%% 数据读取与预处理

trafficData = readtable('traffic\_data.csv');

demandData = readtable('demand\_data.csv');

%% 网络构建与需求分配

G = graph(trafficData.StartNode, trafficData.EndNode);

ODMatrix = zeros(numnodes(G), numnodes(G));

for i = 1:height(demandData)

ODMatrix(demandData.StartNode(i), demandData.EndNode(i)) = demandData.Demand(i);

end

%% 参数调整与优化策略模拟

% 定义几种优化策略

strategies = {'IncreaseCapacity', 'DecreaseDemand'};

% 策略参数范围

paramRange = 0.8:0.1:1.2;

% 初始化结果存储结构

resultsTable = table();

% 结果图初始化

resultsFig = figure;

hold on; % 允许在同一图表上绘制多个数据集

% 遍历每种策略和参数

for strategy = strategies

totalTravelTimes = zeros(size(paramRange)); % 存储每个策略的总旅行时间

for paramIndex = 1:length(paramRange)

param = paramRange(paramIndex);

% 应用策略

[adjustedG, adjustedODMatrix] = applyStrategy(G, ODMatrix, strategy{1}, param);

% 交通流分配 - 用户均衡（UE）

[flow, travelTime] = userEquilibrium(adjustedG, adjustedODMatrix);

% 结果分析

totalTravelTime = sum(travelTime);

totalTravelTimes(paramIndex) = totalTravelTime;

% 将结果添加到表格中

newRow = table(strategy, param, totalTravelTime, 'VariableNames', {'Strategy', 'Parameter', 'TotalTravelTime'});

resultsTable = [resultsTable; newRow];

% 显示每个策略和参数组合的结果

fprintf('策略: %s, 参数: %.1f, 总旅行时间: %.2f 分钟\n', strategy{1}, param, totalTravelTime);

end

% 绘制当前策略下不同参数对总旅行时间的影响

plot(paramRange, totalTravelTimes, 'DisplayName', strategy{1}); % 保证strategy是cell形式正确解引用

end

% 设置图形属性

xlabel('参数值');

ylabel('总旅行时间 (分钟)');

title('不同策略和参数下的总旅行时间');

legend();

hold off;

% 显示结果表格

disp(resultsTable);

%% 函数定义

function [adjustedG, adjustedODMatrix] = applyStrategy(G, ODMatrix, strategy, param)

adjustedG = G;

adjustedODMatrix = ODMatrix;

switch strategy

case 'IncreaseCapacity'

% 增加道路容量

adjustedG.Edges.Weight = G.Edges.Weight \* param;

case 'DecreaseDemand'

% 减少需求

adjustedODMatrix = ODMatrix \* param;

end

end

function [flow, travelTime] = userEquilibrium(G, ODMatrix)

flow = zeros(size(G.Edges, 1), 1);

for origin = 1:size(ODMatrix, 1)

for destination = 1:size(ODMatrix, 2)

if ODMatrix(origin, destination) > 0

[path, ~] = shortestpath(G, origin, destination);

for p = 1:(length(path)-1)

edgeIndex = findedge(G, path(p), path(p+1));

flow(edgeIndex) = flow(edgeIndex) + ODMatrix(origin, destination);

end

end

end

end

travelTime = flow / 100; % 假设旅行时间与流量成正比

end