12-2018

Planar Graph Generation with Application to Water Distribution Networks

Varsha Chauhan
Clemson University, varsha.chauhan1190@gmail.com

Planar Graph Generation With Application To Water Distribution Networks

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

by
Varsha Chauhan
December 2018

Abstract

The study of network representations of physical, biological, and social phenomena can help us better understand their structure and functional dynamics as well as formulate predictive models of these phenomena. However, in some applications there is a deficiency in real-world data-sets for research purposes due to such reasons as the data sensitivity and high costs for data retrieval. Research related to water distribution networks often relies on synthetic data because the real-world is data is not publicly available due to the sensitivity towards theft and misuse.

An important characteristic of water distribution systems is that they can be embedded in a plane, therefore to simulate these system we need realistic networks which are also planar. Currently available synthetic network generators can generate networks that exhibit realism but the planarity is not guaranteed. On the other hand, existing water network generators do not guarantee similarity with the input network and do not scale. In this thesis, we present a flexible method to generate realistic water distribution networks with optimized network parameters such as pipe and tank diameters, tank minimum and maximum levels, and pump sizes. Our model consists of three stages. First, we generate a realistic planar graph from a known water network using the multi-scale randomized editing. Next, we add physical water network characteristics such as pumps, pipes, tanks, and reservoirs to the obtained topology to generate a realistic synthetic water distribution system that can be used for simulation. Finally, we optimize the operational parameters using EPANet simulation tool and multi-objective optimization solver to generate a network with maximum resilience and minimum cost.

Acknowledgments

I take this opportunity to express my profound gratitude and deep regard to my advisor Dr. Ilya Safro for his exemplary guidance, monitoring and encouragement throughout the course of this thesis.

Besides, my advisor I would like to thank my Co-advisor Dr. Kalyan Piratla and and his team Ahmad Momeni and Abdulrahman Bin Mahmoud for their collaboration and help throughout the project. Without their valuable contribution and assistance this project would not have been possible.

I would also like to thank Dr. Alexander Herzog for his support, encouragement and agreeing to be a part of my defense commmittee.

This material is based upon work supported by the National Science Foundation under grants #1522751 and #1745300.

Table of Contents

Title Page . . . . **i**

Abstract . . . . **ii**

Acknowledgments . . . . **iii**

List of Tables . . . . **vi**

List of Figures . . . . **vii**

1 Introduction **1**
1.1 Our Contribution 2

2 Background and Literature Review **4**
2.1 Network Generation Algorithms 4
2.2 Planar Network Generators 8
2.3 Domain Specific Network generators: Water Distribution System Generator 10
2.4 Optimization of Water Distribution Network Design 10

3 Generation of Planar Graph **12**
3.1 Multiscale Planar Graph Generation 12
3.2 Computational Experiments 19

4 Generating WDS from Planar Topology **30**
4.1 Assigning coordinates 30
4.2 Reservoirs 31
4.3 Tanks 32
4.4 Junctions 32
4.5 Pumps 33
4.6 Pipes 33
4.7 Experiments and Results 34

5 Optimizing Water Distribution System Components **39**
5.1 Objectives 39
5.2 Constraint 40
5.3 Decision Variables 41
5.4 Experiments and Results 42

6 Conclusions **44**

Bibliography **46**

List of Tables

4.1 Input Networks 34
4.2 Summary of WDS Components in synthetic WDS generated from Net1_Rossman Network 35
4.3 Summary of WDS Components in synthetic WDS generated from d-town Network 35
4.4 Summary of WDS Components in synthetic WDS generated from Charleston Network 36

5.1 Diameter of Pipe vs Cost 40
5.2 Volume of Tank vs Cost 40
5.3 Decision Variables 41
5.4 Values for Pump Curve 42
5.5 Computational Results for Optimization Solver for synthetic network generated by rescaling Net1_Rossman Network 42

List of Figures

3.1 The V-model for multiscale planar network generation. The original input planar network is coarsened to generate a hierarchy of coarse networks, the process is then reversed generating fine-level networks. The number of level (here 5) depends on the size of input network or the user input. 14
3.2 Computational results on performance of planar Musketeer on power grid graph opsahl-powergrid with 4941 nodes and 6211 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree. 21
3.3 Computational results on performance of planar Musketeer on power grid graph opsahl-powergrid with 4941 nodes and 6211 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality. 22
3.4 Computational results on performance of planar Musketeer on finite-element graph with 4704 nodes and 13427 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree. 23
3.5 Computational results on performance of planar Musketeer on finite-element graph with 4704 nodes and 13427 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality. 24
3.6 Computational results on performance of planar Musketeer on real water network with 407 nodes and 459 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree. 24
3.7 Computational results on performance of planar Musketeer on real water network with 407 nodes and 459 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality 25
3.8 Computational results on performance of planar Musketeer on road network from roadNet-TX with 2001 nodes and 2957 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree. 25
3.9 Computational results on performance of planar Musketeer on road network from roadNet-TX with 2001 nodes and 2957 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality. 26

3.10 Visualization of a road network from roadNet-TX with 2001 nodes and 2957 edges rescaled to at least 5900 nodes and 7000 edges. 27
3.11 Computational results for road network from roadNet-TX with 2001 nodes and 2957 edges rescaled to at least 6000 nodes and 6500 edges. 28
3.12 Computational results for road network real water network with 407 nodes and 459 edges rescaled to at least 1098 nodes and 1500 edges. 28
3.13 Computational results for finite-element graph with 4704 nodes and 13427 edges rescaled to at least 12700 nodes and 36000 edges. 29
3.14 Computational results for power grid graph opsahl-powergrid with with 4941 nodes and 6211 edges rescaled to at least 16500 nodes and 27000 edges. . . 29

4.1 Visualization of input network Net1_Rossman and generated synthetic networks on EPANet software. (a) Original Input Network (b) Synthetic WDS rescaled to at least 7 times (c) Synthetic WDS rescaled to at least 7 times (b) Synthetic WDS rescaled to atleast 14 (d) Synthetic WDS rescaled to atleast 22. 36
4.2 Visualization of input network d-town and generated synthetic networks on EPANet software. (a) Original Input Network (b) Synthetic WDS by replication (c) Synthetic WDS rescaled to at least 2 times. 37
4.3 Visualization of input network Charleston Network and generated synthetic networks on EPANet software. (a) Original Input Network (b) Synthetic WDS by replication (c) Synthetic WDS rescaled to at least 2 times. 38

5.1 Visualization of solution networks for Optimization algorithm on EPANet software with link flow and junction head values after 24 hr simulation. (a) Synthetic WDS with 76 pipes and 1 pump, tank and reservoir (b) Synthetic WDS with 169 pipes, 2 pumps and tanks and 1 reservoir 43

Chapter 1

Introduction

A network is a representation of a set of entities and the relationships between them. The network paradigm is often used to represent physical, biological, engineered and social systems [28]. The study of network representations of physical, biological, and social phenomena can help us better understand their structure and functional dynamics as well as formulate predictive models of these phenomena. However, in some applications there is a deficiency of real-world data-sets for research purposes due to the reasons such as data sensitivity or high costs for data retrieval.

The problem of data scarcity can be tackled by using synthetic data which can mimic both the properties and diversity of real world networks. Such synthetic data can be used for simulations, analysis, and performance/quality verification of algorithms - a crucial task in algorithm engineering. The importance of synthetic networks is well known in the field of Water Distribution Systems research because the real-world data is not publicly available due to the sensitivity [31] of this data towards theft and misuse.

The network topology of water networks undoubtedly plays a key role in the generation of synthesized Water Distribution System (WDS). In particular, an important characteristic of these networks is that they can be (almost fully in most cases) embedded in a plane, i.e. are planar graphs. The currently available planar graph generators generate planar random structure that lack the structural characteristics of real-world networks.

Therefore, the first step we take in this thesis is to model a realistic WDS is generating a realistic planar topology. However, a true representation of an infrastructure system requires not only the network topology, but also its physical and operational characteristics such as location and size of pipes, pumps, tanks and reservoirs. This is because the physical and operational features are crucial in the functioning of a WDS. Therefore, we next focus on introducing the physical design features such as pipes, pumps, tanks, reservoir etc. to the generated planar topology to generate a functional WDS. The generated WDS can successfully mimic properties and characteristics of real-world network and can be used for simulation.

One of the main goals of a WDS is to satisfy the demand of the network while minimizing operational cost and maximizing robustness. The WDS design is a discrete non-linear hard optimization problem that has attracted significant research attention over decades. In this thesis, we use genetic optimization model coupled with hydraulic solver EPANet [38] to determine the physical feasibility of the design in terms of pressure and flow requirements and while minimizing the operational cost and maximizing resilience of the generated WDS.

1.1 Our Contribution

In this thesis, we introduce a toolbox of algorithms to generate a synthetic Water Distribution Network with physical characteristics such as pipes, pumps, junctions, tanks, and reservoirs with optimized operational parameters such as pipe and tank diameters, pump sizes, tank minimum and maximum levels etc. Our toolbox has 3 main algorithms, where each tool can be run independently as well as in conjunction with other.

1.1.1 Generation of Planar Topology

In the first step, we generate realistic planar replicas of a known planar graph that can be rescaled to much larger graphs. The method follows the multi-scale editing approach

[15] in which a given graph is projected into a hierarchy of its coarsened representations (coarse graphs) that are then perturbed by edits at various scales of coarseness in the hierarchy. The method preserves the structural properties including the planarity with *controllable* bias, while introducing realistic variability at multiple scales of coarseness. Because the method belongs to the family of multiscale editing approaches, it generates planar graphs that attempt to replicate properties of the original graph at all levels of its coarse-grained resolutions.

1.1.2 Generation of WDS

In this phase we introduce physical and operational water network design features to generate a functional WDS network from a known planar graph and real-world WDS. In this process we rely on the input WDS network and various specific characteristics that are often found in Water networks such as the location of tanks and reservoirs and allocation of pumps in Water Network to assign the physical features to the planar topology generated in previous step.

1.1.3 Optimizing the Water Distribution System Components

Finally, we use multi-objective solver to optimize the operational cost and resilience of the generated WDS. The input to this step is a known water network which can be the output of previous step or an independent network. We use genetic optimization solver to generate a optimized network in terms of physical design such as diameter of pumps and tanks as well as operation to meet the demand of the system during 24 hour operation period while saving energy and cost of pumps operation.

The first part of this thesis is based on our paper: **Varsha Chauhan, Alexander Gutfraind, and Ilya Safro. Multiscale planar graph generation. arXiv preprint arXiv:1802.09617, 2018.**

Chapter 2

Background and Literature Review

2.1 Network Generation Algorithms

The field of network science and, in particular, network synthesis is vast and cannot be comprehensively reviewed here. Hence, we focus on several particularly illuminating approaches for modeling realistic networks that presumably may be applied as or changed to the first step in realistic planar graph generator. (In contrast to the different versions of random planar graph generators, there is an obvious lack [3] of planar graph generators that generate graphs *that are similar to the original planar graph.* This is the reason, why practitioners and decision makers use other graph generators in combination with planarization postprocessing to generate planar and hopefully realistic planar graphs. This is also a reason for our comparison with these algorithms reinforced with planarization in the next sections.) These approaches fall into two categories, namely, generative models and editing models.

2.1.1 Generative models

Generative models typically construct a network starting with an empty or small seed network and then iteratively add network elements (such as nodes and edges) to match some properties of a network that has to be replicated. These algorithms attempt to preserve the real network properties over the evolution and growth of the synthetic network. Important examples of generative models are the following.

ERGM Exponential Random Graph Models (ERGM) [18] are a class of statistical models, earlier called p-star models, that are popular in the study of large-scale social networks. To build a network, the ERGM first estimates certain parameters by fitting an observed social network and then constructs new networks by sampling from the estimated distribution. For example, in the Bernoulli and Erds-Rnyi ERGM models which generate homogeneous networks, the parameter space is based on common probabilities for each added connection, whereas the Chung-Lu ERGM model [1] for large random graph with given degree distribution, it uses two parameters, namely, α which is logarithmic of number of nodes of degree 1, and β which is log-log rate of decrease of the number of nodes with a given degree and is given by $|\{v|deg(v) = x\}| = y = \frac{e^{\alpha}}{x^{\beta}}$. The model can generate large graphs which depict some of the behaviors of massive realistic graphs and also predict the size and number of large components in the graph from the given values of α and β. ERGM models are successful in generating social networks and exhibit realistic degree distributions and small world structures, but do not give any planarity guarantees, and normally violate planarity. While potentially, this model could serve as the first step in planar network generation (the planarity could be one of the properties or it can be applied with subsequent planarization of synthesized network), we emphasize that it is extremely slow and cannot be applied even on medium size networks, so we cannot experiment with it and compare to our generator.

BTER Block Two-Level Erds-Rnyi model (BTER) [45] is based on the idea that a network contains communities that are Erds-Rnyi graphs in which each pair of vertices is independently connected with probability p. BTER graphs contain dense Erds-Rnyi communities that are found in real-world networks. The algorithm is two-phased. In the first phase a collection of blocks or Erds-Rnyi communities with specified degree distribution is created. Then the blocks made interconnected and excess degree nodes are removed based on

Chung-Lu (CL) model [2] such that each subnetwork is well modeled by CL. BTER has been shown to model realistically a variety of network properties, but as with ERGM, it gives no guarantees of planarity. Also, whether communities in (almost) planar networks have hierarchical and connectedness structure similar to BTER model or not is not explored.

RMAT and Stochastic Kronecker Graphs The Recursive Matrix graph generator introduced by Chakrabarti et al. [7] and its extensions AutoMAT-fast [7] can generate large-scale complex realistic networks. The generator is based on a recursive algorithm that operates on the adjacency matrix of the graph by dividing it into four equal-sized partitions and distributing edges to each partition based on fitting a set of parameters.
The Stochastic Kronecker Graphs (SKG) [23] extends the methods of RMAT. Similarly to RMAT it is a recursive model, which starts with a small initiator matrix and recursively produces large graphs by applying Kronecker products. SKG can be interpreted as network which is a hierarchy of communities which grow recursively to create copies of themselves and every node has unique set of attributes values. The model can generate graphs with static patterns such as degree distribution as well as temporal patterns such as diameter over time. As before, planarity is not guaranteed as well as the community structure similarity with real-world networks that have one.

Multifractal Network Generator In 2010, Pallaa et al. [34] introduced the multifractal network generator which can generate realistic networks with specified statistical features. The method starts with defining a generative measure on a single fractal or unit square and calculating link probability. The network is then scaled to the infinite limit by recursively dividing the fractal into a number of rectangles and introducing connections between them based on the link probability. Although this method was able to generate small scale realistic graphs the recursive method was slow for large complex networks. It is unknown if the generated networks can be constructed to have planar or quasi-planar structure, but the random nature of the construction suggests that planarity would be uncommon even in small graphs. However, the backbone networks generated by this model could be planar and thus possibly relevant to some infrastructure networks (for example, see major gas pipes in [27]). Unfortunately, these networks are also very far from being similar to infrastructures which makes the comparison impossible.

2.1.2 Editing models

The editing models approach starts with a given (real or empirical) network and controllably introduces random changes to its elements (such as nodes and edges) until the network becomes sufficiently different from the original network. These changes are designed to introduce variability while preserving key structural properties during the random editing. Such methods are a promising direction for a relatively more realistic modeling of networks, and that includes properties such as planarity or near-planarity.

Edge-swapping The edge-swapping method [51, 35]is perhaps the first important algorithm in the class of editing models, and it is based on the insight that the degree distribution of a graph is preserved under a chain of edge-swapping operations. Such a chain of edge swaps can even asymptotically achieve important mixing properties giving high variability. Despite these successes, edge-swapping operations can be very disruptive to planarity and other global properties of the graph, and there are no good post-selection methods for achieving planarity.

Multiscale Network Generation In [15], several of us proposed a strategy termed MUSKETEER (Multiscale Entropic Network Generator) for realistic graph generation. The main idea was based on the observation that the properties of real networks that should be preserved during generation are not only those measured at the finest resolution but also those that can be measured at the coarse resolutions. Multiscale generation leverages coarsening schemes used in highly-accurate multiscale solvers for combinatorial optimization such as linear arrangement, compression and partitioning [36, 17, 42, 40, 43]. In such coarsening schemes, nodes in a network are assigned into aggregates (or, typically, very small communities) which are themselves parts of larger aggregates and so on in a hierarchical manner. The algorithm was successful in generating a number of replicas for several real-world original networks, but did not guarantee planarity. This paper continues this line of research and offers an implementation of the multiscale strategy that actually produces planar networks.

ReCoN Staudt et al. [49] later used principles similar to those of multiscale method and developed a fast network generator that could generate large scale replicas of real complex network that are structurally similar to original network. Instead of leveraging multiscale coarsening schemes, ReCoN generated synthetic networks by randomizing the edges between communities which were detected by the community detection methods while keeping the same degrees of nodes. ReCoN is built on top of the LFR generator implemented in [48].

2.2 Planar Network Generators

Planar graphs are the class of graphs that can be embedded in a two-dimensional plane without edge crossings. Designing efficient algorithms for planar graphs is an important subfield in the area of algorithm development and optimization [25]. From the practical perspective, the planarity is also an important characteristic of many physical networks such as roads, utilities, water distribution systems, and some circuit designs. Many of these networks are, in fact, almost planar, that is, one can remove typically small fraction of edges to make them exactly planar. Planar networks with underling graphs have attracted a lot of attention since a landmark paper by Tutte [52]. Most of the research was dedicated on the study of structural properties (including their generation) of random planar graphs or uniform random planar graphs such as triangulations, and meshes. However, the currently available planar graph generators usually generate uniform random graphs by interpolation of planar subgraphs or generate planar subgraphs of a non-planar graph. Unfortunately, they are very far from being practically important for such tasks as generating graphs underlying infrastructure networks since they fail to present most other properties that are viewed as significant in this area, such as the degree distribution, the community structure and others. Some important available planar graph generators are discussed below.

Plantri and Fullgen software. Plantri [5] can generate triangulations, quadrangulations, and convex polytopes using recursive algorithm which is efficient and fast. Fullgen [4] generates fullereness which are planar cubic graphs with 5 or 6 faces. The important characteristic of this software is that it generates only one graph as output from a family of isomorphic graphs saving the space needed to store them. The software also offers the user the option to restrict adjacent pentagons using an input parameter.

Markov Chain Planar Graph Generator. This algorithm was proposed by Denise et al. [11] and is based on Markov Chain that generates planar subgraphs from a non-planar graph. The algorithm defines a Markov Chain on the state space of all subgraphs of the original graph and transitions as follows. If an edge exists in space, it is deleted. If it is not present it is added in case it maintains planarity otherwise it is discarded. The method can successfully generate a planar subgraph in polynomial time.

Delaunay Triangulation and refinement method. This method has been used widely used by researchers to generate mesh networks. In [46] Shewchuk, presented an implementation of 2-Dimensional constraint Delaunay triangulation and Ruppert's [39] Delaunay refinement algorithm for mesh generation.

Geometric graphs. Gilbert [14] proposed a model to construct random plane networks by first selecting points in infinite plane based on Poisson process with density D and then connecting points based on distance R from each other. The random geometric graphs closely represent the graphs generated by percolation process through various porous materials and therefore these graphs are extensively utilized by physicists to study continuum percolation models. Random geometric graphs also have application in communication networks [3].

Planar ErdosRenyi graph. In 1959, Erdos and Renyi [12] introduced a method to generate a random graph with N nodes and m edges by connecting the edges randomly with independent probability p. The Erdos-Renyi planar graph generator generates random planar graph with uniform probability [11] by rejecting the non planar edges thereby preserving planarity [11, 13, 24, 3].

2.3 Domain Specific Network generators: Water Distribution System Generator

WaterNetGen WaterNetGen [26] developed by Murano et al. is an interactive application developed as an extension to well-known WDS optimization tool EPANET [38], which could generate small as well as large network topologies by interconnecting subsystems. WaterNetGen offers an interactive interface through which users can assign WDS components such as pipes, pumps, tanks, reservoirs etc. to the generated topology. The additional parameters such as elevation of tanks or nodes and pipe sizes can also be defined by the user.

Water Distribution System Designer In 2013, Sitzenfrei [47] developed a software package, Water Distribution System Designer that can generate realistic synthetic water networks using GIS data such as population density, housing density and elevation as input data. Sitzenfrei introduced the newly developed graph concatenation approach (GCA) to generate layout of WDSs. The model concatenates different blocks from a database while meeting the requirements of the underlying GIS data. The software provided an interactive GUI to modify all the design parameters by user. The software could generate synthetic WDSs that reflected geometrical properties such as pipe length and diameter of a real-world WDSs.

2.4 Optimization of Water Distribution Network Design

The WDS design optimization is a discrete non-linear NP hard computational problem. Due to the limitation of computational algorithms the problem is tackled using stochastic approach such as genetic algorithm, simulated annealing, shuffled frog-leaping, tabu search algorithm etc.

Chapter 3

Generation of Planar Graph

3.1 Multiscale Planar Graph Generation

Multiscale network generation (MNG) introduced in [15] is an editing model that generates realistic networks. The proposed multiscale planar graph generator follows the main ideas of MNG and makes them applicable on planar graphs.

MNG follows a multilevel coarsening/uncoarsening scheme shown in Figure 3.1. We start with an input graph G and generate a hierarchy of next coarser graphs, $G_0, G_1, ..., G_k$, where k is the number of coarsest level. The number of coarsened levels depends on the structure and size of G. If it is too small or too dense at some level then the hierarchy construction is terminated (i.e., the coarsest level is reached). The definition of coarsened level is generic and based on the weighted aggregation method for combinatorial optimization problems [40, 36, 41, 22]. Currently, it does not depend on the application predefined aggregates in the network such as knowledge about real communities. However, this process can be adjusted as we did in [50]. In order to generate a synthetic graph, we introduce a series of local randomizations at different levels whose numbers can be specified by user input. If user is interested in only local changes without destroying the global structure of the network, only fine levels are specified for randomizations. Otherwise, any realistic changes in global structure will require randomizations at coarse levels. During the uncoarsening, these randomizations are carried forward to the next finer level in the hierarchy. In Algorithm 1, we describe the sequence of steps in generating planar graph. We will now discuss each phase and notation in detail and our approach to generate planar graphs using multiscale method.

Algorithm 1 Multiscale Planar Network Generator MPNG(G_i)

```
1: if $G_i$ is not small or too dense or perturbations are required for $G_i$ at level $i$ by user then
2:     $G_{i+1} \leftarrow$ create aggregated network from $G_i$ (see Alg. 2)
3:     $G^g_{i+1} \leftarrow$ MPNG($G_{i+1}$)          ▷ Return coarser edited network from recursive call
4:     $G^{d'}_i \leftarrow$ interpolateUneditedAggregates from $G^g_{i+1}$
5:     $G^d_i \leftarrow$ interpolateEditedAggregates($G^g_{i+1}$,$G^{d'}_{i+1}$)
6: end if
7: $Q_i \leftarrow$ measure properties of $G_i$
8: $Q^g_i \leftarrow$ editing $G^d_i$ preserving $Q_i$
9: Return $G^g_i$
```

3.1.1 Coarsening

Since the input graph G_0 is planar, the coarsened graphs G_i are also planar, so we follow the same coarsening scheme as that in the original MNG. Algorithm 2 describes the steps involved for generating coarse level graph G_{i+1} from G_i.

Algorithm 2 Coarsening(G_i)

```
1: if $G_i$ is not the coarsest graph then
2:     Find set of seeds ($C$) for coarse network $G_{i+1}$
3:     Find fine-level nodes that belong to each aggregate
4:     Calculate weight of edges connecting aggregates and weights of coarse nodes
5:     Return $G_{i+1}$
6: end if
```

We start with finding set of seeds C and its complement fine-level nodes F which is based on two rules, first, nodes with high volume and connectivity (i.e., major aggregates) are more likely to be included in C and the nodes in F should be “strongly” coupled to enough neighbors in C. To generate coarse level nodes for G_{i+1} we begin with $C = \emptyset$ and

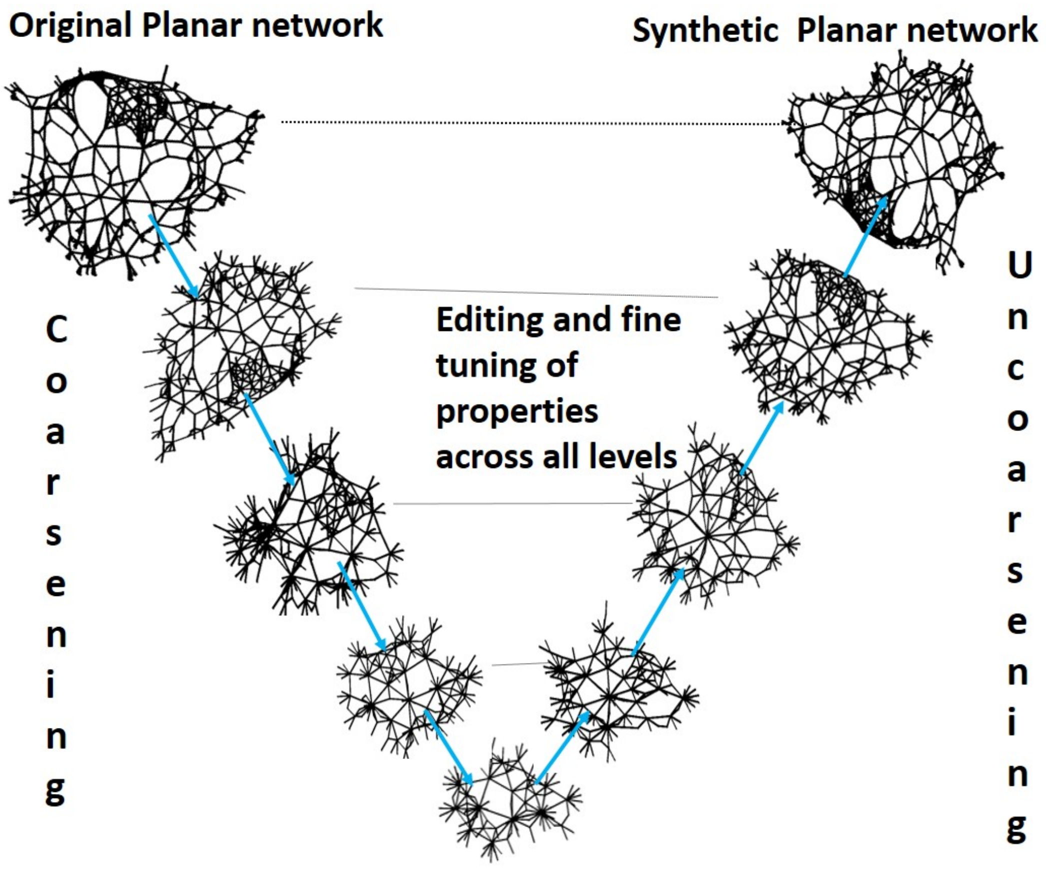


Figure 3.1: The V-model for multiscale planar network generation. The original input planar network is coarsened to generate a hierarchy of coarse networks, the process is then reversed generating fine-level networks. The number of level (here 5) depends on the size of input network or the user input.

$F = V_i$ where V_i is set of nodes in fine level G_i. Next, we iteratively transfer nodes from F to C, such that currently visited node $i \in F$ is added to C if it is not well connected to those already chosen to C [40]. The connection strength between nodes i and j is determined by means of normalized weight of edge ij with respect to C, namely, if node $i \in F$ is not connected strong enough to currently chosen C, i.e.,

$$\frac{\sum_{j \in C} w(ij)}{\sum_{j \in V} w(ij)} \leq \alpha, \tag{3.1}$$

then we move i to C. The connection strength is parametrized using threshold α which is in all experiments 0.5.

The final phase of coarsening is computing the connection strength between the coarse nodes. Here we define the algebraic multigrid interpolation matrix P of size $|V| \times |C|$ (for details see [40]) in which P_{ij} represents the likelihood of i to belong to the j^{th} aggregate. The Laplacian of the coarse graph G_{i+1}, L_{i+1}, can be calculated by the algebraic multigrid coarsening operator $L_{i+1} \leftarrow P^T L_i P$ where, L_i is the Laplacian of ith level graph, and

$$P_{ij} = \{ 1 \ , fori \in C, j = i0, otherwise. \tag{3.2}$$

The edge ij connecting two coarse nodes i and j, is assigned with the weight

$$\sum_{k \neq l} P_{ki} w(kl) P_{lj}$$

and the volume of the i^{th} coarse aggregate is $\sum_j v(j) P_{ji}$.

To this end the $(i+1)$th level graph is generated, and we can measure the properties of ith level graph and store them in Q_i. In general, this step is application dependent as in different applications the preserved properties may vary. Because, in planar graphs of infrastructures it is important to generate realistic path lengths (e.g., not to create shortcuts that connect distant regions in a graph), we are sampling using random walks the distribution of path lengths and shortcuts (second shortest distance between nodes) and store them in Q_i (see [15] for details).

3.1.2 Uncoarsening

Once the coarsest level is reached, we start the uncoarsening. During this process, at each level $i+1$ we choose nodes and edges to be edited (randomized while keeping some properties preserved), to generate edited network G^g_{i+1} at level $i+1$ and then project the newly created graph to generate the next finer level G^g_i. The projection is done in two steps. First, we interpolate the unedited aggregates (nodes and edges) in **interpolate-UneditedAggregates** (Step 3) from G^g_{i+1} to generate graph $G^{d'}_i$. This process is just a reverse interpolation of aggregates based on aggregation data stored in P during the coarsening phase, because the input network is planar the interpolation edges do not create crossing any crossing over edges. This helps in preserving structural properties of original input network, as after this step we have a subgraph $G^{d'}_i$ of original network coarsened at level $i+1$.

In the next step, we interpolate edited aggregates, by first interpolating nodes and adding edges that were trapped within aggregated (or coarse) nodes connecting the fine nodes, i.e., these are edges that connect fine nodes that are coarsened within of same coarse node. Next we interpolate edges in function **interpolateEditedAggrregates** to generate graph G^d_i by adding new edges to graph generated at step 3. The pseudocode for the function is presented in Algorithm 4. This interpolation is likely to introduce crossing over edges, therefore, when we add an edge ij to G^d_i, we check if the network is still planar. If it is not, the edge is discarded. If an edge is discarded, we perform several iterations and find an edge which is similar to the edge ij using properties stored in Q_i during coarsening in Algorithm 1.

After the interpolation is complete and we have a fine-level graph G^d_i, on which we introduce randomizations or editing (discussed below in detail) specified by the user at level i to generate a finer-level random planar network network G^g_i. The topology of the final network depends on the level at which the changes are introduced and the number

Algorithm 3 interpolateUnEditedAggregates (G_i)

```
1: $G_i^{d'}$ ← uncoarsen nodes from $G_{i+1}^g$ using data stored in $P^i$ during coarsening at level $G_i^d$
2: $G_i^d$ ← uncoarsen unedited edges $G_{i+1}^g$ using data stored in $P^i$ during coarsening at level $G_i^d$
3: $G_i^d$ ← interpolateUnEditedAggregates($G_{i+1}^g$ , $G_i^{d'}$)
4: Return $G_i^g$
```

Algorithm 4 interpolateEditedAggregates (G_{i+1}^g , $G_i^{d'}$)

```
1: $G_i^d$ ← uncoarsen nodes from $G_{i+1}^g$
2: $G_i^d$ ← uncoarsen edited edges $G_{i+1}^g$
3: $G_i^d$ ← interpolateEditedAggregates($G_{i+1}^g$ , $G_i^{d'}$)
4: Return $G_i^g$
```

of edited network elements both dependent on user input. At the coarsest level, every network element is an aggregate which interpolate of many network elements at fine level, a small change introduced at this level may generate high-entropy changes which are carried forward to the next fine level, whereas addition of an element at fine levels may introduce elements to the final synthetic network. In general, the changes introduced to deeper levels of aggregation, the more significant changes are introduced in the topology.

3.1.3 Editing

In the final phase we measure the properties of the generated graph G_i^d and compare with the properties of original graph G_i coarsened at level i which is stored in Q_i, thus preserving the local topological structure of the network and preventing addition of edges between nodes which were separated by long distance in original network at coarse level G_i stored in Q_i. We then use an editing process which introduces randomizations in the network to generate a synthetic network. This is a process of deleting and adding new edges both dependent on user input for level i (namely, how much randomization is requires in scale from 0 (no randomization) to 1 (everything is randomized)), however the network elements are carefully chosen based on the structure of unedited network at level i. The number of new edges introduced is dependent on the edge edit parameter. When we insert a new edge we preserve the structural properties of the original network in the coarse level.

In particular, we are interested in two properties, namely, second shortest path length distribution and planarity. The first property is measured and verified similar to previously introduced [15]. The second property is critical for planarity. If inserting the new edge makes the network non-planar we discard it and find an alternate edge that preserves the desired structural properties (in this case the first property) as well as planarity. Technically, it is done by verification of existence of Kuratowski subgraph after adding a new edge. This step is repeated until we find a non-crossing edge that preserves the planarity of the network and thus generating synthetic planar graph G_i^g at coarse level i.

3.1.3.1 Rescaling

Rescaling is a part of the editing phase in which we add new elements (edges and nodes) to the synthetic network. The scaling factor and the coarsened level at which the network is rescaled is controlled by node growth parameter which is provided as an input from the user depending on the user requirement. In general, rescaling at coarsest levels will preserve the local structure of the input network, i.e. the generated network will have increased number of communities whereas rescaling at finer levels will increase the size of communities. The scaling factor ranges from 0 to 1 which decides the percentage of new nodes that are to be added at the level i. This is a two step process, first we introduce a new node (u) and connect to an existing node (v) in the network deleting an existing edge from v to restore the degree of node v. In the next step, we find neighbors of v iteratively over increasing distance from v and connect the newly added node u to the neighbors of node v thus preserving the local topological structure of the network at coarse level G_i stored in Q_i. The process is terminated when the desired number of network elements are added and a rescaled network G_i^g is generated at coarse level i.

3.2 Computational Experiments

In this section we show the computational results summarizing the performance of our multiscale planar network generator in replicating the original and also generating rescaled networks. To test the variability of the generator we used real-world infrastructure networks such as water distribution system, power grid and road network that are either planar or have very few edge crossings that we removed. We used the water network from "The Battle of the Water Networks II" [32] and for road network we used a sub graph of Texas [21] road network from [20]. We also used a finite element large planar sub-graph of a finite-element graph from Boeing collection in [9]. In case of the power grid [20] which was not planar, we generated approximate maximal planar subgraph of the network using Open Graph Drawing Framework (OGDF) [8] to be used as an input to our algorithm.

3.2.1 Replication

We tested our implementation on three sets of parameters, namely, "Musketeer Coarse" (at only two coarsest levels 5% randomizations are allowed), "Musketeer Fine" (at only two finest levels 5% randomizations are allowed), and "Musketeer All" (small 1% randomizations are allowed at all levels). Because randomizations and editing are introduced at all levels, even very little changes at the coarse levels will result in significant changes at the finest level in generated synthetic graph.

We generated 30 network replicas for each network and compared the replicas with the original network based on the following metrics: number of nodes and edges, number of components, clustering coefficient, average degree, total degree-degree assortativity, average harmonic distance, modularity, pagerank and average betweenness centrality. We also compared our results with the existing generative models implemented in [48], namely, ReCoN, RMAT and BTER and stochastic Kronecker graphs by generating replicas of same input network. Since, these models do not necessarily generate planar network, *we post-processed the generated networks to find the maximal planar subgraph of the replicas* using OGDF library which uses edge removing technique, i.e., it adds one edge at a time while preserving planarity, if addition of the edge results in a non-planar graph then the edge is discarded thus generating a planar subgraph. We compared the generated planar graphs with the original graphs for the structural properties mentioned above. Clearly, one may argue that these generators were not developed to planar networks. We, however, note that these methods with planarization post-processing were chosen because there is no other acceptable solution to generate more or less realistically looking planar network that is similar to the original. As mentioned earlier, the available planar graph generators are generative models which either create specific classes of graphs with restricted values for minimum degree and connectivity (e.g., plantri and fullgen or Delaunay triangulation methods) or generate random realistic spatial networks based on give probability p (e.g., planar ErdosRenyi, and spatial Watts-Strogatz generator). Other examples include domain specific generators for road networks (e.g., StreetGen) and power grid random networks that are not necessarily planar networks. To the best of our knowledge, there is no domain independent generator whose goal is to preserve similarity with the input network.

The structural properties of the replicas were normalized such that 1 denotes the property of original network. We performed 30 experiments for each set of parameters the results for which is graphically represented in Figures 3.2-3.9. Our results indicate that multiscale planar graph generator can generate replicas that preserve almost all the properties of the original networks with relatively small deviation. Also, we observe that graphs generated by BTER and RMAT after planarization are close to original network (within $0-2$, where 1 represents the property of original input network after normalization) for properties such as average degree and mean harmonic distance whereas the properties for networks generated by stochastic Kronecker graphs (SKG) are far from those in the original graphs. As such the plots for properties for the networks generated by SKG are not represented in the plots. However, we note that the distortion of properties on the replicas by other network generators may have been the result of the post-processing step (maximal planar sub-graph of the generated replica), which often created more than one connected

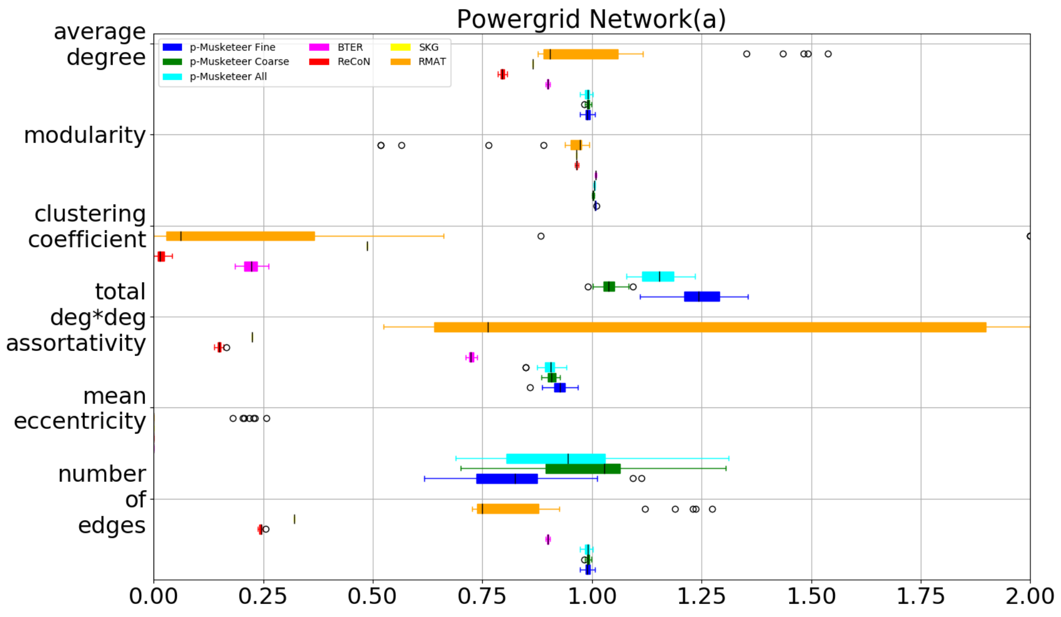


Figure 3.2: Computational results on performance of planar Musketeer on power grid graph opsahl-powergrid with 4941 nodes and 6211 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree.

components.

3.2.2 Rescaling

Our second set of experiments was designed to generate rescaled networks. We tested our implementation on three sets of parameters, namely, “Musketeer Coarse” (30% edge and node addition on 4 coarsest levels are allowed), “Musketeer Fine” (30% edge and node addition on 4 finest levels are allowed), and “Musketeer All” (15% edge and node addition at all levels are allowed). The parameters are chosen such that the generated network has $3 - 4$ times the number of nodes and edges than the original network. We generated 30 rescaled replicas for the same dataset as used in our previous experiment and compared the generated networks with the original network based on the following metrics: number of components, clustering coefficient, average degree, total degree-degree assortativity, average

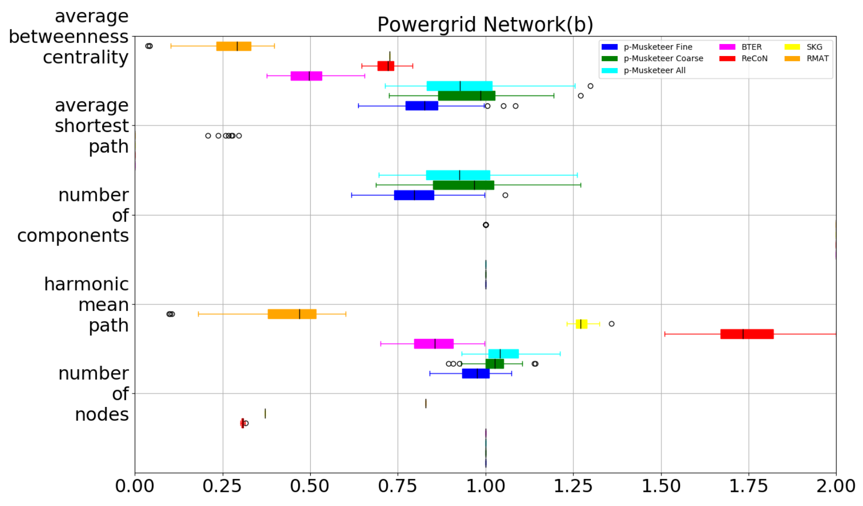


Figure 3.3: Computational results on performance of planar Musketeer on power grid graph opsahl-powergrid with 4941 nodes and 6211 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality.

harmonic distance, modularity, pagerank and average betweenness centrality.

The structural properties of the replicas were normalized such that 1 denotes the property of original network. The comparison for 30 experiments is presented in Figures 3.9-3.12. As depicted in the plots we are able to preserve almost all the properties of original network even when the network is rescaled to more than 3 times the original network. Also, there is no significant variance observed in properties for the three different sets of parameters (coarse,fine and all) used to generate rescaled networks. However, we observed that rescaling by introducing elements at finer levels results in high clustering coefficient in generated network. This is because the planarity constraint restricts addition of long edges (edges between nodes which are far from each other) which in turn forces the algorithm to connect new elements locally at each level i. In case the network elements are introduced at coarsest levels, the locally added edges and nodes are uncoarsened to several finer edges and nodes over the V-cycle of coarsening and uncoarsening, and the near neighbors at level i are drifted apart at level $i+1$.However, network elements added at fine levels are not drifted as a result of levels of coarsening and uncoarsening as described above, and the

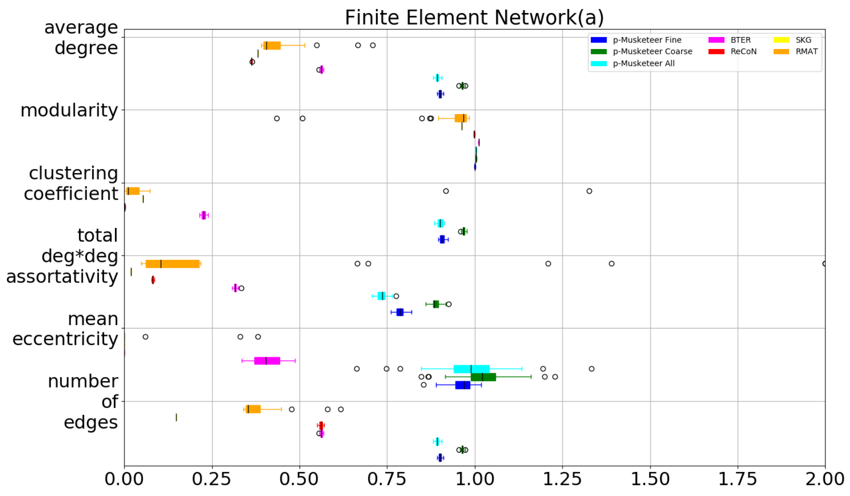


Figure 3.4: Computational results on performance of planar Musketeer on finite-element graph with 4704 nodes and 13427 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree.

edges still connect the nodes locally. Hence, we observe an increased number of triangles (Figure 3.10) or high clustering coefficient (Figures 3.11 - 3.14) for networks generated by introducing elements at fine level as compared to coarse level. As depicted in Figure 3.10 when the network is rescaled by introducing new elements at only coarse levels, we find larger communities (e.g., mesh structures in case of our input road network) in the generated network, whereas if the network is rescaled at fine level we observe smaller communities. The amount of new introduced elements can be controlled by user input which is provided as node growth parameters (from 0 to 1) at certain levels. In our experiment we used 0.3 as node growth rate for coarsest and finest level and 0.10 when introducing network elements at all levels.

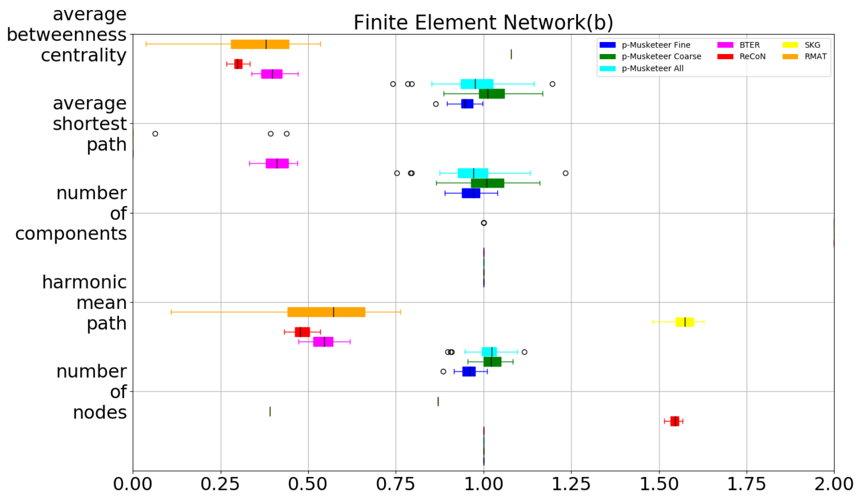


Figure 3.5: Computational results on performance of planar Musketeer on finite-element graph with 4704 nodes and 13427 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality.

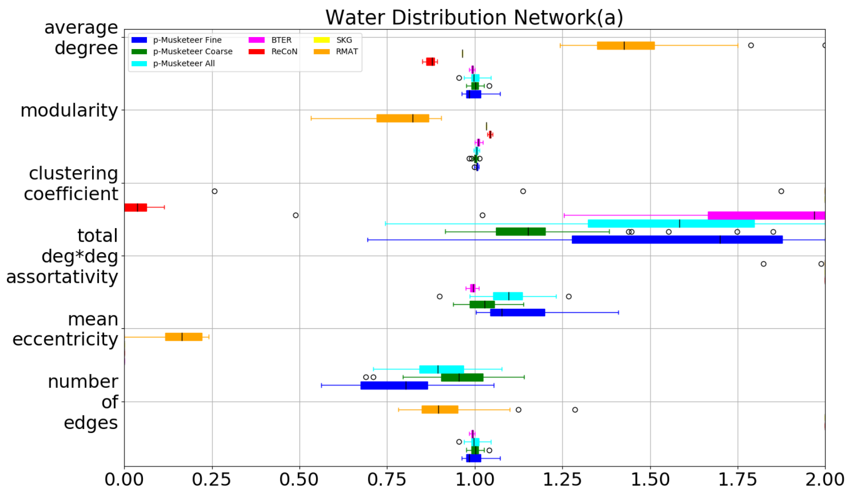


Figure 3.6: Computational results on performance of planar Musketeer on real water network with 407 nodes and 459 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree.

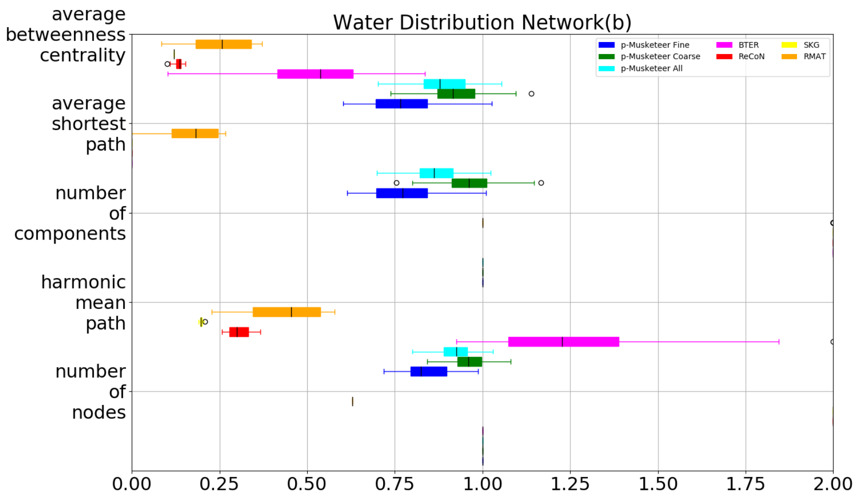


Figure 3.7: Computational results on performance of planar Musketeer on real water network with 407 nodes and 459 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality

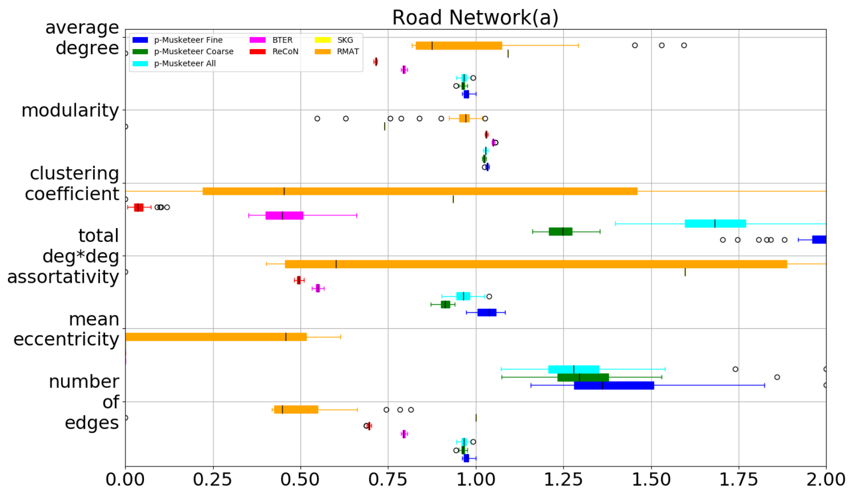


Figure 3.8: Computational results on performance of planar Musketeer on road network from roadNet-TX with 2001 nodes and 2957 edges for clustering coefficient, number of edges, mean eccentricity, total degree *degree assortativity, modularity and average degree.

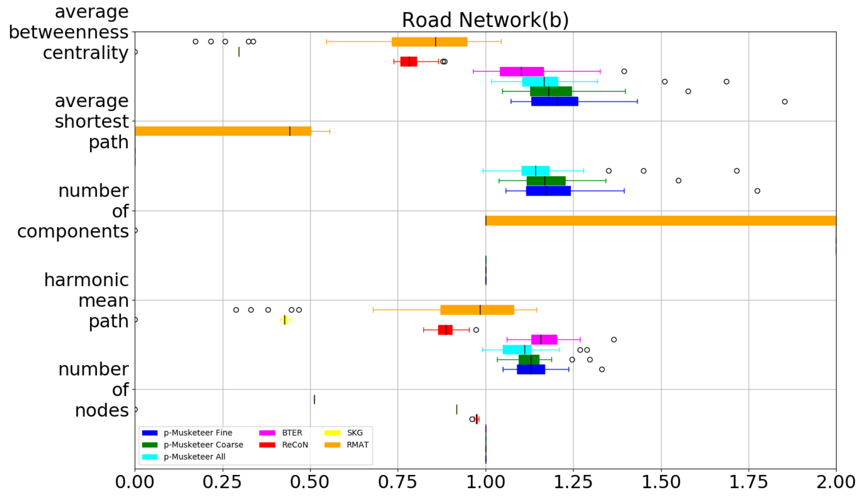


Figure 3.9: Computational results on performance of planar Musketeer on road network from roadNet-TX with 2001 nodes and 2957 edges for number of nodes, harmonic mean path, number of components, average shortest path and average betweenness centrality.

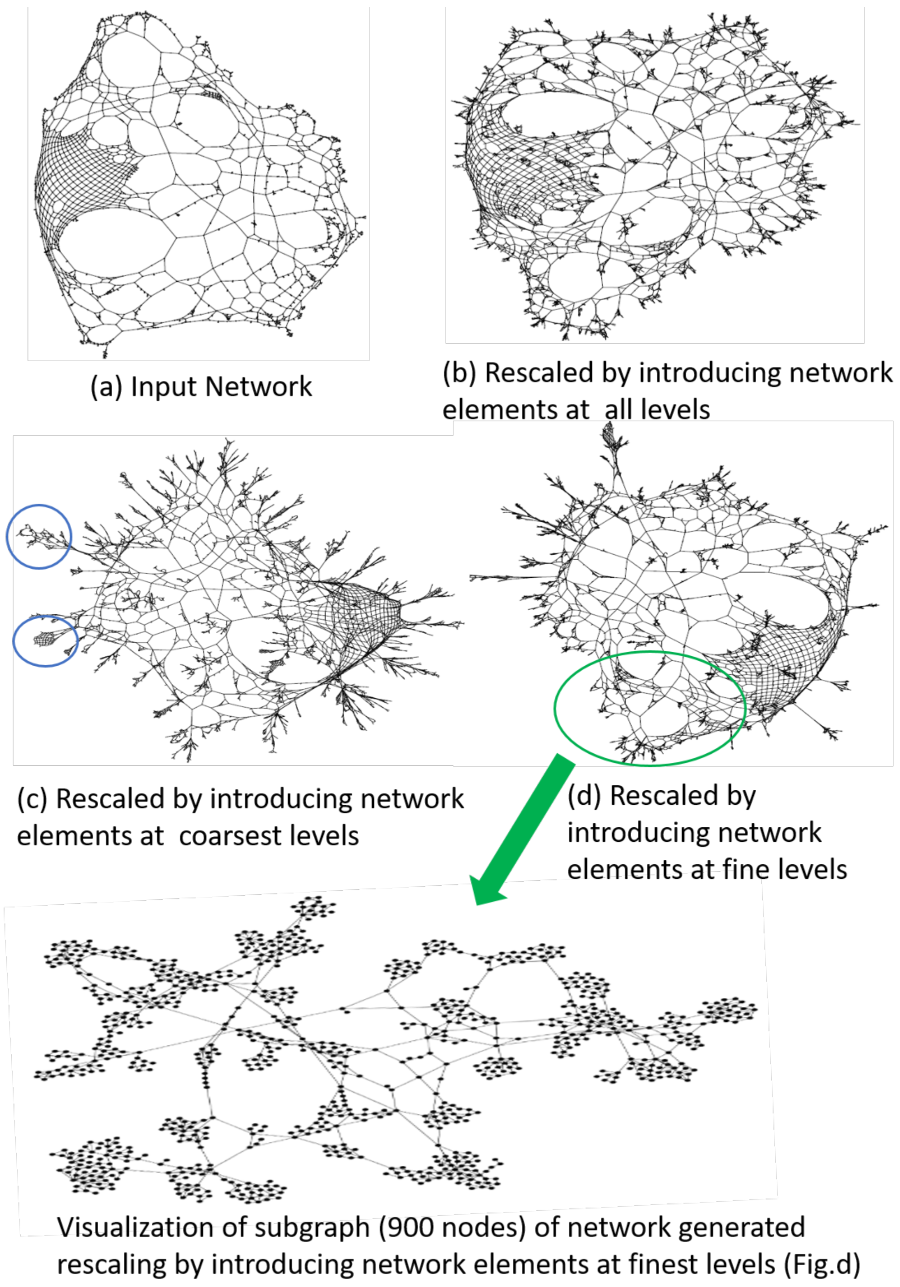

(a) Input Network
(b) Rescaled by introducing network elements at all levels
(c) Rescaled by introducing network elements at coarsest levels
(d) Rescaled by introducing network elements at fine levels
Visualization of subgraph (900 nodes) of network generated rescaling by introducing network elements at finest levels (Fig.d)

Figure 3.10: Visualization of a road network from roadNet-TX with 2001 nodes and 2957 edges rescaled to at least 5900 nodes and 7000 edges.

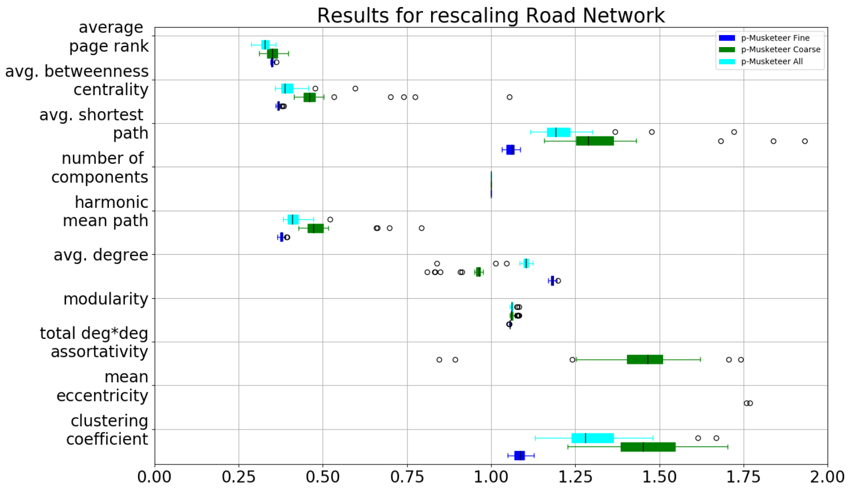


Figure 3.11: Computational results for road network from roadNet-TX with 2001 nodes and 2957 edges rescaled to at least 6000 nodes and 6500 edges.

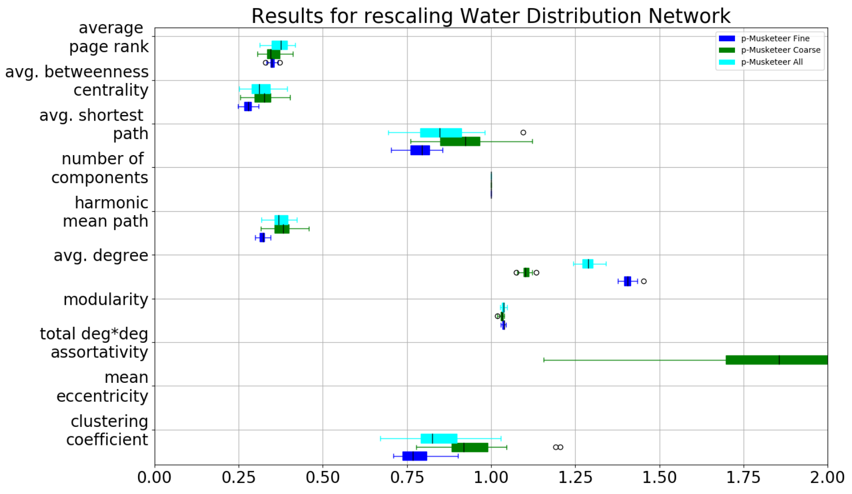


Figure 3.12: Computational results for road network real water network with 407 nodes and 459 edges rescaled to at least 1098 nodes and 1500 edges.

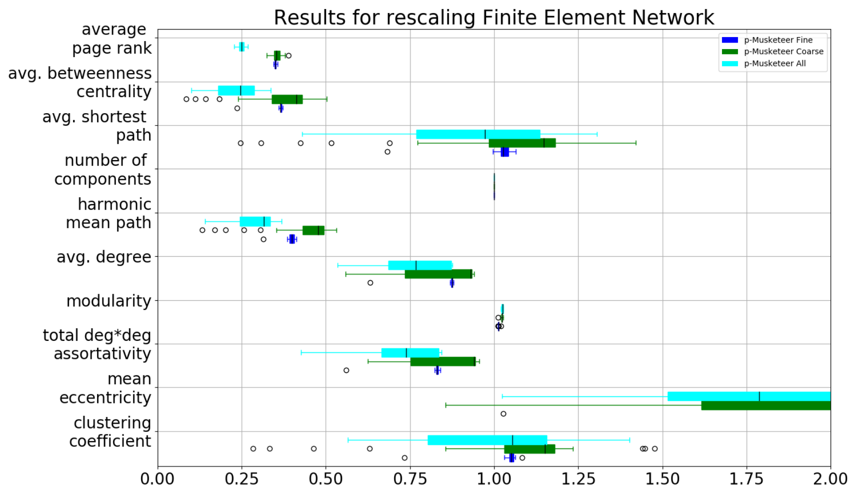


Figure 3.13: Computational results for finite-element graph with 4704 nodes and 13427 edges rescaled to at least 12700 nodes and 36000 edges.

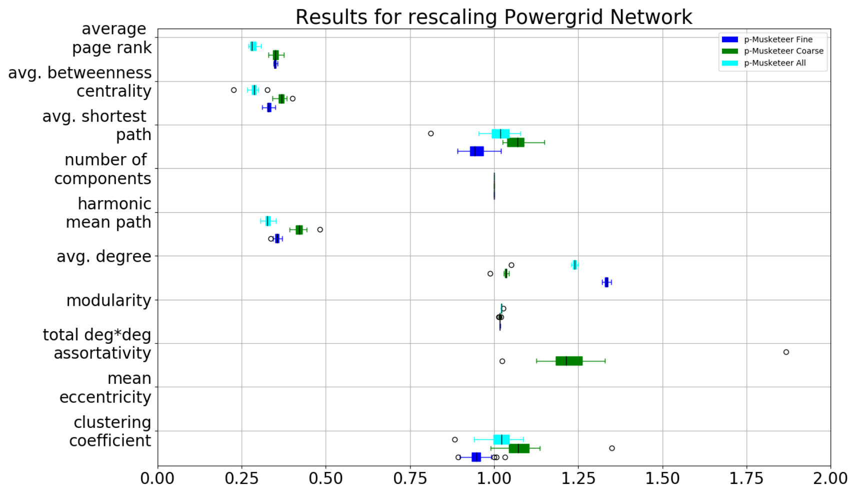


Figure 3.14: Computational results for power grid graph opsahl-powergrid with with 4941 nodes and 6211 edges rescaled to at least 16500 nodes and 27000 edges.

Chapter 4

Generating WDS from Planar Topology

A practical representation of an infrastructure network that can be used by the domain experts requires not only the network topology, but also its physical and operational characteristics. In this thesis we generate a synthetic Water Distribution System (WDS) with realistic topology as well as physical design features such as pipes, pumps, tanks and reservoirs. We try to replicate the input WDS, as such the physical features such as number of tanks and reservoir and parameters such as demand and elevation are based on the input WDS. However, features such as location of tanks and reservoir are based on specific characteristics often found real-world water networks. The generated WDS is in ".inp" format and can be interpreted by various available hydraulic solvers such as EPANet. In the following section we will discuss each feature and the parameter values for each in detail.

4.1 Assigning coordinates

This is the first step in the generation of WDS from a planar topology. Planar embedding of planar graphs is a well known problem with a variety of applications such as circuit design and infrastructure networks. There are various drawing methods for planar embedding [29] such as straight-line drawing and polyline drawing. There exist a number of algorithms such as [33], and [44]) for constructing planar embedding in $O(n^2)$ or faster time. In our model, we use popular visualization algorithms neato [30] and Forceatlas2 [19] to generate nearly planar layout of the generated planar graph. The number of iterations of graph visualization algorithms can be controlled by user. The returned positioning map in x, y coordinate space for nodes are written as "Coordinates" in the generated file. At the end of this step we have a network representation of an geographical area.

4.2 Reservoirs

A Reservoir represents an unlimited source that provides the network with water. In order to assume that the generated water network is a potential valid network, it must have at-least one reservoir/ tank. In case the generated network is rescaled such that it is larger than the original network then we increase the number of reservoirs by the rescaling factor while also introducing a decreasing probability factor β which adds a randomization factor and also restricts assignment of improbable or unrealistic number of reservoirs in the generated network.

Since in reality, a reservoir is more often a lake or river which is connected to the entire network, we make an assumption for a border location for the reservoir and find a node in our generated topology which lies on the boundary of the network and is connected to the remaining network with a single edge. In order to find the boundary nodes we use the position coordinates generated in previous step and create a set R, such that coordinates of nodes in R are boundary points in the previously generated layout. Our next step is based on the assumption that a reservoir is connected to network through single pipe/pump, as such we filter the selected nodes based on degree, namely, if the degree is greater than 1 the node is removed from R. Next, we randomly choose nodes from the set R and label the node as reservoir and repeat the process until we have allocated the required number of reservoirs. Finally, we assign some user specified value to various reservoir parameters such as “head”.

4.3 Tanks

Tanks play a significant role in the WDS design and operation. Using tanks storage capability, the WDSs design to be redundant, resourceful, and reliable in case of contingencies. In our model we assign tanks by first dividing the network into clusters such that they mimic the real-world WDS, this is achieved by minimum edge cut partitioning graph algorithm [6]. We use graph partitioning recursively such that the number of partitions is equal to number of tanks to be allocated. The partitions can be considered as clusters or communities found in real-world water networks, where each cluster has a tank.

Next, we introduce randomness using factor 'β' (a random value between $0-1$) which controls addition of unrealistic number of tanks. We select a random node in each partition which has degree 1 and random value between . If β is greater than some defined value p, we assign the selected node as tank, otherwise no tank is assigned in the partition. We then repeat the process for each partition assigning tanks in each partition with probability p.

Finally, we assign specified values to parameters such as the minimum level, maximum level, diameter and elevation tank. For generalization we assign elevation as maximum elevation of all junctions plus some specified value. The remaining parameters values are optimized later by our optimization algorithm discussed later. The addition of tanks will allow the network to be optimized in terms of operation to secure water delivery to customers during 24 hour operation period while saving energy and cost of pumps operation.

4.4 Junctions

Junctions will denote the network demand nodes which WDS need to satisfy. After the Reservoirs and Tanks are assigned, we label the remaining nodes in the graph as junctions. In our generation model, we assign values for junction parameters such as demand

and elevation that are based on the original input water network. This guarantees that the generated water network has similar demand nodes and geographical features as the input network.

We first, create sets E and D which contain elevation and demand values in the input network, respectively. Next, we assign elevation and demand values to junctions by randomly choosing values from set E and D. However, this causes an uneven distribution of elevation and demand, e.g., a low elevation (demand) node can be in a cluster which has all high elevation (demand) nodes and may result in an unrealistic network. To solve this problem, we apply iterative smoothing, such that the neighboring junctions have similar demand and elevation such that the generated network mimics characteristic of real-world network.

4.5 Pumps

Pumps are essential component to keep the WDS functional by providing an adequate amount of energy needed for WDS operation. An important property typically found in real world water networks is that the pumps are often located near the supply nodes such as reservoirs, and tanks. Our generator replicates this property by utilizing the same recursive partitioning as used for assigning tanks and assigning pumps by selecting edges in a partition which has a tank or reservoir, thus guaranteeing the location of pumps is near the tanks. It should be noted that the direction of flow for the pumps is based on the assumption that the system is filling water from the reservoirs to the tanks. Finally, we assign a user defined value to parameters such as pump curve and pattern, which can be optimized later by our optimization algorithm discussed in next chapter.

4.6 Pipes

Pipes are the veins of the WDSs that carry water through the network. After we assign pumps, the remaining edges are labeled as pipes. The assigned base values are based

Network	Junctions	Pipes	Pumps	Tanks	Reservoirs
Net1_Rossman	9	14	1	1	1
d-town	399	443	11	7	1
Charleston Network	1533	3401	5	4	1

Table 4.1: Input Networks

on the original input network but can also be provided as user input for pipe parameters length, diameter, roughness, minorloss and status. We optimize pipe diameter later using genetic optimization solver discussed in the next chapter.

After assignment of physical features is complete, we write the network in '.inp' format to generate a network file which can be exported to any hydraulic solver.

4.7 Experiments and Results

In this section we show the computational results summarizing the performance of our WDS generator in replicating the original and also generating rescaled networks. To test the variablity of our generated we used 3 different water networks, namely, Net1_Rossman [37], d-town network from "The Battle of the Water Networks II" [32], and Charleston area network details for which are provided in Table 4.7.

We tested our model on 2 different parameters, first replication by introducing 1% edits at all levels and second rescaling by introducing 15% growth at each level of uncoarsening. In case of replication, the generated network has similar number of WDS components as that is original network, whereas in rescaling the number of components was rescaled to meet the needs of larger sized network with random probability parameter $\beta = 1$ and minimum probability $p = 0.5$ (discussed in previous section). Since, Net1_Rossman is a small network (11 nodes and 15 edges), we generated a much larger network by recursively 4.7 generating rescaled networks from the original Network such that the final generated network is rescaled to at least 22 times the original network. We then ran the hydraulic simulation for these networks using EPANet software. The summary and visualization of

Iteration	Junctions	Pipes	Pumps	Tanks	Reservoirs
Original Network	9	14	1	1	1
1	56	76	1	1	1
2	126	169	2	2	1
3	231	308	1	1	1

Table 4.2: Summary of WDS Components in synthetic WDS generated from Net1_Rossman Network

Network	Junctions	Pipes	Pumps	Tanks	Reservoirs
Original Network	399	443	11	7	1
Replicated Network	401	464	2	5	1
Rescaled Network	814	1048	6	11	1

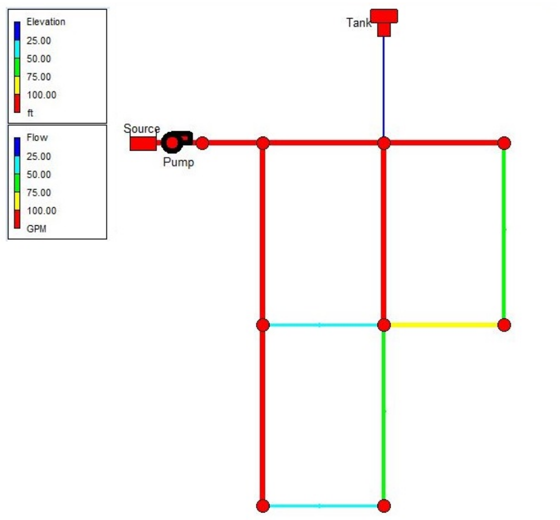
Table 4.3: Summary of WDS Components in synthetic WDS generated from d-town Network

generated WDSs is represented in Tables 4.7-4.7 and Figures 4.7-4.7

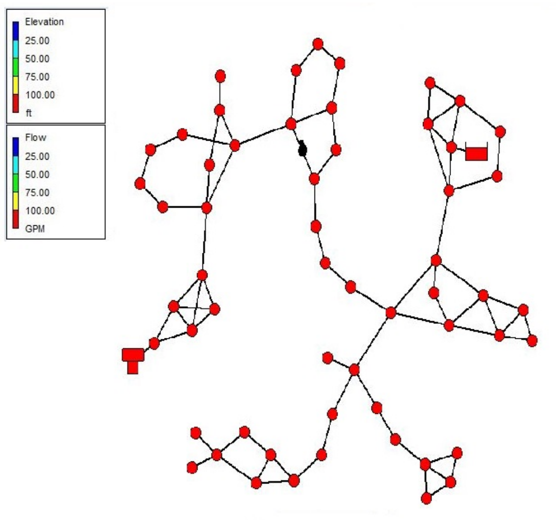
Our results indicate that our Water Distribution system can generate synthetic WDS that preserve the topological as well as physical design of the original WDS. The generator is also successful in replicating the small communities with similar degree, elevation and demand junctions as found in real-world network which is used as the input to our model.

Network	Junctions	Pipes	Pumps	Tanks	Reservoirs
Original Network	1533	3401	5	4	1
Replicated Network	1496	1664	3	3	1
Rescaled Network	3130	3833	3	6	1

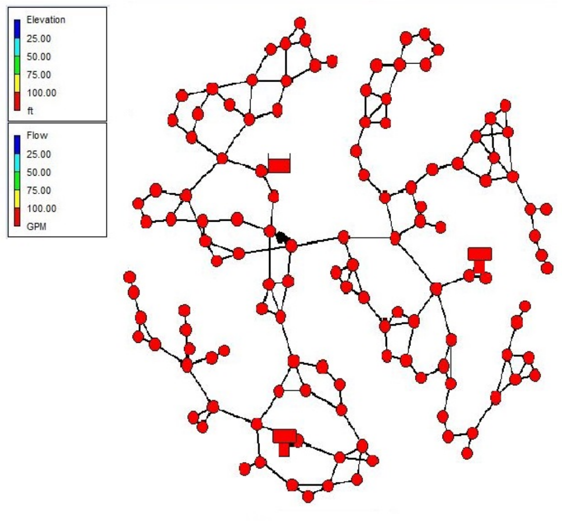
Table 4.4: Summary of WDS Components in synthetic WDS generated from Charleston Network



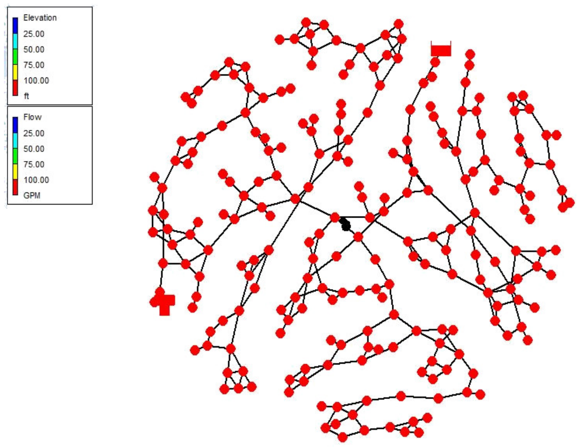
a



b

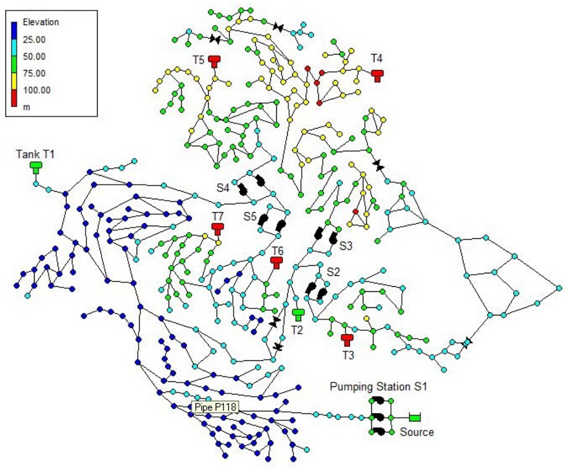


c

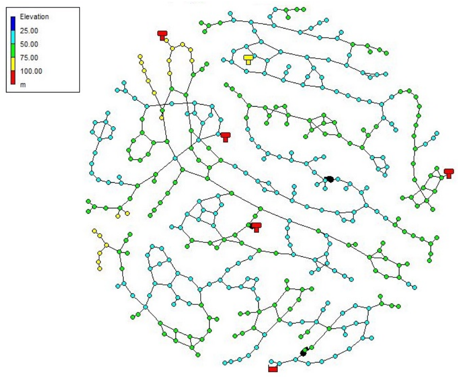


d

Figure 4.1: Visualization of input network Net1_Rossman and generated synthetic networks on EPANet software. (a) Original Input Network (b) Synthetic WDS rescaled to at least 7 times (c) Synthetic WDS rescaled to at least 7 times (b) Synthetic WDS rescaled to atleast 14 (d) Synthetic WDS rescaled to atleast 22.



a



b

c

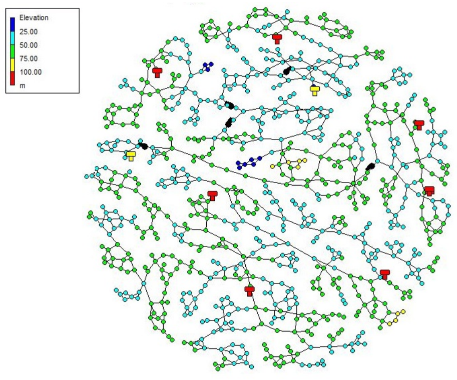


Figure 4.2: Visualization of input network d-town and generated synthetic networks on EPANet software. (a) Original Input Network (b) Synthetic WDS by replication (c) Synthetic WDS rescaled to at least 2 times.

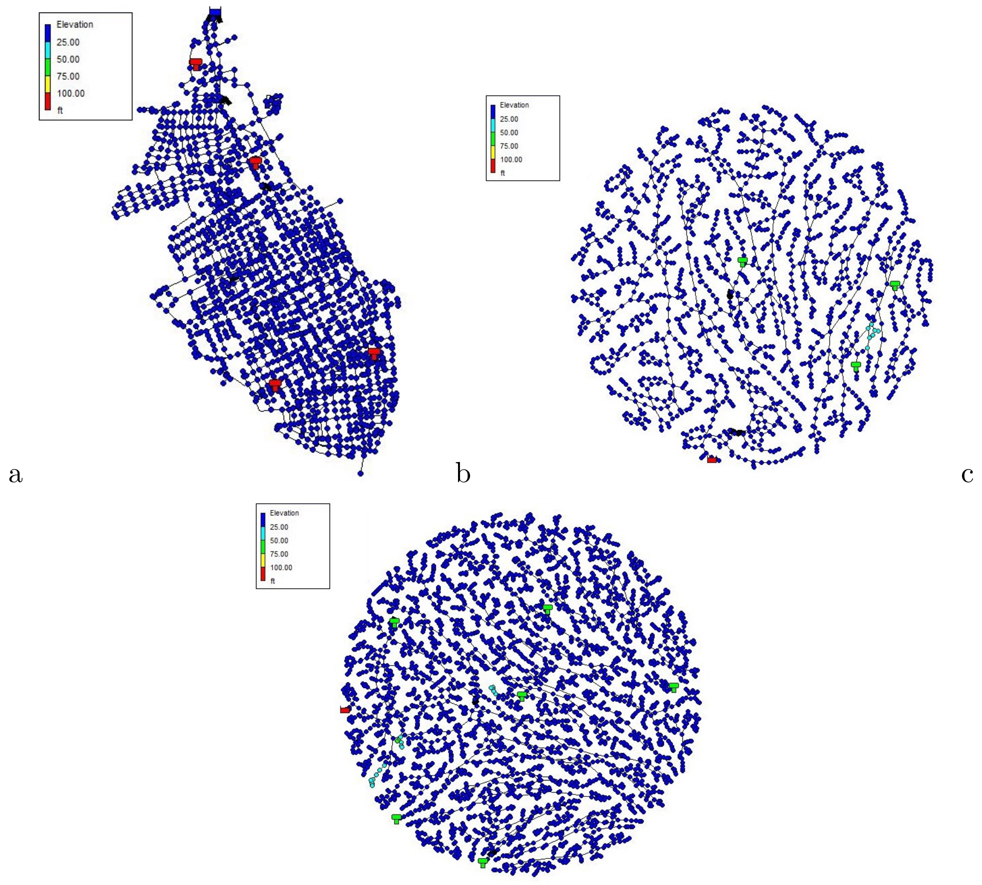


Figure 4.3: Visualization of input network Charleston Network and generated synthetic networks on EPANet software. (a) Original Input Network (b) Synthetic WDS by replication (c) Synthetic WDS rescaled to at least 2 times.

Chapter 5

Optimizing Water Distribution System Components

The WDS design is a discrete non-linear NP-hard optimization problem. The constraints are functions of decision variables such as pipe and tank diameters, pump sizes and operation status (ON and OFF) and is calculated by hydraulic simulation which requires solving conservation of mass and energy. Also, the objectives such as cost and reliability are multi modal convex function. As such we rely on evolutionary algorithms such as genetic algorithm, evolution strategy , differential evolution etc. to find an optimal solution to such a problem. In this thesis, we used NSGAII [10] genetic algorithm to solve WDS optimization problem.

5.1 Objectives

In this thesis, we optimize the generated water network for two objectives, maximum resilience and minimum cost.

Resilience: In this thesis, a weighted metric called resilience is utilized where the reliability of the WDS is measured and maximized. This metric is weighted for the case study since the hydraulic simulation is associated with extended period of 24 hours and

Diameter (in)	6	8	10	12	14	16	18	20	24	30
Cost (USD)	137.76	191.552	242.064	314.224	389.664	469.04	554.32	646.816	828.528	1135.208

Table 5.1: Diameter of Pipe vs Cost

Tank Volume (cubic ft)	50000	100000	250000	500000	1000000
Cost (USD)	115000	145000	325000	425000	600000

Table 5.2: Volume of Tank vs Cost

there is a demand pattern involved at an hourly intervals. The resilience is the ratio of total energy of the demand nodes and energy provided by the source nodes. The energy of a node is a product of head and demand. In our energy quantification we consider a tank as demand node when it is filling whereas when it is considered source when it is providing water to the system.

Cost: In our optimization model we include design as well as operational cost. The design cost is the summation of cost of pipes, pumps and tanks whereas the operational cost is calculated based on the energy used by the system over 365 days of operation period. The cost of pipe and tank is calculated from the diameter of pipe and volume of tank based on Table 5.1 and Table 5.1

5.2 Constraint

The main constraint for the WDSs design is that an adequate amount of pressure is delivered that would satisfy customers demand. The evaluation of each design solution is performed through EPANET solver for an extended period hydraulic simulation to determine the pressures at all the WDS nodes for all design conditions, and the accumulated sum of the nodal pressure shortfalls (NPS) is used as the pressure deficit constraint. In addition, in this work, the tank characteristics are treated as independent design variables. In particular, the design top and bottom water levels for all tanks are specified for each trial solution. All tanks should empty and fill over their operational ranges during the specified

Design Variable	Pipe Diameter	Pump Size	Pump Control	Tank Min. level	Tank Max. level	Tank Diameter
Value	0-10	1-10	0-1	9-10	25-40	25-100
Type	Integer	Integer	Integer	Real	Real	Real

Table 5.3: Decision Variables

average demand day, leaving the specified emergency volumes untouched. The actual maximum and minimum water levels are then identified for each tank during a 24 h simulation (over a 24 h operation time). There will in general be a mismatch between the top and bottom water levels specified and those resulting from the simulation, and also a mismatch between initial and final water levels. The accumulated sum of the mismatch in levels is used as the operating level difference (TLD) constraint.

5.3 Decision Variables

Depending on the final generated network components, design decision variables would be assigned accordingly. The range of each type of variable is given in Table 5.3. For each pipe, the integer values in the pipe sizes, 1 to 10 correspond to the 10 available discrete pipe diameters: 152.4, 203.2, 254, 304.8, 355.6, 406.4, 457.2, 508, 609.6, and 762 mm (6, 8, 10, 12, 14, 16, 18, 20, 24, and 30 inches). For each pump design variable, the integer values in the pipe sizes, 1 to 10 correspond to the 10 available discrete pump curves 5.3. In addition, given a 24 h operation cycle and 1 h time step, the control variables for pump status (0 OFF, 1 ON) give a further 24 design variables for each pump in the WDS. Since cylindrical storage tank where used to supply water to the network, For each tank, the design variables for the new tanks are overflow and minimum normal day elevations, bottom of tank from minimum normal day elevation, and diameter of each tank (4 design variables each). Minimum and maximum boundary values for the decision variables of overflow and minimum normal day elevation, diameter, and bottom of tank from minimum normal day elevation are also given in Table 5.3.

Variable	Flow	Head
1	1000	180
2	2000	200
3	3000	220
4	4000	240
5	5000	260
6	1000	200
7	2000	220
8	3000	240
9	4000	260
10	5000	280

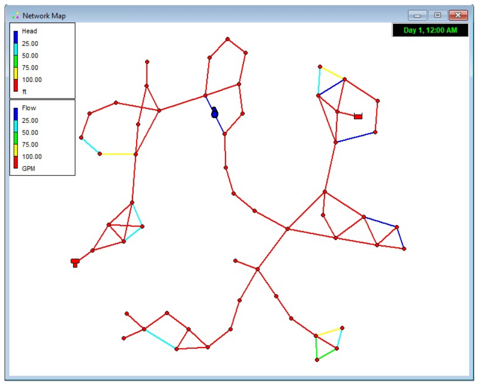
Table 5.4: Values for Pump Curve

Iteration	Number of Decision Variable	Resilience	Cost
1	104	0.353230515057	68873157.0349
2	225	0.731463304832	156375658.068
3	336	No solution	No solution

Table 5.5: Computational Results for Optimization Solver for synthetic network generated by rescaling Net1_Rossman Network

5.4 Experiments and Results

In this thesis, we tested our optimization model by recursively rescaling and optimizing the Net1_Rossman network 4.7 such that the size of generated network is at least 22 times original input network. We used Python implementation of NSGAII [10] genetic algorithm available in Platypus [16] optimization library. We ran our genetic algorithm for 1000 generation where each generation has the population size 20000. We further validated our solution, by importing the solution WDS in EPANet [38] software and running a 24 hours hydraulic simulation. The computational results for our optimization algorithm are represented in Table 5.4 and result of EPANet hydraulic simulation are represented in Figure 5.4

As shown in Table 5.3 our optimization algorithm is successful in finding a feasible solution for small networks. However, the algorithm fails to obtain a feasible solution as the number of decision variables is increased. One of the main reasons for this is the a 

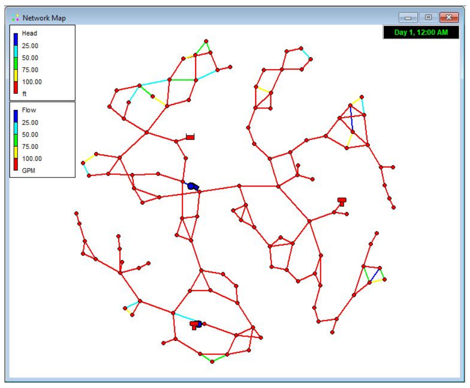
b 

Figure 5.1: Visualization of solution networks for Optimization algorithm on EPANet software with link flow and junction head values after 24 hr simulation. (a) Synthetic WDS with 76 pipes and 1 pump, tank and reservoir (b) Synthetic WDS with 169 pipes, 2 pumps and tanks and 1 reservoir

interdependence of decision variables such as pipe diameter and pump size which is not taken into consideration by a genetic algorithm in which selection of variable values at each run is based on a random generator.

Chapter 6

Conclusions

In this thesis, we introduced a multiscale planar graph generation framework and its implementation using Musketeer framework [15]. We then generated realistic synthetic Water Distribution Systems by adding physical design features to planar topology generated by our planar graph generator. Finally, we used genetic algorithm to optimize physical design and operational features of the generated WDS to obtain a cost efficient and reliable synthetic WDS. While there are clearly enough space for the improvement of this method, our 3 step algorithm provides a flexible and fast way to generate functional synthetic from a known real-world WDS with no manual intervention. Also, the mutiscale planar graph generator introduced in this thesis, to the best of our knowledge, is the first general purpose synthetic planar graph generation method that is able to produce realistic instances.

The Water Distribution System generation and optimization pipeline introduced in this thesis provides several future research directions that can be explored. First, the planar graph generation method proposed in this thesis offers a generic method to generate planar graphs across domains while preserving topological properties. However, the generator can be customized to replicate specific characteristics found in water networks such as maximum degree, size and number of communities, number of cycles etc. by controlling the size of aggregates during coarsening and introducing specific editing instead of random editing.

In this thesis, the assignment of WDS components such as pipes, pumps, tanks, reservoir etc. to nodes and edges of the generated graph is based on various assumptions and specific characteristics found in real-world water networks. In addition, we currently determine the values to the physical design parameters such as pipe diameter and tank diameter, tank sizes etc. randomly. This however often generates a non-feasible WDS. As a future research direction we propose an algorithm that iteratively assigns components and the physical design parameters based on the network requirements.

As discussed in Section 5.4 as the size of decision variables increases the genetic algorithm fails to find a feasible solution further work needs to be done to improve the optimization solver. As a possible next step we would like to investigate the inter-dependency of various decision variables e.g. pipe diameters vs pump sizes to reduce the number of decision variables.

Bibliography

[1] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm, 2000.

[2] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

[3] Marc Barthélemy. Spatial networks. *Physics Reports*, 499(1-3):1–101, 2011.

[4] G Brinkmann. Program fullgen-a program for generating nonisomorphic fullerenes. *see http://cs. anu. edu. au/bdm/plantri*, 2011.

[5] Gunnar Brinkmann, Brendan D McKay, et al. Fast generation of planar graphs. *MATCH Commun. Math. Comput. Chem*, 58(2):323–357, 2007.

[6] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[7] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446. SIAM, 2004.

[8] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (ogdf). *Handbook of Graph Drawing and Visualization*, 2011:543–569, 2013.

[9] T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 97(23), 1997.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[11] Alain Denise, Marcio Vasconcellos, and Dominic JA Welsh. The random planar graph. *Congressus numerantium*, pages 61–80, 1996.

[12] P ERDdS and A R&WI. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[13] Stefanie Gerke and Colin McDiarmid. On the number of edges in random planar graphs. *Combinatorics, Probability and Computing*, 13(2):165–183, 2004.

[14] Edward N Gilbert. Random plane networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961.

[15] Alexander Gutfraind, Ilya Safro, and Lauren Ancel Meyers. Multiscale network generation. In *18th IEEE International Conference on Information Fusion (Fusion)*, pages 158–165. IEEE, 2015.

[16] David Hadka. Moea framework user guide. 2014.

[17] William W Hager, James T Hungerford, and Ilya Safro. A multilevel bilinear programming algorithm for the vertex separator problem. *Computational Optimization and Applications*, 69(1):189–223, 2018.

[18] David R Hunter, Mark S Handcock, Carter T Butts, Steven M Goodreau, and Martina Morris. ergm: A package to fit, simulate and diagnose exponential-family models for networks. *Journal of statistical software*, 24(3):nihpa54860, 2008.

[19] Mathieu Jacomy, Sebastien Heymann, Tommaso Venturini, and Mathieu Bastian. Forceatlas2, a graph layout algorithm for handy network visualization. *Paris http://www. medialab. sciences-po. fr/fr/publications-fr*, 44, 2011.

[20] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[21] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[22] Sven Leyffer and Ilya Safro. Fast response to infection spread and cyber attacks on large-scale networks. *Journal of Complex Networks*, 1(2):183–199, 2013.

[23] Mohammad Mahdian and Ying Xu. Stochastic kronecker graphs. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 179–186. Springer, 2007.

[24] Colin McDiarmid, Angelika Steger, and Dominic JA Welsh. Random planar graphs. *Journal of Combinatorial Theory, Series B*, 93(2):187–205, 2005.

[25] Sascha Meinert and Dorothea Wagner. An experimental study on generating planar graphs. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 375–387. Springer, 2011.

[26] João Muranho, Ana Ferreira, Joaquim Sousa, Abel Gomes, and Alfeu Sá Marques. Waternetgen: an epanet extension for automatic water distribution network models generation and pipe sizing. *Water science and technology: water supply*, 12(1):117–123, 2012.

[27] Mark Newman. *Networks: An Introduction.* Oxford University Press, Inc., New York, NY, USA, 2010.

[28] Mark Newman. *Networks*. Oxford university press, 2018.

[29] Takao Nishizeki and Md Saidur Rahman. *Planar graph drawing*, volume 12. World Scientific Publishing Company, 2004.

[30] Stephen C North. Drawing graphs with neato. *NEATO User manual*, 11:1, 2004.

[31] Jennifer B Nuzzo. The biological threat to us water supplies: toward a national water security policy. *Biosecurity and bioterrorism: biodefense strategy, practice, and science*, 4(2):147–159, 2006.

[32] Avi Ostfeld, James G Uber, Elad Salomons, Jonathan W Berry, William E Hart, Cindy A Phillips, Jean-Paul Watson, Gianluca Dorini, Philip Jonkergouw, Zoran Kapelan, et al. The battle of the water sensor networks (bwsn): A design challenge for engineers and algorithms. *Journal of Water Resources Planning and Management*, 134(6):556–568, 2008.

[33] János Pach and Rephael Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.

[34] G. Palla, L. Lovász, and T. Vicsek. Multifractal network generator. *Proceedings of the National Academy of Sciences*, 107(17):7640, 2010.

[35] A Ramachandra Rao, Rabindranath Jana, and Suraj Bandyopadhyay. A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 225–242, 1996.

[36] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

[37] Lewis A Rossman, Robert M Clark, and Walter M Grayman. Modeling chlorine residuals in drinking-water distribution systems. *Journal of environmental engineering*, 120(4):803–820, 1994.

[38] Lewis A Rossman et al. Epanet 2: users manual. 2000.

[39] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of algorithms*, 18(3):548–585, 1995.

[40] Ilya Safro, Dorit Ron, and Achi Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[41] Ilya Safro, Dorit Ron, and Achi Brandt. A multilevel algorithm for the minimum 2-sum problem. *J. Graph Algorithms Appl.*, 10(2):237–258, 2006.

[42] Ilya Safro, Dorit Ron, and Achi Brandt. Multilevel algorithms for linear ordering problems. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[43] Ilya Safro and Boris Temkin. Multiscale approach for the network compression-friendly ordering. *J. Discrete Algorithms*, 9(2):190–202, 2011.

[44] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148. Society for Industrial and Applied Mathematics, 1990.

[45] Comandur Seshadhri, Tamara G Kolda, and Ali Pinar. Community structure and scale-free collections of erdős-rényi graphs. *Physical Review E*, 85(5):056109, 2012.

[46] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied computational geometry towards geometric engineering*, pages 203–222. Springer, 1996.

[47] Robert Sitzenfrei, Michael Möderl, and Wolfgang Rauch. Automatic generation of water distribution systems based on gis data. *Environmental modelling & software*, 47:138–147, 2013.

[48] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: An interactive tool suite for high-performance network analysis. *CoRR, abs/1403.3005*, 2014.

[49] Christian L. Staudt, Michael Hamann, Alexander Gutfraind, Ilya Safro, and Henning Meyerhenke. Generating realistic scaled complex networks. *Applied Network Science*, 2(1):36, Oct 2017.

[50] Christian L Staudt, Michael Hamann, Ilya Safro, Alexander Gutfraind, and Henning Meyerhenke. Generating scaled replicas of real-world complex networks. In *International Workshop on Complex Networks and their Applications*, pages 17–28. Springer, 2016.

[51] Lionel Tabourier, Camille Roth, and Jean-Philippe Cointet. Generating constrained random graphs using multiple edge switches. *J. Exp. Algorithmics*, 16:1.7:1.1–1.7:1.15, December 2011.

[52] William Thomas Tutte. A census of planar maps. *Canad. J. Math*, 15(2):249–271, 1963.