GTNA
A Framework for the Graph-Theoretic Network Analysis

Benjamin Schiller  Dirk Bradler  Immanuel Schweizer
Max Mühlhäuser  Thorsten Strufe

Graph-Theoretic Network Analyzer
Outline

1. Introduction
2. Modules and Workflow
3. Available Networks
4. Available Metrics
5. Extension
6. Evaluation
7. Summary and Outlook
1. Introduction

Motivation, Our Approach and Requirements

Graph Theoretic Network Analyzer
Motivation

- Importance of complex networks rises

- Need for analysis and evaluation

- Simulation research tool of choice

- BUT: credibility of the results decreases

“... less than 15% of the published MobiHoc papers are repeatable.”

Kurkowski et al. - MANET Simulation Studies: The Incredibles, 2006
Main Objectives

- Many different networks and metrics
- Graph-theoretic analysis of complex networks
- Need for an analysis framework
- Plugin interface for metrics and network generators
Requirements

1. Extensability

2. Run on regular desktop computers

3. Evaluate networks with > 20,000 nodes

4. Require less than 2 GB of memory
2. Modules and Workflow

The four Modules and how they build GTNA
Network Config

- Interface: `gtna.network.Network`

  1. Set of parameters
  2. Constructor
  3. Graph generation

  - E.g. CAN(nodes, dimensions, realities)
Series

- Implementation: *gtna.data.Series*
- Aggregates several simulation runs
  - Instances of the same network config
  - Averages + confidence intervals
- Input
  a. Generated network topology
  b. Imported available traces
Metric

- Interface: `gtna.metrics.Metric`
  1. Sub-metric attributes
  2. Constructor
  3. Computation
  4. Output generation
- Multi-scalar metrics
  - Array of values for each graph
- Single-scalar metrics
  - Single value for each graph
Plotter

- Implementation: *gtna.plot.Plot*
  - Uses *gtna.plot.GNUPlot*

- Plot types
  - Multi, Single, byEdges

- Combination of
  - Multiple metrics
  - Different configuration
  - Different networks
Workflow

Network Config → Metric Config → Plot Config

Series → Metric → Plotter

Graph → Data → Plot

Input

Computation Generation

Output / Input
Instantiating a Network Config

Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };

Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };

Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };
Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[] { can_2_1, can_3_1, can_4_1 };

Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[] { can_2_1, can_2_2, can_2_3 };

Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[] { can_100, can_200, can_300, can_400 };
Instantiating a Network Config

Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };

Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };

Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };

Benjamin Schiller | GTNA → 2. Modules and Workflow → Instantiating a Network Config
Instantiating a Network Config

Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };

Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };

Network can_100 = new CAN(100, 2, 1);
Network can_200 = new CAN(200, 2, 1);
Network can_300 = new CAN(300, 2, 1);
Network can_400 = new CAN(400, 2, 1);
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };
Generating a Series

Network[] can_x_1 =
    new Network[]{ can_2_1, can_3_1, can_4_1 };
Network[] can_2_x =
    new Network[]{ can_2_1, can_2_2, can_2_3 };
Network[] can_x =
    new Network[]{ can_100, can_200, can_300, can_400 };

Series[] s_can_x_1 = Series.generate(can_x_1, 20);
Series[] s_can_2_x = Series.generate(can_2_x, 20);
Series[] s_can_x = Series.generate(can_x, 20);
Generating a Series

```java
Network[] can_x_1 =
    new Network[] { can_2_1, can_3_1, can_4_1 };
Network[] can_2_x =
    new Network[] { can_2_1, can_2_2, can_2_3 };
Network[] can_x =
    new Network[] { can_100, can_200, can_300, can_400 };

Series[] s_can_x_1 = Series.generate(can_x_1, 20);
Series[] s_can_2_x = Series.generate(can_2_x, 20);
Series[] s_can_x = Series.generate(can_x, 20);
```
Generating a Series

Network[] can_x_1 =
    new Network[] { can_2_1, can_3_1, can_4_1 };
Network[] can_2_x =
    new Network[] { can_2_1, can_2_2, can_2_3 };
Network[] can_x =
    new Network[] { can_100, can_200, can_300, can_400 };

Series[] s_can_x_1 = Series.generate(can_x_1, 20);
Series[] s_can_2_x = Series.generate(can_2_x, 20);
Series[] s_can_x = Series.generate(can_x, 20);
Plotting the Results

```java
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x
};

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```
Plotting the Results

```java
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x
};

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```
Plotting the Results

```java
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");
Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x
};

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```
Plotting the Results

![Graph showing EXP(h) vs Height h for different CAN - 100 configurations (d = 2, r = 1), (d = 3, r = 1), and (d = 4, r = 1).]
Plotting the Results

Expansion

CAN - 100 (d = 2, r = 1)
CAN - 100 (d = 2, r = 2)
CAN - 100 (d = 2, r = 3)

Height h

EXP(h)

0 0.2 0.4 0.6 0.8 1.0 1.2

0 1 2 3 4 5 6 7 8 9

0 0.2 0.4 0.6 0.8 1.0

Benjamin Schiller  | GTNA → 2. Modules and Workflow → Plotting the Results
Plotting the Results

Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x
};

Plot.allSingleByEdges(s_all, "./can-by-edges/");
Plotting the Results

Characteristic Path Length

Dimensions

CAN - 100 (d = x, r = 1)

Characteristic Path Length

Nodes

Realities

Benjamin Schiller | GTNA → 2. Modules and Workflow → Plotting the Results
Plotting the Results

Characteristic Path Length

CAN - 100 (d = 2, r = x)
Plotting the Results

Characteristic Path Length

Nodes

Characteristics Path Length

CAN - x (d = 2, r = 1)

Benjamin Schiller | GTNA → 2. Modules and Workflow → Plotting the Results
Plotting the Results

```java
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

Plot.allMulti(s_can_x_1, "./can-dimensions/");
Plot.allMulti(s_can_2_x, "./can-realities/");

Plot.allSingle(s_can_x_1, "./can-dimensions-singles/");
Plot.allSingle(s_can_2_x, "./can-realities-singles/");
Plot.allSingle(s_can_x, "./can-nodes-singles/");

Series[][] s_all = new Series[][]{
    s_can_x_1, s_can_2_x
};

Plot.allSingleByEdges(s_all, "./can-by-edges/");
```
Plotting the Results

Characteristic Path Length

CAN - 100 (d = x, r = 1)
CAN - 100 (d = 2, r = x)
3. Available Networks

Networks already implemented in GTNA
Canonical Networks

- Complete
- Ring
- Star
Network Models

- Barabasi Albert
- De Bruijn
- Erdos Renyi
- Gilbert
- GNC - Growing Network with Copying
- GNR - Growing Network with Re-direction
- Kleinberg
- UnitDisc
- Watts Strogatz
Peer-to-Peer Networks

- CAN
- Chord
- Gnutella 0.4
- Gnutella 0.6
- Kademlia
- ODRI - Optimal Diameter Routing Infrastructure
- Pastry
- PathFinder
- Symphony
# Misc Networks

- **Utilities**
  - Readable File
  - Readable List

- **Transformations**
  - Add Edges Per Node
  - Add Edges Randomly
  - Rewire
  - Same Degree Distribution

1. own format
2. edges only
3. GML
4. caida.org Internet traces
4. Available Metrics

Metrics already implemented in GTNA
Degree Distribution

- (in-/out-)Degree Distribution
- Empirical Complementary (in-/out-)Degree Distribution
- #Nodes, #Edges
- Average Degree
- Min (in-/out-)Degree
- Max (in-/out-)Degree
Shortest Path Length

- Shortest Path Length Distribution
- Expansion
- Local Characteristic Path Length
- Characteristic Path Length
- Diameter
Routing Length

- Routing Length Distribution
- Routing Length Complementary Distribution
- Local Characteristic Routing Length

- Characteristic Routing Length
- Max Routing Length
Network Fragmentation

- Max Cluster Size
- Number of Clusters
- Average Isolated Cluster Size
- Point of Rupture
- Average Number of Clusters
- Max Number of Clusters
Clustering Coefficient

- Local Clustering Coefficient
- Clustering Coefficient
- Weighted Clustering Coefficient
Rich Club Connectivity

- Rich Club Connectivity

Benjamin Schiller | GTNA → 4. Available Metrics → Rich Club Connectivity
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;
    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }
    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[0], nodes[i]);
            if(this.p1)
                edges.add(nodes[i], nodes[0]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}

NOD_CLASS = gtna.metrics.NodesOfDegree
NOD_NAME = Nodes of Degree
NOD_DATA_KEYS = NOD_NOD
NOD_SINGLES_KEYS = NOD_DOMN
NOD_DATA_PLOTS = NOD
NOD_SINGLES_PLOTS = DOMN
NOD_NOD_DATA_NAME = Nodes of Degree
NOD_NOD_DATA_FILENAME = nod
NOD_DOMN_SINGLE_NAME = Degree of Most Nodes
NOD_PLOT_DATA = NOD_NOD
NOD_PLOT_FILENAME = nod
NOD_PLOT_TITLE = Nodes of Degree
NOD_PLOT_X = degree d
NOD_PLOT_Y = # nodes of degree d
DOMN_PLOT_DATA = NOD_DOMN
DOMN_PLOT_FILENAME = domn
DOMN_PLOT_TITLE = Degree of Most Nodes
DOMN_PLOT_X = Degree of Most Nodes
DOMN_PLOT_Y = Degree of Most Nodes
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;
    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }
    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[i], nodes[0]);
            if(this.p1)
                edges.add(nodes[0], nodes[i]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;
    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }
    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[i], nodes[0]);
            if(this.p1)
                edges.add(nodes[0], nodes[i]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}

Implementing a new Network

Benjamin Schiller  |  GTNA → 5. Extension → Implementing a new Network
Implementing a new Network

```java
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;
    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }
    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[i], nodes[0]);
            if(this.p1)
                edges.add(nodes[0], nodes[i]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}
```
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;

    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }

    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[i], nodes[0]);
            if(this.p1)
                edges.add(nodes[0], nodes[i]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}

Parameter
Constructor
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;
    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }
    public Graph generate(){
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++){
            edges.add(nodes[i], nodes[0]);
            if(this.p1)
                edges.add(nodes[0], nodes[i]);
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}
Configuring a new Network

CAN_NETWORK_NAME = CAN

CAN_NETWORK_FOLDER = can

CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d

CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
Configuring a new Network

CAN_NETWORK_NAME = CAN
CAN_NETWORK_FOLDER = can
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
Configuring a new Network

![Graph showing EXP(h) vs. Height h]

- **EXP(h)**
- **Height h**

**CAN - 100 (d = 2, r = 1)**
- **CAN - 100 (d = 3, r = 1)**
- **CAN - 100 (d = 4, r = 1)**

**Characteristic Path Length**

**Dimensions**
Configuring a new Network

CAN_NETWORK_NAME = CAN
CAN_NETWORK_FOLDER = can
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
Configuring a new Network
Configuring a new Network

CAN_NETWORK_NAME = CAN
CAN_NETWORK_FOLDER = can
CAN_NETWORK_DIMENSIONS_NAME = Dimensions
CAN_NETWORK_DIMENSIONS_NAME_LONG = dimensions
CAN_NETWORK_DIMENSIONS_NAME_SHORT = d
CAN_NETWORK_REALITIES_NAME = Realities
CAN_NETWORK_REALITIES_NAME_LONG = realities
CAN_NETWORK_REALITIES_NAME_SHORT = r
Configuring a new Network

![Graph showing characteristic path length vs. dimensions for different network configurations]

- CAN - 100 (d = x, r = 1)

Characteristic Path Length

- Dimensions
- Characteristic Path Length

Graph legend:
- CAN - 100 (d = x, r = 1)
Implementing a new Metric

```java
public class MyMetric extends MetricImpl implements Metric {
    private int[] nod;
    private int domn;
    public MyMetric() { super("NOD"); }
    public void computeData(Graph g) {
        this.nod = new int[g.maxDegree + 1];
        for (int i = 0; i < g.nodes.length; i++)
            this.nod[g.nodes[i].degree]++;
        this.domn = Util.maxIndex(this.nod);
    }
    public Value[] getValues(Value[] values) {
        return new Value[] { new Value("NOD_DOMN", this.domn) };
    }
    public boolean writeData(String folder) {
        DataWriter.write("NOD_NOD", folder, this.nod);
        return true;
    }
}
```
public class MyMetric extends MetricImpl implements Metric {
    private int[] nod;
    private int domn;
    public MyMetric() { super("NOD"); }
    public void computeData(Graph g) {
        this.nod = new int[g.maxDegree + 1];
        for (int i = 0; i < g.nodes.length; i++)
            this.nod[g.nodes[i].degree]++;
        this.domn = Util.maxIndex(this.nod);
    }
    public Value[] getValues(Value[] values) {
        return new Value[] { new Value("NOD_DOMN", this.domn) };
    }
    public boolean writeData(String folder) {
        DataWriter.write("NOD_NOD", folder, this.nod);
        return true;
    }
}
public class MyMetric extends MetricImpl implements Metric {
    private int[] nod;
    private int domn;
    public MyMetric() { super("NOD"); }
    public void computeData(Graph g) {
        this.nod = new int[g.maxDegree + 1];
        for (int i = 0; i < g.nodes.length; i++)
            this.nod[g.nodes[i].degree]++;
        this.domn = Util.maxIndex(this.nod);
    }
    public Value[] getValues(Value[] values) {
        return new Value[] { new Value("NOD_DOMN", this.domn) };
    }
    public boolean writeData(String folder) {
        DataWriter.write("NOD_NOD", folder, this.nod);
        return true;
    }
}
public class MyMetric extends MetricImpl implements Metric {
    private int[] nod;
    private int domn;
    
    public MyMetric() { super("NOD"); }

    public void computeData(Graph g) {
        this.nod = new int[g.maxDegree + 1];
        for (int i = 0; i < g.nodes.length; i++)
            this.nod[g.nodes[i].degree]++;
        this.domn = Util.maxIndex(this.nod);
    }

    public Value[] getValues(Value[] values) {
        return new Value[] { new Value("NOD_DOMN", this.domn) };
    }

    public boolean writeData(String folder) {
        DataWriter.write("NOD_NOD", folder, this.nod);
        return true;
    }
}
public class MyMetric extends MetricImpl implements Metric {
    private int[] nod;
    private int domn;
    public MyMetric() { super("NOD"); }
    public void computeData(Graph g) {
        this.nod = new int[g.maxDegree + 1];
        for (int i = 0; i < g.nodes.length; i++)
            this.nod[g.nodes[i].degree]++;
        this.domn = Util.maxIndex(this.nod);
    }
    public Value[] getValues(Value[] values) {
        return new Value[] { new Value("NOD_DOMN", this.domn) };
    }
    public boolean writeData(String folder) {
        DataWriter.write("NOD_NOD", folder, this.nod);
        return true;
    }
}
Implementing a new Metric

```java
public class MyMetric extends MetricImpl implements Metric {
    private int[] nod;
    private int domn;
    public MyMetric() { super("NOD"); }
    public void computeData(Graph g) {
        this.nod = new int[g.maxDegree + 1];
        for (int i = 0; i < g.nodes.length; i++)
            this.nod[g.nodes[i].degree]++;
        this.domn = Util.maxIndex(this.nod);
    }
    public Value[] getValues(Value[] values) {
        return new Value[] { new Value("NOD_DOMN", this.domn) };
    }
    public boolean writeData(String folder) {
        DataWriter.write("NOD_NOD", folder, this.nod);
        return true;
    }
}
```
Configuring a new Metric

```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length

RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```

Routing Length Distribution vs. Shortest Path Length Distribution
Configuring a new Metric

```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLE_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLE_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```
Configuring a new Metric

RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length

**RL_SINGLES_KEYS = CRL**
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

**CRL_SINGLES_NAME = Characteristic Routing Length**

RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
Configuring a new Metric

RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA.Keys = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
Configuring a new Metric

RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
Configuring a new Metric

Characteristic Routing Length

Routing Length Distribution vs. Shortest Path Length Distribution

CAN - 100 (d = x, r = 1)
Configuring a new Metric

RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length
RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
Configuring a new Metric
Configuring a new Metric

```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length

RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD
RLD_PLOT_FILENAME = rld
RLD_PLOT_TITLE = Routing Length Distribution
RLD_PLOT_X = Routing Length rl
RLD_PLOT_Y = P(rl)
```
Configuring a new Metric

```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length

RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD, SPLD
RLD_PLOT_FILENAME = rld-vs-spld
RLD_PLOT_TITLE = RLD vs. SPLD
RLD_PLOT_X = Routing Length rl / Shortest Path Length l
RLD_PLOT_Y = P(rl) / R(l)
```
Configuring a new Metric

Routing Length Distribution vs. Shortest Path Length Distribution

- RLD - CAN - 100 (d = 2, r = 1)
- SPLD - CAN - 100 (d = 2, r = 1)
- RLD - CAN - 100 (d = 3, r = 1)
- SPLD - CAN - 100 (d = 3, r = 1)
- RLD - CAN - 100 (d = 4, r = 1)
- SPLD - CAN - 100 (d = 4, r = 1)
Configuring a new Metric

RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

CRL_SINGLE_NAME = Characteristic Routing Length

RLD_DATA_NAME = Routing Length Distribution
RLD_DATA_FILENAME = RLD

CRL_PLOT_DATA = CRL
CRL_PLOT_FILENAME = crl
CRL_PLOT_TITLE = Characteristic Routing Length
CRL_PLOT_Y = CRL

RLD_PLOT_DATA = RLD, SPLD
RLD_PLOT_FILENAME = rld-vs-spld
RLD_PLOT_TITLE = Routing Length Distribution vs. Shortest Path Length Distribution
RLD_PLOT_X = Routing Length \( r_l \) / Shortest Path Length \( l \)
RLD_PLOT_Y = \( P(r_l) / R(l) \)
6. Evaluation

Runtime and Memory Usage Evaluation

**Memory usage**

- Erdos Renyi - x (averageDegree = 20.0, bidirectional = true)
- Erdos Renyi - x (averageDegree = 30.0, bidirectional = true)
- Erdos Renyi - x (averageDegree = 40.0, bidirectional = true)
- Erdos Renyi - x (averageDegree = 50.0, bidirectional = true)

**Runtime: Shortest Path Length**

- Erdos Renyi - x (averageDegree = 20.0, bidirectional = true)
- Erdos Renyi - x (averageDegree = 30.0, bidirectional = true)
- Erdos Renyi - x (averageDegree = 40.0, bidirectional = true)
- Erdos Renyi - x (averageDegree = 50.0, bidirectional = true)
Setup

- Computer
  - 2.4 GHs Intel Core 2 Duo, 2 GB of Memory
  - MAC OS 10.5.8, Java version 1.5.0_22
- Networks
  - Erdös Rényi random graph
  - $d_{avg} \in \{20, 30, 40, 50\}$
- Runtime: Shortest Path Length
- Memory usage: all metrics
Runtime

Runtime: Shortest Path Length

Erdos Renyi - x (averageDegree = 20.0, bidirectional = true)
Erdos Renyi - x (averageDegree = 30.0, bidirectional = true)
Erdos Renyi - x (averageDegree = 40.0, bidirectional = true)
Erdos Renyi - x (averageDegree = 50.0, bidirectional = true)
Memory Usage

Memory usage

Erdos Renyi - x (averageDegree = 20.0, bidirectional = true) +
Erdos Renyi - x (averageDegree = 30.0, bidirectional = true) x
Erdos Renyi - x (averageDegree = 40.0, bidirectional = true) *
Erdos Renyi - x (averageDegree = 50.0, bidirectional = true) □

Memory usage (in MB)

Nodes
Requirements

1. Extensibility  ✔
2. Run on regular desktop computers  ✔
3. Evaluate networks with > 20,000 nodes  ✔
4. Require less than 2 GB of memory  ✔
7. Summary and Outlook

Summary and Planned Extensions to GTNA

```java
public class MyNetwork extends NetworkImpl implements Network {
    private boolean p1;

    public MyNetwork(int nodes, boolean p1) {
        super("MY_NETWORK", nodes, new String[]{"P1"}, new String[]{"" + p1});
        this.p1 = p1;
    }

    public Graph generate() {
        Timer timer = new Timer();
        Node[] nodes = Node.init(this.nodes());
        Edges edges = new Edges(nodes, 100);
        for(int i=1; i<nodes.length; i++) {
            edges.add(nodes[0], nodes[i]);
            if(this.p1)
                edges.add(nodes[i], nodes[0]);
            for(int i=1; i<nodes.length; i++) {
                edges.addEdge(nodes[i], nodes[0]);
            }
        }
        edges.fill();
        return new Graph(this.description(), nodes, timer);
    }
}
```
Summary

- Graph-theoretic analysis helps understand complex networks
- Often difficult task
  - adapting specific algorithms to special network formats
- GTNA - Graph-Theoretic Network Analyzer
  - Java-based framework
  - Easily extendable
  - Many network generators and metric implementations exist
Outlook

- Implementing additional metrics
- Providing Internet topology generators
- Analysis of network dynamics
- Graphical user interface
GTNA
A Framework for the Graph-Theoretic Network Analysis

Benjamin Schiller    Dirk Bradler    Immanuel Schweizer
Max Mühlhäuser    Thorsten Strufe

Graph-Theoretic Network Analyzer