1. User Guide: Max Flow Algorithm Toolkit

This document provides a comprehensive guide to using the scripts in this repository for generating graphs, computing maximum flow, benchmarking algorithms, and visualizing results.

1.1. Repository Layout

The repository contains the following main components:

1.1.1. Source Files

`graph.py` — Core graph data structure and utilities. Defines the `Graph` class that represents a directed graph using an adjacency list and provides methods to load graphs from files, add edges, and perform BFS for finding augmenting paths.

`ford_fulkerson.py` — Standard Ford-Fulkerson algorithm implementation using BFS to find augmenting paths.

`scaling_ford_fulkerson.py` — Capacity scaling variant of Ford-Fulkerson that uses delta-scaling to improve performance on graphs with large capacities.

`preflow_push.py` — Preflow-Push (push-relabel) algorithm implementation using height labels and excess flow.

`mad-flow.py` — Unified command-line interface for running max flow algorithms. Supports all three algorithms and can output results in human-readable or JSON format.

`generate_graphs.py` — Graph generation script for creating test datasets. Generates four types of graphs: Bipartite, FixedDegree, Mesh, and Random. Compiles and runs Java graph generators from the `graphGenerationCode/` directory.

`benchmark.py` — Performance benchmarking tool that runs max flow algorithms multiple times on generated graphs, collects timing statistics, and uses multiprocessing for parallel execution.

`plot_results.py` — Visualization tool for benchmark results. Generates bar charts, comparison line charts, and ratio comparison plots. Requires matplotlib to be installed.

1.1.2. Directory Structure

- `GeneratedGraphs/` — Default output directory for generated graph files (organized by type: Bipartite/, FixedDegree/, Mesh/, Random/)
- `BenchmarkResultsData/` — Default output directory for benchmark results (organized by algorithm and graph type)
- `BenchmarkResultsPlots/` — Default output directory for generated plots (organized by algorithm and graph type, plus Comparisons/)
- `graphGenerationCode/` — Java source code for graph generators (compiled on-the-fly by generate_graphs.py)
- `graphs/` — Sample graph files for testing
- `docs/` — Documentation files

1.2. Running Flow Algorithms on a Single Graph

The `mad-flow.py` script provides a unified interface for running any of the three max flow algorithms on a single graph file.

```
# Run Ford-Fulkerson on a graph (default algorithm)
python3 mad-flow.py -g graphs/Mesh/g1.txt

# Use Scaling Ford-Fulkerson algorithm
python3 mad-flow.py -g graphs/Mesh/g1.txt -a scaling_ford_fulkerson
```

```
# Use Preflow-Push algorithm
python3 mad-flow.py -g graphs/Mesh/g1.txt -a preflow_push

# Get JSON output for scripting
python3 mad-flow.py -g graphs/Mesh/g1.txt -a scaling_ford_fulkerson --json

# Specify custom source and sink nodes
python3 mad-flow.py -g graph.txt -s start -t end -a preflow_push
```

Graph files should contain one edge per line in the format:

```
source_node destination_node capacity
```

Example:

```
s a 10
a b 5
b t 15
```

1.3. Generating Graphs for Benchmarking

The `generate_graphs.py` script creates test graphs for benchmarking experiments. It supports four graph types: Bipartite, FixedDegree, Mesh, and Random.

```
# Generate all default graphs
python3 generate_graphs.py

# Generate first 5 graphs of each type
python3 generate_graphs.py -n 5

# Generate only random graphs with custom density
python3 generate_graphs.py --types random --random-density 10

# Generate graphs to a custom output directory
python3 generate_graphs.py -o GeneratedGraphs2 --types bipartite,mesh
```

Graphs are saved to the output directory (default: `GeneratedGraphs/`) organized by type. Each graph file is named descriptively based on its parameters.

1.4. Benchmarking Flow Algorithms

The `benchmark.py` script runs max flow algorithms multiple times on generated graphs and collects performance statistics.

```
# Benchmark all algorithms on all graph types (recommended)
python3 benchmark.py -i GeneratedGraphs -r 10 --clean

# Benchmark specific algorithm only
python3 benchmark.py -i GeneratedGraphs -a ford_fulkerson -r 10 --clean

# Benchmark multiple specific algorithms
python3 benchmark.py -i GeneratedGraphs -a ford_fulkerson,preflow_push -r 10 --clean

# Benchmark specific graph types only
python3 benchmark.py -i GeneratedGraphs -t bipartite,mesh -r 10 --clean
```

Results are saved in the output directory organized as:

```
BenchmarkResultsData/
  <algorithm>/
```

```
    <graph_type>/
      results.json    # Detailed results with all statistics
      results.csv     # Tabular format for spreadsheet analysis
```

1.5. Plotting Benchmark Results

The `plot_results.py` script generates visualizations from benchmark results. **Note: matplotlib must be installed** (`pip3 install matplotlib`).

```
# Generate all plots including comparisons
python3 plot_results.py --clean

# Generate plots with log scale versions
python3 plot_results.py --clean --log-scale

# Generate only comparison plots
python3 plot_results.py --clean --comparison-only

# Plot specific algorithms and graph types
python3 plot_results.py -a ford_fulkerson,preflow_push -t bipartite,mesh --clean
```

Plots are saved in the output directory organized as:

```
BenchmarkResultsPlots/
  <algorithm>/
    <graph_type>/
      mean_runtime.png    # Bar chart: mean runtime vs input size
      max_runtime.png     # Bar chart: max runtime vs input size
      mean_runtime.svg    # SVG versions of above
      max_runtime.svg
  Comparisons/
    <graph_type>/
      mean_runtime_comparison.png      # Line chart comparing all algorithms
      max_runtime_comparison.png
      ratio_comparison.png             # Ratio comparison plot
      [.svg versions of all above]
```