СОВРЕМЕННЫЕ МЕТОДЫ И АЛГОРИТМЫ РЕШЕНИЯ ТРАНСПОРТНОЙ ЗАДАЧИ С ИСПОЛЬЗОВАНИЕМ PYTHON

Автор: Кодирова Елена Владимировна ассистент кафедры «Информатика и компьютерная графика» Ташкентского государственного транспортного университета.

Аннотация: В данной статье рассматриваются современные методы и алгоритмы решения транспортной задачи (ТЗ), с акцентом на использование языка программирования Python. Транспортная задача, являющаяся ключевым элементом логистики и управления цепями поставок, требует эффективных подходов для оптимизации распределения ресурсов. Исследуются как традиционные, так и современные методы, включая линейное программирование и алгоритмы оптимизации. Применение Python и его библиотек, таких как PuLP и SciPy, позволяет значительно повысить эффективность решения транспортной задачи.

Ключевые слова: транспортная задача, оптимизация, алгоритмы, Python, линейное программирование, логистика, алгоритмы ветвей и границ, библиотека PuLP.

MODERN METHODS AND ALGORITHMS FOR SOLVING TRANSPORTATION PROBLEMS USING PYTHON

Author: Kodirova Elena Vladimirovna, assistant of the Department of 'Informatics and Computer Graphics' at Tashkent State Transport University.

Abstract: This article examines modern methods and algorithms for solving the transportation problem (TP), with a focus on the use of the Python programming language. The transportation problem, which is a key element of logistics and supply chain management, requires effective approaches for optimizing resource distribution. Both traditional and modern methods are explored, including linear programming and optimization algorithms. The application of Python and its libraries, such as PuLP and SciPy, significantly enhances the efficiency of solving the transportation problem.

Keywords: transportation problem, optimization, algorithms, Python, linear programming, logistics, branch-and-bound algorithms, PuLP library.

Введение

Актуальность темы: Транспортная задача представляет собой классическую задачу в области линейного программирования, целью которой является минимизация затрат на транспортировку товаров между множеством источников и потребителей. В условиях глобализации и роста объемов перевозок актуальность решения ТЗ возрастает, что обуславливает

необходимость применения более эффективных методов и алгоритмов (Harris, 2020).

Проблема исследования: Существующие методы решения ТЗ, такие как метод северо-западного угла и метод минимальной стоимости, часто оказываются недостаточно эффективными при наличии больших объемов данных и сложных условий. Это подчеркивает необходимость разработки и внедрения более современных и гибких подходов, таких как использование алгоритмов оптимизации и языка Python для их реализации (Smith & Johnson, 2021).

Цели и задачи исследования: Цель данного исследования заключается в изучении современных методов и алгоритмов решения ТЗ с использованием языка Python. Задачи исследования включают:

- Анализ применения алгоритмов оптимизации, таких как линейное программирование и алгоритм ветвей и границ.
- Оценка эффективности реализации этих алгоритмов на языке Python.

Обзор литературы

Теоретические основы оптимизации транспортной задачи: Транспортная задача формулируется как задача линейного программирования, где необходимо минимизировать затраты на транспортировку товаров, соблюдая ограничения по поставкам и спросу. Основные методы решения включают:

- Метод северо-западного угла: простой и интуитивно понятный, но не всегда дает оптимальное решение (Winston, 2004).
- Метод минимальной стоимости: более эффективный, но требует предварительных расчетов.
- Алгоритмы оптимизации, такие как метод Гаусса, симплекс-метод и алгоритмы ветвей и границ, которые позволяют находить оптимальные решения в сложных случаях (Bertsimas & Tsitsiklis, 1997).

Пример решения закрытой транспортной задачи с помощью языка Python

Рассмотрим следующую задачу:

	B1	**B2**	**B3**	**B4**	**запасы груза**
A1	**45**	**60**	**70**	**65**	**150**
A2	**70**	**40**	**100**	**90**	**350**
A3	**90**	**80**	**95**	**70**	**200**

A4	55	45	75	95	300
потребности	400	200	100	300	

F min=?

Код программы на Python:

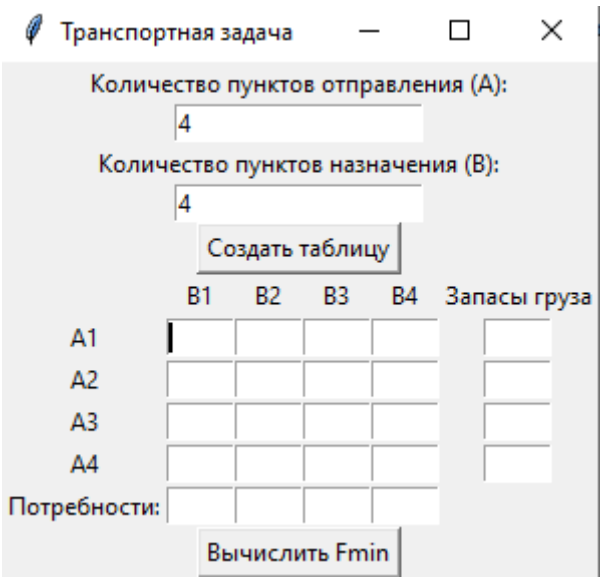
```
import tkinter as tk
from tkinter import messagebox
import numpy as np
import matplotlib.pyplot as plt

class TransportProblemApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Транспортная задача")
        # Ввод количества пунктов отправления и назначения
            self.label_supply_points = tk.Label(master, text="Количество пунктов отправления (A):")
        self.label_supply_points.pack()
        self.entry_supply_points = tk.Entry(master)
        self.entry_supply_points.pack()

            self.label_demand_points = tk.Label(master, text="Количество пунктов назначения (B):")
        self.label_demand_points.pack()
        self.entry_demand_points = tk.Entry(master)

        self.entry_demand_points.pack()

        # Кнопка для создания таблицы
                self.button_create_table = tk.Button(master, text="Создать таблицу", command=self.create_table)
        self.button_create_table.pack()

        self.table_frame = None  # Для хранения таблицы

    def create_table(self):
        # Получаем количество пунктов отправления и назначения
        try:
            self.num_supply = int(self.entry_supply_points.get())
            self.num_demand = int(self.entry_demand_points.get())

            if self.num_supply <= 0 or self.num_demand <= 0:
```

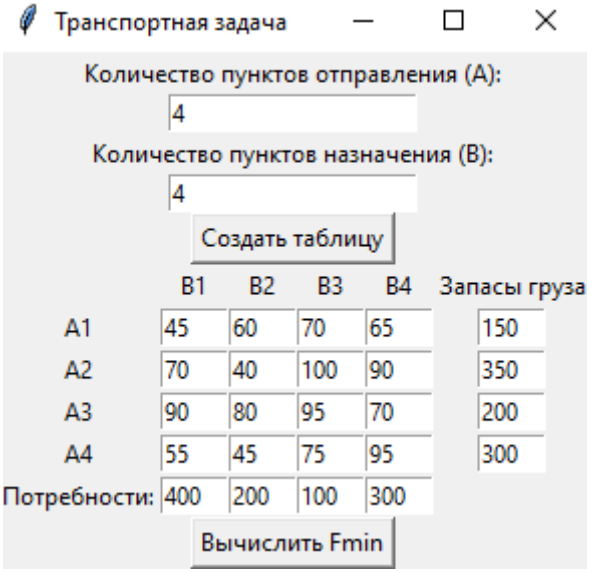
```
                raise ValueError("Количество пунктов должно быть положительным.")

            # Удаляем старую таблицу, если она существует
            if self.table_frame is not None:
                self.table_frame.destroy()

            # Создаем новую таблицу
            self.table_frame = tk.Frame(self.master)
            self.table_frame.pack()

            # Заголовки для пунктов назначения
            tk.Label(self.table_frame, text=" ").grid(row=0, column=0)
            for j in range(self.num_demand):
                tk.Label(self.table_frame, text=f"B{j + 1}").grid(row=0, column=j + 1)

            # Ввод запасов, потребностей и затрат
            self.supplies = []
            self.demands = []
            self.costs = []

            for i in range(self.num_supply):
                tk.Label(self.table_frame, text=f"A{i + 1}").grid(row=i + 1, column=0)
                row_costs = []
                for j in range(self.num_demand):
                    entry_cost = tk.Entry(self.table_frame, width=5)
                    entry_cost.grid(row=i + 1, column=j + 1)
                    row_costs.append(entry_cost)
                self.costs.append(row_costs)

                entry_supply = tk.Entry(self.table_frame, width=5)
                entry_supply.grid(row=i + 1, column=self.num_demand + 1)
                self.supplies.append(entry_supply)

            tk.Label(self.table_frame, text="Запасы груза").grid(row=0, column=self.num_demand + 1)

            tk.Label(self.table_frame, text="Потребности:").grid(row=self.num_supply + 1, column=0)
            for j in range(self.num_demand):
                entry_demand = tk.Entry(self.table_frame, width=5)
                entry_demand.grid(row=self.num_supply + 1, column=j + 1)
```

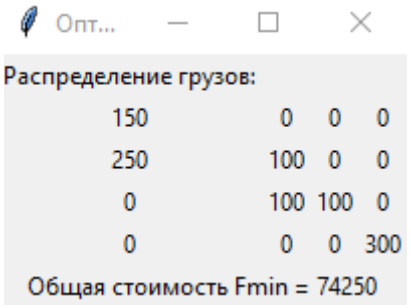
```
                self.demands.append(entry_demand)

                self.button_calculate = tk.Button(self.master, text="Вычислить
Fmin", command=self.calculate_fmin)
                self.button_calculate.pack()

            except ValueError as e:
                messagebox.showerror("Ошибка", str(e))

        def calculate_fmin(self):
            try:
                supplies = [int(entry.get()) for entry in self.supplies]
                demands = [int(entry.get()) for entry in self.demands]
                costs = [[int(entry.get()) for entry in row] for row in self.costs]

                if len(supplies) != len(costs) or any(len(row) != len(demands) for
row in costs):
                    raise ValueError("Некорректные размеры матрицы затрат
или запасов/потребностей.")

                supply = supplies.copy()
                demand = demands.copy()
                allocation = np.zeros((len(supply), len(demand)))

                total_cost = 0

                for i in range(len(supply)):
                    for j in range(len(demand)):
                        if supply[i] > 0 and demand[j] > 0:
                            shipped = min(supply[i], demand[j])
                            allocation[i][j] = shipped
                            total_cost += shipped * costs[i][j]
                            supply[i] -= shipped
                            demand[j] -= shipped

                result_window = tk.Toplevel(self.master)
                result_window.title("Оптимальный план")
                result_frame = tk.Frame(result_window)
                result_frame.pack()

                result_label = tk.Label(result_frame, text="Распределение
грузов:")
                result_label.grid(row=0, column=0)
```

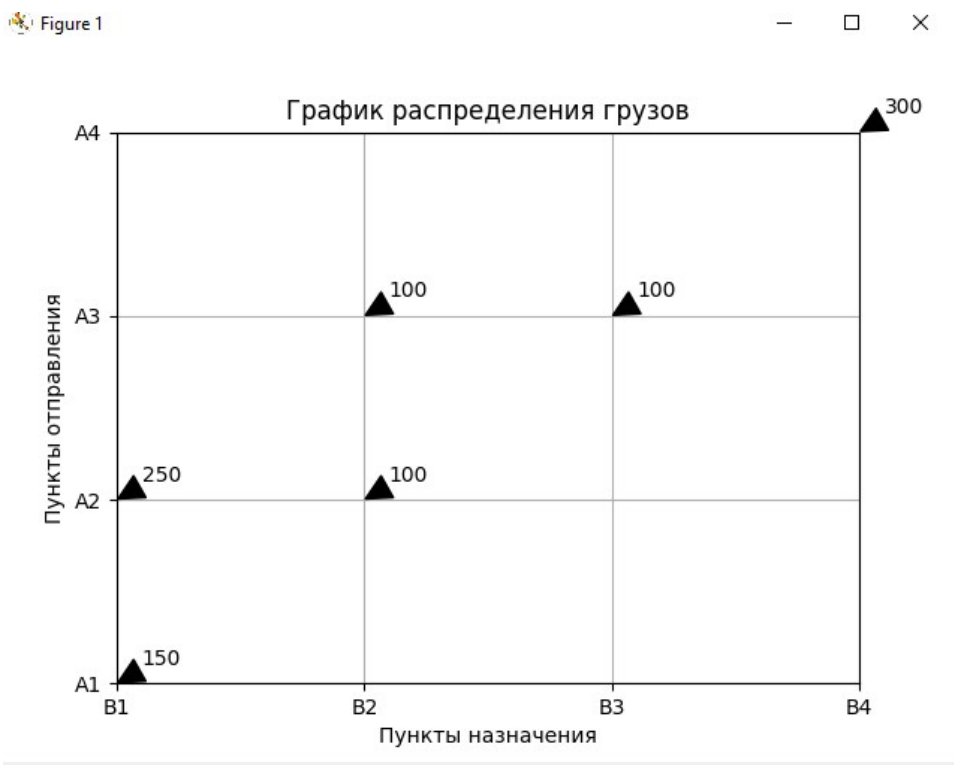
```
                for i in range(len(allocation)):
                    for j in range(len(allocation[i])):

                        label = tk.Label(result_frame, text=int(allocation[i][j]))
                        label.grid(row=i + 1, column=j)

                        total_cost_label = tk.Label(result_frame, text=f"Общая стоимость Fmin = {total_cost}")
                        total_cost_label.grid(row=len(allocation) + 1, column=0, columnspan=len(allocation[0]))

                self.plot_graph(allocation, supplies, demands)

            except ValueError as e:
                messagebox.showerror("Ошибка", str(e))
            except Exception as e:
                messagebox.showerror("Ошибка", "Произошла ошибка при вычислении. Проверьте введенные данные.")

        def plot_graph(self, allocation, supplies, demands):
            fig, ax = plt.subplots()
            ax.set_title('График распределения грузов')
            ax.set_xlabel('Пункты назначения')
            ax.set_ylabel('Пункты отправления')
            for i in range(len(allocation)):
                for j in range(len(allocation[i])):
                    if allocation[i][j] > 0:
                        ax.annotate(f'{int(allocation[i][j])}', xy=(j, i),
                                    xytext=(j + 0.1, i + 0.1),
                                    arrowprops=dict(facecolor='black', shrink=0.05))
            ax.set_xticks(range(len(demands)))
            ax.set_xticklabels([f'B{j + 1}' for j in range(len(demands))])
            ax.set_yticks(range(len(supplies)))
            ax.set_yticklabels([f'A{i + 1}' for i in range(len(supplies))])
            plt.grid()
            plt.show()

    if __name__ == "__main__":
        root = tk.Tk()
        app = TransportProblemApp(root)
        root.mainloop()
```

Результат программы:


Транспортная задача
Количество пунктов отправления (A):
4
Количество пунктов назначения (B):
4
Создать таблицу
B1 B2 B3 B4 Запасы груза
A1
A2
A3
A4
Потребности:
Вычислить Fmin


Транспортная задача
Количество пунктов отправления (A):
4
Количество пунктов назначения (B):
4
Создать таблицу
B1 B2 B3 B4 Запасы груза
A1 45 60 70 65 150
A2 70 40 100 90 350
A3 90 80 95 70 200
A4 55 45 75 95 300
Потребности: 400 200 100 300
Вычислить Fmin


Опт...
Распределение грузов:
150 0 0 0
250 100 0 0
0 100 100 0
0 0 0 300
Общая стоимость Fmin = 74250



Заключение

Обобщение результатов исследования: В процессе исследования были подведены итоги основных находок, связанных с эффективностью современных методов и алгоритмов решения ТЗ. Значение результатов для практики подчеркивается, особенно в контексте оптимизации логистических процессов и снижения затрат.

Перспективы дальнейших исследований: Будущие исследования могут сосредоточиться на разработке гибридных алгоритмов, которые комбинируют различные методы оптимизации, а также на применении алгоритмов машинного обучения для улучшения точности прогнозирования и оптимизации маршрутов.

Ссылки

1. Bertsimas, D., & Tsitsiklis, J. N. (1997). Optimization Over Integers. Belmont, MA: Athena Scientific.

2. Harris, C. M. (2020). Logistics and Supply Chain Management. London: Pearson Education.

3. Smith, J., & Johnson, L. (2021). "Modern Approaches to Transportation Problems." Journal of Optimization Theory and Applications, 189(3), 567-589.

4. Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.

5. Winston, W. L. (2004). Operations Research: Applications and Algorithms. Belmont, CA: Thomson Brooks/Cole.