненты которой определяют тариф доставки груза от конкретного поставщика A_i к конкретному потребителю B_j.

заказы / запасы		B_1	B_2	...	B_n
		b_1	b_2	...	b_n
A_1	a_1	c_{11}	c_{12}	...	c_{1n}
A_2	a_2	c_{21}	c_{22}	...	c_{2n}
...	...	...	...	...	...
A_m	a_m	c_{m1}	c_{m2}	...	c_{mn}

Рисунок 1 – Общий вид транспортной таблицы

Требуется определить план перевозок, обеспечивающий минимальную суммарную стоимость перевозок.

Условие транспортной можно представить графически в виде следующей схемы (рисунок 1) [15, 17].

Обозначим суммарный запас груза у всех поставщиков, как a, а суммарную потребность в грузе у всех потребителей, как b. Тогда a и b можно выразить следующим образом:

$$\begin{cases} a = \sum_{i=1}^{m} a_i \\ b = \sum_{j=1}^{n} b_j \end{cases} \quad (7)$$

Если выполняется равенство $a = b$, то задача называется закрытой, в противном случае ее называют открытой.

Особенность закрытой задачи заключается в том, что при в конечном итоге у поставщиков не окажется невостребованного товара, при то, что все запросы потребителей в товаре будут выполнены.

В открытой транспортной задаче возможны два варианта

- $a > b$: в этом случае у поставщиков останется невостребованный товар;
- $a < b$: в этом случае не будут удовлетворены все потребности заказчиков.

Обозначим x_{ij} $(x_{ij} > 0)$ количество привозимого товара до потребителя B_j от поставщика A_i. Тогда затраты на доставку можно выразить как функцию *z* в соответствии с выражением 8.

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \quad (8)$$

Матрицу X, компоненты которой равны x_{ij} называют планом перевозок. Составление плана перевозок является задачей оптимизации, в которой *z* играет роль целевой функции.

Математическая формулировка транспортной задачи заключается в нахождении плана перевозок *X*, удовлетворяющего ограничениям (9) и дополнительным условиям.

$$\begin{cases} \sum_{j=1}^{n} x_{ij} = a_i, i = 1,2,...,m \\ \sum_{j=1}^{m} x_{ij} = b_j, \ j = 1,2,...,n \end{cases} \quad (9)$$

Оптимальным является то решение транспортной задачи, которое обеспечивает минимальное значение целевой функции z.

Существуют следующие методы решения транспортной задачи [16, 18]:

1. Метод северно-западного угла. Метод северно-западного угла, заключающийся в заполнении опорного плана транспортной задачи с левой верхней ячейки таблицы, содержащей исходные условия.

2. «Метод минимального элемента. В отличие от метода северно-западного угла, в методе минимального элемента выбор пунктов отправления и пунктов назначения производится, ориентируясь на тарифы перевозок, т.е. в каждом шаге нужно выбрать клетку с минимальным тарифом перевозок. Если таких клеток несколько, то выбираем один из них. Надо отметить, что при данном методе определения заполняемой клетки, стоимость перевозок как правило бывает меньше, чем при методе северно-западного угла» [2].

3. «Метод аппроксимации Фогеля. Суть метода аппроксимации Фогеля заключается в следующем. Для каждой строки и для каждого столбца находим разности между двумя записанными в них минимальными тарифами. Полученные разности записываем в специально отведенные для этого столбце и в строке в таблице условий задачи. Среди указанных разностей выбираем максимальную. В строке (или в столбце), которой данная разность соответствует, определяем минимальный тариф. Клетку, в которой он записан, заполняем на данной итерации. Если минимальный

тариф одинаков для нескольких клеток данной строки (столбца), то для заполнения выбираем ту клетку, которая соответствует наибольшей разности между двумя минимальными тарифами в данном столбце (строке). Применение метода аппроксимации Фогеля позволяет получить либо опорный план, близкий к оптимальному, либо сам оптимальный план» [2].

При использования компьютерных мощностей для решения транспортной задачи, описанные методы практически не используют. В настоящее время затруднительно найти программные библиотеки с реализации данных методов.

2.2 Библиотеки Python для решения задач линейного программирования

Для решения оптимизационных задач линейного программирования на языке Python существует множество библиотек.

Таблица 1 – Библиотеки для решения оптимизационных задач

Библиотека	Лицензия	Метод решения	Ссылка
CVXOPT Python Software for Convex Optimization	free software package	Вариации симплекс-метода	https://cvxopt.org/
SciPy	free software package	Вариации симплекс-метода	https://www.scipy.org/
pulp	free software package	Вариации симплекс-метода	https://pypi.org/project/PuLP/

Наиболее известными, свободно распространяемыми библиотеками с открытым исходным кодом являются cvxopt, scipy и pulp (таблица 1).

Все эти библиотеки позволяют решать оптимизационные задачи линейного программирования с использованием вариаций симплекс-метода.

Симплекс-метод - алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве. Сущность метода: построение базисных решений, на которых монотонно убывает линейный функционал, до ситуации, когда выполняются необходимые условия локальной оптимальности. Задача линейного программирования состоит в том, что необходимо максимизировать или минимизировать некоторый линейный функционал на многомерном пространстве при заданных линейных ограничениях [19, 20].

Заметим, что каждое из линейных неравенств на переменные ограничивает полупространство в соответствующем линейном пространстве. В результате все неравенства ограничивают некоторый выпуклый многогранник (возможно, бесконечный), называемый также полиэдральным комплексом. Уравнение W(x) = c, где W(x) — максимизируемый (или минимизируемый) линейный функционал, порождает гиперплоскость L(c). Зависимость от c порождает семейство параллельных гиперплоскостей. Тогда экстремальная задача приобретает следующую формулировку — требуется найти такое наибольшее c, что гиперплоскость L(c) пересекает многогранник хотя бы в одной точке. Заметим, что пересечение оптимальной гиперплоскости и многогранника будет содержать хотя бы одну вершину, причём их будет более одной, если пересечение содержит ребро или k-мерную грань. Поэтому максимум функционала можно искать в вершинах многогранника. Принцип симплекс-метода состоит в том, что выбирается одна из вершин многогранника, после чего начинается движение по его рёбрам от вершины к вершине в сторону увеличения значения функционала.

Когда переход по ребру из текущей вершины в другую вершину с более высоким значением функционала невозможен, считается, что оптимальное значение с найдено.

Последовательность вычислений симплекс-методом можно разделить на две основные фазы:

- нахождение исходной вершины множества допустимых решений,
- последовательный переход от одной вершины к другой, ведущий к оптимизации значения целевой функции.

При этом в некоторых случаях исходное решение очевидно или его определение не требует сложных вычислений, например, когда все ограничения представлены неравенствами вида «меньше или равно» (тогда нулевой вектор совершенно точно является допустимым решением, хотя и, скорее всего, далеко не самым оптимальным). В таких задачах первую фазу симплекс-метода можно вообще не проводить. Симплекс-метод, соответственно, делится на однофазный и двухфазный.

Выводы по главе

Сформулировано математическое описание транспортной задачи в общем виде. Проанализированы наиболее известные свободно распространяемые библиотеки Python (cvxopt, scipy и pulp), реализующие методы решения транспортных задач. Установлено, что в данных библиотеках решение оптимизационных задач линейного программирования реализовано с использованием симплекс-метода.

3 Разработка программного обеспечения для решения транспортной задачи с дополнительными условиями

3.1 Алгоритм решения транспортной задачи

Решение транспортных задач без дополнительных условий хорошо изучено и обычно не вызывает проблем. Это связано с тем, что для решения данного типа задач разработано большое количество программных средств.

Однако на практике, транспортные задачи обладают дополнительными условиями, которыми нельзя пренебречь. Например, любой транспорт обладает ограниченной грузоподъёмность, кроме-того предельный вес груза для каждого транспортного средства ограничен законодательно. Также необходимо учитывать наличие временных задержек, которые обычно связаны с оформлением и проверкой документов на груз. Также требуется учитывать требования поставщиков и потребителей.

Зададим исходные данные транспортной задачи и представим их в виде таблицы 2.

Таблица 2 – Исходные данные транспортной задачи

		Спрос заказчиков		
		b1=20	b2=45	b3=30
Запасы поставщиков	a1=74	C[0]×X[0]	C[1]×X[1]	C[2]×X[2]
	a2=40	C[3]×X[3]	C[4]×X[4]	C[5]×X[5]
	a3=36	C[6]×X[6]	C[7]×X[7]	C[8]×X[8]

В исходных данных используются следующие обозначения:

- a = (a1; a2; a3) – вектор значений объемов однородных товаров, имеющихся в наличии на складах поставщиков (значения компонентов вектора указаны в таблице);
- b = (b1; b2; b3) – вектор значений спроса объемов однородных товаров заказчиками (значения компонентов вектора указаны в таблице);
- C = (7.0; 3.0; 6.0; 4.0; 8.0; 2.0; 1.0; 5.0; 9.0) – вектор со значениями стоимости перевозки одного экземпляра товара от заказчика потребителю, C[1] – первое значение вектора, C[2] – второе значение и т.д.;
- X – вектор значений объемов перевозимых товаров, обеспечивающих минимальные затраты.

Очевидно, что данная транспортная задача является открытой, поскольку выполняется условие 10

$$\sum_i a_i > \sum_i b_i \tag{10}$$

Запишем условия, связанные запасами поставщиков (11).

$$\begin{aligned} mass_1 &\leftarrow (x[0] + x[1] + x[2] \le 74) \\ mass_2 &\leftarrow (x[3] + x[4] + x[5] \le 40) \\ mass_3 &\leftarrow (x[6] + x[7] + x[8] \le 36) \end{aligned} \tag{11}$$

Запишем условия выполнения потребностей заказчиков (12).

$$\begin{aligned} mass_4 &\leftarrow (x[0] + x[3] + x[6] = 20) \\ mass_5 &\leftarrow (x[1] + x[4] + x[7] = 45) \\ mass_6 &\leftarrow (x[2] + x[5] + x[8] = 30) \end{aligned} \tag{12}$$

Тогда данная транспортная задача без дополнительных условий может быть представлена с учетом значений вектора *C* и условий $mass_1 \ldots mass_6$, как (13).

$$\begin{aligned} problem = op(&C[0]\cdot x[0] + C[1]\cdot x[1] + C[2]\cdot x[2] + C[3]\cdot x[3] \\ &+ C[4]\cdot x[4] + C[5]\cdot x[5] + C[6]\cdot x[6] + C[7]\cdot x[7] \\ &+ C[8]\cdot x[8], [mass1, mass2, mass3, mass4, mass5, \\ &mass6, x\_not\_negative]) \end{aligned}, \tag{13}$$

где $x\_not\_negative$ – условие того, что все значения вектора X являются не отрицательными.

3.2 Формирование дополнительных условий для транспортной задачи

Рассмотрим варианты дополнительных условий, которые возникают при решении транспортных задач на практике.

Первый тип условия – это жесткое условие объема поставки от заданного поставщика заданному заказчику (дополнительное условие №1). Для учета этого условия необходимо вести новую переменную. Если второй заказчик должен получить от первого поставщика товара в количестве 30 штук, то новую переменную следует определить так:

$$mass_7 \leftarrow (x[1] = 30) \tag{14}$$

Второй тип условия – смена требуемого объема поставки для заданного заказчика (дополнительное условие №2). Учета этого условия необходимо изменить значение имеющейся переменной. Так, если объем товара, запрашиваемым вторым заказчиком уменьшился до 30, то переменная $mass_5$ из первоначальной записи (15)

$$mass_5 \leftarrow (x[1] + x[4] + x[7] = 45) \tag{15}$$

измениться на (16).

$$mass_5 \leftarrow (x[1] + x[4] + x[7] = 30) \tag{16}$$

Третий тип условия – изменение границы объема поставок для заданного поставщика (дополнительное условие №3). Такая ситуация возникает, когда требуется обеспечить поставку фиксированного объема, например, вторым поставщиком – 40 единиц товара. В этом случае потребуется переопределить переменную $mass_2$, которая имела первоначальную запись (17)

$$mass_2 \leftarrow (x[3] + x[4] + x[5] \le 40) \tag{17}$$

записав условие так, как показано на (18).

$$mass_2 \leftarrow (x[3] + x[4] + x[5] = 40) \tag{18}$$

Четвертый тип условия – установка паритета для указанных поставщиков (дополнительное условие №4). Например, если требуется уровнять объемы поставок второго и третьего поставщика до уровня 30 единиц товара, то необходимо переопределить переменные $mass_2$ и $mass_3$, которые имели первоначальную запись (19)

$$\begin{aligned} mass_2 &\leftarrow (x[3] + x[4] + x[5] \le 40) \\ mass_3 &\leftarrow (x[6] + x[7] + x[8] \le 36) \end{aligned} \tag{19}$$

записав условия так, как показано на (20).

$$\begin{aligned} mass_2 &\leftarrow (x[3] + x[4] + x[5] = 40) \\ mass_3 &\leftarrow (x[6] + x[7] + x[8] = 36) \end{aligned} \tag{20}$$

Теперь рассмотрим программные методы решения транспортной задачи с четырьмя дополнительными условиями (описанными выше).

3.3 Решение задач с использованием методов библиотеки cvxopt

В официальной документации библиотеки CVXOPT указана возможность решения задач оптимизации с учетом условий и ограничений.

CVXOPT – это бесплатный программный пакет для решения задач выпуклой оптимизации на основе языка программирования Python. Он может использоваться в интерактивном интерпретаторе Python, в командной строке или интегрироваться в другие программы с помощью модулей расширения Python. Его основная цель, как заявлено на официальном сайте библиотек – сделать разработку программного обеспечения для приложений выпуклой оптимизации простой, опираясь на возможности стандартных библиотек Python и с учетом сильных стороны Python как языка программирования высокого уровня.

Разработанный программный код для решения транспортной задачи с дополнительным условием № 1 на основе cvxopt показан на рисунке 2.

```
from cvxopt.modeling import variable, op
import time
start = time.time()
x = variable(9, 'x')
c= [7,3,6,4,8,2,1,5,9]
z=(c[0]*x[0] + c[1]*x[1] + c[2]* x[2] + c[3]*x[3] + c[4]*x[4] + c[5]*x[5]
   + c[6]*x[6] + c[7]*x[7] + c[8]*x[8])
mass1 = (x[0] + x[1] +x[2] <= 74)
mass2 = (x[3] + x[4] +x[5] <= 40)
mass3 = (x[6] + x[7] + x[8] <= 36)
mass4 = (x[0] + x[3] + x[6] == 20)
mass5 = (x[1] +x[4] + x[7] == 45)
mass6 = (x[2] + x[5] + x[8] == 30)
mass7 = (x[1] == 30)
x_non_negative = (x >= 0)
problem =op(z,[mass1,mass2,mass3,mass4 ,mass5,mass6, mass7,x_non_negative])
problem.solve(solver='glpk')
print("Результат Xopt:")
for i in x.value:
        print(i)
print("Стоимость доставки:")
print(problem.objective.value()[0])
stop = time.time()
print ("Время :")
print((stop - start)*1000)
cvxopt_t1 =(stop - start)*1000
```

Рисунок 2 – Программный код для решения транспортной задачи с дополнительным условием №1 с применением метода библиотеки cvxopt

```
Результат Xopt:
0.0
30.0
0.0
0.0
0.0
30.0
20.0
15.0
0.0
Стоимость доставки:
245.0
Время :
7.108926773071289
```

Рисунок 3 – Результат решения транспортной задачи с дополнительным условием № 1 с использованием библиотеки cvxopt

Для того чтобы обеспечить решение транспортной задачи с использованием cvxopt необходимо подключить из раздела modeling методы variable и op. Это осуществляется с помощью инструкции import.

Так как в процессе вычислительных экспериментов планируется производить замер времени решения транспортной задачи, то в коде также производится подключение библиотеки time. Отсчет времени решения начинается до инициализации условий транспортной задачи и запускается с помощью метода time.time(). Начальная временная метка сохраняется в переменную start. После того, как будет найдено решение транспортной задачи, осуществляется повторное сохранение метки в переменную stop. Разница, между значениями, хранящимися в переменных stop и start, будет показывать в миллисекундах затраченное время на инициализацию транспортной задачи и поиск ее решения.

Для инициализации транспортной задачи не обходимо с помощью метода variable задать длину вектора X и, в виде строки, название вектора (в нашем случае в качестве названия используется строка ‘x’). Метод Variable переменную, в которой содержится вектор, значения которого будут подбираться в процессе решения задачи оптимизации. После этого определена переменная C, в которой храниться вектор значений стоимости перевозки (это исходные данные транспортной задачи, представленные в таблице 2) и задана переменная Z, в которой храниться целевая функция задачи оптимизации.

Условия задачи оптимизации задаются через переменные булева типа: mass1, mass2, mass3, mass4, mass5, mass6, x_non_negative.

Объединение целевой функции z с условиями задачи обеспечивается с помощью метода op().

Вывод результатов осуществляется с помощью цикла for в котором итеративно выводится найденные значения вектора X. Также выводится рассчитанная стоимость доставки через problem.objective.value()[0] и затраченное время (3.2).

```
from cvxopt.modeling import variable, op
import time
start = time.time()
x = variable(9, 'x')
c= [7,3,6,4,8,2,1,5,9]
z=(c[0]*x[0] + c[1]*x[1] + c[2]*x[2] +c[3]*x[3] + c[4]*x[4] + c[5]*x[5]
   + c[6]*x[6] + c[7]*x[7] + c[8]*x[8])
mass1 = (x[0] + x[1] +x[2] <= 74)
mass2 = (x[3] + x[4] +x[5] <= 40)
mass3 = (x[6] + x[7] + x[8] <= 36)
mass4 = (x[0] + x[3] + x[6] == 20)
mass5 = (x[1] +x[4] + x[7] == 30)
mass6 = (x[2] + x[5] + x[8] == 30)
x_non_negative = (x >= 0)
problem =op(z,[mass1,mass2,mass3,mass4 ,mass5,mass6,x_non_negative])
problem.solve(solver='glpk')
print("Результат Xopt:")
for i in x.value:
        print(i)
print("Стоимость доставки:")
print(problem.objective.value()[0])
stop = time.time()
print ("Время в мс :")
print((stop - start)*1000)
cvxopt_t2 =(stop - start)*1000
```

Рисунок 4 – Программный код для решения транспортной задачи с дополнительным условием №2 с применением метода библиотеки cvxopt

```
Результат Xopt:
0.0
30.0
0.0
0.0
0.0
30.0
20.0
0.0
0.0
Стоимость доставки:
170.0
Время в мс :
4.576444625854492
```

Рисунок 5 – Результат решения транспортной задачи с дополнительным условием № 2 с использованием библиотеки cvxopt

Для проведения вычислительных экспериментов было написано программное обеспечение, предназначенное для решения транспортной задачи на основе библиотеки cvxopt с исходными данными (таблица 2) со всеми типами дополнительных условий:

- транспортная задача с дополнительным условием №1: программный код использование cvxopt показан на рисунке 2, полученное решение и затраченное время показаны на рисунке 3;
- транспортная задача с дополнительным условием №2: программный код использование cvxopt показан на рисунке 4, полученное решение и затраченное время показаны на рисунке 5;

```
from cvxopt.modeling import variable, op
import time
start = time.time()
x = variable(9, 'x')
c= [7,3,6,4,8,2,1,5,9]
z=(c[0]*x[0] + c[1]*x[1] + c[2]* x[2] + c[3]*x[3] + c[4]*x[4] + c[5]*x[5]
   + c[6]*x[6] + c[7]*x[7] + c[8]*x[8])
mass1 = (x[0] + x[1] +x[2] <= 74)
mass2 = (x[3] + x[4] +x[5] == 40)
mass3 = (x[6] + x[7] + x[8] <= 36)
mass4 = (x[0] + x[3] + x[6] == 20)
mass5 = (x[1] +x[4] + x[7] == 45)
mass6 = (x[2] + x[5] + x[8] == 30)
x_non_negative = (x >= 0)
problem =op(z,[mass1,mass2,mass3,mass4 ,mass5,mass6,x_non_negative])
problem.solve(solver='glpk')
print("Результат Xopt:")
for i in x.value:
        print(i)
print("Стоимость доставки:")
print(problem.objective.value()[0])
stop = time.time()
print ("Время в мс:")
print((stop - start)*1000)
cvxopt_t3 = (stop - start)*1000
```

Рисунок 6 – Программный код для решения транспортной задачи с дополнительным условием №3 с применением метода библиотеки cvxopt

```
Результат Xopt:
0.0
45.0
0.0
10.0
0.0
30.0
10.0
0.0
0.0
Стоимость доставки:
245.0
Время в мс:
7.9936981201171875
```

Рисунок 7 – Результат решения транспортной задачи с дополнительным условием № 3 с использованием библиотеки cvxopt

- транспортная задача с дополнительным условием №3: программный код использование cvxopt показан на рисунке 6, полученное решение и затраченное время показаны на рисунке 7;
- транспортная задача с дополнительным условием №4: программный код использование cvxopt показан на рисунке 8, полученное решение и затраченное время показаны на рисунке 9.

```
from cvxopt.modeling import variable, op
import time
start = time.time()
x = variable(9, 'x')
c= [7,3,6,4,8,2,1,5,9]
z=(c[0]*x[0] + c[1]*x[1] + c[2]*x[2] + c[3]*x[3] + c[4]*x[4] + c[5]*x[5]
   + c[6]*x[6] + c[7]*x[7] +c [8]* x[8])
mass1 = (x[0] + x[1] +x[2] <= 74)
mass2 = (x[3] + x[4] +x[5] == 30)
mass3 = (x[6] + x[7] + x[8] == 30)
mass4 = (x[0] + x[3] + x[6] == 20)
mass5 = (x[1] +x[4] + x[7] == 45)
mass6 = (x[2] + x[5] + x[8] == 30)
x_non_negative = (x >= 0)
problem =op(z,[mass1,mass2,mass3,mass4 ,mass5,mass6,x_non_negative])
problem.solve(solver='glpk')
```

Рисунок 8 – Программный код для решения транспортной задачи с дополнительным условием №4 с применением метода библиотеки cvxopt

```
Результат Xopt:
0.0
35.0
0.0
0.0
0.0
30.0
20.0
10.0
0.0
Стоимость доставки:
235.0
Время в мс:
5.482912063598633
```

Рисунок 9 – Результат решения транспортной задачи с дополнительным условием № 4 с использованием библиотеки cvxopt

Приведенные вычислительные эксперименты показали, что для определенных выше исходных данных транспортной задачи при различных типах дополнительных условий (4 типа) методы библиотеки cvxopt позволяют найти правильное решение задачи за 4.6-8.0 мс. Данное время было получено с использование облачного суперкомпьютера, предоставляемого сервисом Google colab.

Теперь перейдем к рассмотрению возможности решения транспортной задачи с использованием методов библиотеки scipy.optimize.

3.4 Решение задач с использованием методов библиотеки scipy.optimize

SciPy – библиотека для языка программирования Python с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов. В данной библиотеке есть встроенные методы для решения задач оптимизации.

Разработанная реализация для решения транспортной задачи с использованием библиотеки SciPy имеет следующие ключевые моменты.

```
from scipy.optimize import linprog
import time
start = time.time()
c = [7, 3,6,4,8,2,1,5,9]
b_ub = [74,40,36]
A_ub = [[1,1,1,0,0,0,0,0,0],
                [0,0,0,1,1,1,0,0,0],
                [0,0,0,0,0,0,1,1,1]]
b_eq = [20,45,30,30]
A_eq = [[1,0,0,1,0,0,1,0,0],
                [0,1,0,0,1,0,0,1,0],
                [0,0,1,0,0,1,0,0,1],
                [0,1,0,0,0,0,0,0,0]]
print(linprog(c, A_ub, b_ub, A_eq, b_eq))
stop = time.time()
print ("Время в мс:")
print((stop - start)*1000)
scipy_t1 =(stop - start)*1000
```

Рисунок 10 – Программный код для решения транспортной задачи с дополнительным условием №1 с применением метода библиотеки scipy

```
     fun: 245.0
 message: 'Optimization terminated successfully.'
     nit: 10
   slack: array([44., 10.,  1.])
  status: 0
 success: True
       x: array([ 0., 30.,  0.,  0.,  0., 30., 20., 15.,  0.])
Время в мс:
11.538982391357422
```

Рисунок 11 – Результат решения транспортной задачи с дополнительным условием №1 с использованием библиотеки scipy

Для решения транспортной задачи используется метод lingprog(), содержащийся в разделе optimize библиотеки scipy. Подключение данного метода осуществляется инструкцией import.

Также как и в предыдущих реализациях программного обеспечения, здесь реализован подсчет затраченного времени для инициализации условий

задачи и поиска ращений. Отсчет времени с использованием метода time(), реализуемого одноименной библиотекой time (рисунок 10).

Задание исходных данных и условий транспортной задачи здесь осуществляется с помощью числовых векторов и матриц – c, b_ub, A_ub, b_eq, A_eq. Получающийся программный код с использование SciPy обладает меньшей наглядностью по сравнению с кодом, основанным на библиотеке cvxopt.

Пример вывода решения транспортной задачи показан на рисунке 11.

```
from scipy.optimize import linprog
import time
start = time.time()
c = [7, 3,6,4,8,2,1,5,9]
b_ub = [74,40,36]
A_ub = [[1,1,1,0,0,0,0,0,0],
                [0,0,0,1,1,1,0,0,0],
                [0,0,0,0,0,0,1,1,1]]
b_eq = [20,30,30]
A_eq = [[1,0,0,1,0,0,1,0,0],
                [0,1,0,0,1,0,0,1,0],
                [0,0,1,0,0,1,0,0,1]]
print(linprog(c, A_ub, b_ub, A_eq, b_eq))
stop = time.time()
print ("Время в мс:")
print((stop - start)*1000)
scipy_t2 = (stop - start)*1000
```

Рисунок 12 – Программный код для решения транспортной задачи с дополнительным условием №2 с применением метода библиотеки scipy

```
     fun: 170.0
 message: 'Optimization terminated successfully.'
     nit: 7
   slack: array([44., 10., 16.])
  status: 0
 success: True
       x: array([ 0., 30.,  0.,  0.,  0., 30., 20.,  0.,  0.])
Время в мс:
11.217117309570312
```

Рисунок 13 – Результат решения транспортной задачи с дополнительным условием №2 с использованием библиотеки scipy

Для анализа быстродействия библиотеки scipy было разработано программное обеспечение для решения транспортной задачи с исходными данными (таблица 2) с учетом всех типов дополнительных условий:

- транспортная задача с дополнительным условием №1: программный код с использованием scipy показан на рисунке 10, полученное решение и затраченное время показаны на рисунке 11;
- транспортная задача с дополнительным условием №2: программный код с использованием scipy показан на рисунке 12, полученное решение и затраченное время показаны на рисунке 13;
- транспортная задача с дополнительным условием №3: программный код с использованием scipy показан на рисунке 14, полученное решение и затраченное время показаны на рисунке 15;
- транспортная задача с дополнительным условием №4: программный код с использованием scipy показан на рисунке 16, полученное решение и затраченное время показаны на рисунке 17.

```
from scipy.optimize import linprog
import time
start = time.time()
c = [7, 3,6,4,8,2,1,5,9]
b_ub = [74,36]
A_ub = [[1,1,1,0,0,0,0,0,0],
                [0,0,0,0,0,0,1,1,1]]
b_eq = [20,45,30,40]
A_eq = [[1,0,0,1,0,0,1,0,0],
                [0,1,0,0,1,0,0,1,0],
                [0,0,1,0,0,1,0,0,1],
                [0,0,0,1,1,1,0,0,0]]
print(linprog(c, A_ub, b_ub, A_eq, b_eq))
stop = time.time()
print ("Время в мс :")
print((stop - start)*1000)
scipy_t3 = (stop - start)*1000
```

Рисунок 14 – Программный код для решения транспортной задачи с дополнительным условием №3 с применением метода библиотеки scipy

```
     fun: 245.0
 message: 'Optimization terminated successfully.'
     nit: 8
   slack: array([29., 26.])
  status: 0
 success: True
       x: array([ 0., 45.,  0., 10.,  0., 30., 10.,  0.,  0.])
Время в мс :
10.866165161132812
```

Рисунок 15 – Результат решения транспортной задачи с дополнительным условием №3 с использованием библиотеки scipy

```
from scipy.optimize import linprog
import time
start = time.time()
c = [7, 3,6,4,8,2,1,5,9]
b_ub = [74]
A_ub = [[1,1,1,0,0,0,0,0,0]]
b_eq = [20,45,30,30,30]
A_eq = [[1,0,0,1,0,0,1,0,0],
                [0,1,0,0,1,0,0,1,0],
                [0,0,1,0,0,1,0,0,1],
                [0,0,0,1,1,1,0,0,0],
                [0,0,0,0,0,0,1,1,1]]
print(linprog(c, A_ub, b_ub, A_eq, b_eq))
stop = time.time()
```

Рисунок 16 – Программный код для решения транспортной задачи с дополнительным условием №4 с применением метода библиотеки scipy

```
     fun: 235.0
 message: 'Optimization terminated successfully.'
     nit: 8
   slack: array([39.])
  status: 0
 success: True
       x: array([ 0., 35.,  0.,  0.,  0., 30., 20., 10.,  0.])
Время в мс:
9.100675582885742
```

Рисунок 17 – Результат решения транспортной задачи с дополнительным условием №4 с использованием библиотеки scipy

На основе результатов вычислительного эксперимента по использованию библиотеки SciPy установлено, что для указанных заданных исходных данных транспортной задачи при различных типах дополнительных условий (4 типа) методы библиотеки позволяют найти правильное решение задачи за 9 – 11.5 мс. Данное время, также как и в предыдущем случае, было получено с использование облачного суперкомпьютера, предоставляемого сервисом Google colab. Таким образом, временные затраты библиотеки SciPy сопоставимы с временными затратами библиотеки cvxopt.

Теперь перейдем к рассмотрению возможности решения транспортной задачи с использованием методов библиотеки pulp.

3.4 Решение задач с использованием методов библиотеки pulp

PuLp - библиотека на языке Python с открытым исходным кодом, содержащая в себе методы для решения задач линейной оптимизации.

Опишем основные особенности в разработанном программном коде для решения транспортной задачи с использованием библиотеки pulp (рисунок 18).

Для того, чтобы инициализировать условия транспортной задачи, в данном случае нам потребуются все имеющиеся в библиотеке pulp методы. Для подключения всех методов использована команда «from pulp import *».

Так же как и в предыдущем случае, в процессе инициализации условий транспортной задачи и поиска ее оптимального решения будет проводиться замер затраченного времени. По этой причине в программном коде с подключается еще одна библиотека – time.

```
from pulp import *
import time
start = time.time()
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
x3 = pulp.LpVariable("x3", lowBound=0)
x4 = pulp.LpVariable("x4", lowBound=0)
x5 = pulp.LpVariable("x5", lowBound=0)
x6 = pulp.LpVariable("x6", lowBound=0)
x7 = pulp.LpVariable("x7", lowBound=0)
x8 = pulp.LpVariable("x8", lowBound=0)
x9 = pulp.LpVariable("x9", lowBound=0)
problem = pulp.LpProblem('0',pulp.LpMaximize)
problem += -7*x1 - 3*x2 - 6* x3 - 4*x4 - 8*x5 -2* x6-1*x7- 5*x8-9* x9, "Функция цели"
problem +=x2==30
problem +=x1 + x2 +x3<= 74,"1"
problem +=x4 + x5 +x6 <= 40, "2"
problem +=x7 + x8+ x9 <= 36, "3"
problem +=x1+ x4+ x7 == 20, "4"
problem +=x2+x5+ x8 == 45, "5"
problem +=x3 + x6+x9 == 30, "6"
problem.solve()
print ("Результат:")
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)
print ("Стоимость доставки:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Время в мс:")
print((stop - start)*1000)
pulp_t1 = (stop - start)*1000
```

Рисунок 18 – Программный код для решения транспортной задачи с дополнительным условием №1 с применением метода библиотеки pulp

```
Результат:
x1 = 0.0
x2 = 30.0
x3 = 0.0
x4 = 0.0
x5 = 0.0
x6 = 30.0
x7 = 20.0
x8 = 15.0
x9 = 0.0
Стоимость доставки:
245.0
Время в мс:
30.191659927368164
```

Рисунок 19 – Результат решения транспортной задачи с дополнительным условием №1 с использованием библиотеки pulp

С помощью метода time() осуществляется сохранения временной метки в переменную start. После выполнения всех расчетов связанных с взаимодействием с методами библиотеки pulp производится повторный вызов метода time() для определения конечно временной метки, сохраняемой в переменную stop. Впоследствии результат вычитания из значения переменной stop значения переменной start позволяет получить затраченное время на инициализацию условий и решение транспортной задачи.

Для инициализации переменных, отвечающих за объемы перевозок ($x_1 \ldots x_9$) используется метод pulp.LpVariable(). В качестве передаваемых параметров указывается в виде строки имя непеременной и минимально возможное значение (lowBound=0).

Затем инициализируются дополнительные условия и тип решаемой оппозиционной задачи. Так как данная библиотека позволяет решать только задачи максимизации, то целевая функция была инвертирована (умножена на “-1”). Для инициализации типа оптимизационной задачи применяется метод pulp.LpPromlem(). Наличие в качестве параметра pulp.LpMaximize указывает, что тип решаемой задачи – поиск максимума целевой функции.

Целевая функция задается через переменную problem. В этуже переменную с помощью оператора «+=» помещаются и все дополнительные условия транспортной задачи.

Запуск поиска решений осуществляется с помощью метода problem.solve() (рисунок 3.17).

Пример вывода результатов решения транспортной задачи с дополнительным условием №1 показан на рисунке 3.18.

```
from pulp import *
import time
start = time.time()
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
x3 = pulp.LpVariable("x3", lowBound=0)
x4 = pulp.LpVariable("x4", lowBound=0)
x5 = pulp.LpVariable("x5", lowBound=0)
x6 = pulp.LpVariable("x6", lowBound=0)
x7 = pulp.LpVariable("x7", lowBound=0)
x8 = pulp.LpVariable("x8", lowBound=0)
x9 = pulp.LpVariable("x9", lowBound=0)
problem = pulp.LpProblem('0',pulp.LpMaximize)
problem += -7*x1 - 3*x2 - 6* x3 - 4*x4 - 8*x5 -2* x6-1*x7- 5*x8-9* x9, "Функция цели"
problem +=x1 + x2 +x3<= 74,"1"
problem +=x4 + x5 +x6 <= 40, "2"
problem +=x7 + x8+ x9 <= 36, "3"
problem +=x1+ x4+ x7 == 20, "4"
problem +=x2+x5+ x8 == 30, "5"
problem +=x3 + x6+x9 == 30, "6"
problem.solve()
print ("Результат:")
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)
print ("Стоимость доставки:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Время в мс:")
print((stop - start)*1000)
pulp_t2 = (stop - start)*1000
```

Рисунок 20 – Программный код для решения транспортной задачи с дополнительным условием №2 с применением метода библиотеки pulp

```
Результат:
x1 = 0.0
x2 = 30.0
x3 = 0.0
x4 = 0.0
x5 = 0.0
x6 = 30.0
x7 = 20.0
x8 = 0.0
x9 = 0.0
Стоимость доставки:
170.0
Время в мс:
25.464773178100586
```

Рисунок 21 – Результат решения транспортной задачи с дополнительным условием №2 с использованием библиотеки pulp

Для проведения вычислительных экспериментов было написано программное обеспечение, предназначенное для решения транспортной задачи на основе библиотеки cvxopt с исходными данными (таблица 2) со всеми типами дополнительных условий:

- транспортная задача с дополнительным условием №1: программный код использование cvxopt показан на рисунке 18, полученное решение и затраченное время показаны на рисунке 19;
- транспортная задача с дополнительным условием №2: программный код использование cvxopt показан на рисунке 20, полученное решение и затраченное время показаны на рисунке 21;

```
from pulp import *
import time
start = time.time()
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
x3 = pulp.LpVariable("x3", lowBound=0)
x4 = pulp.LpVariable("x4", lowBound=0)
x5 = pulp.LpVariable("x5", lowBound=0)
x6 = pulp.LpVariable("x6", lowBound=0)
x7 = pulp.LpVariable("x7", lowBound=0)
x8 = pulp.LpVariable("x8", lowBound=0)
x9 = pulp.LpVariable("x9", lowBound=0)
problem = pulp.LpProblem('0',pulp.LpMaximize)
problem += -7*x1 - 3*x2 - 6* x3 - 4*x4 - 8*x5 -2* x6-1*x7- 5*x8-9* x9, "Функция цели"
problem +=x1 + x2 +x3<= 74,"1"
problem +=x4 + x5 +x6 == 40, "2"
problem +=x7 + x8+ x9 <= 36, "3"
problem +=x1+ x4+ x7 == 20, "4"
problem +=x2+x5+ x8 == 45, "5"
problem +=x3 + x6+x9 == 30, "6"
problem.solve()
print ("Результат:")
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)
print ("Стоимость доставки:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Время в мс :")
print((stop - start)*1000)
pulp_t3 = (stop - start)*1000
```

Рисунок 22 – Программный код для решения транспортной задачи с дополнительным условием №3 с применением метода библиотеки pulp

```
Результат:
x1 = 0.0
x2 = 45.0
x3 = 0.0
x4 = 10.0
x5 = 0.0
x6 = 30.0
x7 = 10.0
x8 = 0.0
x9 = 0.0
Стоимость доставки:
245.0
Время в мс :
32.88459777832031
```

Рисунок 23 – Результат решения транспортной задачи с дополнительным условием №3 с использованием библиотеки pulp

```
from pulp import *
import time
start = time.time()
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
x3 = pulp.LpVariable("x3", lowBound=0)
x4 = pulp.LpVariable("x4", lowBound=0)
x5 = pulp.LpVariable("x5", lowBound=0)
x6 = pulp.LpVariable("x6", lowBound=0)
x7 = pulp.LpVariable("x7", lowBound=0)
x8 = pulp.LpVariable("x8", lowBound=0)
x9 = pulp.LpVariable("x9", lowBound=0)
problem = pulp.LpProblem('0',pulp.LpMaximize)
problem += -7*x1 - 3*x2 - 6* x3 - 4*x4 - 8*x5 -2* x6-1*x7- 5*x8-9* x9, "функция цели"
problem +=x1 + x2 +x3<= 74,"1"
problem +=x4 + x5 +x6 == 30, "2"
problem +=x7 + x8+ x9 == 30, "3"
problem +=x1+ x4+ x7 == 20, "4"
problem +=x2+x5+ x8 == 45, "5"
problem +=x3 + x6+x9 == 30, "6"
problem.solve()
print ("Результат:")
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)
print ("Стоимость доставки:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Время в мс:")
print((stop - start)*1000)
pulp_t4 = (stop - start)*1000
```

Рисунок 24 – Программный код для решения транспортной задачи с дополнительным условием №4 с применением метода библиотеки pulp

- транспортная задача с дополнительным условием №3: программный код использование cvxopt показан на рисунке 22, полученное решение и затраченное время показаны на рисунке 23;
- транспортная задача с дополнительным условием №4: программный код использование cvxopt показан на рисунке 24, полученное решение и затраченное время показаны на рисунке 25.

```
Результат:
x1 = 0.0
x2 = 35.0
x3 = 0.0
x4 = 0.0
x5 = 0.0
x6 = 30.0
x7 = 20.0
x8 = 10.0
x9 = 0.0
Стоимость доставки:
235.0
Время в мс:
16.132354736328125
```

Рисунок 25 – Результат решения транспортной задачи с дополнительным условием №4 с использованием библиотеки pulp

Приведенные вычислительные эксперименты показали, что для заданых исходных данных транспортной задачи при различных типах дополнительных условий (4 типа) методы библиотеки pulp позволяют найти правильное решение задачи за 16.1 – 30.2 мс. Данное время было получено с использование облачного суперкомпьютера, предоставляемого сервисом Google colab. Методы библиотеки pulp оказались 2.5 – 3 раза медленней методов библиотек cvxopt, scipy.

Перейдем к вопросу визуализации полученных результатов вычислительных экспериментов.

3.5 Визуализация результатов вычислительного эксперимента

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.array([[cvxopt_t1, cvxopt_t2, cvxopt_t3, cvxopt_t4],
                            [scipy_t1, scipy_t2, scipy_t3, scipy_t4],
                            [pulp_t1, pulp_t2, pulp_t3, pulp_t4]]),
                  columns=['Доп.условие №1','Доп.условие №2', 'Доп.условие №3',
                           'Доп.условие №4'],
                  index=['cvxopt','scipy','pulp'])
df
```

	Доп.условие №1	Доп.условие №2	Доп.условие №3	Доп.условие №4
cvxopt	7.108927	4.576445	7.993698	5.482912
scipy	11.538982	11.217117	10.866165	9.100676
pulp	30.191660	25.464773	32.884598	16.132355

Рисунок 26 – Программный код для сбора результатов вычислительного эксперимента в таблицу

```
import matplotlib.pyplot as plt

x = np.array([0,1,2,3])
cvxopt_y = np.array([cvxopt_t1, cvxopt_t2, cvxopt_t3, cvxopt_t4])
scipy_y = np.array([scipy_t1, scipy_t2, scipy_t3, scipy_t4])
pulp_y = np.array([pulp_t1, pulp_t2, pulp_t3, pulp_t4])
my_xticks = ['Доп.условие №1','Доп.условие №2','Доп.условие №3','Доп.условие №4']
plt.xticks(x, my_xticks)
plt.title("Сравнение скорости поиска решения")
plt.xlabel("Номер дополнительного условия")
plt.ylabel("Время поиска решения в мс")
plt.plot(x, cvxopt_y, label = 'cvxopt', color="blue")
plt.plot(x, scipy_y, label = 'scipy', color="red")
plt.plot(x, pulp_y, label = 'pulp', color="green")
plt.legend()
plt.show()
```

Рисунок 27 – Программный код для визуализации результатов вычислительного эксперимента по решению транспортных задач

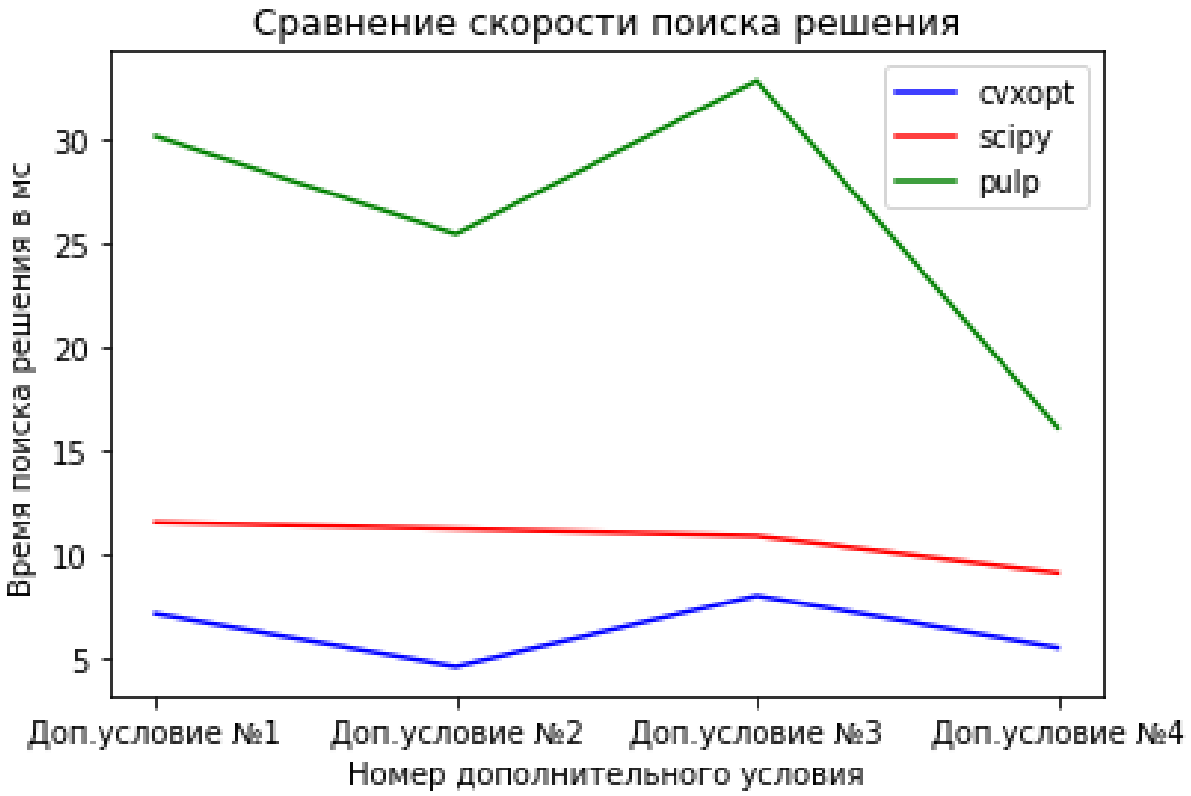

Рисунок 28 – Результат визуализации вычислительного эксперимента, подготовленный с помощью методов библиотеки matplotlib

Таблица 3 – Результаты вычислительных экспериментов по скорости поиска решения транспортной задачи с использованием различных методов

	Время поиска решения задачи в мс		
	Метод библиотеки cvxopt	Метод библиотеки scipy.optimize	Метод библиотеки pulp
Дополнительное условие №1	7.108927	11.538982	30.191660
Дополнительное условие №2	4.576445	11.217117	25.464773
Дополнительное условие №3	7.993698	10.866165	32.884598
Дополнительное условие №4	5.482912	9.100676	16.132355

Выводы по главе

На языке программирования Python разработано программное обеспечение для сравнения методов библиотек cvxopt, scipy, pulp при решении транспортной задачи. Разработанное программное обеспечение производит решение транспортной задачи с заданными пользователем исходными данными, объединяет результаты тестирования методов этих библиотек в таблицу и визуализирует результаты экспериментов в виде графиков.

При тестировании методов библиотек cvxopt, scipy, pulp на задаче с 4 дополнительными условиями установлено, что быстродействие библиотек cvxopt и scipy приблизительно равное, а библиотека pulp затрачивает на решение в 2.5-3 раза больше времени. Однако стоить отметить, что результаты быстродействия во многом зависят от исходных данных, поэтому производить сравнительный анализ методов из данных библиотек необходимо для конкретного экземпляра данных.

Заключение

Отметим основные моменты проведенного исследования:

1. В ходе анализа источников литературы проведен обзор аспектов линейного программирования. Описана математическая постановка задач «максимального паросочетания», «максимального потока» и «игры с нулевой суммой». Установлено что при программном решении оптимизационных задач линейного программирования в подавляющем большинстве случаев используется симплекс-метод.

2. Сформулировано математическое описание транспортной задачи в общем виде. Проанализированы наиболее известные свободно распространяемые библиотеки Python (cvxopt, scipy и pulp), реализующие методы решения транспортных задач. Установлено, что в данных библиотеках решение оптимизационных задач линейного программирования реализовано с использованием симплекс-метода.

3. На языке программирования Python разработано программное обеспечение для сравнения методов библиотек cvxopt, scipy, pulp при решении задач оптимизации. Разработанное программное обеспечение производит решение транспортной задачи с заданными пользователем исходными данными, объединяет результаты тестирования методов этих библиотек в таблицу и визуализирует результаты экспериментов в виде графиков.

4. При тестировании методов библиотек cvxopt, scipy, pulp на задаче с 4 дополнительными условиями установлено, что быстродействие библиотек cvxopt и scipy приблизительно равное, а библиотека pulp затрачивает на решение в 2.5-3 раза больше времени. Однако стоить отметить, что результаты быстродействия во многом зависят от исходных данных, поэтому производить сравнительный анализ методов из данных библиотек необходимо для конкретного экземпляра данных.

Список используемой литературы

1. Александров, А.Э. Стохастическая постановка динамической транспортной задачи с задержками с учетом случайного разброса времени доставки и времени потребления / А.Э. Александров, Н.В. Якушев // Управление большими системами: сборник трудов, 2006. – №12-13. – Институт проблем управления им. В.А. Трапезникова РАН (Москва), 2006 – с. 5-14. – Текст : непосредственный.

2. Алехин, Р.В. Анализ вариаций транспортных задач / Р.В. Алехин, О.В. Косенко // Проблемы автоматизации. Региональное управление. Связь и автоматика - Паруса-2015: сборник трудов IV Всероссийской научной конференции молодых ученых, аспирантов и студентов. 2015. – Издательство: Южный федеральный университет (Ростов-на-Дону), 2015. – с. 159-161. – Текст: непосредственный.

3. Алмазова, Г.М. Транспортная логистика и его задачи / Г.М. Алмазова, Г. Гельдимурадов // Современная наука: проблемы, идеи, тенденции: материалы Международной (заочной) научно-практической конференции. Нефтекамск, 2021. – Издательство: Научно-издательский центр "Мир науки" (ИП Вострецов Александр Ильич) (Нефтекамск), 2021. – с. 11-13. – Текст: непосредственный.

4. Берман, Н.Д. Решение задач оптимизации транспортной логистики в Mathcad Prime 3.1 (на примере транспортной задачи) / Н.Д. Берман, Т.С. Кузьминых // Лучшая научно-исследовательская работа 2017: сборник статей победителей VII Международного научно-практического конкурса. 2017. – Издательство: Наука и Просвещение (Пенза). – с. 12-16. – Текст: непосредственный.

5. Волик В.Г. Обучающий программный комплекс "Транспортная задача" / В.Г. Волик // Перспективные информационные технологии: труды Международной научно-технической конференции. 2016. – Издательство:

Самарский научный центр РАН (Самара), 2016. – с. 728-730. – Текст: непосредственный.

6. Заруднев, Д.И. Задача выбора транспортных средств и ее взаимосвязь с другими задачами транспортной логистики / Д.И. Заруднев, С.М. Мочалин // Формирование транспортно-логистической инфраструктуры. Стратегическое направление повышения конкурентоспособности транспортного комплекса России: Материалы III Международной научно-практической конференции. 2010. – Издательство: ООО "Полиграфический центр КАН" (Омск). – с. 78-82. – Текст: непосредственный.

7. Зедгенизов, А.В. Некоторые аспекты применения геоинформационных систем для задач транспортного планирования / А.В. Зедгенизов // Геология, поиски и разведка полезных ископаемых и методы геологических исследований: Материалы международной научно-технической конференции «Геонауки – 2020». Иркутск, 2020. – Издательство: ИРНИТУ, 2020. – с. 41-45. – Текст: непосредственный.

8. Зуева, Е.А. Транспортная задача с ответственными поставщиками / Е.А. Зуева, В.П. Шаравина // Россия молодая: сборник лучших статей VIII Всероссийской, 61 научно-практической конференции молодых ученых. Кемерово, 2016. – Издательство: Кузбасский государственный технический университет имени Т.Ф. Горбачева (Кемерово), 2016. – с. 39-41. – Текст: непосредственный.

9. Касаткина, Е.В. Математическое моделирование транспортных сетей для решения задачи топливоснабжения региона / Е.В. Касаткина, А.О. Агафонов // Автомобилестроение: проектирование, конструирование, расчет и технологии ремонта и производства: Материалы Всероссийской научно-практической конференции. 2013. – Издательство: Ижевский государственный технический университет имени М.Т. Калашникова (Ижевск), 2013. – с. 6-9. – Текст: непосредственный.

10. Нечитайло, Н.М. Транспортная задача по критерию минимума общего времени с учетом потерь на промежуточную обработку ресурсов / Н.М. Нечитайло // Известия высших учебных заведений. Северо-Кавказский регион. Серия: естественные науки. 2003. – №3 (123). – ФГБОУ ВО Дагестанский государственный технический университет, 2003. – с. 11-14. – Текст: непосредственный.

11. Пилипчук, Л.А. Алгоритм решения производственно-транспортной задачи на обобщенной сети с дополнительными ограничениями / Л.А. Пилипчук // VIII Белорусская математическая конференция: Тезисы докладов международной конференции. 2000. – Издательство: Институт математики НАН Беларуси (Минск), 2000. – с. 82-83. – Текст: непосредственный.

12. Ригерт, Е.М. Решение задач транспортной логистики генетическими алгоритмами / Е.М. Ригерт, В.Н. Зуева // Научный потенциал вуза – производству и образованию: сборник статей по материалам II Международной научно-практической конференции, посвященной 150-летию со дня рождения Б.Л. Розинга. Кубанский государственный технологический университет, Армавирский механико-технологический институт, Кафедра гуманитарных дисциплин. 2020. – Издательство: ООО «Редакция газеты «Армавирский собеседник» (Армавирская типография). – с. 238-242. – Текст: непосредственный.

13. Самородов, А.С. Задача интеллектуальной транспортной системы / А.С. Самородов, Т.В. Мелькумова // Актуальные вопросы совершенствования технической эксплуатации мобильной техники: Материалы Международной научно-практической конференции, посвященной 20-летию кафедры технической эксплуатация транспорта. 2020. – Издательство: Рязанский государственный агротехнологический университет им. П.А. Костычева (Рязань), 2020. – с. 158-162. – Текст: непосредственный.

14. Степанов, В.В. Применение транспортной задачи для оптимизации в межрегиональных сетевых компаниях передачи электроэнергии / В.В. Степанов, М.В. Степанова, Ю.А. Кабанков // VI Международная научно-практическая конференция молодых ученых, посвященная 55-й годовщине полета Ю.А. Гагарина в космос: сборник научных статей. КВВАУЛ им. А.К. Серова. 2016. – Издательство: Общество с ограниченной ответственностью "Издательский Дом - Юг" (Краснодар), 2016. –с. 217-221. – Текст: непосредственный.

15. Тихонова А.Д. Транспортные задачи: понятие, типы, методы решения и практическое применение / Тихонова А.Д. // Российская наука: актуальные исследования и разработки: сборник научных статей IX Всероссийской научно-практической конференции. В 2-х частях. Редколлегия: С.И. Ашмарина, А.В. Павлова (отв. редакторы) [и др.]. 2020. – Издательство: Самарский государственный экономический университет (Самара), 2020. – с.105-109. – Текст: непосредственный.

16. Gao, G. Research on Routing Selection Algorithm Based on Genetic Algorithm / Guohong Gao, Baojian Zhang, Xueyong Li, Jinna Lv // International Conference on Intelligent Computing and Information Science – International Conference, ICICIS 2011, Chongqing, China, January 8-9, 2011. Proceedings, Part II: Intelligent Computing and Information Science. – Springer-Verlag Berlin Heidelberg 2011. – pp. 353-358. – Текст: непосредственный.

17. Karr, Ch. Genetic Algorithm Optimization of a Filament Winding Process / Charles L. Karr, Eric Wilson, Sherri Messimer // Genetic and Evolutionary Computation Conference – Genetic and Evolutionary Computation Conference Chicago, IL, USA, July 12–16, 2003 Proceedings, Part II: Genetic and Evolutionary Computation — GECCO 2003. – Springer-Verlag Berlin Heidelberg 2003. – pp. 2406-2407. – Текст: непосредственный.

18. Karthikeyan, P. Improvement in Genetic Algorithm with Genetic Operator Combination (GOC) and Immigrant Strategies for Multicast Routing in Ad Hoc Networks / P. Karthikeyan, Subramanian Baskar // International

Conference on Swarm, Evolutionary, and Memetic Computing – 4th International Conference, SEMCCO 2013, Chennai, India, December 19-21, 2013, Proceedings, Part I: Swarm, Evolutionary, and Memetic Computing. – Springer International Publishing Switzerland 2013. – pp. 481-491. – Текст: непосредственный.

19. Peng, B. An Adaptive Genetic Simulated Annealing Algorithm for QoS Multicast Routing / Bo Peng, Lei Li // International Conference on Multimedia, Computer Graphics, and Broadcasting – International Conference, MulGraB 2011, Held as Part of the Future Generation Information Technology Conference, FGIT 2011, in Conjunction with GDC 2011, Jeju Island, Korea, December 8-10, 2011. Proceedings, Part II: Multimedia, Computer Graphics and Broadcasting. – Springer-Verlag Berlin Heidelberg 2011. – pp. 338-338. – Текст: непосредственный.

20. Wong, K. A technique for improving the convergence characteristic of genetic algorithms and its application to a genetic-based load flow algorithm / Kit Po Wong, An Li // Asia-Pacific Conference on Simulated Evolution and Learning – First Asia-Pacific Conference, SEAL'96 Taejon, Korea, November 9–12, 1996 Seclected Papers: Simulated Evolution and Learning. – Springer-Verlag Berlin Heidelberg 1997. – pp. 167-176. – Текст: непосредственный.