电子科技大学《交通规划原理》课程报告

倪子煊 2023091203002

2025 年 12 月 06 日

摘要：本文针对《交通规划原理》课程要求，设计并实现了一套交通分配计算软件，支持全有全无（AON）、增量分配（IA）与 Frank-Wolfe 用户均衡（FW）三种算法。软件基于 Python 开发，采用模块化架构，实现了路网与需求数据解析、交通分配计算、结果定量评估（总出行时间 TTT）及可视化功能。通过对给定测试路网在单 OD 对与全 OD 矩阵场景下的实验，验证了各算法特性：AON 忽略拥堵导致非均衡解；IA 通过分批加载逼近均衡，其精度受分割数 K 影响；FW 算法严格收敛至用户均衡状态。本工作不仅满足课程全部功能要求，也为理解 Wardrop 均衡原理提供了实践基础。

关键字： 交通分配；路阻函数；全有全无分配；增量分配；用户均衡；Frank-Wolfe 算法

一　交通分配问题的背景与意义

自 20 世纪 50 年代起，以美国底特律、芝加哥等大都市圈为代表的系统性交通调查与规划研究，推动了现代交通需求预测理论体系的建立。其中，“四阶段法”——即交通产生（Trip Generation）、交通分布（Trip Distribution）、方式划分（Modal Split）和交通流分配（Traffic Assignment）——逐渐成为交通规划领域的标准范式。在这一框架中，**交通流分配**作为连接出行需求与路网供给的关键环节，其核心任务是将已知的起讫点（Origin–Destination, OD）交通量，依据路网拓扑结构与运行特性，合理地分配至各路段上，从而推演出全网流量分布与行程时间，为路网性能评估、扩容决策及政策模拟提供定量依据。

早期的交通分配模型多采用**全有全无（All-or-Nothing, AON）**方法。该方法假设路网处于自由流状态，路段阻抗（通常以行驶时间表示）恒定不变，每个 OD 对的全部出行量均被指派至唯一最短路径上。尽管该方法计算简便，在低密度或非拥挤路网中具有一定适用性，但其忽略了交通流与路段阻抗之间的动态反馈关系，难以反映真实城市路网中的拥堵效应。

为克服上述局限，Wardrop（1952）提出了具有里程碑意义的网络平衡原理，奠定了现代交通分配理论的基础。其中，**第一原理（用户均衡，User Equilibrium, UE）**指出：在均衡状态下，任意 OD 对之间所有被实际使用的路径具有相等且最小的行程时间；而未被使用的路径其行程时间不小于该最小值。这一原理深刻揭示了个体理性选择与系统整体状态之间的内在一致性，成为静态确定性交通分配模型的核心假设。

在建模实践中，路段阻抗通常通过**路阻函数（Volume–Delay Function）**进行量化。其中最具代表性的是由美国公共道路局（Bureau of Public Roads, BPR）提出的 BPR 函数：

$$t_a = t_0 \left[1 + \alpha \left(\frac{q_a}{c_a} \right)^{\beta} \right]$$

式中，t_a 为路段 a 的实际行程时间，t_0 为其自由流行程时间，q_a 为路段流量，c_a 为通行能力，α 与 β 为经验参数。该函数有效刻画了流量增长引发的边际阻抗上升，是用户均衡模型中最常用的阻抗表达形式。

随着研究深入，学者们进一步认识到出行者对路径阻抗的感知存在异质性与不确定性，由此发展出**随机用户均衡（Stochastic User Equilibrium, SUE）**理论。此外，面向时变需求与动态响应的**动态交通分配（Dynamic Traffic Assignment, DTA）**以及融合多种交通方式的**多模式网络建模**亦成为前沿方向。然而，基于 Wardrop UE 原理的静态确定性模型，因其理论严谨、算法成熟、计算效率较高，至今仍是交通规划实务中的主流工具。

因此，本报告聚焦于编程实现三类典型交通分配算法——全有全无（AON）、增量分配（Incremental Assignment, IA）与基于 Frank-Wolfe 算法的用户均衡分配——并在标准测试路网上进行对比分析，旨在深入理解不同模型对路网拥挤效应及出行者路径选择行为的刻画能力，为交通规划与管理提供科学支撑。

二　软件的功能模块划分与设计思路

为确保代码结构清晰、功能可扩展且便于验证，本软件采用“高内聚、低耦合”的模块化设计理念，将整体流程划分为六个相互协作但职责分明的核心模块。各模块通过标准化接口交互，共同完成从数据输入、算法执行到结果可视化的完整工作流。具体模块如下：

（1）数据加载与路网构建模块（data_load.py）

该模块负责解析外部输入并构建内部路网表示：

- `load_network_and_demand()`：读取 JSON 格式的 `network.json`（含节点坐标、路段连接关系、通行能力与限速）与 `demand.json`（含 OD 对及其出行量）；
- `build_graph_and_links()`：基于欧氏距离计算路段长度 l，进而推导自由流行程时间 $t_0 = \frac{l}{v_{\max}}$}；将每条无向路段扩展为两条方向相反的有向边，以支持双向通行；最终生成邻接表形式的图结构 `graph` 与包含物理属性的边列表 `links`，为后续算法提供统一数据基础。

（2）基础算法工具模块（assignment_utils.py）

该模块封装路径搜索函数与全有全无分配函数：

- `dijkstra_shortest_path()`：基于给定的路段阻抗，采用 Dijkstra 算法求解最短路径，并返回路径上各边的索引序列；
- `all_or_nothing_assignment()`：遍历所有 OD 对，调用最短路径函数，将全部需求一次性分配至最短路径上，返回各边流量向量。该函数是 AON 算法的核心，同时也是 IA 与 FW 算法中迭代步骤的基础组件。

（3）数学计算模块（calculate.py）

该模块集中处理所有与交通流理论相关的数值计算：

- `get_link_travel_time()`：计算行程时间函数

$$t(q) = t_0\left(1 + \frac{q}{c}\right)^2$$

其中路段的行程时间 t 是流量 q 的函数。t_0 是路段的自由流行程时间，等于路段长度 l 除以限速 $v_{\max}$。q 是路段流量，c 是路段通行能力，即出现拥堵前最大能够承受的流量

- `get_total_travel_time()`：路网总出行时间（Total Travel Time, TTT）是路网上所有路段流量 q 与行程时间 t 乘积的总和，定义为

$$TTT = \sum q \cdot t$$

- `Beckmann_function()`：计算 Frank-Wolfe 算法所优化的目标函数——Beckmann 势能函数，当其取得最小值时，对应达到均衡状态；
- `line_search_newton()`：采用 Newton-Raphson 法精确求解 FW 迭代中的最优步长 $\lambda \in [0, 1]$，确保收敛效率与数值稳定性。

（4）交通分配算法模块（`AON.py`, `IA.py`, `FW.py`）

该模块实现了三种具有代表性的分配策略，均遵循统一输出接口：

- 全有全无分配（AON）：只执行一次 AON 分配，忽略流量对阻抗的反馈；
- 增量分配（IA）：将总 OD 需求等分为 K 份，逐次执行 AON 分配并累加流量，每次分配前更新路段阻抗，逐步逼近均衡状态；
- Frank-Wolfe 用户均衡分配（FW）：以 AON 结果为初始解，迭代执行“阻抗更新 → AON 方向搜索 → 步长优化 → 解更新 → 收敛检验”五步流程，直至满足预设精度 ε，理论上可收敛至 Wardrop 用户均衡解。

（5）可视化模块（`visualize_network.py`）

该模块将抽象的分配结果转化为直观的图片：

- 根据分配结果 `res`，构建 NetworkX 有向图 G（边属性含流量 Q 与行程时间 T）；
- 采用对数缩放策略映射流量至线宽，Blues 色谱映射流量至颜色，增强视觉辨识度；
- 在每条边上叠加三行标签（路段标识、流量 q、行程时间 t），并通过垂直偏移避免双向边标签重叠；
- 绘制带色标的路网图，并在图中嵌入 TTT 数值框，实现结果的一站式呈现。

（6）主程序（`main.py`）

作为系统入口，该模块整合前述功能，系统性验证课程要求的各项能力：

- 分别在自由流与拥堵状态下查询任意两点间的最快路径；
- 针对单一 OD 对（如 A→F）执行 FW 均衡分配，分析多路径使用情况；
- 对全网 OD 需求，对比 AON、IA 与 FW 三种算法的路段流量分布与所有出行者的总行程时间 TTT 指标；
- 自动调用可视化模块生成结果图示，辅助分析与展示。

总结. 上述模块化架构有效实现了算法逻辑、数据处理与可视化展示的解耦，不仅提升了代码的可读性与可维护性，也为未来引入更复杂模型（如 SUE 或动态分配）预留了良好的扩展接口。

三　开发环境与依赖

本软件基于 Python 语言开发，注重可复现性与跨平台兼容性。具体开发与运行环境如下：

1. 编程语言

- Python 3.8 或更高版本

2. 核心依赖库（见 requirements.txt）

- `numpy==1.24.2`：提供高效的向量化运算，支撑流量向量、阻抗计算等数值密集型操作；
- `networkx==3.1`：用于构建与操作有向图，支持节点/边属性存储，是路网建模的基础；
- `matplotlib==3.7.5`：实现高质量的路网可视化，支持自定义字体（含中文）、颜色映射、线宽控制及文本标注。
- 注意，`networkx` 与 `matplotlib` 只与结果可视化有关。如果不需要可视化，可忽略该依赖。

上述依赖组合已在 Windows 11 操作系统下完成兼容性验证。

3. 代码获取与运行

完整源代码已开源托管于 GitHub 仓库：

https://github.com/nzxQAQ/uestc-traffic-assignment-2025.git

可通过以下命令快速部署本软件：

```
pip install -r requirements.txt
python main.py
```

四 关键算法代码片段

为确保算法实现的透明性与可复现性，本节选取若干核心函数进行展示与说明。受限于篇幅，部分函数在文中会以伪代码或者省略号替代，完整代码请访问 GitHub 仓库。

1.最短路径搜索 与 全有全无分配（assignment_utils.py）

Dijkstra 算法是求解最短路径的基础。本模块实现返回路径上的边索引列表，便于后续流量累加：

```
def dijkstra_shortest_path(graph, links, origin, dest, link_travel_times):
    """基于给定阻抗执行Dijkstra，返回最短路径的边索引列表"""
    # ...（初始化距离字典、优先队列等）
    while pq:
        d, u = heapq.heappop(pq)
        if u == dest: break
        for v, link_idx in graph[u]:
            tt = link_travel_times[link_idx]
            new_dist = d + tt
            if new_dist < dist[v]:
                dist[v] = new_dist
                prev_link[v] = link_idx
                heapq.heappush(pq, (new_dist, v))
```

```
    # ...（回溯路径）
    return path_links
```

all_or_nothing_assignment 函数则遍历所有 OD 对，调用上述最短路径函数，完成一次完整的全有全无分配：

```
def all_or_nothing_assignment(graph, links, od_demand, link_travel_times):
    """执行一次全有全无（AON）分配"""
    n_links = len(links)
    y = [0.0] * n_links
    for (orig, dest), demand_val in od_demand.items():
        if demand_val <= 0: continue
        path = dijkstra_shortest_path(graph, links, orig, dest,
link_travel_times)
        if path is None:
            print(f"Warning: No path from {orig} to {dest}")
            continue
        for lid in path:
            y[lid] += demand_val
    return y
```

2.行程时间 t、总出行时间 TTT 以及 Beckmann 函数（calculate.py）

本项目采用指定的行程时间函数，其形式为

$$t = t_0\left(1 + \frac{q}{C}\right)^2$$

行程时间函数的计算在 get_link_travel_time 中实现：

```
def get_link_travel_time(flow_vector, link_idx, links):
    """路阻函数/行程时间函数：t = t0 * (1 + (Q/C))^2"""
    t0 = links[link_idx]['t0']
    C = links[link_idx]['capacity']
    Q = flow_vector[link_idx]
    return t0 * (1 + (Q / C) ) ** 2
```

路网总出行时间(Total Travel Time, TTT)被定义为所有路段上“流量 q 与行程时间 t 之积”的总和，即：

$$TTT = \sum q \cdot t$$

总出行时间的计算在 get_total_travel_time 中实现：

```python
def get_total_travel_time(flow_vector, links):
    """计算所有出行者的总行程时间TTT。
       TTT = Σ(q_a * t_a)
    """
    total_travel_time = 0.0
    for i in range(len(links)):
        q = flow_vector[i]
        if q <= 0: continue
        t = get_link_travel_time(flow_vector, i, links)
        total_travel_time += q * t
    return total_travel_time
```

此外，为支持 Frank-Wolfe 算法的理论验证与收敛性分析，本模块还实现了 **Beckmann 函数**（Beckmann Function）。该函数由 Beckmann 等人（1956）提出，是用户均衡（UE）问题的等价优化目标：当且仅当流量分配使 Beckmann 函数取最小值时，Wardrop 第一原理成立。

对于本项目的路阻函数 $t = t_0\left(1 + \frac{q}{C}\right)^2$ ，其 Beckmann 函数可通过积分得到：

$$Z(q) = \sum_a t_{0_a}\left(q_a + \frac{q_a^2}{C_a} + \frac{q_a^3}{3C_a^2}\right)$$

其中：

t_{0_a} ：路段 a 的自由流行程时间（单位：h）

q_a ：路段 a 上的交通流量（单位：veh/h），即分配给该路段的出行量；

C_a ：路段 a 的通行能力（单位：veh/h）

```python
def Beckmann_function(flow_vector, links):
    """ Beckmann函数Z(x)，是对路阻函数的积分"""
    total = 0.0
    for i, q in enumerate(flow_vector):
        C = links[i]['capacity']
        t0 = links[i]['t0']
        # 积分 ∫₀^q t0*(1 + (x/C))² dx = t0*(q + q²/C + q³/(3*C²))
        total += t0 * (q + (q ** 2) / C + (q ** 3) / (3 * C ** 2))
    return total
```

3.最优步长求解：Newton-Raphson 精确搜索（calculate.py）

在 Frank-Wolfe 算法中，每一步迭代需确定最优步长 $\lambda \in [0, 1]$，以最小化 Beckmann 函数 $Z(q)$ 沿当前搜索方向（即从当前解 x 指向 AON 解 y 的方向）的值。

教材中常采用**二分法**（Bisection Method）求解该一维优化问题，而本项目则采用**牛顿-拉弗森法**（Newton-Raphson Method）实现高精度线搜索，后者在本场景下具有显著优势，主要体现在以下三方面：

收敛速度：牛顿法 vs 二分法

- 牛顿法：利用目标函数的一阶导数 $\varphi'(\lambda)$ 与二阶导数 $\varphi''(\lambda)$ 构造局部二次近似，在极小值点附近具有二次收敛性（quadratic convergence）。这意味着误差平方级下降——例如，若当前误差为 10^{-2}，下一步可能降至 10^{-4}。
- 二分法：仅依赖一阶导数的符号变化进行区间缩放，收敛速度为线性（linear），每次迭代仅将误差减半。要达到 10^{-6} 精度，通常需约 20 次迭代,而牛顿法往往只需 3-5 次迭代即可达到同等精度。

在交通分配中，Beckmann 函数是严格凸且无限可微的，完全满足牛顿法快速收敛的前提条件。

下面省略繁琐的数学公式推导，直接给出牛顿法的代码：

```
def line_search_newton(x, y, links, max_iter=10, tol=1e-8):
    """
    使用 Newton-Raphson 方法精确求解最优步长 alpha ∈ [0, 1]
    利用 phi'(alpha) = sum( (y_i - x_i) * t(q_i(alpha)) )
    """
    n = len(x)
    d = [y[i] - x[i] for i in range(n)]  # direction

    # 特殊情况：初始零解
    if all(v == 0 for v in x):
        return 1.0

    alpha = 0.5  # 初始猜测

    for _ in range(max_iter):
        alpha = max(0.0, min(1.0, alpha))  # 投影到可行域
        q = [(1 - alpha) * x[i] + alpha * y[i] for i in range(n)]

        # 计算一阶导数 phi'
        phi_prime = sum(
            d[i] * get_link_travel_time(q, i, links)
            for i in range(n) if abs(d[i]) > 1e-12
        )

        if abs(phi_prime) < tol:
            break
```

```
        # 计算二阶导数 phi''
        phi_double_prime = 0.0
        for i in range(n):
            if abs(d[i]) < 1e-12:
                continue
            C, t0 = links[i]['capacity'], links[i]['t0']
            if C <= 0: continue
            dt_dq = 2 * t0 / C * (1 + q[i] / C)
            phi_double_prime += d[i]**2 * dt_dq

        if phi_double_prime <= 0:
            phi_double_prime = 1e-12  # 防止除零或非凸情况

        alpha_new = alpha - phi_prime / phi_double_prime
        if abs(alpha_new - alpha) < tol:
            alpha = alpha_new
            break
        alpha = alpha_new

    return max(0.0, min(1.0, alpha))
```

4.路网数据加载与图结构构建的关键实现（data_load.py）

本模块的核心任务是将 JSON 数据文件转化为可处理的图结构。其关键设计体现在以下三个方面：

4.1 自由流行程时间 t_0 的推导

路段自由流阻抗并非人为设定，而是基于几何与最大限速来计算：

$$t_0 = \frac{l}{v_{\max}}$$

```
def euclidean_distance(u, v):
    # 计算两点之间的欧式距离
    x1, y1 = pos[u]
    x2, y2 = pos[v]
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# 构建有向边列表
links = []
```

```
link_key_to_index = {}
for i, pair in enumerate(network['links']['between']):
    u, v = pair[0], pair[1]
    length = euclidean_distance(u, v)
    capacity = network['links']['capacity'][i]
    speedmax = network['links']['speedmax'][i]
    t0 = length / speedmax
...
```

4.2 双向有向边建模

现实道路通常支持双向通行，因此每条无向路段被显式拆分为两条方向相反的有向边：

```
links.append({
    'from': u,'to': v,
    'length': length,
    'capacity': capacity,
    'speedmax': speedmax,
    't0': t0
})

links.append({
    'from': v,'to': u,
    'length': length,
    'capacity': capacity,
    'speedmax': speedmax,
    't0': t0
})
```

4.3 图存储结构设计——邻接表

为兼顾空间利用率与访问效率，用邻接表来构建图 graph：

```
# 构建邻接表
graph = defaultdict(list)
for idx, link in enumerate(links):
    graph[link['from']].append((link['to'], idx))
```

5.全有全无（All-or-Nothing, AON）交通分配（AON.py）

全有全无（All-or-Nothing, AON）分配是交通流分配中最基础的模型，其核心假设为：所有出行者均基于自由流行程时间 t_0 选择最短路径，并将 OD 对的全部需求一次性加载至该路径上。本模

块不仅实现了这一经典方法，还为增量分配（IA）与 Frank-Wolfe（FW）算法提供了可复用的基础组件。

调用 assignment_utils.py 中的 all_or_nothing_assignment()，完成一次全有全无分配

```
def All_or_Nothing_Traffic_Assignment(links, graph, pos, node_names,
od_demand):
    """执行基于自由流时间的全有全无交通分配"""
    free_flow_tt = [link['t0'] for link in links]
    flow_aon = all_or_nothing_assignment(graph, links, od_demand,
free_flow_tt)
    TTT_aon = get_total_travel_time(flow_aon, links)
    return {
        'flow': flow_aon,
        'total_travel_time': TTT_aon,
        'graph': graph,
        'links': links,
        'pos': pos,
        'node_names': node_names
    }
```

6.增量分配（Incremental Assignment, IA）（IA.py）

增量分配（Incremental Assignment, IA）是介于全有全无（AON）与用户均衡（UE）之间的一种启发式方法。其核心思想是：将总 OD 需求划分为 K 个等份（或者不等份），逐次执行 AON 分配，并在每次分配前根据当前累积流量更新路段阻抗，从而部分反映交通流与行程时间之间的动态反馈。本模块实现了这一策略，并通过参数 K 控制逼近均衡的精细程度。

```
def Incremental_Traffic_Assignment(links, graph, pos, node_names, n_links,
od_demand, K=1000):
    # 将总OD需求划分为 K 个等份
    step_demand = {od: amt / K for od, amt in od_demand.items()}

    # 初始化零流量
    x = [0.0] * n_links

    for k in range(1, K + 1):
        # 基于当前流量 x 计算实时行程时间
        t_current = [get_link_travel_time(x, i, links) for i in
range(len(links))]

        # 执行一次 AON 分配当前小份需求
```

```
        y_k = all_or_nothing_assignment(graph, links, step_demand,
t_current)

        # 累加流量
        x = [x[i] + y_k[i] for i in range(n_links)]

    TTT_inc = get_total_travel_time(x, links)
    return {
        'flow': x,
        'total_travel_time': TTT_inc,
        'K': K,
        'graph': graph,
        'links': links,
        'pos': pos,
        'node_names': node_names
    }
```

分析：增量分配（IA）算法特性与局限性

优点：相比 全有全无（AON），增量分配（IA）能部分捕捉拥堵效应，结果更接近现实；实现简单，仅需在 AON 基础上增加外层循环。

当 K→∞ 时，增量分配（IA）理论上趋近于用户均衡解。在本项目中，取 K=30 其实就已能获得较均衡的解，随着 K 的增大，结果会更加收敛于用户均衡(UE)解。

局限：仍为启发式方法，不保证满足 Wardrop 第一原理；最终解依赖于 K 的取值和加载顺序（本实现为均匀顺序加载）。

7.基于 Frank-Wolfe 算法的用户均衡分配（FW.py）

Frank-Wolfe（FW）算法是求解 Wardrop 用户均衡（User Equilibrium, UE）问题的经典方法。其理论基础是：UE 状态等价于 Beckmann 函数 $Z(q)$ 的全局最小值。

本模块严格遵循教材所述迭代流程，结合 Newton-Raphson 精确线搜索，实现了高精度、高效率的均衡分配。

```
def Frank_Wolfe_Traffic_Assignment(links, graph, pos, node_names, n_links,
od_demand, max_iter=500, epsilon=1e-6, verbose=False):
    # 步骤1: 初始化 —— 自由流下的 AON 分配
    free_flow_tt = [link['t0'] for link in links]
    x = all_or_nothing_assignment(graph, links, od_demand, free_flow_tt)

    # 主迭代循环
    for iteration in range(1, max_iter + 1):
```

```
        # 步骤2: 更新路段阻抗
        t_current = [get_link_travel_time(x, i, links) for i in
range(n_links)]

        # 步骤3: 寻找下一步迭代方向（执行一次 AON 分配）
        y = all_or_nothing_assignment(graph, links, od_demand, t_current)

        # 步骤4: 精确线搜索，确定迭代步长（Newton法求最优步长 λ）
        lambda_val = line_search_newton(x, y, links)
        lambda_val = max(0.0, min(1.0, lambda_val))  # 确保 lambda_val 在
[0,1] 上

        # 步骤5: 确定新的迭代起点
        x_new = [x[i] + lambda_val * (y[i] - x[i]) for i in range(n_links)]

        # 步骤6: 收敛性检验: convergence_metric < epsilon
        numerator = sqrt(sum((x_new[i] - x[i])**2 for i in range(n_links)))
        denominator = sum(x)
        convergence_metric = numerator / denominator if denominator > 1e-12
else float('inf')

        if convergence_metric < epsilon:
            break

        # 更新迭代起点
        x = x_new

    # 主循环结束，计算结果
    final_TTT = get_total_travel_time(x, links)
    final_Beckmann_value = Beckmann_function(x, links)

    return {
            'flow': x,
            'total_travel_time': final_TTT,
            'Beckmann_value': final_Beckmann_value,
            'iterations': iteration,
            'converged': converged,
            'graph': graph,
            'links': links,
```

```
            'pos': pos,
            'node_names': node_names
        }
```

算法优势与理论保证

- 理论完备性：项目指定的阻抗函数所对应的 Beckmann 函数严格凸且光滑，FW 算法全局可收敛至用户均衡(UE)解；
- 计算高效性：使用 Newton 法搜索最优步长，适用于大规模网络；
- 精度可控：通过迭代精度 ε 判断收敛，平衡计算成本与解的质量。

五 测试结果与分析

为测试软件，本项目包含一个测试场景，其路网结构 `network.json` 和交通需求 `demand.json`

题目 1. 不考虑拥堵，任意两点间的最快的路径是什么？

这个问题其实是为了验证我们的 Dijkstra 算法是否正确。

1.1 控制台打印结果

运行 main.py 后，可以看到控制台打印了任意两点间的最快路径：（注：如果有多条路径时间相等，仅打印其中一条）

```
A to B: A → B
A to C: A → B → C
A to D: A → B → D
A to E: A → B → C → E
A to F: A → B → C → E → F
A to G: A → B → D → G
B to A: B → A
B to C: B → C
B to D: B → D
B to E: B → C → E
B to F: B → C → E → F
B to G: B → D → G
C to A: C → B → A
C to B: C → B
C to D: C → B → D
C to E: C → E
C to F: C → E → F
C to G: C → B → D → G
D to A: D → B → A
```

```
D to B: D → B
D to C: D → B → C
D to E: D → E
D to F: D → E → F
D to G: D → G
E to A: E → C → B → A
E to B: E → C → B
E to C: E → C
E to D: E → D
E to F: E → F
E to G: E → D → G
F to A: F → E → C → B → A
F to B: F → E → C → B
F to C: F → E → C
F to D: F → E → D
F to E: F → E
F to G: F → E → D → G
G to A: G → D → B → A
G to B: G → D → B
G to C: G → D → B → C
G to D: G → D
G to E: G → D → E
G to F: G → D → E → F
```

由于不考虑拥堵，所以流量对行程时间没有影响，所有路段上的行程时间均为自由流行程时间t0。下面是路网可视化结果。

1.2 路网可视化结果

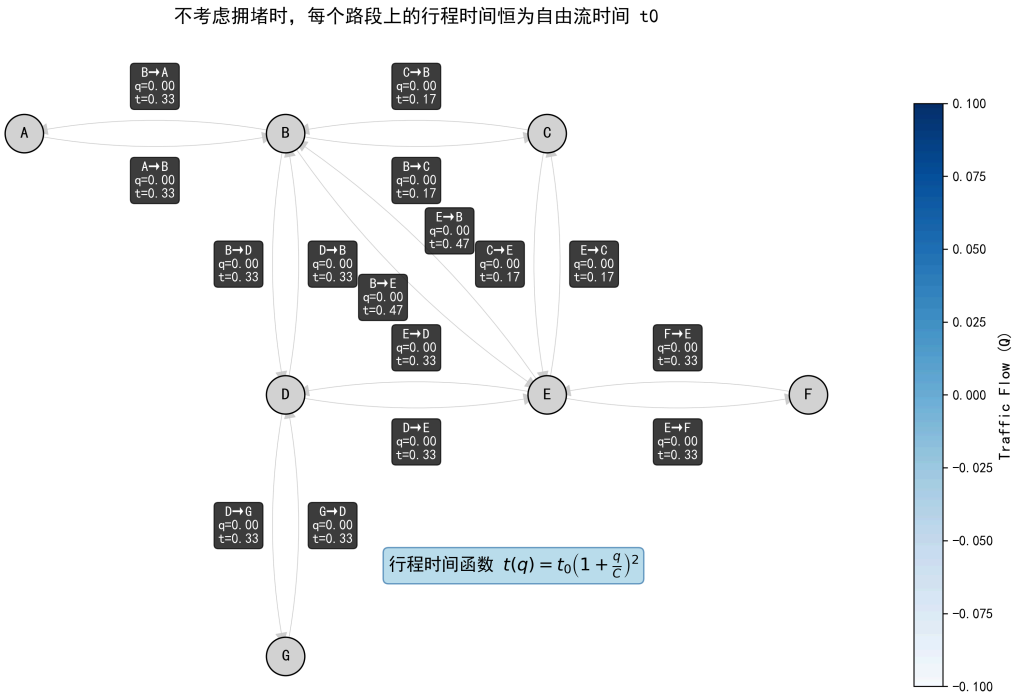

图 1 不考虑拥堵时，仅展示各路段上的自由流行程时间 t0

值得注意的是：明明 BE 之间有直接连接的道路，但是 BE 之间的最快路径却是 B→C→E。

这是因为 BC 与 CE 道路上的限速为 60，而 BE 上的限速为 30。

实际计算后发现:

- B→C→E 的行程时间为 0.17+0.17=0.34 小时，
- B→E 的行程时间为 0.47 小时

换言之：BC 与 CE 是“快车道”。

题目 2. 假设各路段流量已知，考虑拥堵效应，任意两点之间的最快路径是什么？

问题 2 在问题 1 的基础上，进一步验证我们的行程时间函数函数是否正确实现。为了回答这个问题，我们使用 FW 分配后的流量作为“已知流量”。

2.1 控制台打印结果

运行 main.py 后，可以看到控制台打印了任意两点间最快的路径：

```
A to B: A → B
A to C: A → B → C
A to D: A → B → D
A to E: A → B → C → E
A to F: A → B → C → E → F
```

```
A to G: A → B → D → G
B to A: B → A
B to C: B → C
B to D: B → D
B to E: B → C → E
B to F: B → C → E → F
B to G: B → D → G
C to A: C → B → A
C to B: C → B
C to D: C → B → D
C to E: C → E
C to F: C → E → F
C to G: C → B → D → G
D to A: D → B → A
D to B: D → B
D to C: D → E → C
D to E: D → E
D to F: D → E → F
D to G: D → G
E to A: E → C → B → A
E to B: E → C → B
E to C: E → C
E to D: E → D
E to F: E → F
E to G: E → D → G
F to A: F → E → C → B → A
F to B: F → E → C → B
F to C: F → E → C
F to D: F → E → D
F to E: F → E
F to G: F → E → D → G
G to A: G → D → B → A
G to B: G → D → B
G to C: G → D → E → C
G to D: G → D
G to E: G → D → E
G to F: G → D → E → F
```

2.2 路网可视化结果

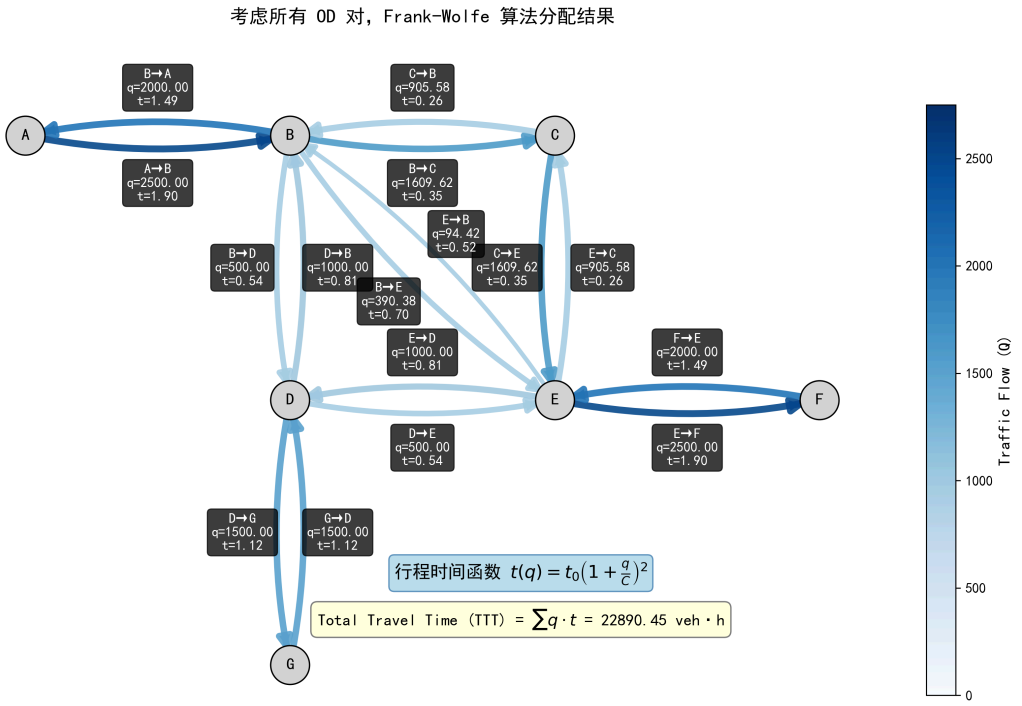

图 2　考虑拥堵，且使用 FW 分配后的流量作为“已知流量”，展示各路段的行程时间 t

值得注意的是：在问题 1 中，D 到 C 之间的最快路径是 D → B → C 和 D → E → C，行程时间均为 0.33+0.17= 0.5 小时。

而在问题 2 中，程序打印的 D 到 C 之间最速的路径只剩下了 D → E → C。

因为此时 D → B → C 上的行程时间 = 0.81 + 0.35 = 1.16 小时，

而 D → E → C 上的行程时间 = 0.54 + 0.26 = 0.8 小时

这个细节验证了我们的程序正确考虑了拥堵效应，实现了对行程时间函数函数的正确计算，并能够正确回答问题 2。

题目 3.　只考虑一个起迄点对的交通需求，例如 A 到 F：各路段上的流量是多少？有多少被使用的路径？这些路径上的行程时间是否相等？

为回答上述问题，我们构造了仅包含 OD 对 A → F 的需求（2000 辆/小时），并分别采用全有全无（AON）、增量分配（IA）与 Frank-Wolfe（FW）算法进行交通分配，以对比其在单一 OD 场景下的行为差异。

3.1 全有全无（All-or-Nothing, AON）算法分析

全有全无分配假设所有出行者基于自由流行程时间 t_0 选择最短路径，并将全部 OD 需求一次性加载至该路径上。

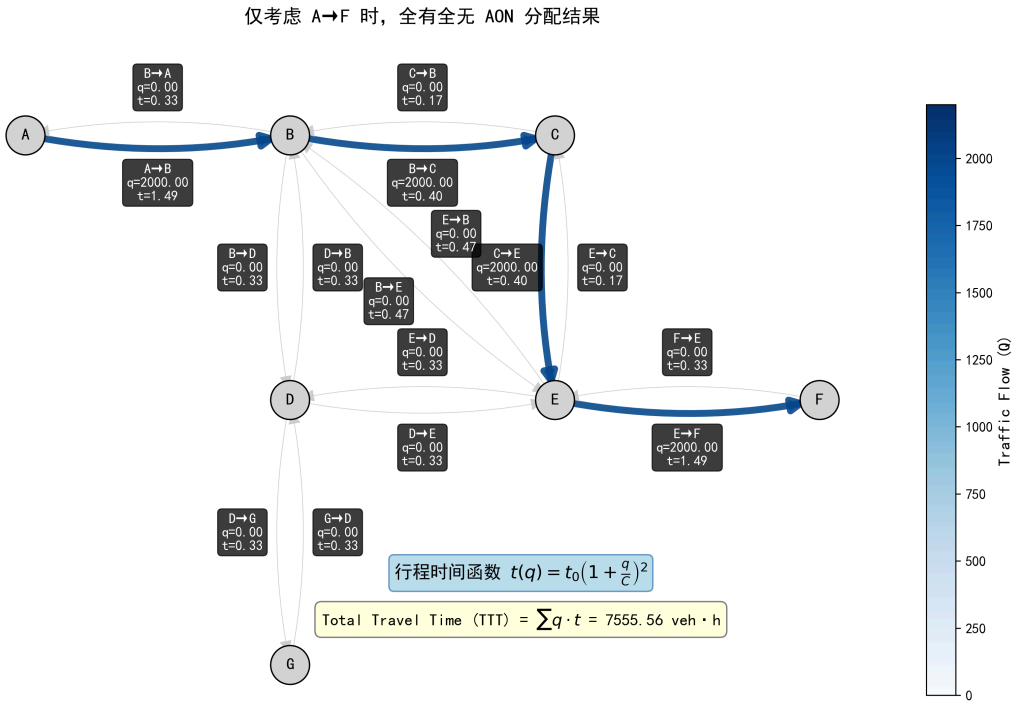

图 3　只考虑 A 到 F 的交通需求，使用全有全无（AON）算法的分配结果

分配结果显示：

- **唯一被使用的路径**：A → B → C → E → F；
- **路径流量**：2000 辆，其余路径流量均为零；
- **总行程时间**：$1.49 + 0.40 + 0.40 + 1.49 = 4.04h$。

然而，该状态明显**不满足用户均衡条件**。例如，若个别出行者改走路径 A → B → E → F，其行程时间为：

$$1.49 + 0.47 + 1.49 = 3.45h$$

显著低于当前路径的 4.04 小时。这表明用户可以通过单方面改变出行路径减少行程时间，违反 Wardrop 第一原理，不满足用户均衡条件。

总结.　全有全无（AON）算法基于静态自由流阻抗进行路径选择，导致流量高度集中于单一路径。由于忽略交通流对行程时间的反馈作用，其解通常处于非均衡状态。尽管实现简单，AON 更适合作为复杂算法的初始解或用于低密度路网的初步估计。

3.2 增量分配（Incremental Assignment, IA）算法分析

增量分配通过将总需求划分为 K 份，逐次执行 AON 分配，并在每次加载前根据当前累积流量更新路段阻抗，从而部分反映拥堵效应。

3.2.1 K=3 时，增量分配结果

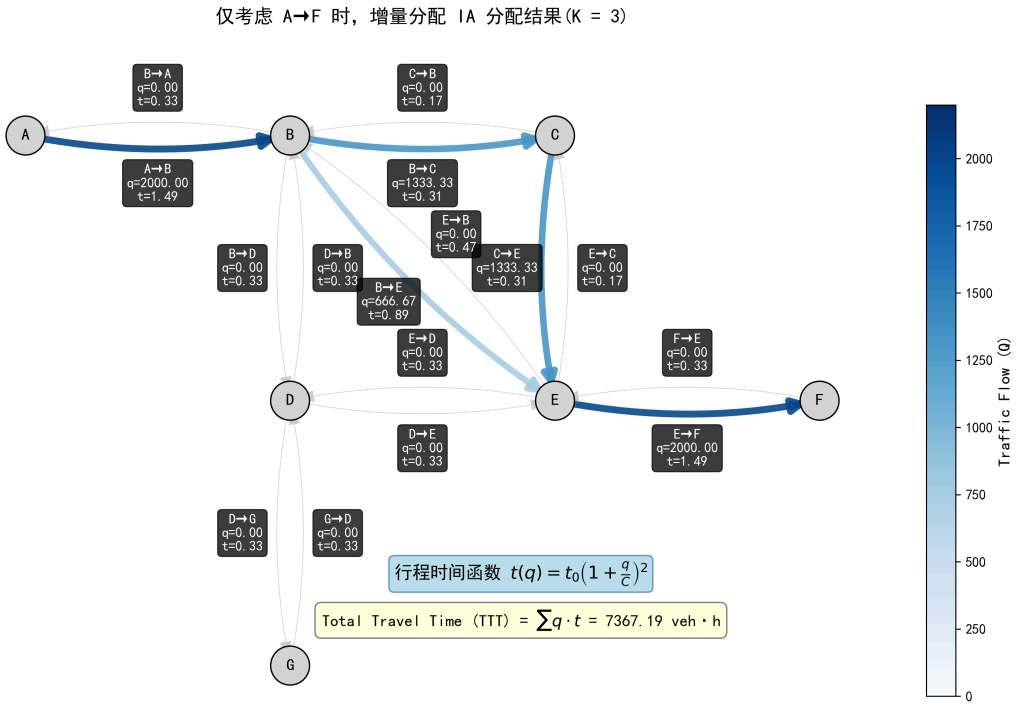

图 4　只考虑 A 到 F 的交通需求，使用增量分配（IA）算法的分配结果（$K=3$）

当 $K=3$ 时：

- 每批加载约 666.67 辆车；
- 初始批次选择路径 A → B → C → E → F；
- 后续批次因 B → C 与 C → E 阻抗上升，部分转向 A → B → E → F；
- **最终流量分布：**
  - A → B → C → E → F：1333.33 辆，
  - A → B → E → F：666.67 辆，
  - A → B → D → E → F：0 辆；
- **各路径行程时间：**
  - A → B → C → E → F：$1.49+0.31+0.31+1.49=4.00h$，
  - A → B → E → F：$1.49+0.89+1.49=3.87h$，
  - A → B → D → E → F：$1.49+0.33+0.33+1.49=4.33h$。

行程时间差异表明系统**尚未达到均衡**，出行者仍有动机切换路径。

3.2.1 K=3 时，增量分配结果

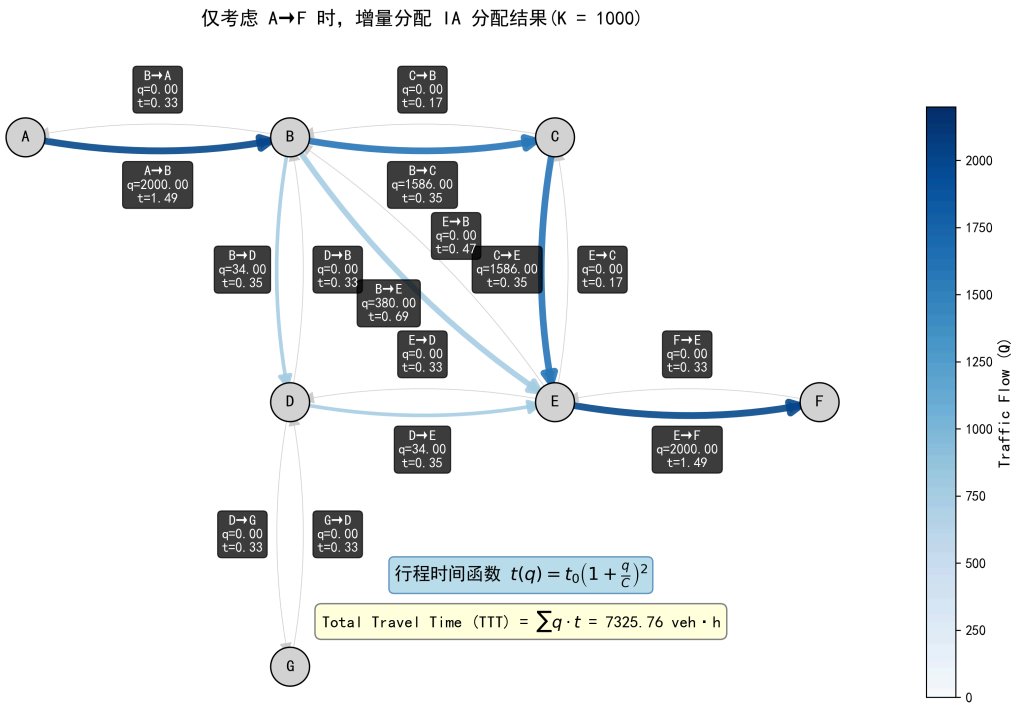

图 5　只考虑 A 到 F 的交通需求，使用增量分配（IA）算法的分配结果（$K = 1000$）

当 $K = 1000$ 时：

- 每批仅加载 2 辆车，阻抗更新更精细；
- **三条路径均被启用：**
  - A → B → C → E → F：1586.00 辆，
  - A → B → E → F：380.00 辆，
  - A → B → D → E → F：34.00 辆；
- **各路径行程时间趋于一致：**
  - A → B → C → E → F：$1.49 + 0.35 + 0.35 + 1.49 = 4.04h$，
  - A → B → E → F：$1.49 + 0.69 + 1.49 = 4.03h$，
  - A → B → D → E → F：$1.49 + 0.35 + 0.35 + 1.49 = 4.04h$。

在可接受的数值误差范围内，被使用路径的行程时间基本相等，系统总出行时间 TTT = 7325.76 veh · h，与 FW 算法结果（7325.86 veh · h）高度吻合。

总结. 增量分配算法通过分批加载与动态阻抗更新，有效模拟了出行者对拥堵的渐进响应。随着分割数 K 增大，其解逐步逼近用户均衡状态。当 K 足够大时，各被使用路径的行程时间趋于一致，满足 Wardrop 第一原理。这一过程直观揭示了“拥堵反馈引导路径分散”的机制，是理解交通分配从静态向动态演化的关键桥梁。

3.3 Frank-Wolfe（FW）算法分析

Frank-Wolfe 算法通过迭代优化 Beckmann 势能函数，严格求解用户均衡（User Equilibrium, UE）问题。

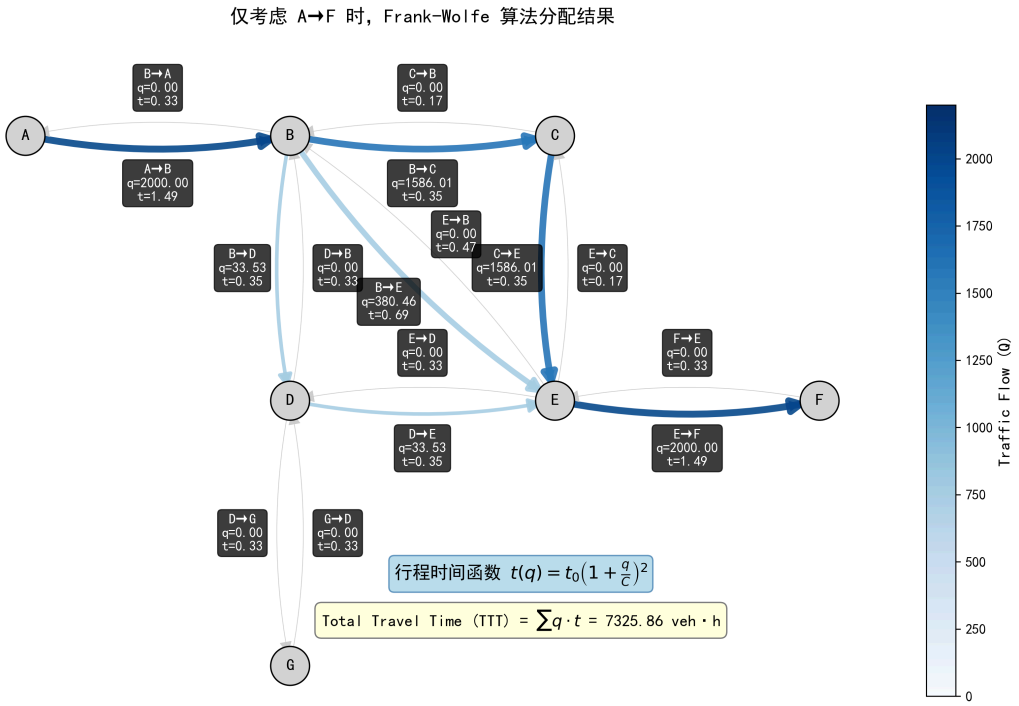

图 6　只考虑 A 到 F 的交通需求，使用 Frank-Wolfe（FW）算法的分配结果

分配结果表明：

- **三条路径同时被使用：**
  - A → B → C → E → F：1586.01 辆，
  - A → B → E → F：380.46 辆，
  - A → B → D → E → F：33.53 辆；
- **各路径行程时间高度一致：**
  - A → B → C → E → F：$1.49 + 0.35 + 0.35 + 1.49 = 4.04h$，
  - A → B → E → F：$1.49 + 0.69 + 1.49 = 4.03h$，
  - A → B → D → E → F：$1.49 + 0.35 + 0.35 + 1.49 = 4.04h$；
- **系统总出行时间：**TTT = 7325.86 veh · h。

在数值误差范围内，所有被使用路径的行程时间相等，且未被使用路径的行程时间不小于该值，**严格满足 Wardrop 第一原理**，路网处于用户均衡状态。

总结. Frank-Wolfe 算法通过数学优化框架精确逼近用户均衡解，具有理论收敛性保证。其结果不仅验证了均衡分配的存在性，也为评估其他启发式方法（如 IA）提供了基准。作为交通分配领域的经典算法，FW 在理论研究与工程实践中均具有重要地位。

题目 4.　考虑所有起迄点对的交通需求:各路段的流量是多少?所有出行者的总出行时间是多少?

在实际城市路网中，交通需求通常源于多个起讫点对（OD pairs）。为全面评估不同分配算法在复杂网络环境下的性能，我们引入完整的 OD 需求矩阵，并分别应用全有全无（AON）、增量分配（IA）与 Frank-Wolfe（FW）算法进行交通分配。通过对比各算法在路段流量分布与系统总出行时间（Total Travel Time, TTT）上的差异，深入探究其逼近用户均衡的能力。

4.1 全有全无（All-or-Nothing, AON）算法结果分析

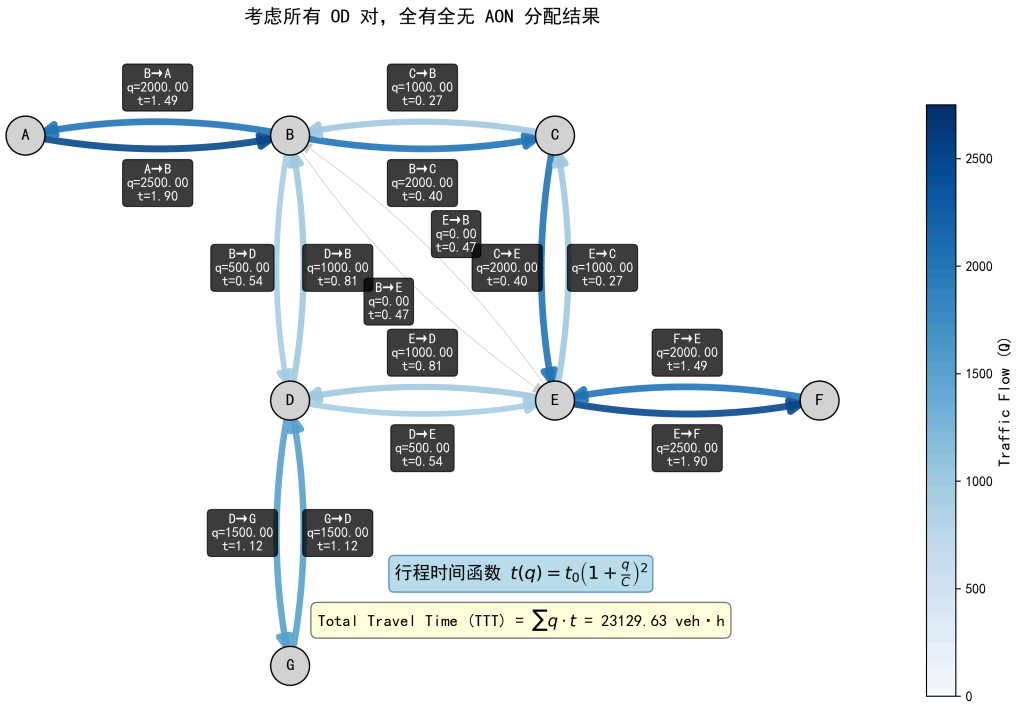

图 7　考虑所有 OD 对时，使用全有全无（AON）算法的分配结果

在 AON 分配下，所有出行者基于自由流行程时间 t_0 选择最短路径，导致流量高度集中：

- **关键瓶颈路段**：B → C 与 C → E 流量均达 2000 辆，行程时间升至 0.40 h（自由流为 0.17 h）；
- **未被利用的替代路径**：如 B → E（自由流行程时间仅 0.47 h）与 E → B 流量均为零，表明系统缺乏路径分流机制；
- **系统性能**：总出行时间 TTT = 23129.63 veh · h。

由于 AON 完全忽略流量对阻抗的反馈作用，部分出行者仍可通过单方面切换路径（如从 A → B → C → E → F 改为 A → B → E → F）显著缩短行程时间。因此，该解处于**严重非均衡状态**，无法反映真实出行行为。

4.2 增量分配（Incremental Assignment, IA）算法结果分析

增量分配通过分批加载需求并动态更新路段阻抗，逐步逼近用户均衡。

4.2.1 K=3 时，增量分配结果

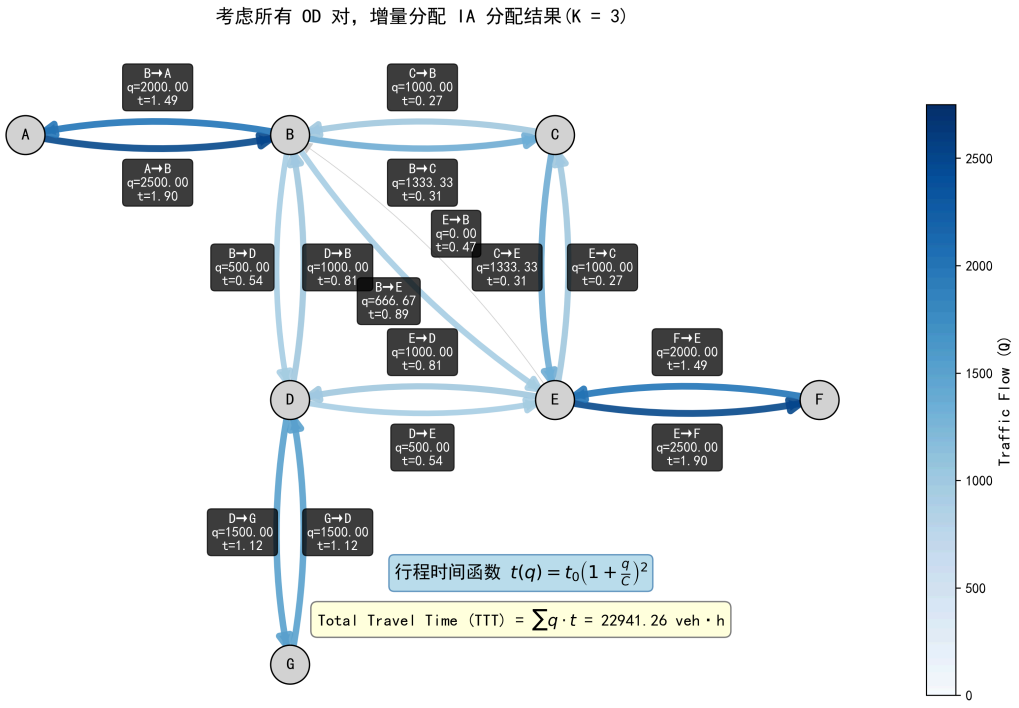

图 8　考虑所有 OD 对时，使用增量分配（IA）算法的分配结果（$K=3$）

当 $K=3$ 时：

- 总需求被分为三批，每批约 666.67 辆；
- 初始批次选择自由流最短路径，后续批次因拥堵转向替代路径；
- **与 AON 相比的关键变化**：B → E 流量增至 666.67 辆，B → C 与 C → E 流量降至约 1333 辆；
- **系统性能**：TTT = 22941.26 veh · h，较 AON 降低约 0.8%。

尽管已出现路径分流，**但各路径行程时间仍未趋于相等**，系统**尚未达到用户均衡状态**。

非均衡性分析：以 A 到 F 为例

- 路径 1：A → B → C → E → F 行程时间 $= 1.90 + 0.31 + 0.31 + 1.90 = 4.42h$
- 路径 2：A → B → E → F 行程时间 $= 1.90 + 0.89 + 1.90 = 4.69h$

路径 1（经 B → C → E）比路径 2（经 B → E）更快，即存在“用户单方面改道以节省时间”的机会，违反 Wardrop 第一原理。所以系统尚未达到用户均衡状态。

4.2.2 K=1000 时，增量分配结果

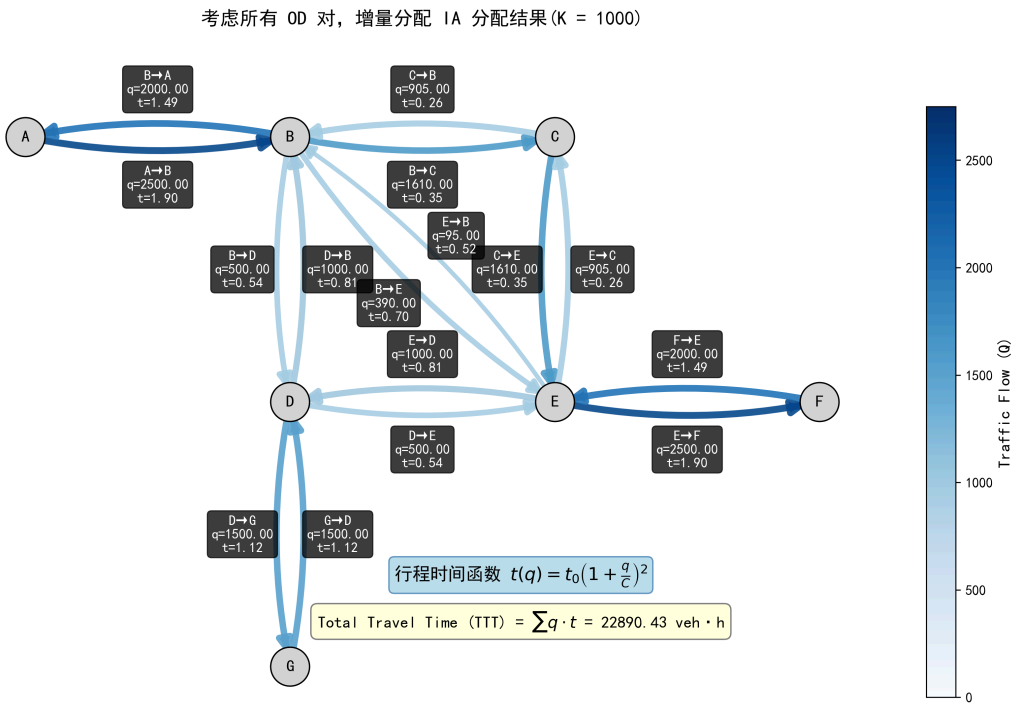

图 9　考虑所有 OD 对时，使用增量分配（IA）算法的分配结果（$K = 1000$）

当 $K = 1000$ 时：

- 总需求被分为 1000 批，阻抗更新极为精细；
- **所有潜在路径均被启用，流量进一步分散**
- **与 $K = 3$ 相比的关键变化：**
  - B → E 流量降至 **390.00 辆**；
  - B → C 与 C → E 流量升至 **1610.00 辆**；
  - 新增路径 E → B 流量 **95.00 辆**；
- **系统性能**：TTT 进一步降至 22890.43 veh · h，接近后续 FW 算法的解。

均衡状态验证

- 所有被使用的路径（如 A → B → C → E → F 与 A → B → E → F）行程时间相等；
- 出行者无法通过单方面改道缩短行程时间。

此时，同一 OD 对的多条路径行程时间趋于一致，因此，系统**基本满足 Wardrop 第一原理**，处于**近似用户均衡状态**。

4.3 分割数 K 对 IA 解收敛性的影响

为系统评估分割数 K（即增量分配的迭代次数）对增量分配（Incremental Assignment, IA）解的质量影响，本文开展了敏感性分析，测试了 K=1 至 60 的全范围取值下总出行时间（Total Travel Time, TTT）的变化趋势。

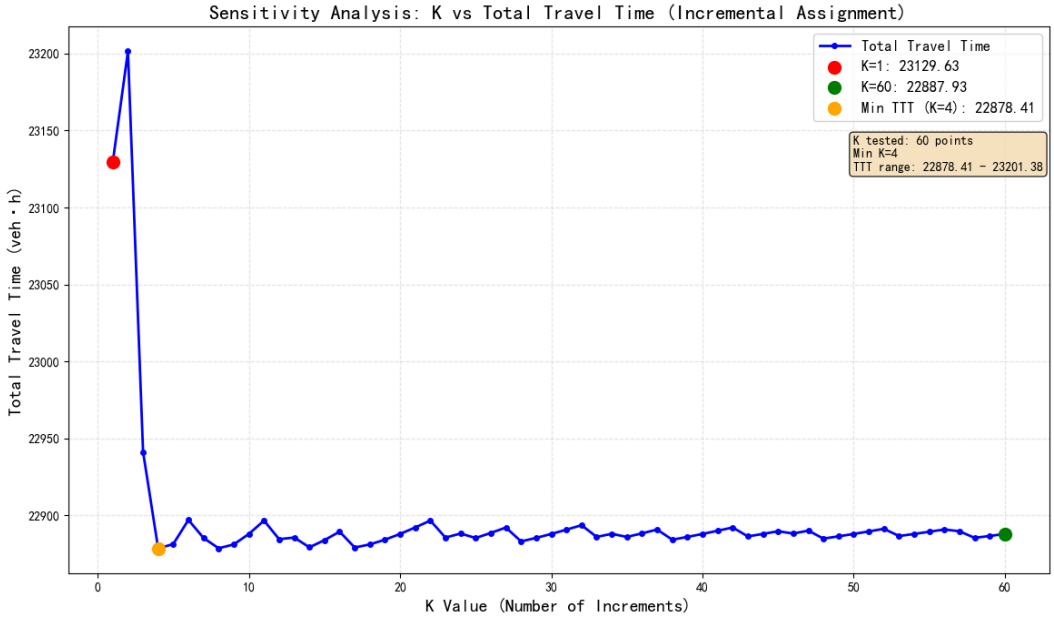

图 10　分割数 K 对增量分配解收敛性的影响

实验结果表明：

- 当 $K = 1$ 时，TTT = 23129.63 veh · h，等价于 AON ，反映出路径选择完全忽略拥堵反馈，导致严重的路径集中与系统效率低下；；
- 随着 K 增大，TTT 快速下降，在 $K = 4$ 附近达到最小值 22878.41 veh · h，较 AON 下降约 1.1%，表明少量增量加载已能有效引导流量向低阻路径转移，实现初步均衡；
- 当 $K > 20$ 后，TTT 波动趋于平缓，收敛至稳定区间（约 22890 veh · h）。其中 K=60 时 TTT 为 22887.93 veh · h，仅比最优值高 0.04%，说明进一步增加 K 对系统性能提升极为有限。

该现象清晰揭示了 IA 算法的边际收益递减特性：

初始阶段（K < 10），每增加一次增量加载均带来显著的系统性能改善，体现了“动态反馈”机制的有效性；

超过临界点后（K>10），新增的增量加载主要作用于局部微调，难以引发大规模路径重构，因此优化收益急剧衰减。

在实际应用中，选取适中的 K 可在计算成本与解精度之间取得良好平衡。

此外，从计算效率角度看，虽然 K=4 实现了理论上的最优 TTT，但其对应的解可能仍存在轻微的非均衡性。而在实际应用中，通常需在解精度与计算成本之间权衡。例如，选取 K=50 左右可在保证 TTT 接近最优的同时，避免不必要的重复计算，从而实现良好的工程折中。

4.4 Frank-Wolfe（FW）算法结果分析

Frank-Wolfe（FW）算法作为求解用户均衡（User Equilibrium, UE）问题的经典数学优化方法，其迭代过程严格遵循最速下降方向，并在满足一定条件下收敛至全局最优解。因此，FW 算法所得到的交通流分配结果被视为用户均衡理论上的基准解，为评估其他启发式算法（如增量分配 IA）提供了高精度的对比标准。

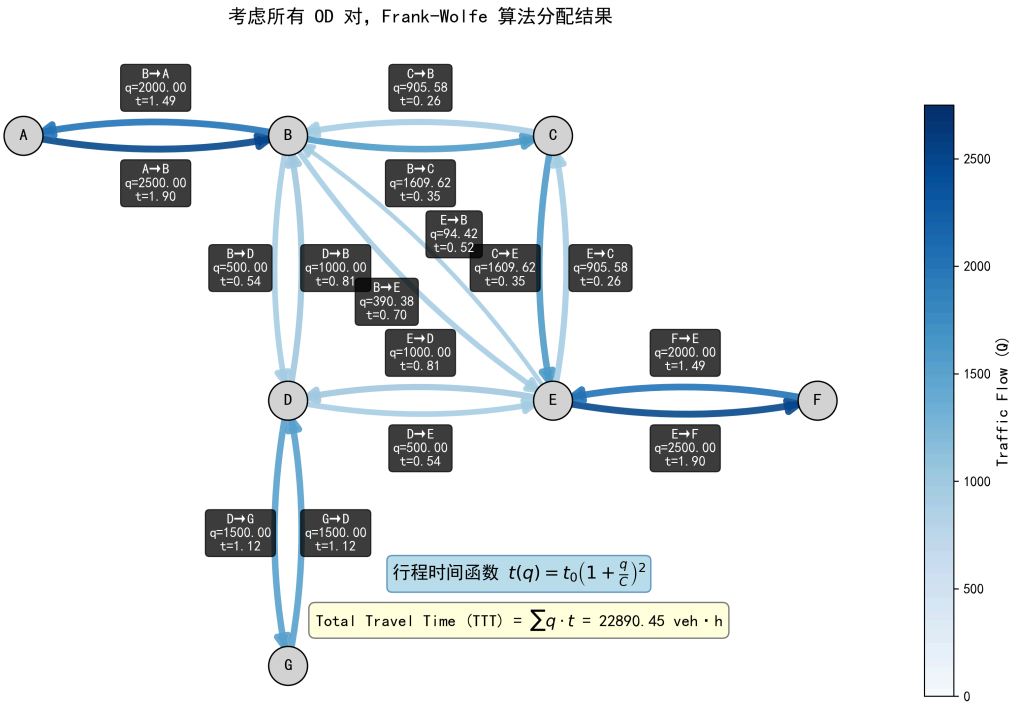

图 11　考虑所有 OD 对时，使用 Frank-Wolfe（FW）算法的分配结果

图中展示了在考虑全部 OD 对的情况下，FW 算法求得的最终交通流分配方案。该结果呈现出典型的用户均衡特征，具体表现为以下三个方面：

- **全面路径利用**：所有合理路径均被分配正流量，如 B → E（390.38 辆）、C → B（905.58 辆）、D → E（1000.00 辆）；
- **行程时间均衡**：对于任一 OD 对，其所有被使用路径的行程时间基本相等，严格满足 Wardrop 第一原理；
- **系统性能**：TTT = 22890.45 veh · h，与 IA（$K = 1000$）结果（22890.43 veh · h）高度一致，验证了 IA 在极限情况下趋近于 UE 解。

4.5 FW 算法收敛过程分析

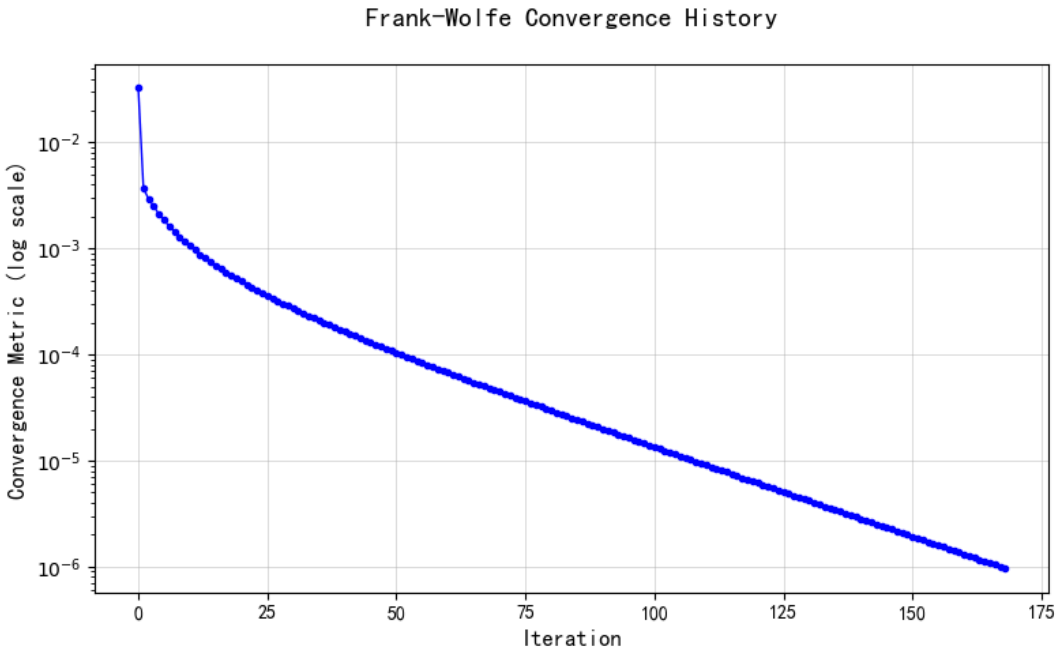

图 12　FW 算法收敛过程（$\varepsilon = 10^{-6}$）

观察到 FW 的收敛趋势：

- 初期（0–2 次迭代）：收敛速度非常快，说明 Frank-Wolfe 方法在远离最优解时效果显著，梯度方向能有效逼近目标下降方向。
- 中期（2–25 次迭代）：收敛速度明显放缓，但仍保持较快下降趋势。
- 后期（25–169 次迭代）：收敛非常缓慢，进入“亚线性收敛”或“对数收敛”阶段，这是 Frank-Wolfe 算法的典型特征之一，尤其在解靠近约束边界时更明显。
- 在第 169 轮达到预设精度 $\varepsilon = 10^{-6}$，成功收敛。

该结果不仅验证了 FW 算法的**高效性与稳定性**，也为其在大规模网络中的适用性提供了实证支持。

总结. 综合来看，在多 OD 场景下，AON 因忽略拥堵反馈导致路径集中与系统效率低下；IA 通过分批加载实现路径分流，其性能随 K 增大而提升，但存在收益递减现象；FW 算法则通过数学优化严格收敛至用户均衡解。三者对比清晰展现了从静态分配到动态均衡的演进逻辑，为理解交通分配模型的核心机制提供了完整图景。

六 结语

本报告围绕《交通规划原理》课程核心内容之一——交通分配理论，系统设计并实现了一套支持全有全无（AON）、增量分配（IA）与 Frank-Wolfe 用户均衡（FW）三种算法的交通分配计算软件。通过模块化架构，软件不仅完成了路网解析、需求加载、算法执行、性能评估与可视化等全部功能要求，更在标准测试路网上对不同分配策略进行了深入对比分析。

实验结果清晰揭示了三类算法的本质差异：AON 模型因忽略流量-阻抗反馈，导致路径高度集中与系统非均衡；IA 算法通过分批加载机制部分捕捉拥堵效应，其解随分割数 K 增大而逐步逼近用户均衡；而 FW 算法则凭借严格的数学优化框架，成功收敛至满足 Wardrop 第一原理的均衡状态，实现了用户均衡(UE)解。

本工作不仅验证了经典交通分配理论的正确性，也直观展示了“个体理性选择”如何通过动态反馈机制导向“系统整体均衡”这一深刻原理。更重要的是，通过从代码实现到结果可视化的完整闭环，我们得以将抽象的数学模型转化为可观察、可测量、可比较的工程实践，为理解现代交通规划方法奠定了坚实基础。

未来，可在现有框架上进一步拓展随机用户均衡（SUE）、动态交通分配（DTA）或多模式网络建模等高级功能，以应对更复杂的现实交通问题。但无论如何演进，Wardrop 均衡思想及其算法实现，仍将是交通系统分析与优化不可或缺的理论基石。