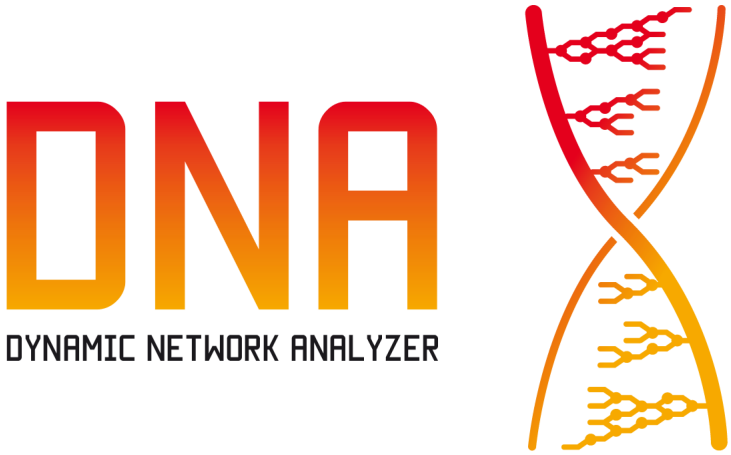

Series Generation
Instruction Manual

René Wilmes

28.04.2015

Dresden University of Technology

Contents

1 Introduction **1**

2 Series-Generation in DNA **2**
2.1 General 2
2.2 Getting started 2
2.3 Auto-generated extra-values 3
2.4 Zip-Modes 3
2.4.1 Batches 4
2.4.2 Runs 4

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, BatchGenerator and an array of metrics. Then a Series object is initialized with these inputs. The second step is the generation itself: Each Series-object has 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 are being removed. The metrics *DegreeDistributionR* (recomputed) and *UndirectedClusteringCoefficientU* (updated) are chosen to be computed.

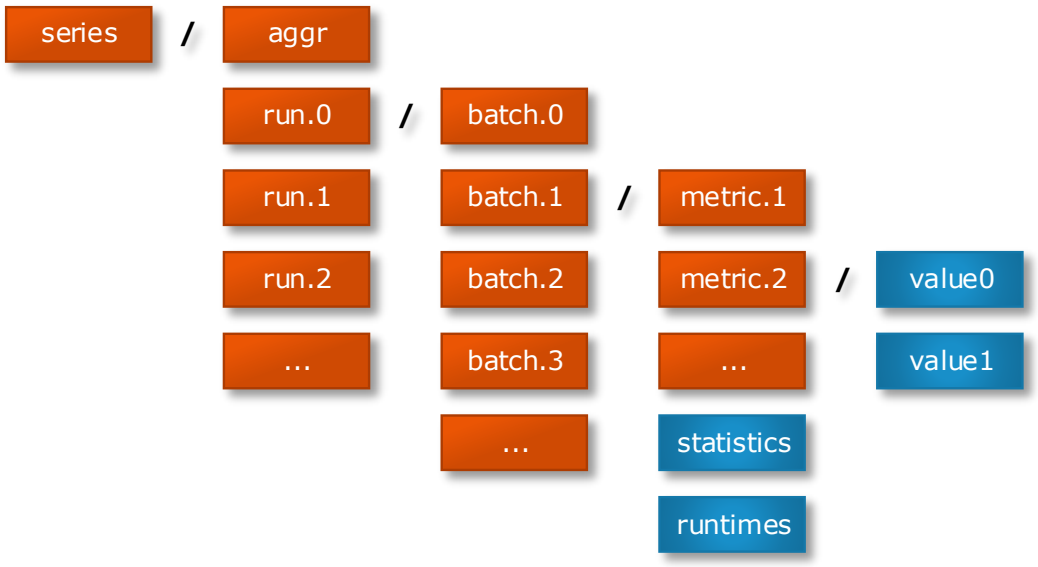

Figure 2.1 – Filesystem structure for storing the results of a series.

2.3 Auto-generated extra-values

The DNA is able to compute additional extra values for distributions and nodevaluelists. This is enabled by default and can be finely configured in the *settings.properties* configuration file. These include *minimum*, *maximum*, *median*, *average* and for certain distributions *binsize* and *denominator*. Consider the following example:

The metric *DegreeDistributionR* calculates the values *degreeMin*, *degreeMax* aswell as the distribution *DegreeDistribution*. DNA will automatically generate the additional values *DegreeDistribution_AVG*, *DegreeDistribution_MAX*, *DegreeDistribution_MED* and *DegreeDistribution_MIN*. Note: Because the *DegreeDistribution* is an int-distribution it is also possible to generate an extra value *DegreeDistribution_DENOMINATOR*, however this is disabled by default.

2.4 Zip-Modes

The generation of series with lots of runs, batches and metrics can lead to a lot of single files on the filesystem. In order to reduce the amount of files aswell as the overall disk space allocation it is possible to enable zip-mode. Note that the use of zip-files leads to a higher cpu load which will result in increased runtimes. The zip-modes can either be enabled via the *settings.properties* configuration file or during runtime with the config-key *GENERATION_AS_ZIP*. Zips are disabled by default:

```
Config.overwrite("GENERATION_AS_ZIP", "none");
```

```
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");

  // generate 2 runs with 100 batches each
  SeriesData sd = s.generate(2, 100);
}
```

Figure 2.2 – Simple generation example.

2.4.1 Batches

In zipped-batch mode every batch will be written and read as a single zip-file. May be enabled via the following command:

```
Config.overwrite("GENERATION_AS_ZIP", "batches");
```

2.4.2 Runs

In zipped-run mode every run will be written and read as a single zip-file. Note that this greatly reduces the amount of files (one for each run and an additional for the aggregation) but also greatly increases cpu load, because the zip-file has to be accessed for each read and write operation.

```
Config.overwrite("GENERATION_AS_ZIP", "runs");
```