Model Implementation Guide

Project Overview

SIMPLINET is a method implementation for simplifying population flow networks using geographic adjacency constraints and dynamic transmission models. The method improves epidemic simulation accuracy while reducing computational complexity, validated on Shenzhen's 1km grid-level network.

Directory Structure

- `Data/`: Stores raw data files for analysis and modeling
  - Flow matrix data
  - Population data
  - Other basic data required for analysis
- `GISData/`: Contains Geographic Information System (GIS) data files
  - All shapefiles use WGS84 coordinate system
  - Geographic boundary files
  - Raster data
- `Output/`: Stores all analysis and model run outputs
  - Clustering results
  - Flow matrix history
  - Rt calculation results
- `Source/`: Contains all source code files
  - `ConstrainedAgglomerativeClustering.py`: Implements constrained agglomerative hierarchical clustering algorithm
  - `EpidemicModel.py`: Implements epidemic transmission model
  - `MatrixCalculatorAdjacency.py`: Calculates adjacency matrix
  - `MatrixCalculatorSimilarity.py`: Calculates similarity matrix
  - `RtCalculator.py`: Uses multithreading to calculate real-time reproduction number
  - `RtCalculator.R`: R script for calculating Rt values
  - `RunEpidemicSimulation.py`: Main script integrating other modules

Basic Running Steps

1. Calculate Matrices:
    - Run `MatrixCalculatorAdjacency.py` to generate adjacency matrix
    - Run `MatrixCalculatorSimilarity.py` to generate similarity matrix
2. Run Epidemic Simulation:
    - Run `RunEpidemicSimulation.py` to simulate epidemic spread
    - Generate final output files using clustering results and Rt values
3. Calculate Rt Values:
    - Run `RtCalculator.py` to calculate real-time reproduction number for each grid
    - This script will call `RtCalculator.R` for specific calculations
4. Perform Clustering:
    - Run `ConstrainedAgglomerativeClustering.py` to perform hierarchical clustering
    - Based on calculated matrices
    - Update and save flow matrix history

Prerequisites

Data File Requirements

- `Data/population.npy`: One-dimensional array containing population data for all grid units
- `Data/flow.npy`: Flow data, k*k matrix (k=1818), excluding areas with no population or undetected population flow
- `GISData/` directory GIS files (for mapping):
    - Boundary series files (.dbf, .shp, .shx, .cpg): For drawing Shenzhen city boundary
    - Grid-Pop-1km-WGS84 series files (.dbf, .shp, .shx, .prj, .cpg): For drawing 1km grid units

Runtime Environment

Python Requirements

- Python 3.x
- Core Scientific Computing Packages:
  - numpy: Basic numerical computation package
  - pandas: Data processing and analysis
  - scipy: Scientific computing
  - matplotlib: Plotting
- GIS-related Packages:
  - geopandas: Geographic data processing
  - shapely: Geometric object operations

R Requirements

- R 4.x
- Required R Packages:
  - EpiEstim: For calculating transmission index Rt

Figure Reproduction Steps

Figure 2: Full Network Flow

Script: `PlotFullNetwork.py`

Functionality:

- Read and process flow.npy data
- Visualize population flow network at Shenzhen's grid scale
- Display flow intensity and spatial distribution

Parameters:

- `min_flow_plot`: Minimum flow threshold
- `max_flow_plot`: Maximum flow threshold
- `n_lines`: Number of flow lines to draw
- `cmap`: Color mapping scheme
- `output_dir`: Output directory

Running Command:

```
python PlotFullNetwork.py
```

Input Files:

- `./GISData/Grid-Pop-1km-WGS84.shp`: Grid unit shapefile
- `./GISData/Boundary.shp`: Shenzhen boundary shapefile
- `./Data/flow.npy`: Flow matrix data
- `./Data/population.npy`: Population data

Output:

- Generated charts will be saved in `Output/` directory

Running Tips:

- Use default parameter settings
- Expected runtime about 600 seconds
- Avoid setting n_lines too high as it may significantly increase runtime
- Ensure all input file paths are correct and files exist

Figure 3: Epidemic Transmission Simulation

Scripts:

1. `Source/RunEpidemicSimulation.py`: Run SEIR model simulation
2. `PlotEpidemic.py`: Plot time series chart (Figure 3a)
3. `PlotPrevelance-rank.py`: Plot spatial transmission chart (Figure 3b)
4. (Optional) `PlotPrevelance.py`: Plot absolute prevalence distribution

Functionality:

- Simulate SEIR model transmission process
- Generate time series charts (Figure 3a)
- Show spatial distribution at different time points (20, 30, 40, 50 days) (Figure 3b)
- Display infected population proportion dynamics

Running Steps:

1. First run disease transmission simulation:

```
python Source/RunEpidemicSimulation.py
```

This script will:

- Call `EpidemicModel.py` for metapopulation model simulation
- Calculate transmission index Rt for each spatial unit
- Calculate transmission similarity matrix
- Save simulation results to `Output/epidemic_model_results.pkl`

2. Plot time series chart (Figure 3a):

```
python PlotEpidemic.py
```

3. Plot spatial transmission chart (Figure 3b):

```
python PlotPrevelance-rank.py
```

Note:

- Can modify `days` parameter in `plot_normalized_rank` function to select different time slices
- Default shows 20, 30, 40, 50 days spatial distribution

4. (Optional) To view absolute prevalence distribution:

```
python PlotPrevelance.py
```

Input Files:

- `Data/flow.npy`: Flow matrix data
- `Data/population.npy`: Population data

Output Files:

- `Output/epidemic_model_results.pkl`: Disease transmission simulation results
- `Output/rt_values.pkl`: Transmission index Rt calculation results
- `Output/matrix_similarity.npy`: Transmission similarity matrix

Running Tips:

- Ensure disease transmission simulation is completed before plotting
- Simulation process may take considerable time
- To modify simulation parameters, adjust `epidemic_params` in `RunEpidemicSimulation.py`

Figure 4: Clustering Flow Analysis

Scripts:

1. `Source/RunEpidemicSimulation.py`: Run SEIR model simulation
2. `Source/ConstrainedAgglomerativeClustering.py`: Perform constrained clustering
3. `PlotCluster.py`: Plot clustering result distribution (Figure 4a)
4. `PlotFlow.py`: Plot population flow network (Figure 4b)
5. `PlotDistribution.py`: Plot flow volume distribution (Figure 4c)

Functionality:

- Generate clustering result distribution under different simplification rates ρ (Figure 4a):
  - Show clustering distribution
  - Gray markers indicate single-node clusters
- Show population flow network (Figure 4b):
  - Line thickness represents inter-cluster flow volume
- Analyze and display flow volume characteristics (Figure 4c):
  - Show flow frequency distribution
  - Show flow cumulative distribution

Running Steps:

1. First run disease transmission simulation:

```
python Source/RunEpidemicSimulation.py
```

2. Execute constrained clustering algorithm:

```
python Source/ConstrainedAgglomerativeClustering.py
```

This script will:

- Execute constrained clustering algorithm
- Save clustering results to `Output/constrained_clustering_results.npz`

3. Plot clustering result distribution (Figure 4a):

```
python PlotCluster.py
```

4. Plot population flow network (Figure 4b):

```
python PlotFlow.py
```

5. Plot flow volume distribution (Figure 4c):

```
python PlotDistribution.py
```

Parameters:

- `simplification_rates`: Control which simplification rates to plot
- Default values: [0.20, 0.40, 0.80, 0.90, 0.95]
- Can modify this parameter in plotting scripts to show different simplification rates

Input Files:

- `Output/epidemic_model_results.pkl`: Disease transmission simulation results
- `Output/constrained_clustering_results.npz`: Clustering results

Output:

- Generated charts will be saved in `Output/` directory
- Includes clustering distribution, flow network, and flow volume distribution charts

Running Tips:

- Ensure all steps are executed in order
- Clustering process may take considerable time
- To view results for other simplification rates, modify `simplification_rates` parameter

Figure 5: Overall Error Analysis

Scripts:

1. `Source/RunEpidemicSimulation.py`: Run SEIR model simulation
2. `Source/ConstrainedAgglomerativeClustering.py`: Perform constrained clustering
3. `PlotEquivalenceOverall.py`: Plot overall error analysis charts

Functionality:

- Overall infection rate over time (Figure 5a):

    - Show infection rate curves under different simplification rates
    - Include magnified view to show subtle differences
    - Peak value differences manually annotated after statistics
- Cumulative infection rate time series (Figure 5b):
    - Show cumulative infection rates under different simplification rates
    - Include magnified view to show subtle differences
    - Peak value differences manually annotated after statistics
- Simplification rate vs simulation error relationship (Figure 5c):
    - Show how simulation error varies with simplification rate

Running Steps:

1. First run disease transmission simulation:

```
python Source/RunEpidemicSimulation.py
```

2. Execute constrained clustering algorithm:

```
python Source/ConstrainedAgglomerativeClustering.py
```

3. Plot overall error analysis charts:

```
python PlotEquivalenceOverall.py
```

Key functions in this script:

- `plot_curves()`: Plot Figure 5a (overall infection rate curves)
- `plot_curves_cumu()`: Plot Figure 5b (cumulative infection rate curves)
- `plot_dauc()`: Plot Figure 5c (simplification rate-error relationship)

Input Files:

- `Output/epidemic_model_results.pkl`: Disease transmission simulation results
- `Output/constrained_clustering_results.npz`: Clustering results

Output:

- Generated charts will be saved in `Output/` directory:
    - Figure 5a: Overall infection rate curves and magnified view
    - Figure 5b: Cumulative infection rate curves and magnified view
    - Figure 5c: Simplification rate-error relationship chart

Running Tips:

- Ensure disease transmission simulation and clustering analysis are completed
- Pay attention to magnified views in generated charts
- Peak value differences need to be manually added to charts
- Save calculated peak value differences for future reference

Figure 6: Spatial Error Analysis

Scripts:

1. `Source/RunEpidemicSimulation.py`: Run SEIR model simulation
2. `Source/ConstrainedAgglomerativeClustering.py`: Perform constrained clustering
3. `PlotBoxAUC.py`: Plot error box plot (Figure 6a)
4. `PlotSpatialAUC.py`: Plot spatial error distribution (Figure 6b)

Functionality:

- Error box plot (Figure 6a):
    - Show error distribution under different simplification rates
    - Display error statistical characteristics through box plot
- Spatial error distribution (Figure 6b):
    - Show spatial error distribution under different simplification rates
    - Include color bar to show error magnitude

Running Steps:

1. First run disease transmission simulation:

```
python Source/RunEpidemicSimulation.py
```

2. Execute constrained clustering algorithm:

```
python Source/ConstrainedAgglomerativeClustering.py
```

3. Plot error box plot (Figure 6a):

```
python PlotBoxAUC.py
```

4. Plot spatial error distribution (Figure 6b):

```
python PlotSpatialAUC.py
```

Parameters:

1. Key parameters in `PlotSpatialAUC.py`:
    - `rho`: Select simplification rates to plot
        - Default values: [0.2, 0.4, 0.6, 0.8, 0.9]
        - Can modify in `plot_dauc_distribution` function
    - `use_log_scale`: Whether to use logarithmic scale
        - Can set in `plot_dauc_distribution` and `plot_colorbar` functions
2. Parameters in `PlotBoxAUC.py`:
    - `use_log_scale`: Whether to use logarithmic scale for error distribution

Input Files:

- `Output/epidemic_model_results.pkl`: Disease transmission simulation results
- `Output/constrained_clustering_results.npz`: Clustering results

Output:

- Generated charts will be saved in `Output/` directory:
    - Figure 6a: Error box plot under different simplification rates
    - Figure 6b: Spatial error distribution and color bar

Running Tips:

- Ensure disease transmission simulation and clustering analysis are completed
- Adjust `use_log_scale` parameter as needed for best visualization
- To view results for other simplification rates, modify `rho` parameter
- Keep default simplification rate settings for comparison with paper results

Figure 7: Method Comparison

Scripts:

1. `Source/RunEpidemicSimulation.py`: Run SEIR model simulation
2. `CommunityDetection.py`: Execute different community detection methods
3. `PlotScatter.py`: Plot performance comparison scatter plot

Functionality:

- Compare performance of different clustering methods:
    - Leiden method

- Louvain method
- Spectral Clustering
- Affinity Propagation
- Administrative division method

Running Steps:

1. First run disease transmission simulation:

```
python Source/RunEpidemicSimulation.py
```

2. Execute different community detection methods:

```
python CommunityDetection.py
```

Need to run with following configurations:

- Leiden method:

  ```
  detector = CommunityDetectionBaseline(
      method='leiden',
      mobility_matrix=mobility_matrix,
      adjacency_matrix=adjacency_matrix,
      param_range=range(0, 500),
      repeats=10
  )
  ```

- Louvain method:

  ```
  detector = CommunityDetectionBaseline(
      method='louvain',
      mobility_matrix=mobility_matrix,
      adjacency_matrix=adjacency_matrix,
      param_range=np.linspace(1e-4, 200, 50),
      repeats=10
  )
  ```

3. Plot performance comparison scatter plot:

```
python PlotScatter.py
```

This script will:

- Load results from all methods
- Use `add_dataset` to add results from each method
- Use `set_special_point` to add administrative division result

- Generate performance comparison scatter plot

Input Files:

- `Data/flow.npy`: Flow matrix data
- `Output/matrix_adjacency.npy`: Adjacency matrix
- Result files from each method:
    - `scCluster_simplify_ratio.npy` and `scCluster_delta_auc.npy`: SIMPLINET results
    - `scLeiden_simplify_ratio.npy` and `scLeiden_delta_auc.npy`: Leiden results
    - `scLouvian_simplify_ratio.npy` and `scLouvian_delta_auc.npy`: Louvain results

Output:

- Generated scatter plot will be saved as `fig-scatter.png`
- Shows performance of different methods in terms of simplification rate and simulation error
- Administrative division result shown with special marker (red star)

Running Tips:

- Use appropriate parameter ranges for each community detection method
- Leiden method suggested range: 0-500
- Louvain method suggested range: 1e-4-200
- Administrative division result added directly in plotting
- Ensure all method result files are generated before plotting

Figure 8: Sensitivity Analysis

Scripts:

1. `Source/RunEpidemicSimulation.py`: Run SEIR model simulation with different β values
2. `Source/ConstrainedAgglomerativeClustering.py`: Perform constrained clustering
3. `PlotBeta.py`: Plot infection curves under different β values (Figure 8a)
4. `PlotBetaAUC.py`: Plot error analysis under different β values (Figure 8b)

Functionality:

- Analyze effect of different transmission rates β on model:

    - β value range: [0.4, 0.6, 0.8, 1.0, 1.2]
    - Each β value requires complete simulation and clustering process
- Generate infection curves under different transmission rates (Figure 8a):
    - Show infection number changes under different β values
    - Include magnified view to show later changes
- Analyze relationship between simplification rate and normalized error under different β values (Figure 8b):
    - Show how simplification rate affects simulation error under different transmission rates
    - Use logarithmic coordinates to show error changes

Running Steps:

1. Run disease transmission simulation for each β value:

```
python Source/RunEpidemicSimulation.py
```

Parameters to modify:

```
self.epidemic_params = {
    'beta':    0.80,   # Use [0.4, 0.6, 0.8, 1.0, 1.2] in sequence
    'sigma':    0.33,
    'gamma':    0.14,
    'Pm':       0.40,
    'si_mean': 4.00,
    'si_std':   2.50
}
```

2. Execute constrained clustering for each β value's results:

```
python Source/ConstrainedAgglomerativeClustering.py
```

3. Plot infection curves under different β values:

```
python PlotBeta.py
```

This script will:

- Load simulation results for all β values
- Use gradient colors to distinguish different β values
- Generate main plot and magnified view

4. Plot error analysis under different β values:

```
python PlotBetaAUC.py
```

This script will:

- Load clustering results for all β values
- Calculate and display ΔAUC under different β values
- Use logarithmic coordinates to show error changes

Input Files:

- Simulation results for each β value:
    - `I_different_beta.pkl`: Infection curve data for different β values
    - `auc_value_beta=*.xlsx`: AUC calculation results for different β values

Output:

- Generated charts will be saved in `Output/` directory:
    - `Beta.png`: Infection curves under different β values (Figure 8a)
    - `BetaAUC.png`: Error analysis under different β values (Figure 8b)

Running Tips:

- Complete simulation and clustering for each β value in sequence
- Ensure result files are correctly generated for each β value
- Recommend using script to batch process different β values
- Save intermediate results to avoid repeated calculations
- Check color mapping clarity after generating charts