**Leiden University**

**Master Computer Science**

# An Improved Maximum-Likelihood Solver for the Analysis of Graph Ensembles

|  |  |
|---|---|
| Name: | Eli van Es |
| Date: | 16/12/2014 |
| 1st supervisor: | Dr. Walter Kosters |
| 2nd supervisor: | Dr. Diego Garlaschelli |

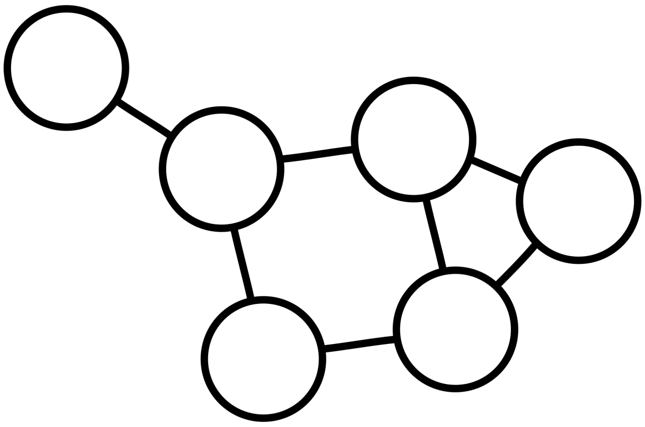
MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

# An Improved Maximum-Likelihood Solver for the Analysis of Graph Ensembles

**Eli van Es**

eli@vortech.nl



In collaboration with:

*VORtech B.V.*
*Leiden Institute of Physics*
*Leiden Institute of Advanced Computer Science*

## Abstract

Graph analysis is gaining popularity and can help in understanding real world systems and data. A common approach is the study of graph ensembles, which has led to a fast and accurate "maximum-likelihood" method for this analysis.

An important part of this new method has been implemented in software but faces several critical issues. Most importantly, the implementation's ability to find good solutions is unpredictable and its run times are problematic.

We review this implementation and propose our improved implementation. We take a critical look at the original problem definitions and apply mathematical-analysis, algorithmic and software-engineering techniques to circumvent the issues of the current implementation. Our work results in an implementation that, among other improvements, always finds good solutions and decreases run times dramatically.

**Acknowledgements**

I would like to thank my supervisors Jok of VORtech and Diego and Walter of Leiden University for their assistance, tips and constructive feedback. Furthermore, I would also like to thank my colleagues at VORtech, in particular Jok, Hisham, Bas and Reijer, for their involvement in the project and sharing their vast knowledge on numerical mathematics.

Without the help of the aforementioned people, the project would not have been so successful and this thesis would not be in the state that it is today.

# Contents

# Chapter 1

# Introduction

Graphs are becoming an increasingly popular tool for modeling systems and data in a wide range of application domains. Higher order properties of graphs are studied to give insight into structural features that can help in understanding these systems and data. See [1] for a broad overview of graph theory and its current applications.

In statistical physics, when analyzing higher order properties of a graph, a common approach is to study not just the graph itself but an ensemble of graphs. This ensemble of graphs contains all graphs that share a certain topological property with the original graph. For example, let the original graph be the graph in Figure 1.1(a) and let the topological property be the number of edges of each node, the ensemble would then be all graphs in Figure 1.1. In other words, the ensemble of Figure 1.1 is constrained by the number of edges of each node.

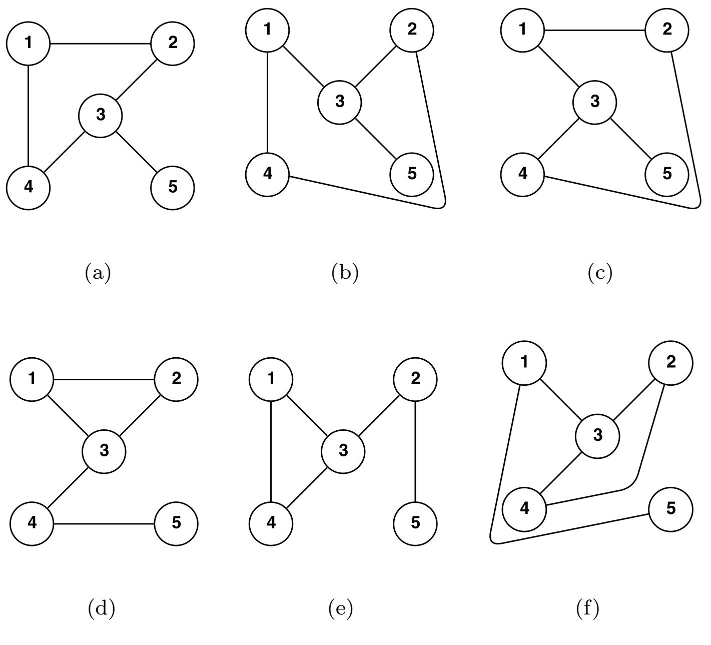


Figure 1.1: Ensemble of graphs, where each node has the same number of edges in each graph of the ensemble. Note that the graphs in each row are isomorphic as unlabeled graphs.

By looking at the higher order properties of the graphs in the ensemble and comparing them to the higher order properties of the original graph, one can say whether the higher order property is a result of the simple constraints that define the ensemble or if it is the result of more complex structural features. This analysis is for example used in [2] to detect structural collapse in the interbank network that eventually led to the financial crisis of 2008.

Existing methods for this analysis exist but are either computationally expensive or highly approximate. In [3] a so called “maximum-likelihood” method is proposed for this analysis that is both fast and accurate. The method is able to compute the probability that an edge exists between any two nodes in the graphs of the ensemble. These probabilities can then be used to either generate graphs from the ensemble or compute the average value and standard deviation of a property of the ensemble. The probabilities are defined as functions of the hidden variables of the graph ensemble and the method describes how to determine these hidden variables. The hidden variables themselves do not represent any physical quantities.

To apply the method in practice, a software tool has been developed that can determine these hidden variables. Let us call this the current implementation/method/software. However, this software implementation has several critical issues. Mainly the software’s run time and scalability are problematic. Furthermore, the software often does not find a good solution. We propose a new approach and software implementation that tackles most of these issues. Let us call this the improved implementation/method/software.

To determine the hidden variables, a mathematical problem has to be solved, which is specified in Chapter 2. We define an improved and extended mathematical model for solving this problem in Chapter 3. The algorithms used for finding a solution and our extensions to them are described in Chapter 4. Details on our implementation are given in Chapter 5, ranging from software engineering aspects to high performance computing. Experiments that we carried out and their results are discussed in Chapter 6. In Chapter 7 we give a summary of what has been achieved and possible future work.

This document is a master’s thesis by Leiden University, supervised by Walter Kosters of the Leiden Institute of Advanced Computer Science and Diego Garlaschelli of the Leiden Institute of Physics. The associated master’s project has been executed as a collaboration between Leiden University and VORtech B.V.

# Chapter 2

# Problem Specification

The maximum-likelihood method, specified in [3], considers an ensemble of graphs $\mathcal{E}$, constrained by properties of our input graph. Each graph has nodes labeled with numbers from $\{1, 2, \ldots, n\}$ and $\mathbf{c} \in \mathbb{N}^N$ consists of the properties, where $N$ equals $n$, $2n$ or $3n$ depending on whether we consider 1, 2 or 3 properties per node.

The method specifies the function $p_{ij}(\mathbf{h}^*)$, where $p_{ij} \colon \mathbb{R}^N \to \mathbb{R}$ gives the probability that in the ensemble $\mathcal{E}$ there is an edge between nodes $i$ and $j$ and where the hidden variable sequence $\mathbf{h}^*$ is the solution of the optimization problem:

$$\begin{aligned} \max_{\mathbf{h}} \quad & \mathcal{L}(\mathbf{h}) \\ \text{subject to} \quad & h_i^{\min} \leq h_i < h_i^{\max} \quad \forall i, \end{aligned} \tag{2.1}$$

where $\mathcal{L} \colon \mathbb{R}^N \to \mathbb{R}$ is the objective function.

Alternatively, the method specifies that the same $\mathbf{h}^*$ can be found by solving the system of equations:

$$\langle \mathbf{c} \rangle(\mathbf{h}) = \mathbf{c}, \tag{2.2}$$

where $\langle \mathbf{c} \rangle \colon \mathbb{R}^N \to \mathbb{R}^N$ gives the ensemble average/expected values of the properties in $\mathbf{c}$ that should constrain the ensemble.

The notation for the input graph is defined in Section 2.1. The likelihood/entropy function $\mathcal{L}$ from equation (2.1) and the expected ensemble constraints function $\langle \mathbf{c} \rangle$ from equation (2.2) depend on the properties that should constrain the graph ensemble. An instance of this is called a configuration model. The authors of [3], [4], [5] and [6] define several configuration models that are further explained in Section 2.2. The current software implementation defines a measure for determining the optimality of a hidden variable vector. We apply reverse engineering and define this measure in Section 2.3.

## 2.1 Input Graph

The various configuration models support undirected/directed and binary/weighted graphs. We represent these graphs using an adjacency matrix.

For binary graphs we have the adjacency matrix $A$, with $A_{ij} = 0$ if there is no edge between nodes $i$ and $j$, $A_{ij} = 1$ if there is an edge between nodes $i$ and $j$ and with zeros on the diagonal of $A$. This matrix is symmetric for undirected graphs and asymmetric for directed graphs. This makes sense because in undirected graphs the edge between nodes $i$ and $j$ is the same as the edge between nodes $j$ and $i$, whereas this is not the case for directed graphs.

For weighted graphs we have the weighted adjacency matrix $W$, with $W_{ij} \in \mathbb{N}$ equal to the weight of the edge between nodes $i$ and $j$ or $W_{ij} = 0$ if there is no edge between nodes $i$ and $j$ and with zeros on the diagonal of $W$. Again this matrix is symmetric for undirected graphs and asymmetric for directed graphs.

## 2.2 Configuration Model

In the general case we defined the ensemble constraints as the vector $\mathbf{c}$ and the hidden variables as the vector $\mathbf{h}$. In the case of a specific configuration model we use a different notation because $\mathbf{c}$ and $\mathbf{h}$ can consist of multiple vectors.

To give an example of this, lets say a configuration model constrains the ensemble on some property $\mathbf{a}$, where $a_i$ is the value of this property for node $i$. We then define the hidden variable vector $\mathbf{x}$, where $x_i$ is the hidden variable value for node $i$. So in this example we get $\mathbf{c} = \mathbf{a}$ and $\mathbf{h} = \mathbf{x}$.

For another example, lets say a configuration model constrains the ensemble on the properties $\mathbf{a}$ and $\mathbf{b}$, where $b_i$ is the value of a different property for node $i$. We then define the hidden variable vectors $\mathbf{x}$ and $\mathbf{y}$, where $y_i$ is another hidden variable value for node $i$. So in this example we get $\mathbf{c} = \mathbf{a}^\frown\mathbf{b}$ and $\mathbf{h} = \mathbf{x}^\frown\mathbf{y}$, where we define $\frown$ as:

$$x^\frown y = \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.3}$$

To analyze different properties of different graph types, the original papers specify several configuration models. The specific configuration models are described in Sections 2.2.1 to 2.2.7.

### 2.2.1 Undirected Binary Configuration Model

The undirected binary configuration model (UBCM) constrains the ensemble on the degree sequence $\mathbf{k}$ defined as:

$$k_i = \sum_{j \neq i} A_{ij} \quad \forall i. \tag{2.4}$$

Described in words, each node in a graph has a degree, which is equal to the number of edges of that node.

The likelihood function is defined in equation (2.5) and the expected degrees are defined in equation (2.6), where $\mathbf{x}$ consists of the hidden variables.

$$\mathcal{L}(\mathbf{x}) = \sum_i \Big( k_i \log(x_i) - \sum_{j>i} \log(1 + x_i x_j) \Big) \tag{2.5}$$

$$\langle k_i \rangle(\mathbf{x}) = \sum_{j \neq i} \frac{x_i x_j}{1 + x_i x_j} \quad \forall i \tag{2.6}$$

The search space of $\mathbf{x}$ is constrained by the bounds:

$$x_i \geq 0 \quad \forall i. \tag{2.7}$$

### 2.2.2 Undirected Weighted Configuration Model

The undirected weighted configuration model (UWCM) constrains the ensemble on the strength sequence $\mathbf{s}$ defined as:

$$s_i = \sum_{j \neq i} W_{ij} \quad \forall i. \tag{2.8}$$

Described in words, each node in a graph has a strength, which is the weighted variant of the degree of that node.

The likelihood function is defined in equation (2.9) and the expected strengths are defined in equation (2.10), where $\mathbf{x}$ consists of the hidden variables.

$$\mathcal{L}(\mathbf{x}) = \sum_i \Big( s_i \log(x_i) + \sum_{j>i} \log(1 - x_i x_j) \Big) \tag{2.9}$$

$$\langle s_i \rangle(\mathbf{x}) = \sum_{j \neq i} \frac{x_i x_j}{1 - x_i x_j} \quad \forall i \tag{2.10}$$

The search space of $\mathbf{x}$ is constrained by the bounds:

$$0 \leq x_i < 1 \quad \forall i. \tag{2.11}$$

### 2.2.3 Undirected Mixed Configuration Model

The undirected mixed configuration model (UMCM) constrains the ensemble on both the degree sequence $\mathbf{k}$ and the strength sequence $\mathbf{s}$ as defined in equations (2.4) and (2.8) respectively.

The likelihood function is defined in equation (2.12) and the expected degrees and strengths are defined in equation (2.13), where $\mathbf{x}$ and $\mathbf{y}$ consist of the hidden variables.

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \sum_i \Big( k_i \log(x_i) + s_i \log(y_i) + \sum_{j>i} \big( \log(1 - y_i y_j) - \log(1 - y_i y_j + x_i x_j y_i y_j) \big) \Big) \tag{2.12}$$

$$\begin{aligned} \langle k_i \rangle(\mathbf{x}, \mathbf{y}) &= \sum_{j \neq i} \frac{x_i x_j y_i y_j}{1 - y_i y_j + x_i x_j y_i y_j} & \forall i \\ \langle s_i \rangle(\mathbf{x}, \mathbf{y}) &= \sum_{j \neq i} \Big( \frac{x_i x_j y_i y_j}{1 - y_i y_j + x_i x_j y_i y_j} / (1 - y_i y_j) \Big) & \forall i \end{aligned} \tag{2.13}$$

The search space of $\mathbf{x}$ and $\mathbf{y}$ is constrained by the bounds:

$$\begin{aligned} x_i &\geq 0 && \forall i \text{ and} \\ 0 \leq y_i &< 1 && \forall i. \end{aligned} \tag{2.14}$$

### 2.2.4 Directed Binary Configuration Model

The directed binary configuration model (DBCM) constrains the ensemble on the out-degree sequence $\mathbf{k}^{out}$ and the in-degree sequence $\mathbf{k}^{in}$ defined as:

$$\begin{aligned} k_i^{out} &= \sum_{j \neq i} A_{ij} \quad \forall i \text{ and} \\ k_i^{in} &= \sum_{j \neq i} A_{ji} \quad \forall i. \end{aligned} \tag{2.15}$$

Described in words, each node in a graph has an out- and in-degree. The out-degree is equal to the number of outgoing edges of that node, while the in-degree is equal to the number of incoming edges of that node.

The likelihood function is defined in equation (2.16) and the expected out- and in-degrees are defined in equation (2.17), where $\mathbf{x}$ and $\mathbf{y}$ consist of the hidden variables.

$$\mathcal{L}(\mathbf{x},\mathbf{y}) = \sum_i \Big( k_i^{out} \log(x_i) + k_i^{in} \log(y_i) - \sum_{j\neq i} \log(1 + x_i y_j) \Big) \tag{2.16}$$

$$\begin{aligned} \langle k_i^{out} \rangle(\mathbf{x},\mathbf{y}) &= \sum_{j\neq i} \frac{x_i y_j}{1 + x_i y_j} \quad \forall i \\ \langle k_i^{in} \rangle(\mathbf{x},\mathbf{y}) &= \sum_{j\neq i} \frac{x_j y_i}{1 + x_j y_i} \quad \forall i \end{aligned} \tag{2.17}$$

The search space of $\mathbf{x}$ and $\mathbf{y}$ is constrained by the bounds:

$$\begin{aligned} x_i &\geq 0 \quad \forall i \text{ and} \\ y_i &\geq 0 \quad \forall i. \end{aligned} \tag{2.18}$$

### 2.2.5 Directed Weighted Configuration Model

The directed weighted configuration model (DWCM) constrains the ensemble on the out-strength sequence $\mathbf{s}^{out}$ and the in-strength sequence $\mathbf{s}^{in}$ defined as:

$$\begin{aligned} s_i^{out} &= \sum_{j\neq i} W_{ij} \quad \forall i \text{ and} \\ s_i^{in} &= \sum_{j\neq i} W_{ji} \quad \forall i. \end{aligned} \tag{2.19}$$

Described in words, each node in a graph has an out- and in-strength, which are the weighted variants of the out- and in-degree of that node.

The likelihood function is defined in equation (2.20) and the expected out- and in-strengths are defined in equation (2.21), where $\mathbf{x}$ and $\mathbf{y}$ consist of the hidden variables.

$$\mathcal{L}(\mathbf{x},\mathbf{y}) = \sum_i \Big( s_i^{out} \log(x_i) + s_i^{in} \log(y_i) + \sum_{j\neq i} \log(1 - x_i y_j) \Big) \tag{2.20}$$

$$\begin{aligned} \langle s_i^{out} \rangle(\mathbf{x},\mathbf{y}) &= \sum_{j\neq i} \frac{x_i y_j}{1 - x_i y_j} \quad \forall i \\ \langle s_i^{in} \rangle(\mathbf{x},\mathbf{y}) & \sum_{j\neq i} \frac{x_j y_i}{1 - x_j y_i} \quad \forall i \end{aligned} \tag{2.21}$$

The search space of $\mathbf{x}$ and $\mathbf{y}$ is constrained by the bounds:

$$\begin{aligned} 0 \leq x_i < 1 \quad \forall i \text{ and} \\ 0 \leq y_i < 1 \quad \forall i. \end{aligned} \tag{2.22}$$

### 2.2.6 Reciprocal Binary Configuration Model

The reciprocal binary configuration model (RBCM) constrains the ensemble on the out-only-degree sequence $\mathbf{k}^{\rightarrow}$, the in-only-degree sequence $\mathbf{k}^{\leftarrow}$ and the reciprocal-degree sequence $\mathbf{k}^{\leftrightarrow}$ defined as:

$$\begin{aligned} k_i^{\rightarrow} &= \sum_{j\neq i} \big(A_{ij}(1-A_{ji})\big) \quad \forall i, \\ k_i^{\leftarrow} &= \sum_{j\neq i} \big(A_{ji}(1-A_{ij})\big) \quad \forall i \text{ and} \\ k_i^{\leftrightarrow} &= \sum_{j\neq i} (A_{ij}A_{ji}) \qquad\qquad \forall i. \end{aligned} \tag{2.23}$$

Described in words, each node in a graph has an out-only-, in-only- and reciprocal degree. The out-only-degree is equal to the number of outgoing edges that do not have an incoming counterpart, the in-only-degree is equal to the number of incoming edges that do not have an outgoing counterpart and the reciprocal-degree is equal to the number of outgoing edges that have an incoming counterpart.

The likelihood function is defined in equation (2.24) and the expected out-only-, in-only- and reciprocal-degrees are defined in equation (2.25), where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ consist of the hidden variables.

$$\mathcal{L}(\mathbf{x},\mathbf{y},\mathbf{z}) = \sum_i \Big(k_i^{\rightarrow}\log(x_i) + k_i^{\leftarrow}\log(y_i) + k_i^{\leftrightarrow}\log(z_i) - \sum_{j>i}\log(1+x_iy_j+x_jy_i+z_iz_j)\Big) \tag{2.24}$$

$$\begin{aligned} \langle k_i^{\rightarrow}\rangle(\mathbf{x},\mathbf{y},\mathbf{z}) &= \sum_{j\neq i} \frac{x_iy_j}{1+x_iy_j+x_jy_i+z_iz_j} \quad \forall i \\ \langle k_i^{\leftarrow}\rangle(\mathbf{x},\mathbf{y},\mathbf{z}) &= \sum_{j\neq i} \frac{x_jy_i}{1+x_iy_j+x_jy_i+z_iz_j} \quad \forall i \\ \langle k_i^{\leftrightarrow}\rangle(\mathbf{x},\mathbf{y},\mathbf{z}) &= \sum_{j\neq i} \frac{z_iz_j}{1+x_iy_j+x_jy_i+z_iz_j} \quad \forall i \end{aligned} \tag{2.25}$$

The search space of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ is constrained by the bounds:

$$\begin{aligned} x_i &\geq 0 \quad \forall i, \\ y_i &\geq 0 \quad \forall i \text{ and} \\ z_i &\geq 0 \quad \forall i. \end{aligned} \tag{2.26}$$

### 2.2.7 Reciprocal Weighted Configuration Model

The reciprocal weighted configuration model (RWCM) constrains the ensemble on the out-only-strength sequence $\mathbf{s}^{\rightarrow}$, the in-only-strength sequence $\mathbf{s}^{\leftarrow}$ and the reciprocal-strength sequence $\mathbf{s}^{\leftrightarrow}$ defined as:

$$\begin{aligned} s_i^{\rightarrow} &= \sum_{j\neq i} \big(W_{ij} - \min(W_{ij},W_{ji})\big) \quad \forall i, \\ s_i^{\leftarrow} &= \sum_{j\neq i} \big(W_{ji} - \min(W_{ij},W_{ji})\big) \quad \forall i \text{ and} \\ s_i^{\leftrightarrow} &= \sum_{j\neq i} \min(W_{ij},W_{ji}) \qquad\qquad \forall i. \end{aligned} \tag{2.27}$$

Described in words, each node in a graph has an out-only-, in-only- and reciprocal strength, which are the weighted variants of the out-only-, in-only and reciprocal-degree of that node.

The likelihood function is defined in equation (2.28) and the expected out-only-, in-only- and reciprocal-strengths are defined in equation (2.29), where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ consist of the hidden variables.

$$\begin{aligned}\mathcal{L}(\mathbf{x},\mathbf{y},\mathbf{z}) = \sum_i \Big( & s_i^{\rightarrow}\log(x_i) + s_i^{\leftarrow}\log(y_i) + s_i^{\leftrightarrow}\log(z_i) + \\ & \sum_{j>i}\big(\log(1-x_iy_j) + \log(1-x_jy_i) + \log(1-z_iz_j) - \log(1-x_iy_jx_jy_i)\big)\Big)\end{aligned} \tag{2.28}$$

$$\begin{aligned}\langle s_i^{\rightarrow}\rangle(\mathbf{x},\mathbf{y},\mathbf{z}) &= \sum_{j\neq i}\frac{x_iy_j(1-x_jy_i)}{(1-x_iy_j)(1-x_ix_jy_iy_j)} && \forall i\\ \langle s_i^{\leftarrow}\rangle(\mathbf{x},\mathbf{y},\mathbf{z}) &= \sum_{j\neq i}\frac{x_jy_i(1-x_iy_j)}{(1-x_jy_i)(1-x_ix_jy_iy_j)} && \forall i\\ \langle s_i^{\leftrightarrow}\rangle(\mathbf{x},\mathbf{y},\mathbf{z}) &= \sum_{j\neq i}\frac{z_iz_j}{1-z_iz_j} && \forall i\end{aligned} \tag{2.29}$$

The search space of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ is constrained by the bounds:

$$\begin{aligned}0 \leq x_i < 1 &\quad \forall i,\\ 0 \leq y_i < 1 &\quad \forall i \text{ and}\\ 0 \leq z_i < 1 &\quad \forall i.\end{aligned} \tag{2.30}$$

## 2.3 Optimality Measurement

The current software implementation defines a measurement for optimality in the form of an error function, where the error indicates the distance from the optimum. So an error of zero means optimal. This error function is defined in equation (2.31), where $\oslash$ is a component-wise division.

$$\text{err}(\mathbf{h}) = \max\Big(\big|\mathbf{c} - \langle\mathbf{c}\rangle\big| \oslash \mathbf{c}\Big) \tag{2.31}$$

The error function looks at the difference between the observed and expected ensemble constraints given by the hidden variables, which seems to be a very intuitive measure. Furthermore the difference is scaled by the observed constraints to compensate for very heterogeneous ensemble constraints. To condense the resulting vector into a single scalar the maximum value is taken. It is more common to take the norm of a vector for this purpose, but this approach is acceptable.

A vector $\mathbf{h}$ can be seen as optimal once $\text{err}(\mathbf{h})$ is below a user-specified threshold $\varepsilon$.

# Chapter 3

# Mathematical Model

Because both problems (2.1) and (2.2) have the same solution, our idea is to focus on the optimization problem only. The reasoning is that if a good approach can be defined to solve the optimization problem, a solution is already found and no further solving of equations is necessary. Focusing on one problem avoids needless complexity.

The optimization problem (2.1) is redefined in the general case as:

$$\begin{array}{lll} \max\limits_{\mathbf{h}} & \mathcal{L}(\mathbf{h}) & \\ \text{subject to} & g_\ell(\mathbf{h}) \leq 0 & \forall \ell \\ & h_i^{\min} \leq h_i < h_i^{\max} & \forall i. \end{array} \tag{3.1}$$

Here each element of $\mathbf{g}\colon \mathbb{R}^N \to \mathbb{R}^M$ is a non-linear inequality constraint function on the search space.

The origin of these new constraints is discussed in Section 3.1. Furthermore, derivatives have been added to the model (Section 3.2) and all new terms are defined for the configuration models in Section 3.3.

## 3.1 Search-Space Constraints

The constraints on the search space defined in Chapter 2 are put in place to make sure that $\mathcal{L}$ stays in the realm of real numbers and that the values of $\langle \mathbf{c} \rangle$ make sense. However, we determined that these bounds have been defined too narrow.

$\mathcal{L}$ leaving the realm of real numbers occurs when the logarithms in the function take negative numbers. Taking UWCM as an example and looking at the logarithms, we have $\log(x_i)$ and $\log(1 - x_i x_j)$. To prevent the first logarithm from taking negative numbers the constraint $x_i \geq 0$ makes sense. For the second logarithm this is not only the case if the constraints $x_i < 1$ and $x_j < 1$ are satisfied, but also if the constraint $x_i x_j < 1$ is satisfied. Using our new constraint results in a much wider search space, as illustrated in Figure 3.1.

Early experiments showed that the optimum is often found in the space that was first made inaccessible by the current constraints. These improved non-linear constraints can be defined for UWCM, UMCM, DWCM and RWCM and are included in the model.

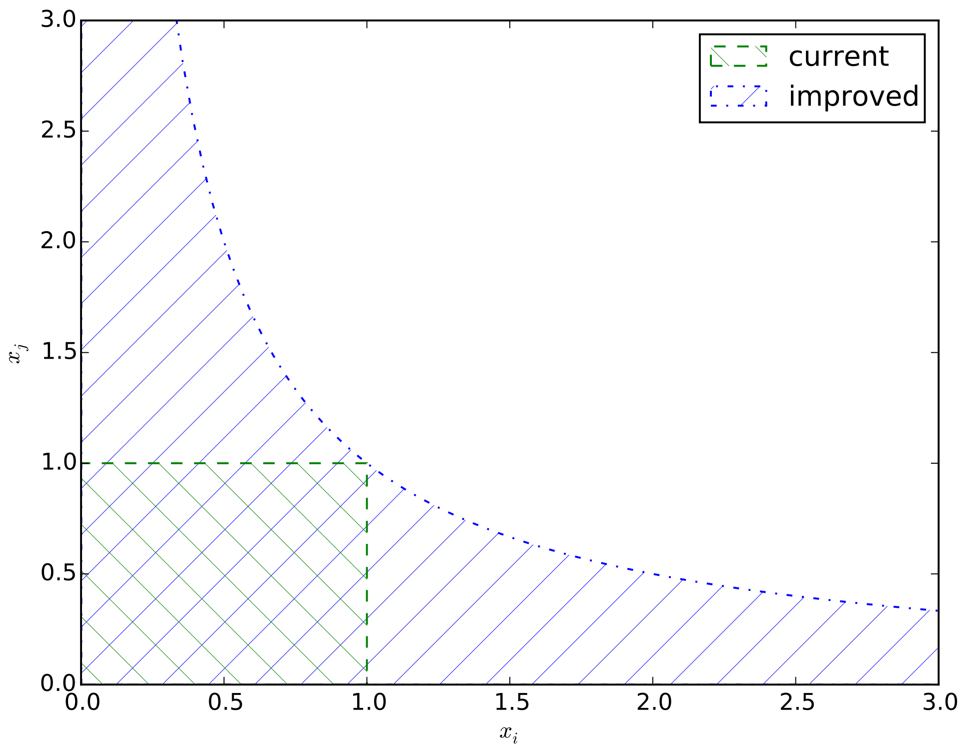


Figure 3.1: Two-dimensional search space of UWCM showing the search space constrained by the current constraints and the improved constraints.

### 3.1.1 Scalability

Our improved non-linear constraints come with a price however. Taking UWCM as an example again, we now have a constraint on the product $x_i x_j$ for all $i$ and all $j > i$, so $M = \frac{n(n-1)}{2}$. The number of these constraints scales close to quadratically with $n$ and storing their values will result in memory issues for larger graphs (say 50,000 nodes and up).

To solve this issue, better scalable constraints should be specified. To do this for UWCM we use the following claim. If $\mathbf{s}$ satisfies:

$$s_1 \leq s_2 \leq \ldots \leq s_n, \tag{3.2}$$

then $\mathbf{x}$ at the optimum satisfies:

$$x_1 \leq x_2 \leq \ldots \leq x_n. \tag{3.3}$$

We have proved that this claim is true in Appendix A.

The constraint $x_i x_j < 1$ shows that only one $x_i$ can be greater than 1 and thanks to the claim its index can be determined by looking at $\mathbf{s}$. Defining that index as $i_{\max}$ we can define the following more scalable non-linear constraints:

$$x_i < \frac{1}{x_{i_{\max}}} \quad \forall i \neq i_{\max}. \tag{3.4}$$

This same analysis can be applied to UMCM. However, for DWCM and RWCM we currently do not have better scalable constraints.

## 3.2 Derivatives

Many solvers require the first and second derivatives of the objective function and of the non-linear search space constraints. These derivatives can be derived analytically or approximated. The gradient $\nabla\mathcal{L}$ (a vector) consists of all first derivatives of our objective function $\mathcal{L}$ and the Hessian $\nabla^2\mathcal{L}$ (a symmetric matrix) consists of all second derivatives.

The current implementation uses the approximations finite differences for the gradient and BFGS for the Hessian. For the improved implementation we have derived the required gradients and Hessians analytically which has several benefits.

First of all, the finite differences method requires $N$ evaluations of $\mathcal{L}$ to make its approximation. Since the complexity of $\mathcal{L}$ is $\mathcal{O}(n^2)$ this results in a complexity of $\mathcal{O}(n^3)$ at every iteration for gradient approximations. The complexity of the analytical gradient is $\mathcal{O}(n^2)$, which scales much better.

Secondly, since the improved implementation no longer uses approximations for the derivatives but the exact values, the solver should converge better. This difference in convergence is displayed in Figure 3.2.

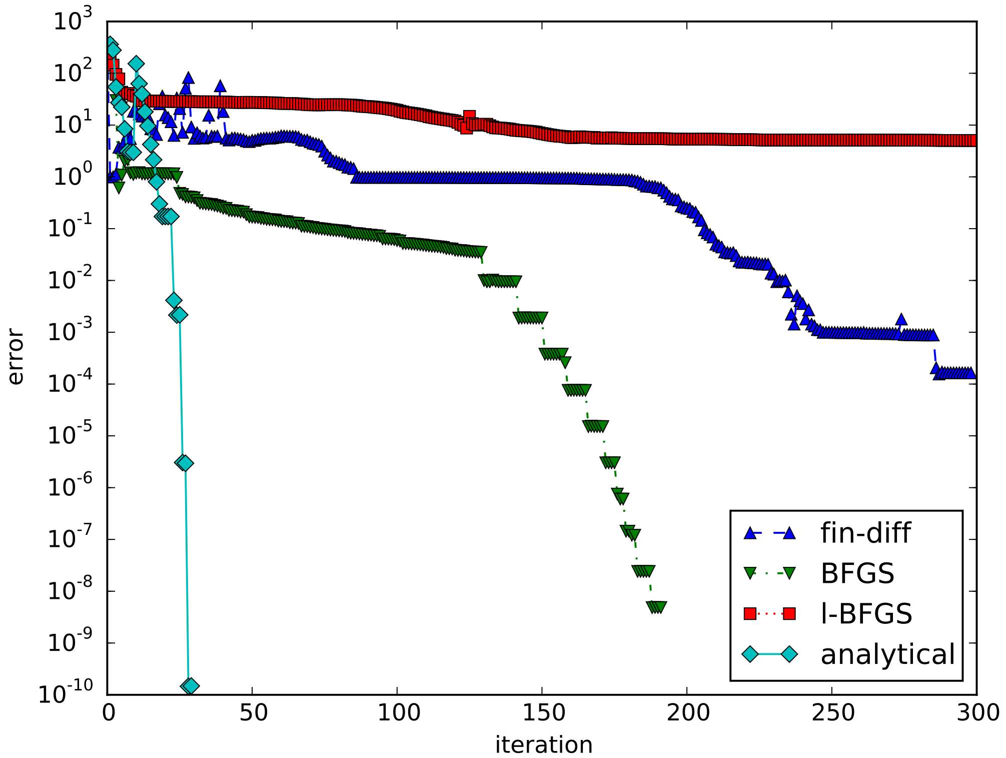


Figure 3.2: Convergence of UWCM for the ITN graph (further explained in Section 6.1.2) using an Interior-Point solver with approximate and analytical derivatives. Error on the vertical axis corresponds to the error function (Equation (2.31)). (While this is only the convergence for one model and graph, other models and graphs give similar results.) The figure shows finite differences for the gradient and BFGS for the Hessian (fin-diff), analytical for the gradient and BFGS for the Hessian (BFGS), analytical for the gradient and l-BFGS for the Hessian (l-BFGS) and all derivatives analytically (analytical).

A downside of the analytical Hessian is that the matrix is very dense, which means that for larger graphs (say 50,000 nodes and up) this will result in memory issues. The limited memory variant of the BFGS approximation (l-BFGS) could be a solution in these cases. However, Figure 3.2 shows that the l-BFGS approximation performs badly for the maximum likelihood problems. The convergence differences between analytical, BFGS and l-BFGS show us that noise in the Hessian (due to approximation) has a substantial effect on the performance for this problem.

## 3.3 Configuration Model Additions

We have improved the configuration models with more accurate constraints and added derivatives. These changes are defined for the configuration models in Sections 3.3.1 to 3.3.7.

### 3.3.1 Undirected Binary Configuration Model

For UBCM we have not changed the search space constraints, since they were already correct.

We have derived and added the first and second partial derivatives of $\mathcal{L}$, which are further defined in Appendix B.1. This results in the vector $\nabla\mathcal{L}$ with $n$ non-zero elements and the matrix $\nabla^2\mathcal{L}$ with $n^2$ non-zero elements.

### 3.3.2 Undirected Weighted Configuration Model

For UWCM we have redefined the bounds on the search space to:

$$\begin{aligned} x_i &\geq 0 \quad \forall i \text{ and} \\ x_i &< 1 \quad \forall i \neq i_{\max}. \end{aligned} \tag{3.5}$$

Furthermore, we introduce $M = n - 1$ non-linear constraints on the search space. Equation (3.6) defines these constraints as the sequence of functions $\mathbf{g}$, written in the standard form of the problem definition (3.1).

$$g_\ell(\mathbf{x}) = x_i - \frac{1}{x_{i_{\max}}} \quad \forall i \neq i_{\max} \tag{3.6}$$

We have derived and added the first and second partial derivatives of $\mathcal{L}$ and $g_\ell$, which are further defined in Appendix B.2. This results in the vector $\nabla\mathcal{L}$ with $n$ non-zero elements, the matrix $\nabla^2\mathcal{L}$ with $n^2$ non-zero elements, the $M$ vectors $\nabla g_\ell$ with 2 non-zero elements each and the $M$ matrices $\nabla^2 g_\ell$ with 1 non-zero element each.

### 3.3.3 Undirected Mixed Configuration Model

For UMCM we have redefined the bounds on the search space to:

$$\begin{aligned} x_i &\geq 0 \quad \forall i, \\ y_i &\geq 0 \quad \forall i \text{ and} \\ y_i &< 1 \quad \forall i \neq i_{\max}. \end{aligned} \tag{3.7}$$

Furthermore, we introduce $M = n - 1$ non-linear constraints on the search space. Equation (3.8) defines these constraints as the sequence of functions $\mathbf{g}$, written in the standard form of the problem definition (3.1).

$$g_\ell(\mathbf{y}) = y_i - \frac{1}{y_{i_{\max}}} \quad \forall i \neq i_{\max} \tag{3.8}$$

We have derived and added the first and second partial derivatives of $\mathcal{L}$ and $g_\ell$, which are further defined in Appendix B.3. This results in the vector $\nabla\mathcal{L}$ with $2n$ non-zero elements, the matrix $\nabla^2\mathcal{L}$ with $(2n)^2$ non-zero elements, the $M$ vectors $\nabla g_\ell$ with 2 non-zero elements each and the $M$ matrices $\nabla^2 g_\ell$ with 1 non-zero element each.

### 3.3.4 Directed Binary Configuration Model

For DBCM we have not changed the search space constraints, since they were already correct.

We have derived and added the first and second partial derivatives of $\mathcal{L}$, which are further defined in Appendix B.4. This results in the vector $\nabla\mathcal{L}$ with $2n$ non-zero elements and the matrix $\nabla^2\mathcal{L}$ with $2n + n(n-1)$ non-zero elements.

### 3.3.5 Directed Weighted Configuration Model

For DWCM we have redefined the bounds on the search space to:

$$\begin{aligned} x_i &> 0 \quad \forall i \text{ and} \\ y_i &> 0 \quad \forall i. \end{aligned} \tag{3.9}$$

Furthermore, we introduce $M = n(n-1)$ non-linear constraints on the search space. Equation (3.10) defines these constraints as the sequence of functions $\mathbf{g}$, written in the standard form of the problem definition (3.1).

$$g_\ell(\mathbf{x}, \mathbf{y}) = x_i y_j - 1 \quad \forall i \neq j \tag{3.10}$$

We have derived and added the first and second partial derivatives of $\mathcal{L}$ and $g_\ell$, which are further defined in Appendix B.5. This results in the vector $\nabla\mathcal{L}$ with $2n$ non-zero elements, the matrix $\nabla^2\mathcal{L}$ with $2n + n(n-1)$ non-zero elements, the $M$ vectors $\nabla g_\ell$ with 2 non-zero elements each and the $M$ matrices $\nabla^2 g_\ell$ with 1 non-zero element each.

### 3.3.6 Reciprocal Binary Configuration Model

For RBCM we have not changed the search space constraints, since they were already correct.

We have derived and added the first and second partial derivatives of $\mathcal{L}$, which are further defined in Appendix B.6. This results in the vector $\nabla\mathcal{L}$ with $3n$ non-zero elements and the matrix $\nabla^2\mathcal{L}$ with $(3n)^2$ non-zero elements.

### 3.3.7 Reciprocal Weighted Configuration Model

For RMCM we have redefined the bounds on the search space to:

$$\begin{aligned} x_i &> 0 \quad \forall i, \\ y_i &> 0 \quad \forall i \text{ and} \\ z_i &> 0 \quad \forall i. \end{aligned} \tag{3.11}$$

Furthermore, we introduce $M = 1.5n(n-1)$ non-linear constraints on the search space. Equation (3.12) defines these constraints as the sequence of functions $\mathbf{g}$, written in the standard form of the problem definition (3.1).

$$\begin{aligned} g_\ell(\mathbf{x}, \mathbf{y}) &= x_i y_j - 1 \quad && \forall i \neq j \\ g_\ell(\mathbf{z}) &= z_i z_j - 1 && \forall j > i \end{aligned} \tag{3.12}$$

We have derived and added the first and second partial derivatives of $\mathcal{L}$ and $g_\ell$, which are further defined in Appendix B.7. This results in the vector $\nabla\mathcal{L}$ with $3n$ non-zero elements, the matrix $\nabla^2\mathcal{L}$ with $4n + 5\frac{n(n-1)}{2}$ non-zero elements, the $M$ vectors $\nabla g_\ell$ with 2 non-zero elements each and the $M$ matrices $\nabla^2 g_\ell$ with 1 non-zero element each.

# Chapter 4

# Solution Algorithm

To find the solution to the problem specified in Chapter 2 (and extended in Chapter 3) a strategy is needed. For various aspects, we analyze the approach of the current implementation and give details on how the improved implementation circumvents the bottlenecks.

## 4.1 Solvers

The current implementation tries to solve both problem (2.1) and (2.2) sequentially, using the outcome of the first solve to start the second solve. For the optimization problem an Interior-point method [7] is used. For the equation solving a Trust-region method [8] is used that attempts to minimize the sum of squares of the system.

The choice of these solvers is strange in the sense that the Interior-point method is very popular for constrained non-linear optimization, but the current implementation has no non-linear constraints, only simple bounds. Furthermore, the Trust-region method used is not able to handle bounds on the search space. Since the Trust-region method is applied after the Interior-point method this means that the algorithm could start looking in the infeasible space for a solution.

The procedure taken by the current implementation is shown in Algorithm 1, where maxEval is a budget on the number of function evaluations that the solver may use before it terminates. InteriorPoint and TrustRegion are abstractions of the respective solvers, where the first argument is the problem to solve and the second argument the starting point.

**Algorithm 1** Find hidden variables $\mathbf{h}$ with maximum error $\varepsilon$

```
for all $i$ do
  $h_i \leftarrow 0.9$
end for
$\mathbf{h} \leftarrow \text{InteriorPoint}(\max \mathcal{L}, \mathbf{h}, \text{maxEval} = 10,000)$
if $\text{err}(\mathbf{h}) > \varepsilon$ then
  $\mathbf{h} \leftarrow \text{TrustRegion}(\langle \mathbf{c} \rangle = \mathbf{c}, \mathbf{h}, \text{maxEval} = 10,000)$
  if $\text{err}(\mathbf{h}) > \varepsilon$ then
    $\mathbf{h} \leftarrow \text{TrustRegion}(\langle \mathbf{c} \rangle = \mathbf{c}, \mathbf{h}, \text{maxEval} = 100,000)$
  end if
end if
return $\mathbf{h}$
```

To solve the problem (3.1) in the improved implementation, we have used several solvers with varying results. These are detailed in Sections 4.1.1 to 4.1.3.

### 4.1.1 Gradient Descent

Gradient Descent is a relatively simple solver that iteratively updates its current solution in the direction of the gradient of the objective function until it reaches a local minimum. Algorithm 2 shows the approach that we used.

**Algorithm 2** Gradient Descent

```
$\mathbf{h} \leftarrow$ some starting point
$\mathbf{h}^* \leftarrow \mathbf{h}$
while not terminate do
  $\mathbf{h}^* \leftarrow \mathbf{h}^* - \frac{\alpha}{N}\big(-\nabla\mathcal{L}(\mathbf{h}^*)\big)$
  if $-\mathcal{L}(\mathbf{h}) > -\mathcal{L}(\mathbf{h}^*)$ and not constraints violated then
    $\mathbf{h} \leftarrow \mathbf{h}^*$
    $\alpha \leftarrow 1.26\,\alpha$
  else
    $\mathbf{h}^* \leftarrow \mathbf{h}$
    $\alpha \leftarrow 0.096\,\alpha$
  end if
end while
return $\mathbf{h}$
```

Here we update the step size $\alpha$ using a "bold driver" algorithm, where 1.26 and 0.096 are numbers that happened to work well after a few experiments. The constraints are enforced after every update.

While the solver is able to find the optimum of the likelihood function, the number of iterations required grows exponentially with the number of nodes. Already requiring millions of iterations for a graph of 16 nodes, which makes this solver inadequate.

### 4.1.2 Conjugate Gradient

A non-linear Conjugate Gradient solver is at its core similar to the Gradient Descent solver. However, it improves the update rule by determining optimal values for $\alpha$ using line search instead of the bold driver algorithm. Furthermore, an extra term $\beta$ is added to the update of the solution.

Our implementation is based on [9] and uses Strong-Wolfe line search and combined Polak-Ribiere Fletcher-Reeves for $\beta$.

While this solver is more advanced than Gradient Descent and is known to have better convergence, for this problem the results do not differ much from Gradient Descent.

### 4.1.3 Interior Point

The Interior-point solver [7] is fundamentally different in the way it handles the constraints, which is by finding a solution that satisfies the Karush-Kuhn-Tucker (KKT) conditions (equation (4.1)), where the elements of $\boldsymbol{\mu}$ are called the KKT multipliers, which must exist if $\mathbf{h}$ is optimal.

$$\nabla\mathcal{L}(\mathbf{h}) - \sum_{\ell} \mu_\ell \nabla g_\ell(\mathbf{h}) = 0 \tag{4.1}$$

For the Interior-point method this is done by approximating the problem with a linear system of equations in a neighborhood around the current **h** and solving this system. By repeating this process over several iterations an optimum is found.

Early experiments showed us that this solver works considerably better than the Gradient Descent and Conjugate Gradient solvers and we therefore use it throughout the rest of this work.

## 4.2 Termination Criteria

Algorithm 1 shows how the current implementation uses relatively few checks of the error function from equation (2.31). Also the solvers that are run in between the checks have very strict termination tolerances ($10^{-32}$). Since this is lower than what can reasonably be expected from double precision floating point numbers, it causes the solver to run until its evaluation budget has been used in most cases.

An approach could be to run the error function after each iteration and check if the solver has converged. However, the complexity of the error function is $\mathcal{O}(n^2)$, so running it every iteration comes with quite some extra computational overhead.

Early experiments showed that the norm of the gradient $||\nabla\mathcal{L}||$ is somewhat proportional to the error function. Since the gradient is computed each iteration anyway, we take the approach in Algorithm 3 to reduce the number of calls to the error function.

**Algorithm 3** Do iterations of the solver until err(**h**) is below $\varepsilon$

```
$E \leftarrow 10\,\varepsilon$
while not terminate do
  $\mathbf{h}, \nabla\mathcal{L} \leftarrow$ do generic solver iteration
  if $||\nabla\mathcal{L}|| \leq E$ then
    if $\mathrm{err}(\mathbf{h}) \leq \varepsilon$ then
      return $\mathbf{h}$
    end if
    $E \leftarrow 0.1\,E$
  end if
end while
```

The “not terminate” could be a user specified criteria such as a maximum number of iterations or function evaluations.

## 4.3 Starting Point

The current implementation uses a fixed starting point of 0.9. Early experiments showed that most components of an optimal **h** are somewhere around 0.9, making this a reasonable starting point.

Our improved implementation specifies two more methods for determining a starting point. For the first method every component of **h** is picked from a uniform random distribution between 0 and 1. The second method is inspired by the claim from equations (3.2) and (3.3) and uses the scaled ensemble constraints to get an approximation of the optimum:

$$h_i = \frac{c_i}{\max(\mathbf{c})} \quad \forall i. \tag{4.2}$$

Which starting point method works best depends heavily on the input graph, making it difficult to specify which method should be used. However, in the rare case where the improved implementation does not converge, choosing a different starting point method often helps.

## 4.4 Unconnected Nodes

Early experiments showed that trying to solve the problem for graphs with unconnected nodes led to convergence issues. Specifically, if node $i$ is unconnected the value of $\frac{\partial \mathcal{L}}{\partial h_i}$ would not get near zero (not near optimal) during the solver iterations. Further research showed us that this same issue occurred not only for unconnected nodes but for any ensemble constraint that is equal to 0.

The optimal value of a hidden variable with a respective ensemble constraint equal to 0 is 0. This makes sense because any edges in the graph that are related to an ensemble constraint should not exist if the ensemble constraint equals 0. Hence the probability for these edges should be 0, which can be achieved by setting the respective hidden variable to 0.

For this reason our improved implementation constrains these hidden variables to be equal to 0, effectively removing them from the optimization problem.

# Chapter 5

# Implementation

The current software is implemented in MATLAB and heavily depends on MATLAB's Optimization Toolbox for the implementation of its solvers, whereas the improved software is implemented in C/C++. For the current and improved implementation this chapter discusses the programming environment (Section 5.1), the software framework (Section 5.2), the solver (Section 5.3) and the parallelization of parts of the implementation (Section 5.4).

## 5.1 Environment

Software implemented in MATLAB can run on many platforms. To not lose this cross-platform advantage when transitioning to C/C++, we use CMake (cross-platform make). CMake generates GNU make files on Linux, Visual Studio projects on Windows, etc., allowing the C/C++ software to be compiled and run on many platforms as well.

We build the improved software as a library with a C interface. This setup allows the software to be used from many different programming languages and environments, including FORTRAN, C/C++, Python, MATLAB, Java and many more.

We also equipped the improved software with an automated test suite, including regression and unit tests. This allows us to easily check if all components are still working correctly after changes are made to the software. We implemented the test suite using the GTest framework.

## 5.2 Software Framework

While the current software is implemented in a procedural way, we take an object-oriented approach for the improved software. The idea is to make sure that the software can be extended and improved more easily this way.

Figure 5.1 shows the conceptual class diagram of the improved software. The configuration model (CM) class is an abstract interface to functions such as the likelihood, derivatives and error and the implementation of these functions is defined in the specific underlying configuration model. This allows the solver to interface with the configuration model independent of which specific model is solved. The solver (Solver) class itself is also an abstract interface, allowing for many different implementations in the specific underlying solvers. Furthermore, functionality that can be shared between solvers, such as determining a starting point and termination criteria is implemented in the solver base class.

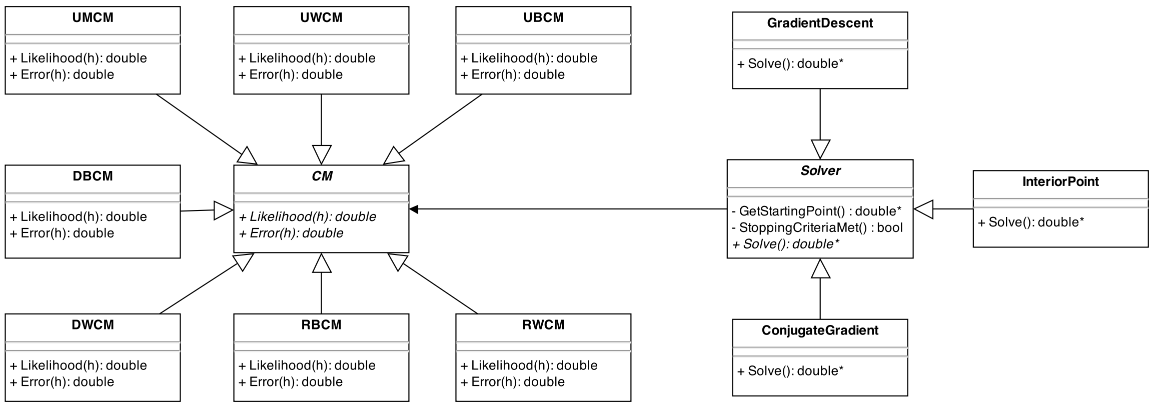


Figure 5.1: Conceptual UML class diagram of the current software implementation.

The main advantage of this object-oriented approach is that new configuration models and solvers can now be added to the software without having to interfere with other components of the software.

## 5.3 Solver

The Interior-point [10] and Trust-region [11] solvers used by the current software are implemented by MATLAB's optimization toolbox. The main disadvantage of this toolbox is that it is not free to use, in the case of MATLAB meaning that users of the software must also purchase the toolbox in order for the software to work.

In the improved software we use the free and open-source C++ library Ipopt [12] for its Interior-point solver. Ipopt is widely-used and a well-known implementation of the Interior-point method. Its good reputation and easy to use C++ interface make this our preferred implementation over other libraries. It uses the BLAS and LAPACK libraries for high-performance linear-algebra operations and the METIS library for matrix ordering.

At every iteration of Ipopt's algorithm a linear system of equations has to be solved. Ipopt can interface with various third-party linear solvers to solve this system. Experiments showed that, for the maximum likelihood problems, the approach of [13] had the best effect on Ipopt's convergence.

We store all matrices in the sparse format to save memory. This means that we only store the non-zero values and their indices and assume that all values not stored are 0. Furthermore, for symmetric matrices we do not have to store anything left of the diagonal since we can determine these values by looking at the values to the right of the diagonal. Especially for the matrix $\nabla \mathbf{g}$ (the $N \times M$ matrix of all $\nabla g_\ell$ vectors) this saves a lot of memory since it only has $2M$ non-zeros.

## 5.4 Parallelization

Many operations executed by Ipopt are implemented in a parallel way, using OpenMP to benefit from all CPU-cores in a user's machine. Even though the C/C++ implementation of the function evaluations is already a lot faster than their MATLAB counterparts, we noticed that the bigger the problem size, the bigger portion of the run time would be in function evaluations. For this reason we implemented the Likelihood, gradient, Hessian and error functions in a parallel way, using OpenMP as well.

The implementation of all function evaluations consists of an outer loop and inner loop (causing their quadratic scaling). Here the loops are either iterating over the elements of the gradient/Hessian or are iterations of a summation. For the parallel implementation we split the work of the outer loop over the available CPU-cores more or less equally. We do this using OpenMP's static or dynamic scheduling, depending on the shape of the inner loop.

For example, a common summation is $\sum_i \sum_{j>i}$, causing the workload of a single iteration of the outer summation to vary a lot. In these cases we use dynamic scheduling to balance the workload. In other cases a common summation is $\sum_i \sum_{j\neq i}$, which has a lot more balanced workload at each iteration of the outer summation. In these cases we use static scheduling to save the overhead cost of dynamic scheduling.

# Chapter 6

# Experiments and Results

We have carried out many experiments to determine the performance of the improved software. This chapter gives an overview of the most important ones.

All experiments involving run times have been carried out on a Linux machine with an Intel Core i5-3550 CPU, at a clock speed of 3.3 GHz and with 8 GB of RAM. For all experiments we set the user defined error threshold to $\varepsilon = 10^{-6}$.

Section 6.1 describes the data we used for the experiments, Section 6.2 compares the performance of the current and improved software, Section 6.3 discusses the speedup of the function evaluations and the overall performance of the improved software is detailed in Section 6.4.

## 6.1 Data

The graph data that we used for the experiments can be divided into two categories, generated graphs (Section 6.1.1) and a real-world graph (Section 6.1.2).

### 6.1.1 Generated Graphs

To get a large number of graphs of varying sizes for testing purposes, we implemented a method for generating graphs. The method for determining the edge probabilities and sampling is taken from [4]. The method requires a hidden variable sequence as input. We draw this sequence at random from a beta distribution with parameters $\alpha = 5$ and $\beta = 1$. This gives a distribution that closely resembles hidden variable sequences that are often found at the maximum of $\mathcal{L}$.

We generate several graphs, for sizes ranging from 100 to 5,000 nodes and for each configuration model.

### 6.1.2 International Trade Network

The International Trade Network (ITN) data set features undirected weighted graphs, where each node represents a country and the weight of an edge represents the trade between two countries. The amount of trade is extracted from the import/export databases publicly available at [14].

The data set spans the years 1950 to 2000, with one graph every 10 year jump. These graphs vary in sizes over the years, from 85 and up to 183 nodes.

## 6.2 Comparing current and improved

Figure 6.1 shows the time required by the current and improved software to find a solution for generated graphs of varying sizes. To get a quantitative measurement of the scaling, we fit the data points to a power law function using a least squares method. While the scaling has not been improved, the run times of the improved method are significantly faster. For example, for 500 nodes the run time of the current method is 3,264 seconds, while the run time of the improved method is only 6 seconds, which is more than 500 times faster.

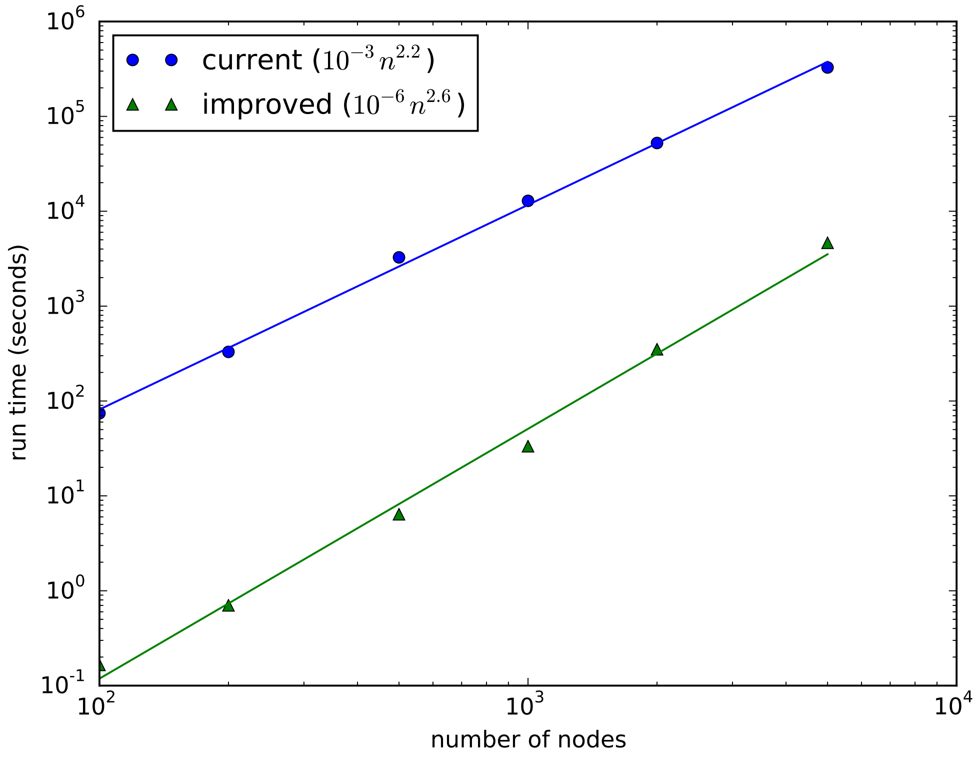


Figure 6.1: Required time (in seconds) for the software (current vs improved) to find a solution. This is done using generated graphs, up to 5000 nodes, for the Undirected Mixed Configuration Model. (Which is one of the more difficult models with respect to convergence.) The data points have been fit to power law functions ($t(n) = \alpha n^k$). Note the logarithmic scale on both axes.

Table 6.1 shows the error of the solutions found by the current and improved software. The improved method always finds a good solution while the current method's results seem a bit random. Also, increasing the evaluation budget of the current method does not result in lower errors.

Figure 6.2 shows the time required by the current and improved software to find a solution for the graphs of the ITN data. Table 6.2 shows the respective error of the solution. The results are similar to the generated graphs. (Note that the difference in nodes is smaller here, only 85 to 183). The improved method always finds a good solution while the current method never does. Same as with the generated graphs, increasing the evaluation budget of the current method does not result in lower errors.

| number of nodes | err($\mathbf{h}$) current | err($\mathbf{h}$) improved |
|---|---|---|
| 100 | $1.681\times10^{-15}$ | $2.691\times10^{-8}$ |
| 200 | $1.038\times10^{-14}$ | $1.592\times10^{-7}$ |
| 500 | $1.001\times10^{0}$ | $5.261\times10^{-7}$ |
| 1000 | $2.560\times10^{-14}$ | $6.368\times10^{-7}$ |
| 2000 | $8.158\times10^{-1}$ | $1.055\times10^{-10}$ |
| 5000 | $2.681\times10^{0}$ | $5.238\times10^{-10}$ |

Table 6.1: Error of the solution found by the software (current vs improved). This is done using generated graphs, up to 5000 nodes, for the Undirected Mixed Configuration Model. Everything greater than $\varepsilon$ means no acceptable solution is found.

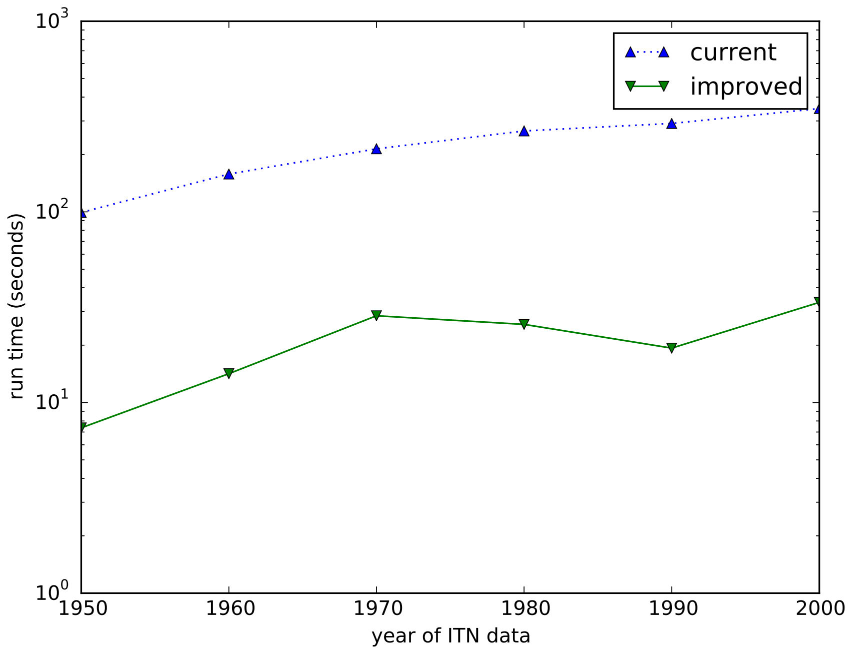


Figure 6.2: Required time (in seconds) for the software (current vs improved) to find a solution. This is done using the ITN graphs for the Undirected Mixed Configuration Model. Note the logarithmic scale on the run-time axis.

| year of ITN data | err($\mathbf{h}$) current | err($\mathbf{h}$) improved |
|---|---|---|
| 1950 | $2.659\times10^{0}$ | $6.192\times10^{-9}$ |
| 1960 | $7.598\times10^{1}$ | $5.803\times10^{-9}$ |
| 1970 | $3.127\times10^{1}$ | $1.381\times10^{-8}$ |
| 1980 | $3.806\times10^{1}$ | $5.674\times10^{-9}$ |
| 1990 | $1.164\times10^{2}$ | $2.245\times10^{-9}$ |
| 2000 | $9.776\times10^{0}$ | $8.093\times10^{-9}$ |

Table 6.2: Error of the solution found by the software (current vs improved). This is done using the ITN graphs for the Undirected Mixed Configuration Model. Everything greater than $\varepsilon$ means no acceptable solution is found.

The most important result here is that the improved software always finds an acceptable solution, which is mainly caused by two factors. Because of the improved search space constraints (Section 3.1) the improved software searches areas of the search space that are inaccessible in the current software. Furthermore, we showed that the use of analytical derivatives (Section 3.2) helps in finding solutions with a lower error.

Notable is that for some cases the error of the current software is lower than the error of the improved software. This is mainly caused by the improved termination criteria (Section 4.2). The improved software is more likely to terminate when a solution with an error lower than $\varepsilon$ is reached, while the current software will often run until its evaluation budget is spent.

Remarkable is that the scaling did not improve, in fact the current implementation has better scaling than the improved implementation. This is caused by the function evaluations required by the finite-differences approximations (Section 3.2) and the evaluation budget. Starting from 5,000 nodes, the current implementation needs so many function evaluations per iteration that it can only finish one iteration before the budget of 10,000 is used. This means that the run-time scaling will be limited, but a good solution will never be found.

## 6.3 Function evaluations

Figure 6.3 shows the speedup of the parallelized likelihood, gradient, Hessian and error functions, where speedup is defined as the run time of the sequential version divided by the run time of the parallel version. From 100 nodes onwards, the speed up is pretty constant and close to optimal, close to 4 times faster on 4 cores.

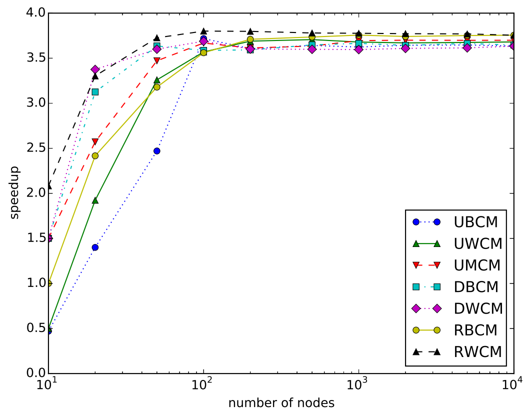
Only for very small graphs the parallelized version performs worse than the sequential version. This is likely due to the overhead of the parallelization (creating threads, distributing work, etc.) being significant compared to the computation itself for these small graphs.

## 6.4 Overall performance

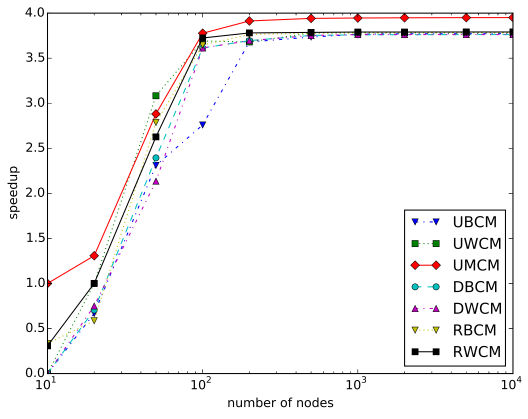
Figure 6.4 shows the time required by the improved software to find a solution for generated graphs of varying sizes for all models. Some models do not have measurements for 5,000 nodes because of memory limitations. To get a quantitative measurement of the scaling, the data points have been fit to a power law function using a least squares method. Table 6.5 shows the respective error of the solution. The scaling of all models is similar and the improved method always finds a good solution.

Interesting is that the number of iterations required is more or less constant for different numbers of nodes. So the quadratic scaling is due to the complexity of calculations at every iteration. This means that further improving the convergence could improve the run times, but will not dramatically improve the scaling.

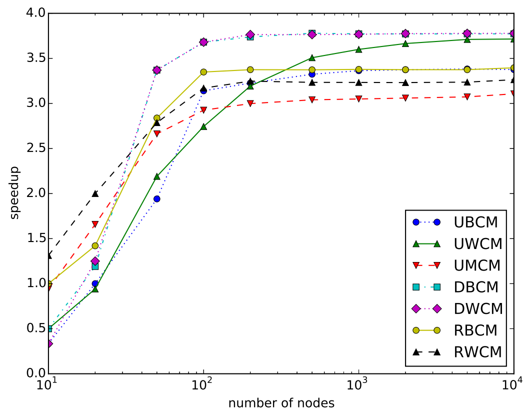
Furthermore, DWCM and RWCM perform somewhat worse than the other models. This is likely due to their many non-linear constraint and the resulting added computational complexity. If more scalable constraints can be found for DWCM and RWCM, as we did for UWCM and UMCM, their performance could be more in line with the other models.



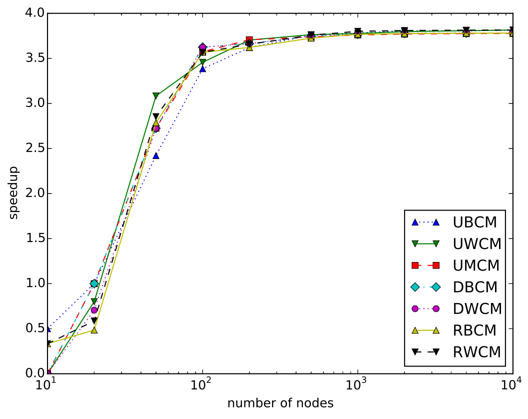
(a) Likelihood function



(b) Likelihood gradient function



(c) Likelihood Hessian function



(d) Error function

Figure 6.3: Speedup of the parallelized functions on 4 cores compared to the sequential version. Up to 10,000 nodes and for all models. Note that nodes refer to the nodes in the input graph, not the nodes in a computer cluster. Also note the logarithmic scale on the nodes axis.

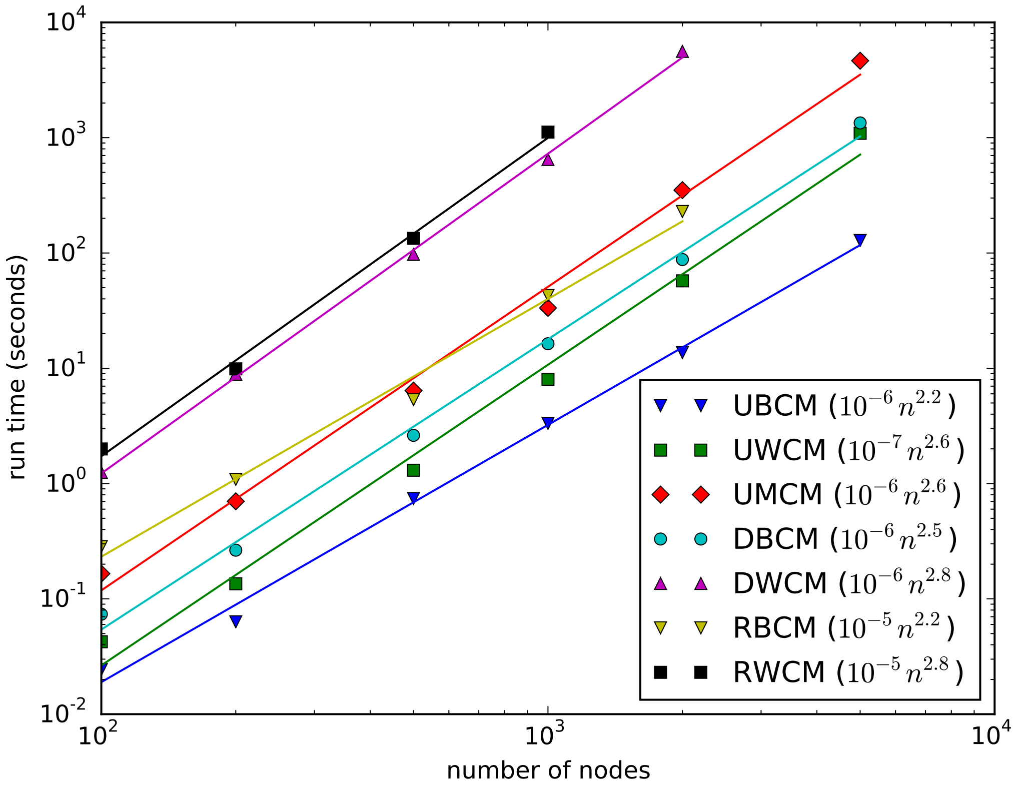


Figure 6.4: Required time (in seconds) for the improved software to find a solution. This is done using generated graphs, up to 5,000 nodes, for all models. The data points have been fit to power law functions ($t(n) = \alpha n^k$). Note the logarithmic scale on both axes.

| number of nodes | UBCM | UWCM | UMCM | DBCM | DWCM | RBCM | RWCM |
|---|---|---|---|---|---|---|---|
| 100 | $2.784\times10^{-10}$ | $3.837\times10^{-8}$ | $2.691\times10^{-8}$ | $9.770\times10^{-15}$ | $3.225\times10^{-8}$ | $3.580\times10^{-10}$ | $1.406\times10^{-7}$ |
| 200 | $5.447\times10^{-11}$ | $8.263\times10^{-8}$ | $1.592\times10^{-7}$ | $2.717\times10^{-8}$ | $1.075\times10^{-7}$ | $1.833\times10^{-13}$ | $5.039\times10^{-7}$ |
| 500 | $2.769\times10^{-8}$ | $4.544\times10^{-7}$ | $5.261\times10^{-7}$ | $3.311\times10^{-11}$ | $2.955\times10^{-11}$ | $6.268\times10^{-11}$ | $7.992\times10^{-11}$ |
| 1000 | $4.369\times10^{-11}$ | $5.548\times10^{-7}$ | $6.368\times10^{-7}$ | $2.178\times10^{-8}$ | $7.421\times10^{-7}$ | $8.163\times10^{-11}$ | $2.311\times10^{-10}$ |
| 2000 | $4.856\times10^{-11}$ | $7.352\times10^{-11}$ | $1.055\times10^{-10}$ | $2.114\times10^{-10}$ | $6.670\times10^{-11}$ | $1.151\times10^{-10}$ | - |
| 5000 | $2.123\times10^{-8}$ | $2.249\times10^{-10}$ | $5.238\times10^{-10}$ | $5.477\times10^{-8}$ | - | - | - |

Figure 6.5: Error of the solution found by the improved software. This is done using generated graphs, up to 5,000 nodes, for all models. “-” means no data because of memory limitations.

Figure 6.6 shows the peak memory usage of the improved software for generated graphs of varying sizes for all models. Some models do not have measurements for 5,000 nodes because of memory limitations. To get a quantitative measurement of the scaling, the data points have been fit to a power law function using a least squares method.

The effect of the dense Hessian can be seen in the near quadratic scaling. Furthermore, DWCM's and RWCM's non-linear constraints contribute to a slight worse scaling.

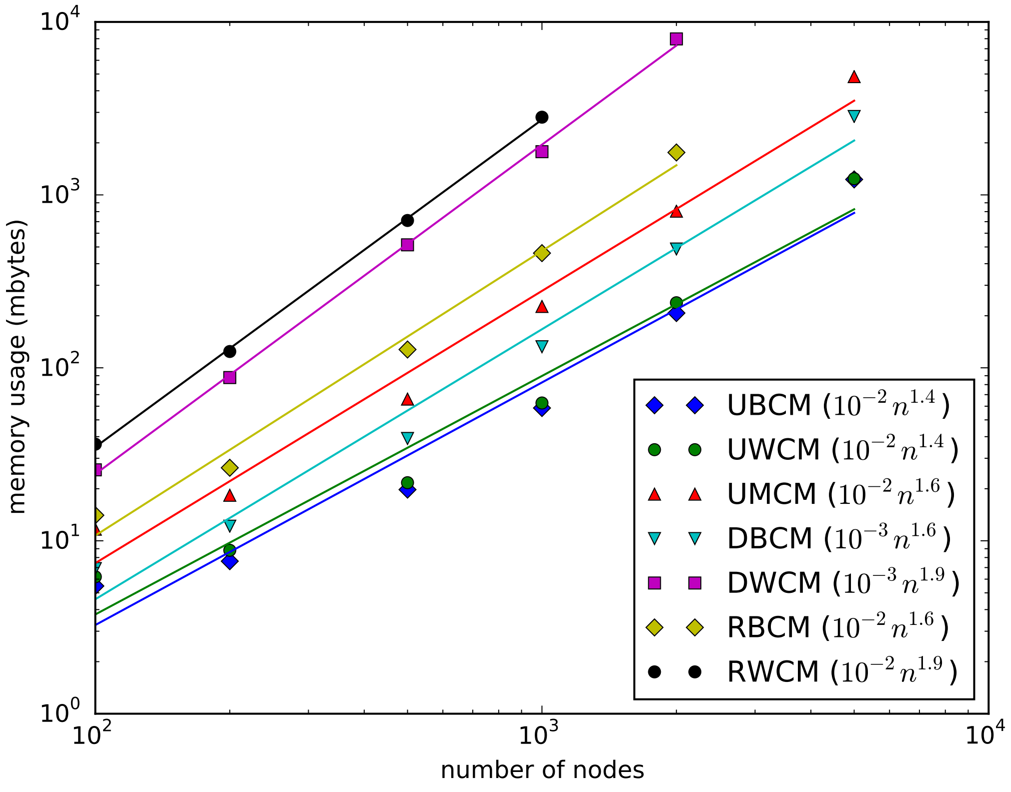


Figure 6.6: Peak memory usage (in mega bytes) of the improved software. This is done using generated graphs, up to 5,000 nodes, for all models. The data points have been fit to power law functions ($\text{mem}(n) = \alpha n^k$). Note the logarithmic scale on both axes.

# Chapter 7

# Conclusions and Future Work

For the analysis of graph ensembles, [3] proposed a fast and accurate “maximum-likelihood” method. An important step in this method is determining the hidden variables of a graph. We have reviewed a current implementation for determining these hidden variables and proposed an improved approach and implementation.

We have shown that the improved implementation performs significantly better than the current implementation. The run time of the improved method is in the range of hundreds of times faster than the current implementation. Also the improved implementation always finds good solutions, while this is rather random for the current implementation.

We have studied and implemented different models that are used for the analysis of different types of graphs and graph properties. Our implementation works for all these models, with only slight variations in the performance.

Furthermore, we have proposed several software engineering improvements such as object-oriented design and testing methods that improve the maintainability of the software implementation. Also, no longer requiring MATLAB and its toolboxes makes our software implementation more accessible to users.

To achieve all this, we improved problem definitions, applied mathematical analysis methods, improved solution algorithms and implemented this in a fast, accessible, maintainable and testable way. With faster and more accurate results, we believe that our improved implementation will help others in the analysis of graph ensembles, leading to the identification of relevant patterns in real world graphs.

The improved implementation currently does have some issues that prevent the analysis of larger graphs. Especially its near quadratically scaling memory usage is problematic and should be dealt with in future work.

One direction could be to define more scalable non-linear search space constraints for the DWCM and RWCM models, as we have done for the UWCM and UMCM models. This will make sure that the memory consumption of these models is more in line with the other models, allowing for the analysis of larger graphs.

Defining more scalable constraints does not however solve the issue with the dense Hessian. A distributed approach could be a solution, where our software is implemented in such a way that the computation and used memory is distributed over a network of computers (for example using MPI). This would allow for the analysis of much larger graphs, as long as one has enough computers available.

# Bibliography

[1] M. Newman, *Networks: An introduction*, Oxford University Press (2010)

[2] T. Squartini, I. van Lelyveld, D. Garlaschelli, *Early-warning signals of topological collapse in interbank networks*, Scientific Reports 3: 3357 (2013)

[3] T. Squartini, D. Garlaschelli, *Analytical maximum-likelihood method to detect patterns in real networks*, New Journal of Physics 13: 083001 (2011)

[4] T. Squartini, R. Mastrandrea, D. Garlaschelli, *Unbiased sampling of network ensembles with soft constraints*, arXiv:1406.1197 (2014)

[5] T. Squartini, F. Picciolo, F. Ruzzenenti, D. Garlaschelli, *Reciprocity of weighted networks*, Scientific Reports 3: 2729 (2013)

[6] R. Mastrandrea, T. Squartini, G. Gagiolo, D. Garlaschelli, *Enhanced reconstruction of weighted networks from strengths and degrees*, New Journal of Physics 16: 043022 (2014)

[7] R. Byrd, M. Hribar, J. Nocedal, *An interior point algorithm for large-scale nonlinear programming*, SIAM J. Optim. 9: 877–900 (1998)

[8] T. Coleman, Y. Li, *An interior, trust region approach for nonlinear minimization subject to bounds*, SIAM J. Optim. 6: 418–445 (1996)

[9] J. Nocedal, S. Wright, "Conjugate gradient methods" in *Numerical optimization*, 2nd ed. Springer (2006)

[10] MathWorks Inc., *fmincon interior point algorithm*, `http://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html#brnpd5f`, [visited on 5/12/2014].

[11] MathWorks Inc., *Trust-region reflective fsolve algorithm*, `http://www.mathworks.com/help/optim/ug/equation-solving-algorithms.html#brnpdsm`, [visited on 5/12/2014].

[12] A. Wächter, L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Mathematical Programming 106: 25–57 (2006)

[13] J. Hogg, A. Scott, *On the effects of scaling on the performance of Ipopt*, arXiv:1301.7283 (2013)

[14] International Trade Centre, *Trade statistics*, `http://www.intracen.org/`, [visited on 5/12/2014].

# Appendix A

# Hidden Variable Order

For the undirected weighted configuration model, we claim that if $\mathbf{s}$ satisfies:

$$s_1 \leq s_2 \leq \ldots \leq s_n, \tag{A.1}$$

then $\mathbf{x}$ at the maximum of $\mathcal{L}$ (equation (2.9)) satisfies:

$$x_1 \leq x_2 \leq \ldots \leq x_n, \tag{A.2}$$

where we assume that this maximum exists in the search space defined in Section 3.1. All first partial derivatives of $\mathcal{L}$ (equation (B.3)) must be equal to 0 at this maximum. We use this in equation (A.3) to prove the claim.

$$\begin{aligned}
\frac{s_i}{x_i} - \sum_{j\neq i} \frac{x_j}{1 - x_i x_j} &= 0 \qquad \Rightarrow \\
\sum_{j\neq i} \frac{x_j}{1 - x_i x_j} &= \frac{s_i}{x_i} \qquad \Rightarrow \\
\sum_{j\neq i} \frac{x_j x_i}{1 - x_i x_j} &= s_i \qquad \Rightarrow \\
\sum_{j\neq i} \frac{x_j x_i}{1 - x_i x_j} - \sum_{j\neq k} \frac{x_j x_k}{1 - x_k x_j} &= s_i - s_k \geq 0 \quad \text{if } k < i \qquad \Rightarrow \\
\sum_{j\neq i,k} \left( \frac{x_j x_i}{1 - x_i x_j} - \frac{x_j x_k}{1 - x_k x_j} \right) + \frac{x_k x_i}{1 - x_i x_k} - \frac{x_i x_k}{1 - x_k x_i} &\geq 0 \quad \text{if } k < i \qquad \Rightarrow \\
\sum_{j\neq i,k} \frac{x_j x_i (1 - x_k x_j) - x_j x_k (1 - x_i x_j)}{(1 - x_i x_j)(1 - x_k x_j)} &\geq 0 \quad \text{if } k < i \qquad \Rightarrow \\
\sum_{j\neq i,k} \frac{x_j (x_i - x_k)}{(1 - x_i x_j)(1 - x_k x_j)} &\geq 0 \quad \text{if } k < i \qquad \Rightarrow \\
(x_i - x_k) \sum_{j\neq i,k} \frac{x_j}{(1 - x_i x_j)(1 - x_k x_j)} &\geq 0 \quad \text{if } k < i \qquad \Rightarrow \\
x_i - x_k &\geq 0 \quad \text{if } k < i \qquad \Rightarrow \\
x_i &\geq x_k \quad \text{if } k < i
\end{aligned} \tag{A.3}$$

# Appendix B

# Likelihood Derivatives

All first and second partial derivatives of the likelihood functions and non-linear constraints of the configuration models are defined here in Sections B.1 to B.7.

## B.1 Undirected Binary Configuration Model

For UBCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.5)) are defined in equations (B.1) and (B.2) respectively.

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{k_i}{x_i} - \sum_{j \neq i} \frac{x_j}{1 + x_i x_j} \tag{B.1}$$

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{k_i}{x_i^2} + \sum_{j \neq i} \frac{x_j^2}{(1 + x_i x_j)^2} \\ \frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j} &= \frac{-1}{1 + x_i x_j} + \frac{x_i x_j}{(1 + x_i x_j)^2} \end{aligned} \tag{B.2}$$

## B.2 Undirected Weighted Configuration Model

For UWCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.9)) are defined in equations (B.3) and (B.4) respectively.

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{s_i}{x_i} - \sum_{j \neq i} \frac{x_j}{1 - x_i x_j} \tag{B.3}$$

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{s_i}{x_i^2} - \sum_{j \neq i} \frac{x_j^2}{(1 - x_i x_j)^2} \\ \frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j} &= \frac{-1}{1 - x_i x_j} - \frac{x_i x_j}{(1 - x_i x_j)^2} \end{aligned} \tag{B.4}$$

All non-zero first and second partial derivatives of the non-linear constraint function $g_\ell$ (equation (3.6)) are defined in equations (B.5) and (B.6) respectively.

$$\begin{aligned}\frac{\partial g_\ell}{\partial x_i} &= 1\\ \frac{\partial g_\ell}{\partial x_{i_{\max}}} &= \frac{1}{x_{i_{\max}}^2}\end{aligned} \tag{B.5}$$

$$\frac{\partial^2 g_\ell}{\partial x_{i_{\max}}^2} = \frac{-2}{x_{i_{\max}}^3} \tag{B.6}$$

## B.3 Undirected Mixed Configuration Model

For UMCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.12)) are defined in equations (B.7) and (B.8) respectively.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x_i} &= \frac{k_i}{x_i} - \sum_{j\neq i} \frac{x_j y_i y_j}{1 - y_i y_j + x_i x_j y_i y_j}\\ \frac{\partial \mathcal{L}}{\partial y_i} &= \frac{s_i}{y_i} - \sum_{j\neq i}\Big(\frac{y_j}{1-y_i y_j} - \frac{y_j - x_i x_j y_j}{1 - y_i y_j + x_i x_j y_i y_j}\Big)\end{aligned} \tag{B.7}$$

$$\begin{aligned}\frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{k_i}{x_i^2} + \sum_{j\neq i} \frac{x_j^2 y_i^2 y_j^2}{(1 - y_i y_j + x_i x_j y_i y_j)^2}\\ \frac{\partial^2 \mathcal{L}}{\partial y_i^2} &= -\frac{s_i}{y_i^2} - \sum_{j\neq i}\Big(\frac{y_j^2}{(1-y_i y_j)^2} + \frac{(-y_j + x_i x_j y_j)(y_j - x_i x_j y_j)}{(1 - y_i y_j + x_i x_j y_i y_j)^2}\Big)\\ \frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j} &= \frac{-y_j y_i}{1 - y_i y_j + x_i x_j y_i y_j} + \frac{x_i x_j y_i^2 y_j^2}{(1 - y_i y_j + x_i x_j y_i y_j)^2}\\ \frac{\partial^2 \mathcal{L}}{\partial y_i \partial y_j} &= \frac{1 - x_i x_j}{1 - y_i y_j + x_i x_j y_i y_j} - \frac{(-y_j + x_i x_j y_j)(y_i - x_i x_j y_i)}{(1 - y_i y_j + x_i x_j y_i y_j)^2} - \frac{1}{1 - y_i y_j} - \frac{y_i y_j}{(1-y_i y_j)^2}\\ \frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_i} &= \sum_{j\neq i}\Big(\frac{x_j y_i y_j(-y_j + x_i x_j y_j)}{(1 - y_i y_j + x_i x_j y_i y_j)^2} - \frac{x_j y_j}{1 - y_i y_j + x_i x_j y_i y_j}\Big)\\ \frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_j} &= \frac{-x_j y_i y_j (y_j - x_i x_j y_i)}{(1 - y_i y_j + x_i x_j y_i y_j)^2} - \frac{x_j y_i}{1 - y_i y_j + x_i x_j y_i y_j}\end{aligned} \tag{B.8}$$

All non-zero first and second partial derivatives of the non-linear constraint function $g_\ell$ (equation (3.8)) are defined in equations (B.9) and (B.10) respectively.

$$\begin{aligned}\frac{\partial g_\ell}{\partial y_i} &= 1\\ \frac{\partial g_\ell}{\partial y_{i_{\max}}} &= \frac{1}{y_{i_{\max}}^2}\end{aligned} \tag{B.9}$$

$$\frac{\partial^2 g_\ell}{\partial y_{i_{\max}}^2} = \frac{-2}{y_{i_{\max}}^3} \tag{B.10}$$

## B.4 Directed Binary Configuration Model

For DBCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.16)) are defined in equations (B.11) and (B.12) respectively.

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_i} &= \frac{k_i^{out}}{x_i} - \sum_{j \neq i} \frac{y_j}{1 + x_i y_j} \\
\frac{\partial \mathcal{L}}{\partial y_i} &= \frac{k_i^{in}}{y_i} - \sum_{j \neq i} \frac{x_j}{1 + x_j y_i}
\end{aligned} \tag{B.11}
$$

$$
\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{k_i^{out}}{x_i^2} + \sum_{j \neq i} \frac{y_j^2}{(1 + x_i y_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial y_i^2} &= -\frac{k_i^{in}}{y_i^2} + \sum_{j \neq i} \frac{x_j^2}{(1 + x_j y_i)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_j} &= \frac{-1}{1 + x_i y_j} + \frac{x_i y_j}{(1 + x_i y_j)^2}
\end{aligned} \tag{B.12}
$$

## B.5 Directed Weighted Configuration Model

For DWCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.20)) are defined in equations (B.13) and (B.14) respectively.

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_i} &= \frac{s_i^{out}}{x_i} - \sum_{j \neq i} \frac{y_j}{1 - x_i y_j} \\
\frac{\partial \mathcal{L}}{\partial y_i} &= \frac{s_i^{in}}{y_i} - \sum_{j \neq i} \frac{x_j}{1 - x_j y_i}
\end{aligned} \tag{B.13}
$$

$$
\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{s_i^{out}}{x_i^2} - \sum_{j \neq i} \frac{y_j^2}{(1 - x_i y_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial y_i^2} &= -\frac{s_i^{in}}{y_i^2} - \sum_{j \neq i} \frac{x_j^2}{(1 - x_j y_i)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_j} &= \frac{-1}{1 - x_i y_j} - \frac{x_i y_j}{(1 - x_i y_j)^2}
\end{aligned} \tag{B.14}
$$

All non-zero first and second partial derivatives of the non-linear constraint function $g_\ell$ (equation (3.10)) are defined in equations (B.15) and (B.16) respectively.

$$
\begin{aligned}
\frac{\partial g_\ell}{\partial x_i} &= y_j \\
\frac{\partial g_\ell}{\partial y_j} &= x_i
\end{aligned} \tag{B.15}
$$

$$\frac{\partial^2 g_\ell}{\partial x_i \partial y_j} = 1 \tag{B.16}$$

## B.6 Reciprocal Binary Configuration Model

For RBCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.24)) are defined in equations (B.17) and (B.18) respectively.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_i} &= \frac{k_i^{\rightarrow}}{x_i} - \sum_{j\neq i} \frac{y_j}{1 + x_i y_j + x_j y_i + z_i z_j} \\
\frac{\partial \mathcal{L}}{\partial y_i} &= \frac{k_i^{\leftarrow}}{y_i} - \sum_{j\neq i} \frac{x_j}{1 + x_i y_j + x_j y_i + z_i z_j} \\
\frac{\partial \mathcal{L}}{\partial z_i} &= \frac{k_i^{\leftrightarrow}}{z_i} - \sum_{j\neq i} \frac{z_j}{1 + x_i y_j + x_j y_i + z_i z_j}
\end{aligned} \tag{B.17}$$

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{k_i^{\rightarrow}}{x_i^2} + \sum_{j\neq i} \frac{y_j^2}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial y_i^2} &= -\frac{k_i^{\leftarrow}}{y_i^2} + \sum_{j\neq i} \frac{x_j^2}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial z_i^2} &= -\frac{k_i^{\leftrightarrow}}{z_i^2} + \sum_{j\neq i} \frac{z_j^2}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j} &= \frac{y_i y_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial y_i \partial y_j} &= \frac{x_i x_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial z_i \partial z_j} &= \frac{-1}{1 + x_i y_j + x_j y_i + z_i z_j} + \frac{z_i z_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_i} &= \sum_{j\neq i} \frac{x_j y_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial z_i} &= \sum_{j\neq i} \frac{y_j z_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial y_i \partial z_i} &= \sum_{j\neq i} \frac{x_j z_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_j} &= \frac{-1}{1 + x_i y_j + x_j y_i + z_i z_j} + \frac{x_i y_j}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial z_j} &= \frac{y_j z_i}{(1 + x_i y_j + x_j y_i + z_i z_j)^2} \\
\frac{\partial^2 \mathcal{L}}{\partial y_i \partial z_j} &= \frac{x_j z_i}{(1 + x_i y_j + x_j y_i + z_i z_j)^2}
\end{aligned} \tag{B.18}$$

## B.7 Reciprocal Weighted Configuration Model

For RWCM all non-zero first and second partial derivatives of $\mathcal{L}$ (equation (2.28)) are defined in equations (B.19) and (B.20) respectively.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_i} &= \frac{s_i^{\rightarrow}}{x_i} - \sum_{j\neq i}\Big(\frac{y_j}{1-x_i y_j} - \frac{x_j y_i y_j}{1-x_i x_j y_i y_j}\Big)\\
\frac{\partial \mathcal{L}}{\partial y_i} &= \frac{s_i^{\leftarrow}}{y_i} - \sum_{j\neq i}\Big(\frac{x_j}{1-x_j y_i} - \frac{x_i x_j y_j}{1-x_i x_j y_i y_j}\Big)\\
\frac{\partial \mathcal{L}}{\partial z_i} &= \frac{s_i^{\leftrightarrow}}{z_i} - \sum_{j\neq i}\frac{z_j}{1-z_i z_j}
\end{aligned} \tag{B.19}$$

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial x_i^2} &= -\frac{s_i^{\rightarrow}}{x_i^2} + \sum_{j\neq i}\Big(\frac{x_j^2 y_i^2 y_j^2}{(1-x_i x_j y_i y_j)^2} - \frac{y_j^2}{(1-x_i y_j)^2}\Big)\\
\frac{\partial^2 \mathcal{L}}{\partial y_i^2} &= -\frac{s_i^{\leftarrow}}{y_i^2} + \sum_{j\neq i}\Big(\frac{x_i^2 x_j^2 y_j^2}{(1-x_i x_j y_i y_j)^2} - \frac{x_j^2}{(1-x_j y_i)^2}\Big)\\
\frac{\partial^2 \mathcal{L}}{\partial z_i^2} &= -\frac{s_i^{\leftrightarrow}}{z_i^2} - \sum_{j\neq i}\frac{z_j^2}{(1-z_i z_j)^2}\\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j} &= \frac{y_i y_j}{1-x_i x_j y_i y_j} + \frac{x_i x_j y_i^2 y_j^2}{(1-x_i x_j y_i y_j)^2}\\
\frac{\partial^2 \mathcal{L}}{\partial y_i \partial y_j} &= \frac{x_i x_j}{1-x_i x_j y_i y_j} + \frac{x_i^2 x_j^2 y_i y_j}{(1-x_i x_j y_i y_j)^2}\\
\frac{\partial^2 \mathcal{L}}{\partial z_i \partial z_j} &= \frac{-1}{1-z_i z_j} - \frac{z_i z_j}{(1-z_i z_j)^2}\\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_i} &= \sum_{j\neq i}\Big(\frac{x_j y_j}{1-x_i x_j y_i y_j} + \frac{x_i x_j^2 y_i y_j^2}{(1-x_i x_j y_i y_j)^2}\Big)\\
\frac{\partial^2 \mathcal{L}}{\partial x_i \partial y_j} &= \frac{-1}{1-x_i y_j} + \frac{x_j y_i}{1-x_i x_j y_i y_j} - \frac{x_i y_j}{(1-x_i y_j)^2} + \frac{x_i x_j^2 y_i^2 y_j}{(1-x_i x_j y_i y_j)^2}
\end{aligned} \tag{B.20}$$

All non-zero first and second partial derivatives of the non-linear constraint function $g_\ell$ (equation (3.12)) are defined in equations (B.21) and (B.22) respectively.

$$\begin{aligned}
\frac{\partial g_\ell}{\partial x_i} &= y_j\\
\frac{\partial g_\ell}{\partial y_j} &= x_i\\
\frac{\partial g_\ell}{\partial z_i} &= z_j\\
\frac{\partial g_\ell}{\partial z_j} &= z_i
\end{aligned} \tag{B.21}$$

$$\begin{aligned} \frac{\partial^2 g_\ell}{\partial x_i \partial y_j} &= 1 \\ \frac{\partial^2 g_\ell}{\partial z_i \partial z_j} &= 1 \end{aligned} \tag{B.22}$$