GTNA

A Framework for the Graph-Theoretic Network Analysis

Benjamin Schiller Dirk Bradler Immanuel Schweizer
Max Mühlhäuser Thorsten Strufe

TECHNISCHE UNIVERSITÄT DARMSTADT

GTNA

Graph-Theoretic Network Analyzer

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

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

1. Introduction

Motivation, Our Approach and Requirements

TECHNISCHE UNIVERSITÄT DARMSTADT

GTNA

Graph-Theoretic Network Analyzer

Motivation

networks like
Internet, P2P, WSN, MobiAdHoc
become omnipresent in our daily lifes

- Importance of complex networks rises
- Need for analysis and evaluation
- Simulation research tool of choice
- BUT: credibility of the results decreases

different simulation tools exist, BUT
- comparison hardly possible
- results differ significantly
due to the complexity of nw sim

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

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

survey simulation studies of MobiHoc
(= ACM international Symp. on Mobile
Ad Hoc Networking and Computing)

Main Objectives

to overcome these difficulties

TECHNISCHE UNIVERSITÄT DARMSTADT

- Graph-theoretic analysis of network snapshots

don't capture dynamics
BUT: allow for in-dept structural comparison

- Many different networks and metrics

many different implementations exist
mostly designed for a special group of metrics / topologies

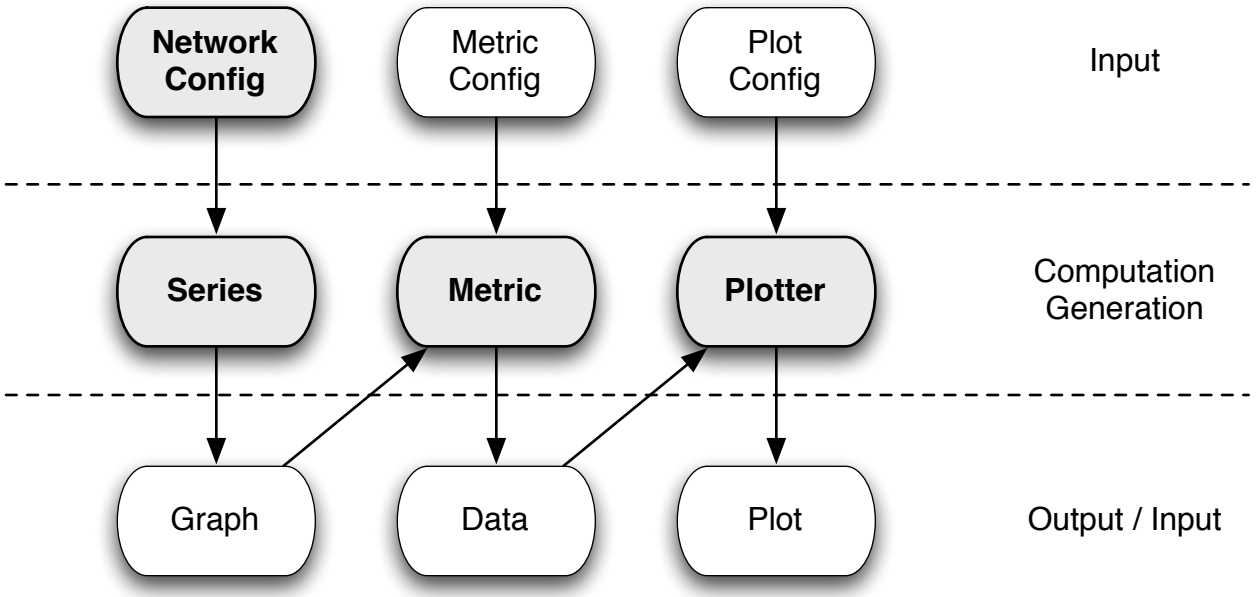
- Need for an analysis framework

=> need for a comprehensive, extendible tool for the concise evaluation of graphs

- Plugin interface
  - Snapshots
  - Topology generators
  - Metrics

2. Modules and Workflow

The four Modules and how they build GTNA



Network Config

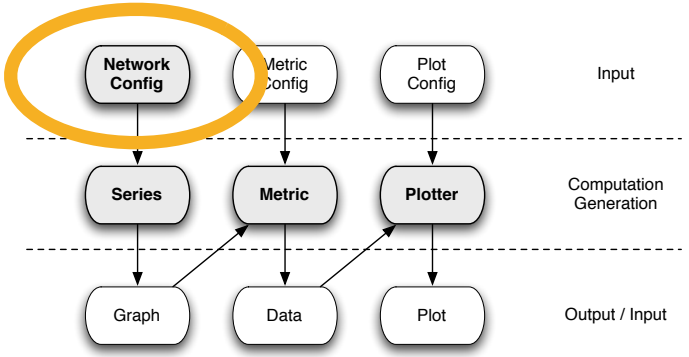
- Interface: *gtna.network.Network*

A. *Import snapshots from any source*

   1. *Snapshot in readable format*

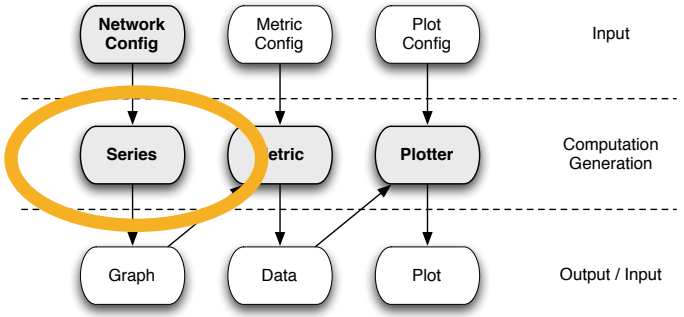
B. *Generate snapshot inside of GTNA*

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



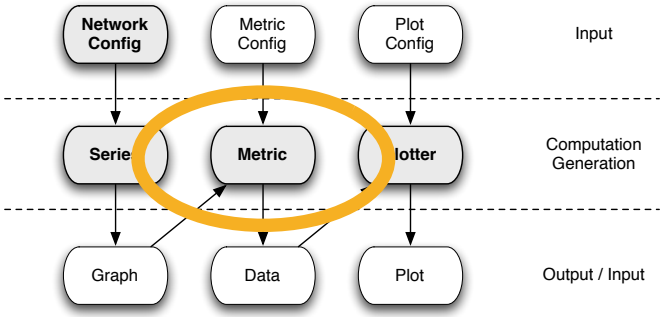
Series

- Implementation: *gtna.data.Series*
- Input
  - Network config
- Computes
  - Metrics for multiple network instances
- Aggregates data
  - Averages + confidence intervals



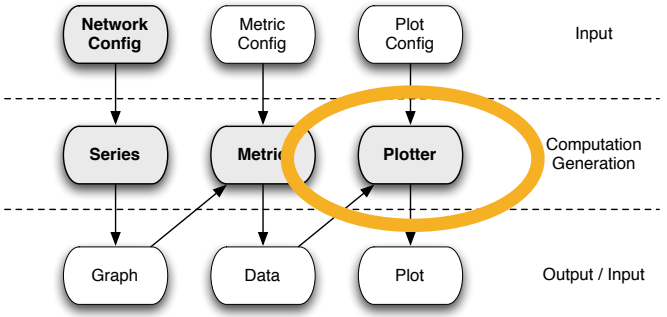
Metric

- Interface: *gtna.metrics.Metric*
- Types
  1. Multi-scalar metric
  2. Single-scalar metric
- Implementation
  1. Sub-metric attributes
  2. Constructor
  3. Computation
  4. Output generation

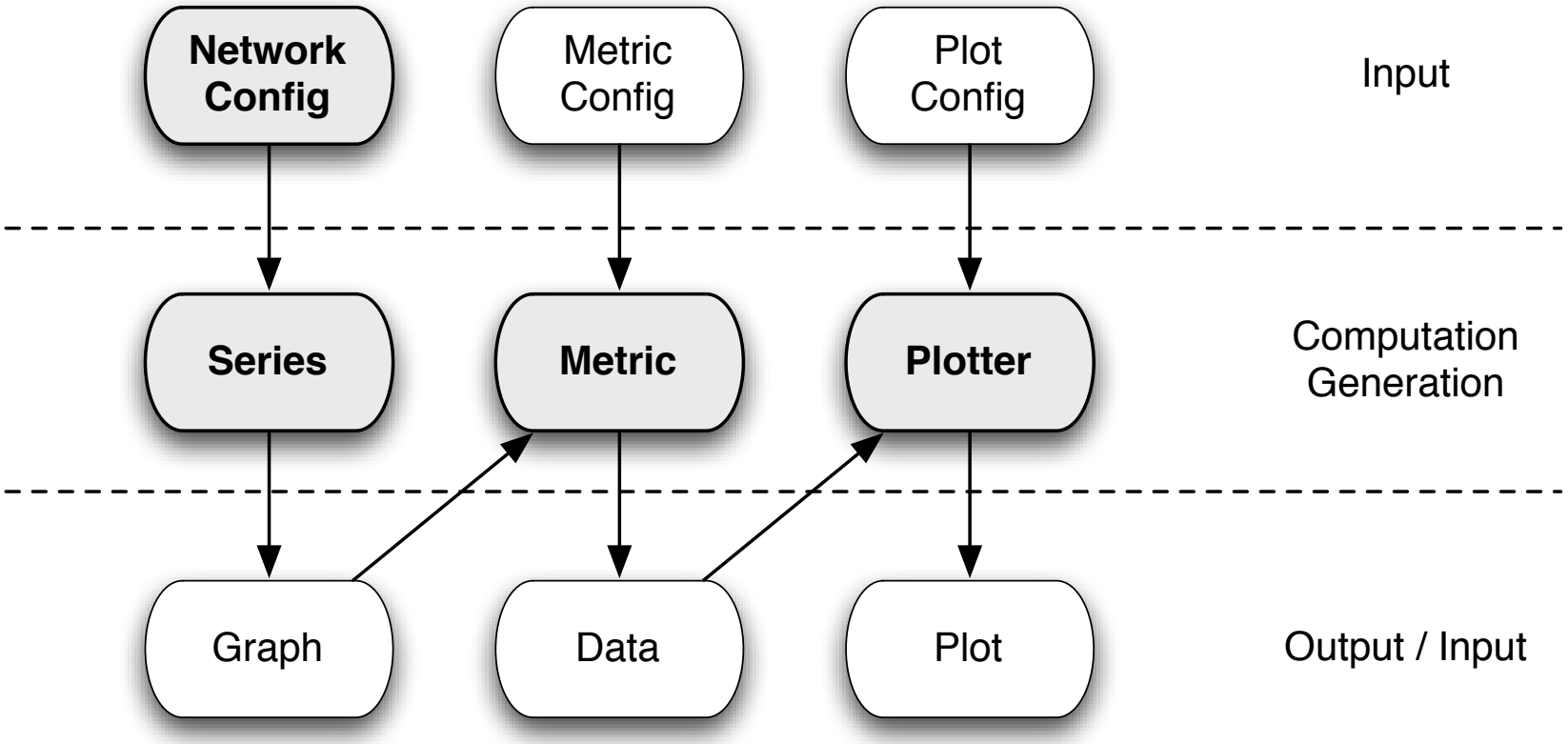


Plotter

- Implementation: *gtna.plot.Plot*
- Combinations of
  1. Multiple metrics
  2. Different configuration
  3. Different networks
- Types
  1. Multi-scalar metrics
  2. Single-scalar metrics
  3. Single-scalar metrics by edges



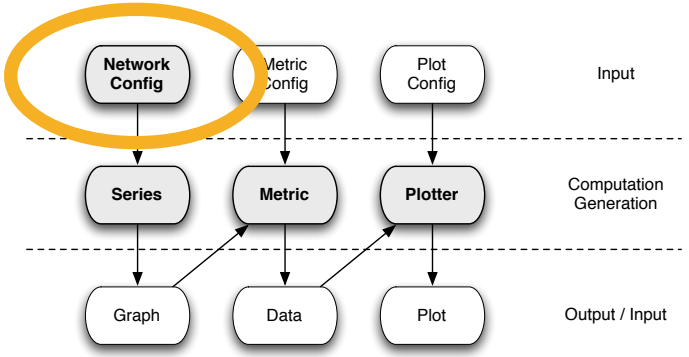
Workflow


Network Config
Metric Config
Plot Config
Input
Series
Metric
Plotter
Computation Generation
Graph
Data
Plot
Output / Input

Instantiating a Network Config

TECHNISCHE UNIVERSITÄT DARMSTADT

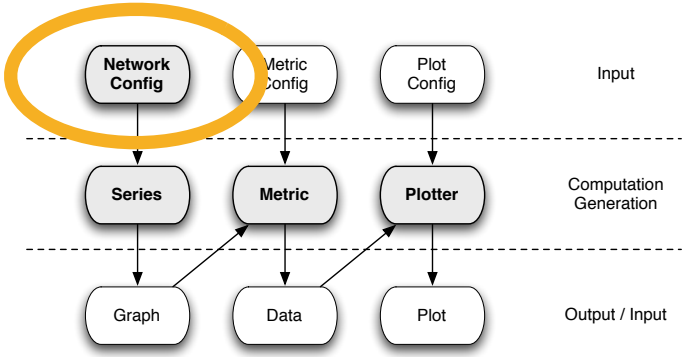
```
Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
  new Network[]{ can_2_1, can_3_1, can_4_1 };

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

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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 };
```



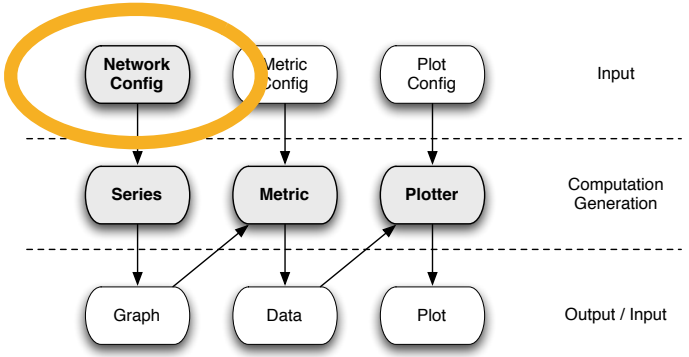
Instantiating a Network Config

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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 };
```

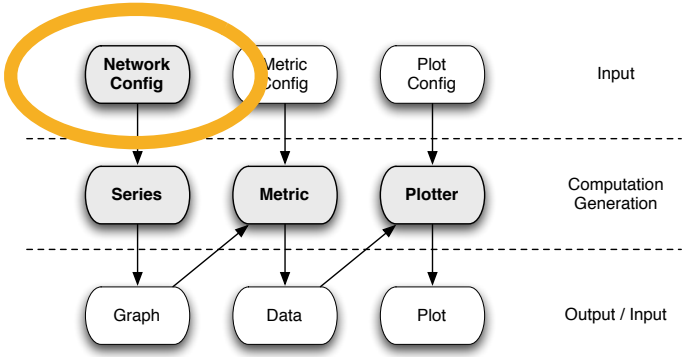
```
Network can_2_1 = new CAN(100, 2, 1);
Network can_2_2 = new CAN(100, 2, 2);
Network can_2_3 = new CAN(100, 2, 3);
Network[] can_2_x =
  new Network[]{ can_2_1, can_2_2, can_2_3 };
```

```
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

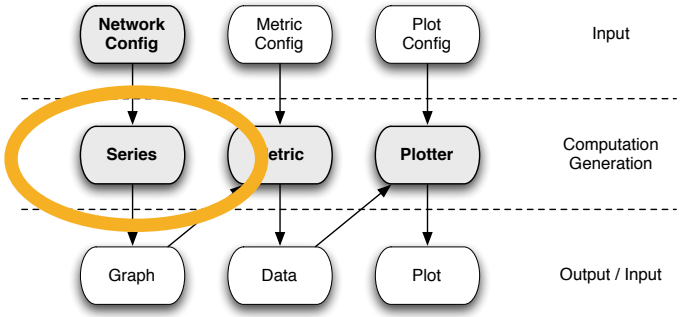
```
Network can_2_1 = new CAN(100, 2, 1);
Network can_3_1 = new CAN(100, 3, 1);
Network can_4_1 = new CAN(100, 4, 1);
Network[] can_x_1 =
  new Network[]{ can_2_1, can_3_1, can_4_1 };

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 };
```



Generating a Series

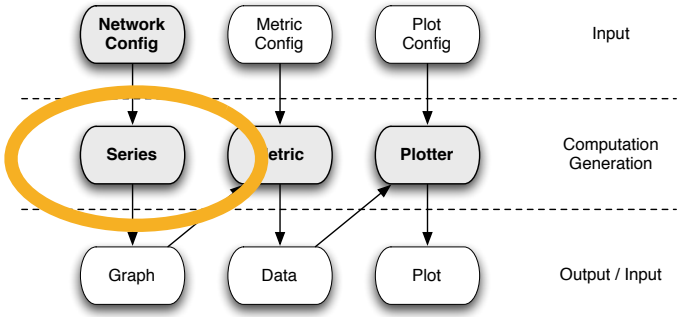
TECHNISCHE UNIVERSITÄT DARMSTADT

```
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);
```



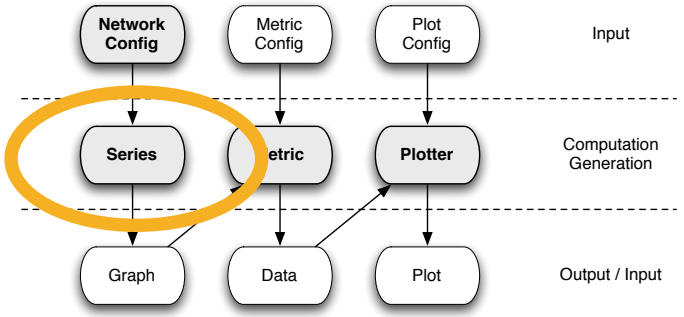
Generating a Series

```
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);
```

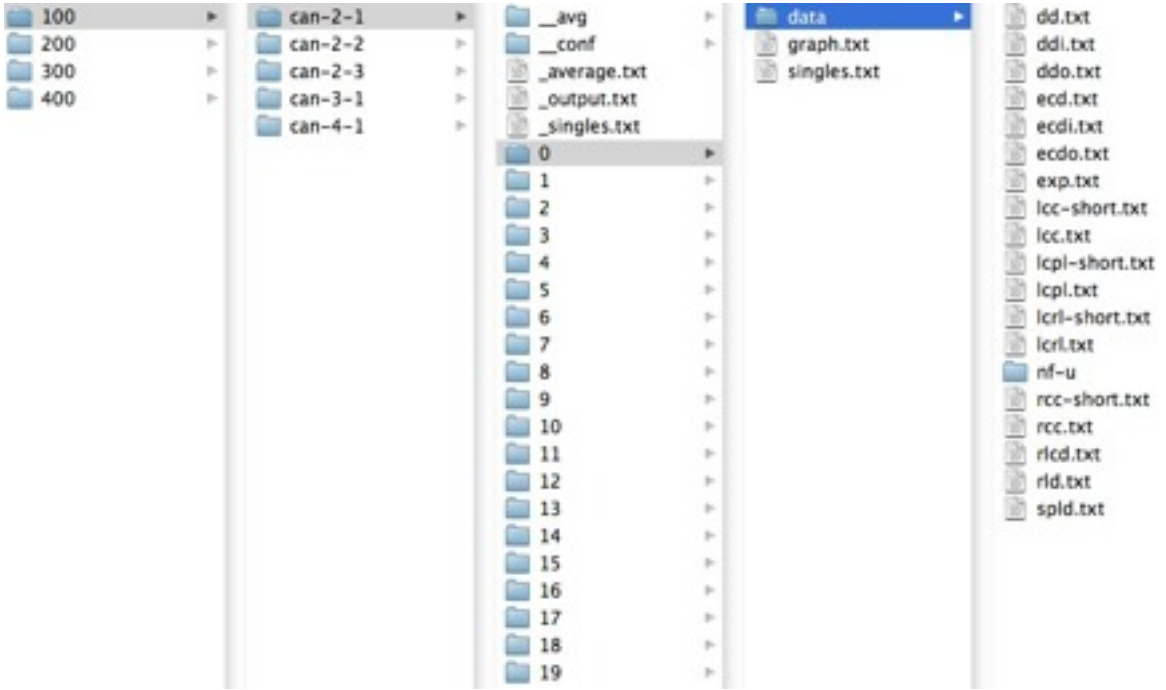


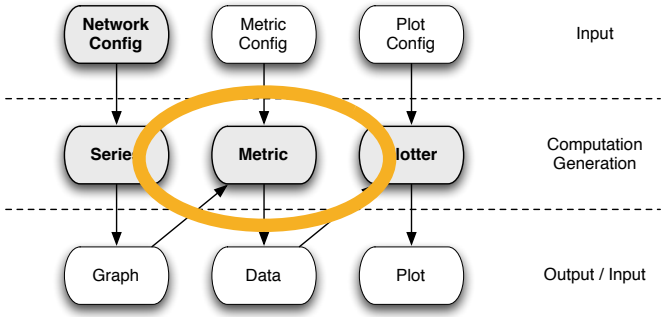
Generating a Series

```
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);
```



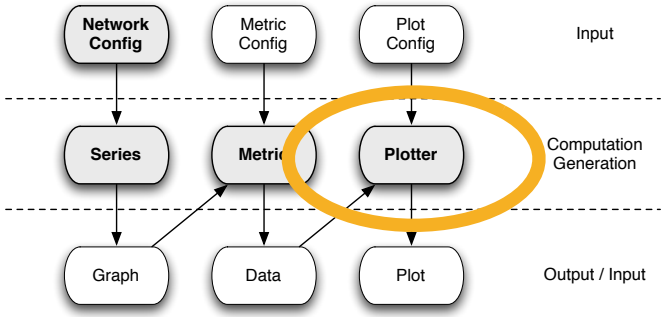
Data Generation





Plotting the Results

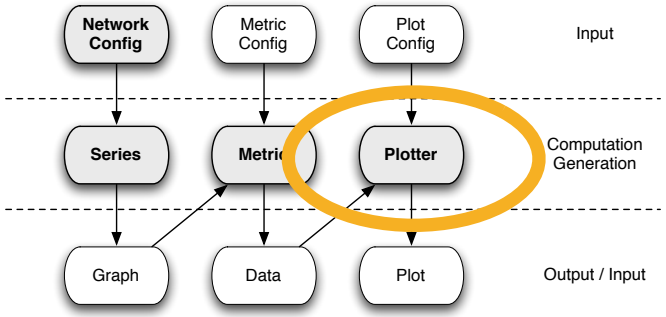
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

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

TECHNISCHE UNIVERSITÄT DARMSTADT

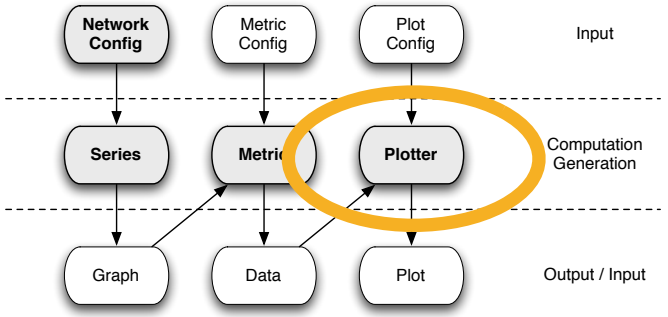
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

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/");
```

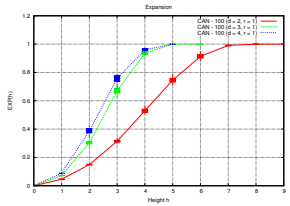


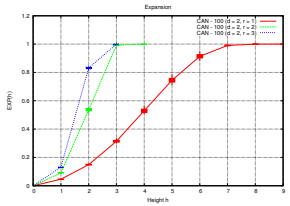
Plotting the Results

TECHNISCHE UNIVERSITÄT DARMSTADT

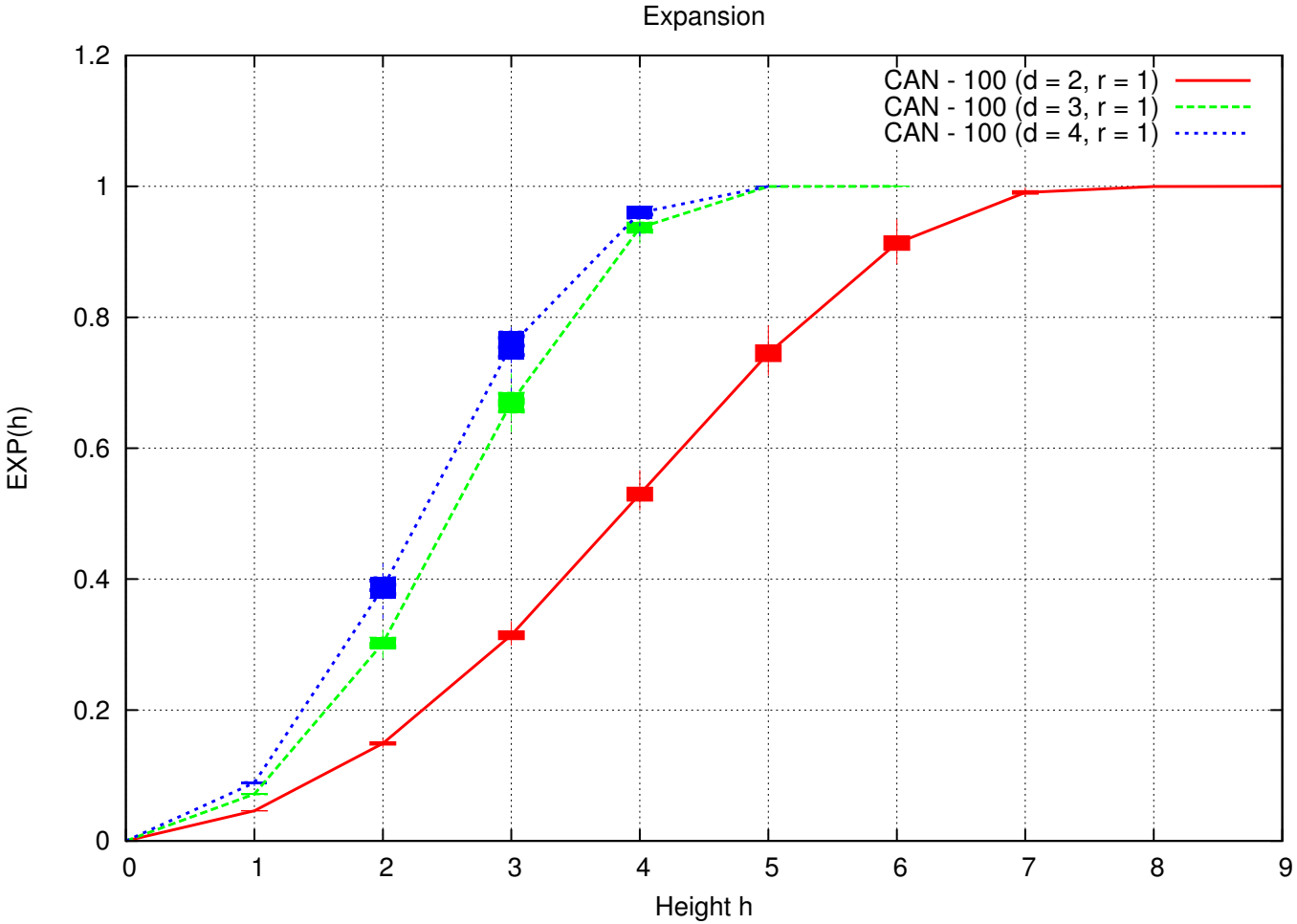
```
Series[] s_can_x_1 = Series.get(can_x_1);
Series[] s_can_2_x = Series.get(can_2_x);
Series[] s_can_x = Series.get(can_x);

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/");
```

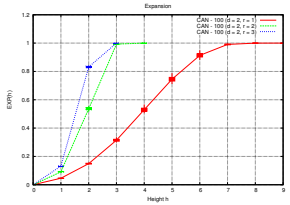




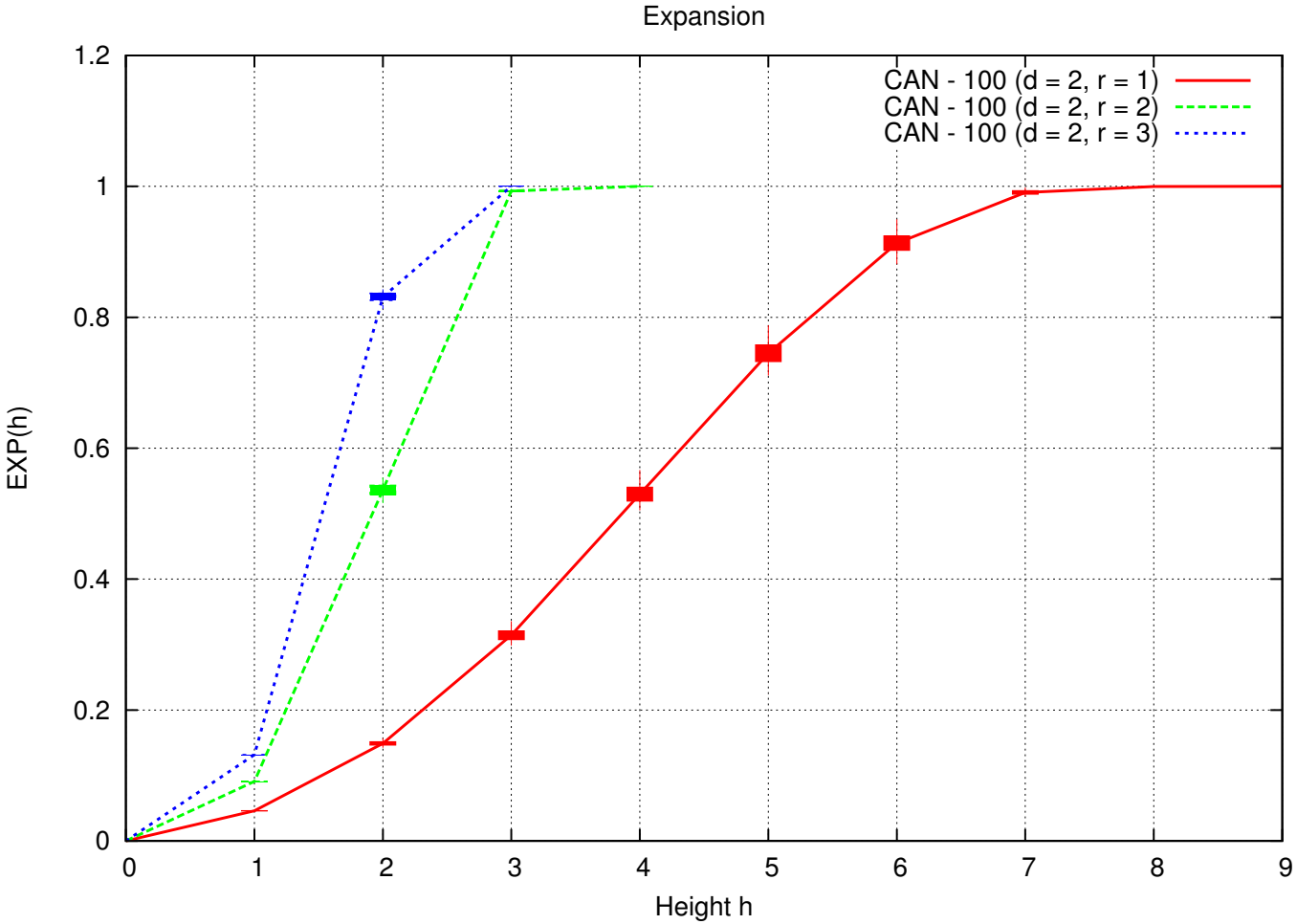


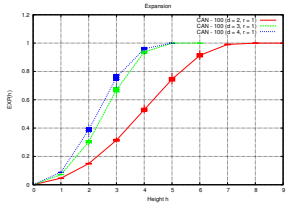
Plotting the Results





Plotting the Results

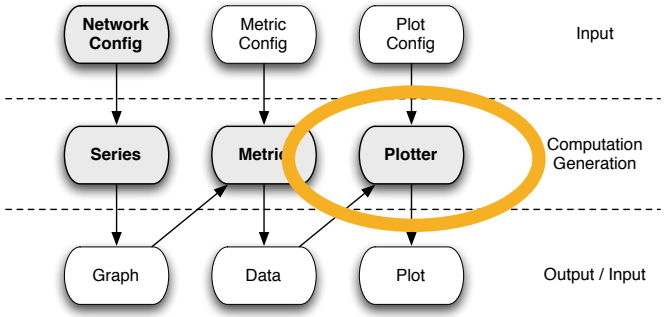


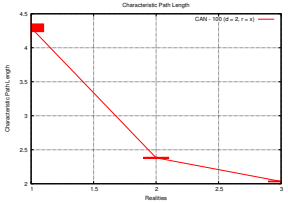


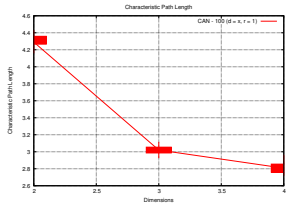
Plotting the Results

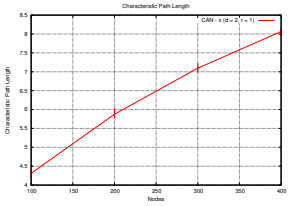
TECHNISCHE UNIVERSITÄT DARMSTADT

```
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/");
```

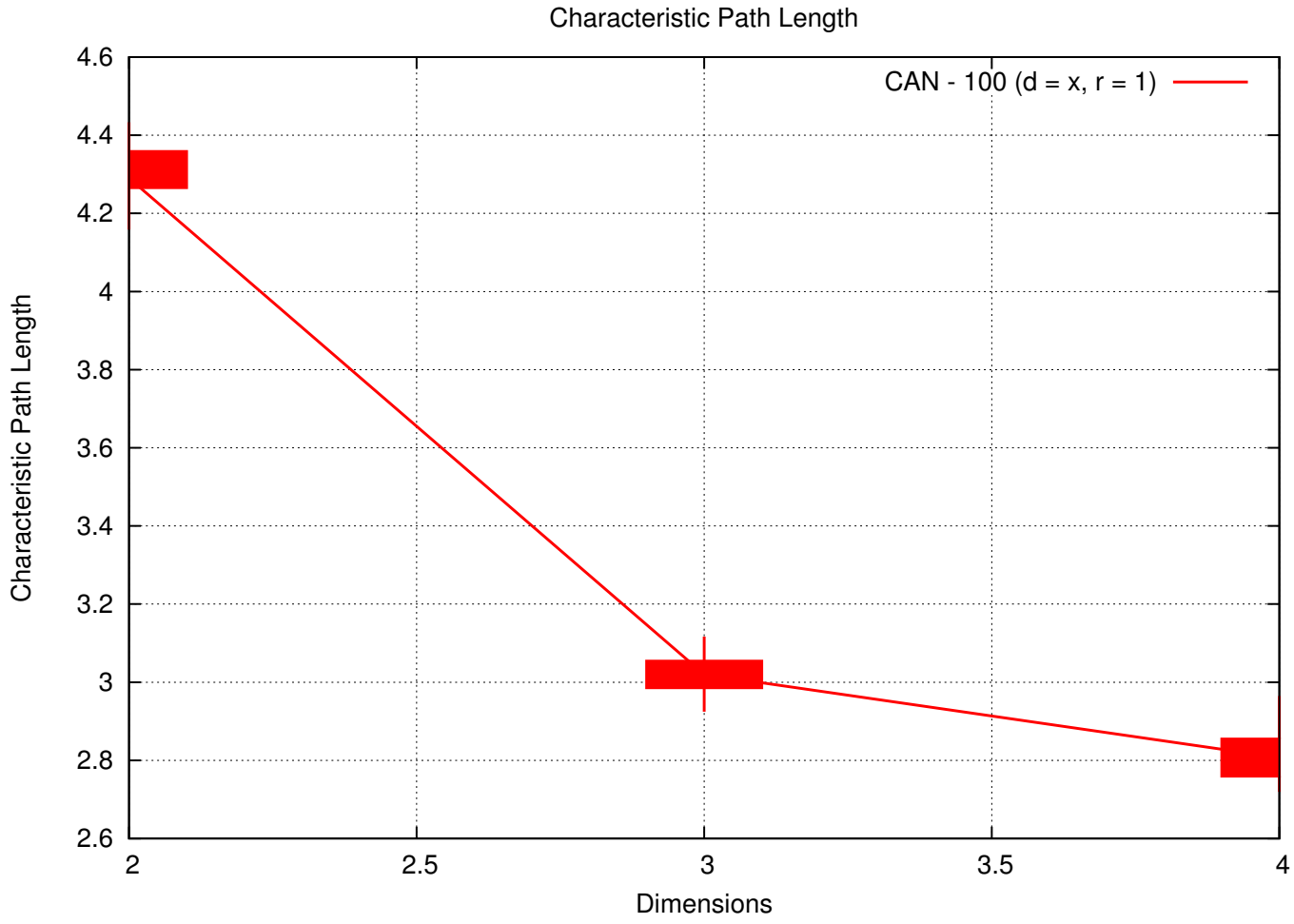


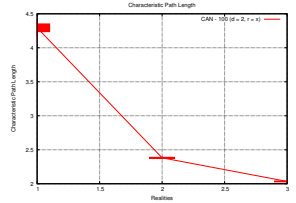


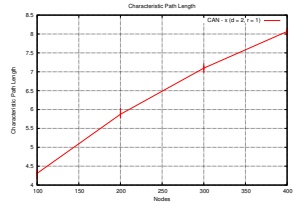




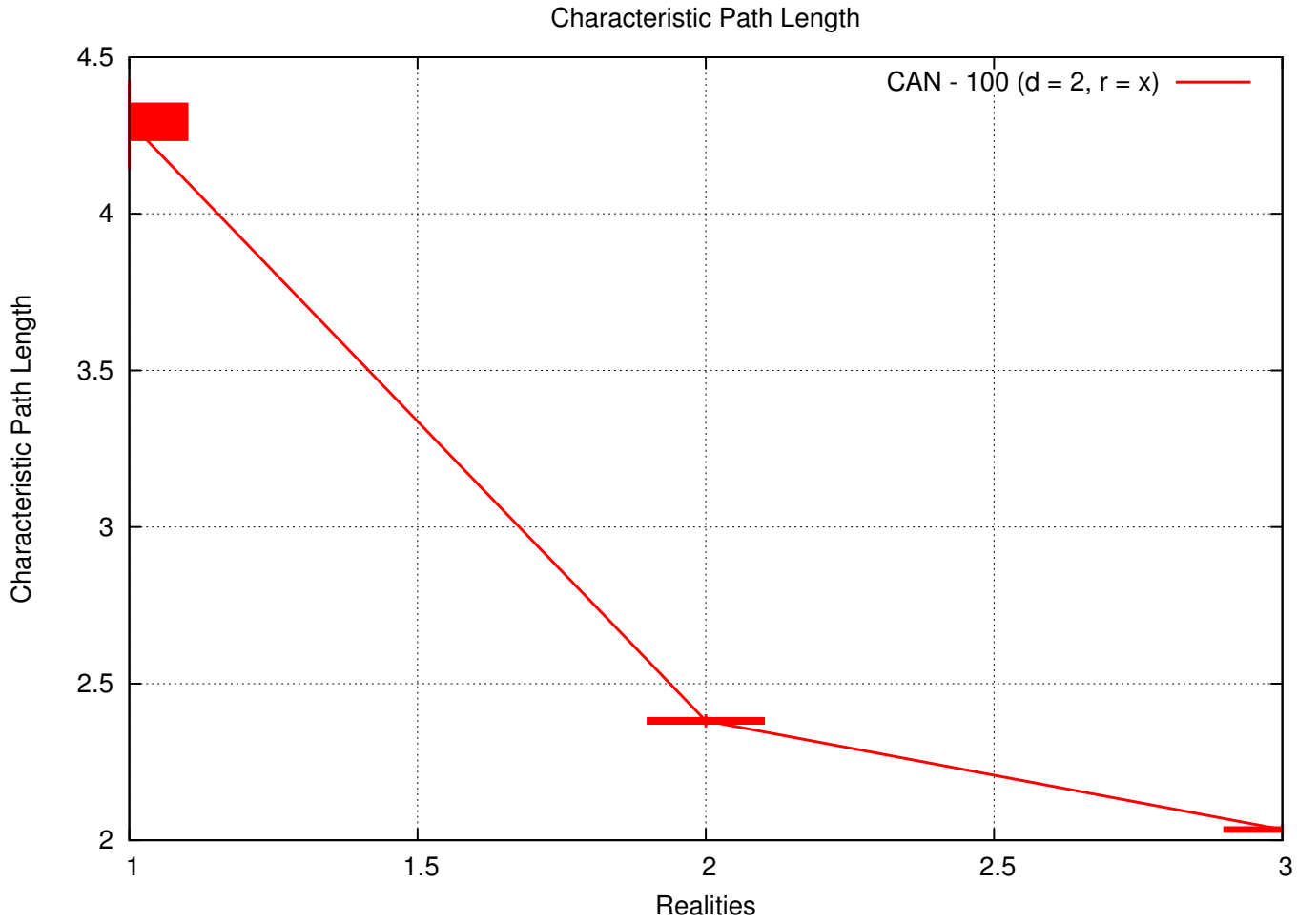
Plotting the Results

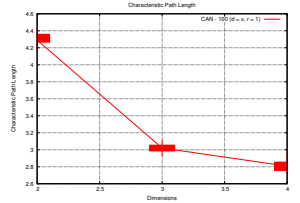


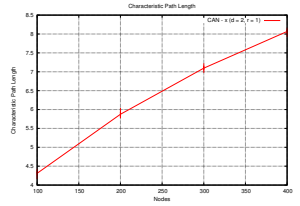




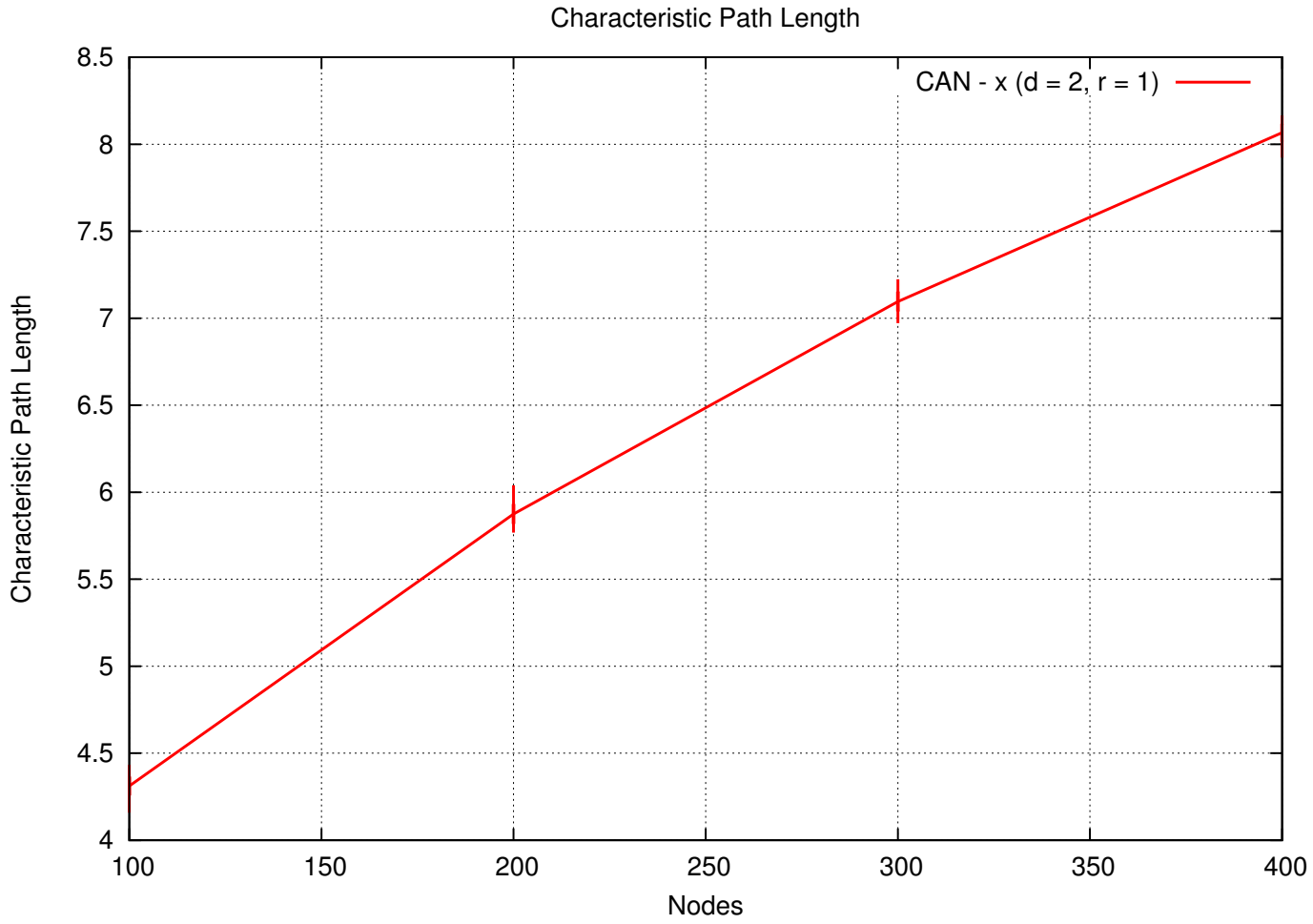
Plotting the Results

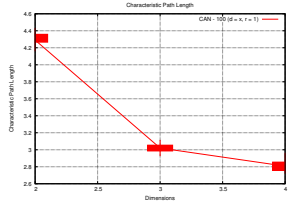


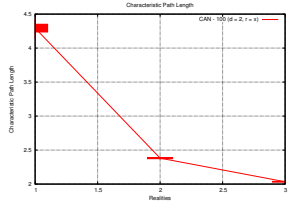




Plotting the Results

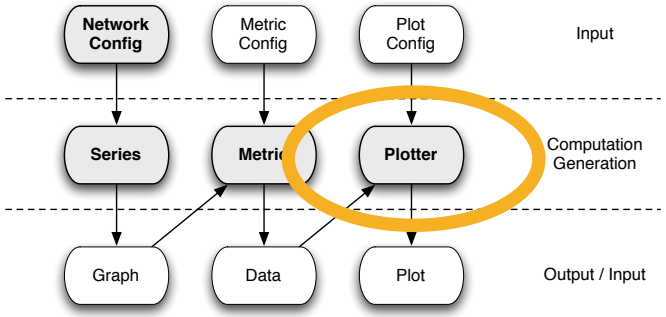


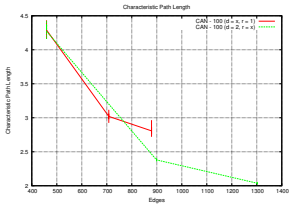




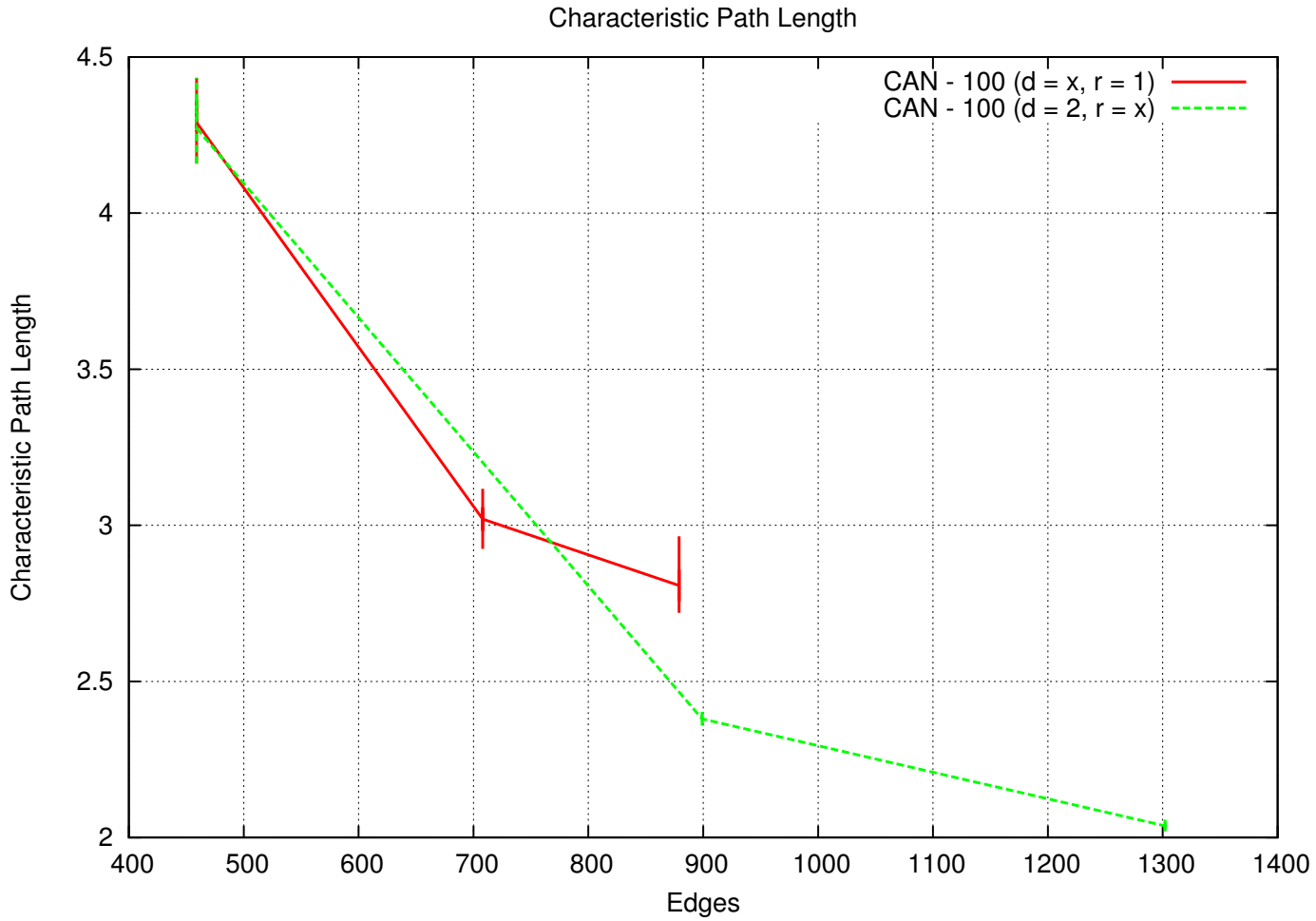
Plotting the Results

```
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/");
```



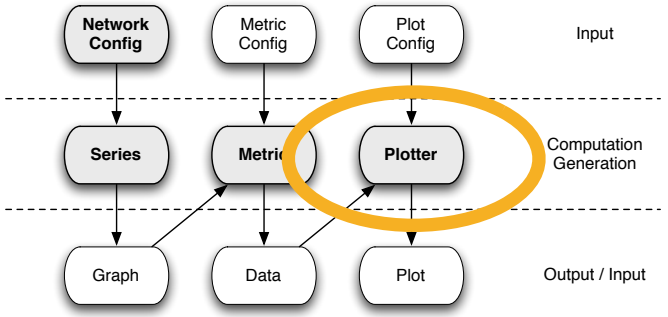


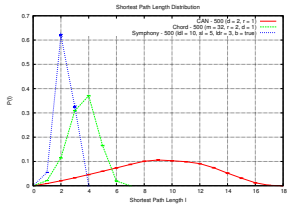
Plotting the Results

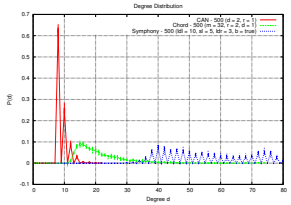


Plotting the Results (cont.)

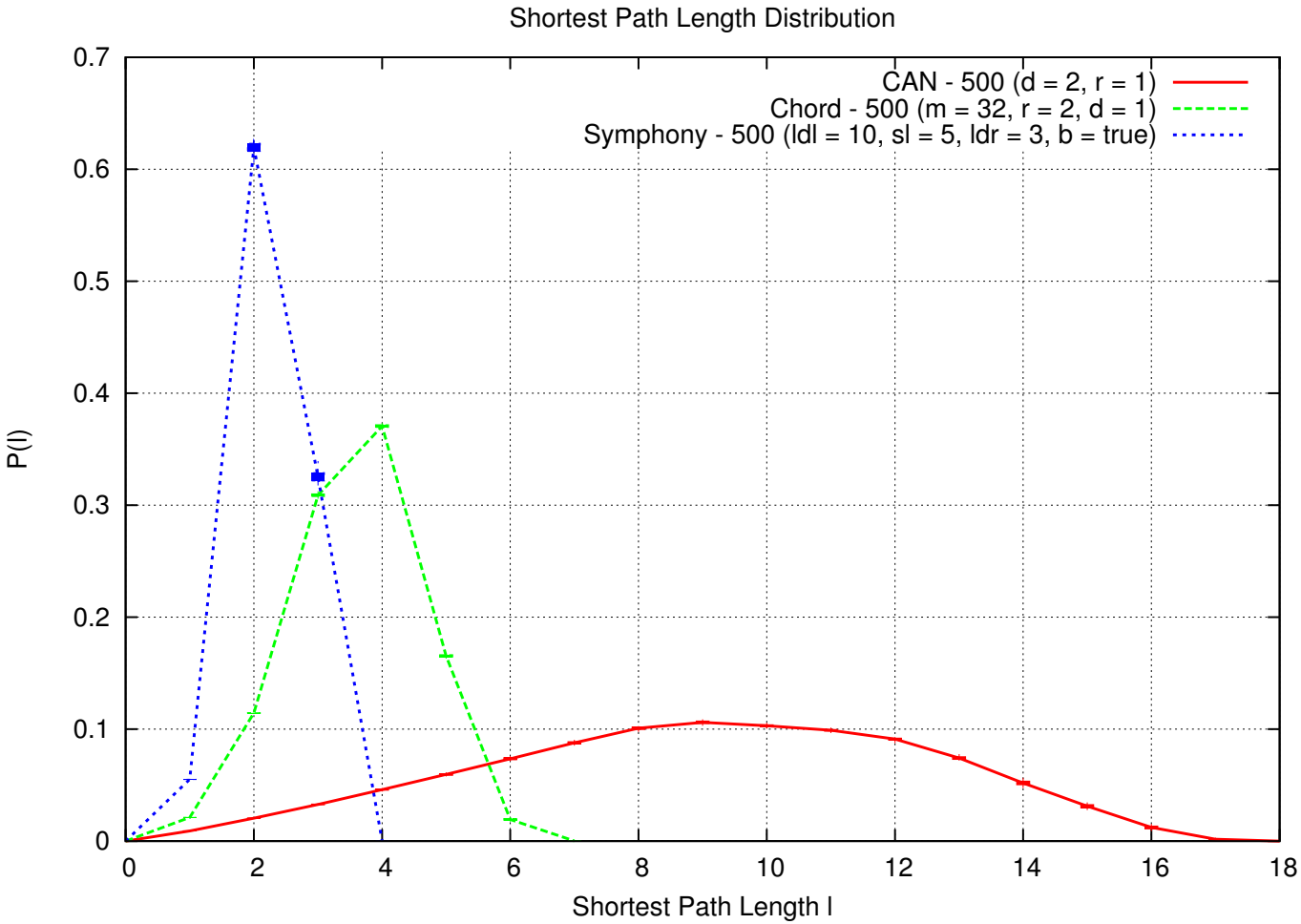
```
Series can = Series.get(new CAN(500, 2, 1));
Series chord = Series.get(new Chord(500, 32, 2, 1));
Series sy = Series.get(new Symphony(500, 10, 5, 3, true));

Series[] all = new Series[]{can, chord, sy};

Plot.allMulti(all, "./data/all/");
```

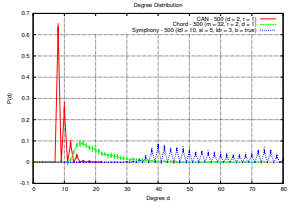




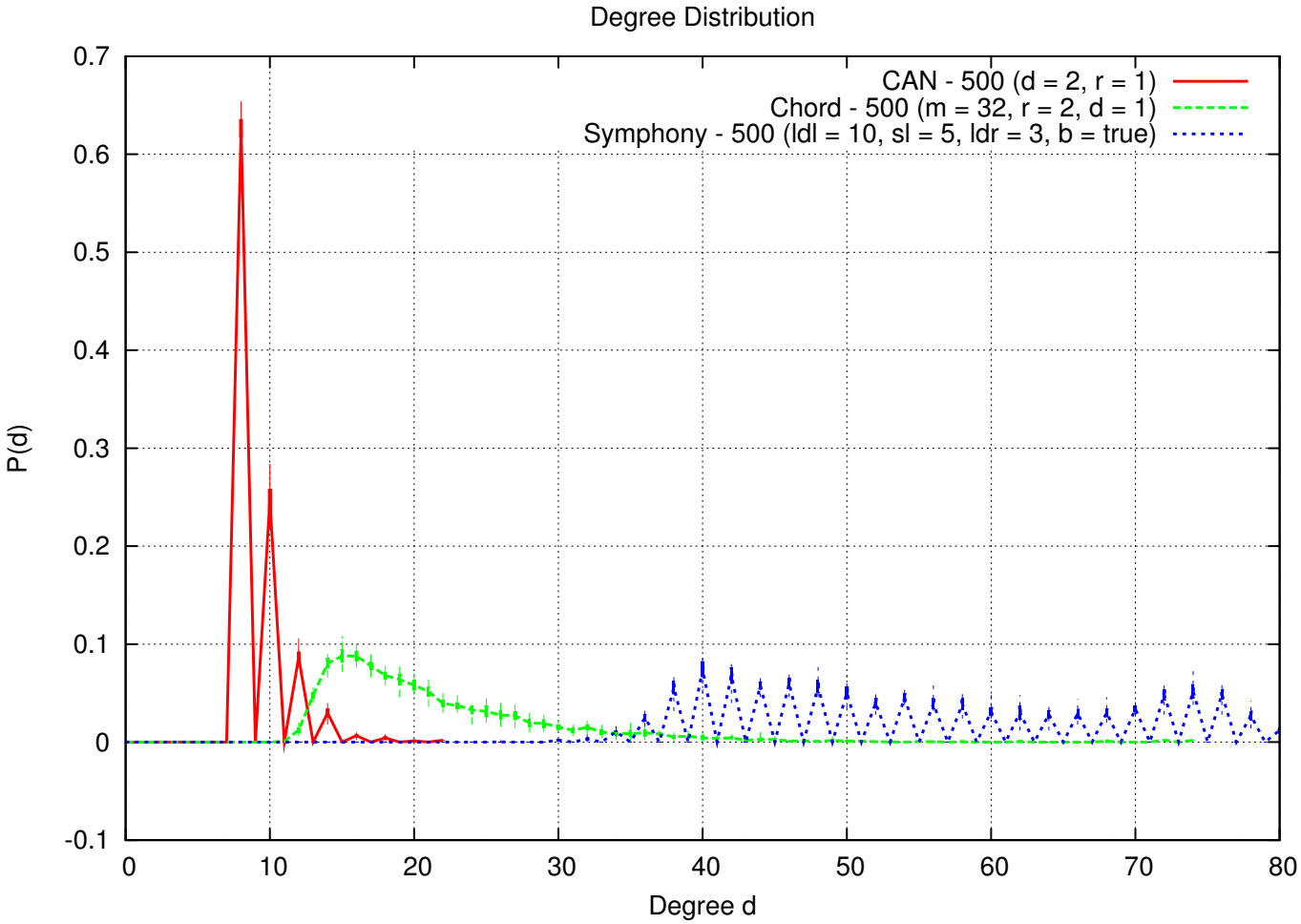


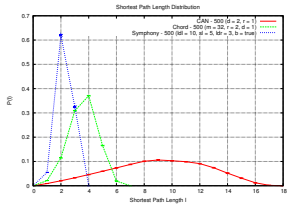
Plotting the Results (cont.)





Plotting the Results (cont.)



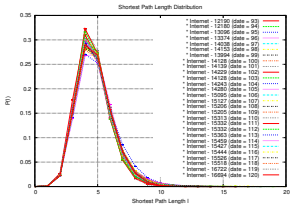


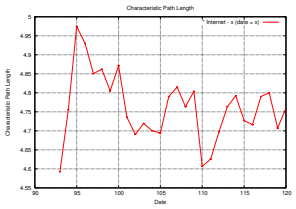
Internet Traces from caida.org

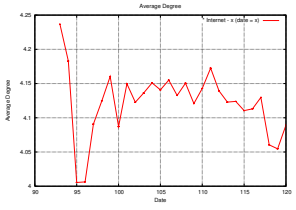
TECHNISCHE UNIVERSITÄT DARMSTADT

caida

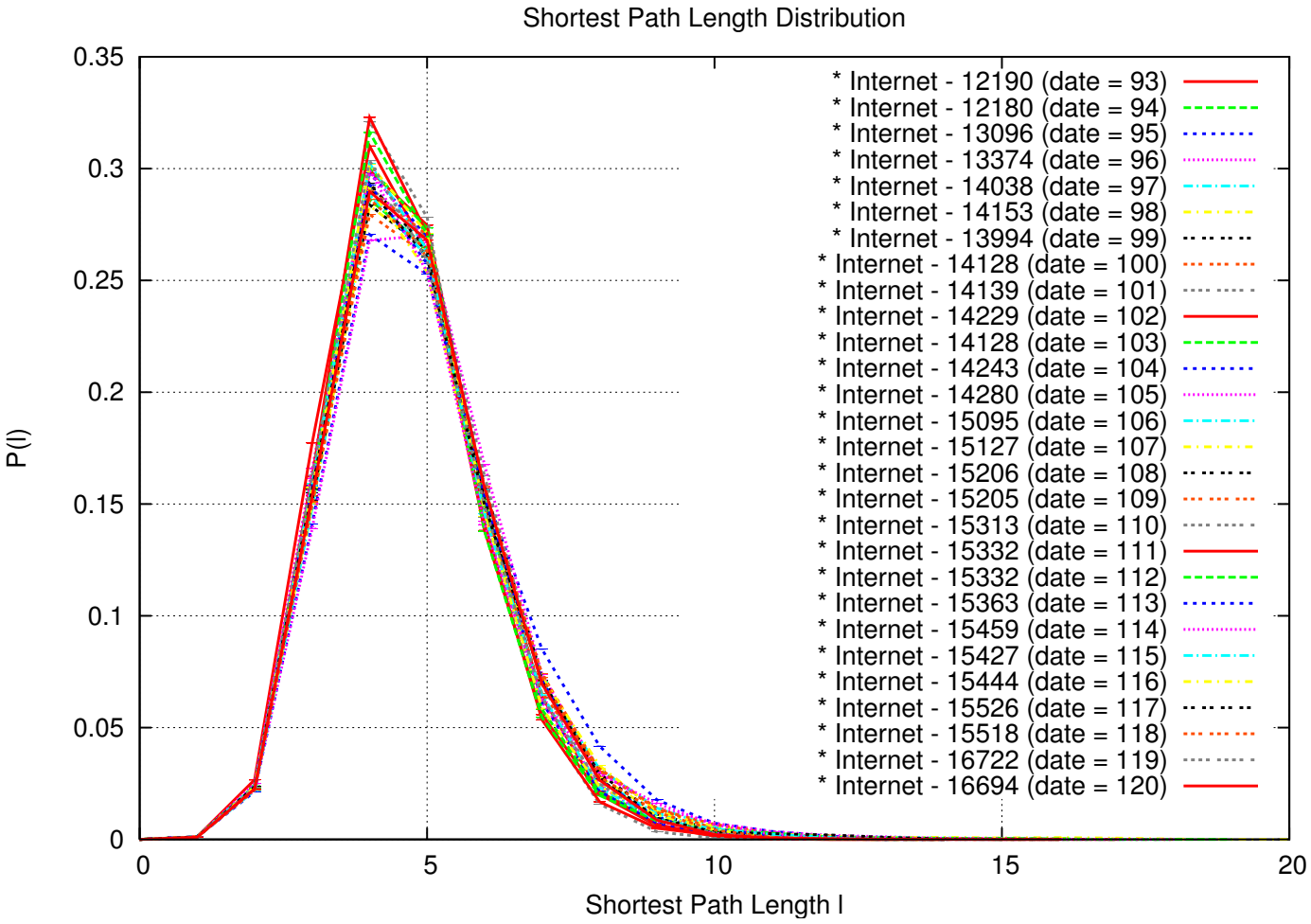
The Cooperative Association for Internet Data Analysis

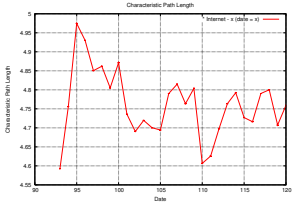


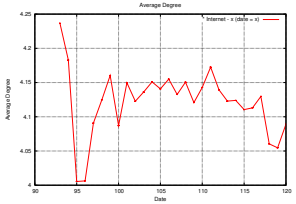




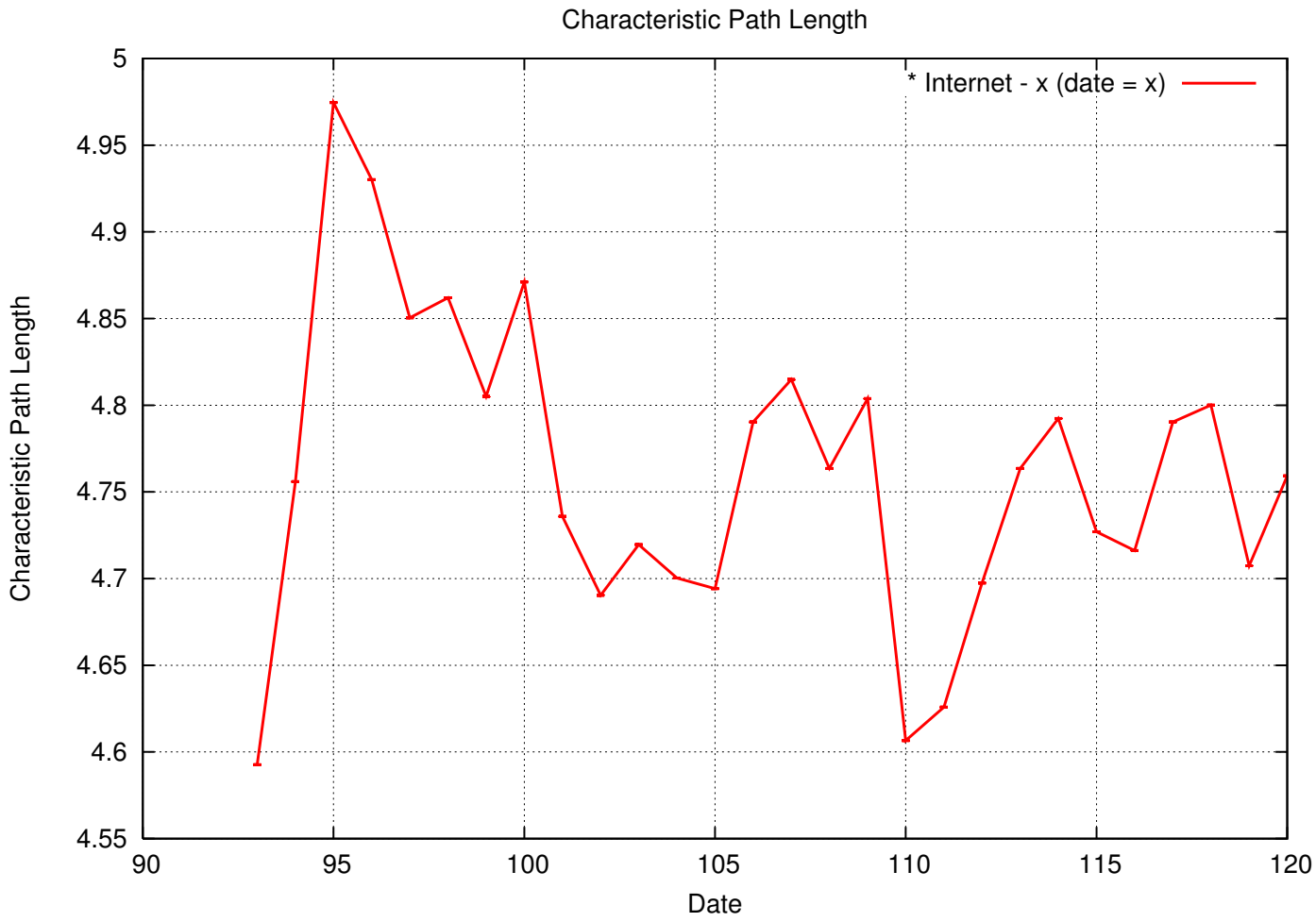
Internet Traces from caida.org

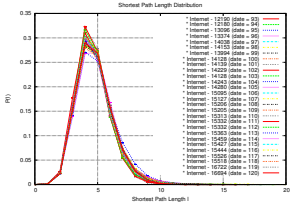


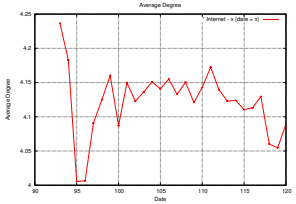




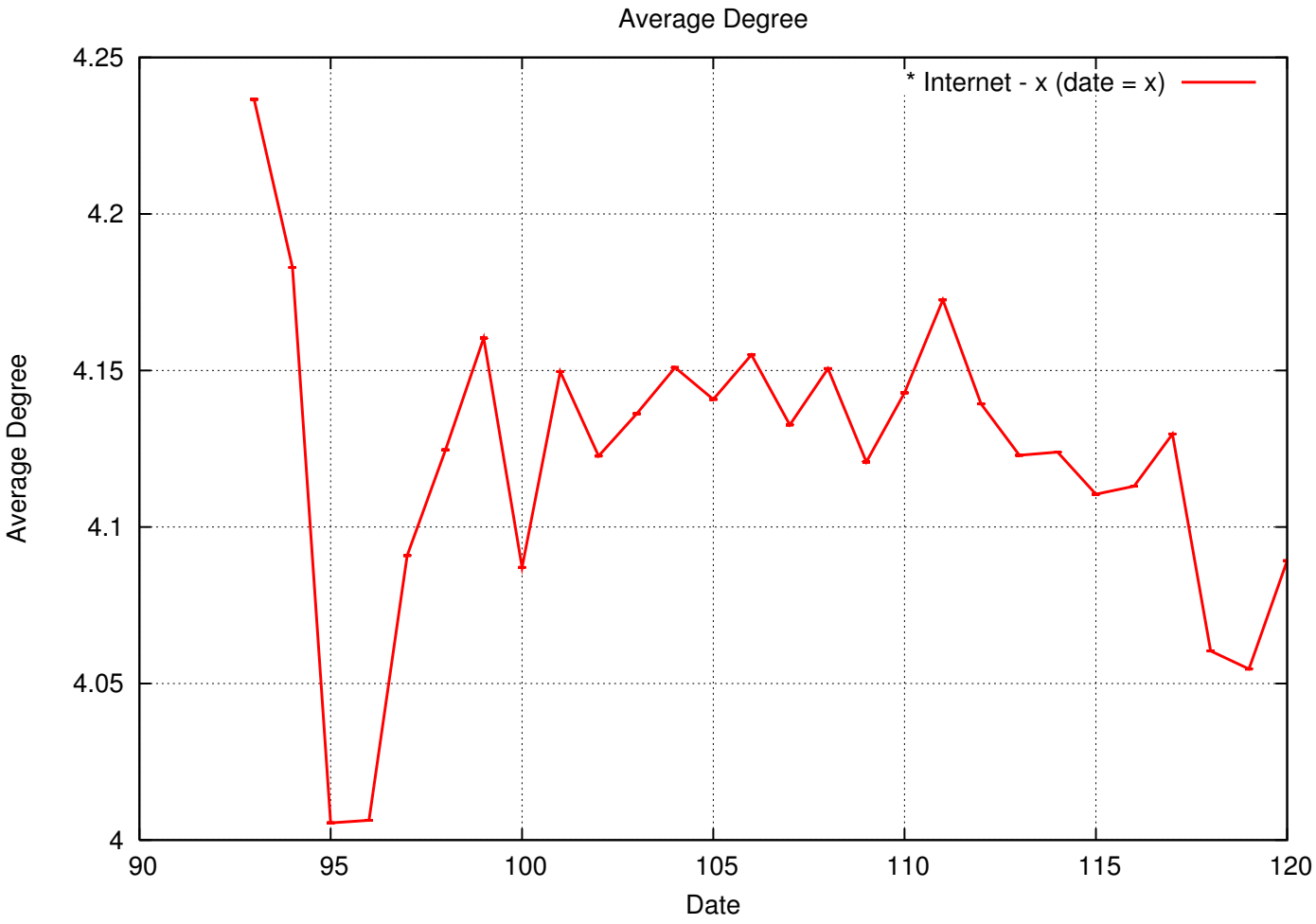
Internet Traces from caida.org

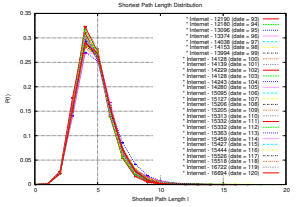


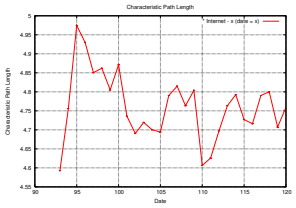




Internet Traces from caida.org

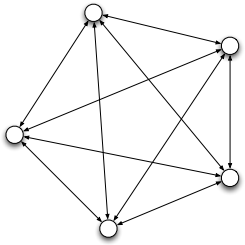


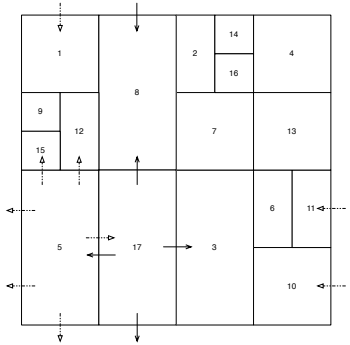


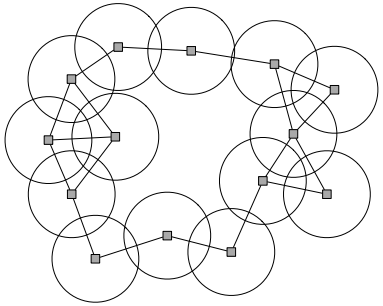


3. Available Networks

Networks already implemented in GTNA



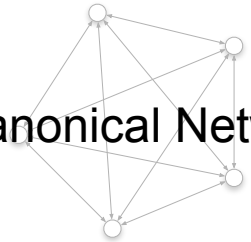




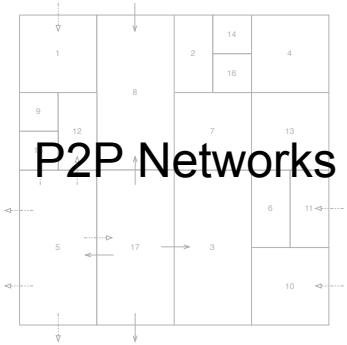
```
graph [
  comment "This is a sample graph"
  directed 1
  IsPlanar 1
  node [
    id 1
    label
    "Node 1"
  ]
  node [
    id 2
    label
    "Node 2" ]
  node [
    id 3
    labe
    "Node 3"
  ]
  edge [
    source 1
    target 2
    label "Edge from node 1 to node 2"
  ]
  edge [
    source 2
    target 3
    label "Edge from node 2 to node 3"
  ]
  edge [
    source 3
    target 1 label
    "Edge from node 3 to node 1"
  ]
]
```

3. Available Networks

Networks already implemented in GTNA

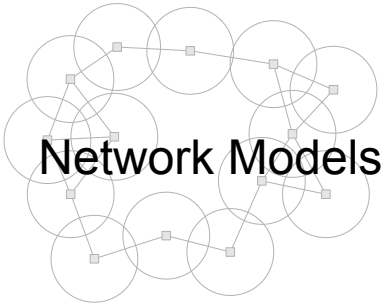


Canonical Networks

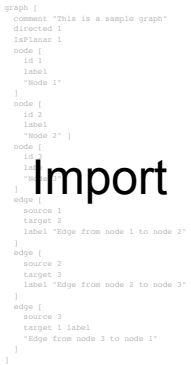


P2P Networks

Transformations



Network Models

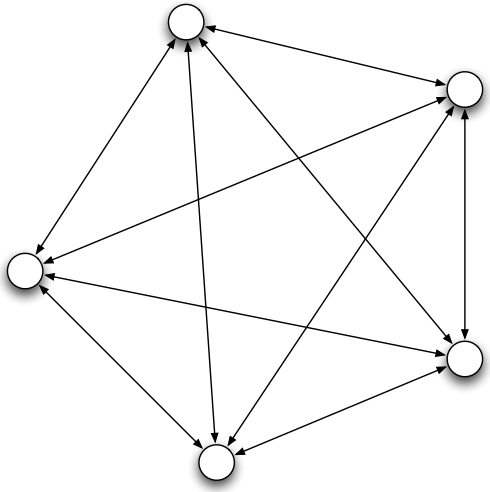


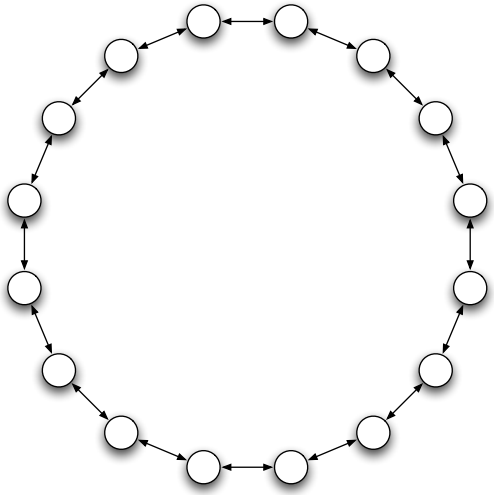
Import

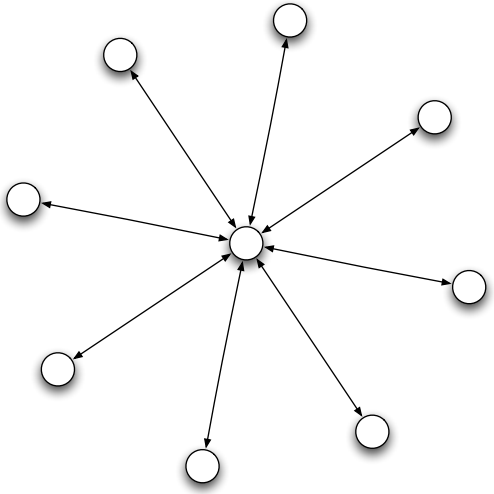
Canonical Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Complete
- Ring
- Star



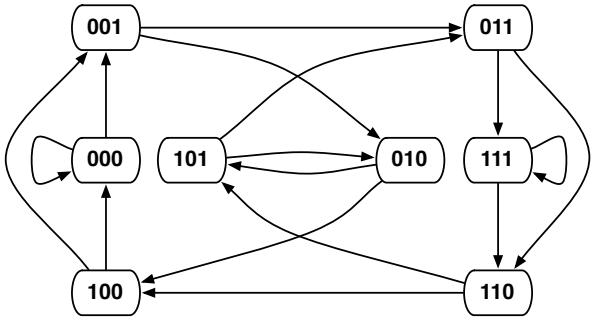


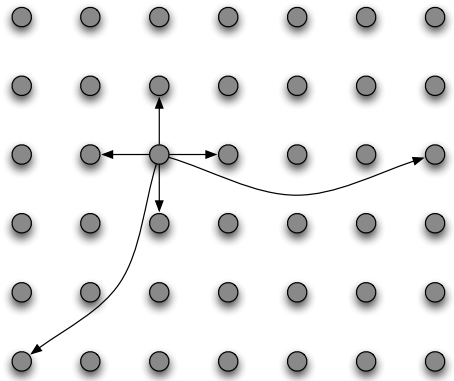


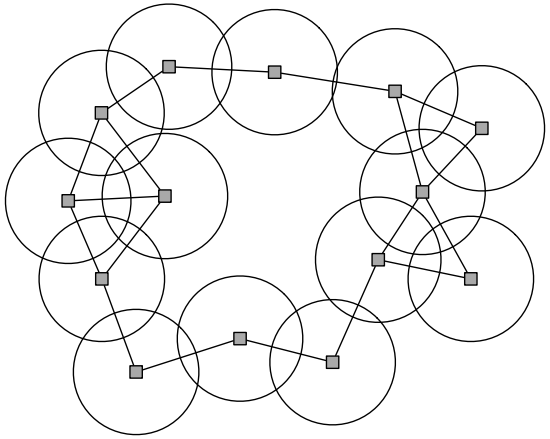
Network Models

TECHNISCHE UNIVERSITÄT DARMSTADT

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

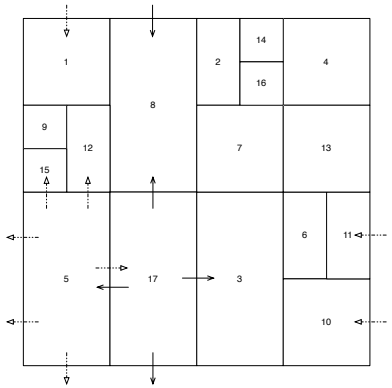


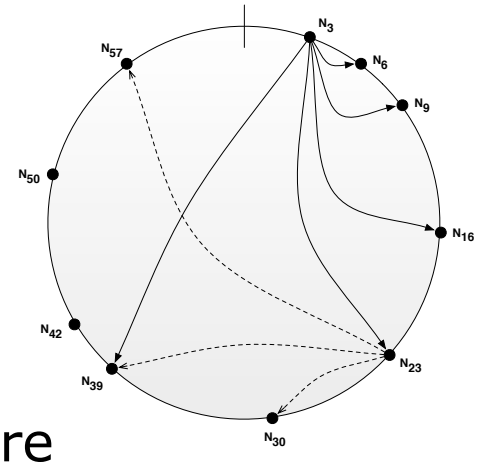


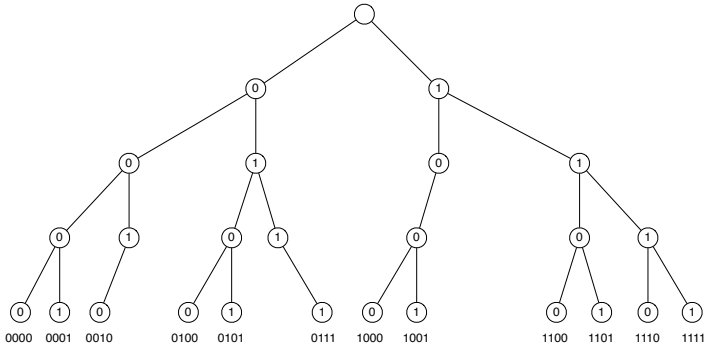


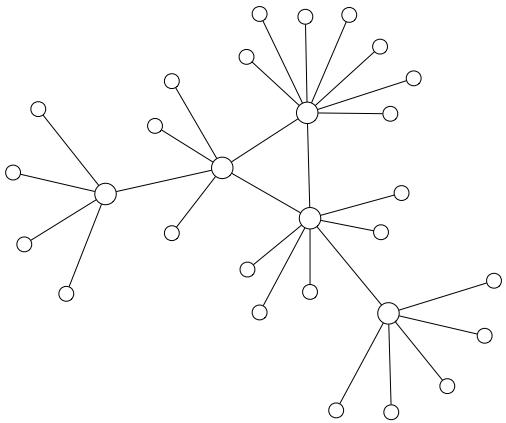
Peer-to-Peer Networks

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









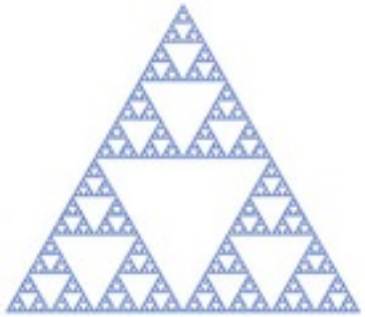
Misc Networks

- Import
  - Readable File
  - Readable List

```
graph [
  comment "This is a sample graph"
  directed 1
  IsPlanar 1
  node [
    id 1
    label
    "Node 1"
  ]
  node [
    id 2
    label
    "Node 2" ]
  node [
    id 3
    labe
    "Node 3"
  ]
  edge [
    source 1
    target 2
    label "Edge from node 1 to node 2"
  ]
  edge [
    source 2
    target 3
    label "Edge from node 2 to node 3"
  ]
  edge [
    source 3
    target 1 label
    "Edge from node 3 to node 1"
  ]
]
```

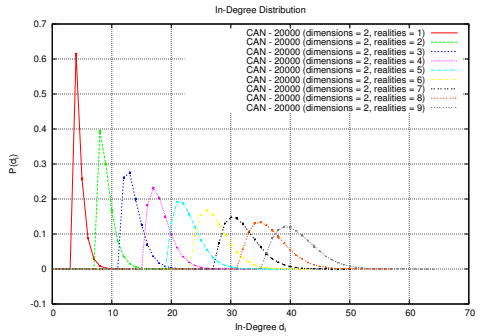
1. own format
2. edges only
3. GML
4. caida.org Internet traces

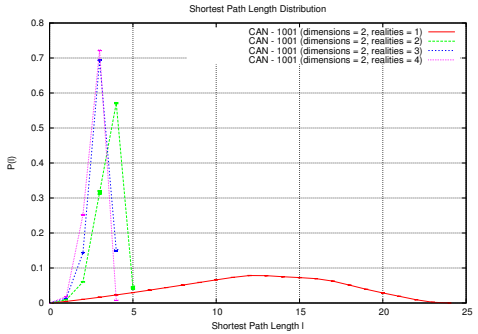
- Transformations
  - Add Edges Per Node
  - Add Edges Randomly
  - Rewire
  - Same Degree Distribution

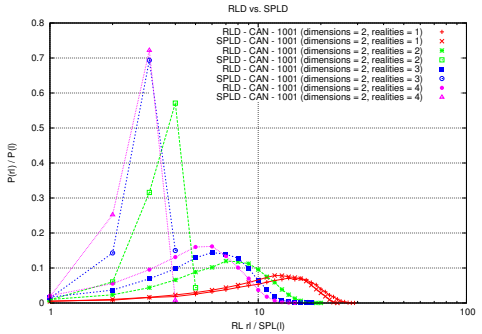


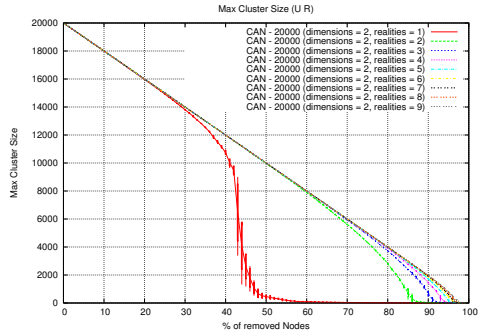
4. Available Metrics

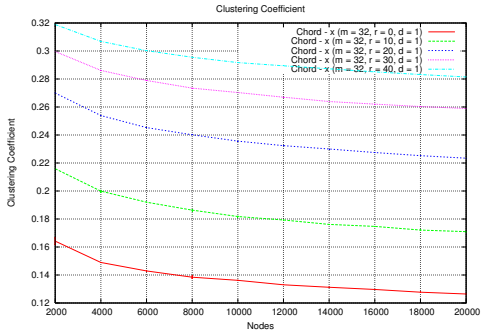
Metrics already implemented in GTNA

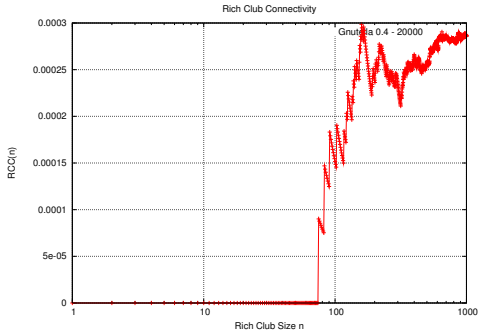






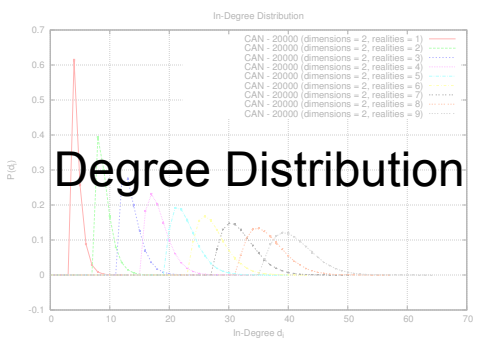


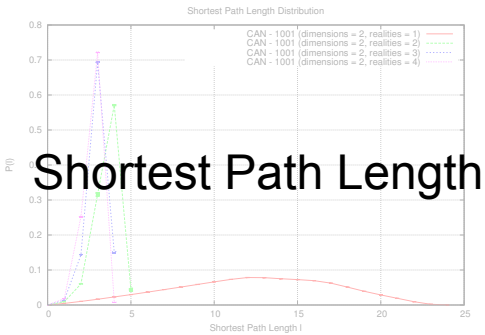


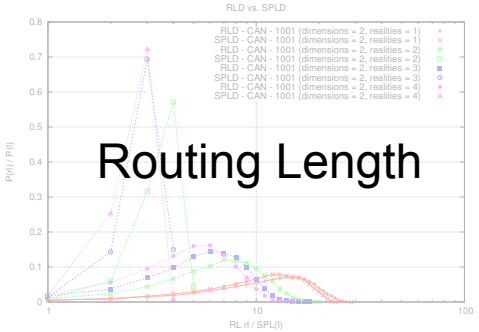


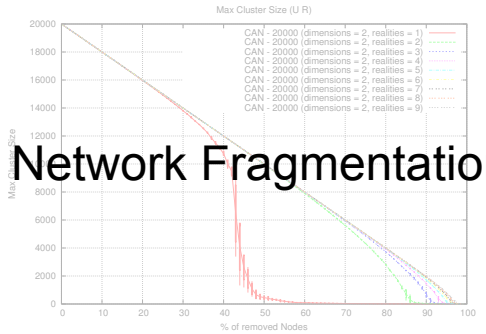
4. Available Metrics

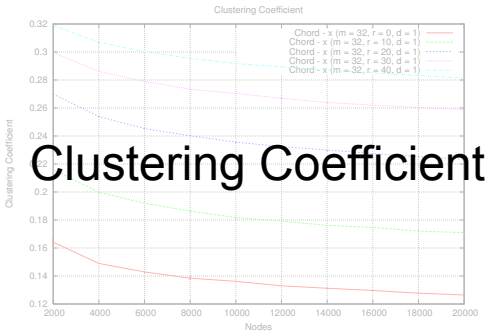
Metrics already implemented in GTNA







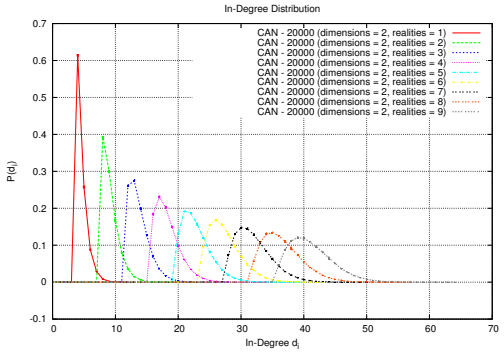


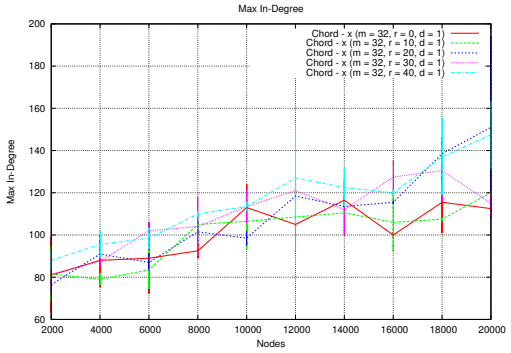


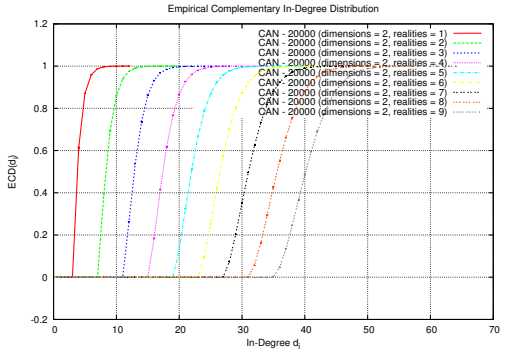
Degree Distribution

- (in-/out-)Degree Distribution
- Empirical Complementary (in-/out-)Degree Distribution

- #Nodes, #Edges
- Average Degree
- Min (in-/out-)Degree
- Max (in-/out-)Degree



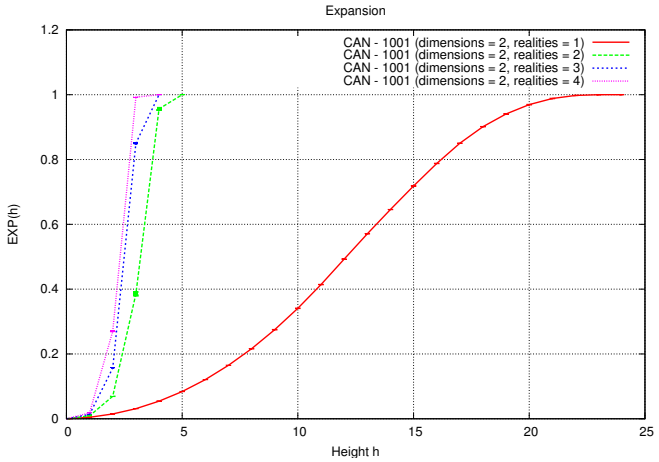


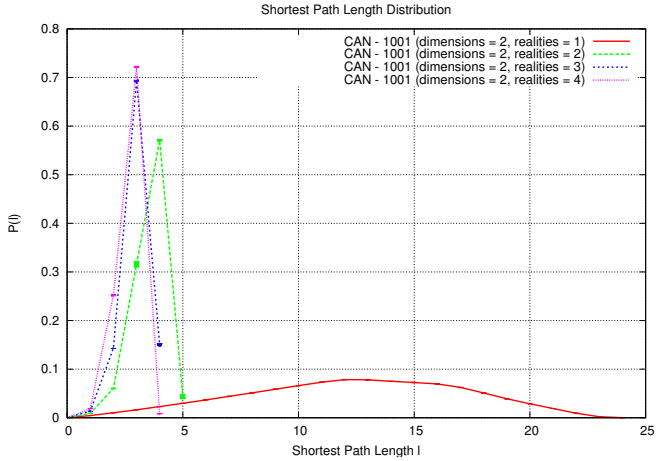


Shortest Path Length

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

- Characteristic Path Length
- Diameter

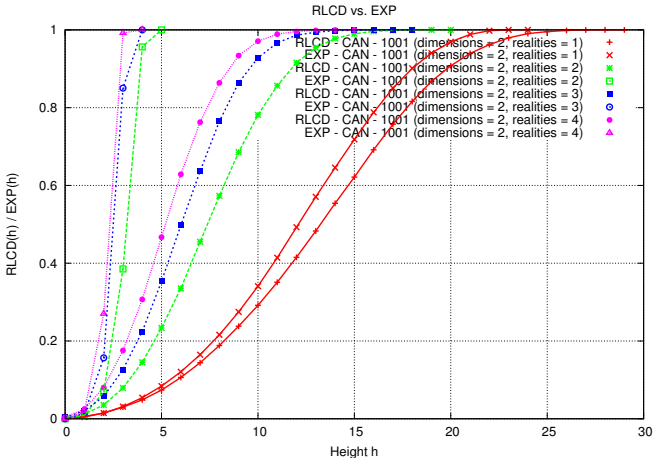


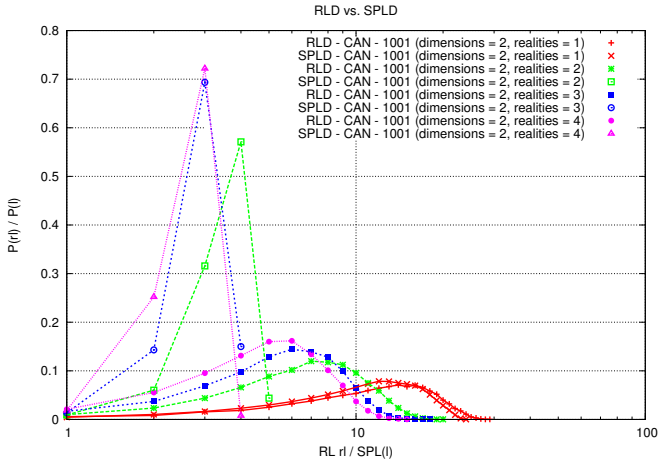


Routing Length

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

- Characteristic Routing Length
- Max Routing Length

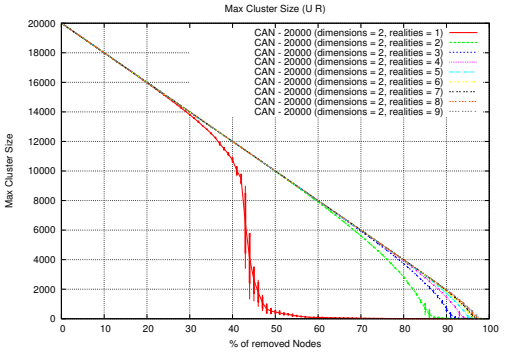


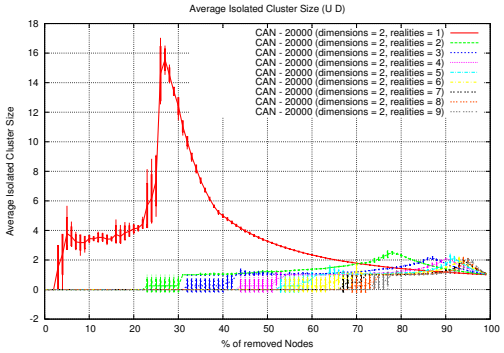


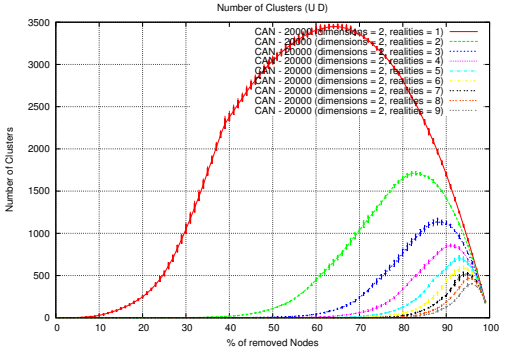
Network Fragmentation

- Max Cluster Size
- Number of Clusters
- Average Isolated Cluster Size

- Point of Rupture
- Average Number of Clusters
- Max Number of Clusters



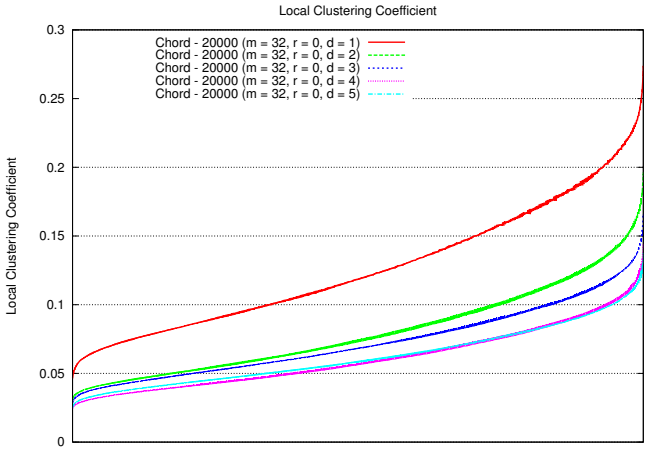


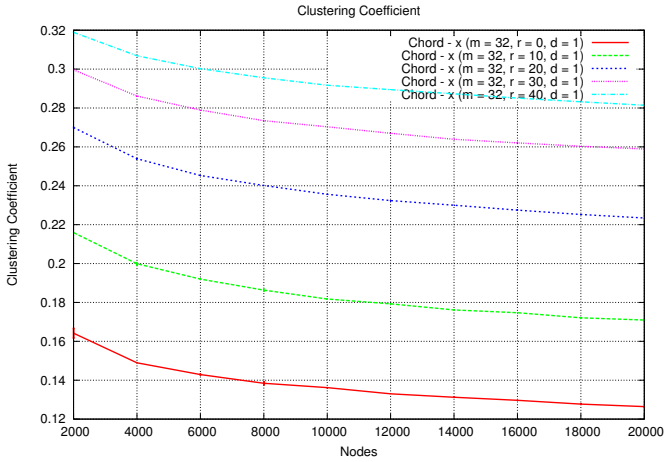


Clustering Coefficient

- Local Clustering Coefficient

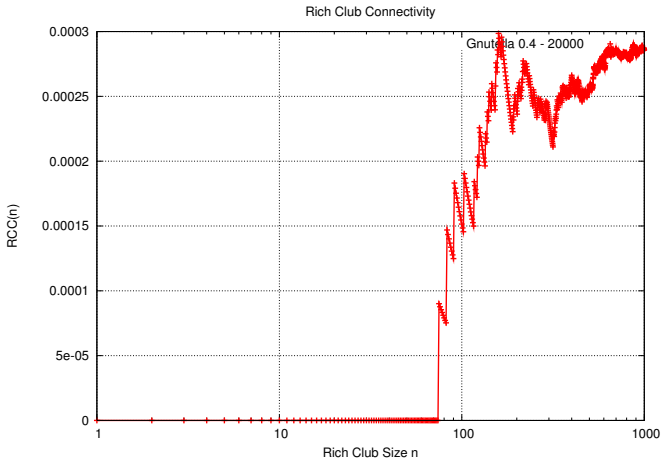
- Clustering Coefficient
- Weighted Clustering Coefficient

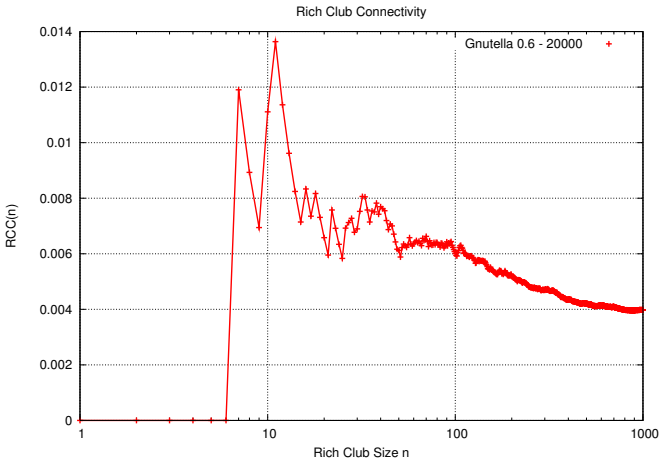




Rich Club Connectivity

- Rich Club Connectivity





5. Extension

Implementation and Configuration of new Networks / Metrics

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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_Y = Degree of Most Nodes
```

Implementing a new Network

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

Implementing a new Network

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

Implementing a new Network

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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);
  }
}
```

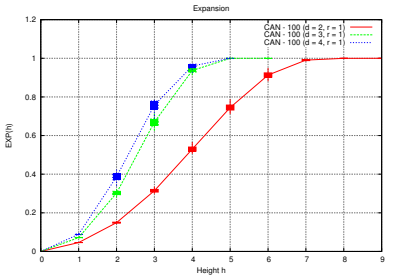
Parameter

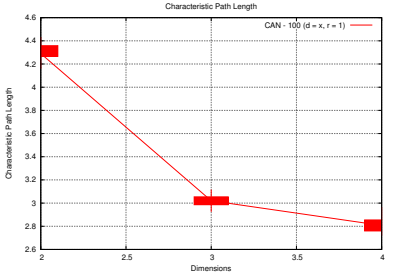
Constructor

Generator

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
```

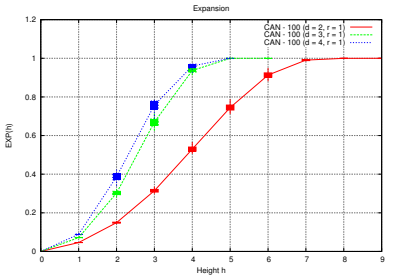


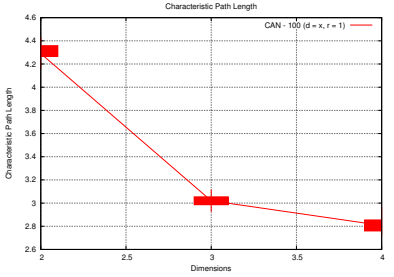


Configuring a new Network

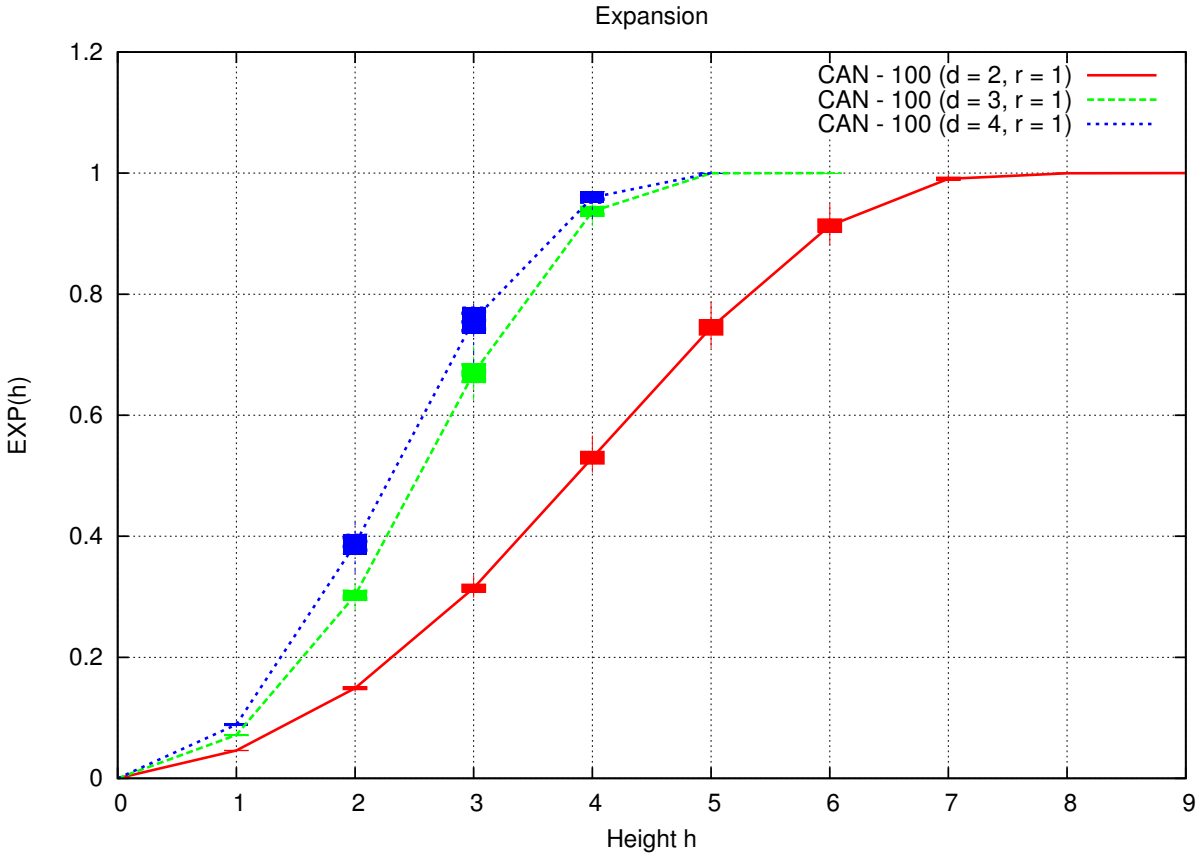
TECHNISCHE UNIVERSITÄT DARMSTADT

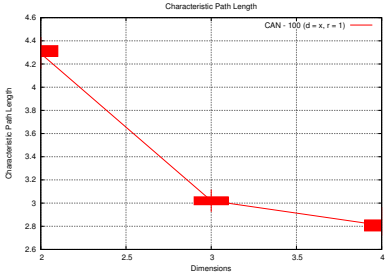
```
CAN_NETWORK_NAME = CAN

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

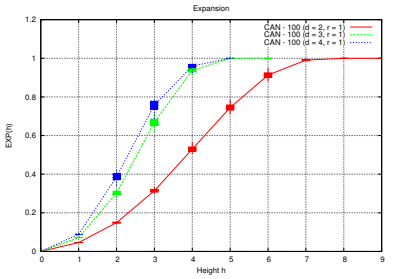


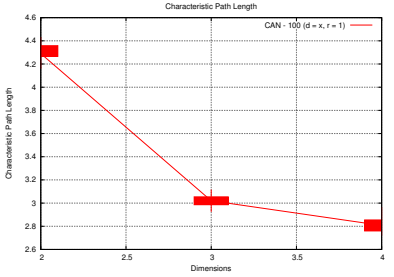


Configuring a new Network

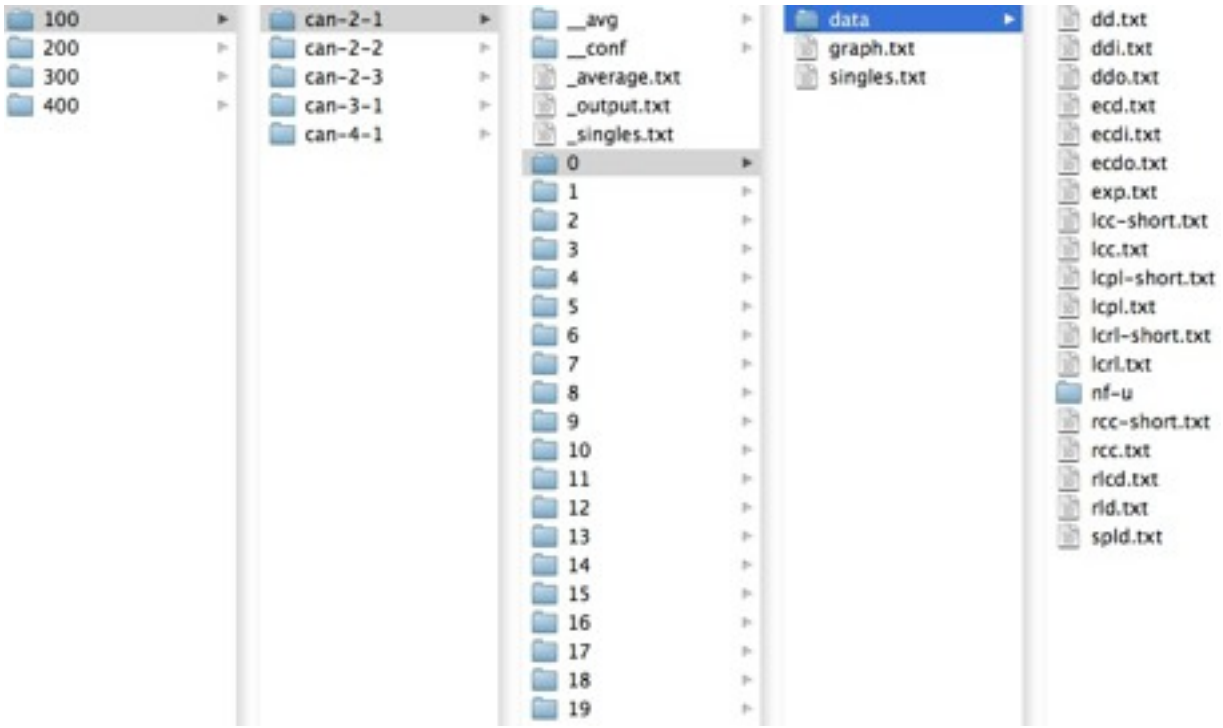
TECHNISCHE UNIVERSITÄT DARMSTADT

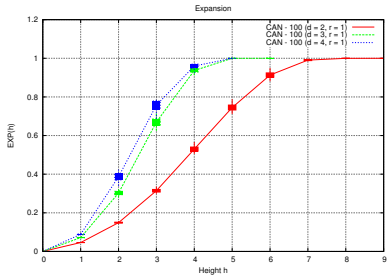
```
CAN_NETWORK_NAME = CAN

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
```

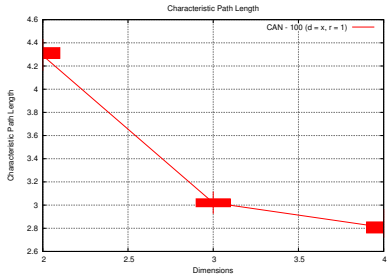




Configuring a new Network



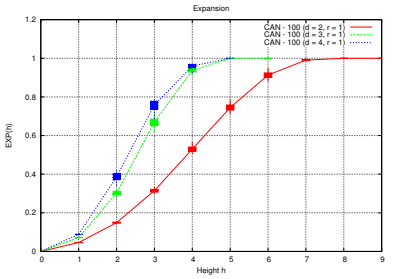


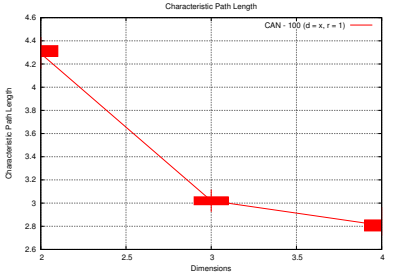


Configuring a new Network

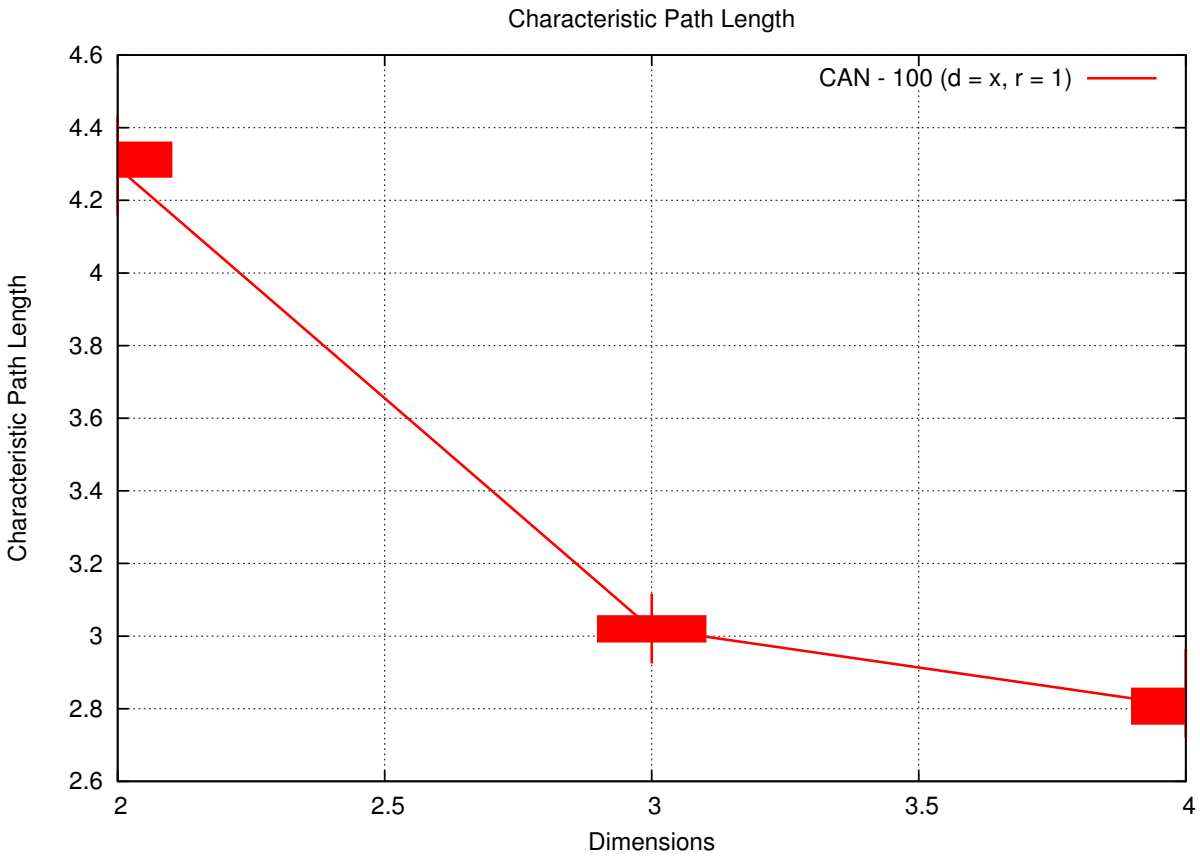
TECHNISCHE UNIVERSITÄT DARMSTADT

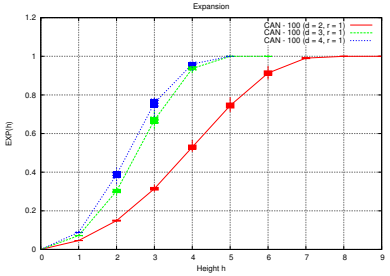
```
CAN_NETWORK_NAME = CAN

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





Implementing a new Metric

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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;
  }
}
```

Attributes

Implementing a new Metric

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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;
  }
}
```

Attributes

Constructor

Implementing a new Metric

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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;
  }
}
```

Attributes

Constructor

Computation

Implementing a new Metric

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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;
  }
}
```

Attributes

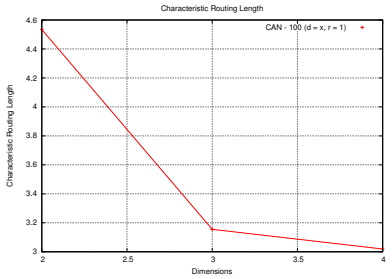
Constructor

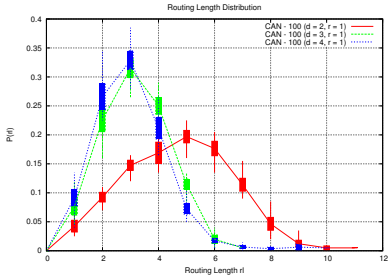
Computation

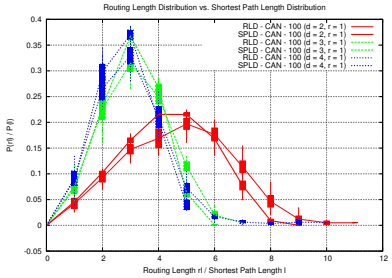
Output

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)
```



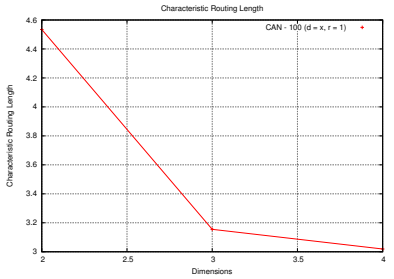


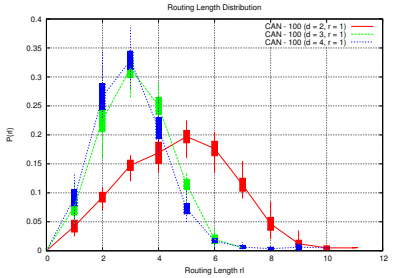


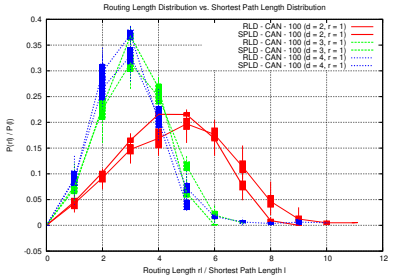
Configuring a new Metric

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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)
```

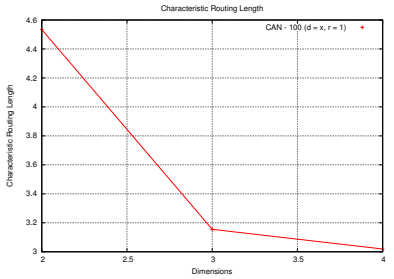


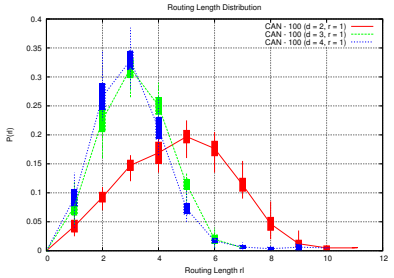


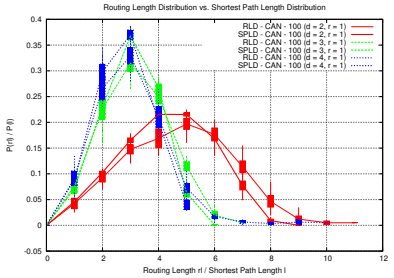


Configuring a new Metric

```
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)
```

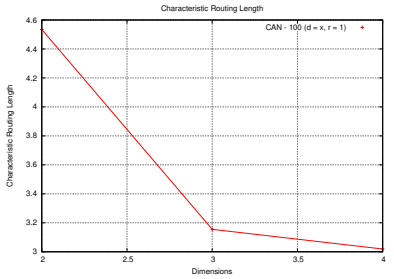


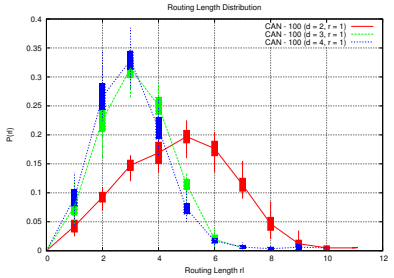


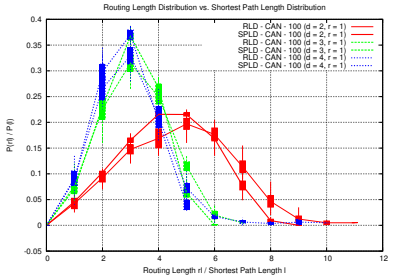


Configuring a new Metric

```
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)
```

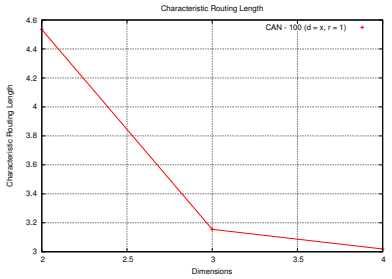


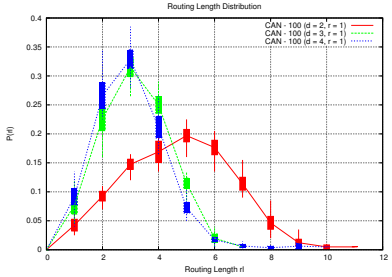


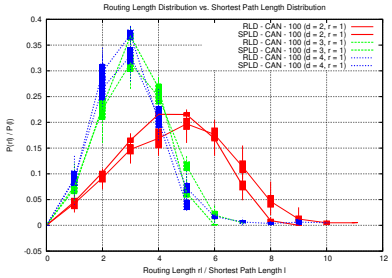


Configuring a new Metric

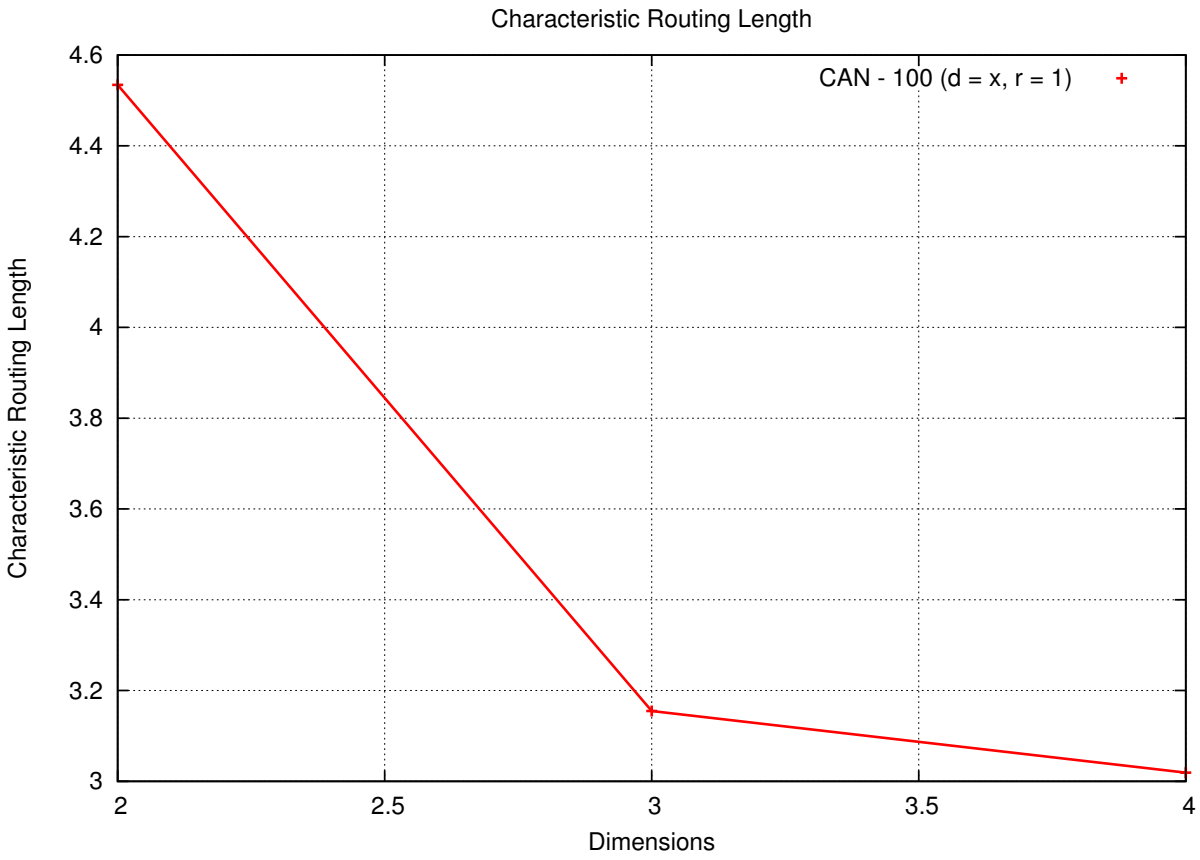
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

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)
```

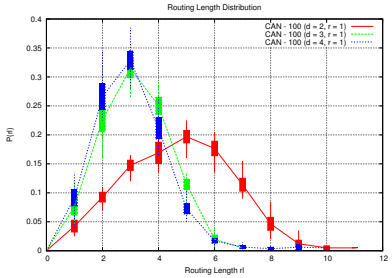


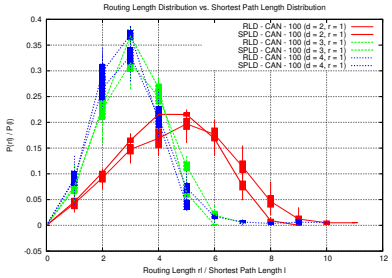




Configuring a new Metric

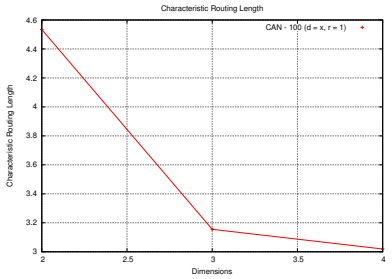


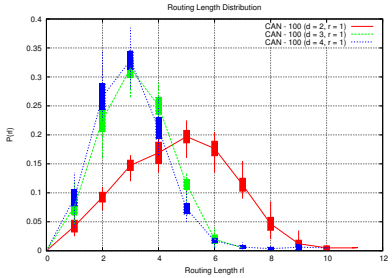


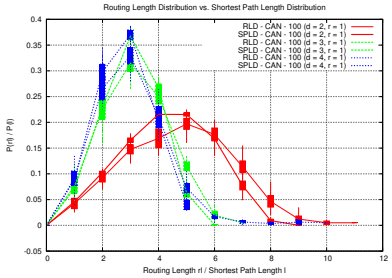


Configuring a new Metric

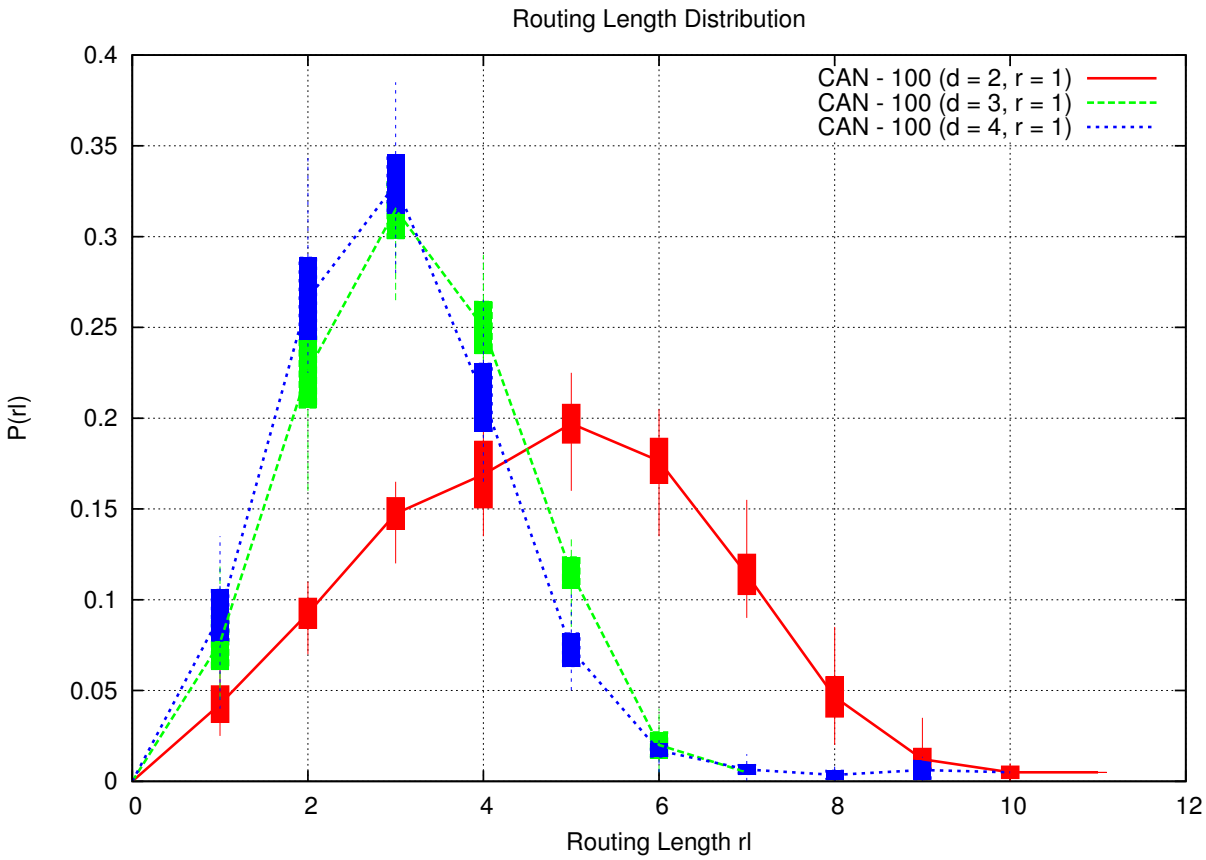
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

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)
```

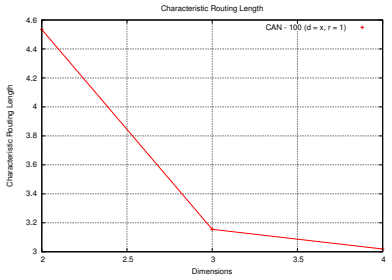


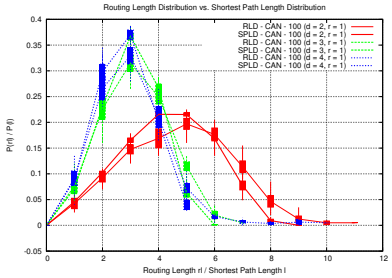




Configuring a new Metric

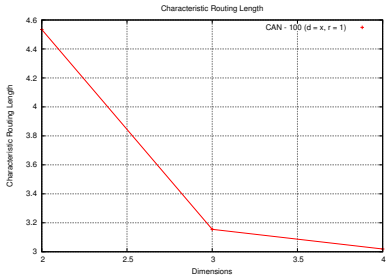


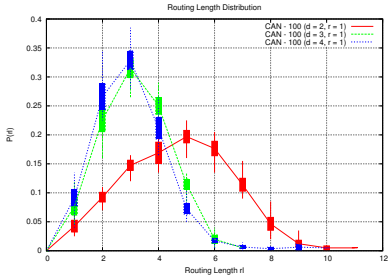


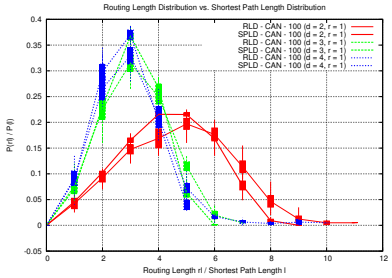


Configuring a new Metric

```
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)
```

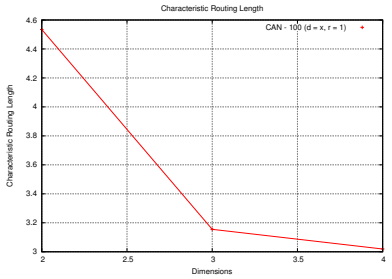


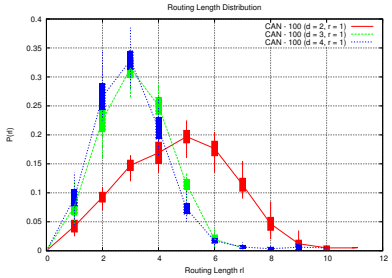


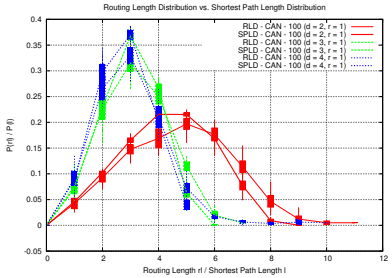


Configuring a new Metric

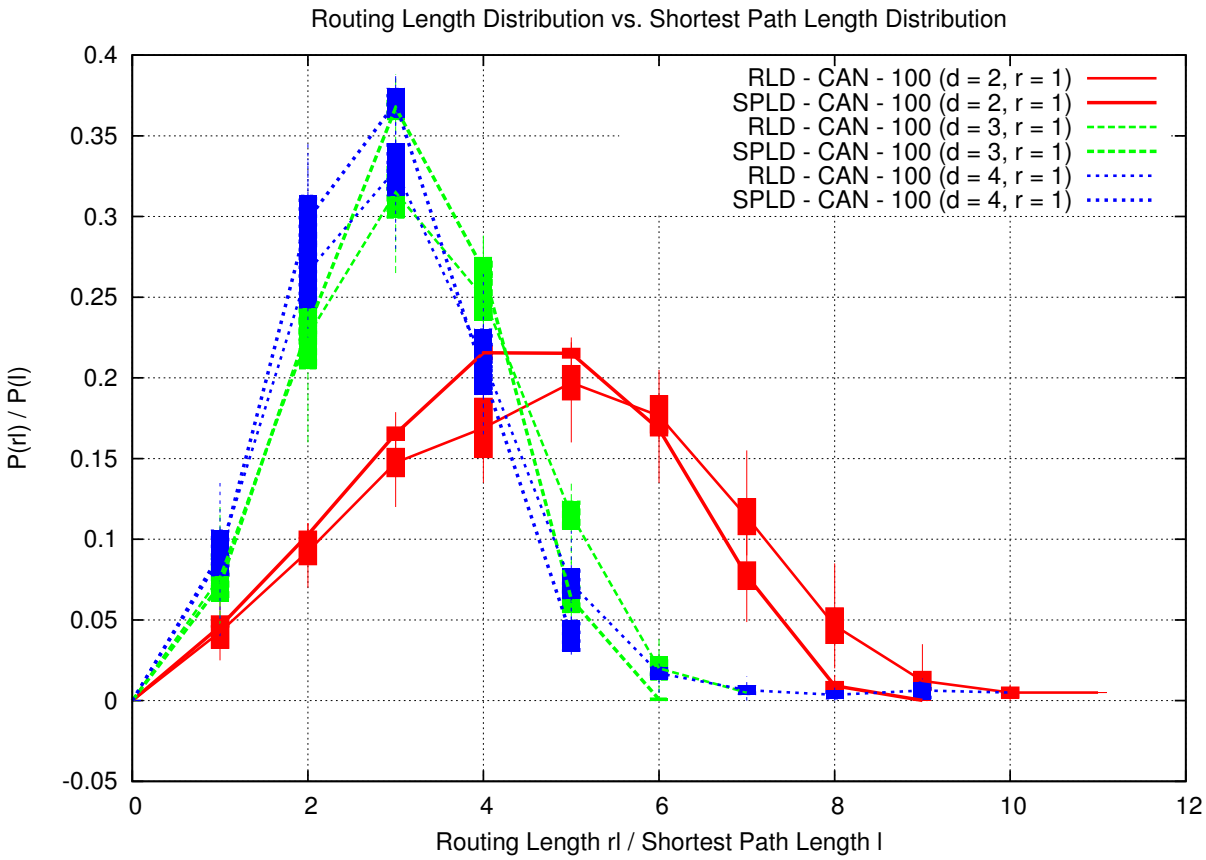
```
RL_CLASS = gtna.metrics.RoutingLength
RL_NAME = Routing Length
RL_SINGLES_KEYS = CRL
RL_DATA_KEYS = RLD
RL_SINGLES_PLOTS = CRL
RL_DATA_PLOTS = RLD

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)
```

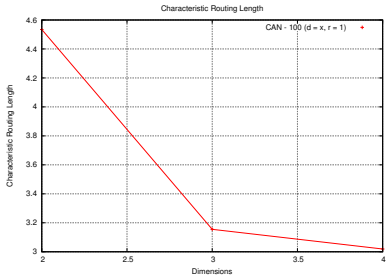


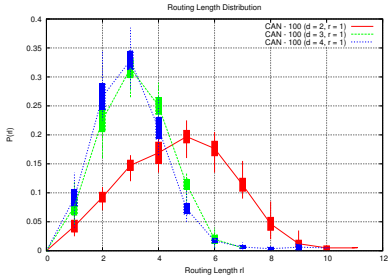




Configuring a new Metric

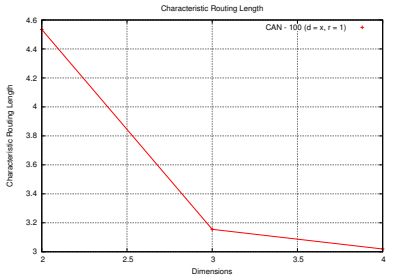


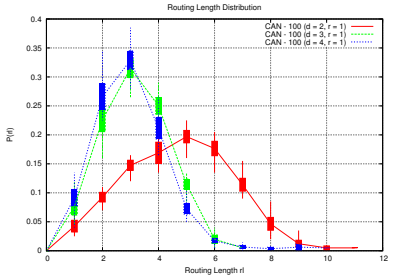


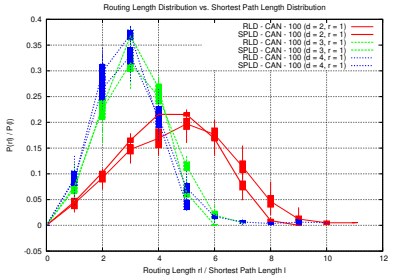


Configuring a new Metric

```
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)
```

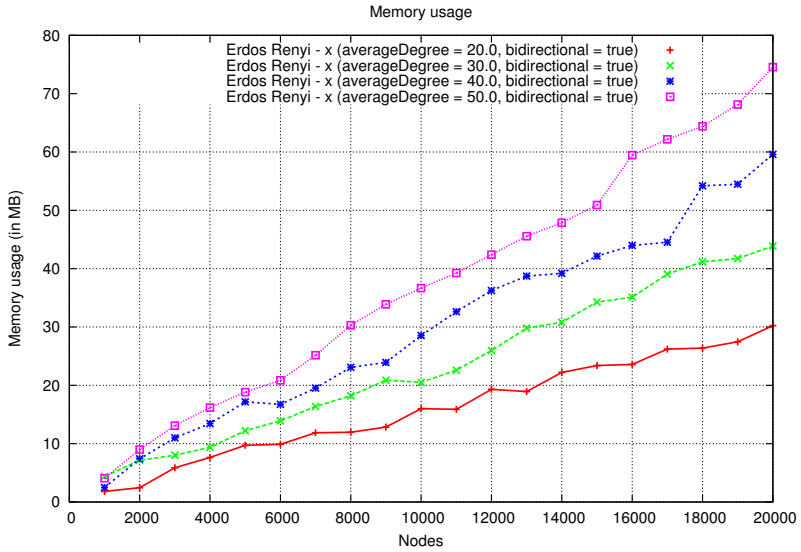


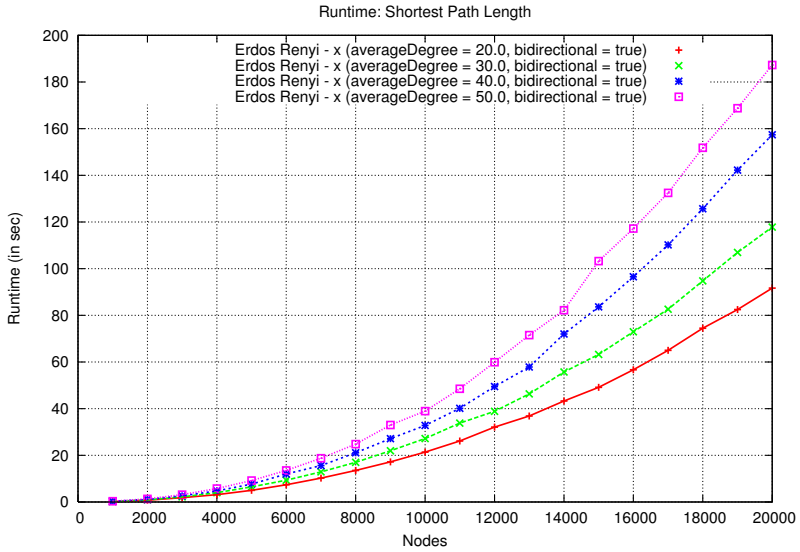




6. Evaluation

Runtime and Memory Usage Evaluation



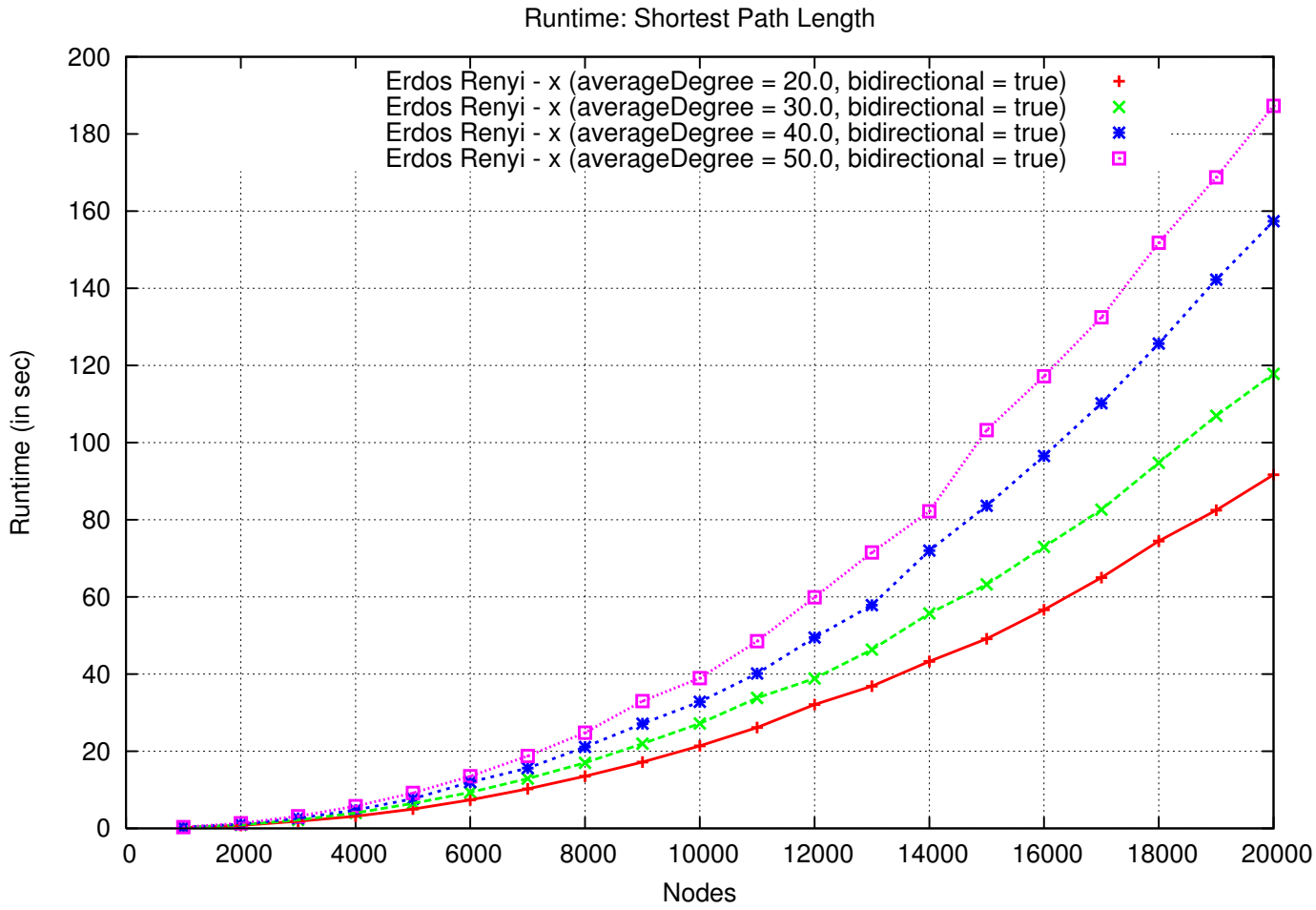


Setup

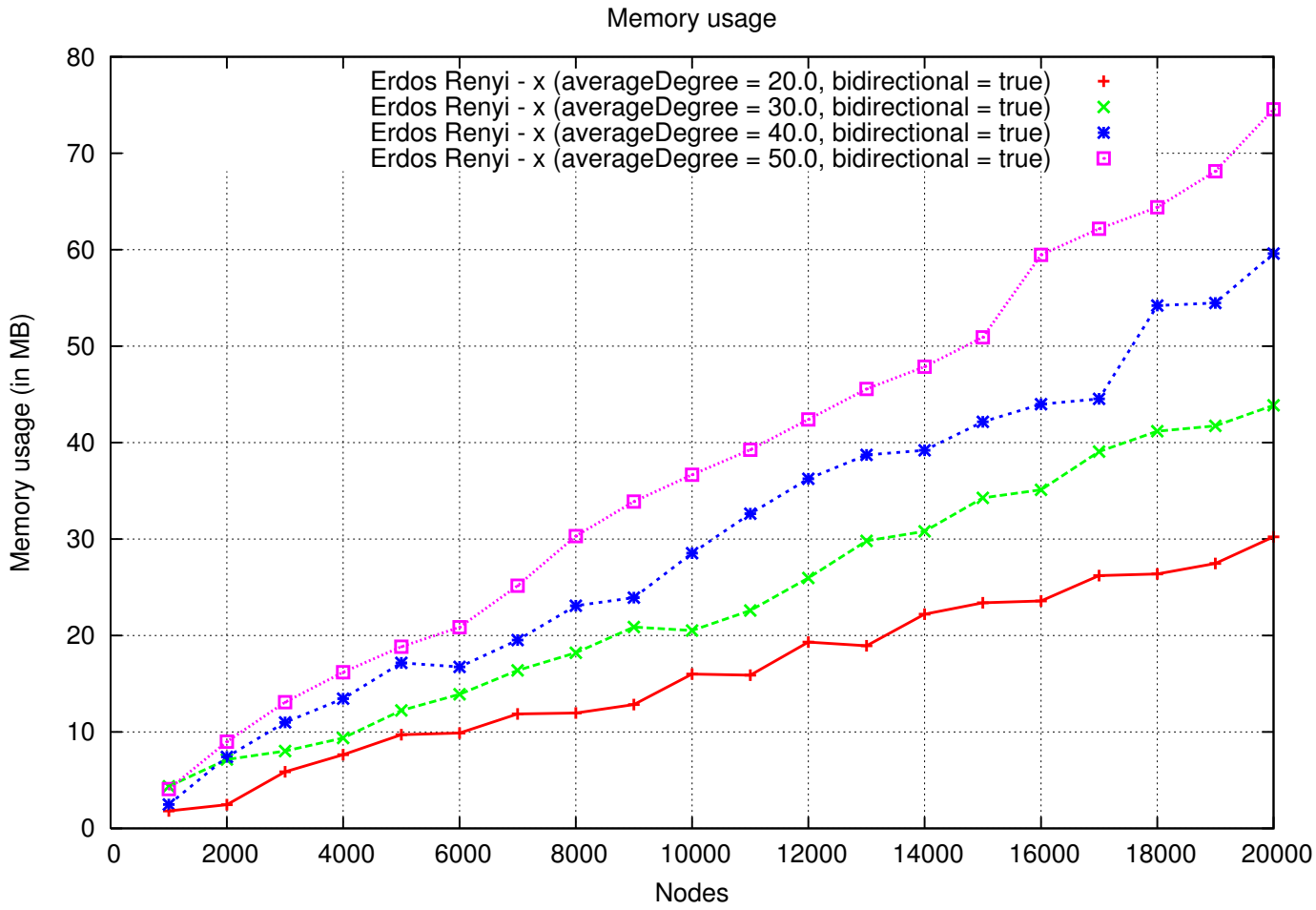
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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)
Runtime (in sec)
Nodes
0
20
40
60
80
100
120
140
160
180
200
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

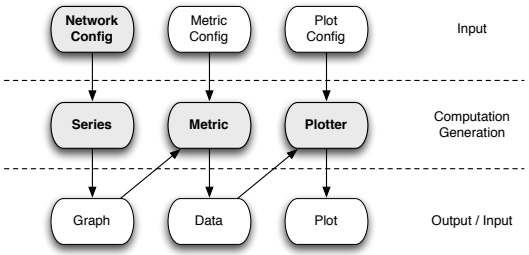
Memory Usage

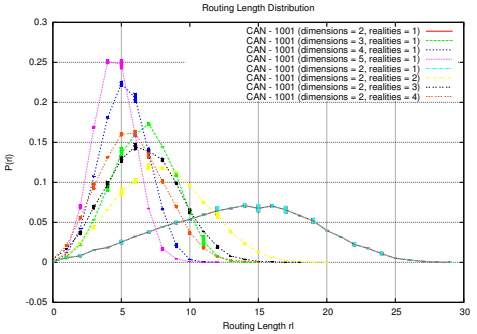


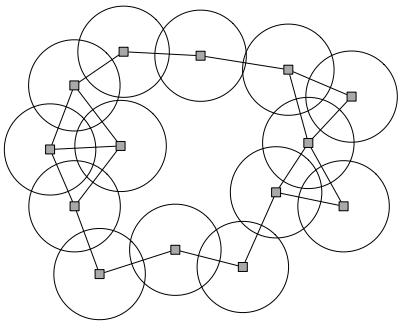
7. Conclusion and Outlook

Planned Extensions to GTNA

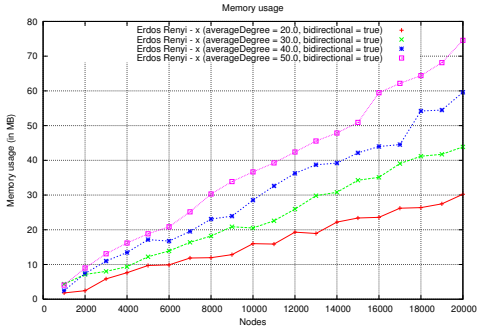
GTNA
Graph-Theoretic Network Analyzer







```
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);
     }
}
```



Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT

- Implementing additional metrics
- Providing Internet topology g
- Add dynamics to snapshot creation
- Graphical user interface

Motifs
Role-to-role connectivity profiles

PFP - Positive-Feedback Preference model
PARG - Parallel Addition and Reqiring Growth model

Different procedures for topology generation (join, crash, leave)

Configuration
Plot-Organization

GTNA

A Framework for the Graph-Theoretic Network Analysis

Benjamin Schiller Dirk Bradler Immanuel Schweizer
Max Mühlhäuser Thorsten Strufe

TECHNISCHE UNIVERSITÄT DARMSTADT

GTNA

Graph-Theoretic Network Analyzer

http://tk.informatik.tu-darmstadt.de ➔ Research ➔ Smart Civil Security ➔ GTNA

schiller@cs.tu-darmstadt.de