30538 Final Project: Institutional and Functional Integration of State-orchestrated regional strategy: Yangtze River Delta

2024-12-05

Student Information

Partner1: Zijing Zhao (cnetid: zijingz, github username: zijing328) Partner2: Zekai Shen (cnetid:zekaishen, github username:zac-shen)

Plot Inter-city Population Flow

```
# package needed
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib.colors as mcolors
from shapely.geometry import LineString
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
import os
from matplotlib.patches import Patch, PathPatch, Circle,Polygon
from matplotlib.collections import PatchCollection
from matplotlib.path import Path
from matplotlib.lines import Line2D
import numpy as np
```

Create plot for inter-city population flow

```
#convert dta file into excel file
dta_file_path =
 ↪  r"/Users/zhaozijing/Documents/GitHub/30538_final_project/data/      18-23.dta"
data = pd.read_stata(dta_file_path)

#save the excel file
excel_file_path =
 ↪  r"/Users/zhaozijing/Documents/GitHub/30538_final_project/data/      18-23.xlsx"
data.to_excel(excel_file_path, index=False)

print(f"document has been saved to: {excel_file_path}")
```

```
document has been saved to:
/Users/zhaozijing/Documents/GitHub/30538_final_project/data/      18-23.xlsx
```

```
# create inter-city average migration index

df =
 ↪  pd.read_excel("/Users/zhaozijing/Documents/GitHub/30538_final_project/data/      18-23.x

#rename the column into english, start city as city1, destination city as
 ↪  city2
df = df.rename(columns={"  ": "city1"})
df = df.rename(columns={"  ": "city2"})
df = df.rename(columns={" ": "year"})
df = df.rename(columns={"    ": "migration_index"})

# Define a standard mapping for commonly accepted English translations of
 ↪  Chinese city names
city_name_mapping = {
    " ": "Shanghai",
    " ": "Hangzhou",
    " ": "Ningbo",
    " ": "Wenzhou",
    " ": "Jiaxing",
    " ": "Huzhou",
    " ": "Shaoxing",
    " ": "Jinhua",
    " ": "TAIzhou",
```

```
    " ": "Zhoushan",
    " ": "Quzhou",
    " ": "Lishui",
    " ": "Nanjing",
    " ": "Wuxi",
    " ": "Changzhou",
    " ": "SUzhou",
    " ": "Nantong",
    " ": "Yangzhou",
    " ": "Zhenjiang",
    " ": "Taizhou",
    " ": "Xuzhou",
    " ": "Yancheng",
    " ": "Huaian",
    "  ": "Lianyungang",
    " ": "Suqian",
    " ": "Wuhu",
    "  ": "Maanshan",
    " ": "Tongling",
    " ": "Anqing",
    " ": "Huangshan",
    " ": "Chuzhou",
    " ": "Fuyang",
    " ": "Suzhou",
    " ": "Luan",
    " ": "Chizhou",
    " ": "Xuancheng",
    " ": "Hefei",
    " ": "Huaibei",
    " ": "Bengbu",
    " ": "Bozhou",
    " ": "Chuzhou",
    " ": "Huainan",
}

# Apply the mapping to city1 and city2 columns
df['city1'] = df['city1'].map(city_name_mapping)
df['city2'] = df['city2'].map(city_name_mapping)

# make sure that citypair ij and citypair ji is considered as the same
 ↪  citypair
df['city_pair'] = df.apply(lambda row: tuple(sorted([row['city1'],
 ↪  row['city2']])), axis=1)
```

```
# calculate the average value of migration index between citypair ij and
 ↪  citypair ji
df_avg = df.groupby(['city_pair', 'year'],
 ↪  as_index=False)['migration_index'].mean()

# resplit the citypair
df_avg['city1'] = df_avg['city_pair'].apply(lambda x: x[0])
df_avg['city2'] = df_avg['city_pair'].apply(lambda x: x[1])

# output dataframe
df_avg_export = df_avg[['city_pair', 'city1', 'city2', 'year',
 ↪  'migration_index']]
df_avg_export.columns = ['city_pair', 'city1', 'city2', 'year',
 ↪  'avg_migration_index']

# save as excel
output_file_path =
 ↪  "/Users/zhaozijing/Documents/GitHub/30538_final_project/data/avg_migration_18-23.xlsx"
df_avg_export.to_excel(output_file_path, index=False)

print(f"document has been saved to:{output_file_path}")
```

```
document has been saved
to:/Users/zhaozijing/Documents/GitHub/30538_final_project/data/avg_migration_18-23.xlsx
```

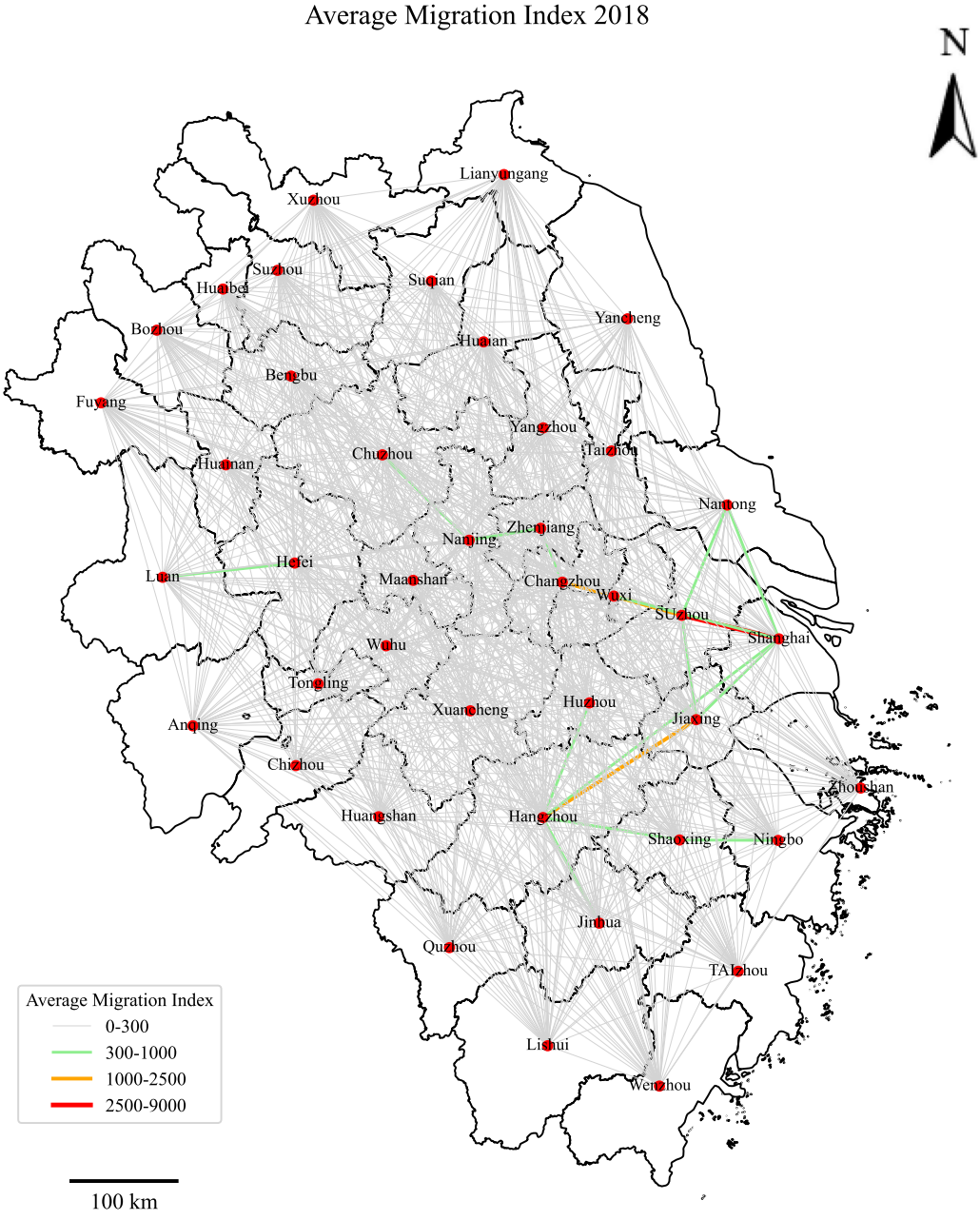
```
## Create Static Plot of inter-city average migration index
os.environ['OMP_NUM_THREADS'] = '1'
plt.rcParams['font.family'] = 'Times New Roman'

# load excel
df =
 ↪  pd.read_excel("/Users/zhaozijing/Documents/GitHub/30538_final_project/data/avg_migration_

# load shapefile
city_boundaries =
 ↪  gpd.read_file("/Users/zhaozijing/Documents/GitHub/30538_final_project/data/shapefile/   _

# create city centroid
city_boundaries['centroid'] = city_boundaries['geometry'].centroid
city_centers = city_boundaries.set_index('cityE')['centroid'].to_dict()
```

```
# create lines
def create_linestring(row):
    point1 = city_centers[row['city1']]
    point2 = city_centers[row['city2']]
    return LineString([point1, point2])

# create lines according to average migration index value
df['geometry'] = df.apply(create_linestring, axis=1)

# convert to GeoDataFrame
avg_migration_network = gpd.GeoDataFrame(df, geometry='geometry',
 ↪  crs=city_boundaries.crs)

# Define color mapping for imbalance index ranges
color_map = {
    '0-300': 'lightgrey',
    '300-1000': 'lightgreen',
    '1000-2500': 'orange',
    '2500-9000': 'red',
}

width_map = {
    '0-300': 0.3,
    '300-1000': 1.5,
    '1000-2500': 2.0,
    '2500-9000': 2.5,
}

# Categorize imbalance values
def get_color_and_width(value):
    if 0 <= value < 300:
        return color_map['0-300'], width_map['0-300']
    elif 300 <= value < 1000:
        return color_map['300-1000'], width_map['300-1000']
    elif 1000 <= value < 2500:
        return color_map['1000-2500'], width_map['1000-2500']
    elif 2500 <= value < 9000:
        return color_map['2500-9000'], width_map['2500-9000']
    return 'grey', 0.5  # Default color for any value outside specified
     ↪  ranges

avg_migration_network['color'], avg_migration_network['width'] =
 ↪  zip(*avg_migration_network['avg_migration_index'].apply(get_color_and_width))
```

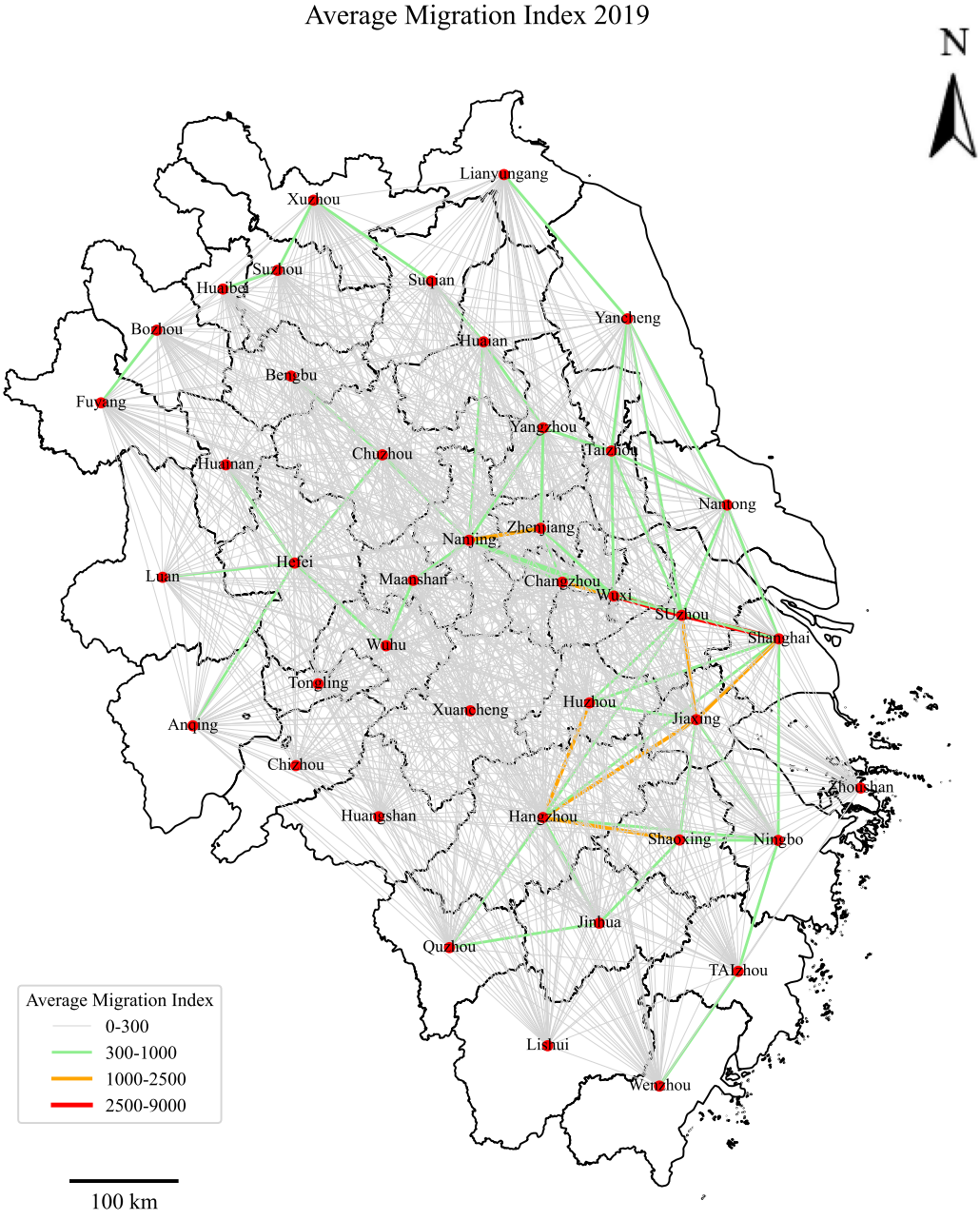
```
# create plot
for year in avg_migration_network['year'].unique():
    fig, ax = plt.subplots(1, figsize=(10, 12))
    # province_boundaries.plot(ax=ax, color='none', edgecolor='black')
    city_boundaries.plot(ax=ax, color='none', edgecolor='black')

    # add city name
    for idx, row in city_boundaries.iterrows():
        ax.text(row.geometry.centroid.x, row.geometry.centroid.y,
↪   row['cityE'], fontsize=9, ha='center', va='center', color='black',
↪   zorder=6)
        ax.plot(row.geometry.centroid.x, row.geometry.centroid.y, 'o',
↪   color='red', markersize=5, zorder=4)  # add red central dot

  # create logistics line according to average migration index
    year_data = avg_migration_network[avg_migration_network['year'] == year]
    for _, row in year_data.iterrows():
        line = row['geometry']
        color = row['color']
        width = row['width']
        ax.plot(*line.xy, color=color, linewidth=width)

    # Define legend elements for line thickness and color based on migration
     ↪  imbalance index
    line_elements = [
    mlines.Line2D([], [], color='lightgrey', linewidth=0.3, label='0-300'),
    mlines.Line2D([], [], color='lightgreen', linewidth=1.5,
↪   label='300-1000'),
    mlines.Line2D([], [], color='orange', linewidth=2.0, label='1000-2500'),
    mlines.Line2D([], [], color='red', linewidth=2.5, label='2500-9000')]

    legend1 = ax.legend(handles=line_elements, loc='lower left',
↪   bbox_to_anchor=(0.05, 0.1),fontsize=10, title="Average Migration Index")

    # create scale bar
    scalebar_location = (0.11, 0.06)  # adjust position of scale bar
    scale_length = 100000  # set the scale length
    x_length = scale_length / (ax.get_xlim()[1] - ax.get_xlim()[0])

    # draw scale line
```

```
    ax.plot([scalebar_location[0], scalebar_location[0] + x_length],
            [scalebar_location[1], scalebar_location[1]],
            color='black', transform=ax.transAxes, lw=2)

    # add scale text
    ax.text(scalebar_location[0] + x_length / 2, scalebar_location[1] - 0.01,
↪   '100 km',
            horizontalalignment='center', verticalalignment='top',
            transform=ax.transAxes, fontsize=12)

    # add north arrow
    north_arrow_img =
↪   plt.imread("/Users/zhaozijing/Downloads/north_arrow1.png")
    imagebox = OffsetImage(north_arrow_img, zoom=0.5)
    ab = AnnotationBbox(imagebox, (0.95, 0.95), xycoords='axes fraction',
↪   frameon=False)
    ax.add_artist(ab)

    # Set the year title and remove the axis
    ax.set_title(f'Year {year}', loc='center')
    ax.set_xticks([])
    ax.set_yticks([])

    # save the picture
    plt.title(f"Average Migration Index {year}", fontdict={'fontname': 'Times
↪   New Roman', 'fontsize': 15})
    plt.axis('off')

↪   plt.savefig(f"/Users/zhaozijing/Documents/GitHub/30538_final_project/plot/avg_migration_
↪   format='jpg', dpi=300)
    plt.show()
```

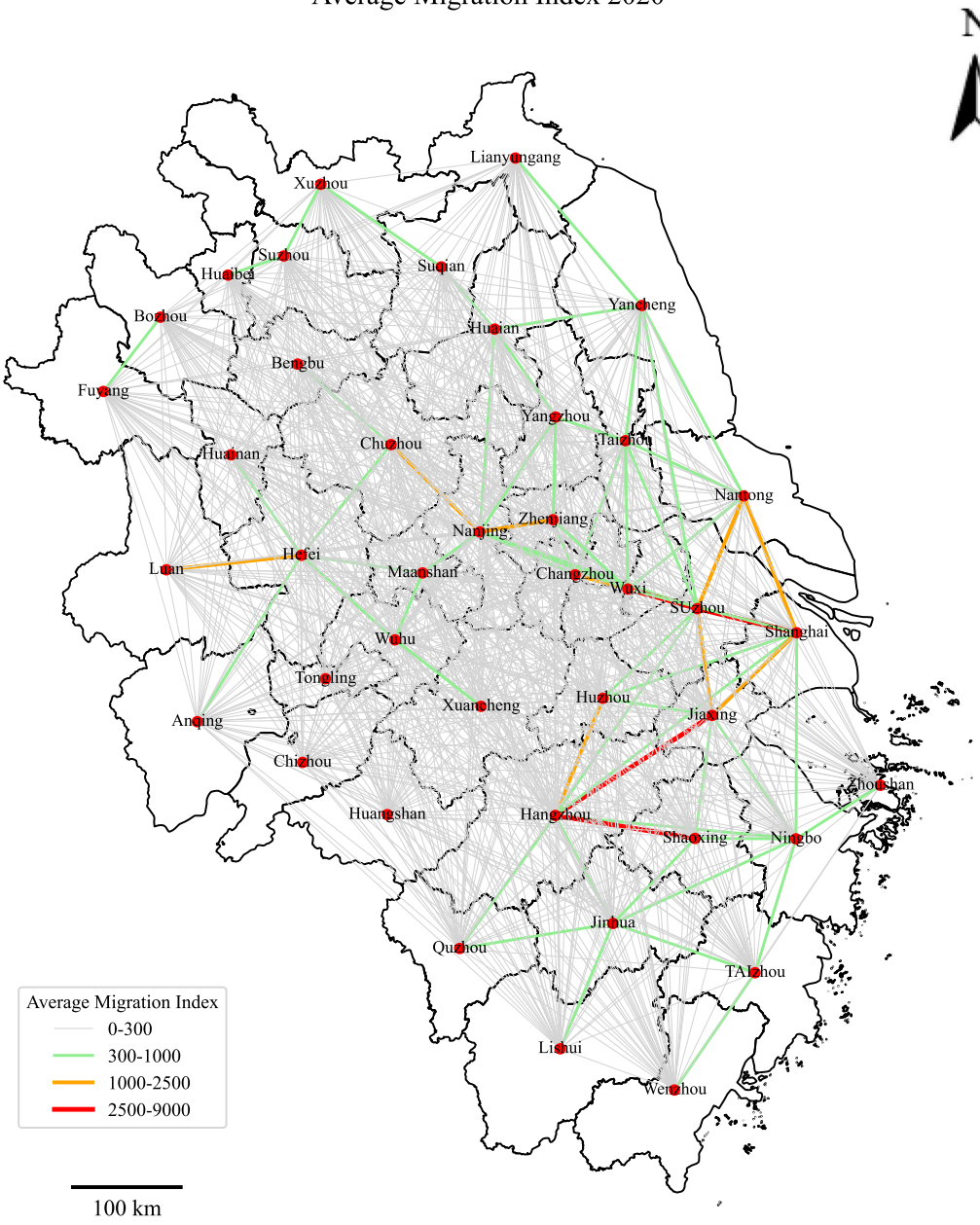
Average Migration Index 2018



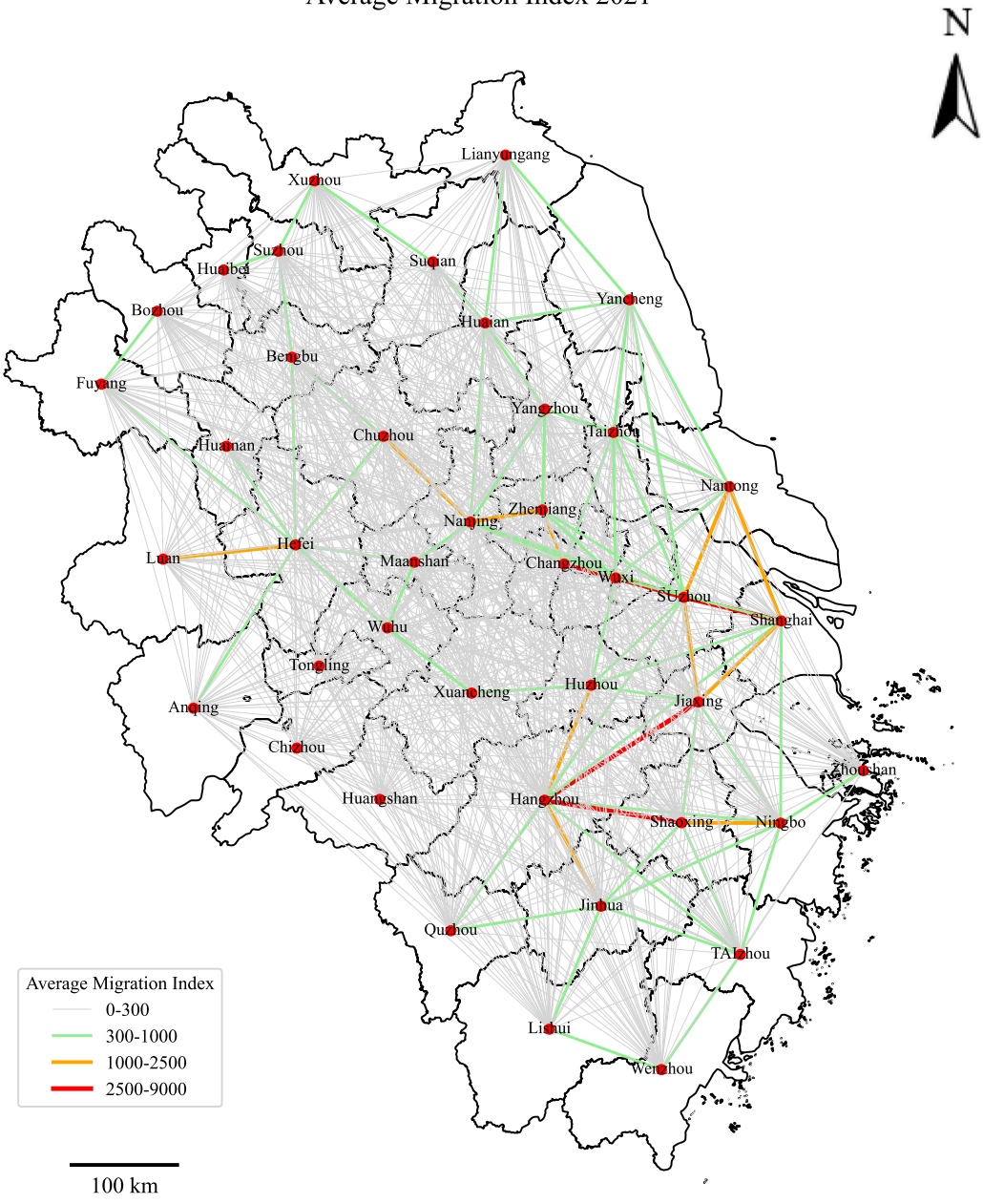
Average Migration Index 2019



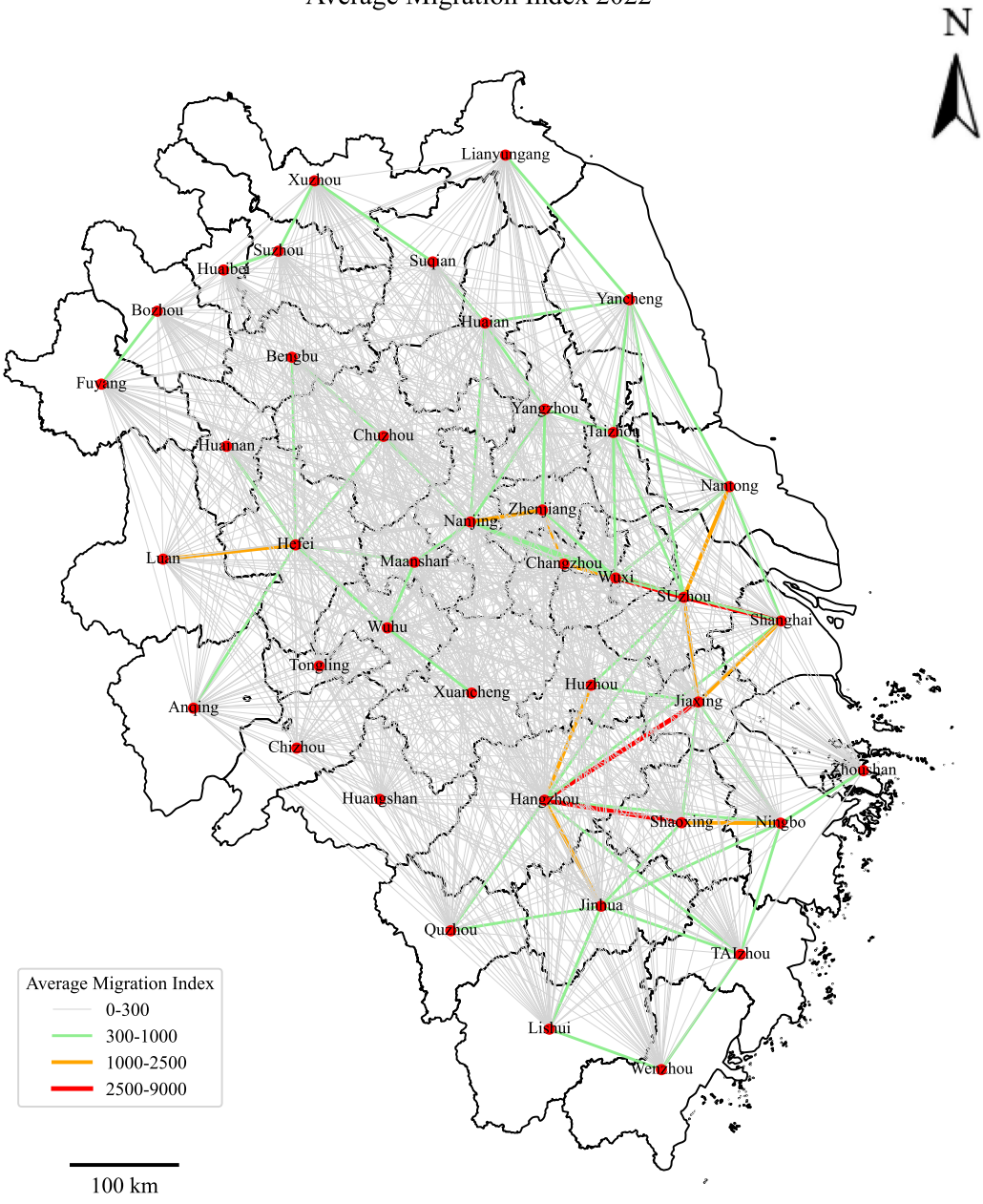
Average Migration Index 2020



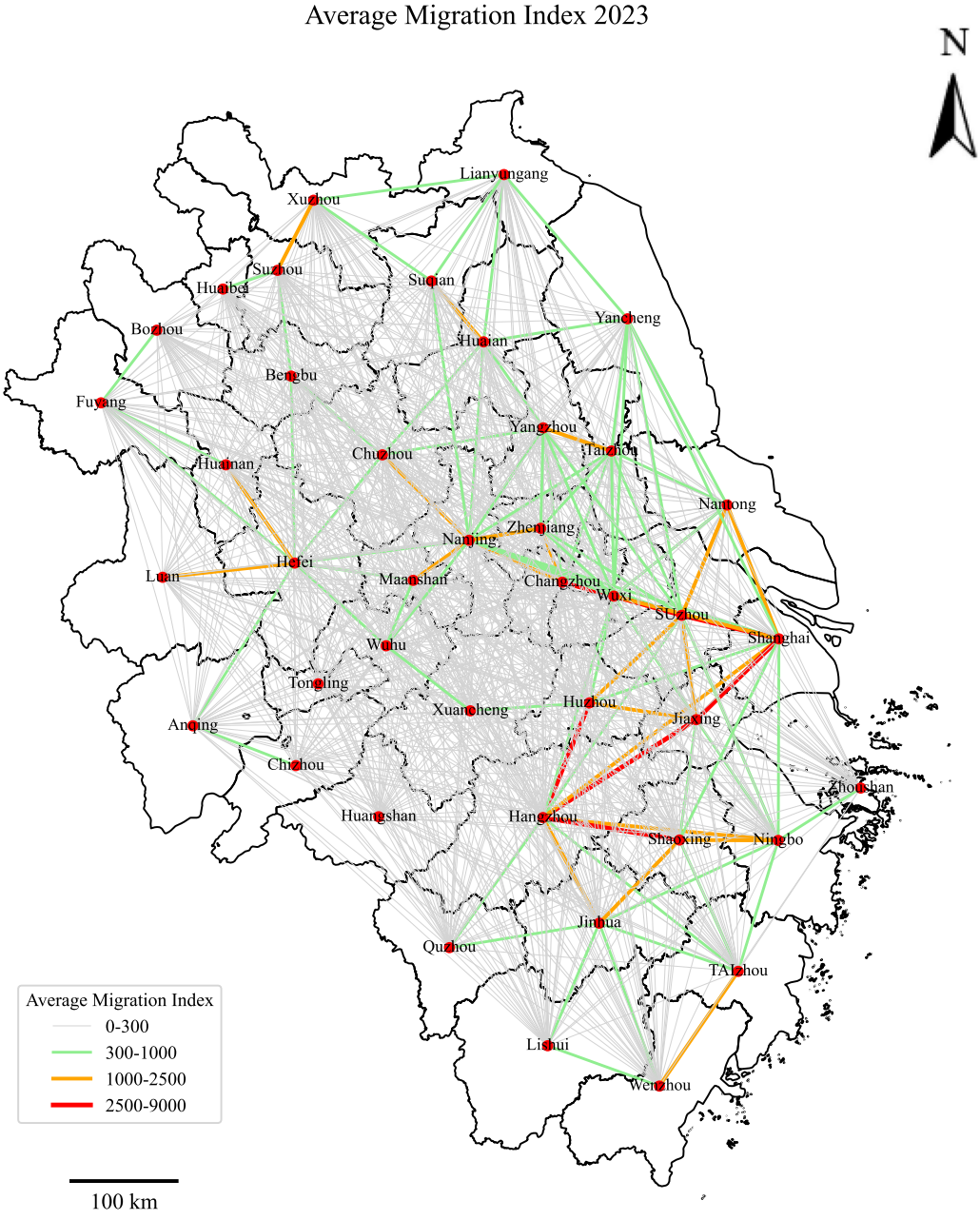
Average Migration Index 2021



Average Migration Index 2022



Average Migration Index 2023



Plot Inter-city Cooperation

```
os.environ['OMP_NUM_THREADS'] = '1'
plt.rcParams['font.family'] = 'Times New Roman'

# load shapefile
city_boundaries =
 ↪  gpd.read_file("/Users/zhaozijing/Documents/GitHub/30538_final_project/data/shapefile/   _

# load inter-city cooperation data
df =
 ↪  pd.read_excel("/Users/zhaozijing/Documents/GitHub/30538_final_project/data/citypair_outpu
 ↪  sheet_name="sum")
df = df[(df['year'] >= 2018) & (df['year'] <= 2023)]

# Define a standard mapping for commonly accepted English translations of
 ↪  Chinese city names
city_name_mapping = {
    " ": "Shanghai",
    " ": "Hangzhou",
    " ": "Ningbo",
    " ": "Wenzhou",
    " ": "Jiaxing",
    " ": "Huzhou",
    " ": "Shaoxing",
    " ": "Jinhua",
    " ": "TAIzhou",
    " ": "Zhoushan",
    " ": "Quzhou",
    " ": "Lishui",
    " ": "Nanjing",
    " ": "Wuxi",
    " ": "Changzhou",
    " ": "SUzhou",
    " ": "Nantong",
    " ": "Yangzhou",
    " ": "Zhenjiang",
    " ": "Taizhou",
    " ": "Xuzhou",
    " ": "Yancheng",
    " ": "Huaian",
```

```
    "  ": "Lianyungang",
    " ": "Suqian",
    " ": "Wuhu",
    "  ": "Maanshan",
    " ": "Tongling",
    " ": "Anqing",
    " ": "Huangshan",
    " ": "Chuzhou",
    " ": "Fuyang",
    " ": "Suzhou",
    " ": "Luan",
    " ": "Chizhou",
    " ": "Xuancheng",
    " ": "Hefei",
    " ": "Huaibei",
    " ": "Bengbu",
    " ": "Bozhou",
    " ": "Chuzhou",
    " ": "Huainan",
}

# Apply the mapping to city1 and city2 columns
df['cityi'] = df['cityi'].map(city_name_mapping)
df['cityj'] = df['cityj'].map(city_name_mapping)
df['sum'] = df['meeting_count'] + df['agreement_count'] + df['project_count']
 ↪  + df['visit_count']

# Calculate the centroid and radius
def get_circle(x1, y1, x2, y2, x3, y3):
    a = x1 - x2
    b = y1 - y2
    c = x1 - x3
    d = y1 - y3
    a1 = ((x1 ** 2 - x2 ** 2) + (y1 ** 2 - y2 ** 2)) / 2.0
    a2 = ((x1 ** 2 - x3 ** 2) + (y1 ** 2 - y3 ** 2)) / 2.0
    theta = b * c - a * d
    if abs(theta) < 1e-7:
        raise ValueError("Three collinear points do not define a circle.")
    x0 = (b * a2 - d * a1) / theta
    y0 = (c * a1 - a * a2) / theta
    r = np.sqrt((x1 - x0) ** 2 + (y1 - y0) ** 2)
    return x0, y0, r
```

```
# add north arrow
def add_north(ax, x, y, text_size, arrow_width, text_pad, arrow_height,
 ↪  line_width, add_circle):
    x_min, x_max = ax.get_xlim()
    y_min, y_max = ax.get_ylim()
    width = x_max - x_min
    height = y_max - y_min

    left = (x_min + width * (x - arrow_width / 2), y_min + height * (y -
 ↪  arrow_height))
    right = (x_min + width * (x + arrow_width / 2), left[1])
    top = (x_min + width * x, y_min + height * y)

    left_patch = Polygon([left, top, (top[0], left[1])], closed=True,
 ↪  edgecolor='black', facecolor='black')
    ax.add_patch(left_patch)

    right_patch = Polygon([right, top, (top[0], left[1])], closed=True,
 ↪  edgecolor='black', facecolor='white')
    ax.add_patch(right_patch)

    if add_circle:
        x0, y0, r = get_circle(left[0], left[1], top[0], top[1], right[0],
 ↪  right[1])
        circle = Circle((x0, y0), r, edgecolor='black', facecolor='none',
 ↪  linewidth=line_width)
        ax.add_patch(circle)
        ax.text(x0, y0 + 1.2*r + text_pad, 'N', horizontalalignment='center',
 ↪  verticalalignment='center',
                fontsize=text_size, family='Arial', zorder=5)

# attrieve year
years = df['year'].unique().tolist()

# categorize core city and peripheral city
city_names = city_boundaries[['cityE', 'geometry']]
core_cities = ['Bengbu', 'Changzhou', 'Chuzhou', 'Hangzhou', 'Hefei',
 ↪  'Huaian', 'Huainan', 'Huangshan', 'Huzhou', 'Jiaxing', 'Liuan',
 ↪  'Maanshan', 'Nanjing', 'Nantong', 'Ningbo', 'Quzhou', 'Shanghai',
 ↪  'Shaoxing', 'SUzhou', 'TAIzhou', 'Wuhu', 'Wuxi', 'Xuancheng', 'Yangzhou',
 ↪  'Zhenjiang', 'Zhoushan']
```

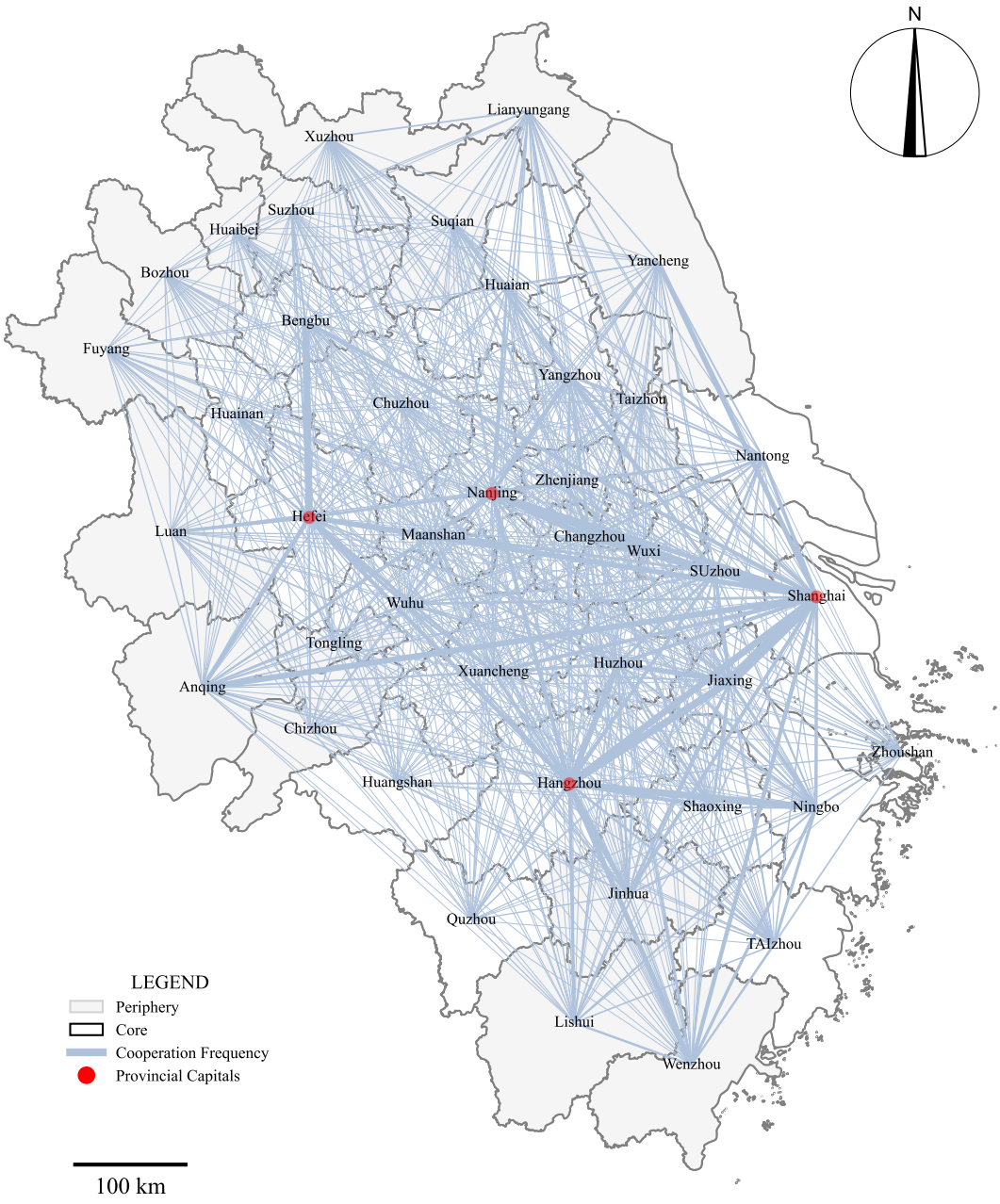
```
city_names['city_type'] = 'peripheral'
city_names.loc[city_names['cityE'].isin(core_cities), 'city_type'] = 'core'

# identify color of core city and peripheral city
color_map = {'core': 'white', 'peripheral': '#f5f5f5'}

# create city centroid
city_centers =
 ↪  city_boundaries.set_index('cityE')['geometry'].centroid.to_dict()

# create lines using city centroid
def create_linestring(row):
    point1 = city_centers[row['cityi']]
    point2 = city_centers[row['cityj']]
    return LineString([point1, point2])

df['geometry'] = df.apply(create_linestring, axis=1)
logistics_network = gpd.GeoDataFrame(df, geometry='geometry',
 ↪  crs=city_boundaries.crs)


# create map
for year in df['year'].unique():
    fig, ax = plt.subplots(1, figsize=(10, 12))
    #province_boundaries.plot(ax=ax, color='none', edgecolor='grey')
    city_boundaries.plot(ax=ax, color='none', edgecolor='grey')
     # create city name
    for idx, row in city_names.iterrows():
        city_boundary = city_boundaries[city_boundaries['cityE'] ==
 ↪  row['cityE']]
        city_boundary.plot(ax=ax, color=color_map[row['city_type']],
 ↪  edgecolor='grey')
        ax.text(row.geometry.centroid.x, row.geometry.centroid.y,
 ↪  row['cityE'], fontsize=8, ha='center', va='center', color='black')
    # create logistics line
    year_data = logistics_network[logistics_network['year'] == year]
    year_data.plot(ax=ax, linewidth=year_data['sum']/65, color='#AFC2DC',
 ↪  zorder=2)

    # set marks for capital cities
    cities_to_mark = ['Nanjing', 'Hefei', 'Shanghai', 'Hangzhou']
    patches = []
```

```
    for city in cities_to_mark:
        city_point = city_centers[city]
        circle = Circle((city_point.x, city_point.y), 5000, color='red',
↪   alpha=0.5)
        patches.append(circle)

    p = PatchCollection(patches, match_original=True, zorder=3)
    ax.add_collection(p)

     # create scale bar
    scalebar_location = (0.11, 0.06)
    scale_length = 100000
    x_length = scale_length / (ax.get_xlim()[1] - ax.get_xlim()[0])

    ax.plot([scalebar_location[0], scalebar_location[0] + x_length],
            [scalebar_location[1], scalebar_location[1]],
            color='black', transform=ax.transAxes, lw=2)

    ax.text(scalebar_location[0] + x_length / 2, scalebar_location[1] - 0.01,
↪   '100 km',
            horizontalalignment='center', verticalalignment='top',
            transform=ax.transAxes, fontsize=12)

    # add north arrow
    add_north(ax, x=0.88, y=0.95, text_size=10, arrow_width=0.02,
              text_pad=0.01, arrow_height=0.1, line_width=0.5,
↪   add_circle=True)

    # add legend
    legend_elements = [
    Patch(facecolor='#f5f5f5', edgecolor='lightgrey', label='Periphery'),
    Patch(facecolor='none', edgecolor='k', label='Core'),
    Line2D([0], [0], color='#AFC2DC', lw=4, label='Cooperation Frequency'),
    Line2D([0], [0], marker='o', color='w', markersize=10,
↪   markerfacecolor='red', label='Provincial Capitals')
]

    leg = ax.legend(handles=legend_elements, title='LEGEND', loc='upper
↪   left', bbox_to_anchor=(0.092,0.22), fontsize='small',
↪   title_fontsize='medium')
    leg.get_frame().set_linewidth(0.0)
    # add year title
```

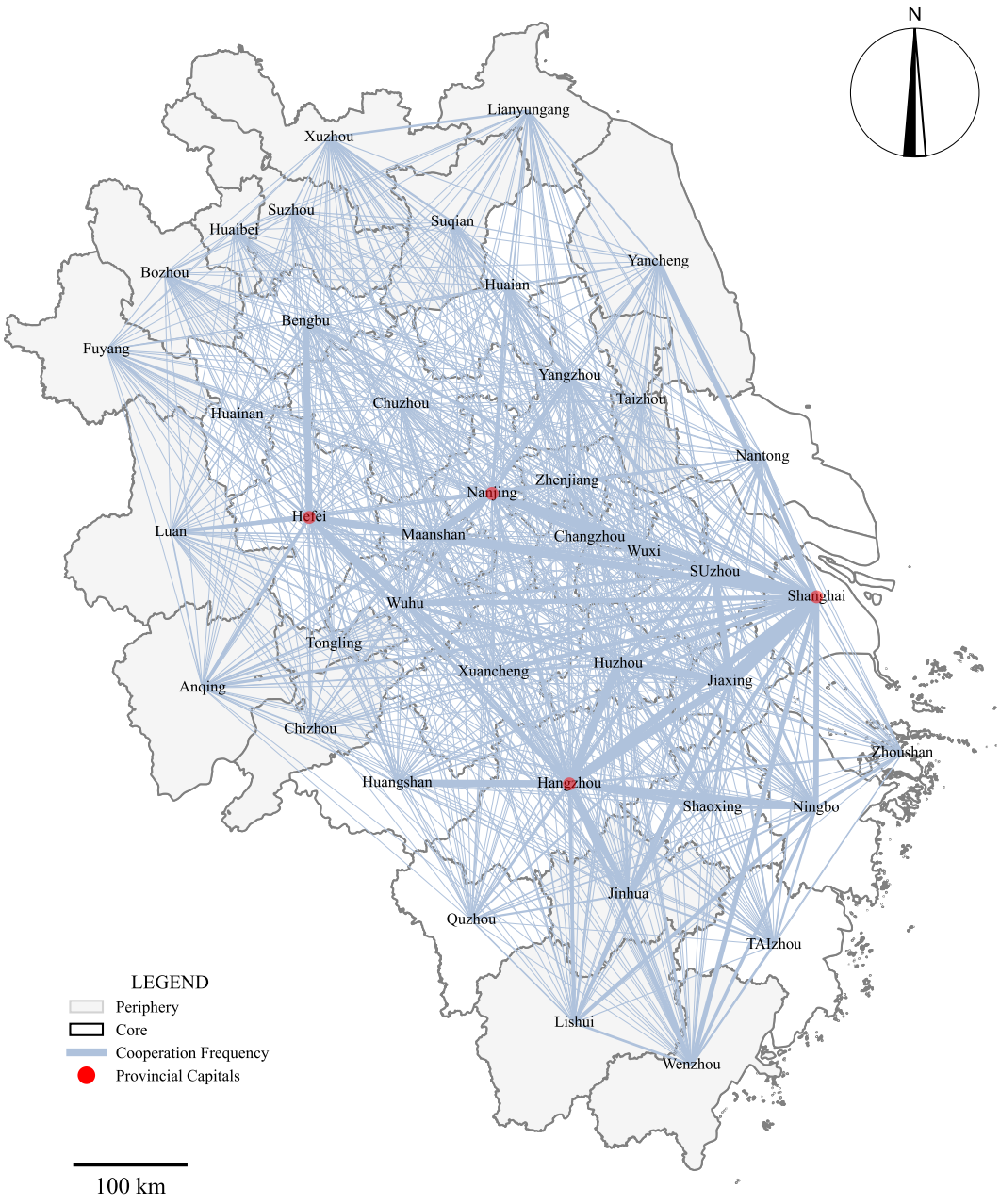
```
    ax.set_title(f'Year {year}', loc='center')
    # remove x and y-axis
    ax.set_xticks([])
    ax.set_yticks([])
    # set title
    plt.title(f"Inter-City Cooperation {year}", fontdict={'fontname': 'Times
↪   New Roman', 'fontsize': 25})
    plt.axis('off')

↪   plt.savefig(f"/Users/zhaozijing/Documents/GitHub/30538_final_project/plot/coop_network_su
↪   format='jpg',dpi=300)
    plt.show()
```

```
/Users/zhaozijing/anaconda3/lib/python3.12/site-packages/geopandas/geodataframe.py:1819:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
```

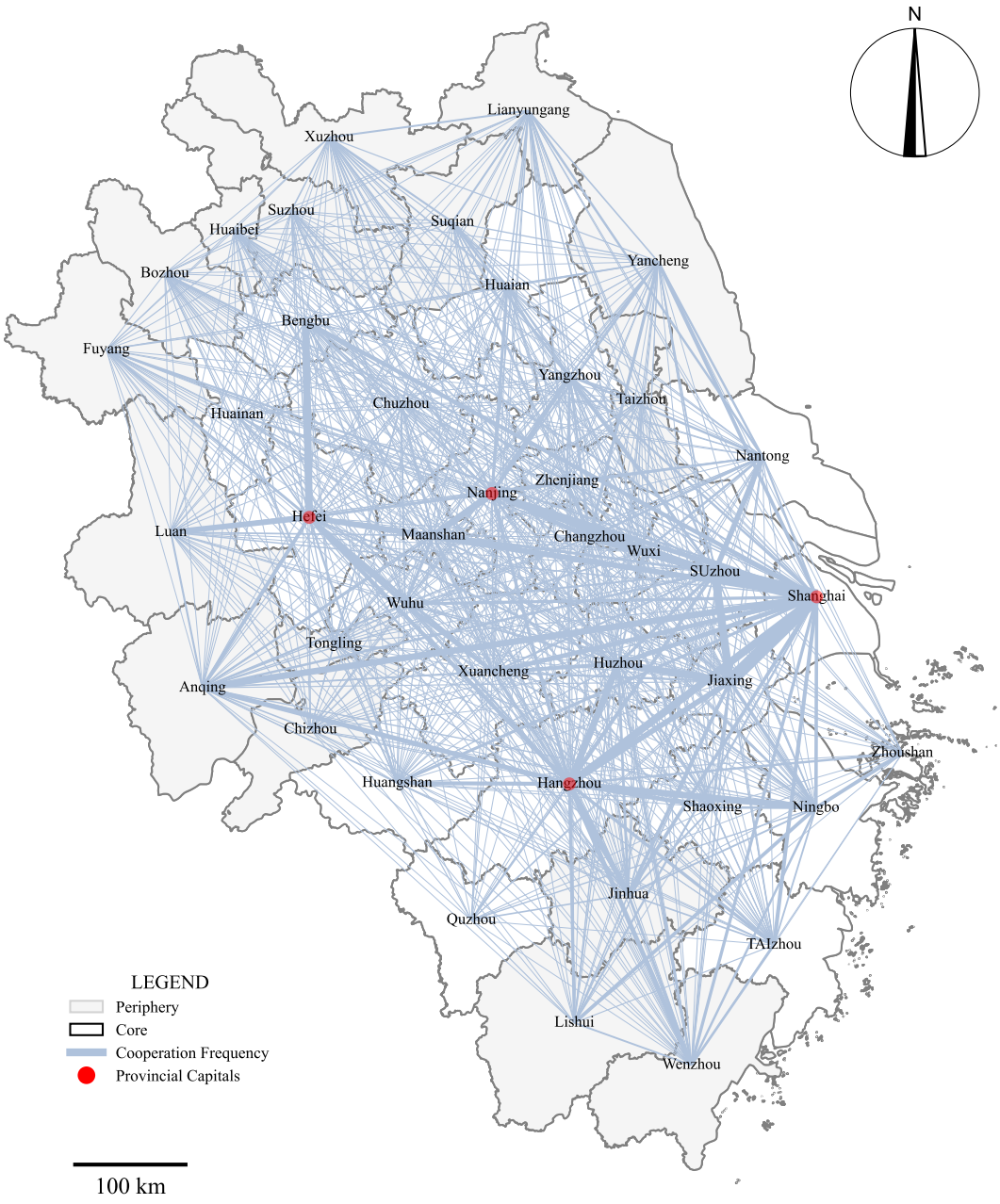
Inter-City Cooperation 2018



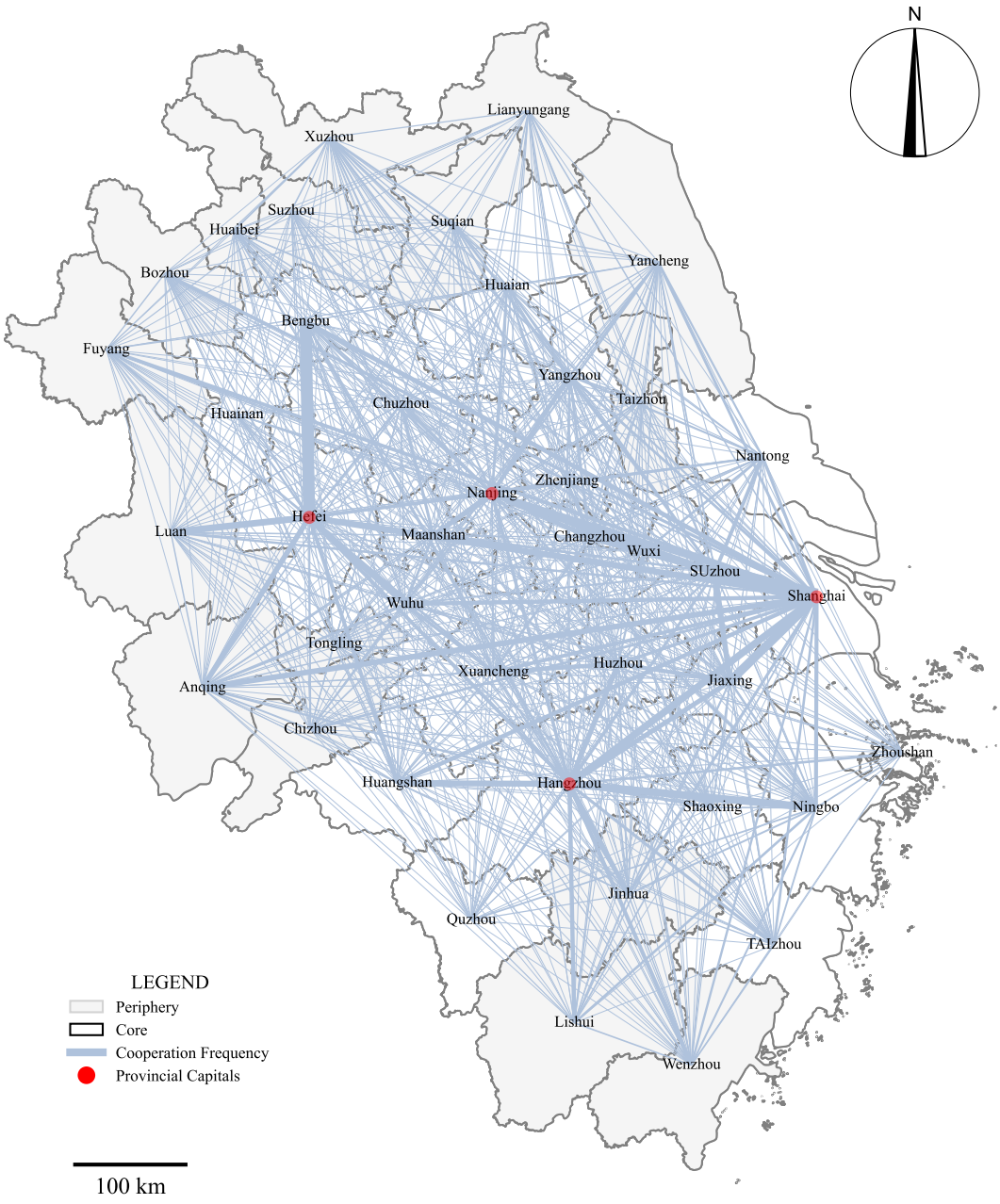
Inter-City Cooperation 2019



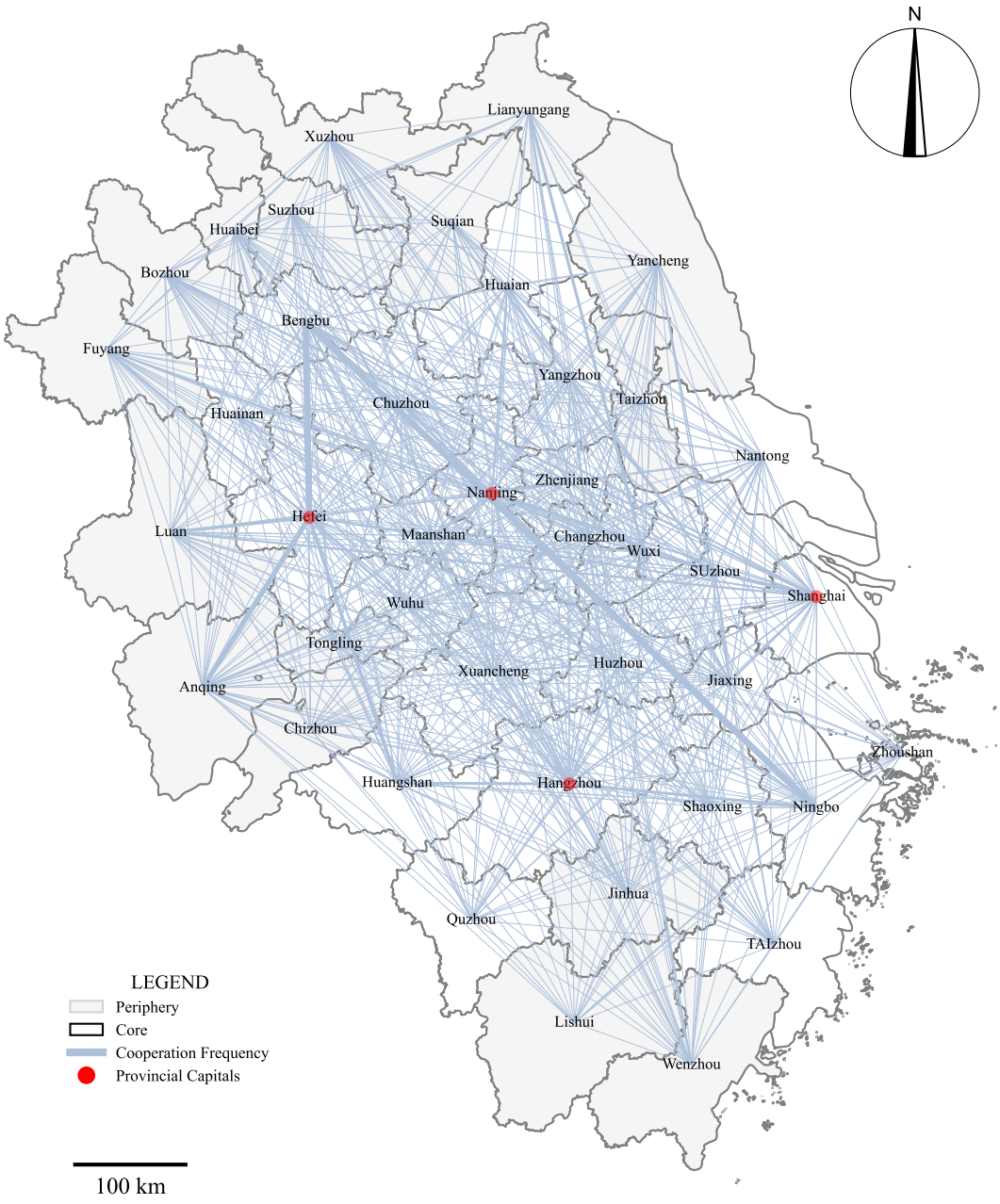
Inter-City Cooperation 2020



Inter-City Cooperation 2021



Inter-City Cooperation 2022



Inter-City Cooperation 2023