

# スタートアップゼミ発表資料

## 【交通量配分課題】

03-250032 学部4年

富矢 航大

# 目次

【0】使用したデータについて

モデルの概要

【1】松山市の渋滞の現状の把握

【2】松山市におけるBraessのパラドックス

【3】中ノ川通りの東ゆきを閉鎖して計算してみると

【4】JR松山駅高架化工事が道路渋滞に与える影響

【5】考察

## 【0】使用したデータについて

- ・松山市中心部の交通ネットワークデータ

（ノード数857、リンク数2831）

- ・2007年松山都市圏パーソントリップ調査（50102トリップ）

本検証ではこのうち朝8時台出発するもののみ抽出して使用

# 【0】モデルの概要

道路データ・PTデータで利用者均衡配分

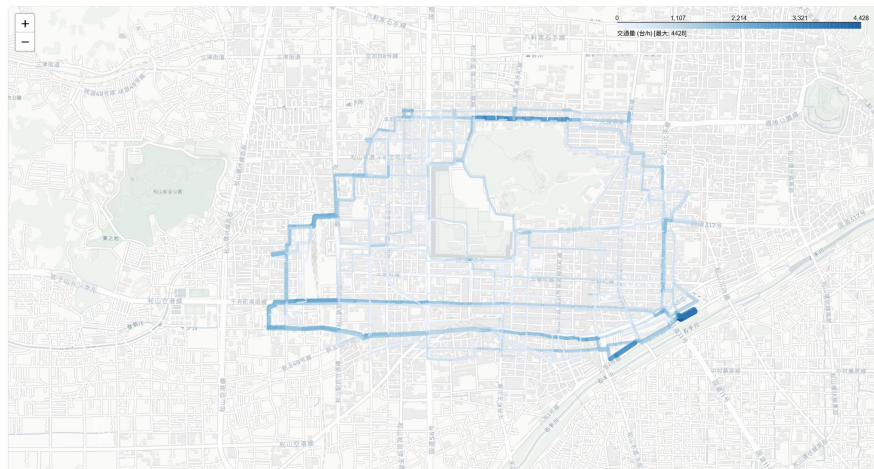
(計算の設定)

- ・自由流旅行時間(交通量0のときの旅行時間)  
: 時速30km/hで走ったときの所要時間
- ・交通容量  
: 車線数×1800台
- ・ $\alpha=0.15$   $\beta=4$

$$t_a(x_a) = \underbrace{t_{a0}}_{\substack{\text{自由流旅行時間} \\ (x_a=0\text{のときの} \\ \text{時間})}} \left\{ 1 + \alpha \left( \frac{x_a}{C_a} \right)^\beta \right\}$$

$\underbrace{C_a}_{\text{交通容量}}$

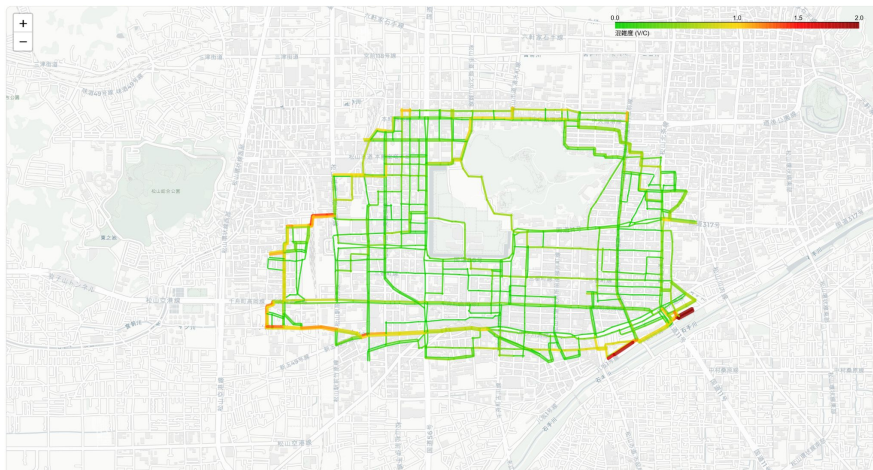
# 【1】松山市の渋滞の現状の把握



(図)現在のデータから、道路ごとの交通量 $v$ を図示したもの。

(図を分かりやすくするため、 $v < 10$ 台の細い道は色付けされていない。)  
(同じノード間の逆向きのリンクは左側通行で表示)

# 【1】松山市の渋滞の現状の把握



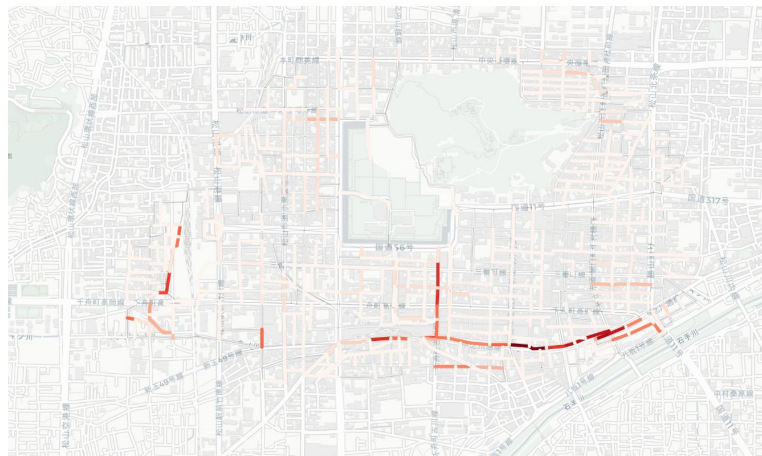
(図)現在のデータから、道路ごとの混雑度 $v/c$ を図示したもの。

(図を分かりやすくするため、 $v/c < 0.01$ の細い道は色付けされていない。)  
(同じノード間の逆向きのリンクは左側通行で表示)

## 【2】松山市におけるBraessのパラドックス

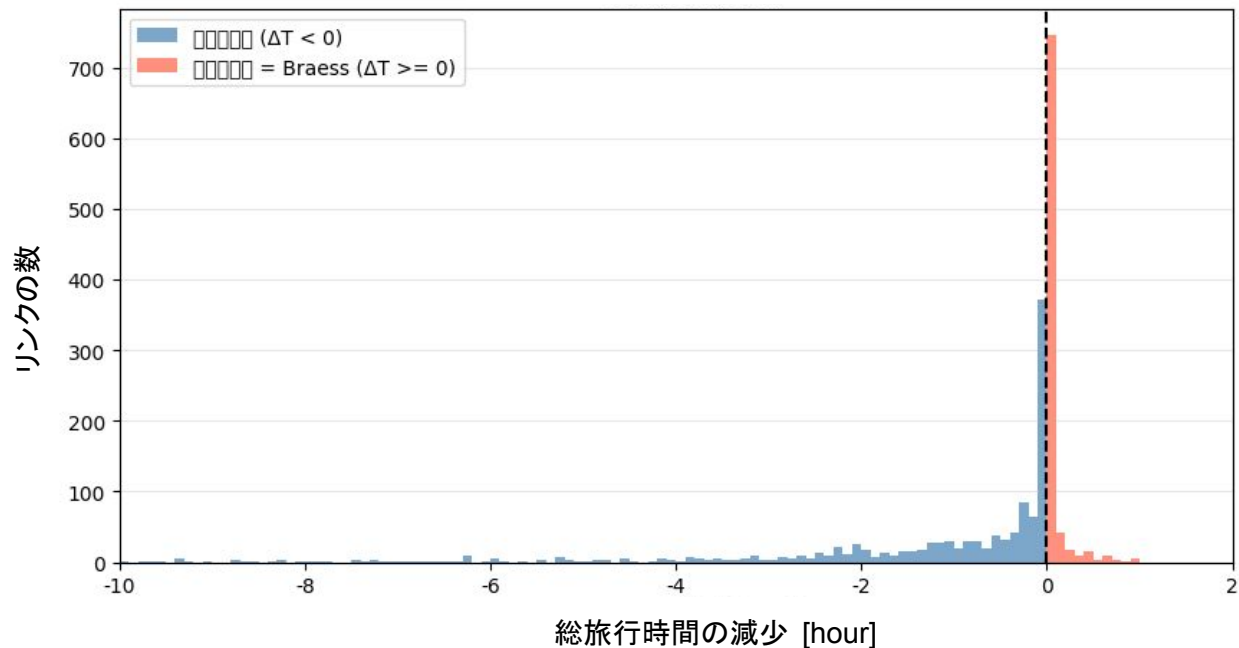
これらの道のなかに、渋滞悪化の原因となっている道路(Braessのパラドックス)があるかどうかを調べる。

全2140本の道を1本ずつ順番に消して利用者均衡配分を繰り返し行い、道を消す前と総旅行時間を比較した。



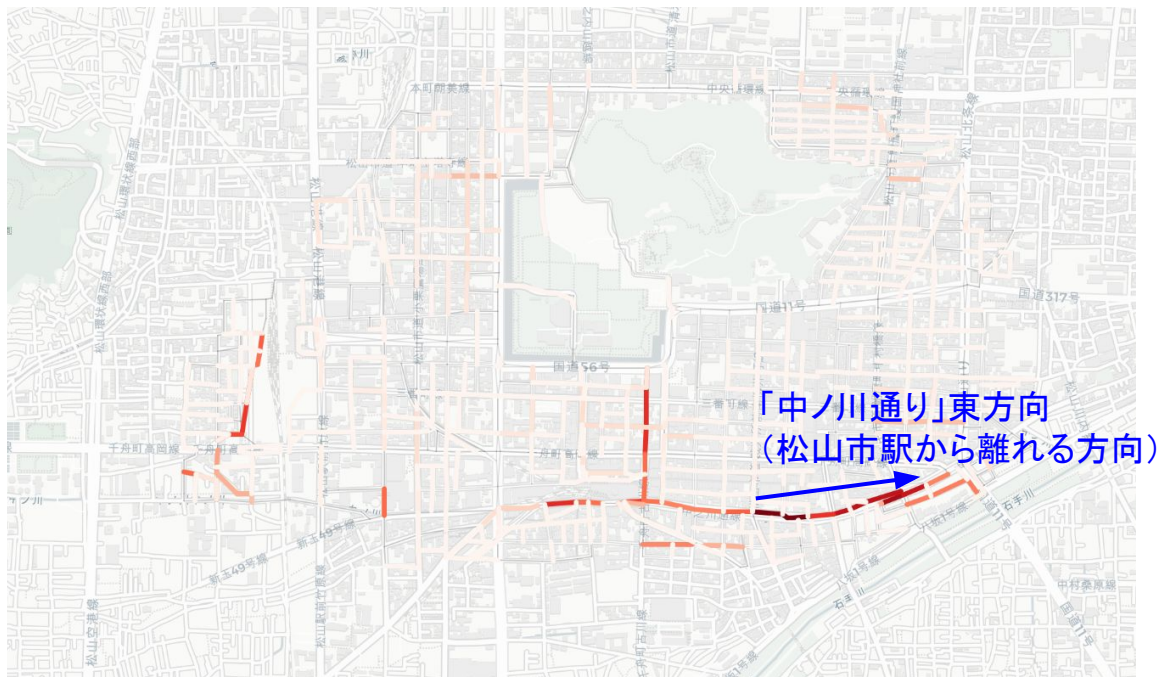
図の色のついた道が、消すことによって総旅行時間を減少させる道  
(色が濃いほど減少効果大きい)  
合計で854本(全体の40%)が、総旅行時間を短縮する効果があることがわかった。

## 【2】松山市におけるBraessのパラドックス



854本のうちほとんどは、総旅行時間の短縮はわずか。0.1時間以内が750本)  
一方で、1時間近く短縮できるリンクもいくつか見られる。

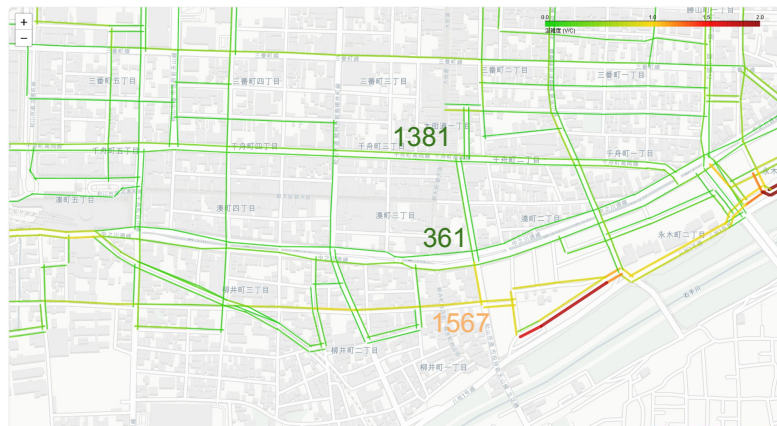
## 【2】松山市におけるBraessのパラドックス



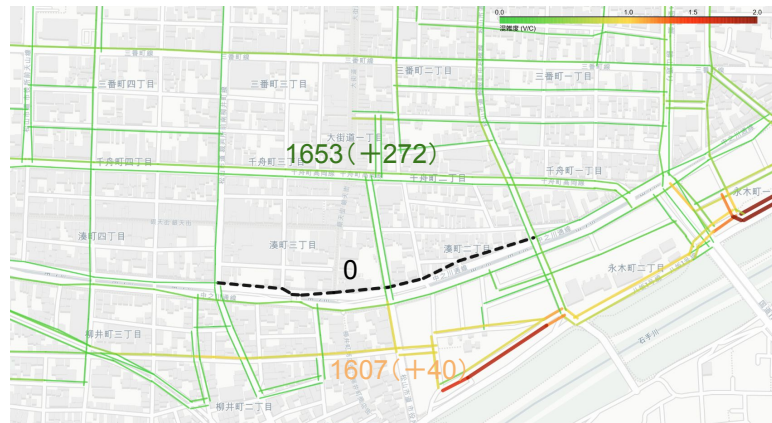
ストリートビューより。片側3車線で交通  
量も多い主要幹線道路

### 【3】中ノ川通りの東ゆきを閉鎖して計算してみると

通常時



閉鎖後(黒点線を閉鎖)

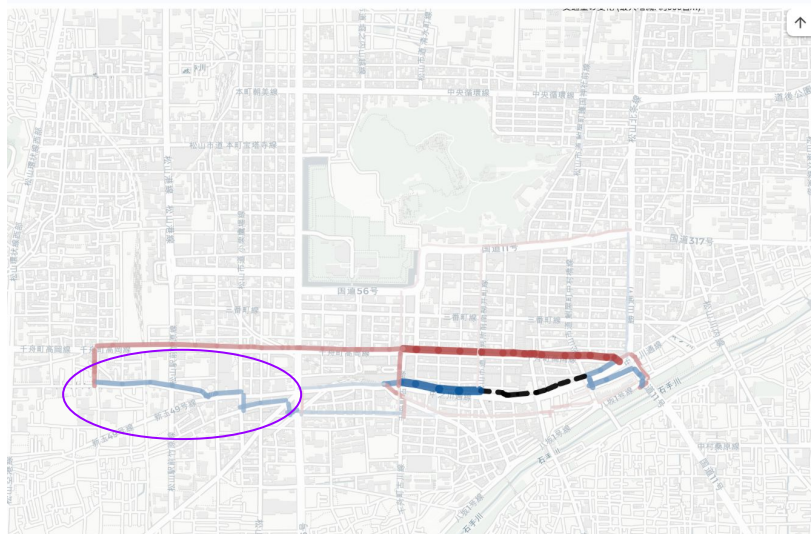


閉鎖された道を走っていた車は、ほとんどが南北の隣接する道に流れた。  
これによる総旅行時間の短縮効果は0.53時間(32分)

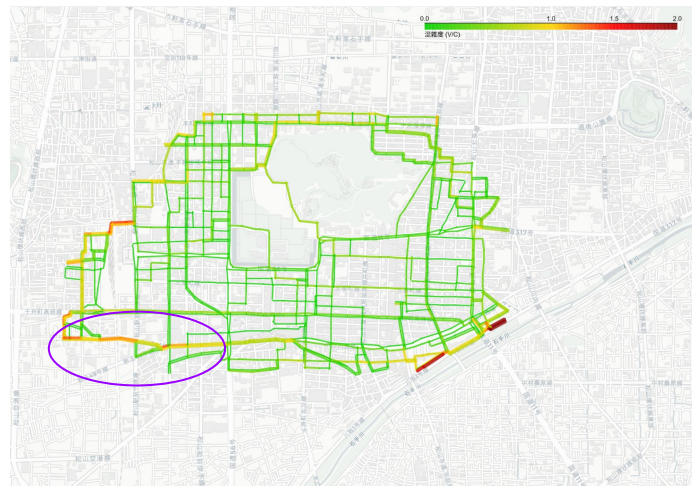
どうして総旅行時間が短縮しているのかは混雑度の図だけでは分からず、別の見方が必要がある。

※ももとの総旅行時間は2206.54時間

### 【3】中ノ川通りの東ゆきを閉鎖して計算してみると



(再掲)



赤色が閉鎖前と比べて交通量が増加した道・青色が減少した道を表している。色が濃いほど変化が大きい。

中ノ川通りの閉鎖により、図のむらさき色で囲った混雑区間を流れる車が減少した  
→ここで時間が短縮されている！

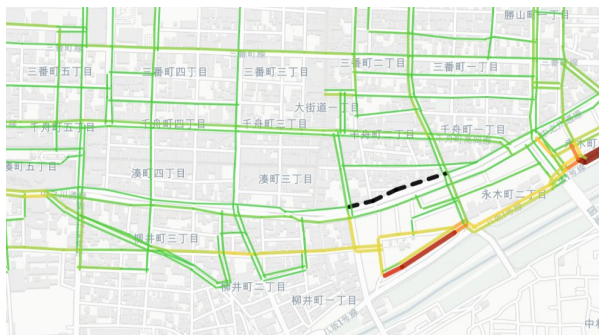
### 【3】中ノ川通りの東ゆきを閉鎖して計算してみると

ただ、リンクを1つだけ閉鎖した場合は最大で0.91時間の短縮だったので、これだと効果最大化できず、閉鎖しすぎな様子

→どこかに最適な閉鎖数が存在するはずで、それを探る

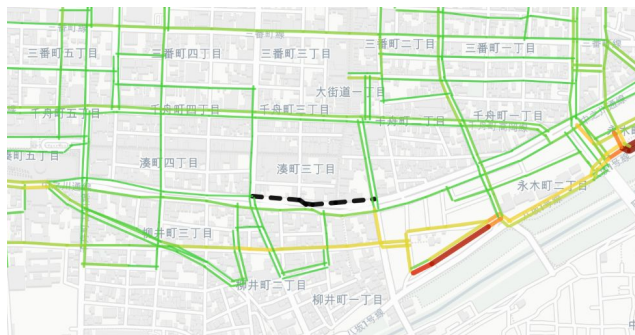
閉鎖する組み合わせを色々変えて検証

案① 閉鎖を東側の半分に減らす



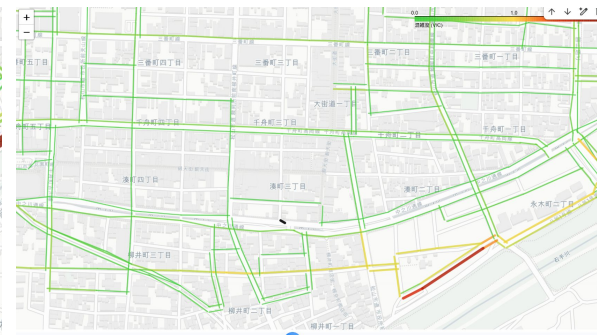
0.747252時間減少

案② 閉鎖を西側の半分に減らす



0.616249時間減少

案③ シミュレーションでできる最短の区間(約10m)のみ閉鎖



0.913798時間減少

### 【3】中ノ川通りの東ゆきを閉鎖して計算してみると

最適解は、最低限の閉鎖に留める方法。

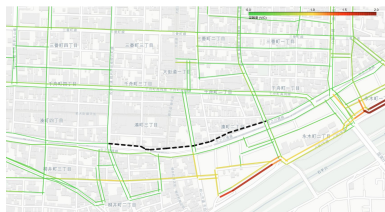
今回のリンク閉鎖はここよりも西側の混雑区間の流量を減らすことが目的なので、通り抜けさえできなくすれば十分。

(閉鎖の区間を長くすると、閉鎖区間の南北の道の混雑の悪化で総旅行時間が増えてしまう。)

**重ね合わせの原理は全く成立しない。**

さらなる混雑緩和を目指す

(仮説) どうせ通行止めにするなら、混雑する西ゆきの車線として運用すれば総旅行時間がさらに短縮されるのでは？



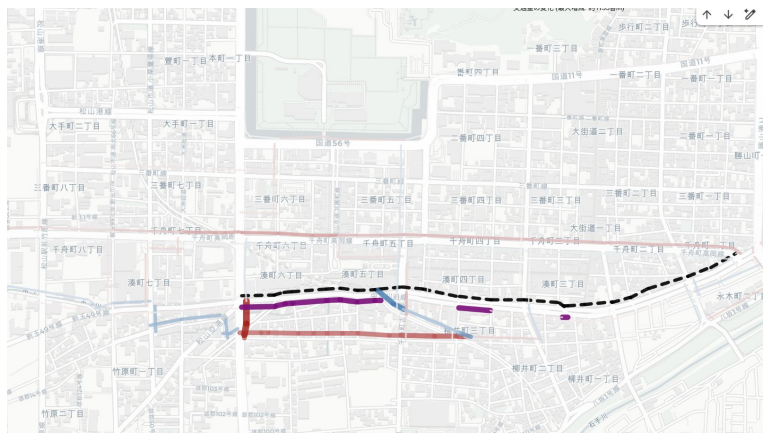
(結果) 0.56時間減少

閉鎖しただけのときは0.53時間減少だったので、計算上はわずかに効果があることが分かる。

### 【3】中ノ川通りの東ゆきを閉鎖して計算してみると

今回西ゆきに変更したのは500mほどで、西ゆきの混雑緩和は限定的だった。  
→もっと大規模にやったらどうなるか。

3倍の1500mほどにして検証。変更したリンクは、いずれも序盤の解析で、無くしたほうが総旅行時間が改善する道路ということになっている。

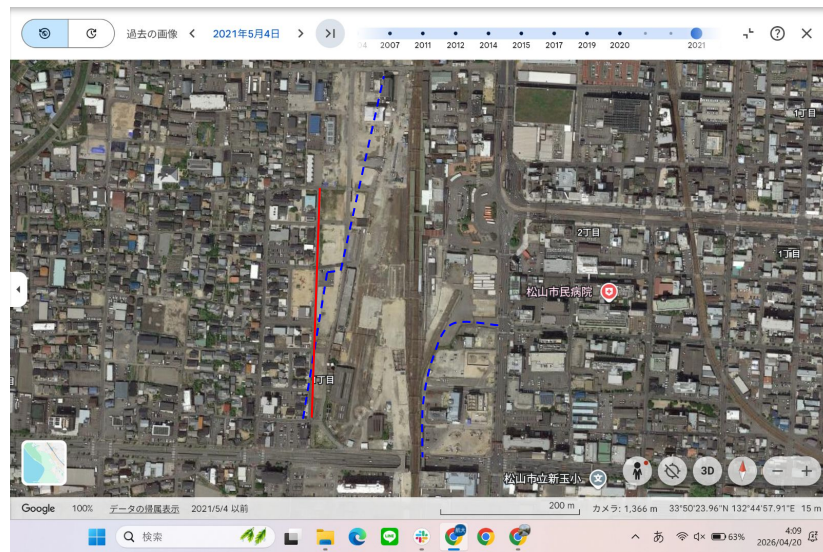
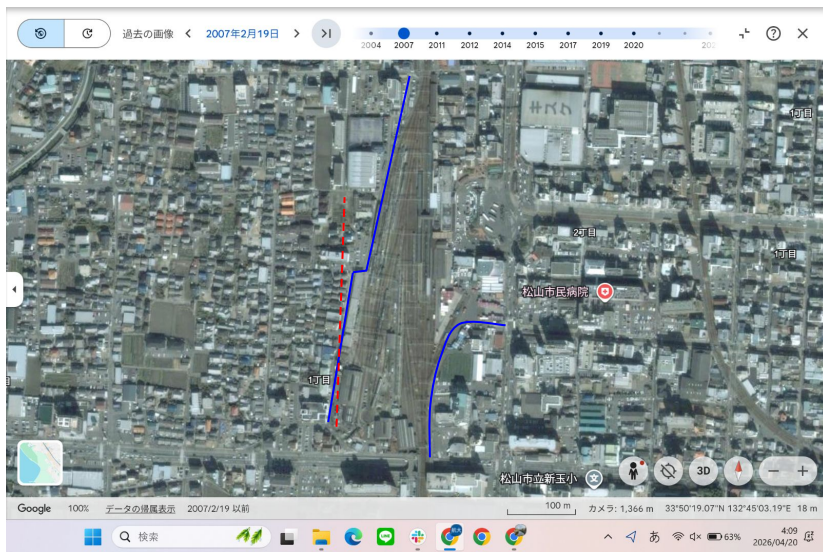


(結果)5.77時間悪化

松山市駅前を東に行けなくなったことで、南側の道が大渋滞してしまった。

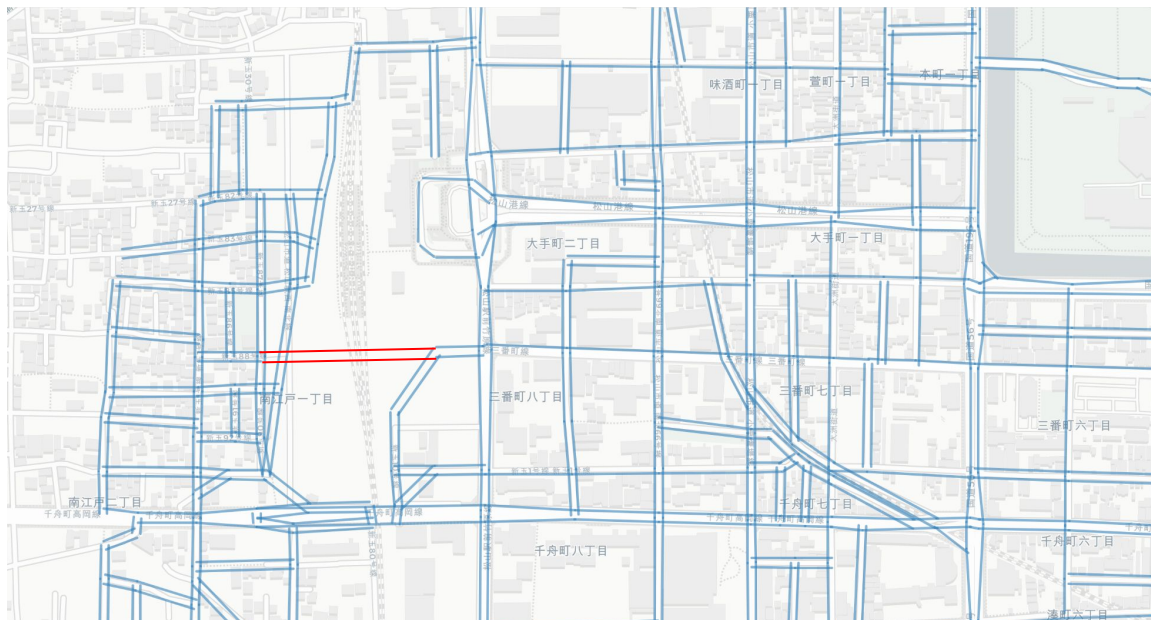


## 【4】JR松山駅高架化工事が道路渋滞に与える影響



PT調査の実施時点と現在で既存の道路割が変わっているので、完全な比較検討は不可能。  
→2007年時点の道路割において、JR松山駅を貫く仮想の道路を設定して検証。

## 【4】JR松山駅高架化工事が道路渋滞に与える影響

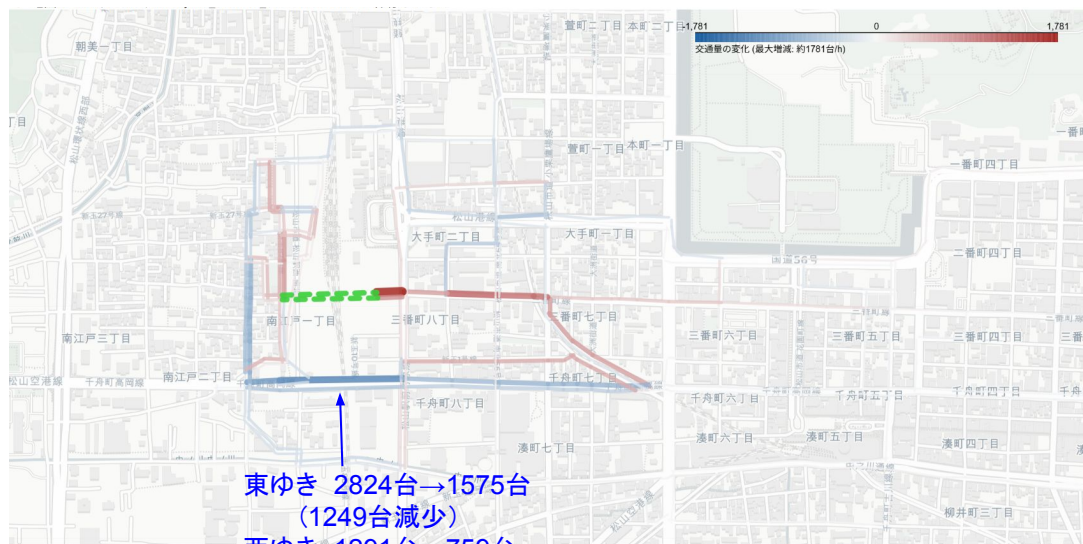


図の赤線部分に、  
・片側2車線  
・距離286m  
のリンクを新設して均衡配分を実施

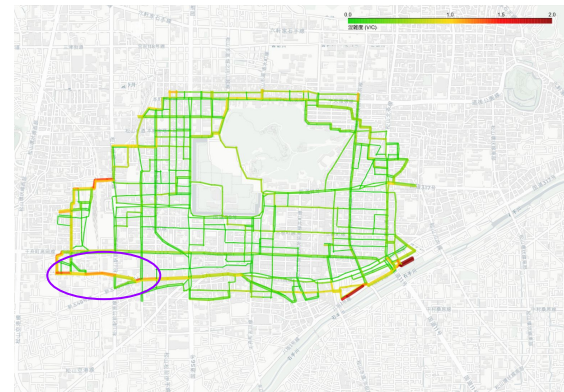
# 【4】JR松山駅高架化工事が道路渋滞に与える影響

## 【分析結果】

総旅行時間 18.89時間減少



(再掲)



## 【5】考察

分担交通量一定のもとで総旅行時間を削減するためには、  
最渋滞ポイントに流れ込む交通量を減らすことが重要

リンクパフォーマンス関数 (BPK関数) が下に凸の増加関数のため、  
混雑率の高いリンクを減らすことは大きな効果がある

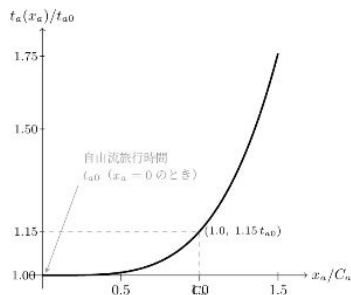


図 2.4 BPR 関数のグラフ ( $\alpha = 0.15, \beta = 4$ )。横軸は交通量  $x_a$  を容量  $C_a$  で正規化した値、縦軸は旅行時間  $t_a(x_a)$  を自由流旅行時間  $t_{a0}$  で正規化した値。  $x_a = C_a$  (容量到達点) での旅行時間は  $1.15 t_{a0}$  となる (縦軸)。

中ノ川通りの閉鎖・JR松山駅の道路新設は、いずれもこの点で効果があった。

## 【5】考察

モデルで再現しきれていない現実との差について考察する。

- ・自由旅行時間／交通容量の定め方  
(3車線の高規格道路の交通容量) $>3\times$ (1車線の細い道の交通容量)、なはず
- ・道路閉鎖によるロードサイド店舗への悪影響

# LLMの利用について

基本的な計算コードはすべていただいたので、

- ①その計算コードを使う環境整備
  - ②計算コードで何の計算をしているのか理解の手助け
  - ③課題でやりたくなかった分析に必要な新たなコードを書く
- の2点でGeminiを使いました。

## ①Colabのセットアップ

Gitからダウンロードしたものを、使い慣れたColaboratoryで使えるようにするやり方を、スクリーンショット送る→この次にやることは？と質問を繰り返した。  
ファイルの置き場所など、指示通りに回したら計算回せるようになった。

## ②コードの説明

Google Colaboratoryのスクリーンショットを撮り、この計算は何をしている？と質問  
→スクリーンショットを送った範囲だけで何をしているか分かってコードの理解に役立った

## ③新たな計算コードの作成

ここまでの会話でClaboratoryの中身をほとんどスクリーンショットで送っており、「追加で〇〇をやるコードを書いて」と打ち込むと、それをすぐに作成してくれ、エラーも比較的少なかった。

エラーについては、エラーの画面のスクリーンショットを送りつけて修正させることで対処

次ページ以降にGeminiで生成したコードを掲載します。

(「」がGeminiに送った文言)



# LLMの利用について

## ・道路の閉鎖の前後で交通量が増えた道、減った道を地図に示す

「混雑度可視化で、指定した道の消した時の混雑状況を出すコードが欲しい」という指示だけで生成。  
→出力は14頁の地図

```
import numpy as np
import pandas as pd
import folium
import branca.colormap as cm
from pathlib import Path
import time
import os

def analyze_multiple_closures(target_links, output_filename="flow_diff_max.html"):
    print(f"--- 1. 指定された {len(target_links)} 本のリンクの閉鎖シミュレーションを開始 ---")
    print(f"対象ID: {target_links}")

    # ベースマップ (地図) のマップを取得
    m = folium.Map(location=target_links[0], zoom_start=15, tiles='OpenStreetMap')

    # フロウベースマップを生成
    flow_base = FlowMap(m, target_links=target_links, verbose=False)

    # 閉鎖前後のネットワークを生成して計算
    original_car_links = net_car_links.copy()
    mask = original_car_links[["id", "start", "end"]]
    filtered_car_links = original_car_links[mask, :].copy()

    temp_dir = Path(tempfile.gettempdir())
    temp_link_csv_path = temp_dir / "temp_filtered_links.csv"
    filtered_car_links.to_csv(temp_link_csv_path, index=False)
    net_closed = NetworkXDCSV(temp_link_csv_path)

    # 経路計算
    flow_closed, ... = flow_max_flow_net_closed, id_list_max_iter=200, tol=1e-4, verbose=False)

    # 全体の経路コスト(1)を計算して表示
    T0 = net_total_travel_time(flow_base)
    T1 = net_total_travel_time(flow_closed)
    delta_T = T0 - T1
    print(f"経路コスト差: {delta_T}")
    print(f"経路コスト削減率: {delta_T / T0 * 100}")

    # 閉鎖前後のデータを比較して差分を計算
    df_compare = original_car_links[["id", "start", "end", "capacity", "start_lon", "end_lon", "end_lat", "end_lon"]].copy()
    df_compare["flow_before"] = flow_base

    filtered_car_links["flow_after"] = flow_closed
    df_compare["df_compare"] = df_compare[["flow_after"], "end_lon", "end_lat", "end_lon"]
    df_compare["flow_diff"] = df_compare["flow_after"] - df_compare["flow_before"]

    print(f"--- 2. 差分マップの生成 ---")
    diff_max_diff = df_compare["df_compare"].iloc[target_links][["flow_diff"]].abs()
    max_diff = diff_max_diff.max() if not diff_max_diff.empty else 1.0
    if max_diff == 0:
        max_diff = 1.0

    colormap = cm.LinearColormap(
        colors=["red", "yellow", "green"],
        vmin=max_diff, vmax=max_diff,
        caption=f"交通量の変化 (最大増減: {max_diff}, 単位: 車/分)"
    )

    m = folium.Map(location=(33.84, 132.76), zoom_start=15, tiles="CartoDB positron")
    m.add_child(colormap)
    affect_val = 0.0005

    for row in df_compare.dropna(subset=["start_lon", "start_lat", "end_lat", "end_lon"]):
        s_lon, s_lat = float(row["start_lon"], float(row["start_lat"]))
        e_lon, e_lat = float(row["end_lon"], float(row["end_lat"]))

        dx, dy, e_lon = s_lon, s_lat, e_lon - s_lon
        length = np.sqrt(dx**2 + dy**2)
        if length > 0:
            nx, ny = dx / length + affect_val, dy / length + affect_val
            s_lon, s_lat, e_lon = s_lon + nx, e_lat + ny, s_lon + nx, e_lat + ny

        points = [(s_lon, s_lat), (e_lon, e_lat)]
        if link_id in target_links:
            # 閉鎖された道すべて黄緑色
            if link_id in target_links:
                folium.PolyLine(
                    locations=points, color='black', weight=3, dash_array=[10, 10], opacity=0.5,
                    tooltip=f"<b>閉鎖 ID:{link_id}</b><br><b>減少量: {diff:.0%}</b> 単位:"
                ).add_to(m)
            continue

        # ノイズ除去
        if abs(dx) < 10:
            continue
        color = colormap(diff)
        weight = 2 + abs(diff) / max_diff + 8
        status = "閉鎖" if diff < 0 else "増大"
        tooltip_html = ""
        if abs(dx) > 10:
            tooltip_html += f"<b>リンクID:</b> {link_id}<br>"
            tooltip_html += f"<b>変化量:</b> {color.status} {color:color} {font-weight:bold}<b>{diff}</b> 単位: {status}<br>"
            tooltip_html += f"<b>閉鎖前:</b> {flow_before}、<b>閉鎖後:</b> {flow_after}、<b>差:</b> {diff}"
        else:
            tooltip_html += f"<b>リンクID:</b> {link_id}<br>"
            tooltip_html += f"<b>変化量:</b> {color.status} {color:color} {font-weight:bold}<b>{diff}</b> 単位: {status}<br>"
            tooltip_html += f"<b>閉鎖前:</b> {flow_before}、<b>閉鎖後:</b> {flow_after}、<b>差:</b> {diff}"

        folium.PolyLine(
            locations=points, color=color, weight=weight, opacity=0.5, tooltip=tooltip_html
        ).add_to(m)

    # 差分マップの生成
    net_closed.to_csv(temp_dir / "net_closed_links.csv")
    save_path = RESULT_DIR / output_filename
    m.save(temp_dir / save_path)
    print(f"--- 完了 ---")
    return m

# 実行例
target_links = [954, 1563, 128, 1562, 1782, 132, 134, 133]
map_multiple = analyze_multiple_closures(target_links, "natsvava_multiple_closures.html")
map_multiple

--- 1. 指定された 8 本のリンクの閉鎖シミュレーションを開始 ---
対象ID: [954, 1563, 128, 1562, 1782, 132, 134, 133]
--- 2. 差分マップの生成 ---
--- 完了 --- 地図名: data/result/natsvava_multiple_closures.html に保存しました ---

【実行結果】
閉鎖前 経路コスト: 2204.54
閉鎖後 経路コスト: 2096.01
--- 2. 差分マップの生成 ---
--- 完了 --- 地図名: data/result/natsvava_multiple_closures.html に保存しました ---
```



# LLMの利用について

## ・リンクを追加する

「リンクを追加して計算を回すコードってすぐに作れる？JR松山駅の高架化の効果を検討したくて」

```
REPORT nodes as no
REPORT paths as pd
REPORT flows as fl
REPORT branca.colormap as cm
FROM paths() REPORT Path
REPORT fls

def simulate new roadflow links df, output filename="new road effect.htm":
  print("==== 1. 新架橋の追加によるネットワークの変更 ====")

  # ベースグラフ (現状) のグラフと初期値を準備
  if 'flows' not in libcols():
    print("ベースグラフを生成します。")
    flows_base, ... = frank_walker_init, od_list_max(Iter=200, tol=1e-4, verbose=False)
  else:
    flows_base = flows

  T0 = net_total_travel_time(flows_base)

  # 既知のリンクデータを読み込み、追加 (concat) する
  # 時刻表 car.links = net_car.links.csv()

  # 新たな経路のデータと完全一致させる
  # boolean型が "car" と "net" は True/False を設定するように修正
  for col in original_car.links.columns:
    if col not in new_links_df.columns:
      if col == "cat":
        new_links_df[col] = True # 新架橋は自動車専用と仮定
      elif col == "net":
        new_links_df[col] = False # 新架橋は歩行者専用ではないと仮定
      else:
        new_links_df[col] = 0 # 足りない列は初期値である

  new_links_df = new_links_df[original_car.links.columns]

  # データをマッピングする
  expanded_links = pd.concat([original_car.links, new_links_df, ignore_index=True])

  # 拡張後のグラフをグラフライブラリを作成
  TempLink = GraphLink(flows_base, expanded_links)
  temp_links_csv_path = temp_dir + "temp_expanded_links.csv"
  expanded_links.to_csv(temp_links_csv_path, index=False)

  # 新しいネットワークを作成して初期値を計算
  print("==== 2. 新しいネットワークで初期値を計算中 ====")
  net_expanded = NetworkXIOCSV_CV2_TempLink_csv_path
  (flows_expanded, ... = frank_walker_init_expanded, od_list_max(Iter=200, tol=1e-4, verbose=False))

  # 初期値計算 (A1)
  T_expanded = net_expanded.total_travel_time(flows_expanded)
  delta_T = T0 - T_expanded

  print("==== 3. 初期値計算結果 (初期値計算の結果) ====")
  print("初期値計算結果: T_expanded = %f" % T_expanded)
  print("delta_T = %f" % delta_T)
  print("初期値計算結果: 全体で (delta_T): %f (A・時間) の効果を実現できました！")

  # 出力
  print("==== 4. 出力 ====")
  # 元のグラフと拡張後の (既知+追加+既知) のグラフを出力
  df_colormap = expanded_links[['df', 'start', 'end', 'category', 'start_lat', 'start_lon', 'end_lat', 'end_lon']], cm
  # ネットワークマッピング (拡張後のネットワークのみとする)
  df_colormap = df_colormap.merge(original_car.links[['df', 'id', 'lat', 'lon', 'type']], how="left")
  df_colormap["flow_before"] = df_colormap["df"].apply(lambda row: original_car.links["df", flows_base], axis=1)
  df_colormap["flow_after"] = flows_expanded
  df_colormap["flow_diff"] = df_colormap["flow_after"] - df_colormap["flow_before"]

  print("==== 5. 差分マップ (拡張後の結果) ====")
  # 拡張後のネットワークのマッピングの最大値を求める
  max_diff = df["max_diff"] if df_colormap["df"].isna().sum() == 0 else df["flow_diff"].max()
  if max_diff == 0:
    print("max_diff = 0")
  else:
    print("max_diff = %f" % max_diff)

  # 出力
  new_new_road = simulate_new_roadflow_road_data("Networks.station_simulation.htm")
  return new_road
```

追加するリンクについては、接続するノードID、それぞれの緯度経度、長さについて手動入力のコ드가書かれた。ノードの一覧表があるのでこの部分は省略可能と思われるが、追加したいのは1本だけだったのでこれ以上はコードを書かず、手作業で対応した。