Gravity model for synthesizing temporal commuting networks

Gabriel F. Costa

The gravity model is a widely used mathematical framework for modeling human mobility patterns, including commuting networks. It is based on the idea that the probability of interaction between two locations is proportional to the product of their populations and inversely proportional to their distance. This model has proven to be effective in capturing the fundamental features of human mobility, such as the attraction of population centers, the decay of interactions with distance, and the dependence on transportation infrastructure.

In the context of metapopulation network modeling, the gravity model has several important applications. It can be used to generate synthetic mobility networks that capture the spatial and temporal dynamics of human movement, which is essential for understanding the spread of infectious diseases or the effects of transportation policies. Additionally, the gravity model can inform the design of intervention strategies by identifying the key locations that act as "hubs" for mobility flows.

The gravity model is represented by the following equations:

$$p_{ij} = \alpha \frac{P_i^\beta P_j^\beta}{d_{ij}^\gamma}$$

where p_{ij} is the probability of interaction between locations i and j, P_i and P_j are their respective populations, d_{ij} is the distance between them, and α, β, and γ are constants that control the strength of the interaction.

In summary, the gravity model provides a powerful tool for understanding and predicting human mobility patterns, and it has numerous applications in the field of metapopulation network modeling. By accurately capturing the spatial and temporal dynamics of human movement, the gravity model can inform public health policies, transportation planning, and other important decision-making processes.

Implementing the generator:

```python

def generate_synthetic_gravity_network(num_nodes, alpha, beta, T, p_mean, p_std):
    """
    Generates a synthetic temporal mobility network using the gravity model and a normal distribution for the edge
    probabilities.

    :param num_nodes: Number of nodes in the network.
    :param alpha: A constant parameter in the gravity model.
    :param beta: A constant parameter in the gravity model.
    :param T: Number of time steps in the network.
    :param p_mean: Mean value of the normal distribution for edge probabilities.
    :param p_std: Standard deviation of the normal distribution for edge probabilities.
    :return: A 3D numpy array with shape (num_nodes, num_nodes, T), representing the mobility network.
    """
    # Initialize the network
    network = np.zeros((num_nodes, num_nodes, T))

    # Calculate the edge probabilities using the gravity model and a normal distribution
    for i in range(num_nodes):
        for j in range(i + 1, num_nodes):
            p_ij = alpha * (1 / np.sqrt(i + 1)) * (1 / np.sqrt(j + 1)) * np.exp(-beta * np.sqrt((i - j)**2))
            p_ij = np.clip(p_ij, 0, 1)  # Ensure the edge probability is between 0 and 1
            for t in range(T):
                network[i, j, t] = np.random.normal(p_ij, p_std)
                network[j, i, t] = network[i, j, t]

    return network
```

Implementing the plot network function:

```python
def plot_network(figname, network, node_labels=None, time_step=0):
    """
    Plots a network at a given time step.

    :param network: A 3D numpy array with shape (num_nodes, num_nodes, T), representing the mobility network.
    :param node_labels: List of node labels. If None, node indices are used as labels.
    :param time_step: The time step to plot.
    """
    num_nodes = network.shape[0]

    f = plt.figure(figsize=(12, 6), dpi=300)

    # Create a graph object
    G = nx.Graph()

    # Add nodes to the graph
    if node_labels is None:
        G.add_nodes_from(range(num_nodes))
    else:
        node_labels = {i: label for i, label in enumerate(node_labels)}
        G.add_nodes_from(node_labels.items())

    # Add edges to the graph
    for i in range(num_nodes):
        for j in range(i + 1, num_nodes):
            if network[i, j, time_step] > 0:
                G.add_edge(i, j, weight=network[i, j, time_step])

    # Define node positions
    pos = nx.spring_layout(G)

    # Draw nodes and edges
    nx.draw_networkx_nodes(G, pos, node_color='orangered')
    nx.draw_networkx_edges(G, pos, edge_color='coral', style='solid')

    # Draw node labels
    if node_labels is None:
        node_labels = {i: str(i) for i in range(num_nodes)}
    nx.draw_networkx_labels(G, pos, labels=node_labels)

    plt.title('Synthetic Network ' + figname[14], fontsize=18)

    # Save the plot
    return f.savefig(figname)
```