In [2]:
```
import osmnx as ox
import networkx as nx
import netCDF4 as nc
import numpy as np
import matplotlib.pyplot as plt
from shapely import MultiPolygon, Polygon
from matplotlib.patches import Rectangle
from shapely.ops import unary_union
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.colors as mcolors
import pandas as pd

import csv
```

In [2]:
```
# Load the road network for a Chicago

graph_path = r'C:\Users\elchi\Desktop\UIC_Chicago\Knowledge_graph\Backend\Backtrack


G = ox.load_graphml(graph_path)
G = ox.routing.add_edge_speeds(G)
G = ox.routing.add_edge_travel_times(G)
```

In [3]:
```
# Create bounding boxes to get random origin and destination points

# Top box (corrected coordinates in proper order: clockwise from top-left)
# Southwest is bottom-left, Northeast is top-right

Top = 41.955628 #Cuarto numero de la pagina
Bottom = 41.969293 #Segudo numero de la pagina
East = -87.649612 #Tercer numero de la pagina
West = -87.790712 #Primer numero de la pagina

Top_Southwest = (Top, West)      # (41.985628, -87.870712)
Top_Southeast = (Top, East)      # (41.985628, -87.649612)
Top_Northeast = (Bottom, East)   # (41.969293, -87.649612)
Top_Northwest = (Bottom, West)

# Bottom box (corrected coordinates in proper order: clockwise from top-left)

#http://bboxfinder.com/#41.877741,-87.802972,41.894100,-87.615461


Top = 41.894100 #Cuarto numero de la pagina
Bottom = 41.877741 #Segudo numero de la pagina
East = -87.615461 #Tercer numero de la pagina
West = -87.762972 #Primer numero de la pagina

Bottom_Southwest = (Bottom, West)  # (41.877741, -87.802972)
Bottom_Southeast = (Bottom, East)  # (41.877741, -87.615461)
Bottom_Northeast = (Top, East)     # (41.894100, -87.615461)
Bottom_Northwest = (Top, West)    # (41.894100, -87.802972)
```

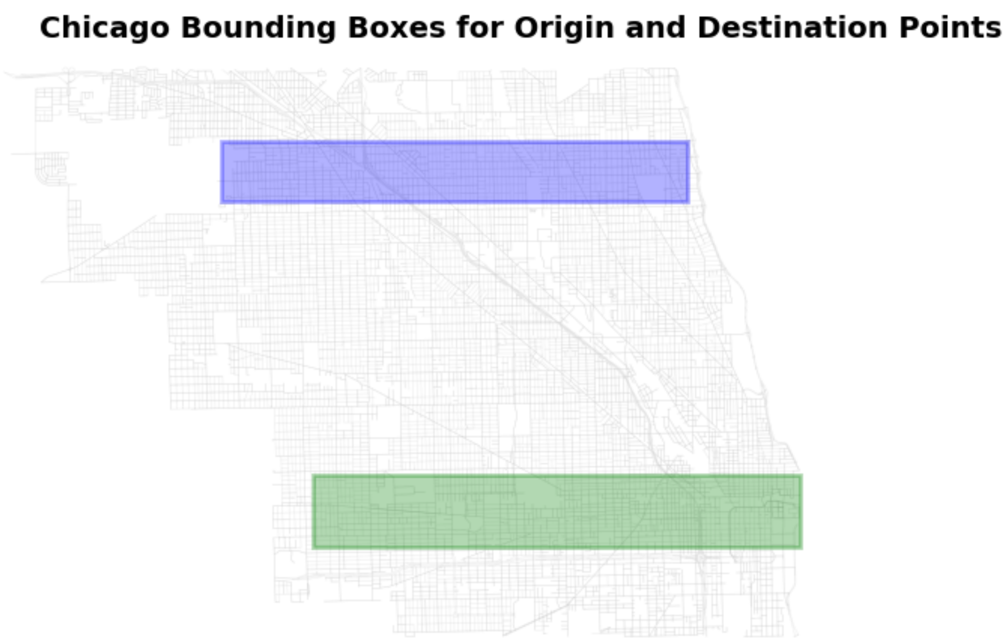
```python
# Create polygons for the bounding boxes (clockwise order)
top_box = Polygon([Top_Northwest, Top_Northeast, Top_Southeast, Top_Southwest])
bottom_box = Polygon([Bottom_Northwest, Bottom_Northeast, Bottom_Southeast, Bottom_

# Verify polygons are valid
print("Top box is valid:", top_box.is_valid)
print("Bottom box is valid:", bottom_box.is_valid)
print("Top box area:", top_box.area)
print("Bottom box area:", bottom_box.area)
```

```
Top box is valid: True
Bottom box is valid: True
Top box area: 0.0019281315000003682
Bottom box area: 0.002413132449000339
```

In [4]:

```python
# Map the bounding boxes for Chicago
fig, ax = plt.subplots(1, 1, figsize=(12, 10))

# Plot the street network first as background
ox.plot_graph(G, ax=ax, node_size=0, edge_color='#cccccc', edge_linewidth=0.3,
              show=False, close=False, edge_alpha=0.4)

# Set the map extent to show both Chicago boxes
min_lon, min_lat = -87.88, 41.67  # Lower-left corner for Chicago area
max_lon, max_lat = -87.52, 41.99  # Upper-right corner for Chicago area

ax.set_xlim(min_lon, max_lon)
ax.set_ylim(min_lat, max_lat)

# Top box rectangle (Origins)
top_width = Top_Northeast[1] - Top_Northwest[1]  # Longitude difference
top_height = Top_Northwest[0] - Top_Southwest[0]  # Latitude difference

top_bbox_patch = Rectangle(
    (Top_Southwest[1], Top_Southwest[0]),  # Lower-left corner (lon, lat)
    top_width,
    top_height,
    linewidth=2,
    edgecolor='blue',
    facecolor='blue',
    alpha=0.3
)

# Bottom box rectangle (Destinations)
bottom_width = Bottom_Northeast[1] - Bottom_Northwest[1]  # Longitude difference
bottom_height = Bottom_Northwest[0] - Bottom_Southwest[0]  # Latitude difference

bottom_bbox_patch = Rectangle(
    (Bottom_Southwest[1], Bottom_Southwest[0]),  # Lower-left corner (lon, lat)
    bottom_width,
    bottom_height,
    linewidth=2,
    edgecolor='green',
    facecolor='green',
    alpha=0.3
)
```

```python
# Add the bounding boxes to the map
ax.add_patch(top_bbox_patch)
ax.add_patch(bottom_bbox_patch)

# Add labels for the boxes
# ax.text(-87.76, 41.975, 'ORIGIN AREA\n(Top Box)', ha='center', va='center',
#         bbox=dict(boxstyle='round,pad=0.5', facecolor='lightblue', alpha=0.8),
#         fontsize=11, fontweight='bold')
# ax.text(-87.67, 41.755, 'DESTINATION AREA\n(Bottom Box)', ha='center', va='center
#         bbox=dict(boxstyle='round,pad=0.5', facecolor='lightgreen', alpha=0.8),
#         fontsize=11, fontweight='bold')

# Add title and display
ax.set_title("Chicago Bounding Boxes for Origin and Destination Points", fontsize=1
ax.set_xlabel('Longitude', fontsize=12)
ax.set_ylabel('Latitude', fontsize=12)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



In [5]:
```
# Now we generate random points within these bounding boxes for origins and destina
def get_element_at_point(lat, lon, rain_grid, lats, lons):
        i = np.argmin(np.abs(lats[:, 0] - lat))
        j = np.argmin(np.abs(lons[0, :] - lon))
        return rain_grid[i, j]

rutas = []

origin_points = []
destination_points = []
```

```
avaliable_dataset_names = [
    "20250610",
    "20250620",
    "20250621",
    "20250622",
    "20250623",
    "20250624",
    "20250625",
    "20250626",
    "20250627",
    "20250628",
    "20250629",
    "20250630",
    "20250701",
    "20250702",
    "20250703",
    "20250704",
    "20250705",
    "20250706",
    "20250707",
    "20250708",
    "20250709",
    "20250710",
    "20250711",
    "20250712",
    "20250713",
    "20250714",
    "20250715",
    "20250716",
    "20250717",
    "20250718",
    "20250719",
    "20250720",
    "20250721",
    "20250722",
    "20250723",
    "20250724",
    "20250725",
    "20250726",
    "20250727",
    "20250728",
    "20250729",
    "20250730",
    "20250731",
    "20250801",
    "20250802",
    "20250803",
    "20250804",
    "20250805",
    "20250806",
    "20250807",
    "20250808",
    "20250809",
    "20250810",
    "20250811",
    "20250812",
```

```
    "20250813",
    "20250814",
    "20250815",
    "20250816",
    "20250817",
    "20250818",
    "20250819",
    "20250820",
    "20250821",
    "20250822",
    "20250823",
    "20250824",
    "20250825",
    "20250826",
    "20250827",
    "20250828",
    "20250829",
    "20250830",
    "20250831",
    "20250901",
    "20250902",
    "20250903",
    "20250904",
    "20250905",
    "20250906",
    "20250907",
    "20250908",
    "20250909",
    "20250910",
    "20250911",
    "20250912",
    "20250913",
    "20250914",
    "20250915",
    "20250916",
    "20250917",
    "20250918",
    "20250919",
    "20250920",
    "20250921",
    "20250922",
    "20250923",
    "20250924",
    "20250925",
    "20250926",
    "20250927",
    "20250928",
    "20250929",
    "20250930",
]

avaliable_time_readings = ["t00z", "t12z"]

# We generate 5 random routes and calculate the shortest path for each, then we loa
# Across all datasets
for i in range(100):
```

```
    orig = ox.distance.nearest_nodes(G,
                                     Y=np.random.uniform(Top_Southwest[0], Top_Nort
                                     X=np.random.uniform(Top_Southwest[1], Top_Nort
    dest = ox.distance.nearest_nodes(G,
                                     Y=np.random.uniform(Bottom_Southwest[0], Botto
                                     X=np.random.uniform(Bottom_Southwest[1], Botto
    # ruta = nx.shortest_path(G, orig, dest, weight="travel_time")
    origin_points.append(orig)
    destination_points.append(dest)


for i in range(len(origin_points)):
    ruta = nx.shortest_path(G, origin_points[i], destination_points[i], weight="tra
    rutas.append([ruta, origin_points[i], destination_points[i]])


for dataset_name in avaliable_dataset_names:  # Changed variable name for clarity

    for time_reading in avaliable_time_readings:  # Changed variable name for clari

        base_path = rf'V:\Datos_CLEETS\Data_forecast\{dataset_name}\{time_reading}\

        rain_path = rf'{base_path}\RAIN.nc'
        wind_speed = rf'{base_path}\WSPD10.nc'
        wind_direction = rf'{base_path}\WDIR10.nc'
        heat_index = rf'{base_path}\T2.nc'
        relative_humidity = rf'{base_path}\RH2.nc'

        # Add your processing code here
        print(f"Processing dataset: {dataset_name}, time: {time_reading}")

        # Apply it to the graph

        ds = nc.Dataset(rain_path)
        lats = ds.variables['XLAT'][1, :, :]
        lons = ds.variables['XLONG'][1, :, :]
        RAIN = ds.variables['RAIN'][:]

        ds_wspd = nc.Dataset(wind_speed)
        ds_wdir = nc.Dataset(wind_direction)
        ds_hi = nc.Dataset(heat_index)
        ds_rh = nc.Dataset(relative_humidity)

        time = 1
        for u, v, k, data in G.edges(keys=True, data=True):
            y1, x1 = G.nodes[u]['y'], G.nodes[u]['x']
            y2, x2 = G.nodes[v]['y'], G.nodes[v]['x']
            lat, lon = (y1 + y2) / 2, (x1 + x2) / 2

            rain = ds.variables['RAIN'][time, :, :]  # timestep + zone offset
            rain_mm = get_element_at_point(lat, lon, rain, lats, lons)
            penalty_factor = 1 + rain_mm * 1000  # simple scaling
            data["rain_length"] = data.get("travel_time", 1) * penalty_factor
```

```
            wind_speed = ds_wspd.variables['WSPD10'][time, :, :]
            wind_direction = ds_wdir.variables['WDIR10'][time, :, :]
            wind_spd_at_point = get_element_at_point(lat, lon, wind_speed, lats, lo
            wind_dir_at_point = get_element_at_point(lat, lon, wind_direction, lats
            sin_lut = np.sin(np.radians(np.arange(360)))
            cos_lut = np.cos(np.radians(np.arange(360)))
            crosswind = wind_spd_at_point * sin_lut[int(wind_dir_at_point) % 360]
            headwind = wind_spd_at_point * cos_lut[int(wind_dir_at_point) % 360]
            wind_penalty = 1 + (abs(crosswind) + max(0, headwind)) * 1000  # arbitr
            data['wind_penalty'] = data.get("travel_time", 1) * wind_penalty

            heat_index = ds_hi.variables['T2'][time, :, :]
            heat_at_point = get_element_at_point(lat, lon, heat_index, lats, lons)
            heat_penalty = 1 + heat_at_point * 1000
            data['heat_penalty'] = data.get("travel_time", 1) * heat_penalty

            relative_humidity = ds_rh.variables['RH2'][time, :, :]
            rh_at_point = get_element_at_point(lat, lon, relative_humidity, lats, l
            rh_penalty = 1 + rh_at_point * 1000
            data['rh_penalty'] = data.get("travel_time", 1) * rh_penalty
        print("Graph updated with weather penalties.")
        for i in range(len(origin_points)):
            ruta = nx.shortest_path(G, origin_points[i], destination_points[i], wei
            route_gdf = ox.routing.route_to_gdf(G, ruta)
            route_length = int(route_gdf["length"].sum())
            route_time = int(route_gdf["travel_time"].sum())
            rain_length = int(route_gdf["rain_length"].sum())
            heat_length = int(route_gdf["heat_penalty"].sum())
            wind_length = int(route_gdf["wind_penalty"].sum())
            rh_length = int(route_gdf["rh_penalty"].sum())
            # print(f"Route from {origin_points[i]} to {destination_points[i]}: Len
            rutas.append([ruta, origin_points[i], destination_points[i], route_leng
    time += 1
```

```
Processing dataset: 20250610, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250610, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250620, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250620, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250621, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250621, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250622, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250622, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250623, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250623, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250624, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250624, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250625, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250625, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250626, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250626, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250627, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250627, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250628, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250628, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250629, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250629, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250630, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250630, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250701, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250701, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250702, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250702, time: t12z
Graph updated with weather penalties.
```

```
Processing dataset: 20250703, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250703, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250704, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250704, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250705, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250705, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250706, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250706, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250707, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250707, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250708, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250708, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250709, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250709, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250710, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250710, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250711, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250711, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250712, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250712, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250713, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250713, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250714, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250714, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250715, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250715, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250716, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250716, time: t12z
Graph updated with weather penalties.
```

```
Processing dataset: 20250717, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250717, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250718, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250718, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250719, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250719, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250720, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250720, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250721, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250721, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250722, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250722, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250723, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250723, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250724, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250724, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250725, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250725, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250726, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250726, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250727, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250727, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250728, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250728, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250729, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250729, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250730, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250730, time: t12z
Graph updated with weather penalties.
```

```
Processing dataset: 20250731, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250731, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250801, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250801, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250802, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250802, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250803, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250803, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250804, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250804, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250805, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250805, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250806, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250806, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250807, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250807, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250808, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250808, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250809, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250809, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250810, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250810, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250811, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250811, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250812, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250812, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250813, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250813, time: t12z
Graph updated with weather penalties.
```

```
Processing dataset: 20250814, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250814, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250815, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250815, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250816, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250816, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250817, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250817, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250818, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250818, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250819, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250819, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250820, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250820, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250821, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250821, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250822, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250822, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250823, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250823, time: t12z
Graph updated with weather penalties.
Processing dataset: 20250824, time: t00z
Graph updated with weather penalties.
Processing dataset: 20250824, time: t12z
```

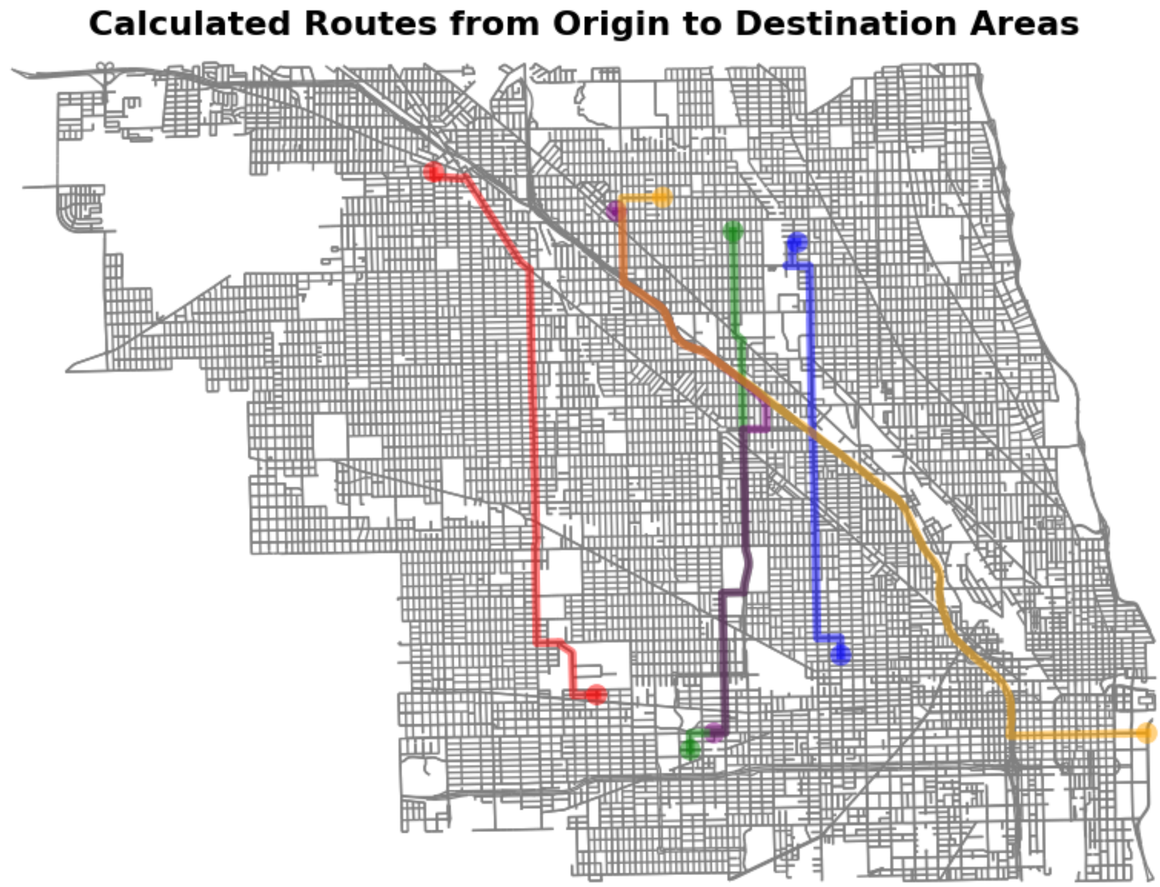
```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[5], line 158
    154 print(f"Processing dataset: {dataset_name}, time: {time_reading}")
    156 # Apply it to the graph
--> 158 ds = nc.Dataset(rain_path)
    159 lats = ds.variables['XLAT'][1, :, :]
    160 lons = ds.variables['XLONG'][1, :, :]

File src\\netCDF4\\_netCDF4.pyx:2521, in netCDF4._netCDF4.Dataset.__init__()

File src\\netCDF4\\_netCDF4.pyx:2158, in netCDF4._netCDF4._ensure_nc_success()

FileNotFoundError: [Errno 2] No such file or directory: 'V:\\Datos_CLEETS\\Data_fore
cast\\20250824\\t12z\\outputs\\RAIN.nc'
```

In [6]:

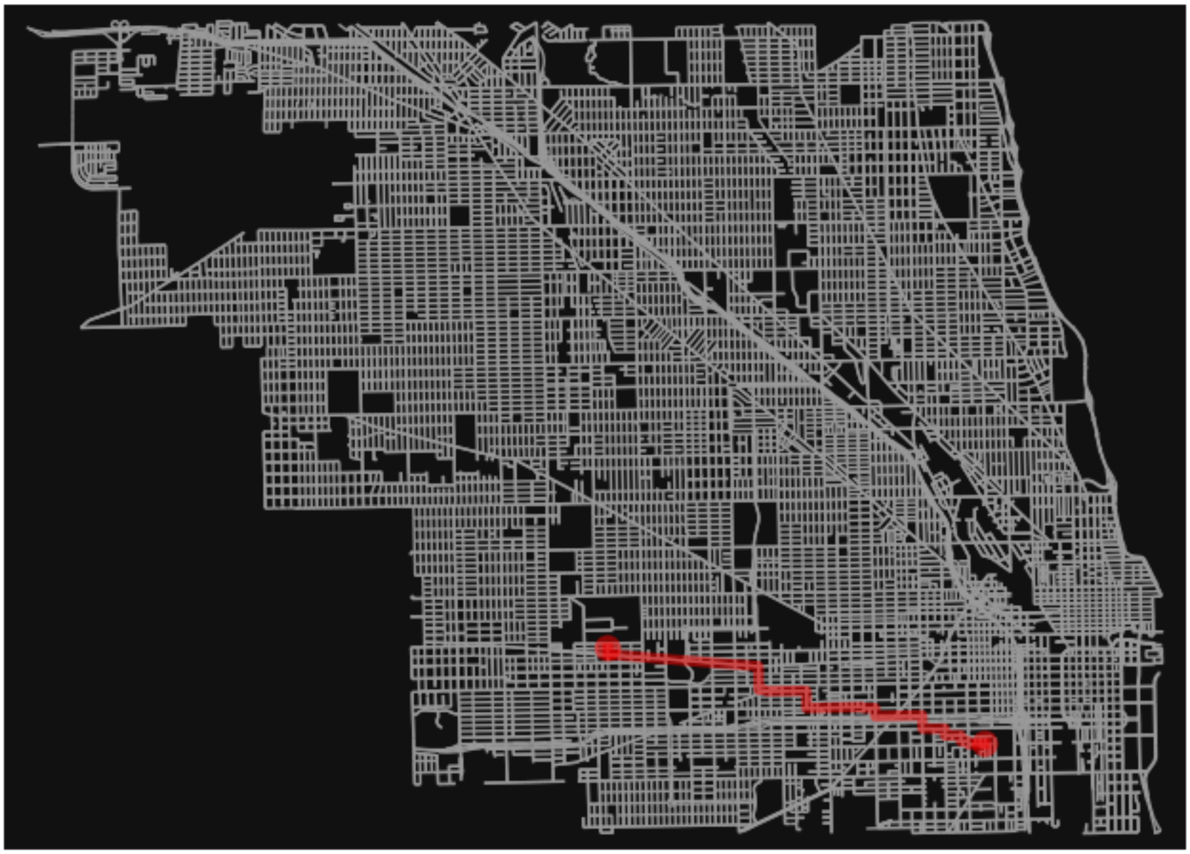
```
# rutas = np.array(rutas)
# np.savetxt(r'C:\Users\elchi\Desktop\UIC_Chicago\Knowledge_graph\Backend\DatasetCr
# np.save(r'C:\Users\elchi\Desktop\UIC_Chicago\Knowledge_graph\Backend\DatasetCreat


with open("rutas3.csv", "w", newline="", encoding="utf-8") as f:
    writer = csv.writer(f)
    for row in rutas:
        # convert nested items like (lat, lon) to strings
        writer.writerow([str(x) for x in row])
```

In [ ]:

```
# Plot the first 5 routes on the map just for visualization

fig, ax = plt.subplots(figsize=(10, 10))


ax.set_xlim(min_lon, max_lon)
ax.set_ylim(min_lat, max_lat)
colors = list(mcolors.TABLEAU_COLORS.keys())


ox.plot_graph(G, ax=ax, node_size=0, edge_color='gray', edge_linewidth=1, show=Fals
ox.plot_graph_routes(
    G,
    routes=[rutas[0][0], rutas[1][0], rutas[2][0], rutas[3][0], rutas[4][0]],
    route_colors=["red", "blue", "green", "purple", "orange"],
    route_linewidth=3,
    node_size=0,
    ax=ax,
    show=False,
    close=False
)
plt.title("Calculated Routes from Origin to Destination Areas", fontsize=16, fontwe
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```



In [ ]:
```
# Allowes you to plot a route from a list of nodes

import osmnx as ox
import networkx as nx
import matplotlib.pyplot as plt

def plot_route_from_nodes(G, node_list):
    """
    Takes a graph G and a list of OSM node IDs (at least 2),
    calculates the route, and plots it.
    """
    if len(node_list) < 2:
        raise ValueError("You need at least two nodes")

    # Build the route by chaining consecutive nodes
    route = []
    for i in range(len(node_list) - 1):
        segment = nx.shortest_path(G, node_list[i], node_list[i + 1], weight='lengt
        route.extend(segment if i == 0 else segment[1:])  # avoid duplicating joint

    # Plot
    fig, ax = ox.plot_graph_route(G, route, route_color='red', route_linewidth=4, n
    plt.show()
    return route

plot_route_from_nodes(G, [256591258, 261252855])
```



```
Out[ ]: [256591258,
 2004538727,
 2004538828,
 1308245851,
 1308245884,
 427144783,
 261218224,
 261126706,
 261126707,
 261126708,
 261126696,
 1309327401,
 1309327400,
 1324341002,
 1308131706,
 5892708079,
 708281409,
 261151968,
 345098142,
 261132814,
 261110014,
 261261490,
 261154210,
 261154211,
 261154212,
 261154216,
 261154221,
 261154225,
 261154231,
 261154237,
 261154242,
 261154246,
 1884474255,
 345098531,
 495783101,
 261271731,
 261141989,
 261164026,
 261229569,
 261154462,
 261154461,
 261282575,
 261143249,
 261220999,
 261221001,
 261221002,
 261123885,
 261221003,
 261200631,
 261176532,
 9924006950,
 9924006955,
 261221005,
 261221007,
 261221008,
 9924006959,
```

```
 261221010,
 733198174,
 9924006965,
 733197991,
 261252855]
```

In [6]:
```
df = pd.read_csv("rutas3.csv")


# print("Columns:", df.columns.tolist())
# print(df.head())
# criteria_cols = ["Route_array","origin_point","destination_point","route_length",
criteria_cols = ["rain_length", "heat_length", "wind_length", "rh_length"]
X = df[criteria_cols].astype(float).values


# ===========================
# 3. Normalize (min-max)
# ===========================
def minmax_norm(X):
    mn = X.min(axis=0)
    mx = X.max(axis=0)
    denom = (mx - mn) + 1e-12  # avoid divide-by-zero
    return (X - mn) / denom
X_norm = minmax_norm(X)
# ===========================
# 4. Entropy Weights
# ===========================
def entropy_weights(X_norm):
    eps = 1e-12
    P = X_norm / (X_norm.sum(axis=0, keepdims=True) + eps)
    n = X_norm.shape[0]
    k = 1.0 / np.log(n)
    P_safe = np.where(P <= 0, eps, P)
    e = -k * (P_safe * np.log(P_safe)).sum(axis=0)
    d = 1 - e
    w = d / (d.sum() + eps)
    return w, e, d

w, e, d = entropy_weights(X_norm)


# ===========================
# 5. Compute combined weight
# ===========================
df["combined_weight_raw"] = X_norm.dot(w)
df["combined_weight"] = (df["combined_weight_raw"] - df["combined_weight_raw"].min(
    df["combined_weight_raw"].max() - df["combined_weight_raw"].min() + 1e-12
)

# ===========================
# 6. Save & Inspect
# ===========================
print("\nEntropy weights:")
for col, weight in zip(criteria_cols, w):
    print(f"{col}: {weight:.5f}")
```

```
# Optional: add normalized columns for debugging
for i, col in enumerate(criteria_cols):
    df[col + "_norm"] = X_norm[:, i]

df.to_csv("weighted_routes.csv", index=False)
print("\nSaved results → weighted_routes.csv")

# Preview
print(df[["route_length", "rain_length", "heat_length", "wind_length", "rh_length",
```

```
Entropy weights:
rain_length: 0.85834
heat_length: 0.02850
wind_length: 0.09648
rh_length: 0.01668

Saved results → weighted_routes.csv
   route_length  rain_length  heat_length  wind_length  rh_length  \
0          9111          696    202760867      3102078   53035664
1         14644          900    262261816      4159704   69429432
2         12500          959    279479346      4526634   74779701
3         12336          806    234810297      3656460   62242913
4         12115          925    269469946      4139191   70250218

   combined_weight
0         0.028335
1         0.049202
2         0.055779
3         0.039569
4         0.050491
```

In [ ]:

```
"""
Run 1:
Entropy weights:
rain_length: 0.72908
heat_length: 0.06024
wind_length: 0.17965
rh_length: 0.03103

Saved results → weighted_routes.csv
   route_length  rain_length  heat_length  wind_length  rh_length  \
0         11008          857    254390804      6188351   66320636
1          8189          636    188695943      4750856   49942574
2         10081          849    251999272      6300381   66342398
3         11384          780    231335065      5768970   60907531
4         15493          838    248765820      6239926   65942680

   combined_weight
0         0.155095
1         0.061070
2         0.154738
3         0.123699
4         0.150646
```

```
Run 2 (rutas 3.csv):

Entropy weights:
rain_length: 0.85834
heat_length: 0.02850
wind_length: 0.09648
rh_length: 0.01668

Saved results → weighted_routes.csv
   route_length  rain_length  heat_length  wind_length  rh_length  \
0          9111          696    202760867      3102078   53035664
1         14644          900    262261816      4159704   69429432
2         12500          959    279479346      4526634   74779701
3         12336          806    234810297      3656460   62242913
4         12115          925    269469946      4139191   70250218

   combined_weight
0         0.028335
1         0.049202
2         0.055779
3         0.039569
4         0.050491


"""
```

Out[ ]:

```
'\n\nEntropy weights:\nrain_length: 0.72908\nheat_length: 0.06024\nwind_length: 0.17965\nrh_length: 0.03103\n\nSaved results → weighted_routes.csv\n   route_length  rain_length  heat_length  wind_length  rh_length  0         11008          857    254390804      6188351   66320636   \n1          8189          636    188695943      4750856   49942574   \n2         10081          849    251999272      6300381   66342398   \n3         11384          780    231335065      5768970   60907531   \n4         15493          838    248765820      6239926   65942680   \n\n   combined_weight  \n0         0.155095  \n1         0.061070  \n2         0.154738  \n3         0.123699  \n4         0.150646  \n\n\n\n\n'
```