DYNAMIC NETWORK ANALYZER

Series Generation Instruction Manual

René Wilmes

25.04.2015

Dresden University of Technology

Contents

Chapter 1

Introduction

The DNA – Dynamic Network Analyzer is a framework for graph-theoretic Analysis of Dynamic Networks. It offers sophisticated ways to generate and analyze graphs. The built-in Plotting- and LaTeX-output features allow to visualize and compare data by generating LaTeX-documents containing data and plots.

This manual will show how DNA can be used to generate and analyze graphs. It covers the most important aspects of graph generation using DNA including what options and parameters may be utilized in order to customize the outcome and reach certain results. Examples are going to be shown to give the user an insight on the impact of certain customizations.

Chapter 2

Series-Generation in DNA

This chapter will deal with the general series-generation mechanisms in DNA and how to use them. Different approaches and configurations will be shown on examples to give a quick and easy insight into DNA usage.

2.1 General

The generation process uses a GraphGenerator to generate the graph and a BatchGenerator to compile graph changes into batches. DNA offers a variety of different Graph- and BatchGenerators for different graphs and purposes. Due to the nature of dynamic graphs, metrics can either calculate their values based on initial information and additional updates or do a full recompute after each update. During generation DNA will store statistics, runtimes and computed data in series objects. A series contains one or more runs, each of which represent one separate simulation. A run is divided into several batches, which contain statistics, runtimes and metric datas, see 2.1. The *aggr*-directory contains aggregated data of all runs contained in the specific series. For more details on the DNA architecture and theoretical background check the DNA paper. TODO: Citation.

2.2 Getting started

The usual workflow is pretty simple. First one initializes a GraphGenerator, Batch-Generator and an Array of metrics. Then a Series object is initialized with these inputs. The second step is the generation itself, which is also quite simple. Each Series-objects have several generation-methods for different purposes. The example in 2.2 shows how one could generate a random, undirected graph with initially 100 nodes and 300 edges. With each batch 5 nodes and 20 edges are randomly being added while also 15 edges get removed. The metrics *DegreeDistributionR* (recomputed) and *UndirectedClusteringCoefficientU* (updated) are chosen to be computed.

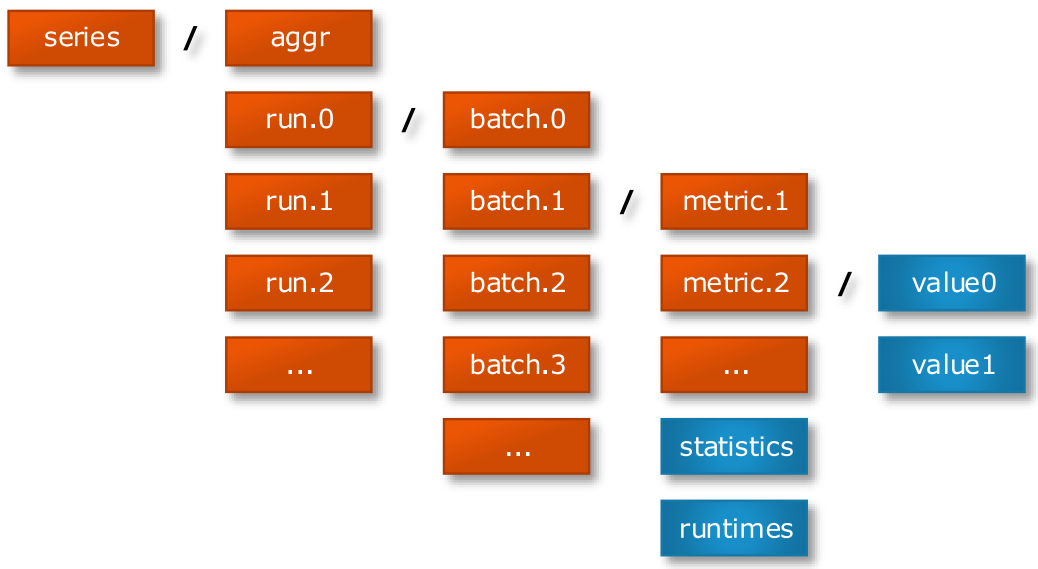

Figure 2.1 – Filesystem structure for storing the results of a series.

```
public static void main(String[] args) throws AggregationException,
    IOException, MetricNotApplicableException,
    InterruptedException {
  String seriesDir = dir + "series/";

  // initialization, 100 nodes and 300 edges
  GraphGenerator gg1 = new RandomGraph(GDS.undirected, 100, 300);

  // 5 nodes-added, 0 nodes-removed, 20 edges-added, 15 edges-removed
  BatchGenerator bg1 = new RandomBatch(5, 0, 20, 15);

  // choose metrics
  Metric[] m1 = new Metric[] { new DegreeDistributionR(),
      new UndirectedClusteringCoefficientU() };

  // create series and generate it
  Series s = new Series(gg1, bg1, m1, seriesDir, "series");
  SeriesData sd = s.generate(1, 100);
}
```

Figure 2.2 – Simple generation example.