Saarland University
Department of Computer Science

Bachelor Thesis

Learning Dynamical Processes to Infer the Underlying Network Structure

Author: Julian Zimmerlin
Supervisor: Univ.-Prof. Dr. Verena Wolf
Advisor: Gerrit Großmann

submitted on: April 14, 2021

Reviewers:
1. Univ.-Prof. Dr. Verena Wolf
2. Univ.-Prof. Dr. Jilles Vreeken

Statement

I hereby declare that this thesis is my own unaided work. All direct or indirect sources used are acknowledged as references.

Statement

I hereby confirm the congruence of the contents of the printed version and the electronic version of the thesis.

Declaration of Consent

I consent to both versions of my thesis being added to the library of the Computer Science Department, making them accessible to the public.

Saarbrücken, _____ _____

Abstract

Many natural phenomena, from the spread of viruses in social networks to the propagation of electrical signals in the brain, can be modeled as dynamical processes operating on networks of individual components. Knowledge of the underlying interaction structure is crucial to understanding the behavior of such systems. However, the network structure often cannot be directly observed and thus has to be inferred purely from observations of the individual components (nodes). In this thesis, we address this *network reconstruction* problem from a model-free perspective, that is, without assuming a particular dynamical model. To this end, we train a graph neural network on a given candidate network structure to predict the progression of the dynamical process. With the fundamental assumption that the network structure which enables the best prediction performance on the dataset of observations *is* the true network structure, network reconstruction can then be framed as a combinatorial optimization problem, where the prediction loss is to be minimized. We apply different local search search methods to this problem, which use custom-designed heuristics to efficiently search the neighborhood of a candidate network for promising next steps. Unlike traditional correlation-based methods, our combinatorial prediction-based methods can successfully reconstruct networks with highly non-linear dynamics. Furthermore, some of our methods outperform state-of-the-art prediction-based methods in terms of reconstruction accuracy on six datasets generated on small networks. We identify scalability as a key issue for our combinatorial prediction-based methods, which is due to the time complexity and the exploding size of the search space on larger networks. Overall, this work contributes to establishing the prediction-based machine learning approach to network reconstruction as a valid alternative to traditional statistical methods.

Contents

Chapter 1

Introduction

1.1 The network reconstruction problem

Many natural phenomena, from the spread of viruses in social networks to the propagation of electrical signals in the brain, can be modeled as dynamical processes operating on networks of individual components. Knowledge of the underlying network structure of such systems is crucial to understanding their behavior, enabling accurate predictions about the future and controlling the system through targeted interventions. However, the network structure is unobservable in many cases and has to be inferred purely from observations of the dynamical process operating on it.

In this thesis, we address the problem of inferring the hidden network structure from observations of a dynamical process operating on a network, as illustrated in Figure 1.1. Our goal is not simply to find correlations or similarities between nodes (*functional connectivity*), but to find the true network structure underlying the system (*structural connectivity*), filtering out unwanted network effects such as correlations caused by a common neighbor. We tackle this problem from a model-free perspective, i.e. without assuming knowledge of the laws that govern the dynamical process. Instead, we aim to find a general-purpose method that can be applied to many kinds of dynamical processes, ranging from stochastic binary processes (e.g. infections) over continuous ones (e.g. electrical signals) to vector-valued processes (e.g. objects moving in three-dimensional space).

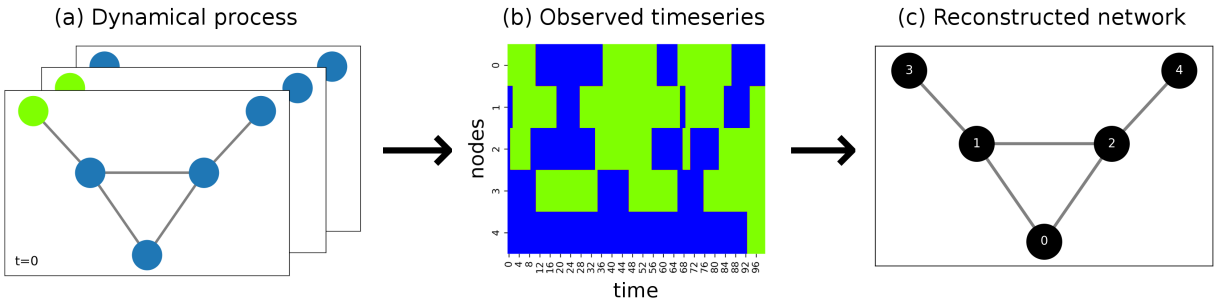

Figure 1.1: The basic problem setting, illustrated on a simple 5-node network with a binary dynamic. Our goal is to go from (b) to (c), reconstructing the original network as accurately as possible.

1.2 Applications

Applications of the network reconstruction problem (sometimes also referred to as *network inference*) arise anytime one wishes to infer the internal structure of an interacting system from external observations. One important application domain is neuroscience, where much research has gone into uncovering structural connections from observations of the electrical activity of neurons [1]. Here, the real-valued electrical signals, recorded for example with optical imaging methods, are used to infer the underlying synaptic network. In climate science, observations of rainfall events have been used to discover global atmospheric teleconnection patterns [2]. And in systems biology, gene regulatory networks, which describe the interactions of molecular regulators in a cell, are inferred from gene expression data [3].

In all these application domains (and more), knowledge of the network structure is highly valuable. It can be used to better understand the dynamical process [4, 5], to make better predictions about the behavior of the system [6], and in some cases to enable control of the system by elucidating how the network will be affected by targeted interventions or node failures [7]. It is therefore no surprise that the interest in and the amount of research on robust network inference methods for all kinds of settings is growing rapidly.

1.3 Methodical outline

The idea on which our network reconstruction approach is based is the following: We assume that the network structure which enables the *best predictions* about future states of the dynamic is the *true* network. In other words, we use prediction quality as a measure of structure quality. To be precise, we train a graph neural network that uses the candidate network structure to predict the dynamic at time step $t+1$ given the state of the dynamic at time step t. This idea was first used by Kipf et al. in [8], where the dynamics predictor is used as the decoder in a variational autoencoder architecture, and further developed by Zhang et al. [9]. In these works, the optimization of the network structure happens indirectly by iteratively altering a distribution over network structures, where the probability for the existence of an edge is modeled independently of the rest of the network. For further discussion on these methods, see Section 2.3.

In contrast, the methods presented in this work will directly optimize the network structure, without a distribution "behind the scenes". Since we restrict ourselves to unweighted, undirected graphs, we have a combinatorial optimization problem on our hands: The search space is the space of binary symmetric $N \times N$ matrices, where N stands for the number of nodes in the network, and the objective to be minimized is the prediction loss of a graph neural network trained on the network structure represented by the candidate matrix.

Our search methods will be based on local search, i.e. they will iteratively improve a candidate network structure by making local changes to it. The precise meaning of the word *local* in this context is not obvious, but very important, because it determines which steps the algorithm can take in a single iteration. Because of this, we will discuss and experiment with different definitions of local neighborhoods. A big difficulty for our problem is that the evaluation of the objective function for a candidate matrix involves training a neural network on a time series of observations, which means that these evaluations are computationally expensive and time-consuming. Consequently, straightforward local search methods which involve evaluating the objective function on all points in the

neighborhood are impractical here. Instead, we will rely on heuristics such as gradient information to guide the local search.

1.4 Structure of the thesis

The rest of this thesis is organized as follows:

- Chapter 2 contains the necessary background, including a short survey of existing approaches for network reconstruction, a brief introduction to graph neural networks and an overview of the dynamical models used in this work.
- In Chapter 3, we investigate the underlying assumption of prediction-based network reconstruction and try to find out how well suited prediction quality is as a proxy for structure quality. To do this, we analyze the *loss surface* of small problem instances under different dynamical models, dataset sizes and training budgets.
- In Chapter 4, we discuss how local search methods can be applied in the context of prediction-based network reconstruction. This includes the choice of step size, heuristics for searching the neighborhood for promising next steps and the search algorithm itself.
- In Chapter 5, our experimental results are presented and discussed. We evaluate the performance of the heuristics introduced in the previous chapter and compare the different local search methods to Zhang et al.'s method and correlation-based baselines.
- Chapter 6 contains discussion on the scalability of our methods as well as several ideas and opportunities for future work.
- Chapter 7 concludes the thesis with a short summary.

Chapter 2

Background

2.1 Notation

We represent a network (graph) $G = (V, E)$ as a tuple of a set of nodes $V = \{1, \ldots, N\}$ and a set of edges $E \subseteq V \times V$. The adjacency matrix $A \in \{0,1\}^{N \times N}$ of a graph is the binary matrix with $(i,j) \in E \iff A_{ij} = 1$. A graph is undirected if and only if $\forall (i,j) \in E : (j,i) \in E$, or equivalently $A = A^\top$. In an undirected graph, the set $\mathrm{N}(i) := \{\, j \in V \mid (i,j) \in E \,\}$ is called the neighborhood of a node i. The number of neighbors of a node, $\deg(i) := |\mathrm{N}(i)|$, is called the degree of the node.

The set $\mathscr{A}_N := \{\, A \in \{0,1\}^{N \times N} \mid A = A^\top \wedge \forall i \in V : A_{ii} = 0 \,\}$ denotes the set of all possible adjacency matrices for undirected N-node graphs without self-loops (also called simple graphs). As a distance measure on $\mathscr{A}_N$, we use the number of edge additions and deletions between the matrices: $\mathrm{dist}(A,B) := \left|\{\, (i,j) \in \{1,\ldots,N\}^2 \mid j > i \ \wedge\ A_{ij} \neq B_{ij} \,\}\right|$. We denote the *true* adjacency matrix underlying some given dataset as A^*.

We denote the time series of dynamical observations on a network as $X := (X^1, \ldots, X^T)$, where each individual observation $X^t := (X_1^t, \ldots, X_N^t)$ contains the dynamical states of all nodes at some time step t. We write $X_i := (X_i^1, \ldots, X_i^T)$ for the time series of a single node i. Note that a single dynamical observation X_i^t can be a binary or a continuous value, or even a vector itself, depending on the application domain.

Furthermore, in the case of stochastic dynamics, we write $\mathbf{X}_i^t$ for the random variable associated with node i at time step t. Analogously to the observed values, we define $\mathbf{X}^t := (\mathbf{X}_1^t, \ldots, \mathbf{X}_N^t)$ and $\mathbf{X}_i := (\mathbf{X}_i^1, \ldots, \mathbf{X}_i^T)$.

2.2 Existing approaches for NR

In general, approaches for network reconstruction can be grouped into two categories: *model-based* and *model-free* approaches [10].

Model-based methods

Model-based approaches aim to solve the network reconstruction problem for a particular *known* dynamical model, and thus are application-specific. An example in this category is the work done by Prasse and Van Mieghem [11, 12] for the SIS epidemics model explained in Section 2.7.2. The authors tackle the problem from a probabilistic perspective; specifically, they try to reconstruct the

unknown network structure given the time series of node states for some time span using *maximum-likelihood estimation*. The key theoretical result of their work is that this maximum-likelihood problem is NP-hard, which is proven by a polynomial-time reduction from the maximum cut problem. Interestingly, this result even holds when restricting the problem to any subclass of connected network structures, such as paths or star graphs. This finding deserves quite some attention, because it shows that network reconstruction is difficult *even when the dynamical model is known*, and of course it is even more difficult in the model-free setting that we consider in this work.

Model-based methods are also widely used to reconstruct biological neural networks. As explained in [10], the general procedure is to (1) assume some generative model for neural data, (2) determine model parameters to fit observed data and (3) use the information contained in the fitted model parameters to decide which edges are present in the network (e.g. by thresholding). Here, one must keep in mind that any hand-crafted generative model for neural data is a simplification of real neural dynamics. Thus, while the learned parameters explain the observations *in the assumed model*, there is no guarantee that the inferred connections are also present in the underlying biological network. Intuitively, one would expect that the correspondence between inferred connections and reality depends on how closely the generative model models the real dynamical process.

Model-free methods

Model-free approaches, on the other hand, do not assume a specific model for the dynamical process. This category includes methods from the field of descriptive statistics such as Granger causality [13], which measures whether a given time series is useful in forecasting another time series. It also includes information theoretic measures such as transfer entropy [14], which measures the reduction in uncertainty about the future of a variable when taking the history of another variable into account. The main advantages of such statistical or information-theoretic approaches are that they are conceptually and computationally relatively simple, and thus can be applied to networks with thousands of nodes or more. However, many of these measures are unable to distinguish between direct and indirect influences, which means they can only measure functional connectivity rather than structural connectivity. Partial correlation [15] can distinguish between indirect and direct interactions, but it only captures linear relationships and does not take the time ordering of observations into account, which is a significant loss of information. A good overview of these and other methods in the context of brain networks can be found in [10].

It should be noted that the term “model-free” does not mean that these methods make no assumptions at all about the kind of dynamics to which they can be applied. For example, basic correlation-based techniques measures will fail when the interactions between nodes are highly non-linear. Likewise, the methods presented in this work also make certain assumptions, even though they are meant to be as broadly applicable as possible. For further discussion, see Section 2.5 (final paragraph).

Recently, researchers have started to apply novel machine learning methods to network reconstruction problems. An early work in this category was published by Romaszko (2015) in the context of brain networks [16]. In his approach, the network reconstruction problem is divided edge by edge into binary classification problems. Roughly speaking, the author trains a convolutional neural network (CNN) which takes the time series of a pair of neurons as input and returns whether or not these two neurons are connected. Then, by iterating over all possible node pairs, the whole network

can be reconstructed. Note that this is a supervised approach, so it requires labeled training data, which can be hard to obtain in practice. Another problem is that of indirect interactions: When some node (1) is connected to some node (2), and node (2) is connected to a third node (3), the time series of nodes (1) and (3) might still be correlated even though they are not connected. An edge by edge approach thus runs the risk of inferring a false positive edge, since it does not consider that the similarity between the time series can already be explained by the indirect connection via node (2). This is the same problem that also plagues many statistical approaches, as mentioned in the beginning of this subsection.

As a consequence, we can see that a new network-centric approach is needed, which treats the network as a whole instead of splitting the problem edge by edge. This is done in what we call *prediction-based network reconstruction*, which is introduced in the next section.

2.3 Prediction-based network reconstruction

In prediction-based network reconstruction, the NR problem is not divided edge by edge. Rather, the network is treated as a whole and represented as an adjacency matrix $A \in \{0,1\}^{N \times N}$, where N is the number of nodes in the network and each entry a_{ij} indicates the presence (1) or absence (0) of an edge between the nodes i and j.

The term "prediction-based" refers to the underlying assumption that these methods use to assess a candidate network structure A: *The better the predictions about future node states that can be made using A, the closer A is to the true structure A^*.* This is, of course, an informal statement, mainly because we did not at all specify *how* predictions about future node states are made given a candidate network structure. This question will be addressed in the next section. But first, let us take a moment to reflect on this fundamental assumption in all its informality.

The nice thing about this assumption is that it provides an indirect way of assessing the quality of a candidate network without knowledge of the true network structure. With a concrete implementation of a dynamics predictor, it therefore enables us to tackle NR as an optimization problem, with prediction quality as the objective function. Furthermore, it is very general in the sense that it can be applied to all kinds of dynamical models, be they discrete or continuous. It also makes sense intuitively that the dynamics predictor should be able to make better predictions when it is given the correct network structure rather than a structure that is completely wrong. But it may not be so intuitive why the true network structure should perform better than a fully connected one: Should the dynamics learner not be able to predict the future states of a node just as well when it takes in information from the whole graph, irrespective of which nodes actually influence it, at least if it is trained long enough? This question, among others, will be addressed in Chapter 3. First, we need to get a better understanding of how the existing methods built on this assumption work.

2.4 Previous prediction-based methods

A foundational paper on prediction-based network reconstruction was published by Kipf et al. in 2018 [8]. Their approach is formulated as a variational autoencoder, which learns the network structure simultaneously with the dynamical process. Here, the encoder generates a network structure given the time series of dynamics observations, and the decoder predicts the future dynamical states given the network structure and the node states at some specific time step. Closely related is

the work by Zhang et al. (2019) [9], which replaces the encoder used by Kipf et al. with a simpler module.

The two-module framework used in [9], which they call *Gumbel graph network* (GGN), is illustrated in Figure 2.1. The task of the network generator is to generate candidate network structures, and the hope is that the generated structures get closer and closer to the ground truth structure during training. The task of the dynamics predictor is to evaluate how well the candidate structure fits the observed data. To answer this question, it tries to predict the node states $\mathbf{X}^{t+1}$ at the next time step given the node states $\mathbf{X}^t$ at time step t using the candidate network. Using some loss function, the predictions are compared to the actual observations at time $t+1$. The better the network structure, the lower this prediction loss (averaged over all samples in the dataset) should be.

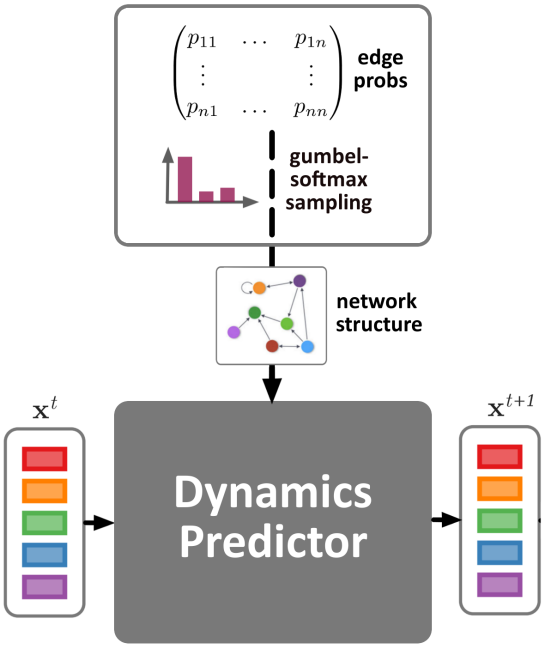

Figure 2.1: A schematic representation of the modular GGN architecture from [9].

Of course, at the very beginning of the training process, when the parameters of the dynamics predictor are still random, the prediction loss is not a good indicator of structure quality. During training, however, the dynamics predictor should gradually learn to emulate the dynamical model underlying the given dataset. When it has learned the dynamical model, the prediction loss should accurately reflect network quality.

The training of both modules happens by minimizing the prediction loss via backpropagation and gradient descent. We will not go into more detail on the training procedure here as it is not very relevant for our purposes; the interested reader is referred to [9]. We will now look at the internals of both modules in a bit more detail.

The network generator contains an edge probability matrix $P \in [0,1]^{N\times N}$[1]. In this matrix, each entry p_{ij} contains the probability that there is an edge between the nodes i and j. So, this matrix represents a distribution over network structures that assumes that all edge probabilities are independent of one another. To generate a network structure, one could in principle simply sample from this distribution. However, drawing a sample from a categorical distribution is not a differentiable operation, which would render the usage of gradient descent to update the probability matrix P impossible. To circumvent this problem, the authors employ a special sampling technique called *Gumbel-Softmax* sampling [17]. With this technique, the entries of the sampled matrix $\hat{A}$ are calculated as follows:

$$a_{ij} = \frac{\exp\big((\log(p_{ij}) + \xi_{ij})/\tau\big)}{\exp\big((\log(p_{ij}) + \xi_{ij})/\tau\big) + \exp\big((\log(1-p_{ij}) + \xi'_{ij})/\tau\big)} \tag{2.1}$$

[1] To be precise, what is actually stored in the implementation is the *logarithm* of these probabilities. If actual probabilities were used, one would need to ensure that the parameters remain bounded in [0,1], which is not straightforward to achieve when using gradient descent to update the parameters. But, since it is conceptually simpler, we will continue to say that the network generator contains probabilities and not log-probabilities.

with $\xi_{ij}, \xi'_{ij} \sim \text{Gumbel}(0,1)$ sampled i.i.d. from a Gumbel distribution and $\tau > 0$ specifying the softmax temperature. Note that the sampled adjacency matrix is not binary, as it contains entries in the interval $[0,1]$. Due to the special properties of the Gumbel distribution, however, we have that as $\tau \to 0$, the samples become identical to samples from Bernoulli(p_{ij}), meaning that the samples will take a value very close to one with probability p_{ij} and a value very close to zero otherwise. On the other hand, when $\tau \to \infty$, $a_{ij} \to 0.5$ (irrespective of p_{ij} !). For this reason, the parameter τ is initialized to a fairly large value (e.g. 10) and gradually decreased during the training process. For more details on Gumbel-Softmax sampling, the reader is referred to [17]. For our purposes, the most important thing to know is that this process enables optimization of the parameters of the network generator with gradient descent, because expression 2.1 is differentiable with respect to p_{ij}.

Now that we know how the network generator works internally, let us take a look at the dynamics predictor. The dynamics predictor is implemented as a graph neural network, which will be introduced in the next section.

2.5 Dynamics prediction with graph neural networks

In general, graph neural networks (GNNs) are neural networks that are especially designed for graph-structured input data. Motivated by the groundbreaking results that convolutional neural networks (CNNs) have achieved on grid-structured data such as images (e.g. [19, 20]), graph *convolutional* networks generalize the concept of local convolutions from the traditional two- or three-dimensional grid domain to arbitrary graph structures. Just like CNNs for grids, graph convolutional networks share filter parameters across different regions of the graph, which dramatically reduces the number of model parameters compared to a fully connected network and thus enables efficient parameter learning. Additionally, sharing parameters leads to a property called *permutation equivariance* [18], which (loosely speaking) refers to the fact that graph networks are able to respond to learned patterns in *any* region of the graph, not just in the region(s) where the pattern occurred in the training data.

Figure 2.2 shows the difference between convolutions on a regular grid and a graph. In contrast to the regular grid domain, the local neighborhood of a node in a graph is unordered and the neighborhoods of different nodes have different sizes. As a consequence, the convolution operation needs to be adapted to these circumstances.

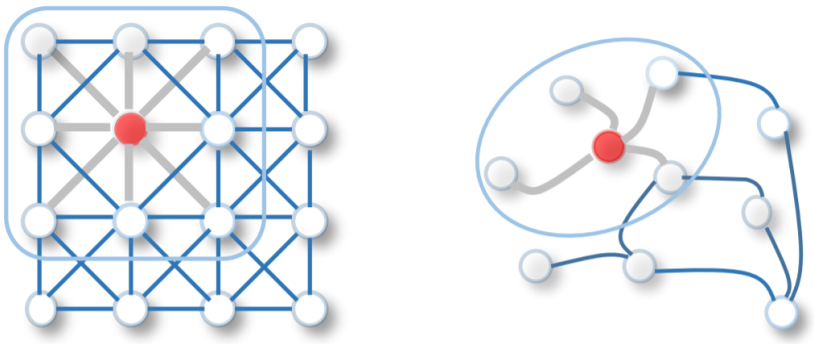

Figure 2.2: Left: Convolution on a 2D grid. Right: Generalization to arbitrary graph structures. *Image source: [18]. ©2020 IEEE.*

Model architecture

We focus on the particular variant used in [8] and [9], which we call *message passing graph convolutional network* (MPGCN). In this model, the graph convolution is implemented by passing messages between neighboring nodes and aggregating them for each node by summation.

In the following equations, $m_{ij}^t \in \mathbb{R}^H$ is the message from node i to node j, and $h_i^t \in \mathbb{R}^H$ is the hidden representation of node i, where the size H can be chosen arbitrarily (we choose $H = 128$ in our implementation). The convolution operation is implemented as follows:

$$m_{ij}^t = f_{edge}([X_i^t, X_j^t]) \qquad \forall (i,j) \in E \tag{2.2}$$

$$h_i^t = f_{node}\Big(\sum_{j \in \mathrm{N(i)}} m_{ij}^t\Big) \qquad \forall i \in V \tag{2.3}$$

where $[\cdot,\cdot]$ represents vector concatenation. The functions f_{edge} and f_{node} are implemented as small multi-layer perceptrons (MLPs) with two linear layers and ReLU activation functions. Note that h_i^t contains information about the states node i and its direct neighbors at time step t, and nothing more than that. From the hidden representation and the original node state, the predictions $o_i^{t+1} \in \mathbb{R}^I$ (I matches the dimensions of the dynamic) for the next step are calculated as follows for continuous dynamics:

$$o_i^{t+1} = f_{output}([X_i^t, h_i^t]) \qquad \forall i \in V \tag{2.4}$$

where f_{output} is either a small MLP or just a linear layer. Optionally, one can also introduce *skip connections* [20]:

$$o_i^{t+1} = X_i^t + f_{output}([X_i^t, h_i^t]) \qquad \forall i \in V \tag{2.5}$$

For discrete dynamics with K states, it is necessary to convert the K-dimensional output of the MPGCN to a probability distribution over the possible node states, so a softmax function is added at the end:

$$o_i^{t+1} = \mathrm{softmax}\big(f_{output}([X_i^t, h_i^t])\big) \qquad \forall i \in V \tag{2.6}$$

with $\mathrm{softmax}(z)_k = \frac{\exp(z_k)}{\sum_{l=1}^K \exp(z_l)} \quad \forall 1 \le k \le K$.

Discussion

Note that all of the model parameters are hidden in the MLPs f_{node}, f_{edge}, f_{output}. The same MLPs are used for all nodes in the network, which ensures that the learned dynamic is the same everywhere — it is not possible for different nodes to learn different dynamics. Also note that in contrast to many applications of graph convolutional networks (and traditional image CNNs), only a single convolutional operation is performed. Theoretically, it would also be possible to stack multiple convolutional layers by iteratively applying equations 2.2 and 2.3 to the hidden representation, but, intentionally, this is not done. Using just a single convolution means that only the direct neighbors of a node can influence its next state. This means that our reconstruction method will need to put an edge between any nodes that directly influence each other — which is exactly what we want.

Due to the expressive power of neural networks [21], we have reason to believe that a wide variety of non-linear dynamics can be learned with this architecture, as long as they satisfy the

following assumptions:

1. *Markov property in time*:

$$\Pr\left(\mathbf{X}^{t+1} \,\middle|\, \mathbf{X}^t, \ldots, \mathbf{X}^1\right) = \Pr\left(\mathbf{X}^{t+1} \,\middle|\, \mathbf{X}^t\right)$$

   if both sides are well-defined, i.e. $\Pr\left(\mathbf{X}^t, \ldots, \mathbf{X}^1\right) > 0$. The dynamical state at time step $t+1$, $\mathbf{X}^{t+1}$ is independent of the previous time steps $\mathbf{X}^1, \ldots, \mathbf{X}^{t-1}$ given the dynamical states at time step t, $\mathbf{X}^t$. In other words: $\mathbf{X}^t$ contains all the information we need to predict $\mathbf{X}^{t+1}$.

2. *Markov property on the network*:

$$\Pr\left(\mathbf{X}_i \,\middle|\, \{\, \mathbf{X}_j \mid j \in V, j \neq i \,\}\right) = \Pr\left(\mathbf{X}_i \,\middle|\, \{\, \mathbf{X}_j \mid j \in \mathrm{N}(i) \,\}\right)$$

   if both sides are well-defined, i.e. $\Pr\big(\{\mathbf{X}_j \mid j \in V, j \neq i\}\big) > 0$. The dynamic of some node i, $\mathbf{X}_i$ is independent of all other nodes in the network given the dynamics time series of its neighbors $\mathbf{X}_j$, $j \in \mathrm{N}(i)$. In other words: The neighbors of a node (together with the node itself) contain all the information we need to predict the dynamics for the node.

These two assumptions are "hard-coded" in the architecture of the dynamics learner explained above. Assumption 1 is needed because we only use information from time step t to predict time step $t+1$. Assumption 2 is needed because only one convolutional operation is performed, which means that only information from the node's neighbors and the node itself is used for prediction. It *is*, however, possible to relax these assumptions by changing the architecture: In [8], the authors add gated recurrent units (GRUs) to the MPGCN to deal with dynamics that do not satisfy the Markovian assumption in time. In this work, we will restrict ourselves to dynamical models that fulfill both assumptions.

2.6 Critique

Now that we understand Zhang et al.'s approach, let us give a brief critique. As mentioned before, the network reconstruction problem for unweighted graphs is fundamentally a combinatorial problem. With the Gumbel graph network architecture, Zhang et al. solve this problem indirectly by optimizing a distribution over network structures, represented by an edge probability matrix P, where all edge probabilities are modeled *independently of the rest of the network*. However, as we have argued at the end of Section 2.2, the existence of an edge actually heavily depends on which other edges are present in the network. Therefore, this independent modeling might be suboptimal. Unfortunately, it does not seem feasible to include these dependencies in a probabilistic model over graph structures because the number of required parameters would explode (when conditioning the existence of an edge a_{ij} on k other edges, 2^k numbers are needed to fully specify the conditional probability distribution of this single edge a_{ij}).

In this work, we will take a different route and try to tackle network reconstruction directly as a combinatorial optimization problem, without modeling a distribution over graph structures. Thus, in Zhang et al.'s terminology, the network generator is replaced by some combinatorial optimization algorithm. We focus on local search algorithms, the details of which will be discussed in Chapter 4. To evaluate a candidate structure, we train the dynamics predictor *from scratch* on a given timeseries of observations and then record the average loss over all samples in the dataset (see Section 3.1.1 for more details). Note that this is in contrast to Zhang et al.'s approach, since they train the dynamics

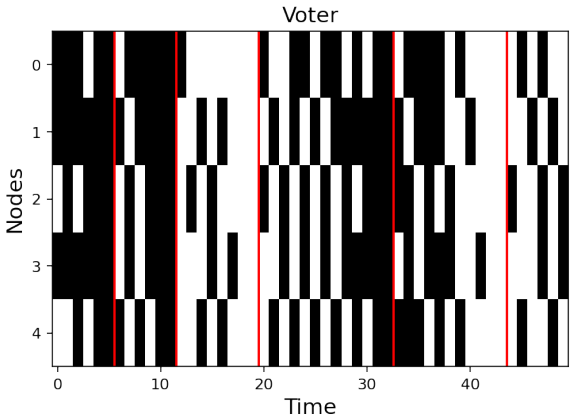

Figure 2.3: Example voter time series for a 5-node graph. The red lines indicate reinitializations of the dynamical state due to consensus.

predictor continuously without ever reinitializing its weights.

2.7 Dynamical models

In this section, the dynamical models that we use for our experiments are introduced.

2.7.1 Voter

The voter model is a well-established model of the behavior of interacting systems and has been studied extensively on regular lattices [22] as well as on heterogeneous graphs [23, 24]. One interpretation of the voter model is that it models the spread of a (binary) opinion in a social network. The dynamical rules are quite simple: In each time step, each node chooses one of its neighbors uniformly at random and copies its state (opinion). Representing the binary states as 0 and 1, the state probabilities for a node i at time $t+1$ can be expressed as follows:

$$\Pr\left(\mathbf{X}_i^{t+1}=1 \,\middle|\, \mathbf{X}^t=X^t\right) = \frac{1}{\deg(i)} \sum_{j\in \mathrm{N}(i)} X_j^t$$
$$\Pr\left(\mathbf{X}_i^{t+1}=0 \,\middle|\, \mathbf{X}^t=X^t\right) = \frac{1}{\deg(i)} \sum_{j\in \mathrm{N}(i)} (1-X_j^t)$$

Note that in this model, the next state of the node is independent of its own previous state (unless it has a self-loop). We say that the dynamic has reached *consensus* at time step t when $X^t = 1$ or $X^t = 0$. It is obvious that in this case, the same property also holds for any $t' > t$. As shown in [24] (Theorem 2.2), the voter model converges to consensus *almost surely* for any finite connected graph, irrespective of the initial conditions.[2] This can be a problem for network reconstruction if it reaches consensus too quickly, because in this case no further interactions between nodes can be observed. To make network reconstruction possible in these cases, one has to observe multiple dynamics time series instead of just a single one in order to capture more interactions. This is also what we do in our experiments.

[2] To be precise, the authors of [24] use a slightly different model with *asynchronous updates*, so the theorem does not directly apply to our model. The basic problem of consensus remains.

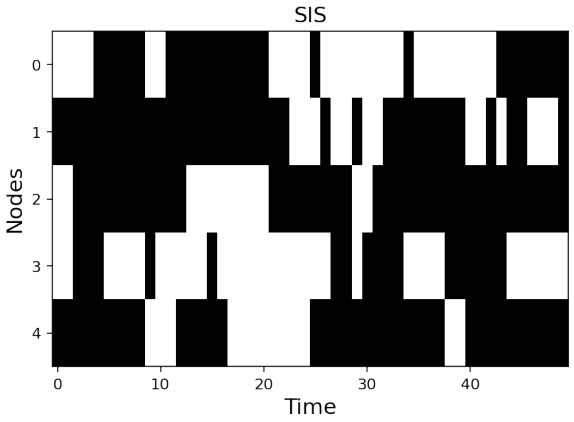

Figure 2.4: Example SIS time series for a 5-node graph ($\beta = \mu = 0.2$).

2.7.2 Susceptible-Infected-Susceptible

The SIS model models the spread of a contagion in a network (e.g. a virus in a social network, or malware in a computer network). The abbreviation stands for Susceptible-Infected-Susceptible, which refers to the two possible states of the nodes, which are represented as 0 (susceptible) and 1 (infected). In this model, infected nodes can infect their susceptible neighbors with a certain probability β in each time step. Also in each time step, infected nodes can recover and become susceptible again with probability μ. The transition probabilities are thus given by the following equations:

$$\begin{aligned}
\Pr\left(\mathbf{X}_i^{t+1} = 0 \,\middle|\, \mathbf{X}_i^t = 1, \mathbf{X}^t = X^t\right) &= \mu \\
\Pr\left(\mathbf{X}_i^{t+1} = 1 \,\middle|\, \mathbf{X}_i^t = 1, \mathbf{X}^t = X^t\right) &= 1-\mu \\
\Pr\left(\mathbf{X}_i^{t+1} = 0 \,\middle|\, \mathbf{X}_i^t = 0, \mathbf{X}^t = X^t\right) &= (1-\beta)^{\mathrm{N}_{\mathrm{inf}}^{\mathrm{t}}(i)} \\
\Pr\left(\mathbf{X}_i^{t+1} = 1 \,\middle|\, \mathbf{X}_i^t = 0, \mathbf{X}^t = X^t\right) &= 1-(1-\beta)^{\mathrm{N}_{\mathrm{inf}}^{\mathrm{t}}(i)}
\end{aligned}$$

where $\mathrm{N}_{\mathrm{inf}}^{\mathrm{t}}(i) = \sum_{j \in \mathrm{N}(i)} X_j^t$ is the number of infected neighbors of node i at time t.

Similarly to consensus in the voter model, in the SIS model it is possible that the infection *dies out*, i.e. $X^t = 0 \ \forall t \geq t'$ for some t'. So, when simulating SIS dynamics we take care to choose a sufficiently high infection rate and a sufficiently low recovery rate to make sure this does not happen. For a theoretical analysis of the critical infection rate that is needed to keep the infection "alive" on random graphs, see [25]. When working with real-world observations of a SIS dynamical process with an infection rate below the critical threshold, one has to resort to observing multiple time series if a single observation does not capture enough interactions to reconstruct the network.

2.7.3 Coupled map

Coupled maps are interacting systems of discrete elements with discrete-time, continuous-state dynamics and are widely used to study the chaotic properties of physical systems such as boiling liquids [26]. The most well-known variant of a coupled map is the coupled map lattice (CML) [27], where the discrete interacting elements constitute a regular lattice (grid). In our case, the interacting elements are the nodes of an irregular graph.

In contrast to the voter and SIS models, the state transitions are determined by a deterministic

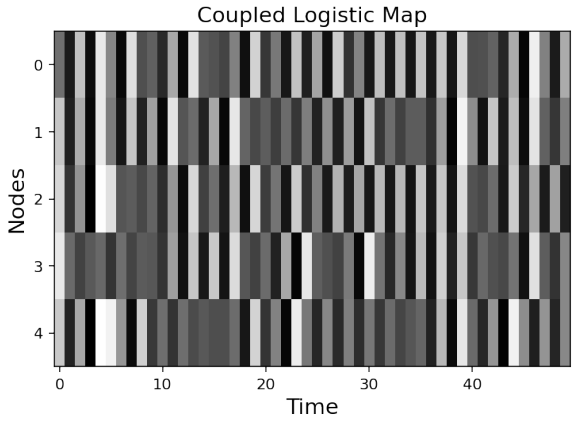

Figure 2.5: Example coupled logistic map time series for a 5-node graph ($r = 4$, $s = 0.2$).

function instead of a stochastic process:

$$X_i^{t+1} = (1-s)f(X_i^t) + \frac{s}{\deg(i)} \sum_{j \in \mathrm{N}(i)} f(X_j^t)$$

where $s \in [0,1]$ determines the coupling strength (we use weak coupling with $s = 0.2$ in our experiments) and f is a "local map". Like Zhang et al. in [9], we choose a logistic map [28]:

$$f(x) = rx(1-x)$$

where $r \in [0,4]$ is a crucial parameter that determines whether the complexity of the system's behavior: Below the critical threshold of $r \approx 3.57$ [29], the logistic sequence either converges to a fixed value or oscillates between a small set of values, whereas it erupts into chaotic behavior for larger values of r (with certain exceptions), as illustrated in Figure 2.6[3]. For non-chaotic values of r, we will again need multiple observations from different initial conditions, since no more interactions between nodes can be observed once the network has reached a steady state or regular oscillations.

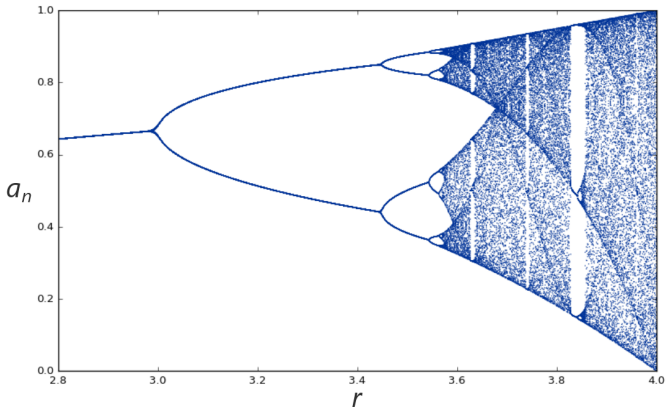

Figure 2.6: Bifurcation diagram of the logistic map. For r-values between 2.8 and 4, it shows 100 elements of the logistic sequence $\big(a_{n+1} = ra_n(1-a_n),\ a_0 = 0.5\big)$ in a vertical line (to be precise, the values of the sequence for $n = 100, \ldots, 199$ are shown). *Image source: [30]. The original caption has been removed and axis labels have been changed. Usage permitted under the CC BY 4.0 license [31].*

[3] Note that the figure shows values of a single logistic map, in other words a one-node network.

Chapter 3

The Loss Landscape

In this chapter, we investigate the underlying assumption of prediction-based network reconstruction introduced in Section 2.3. How good of an indicator is the prediction loss of a GNN trained on a candidate network structure in assessing the (dis)similarity of that structure to the true network structure? Is it true that the ground truth network structure produces the lowest prediction loss, or is it inferior to network structures which have more links, such as fully connected networks? And how smooth is the landscape of prediction losses on the search space of network structures — is it more or less convex, or is it riddled with local minima? The answers to these questions depend on various factors, of which we are going to investigate three: (1) The type of dynamics (i.e. the dynamical model), (2) the size of the dataset of dynamics observations and (3) the effect of varying "training budgets" (i.e. the number of epochs that each matrix is trained for before evaluation).

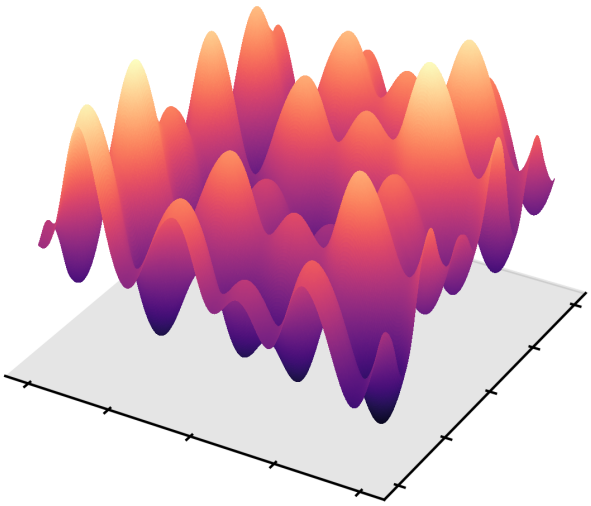

Figure 3.1: A difficult optimization landscape riddled with local optima on a continuous two-dimensional search space. In our problem setting, the search space is discrete and high-dimensional.

In short, our experimental approach (described in detail in the next section) can be summarized as follows: Depending on the factor we want to study, we generate a number of dynamics time series which only vary in this factor (e.g. dataset size). Then, for each of the time series, we evaluate the prediction quality for every possible network structure. This allows us to study the effect of the factor by comparing the resulting *optimization landscapes*.

What is an optimization landscape?

The notion of an optimization landscape (also called "fitness landscape", "loss surface" or similar terms) is a way of thinking about optimization problems which originates from evolutionary biology [32]. Simply put, one can imagine the optimization surface as a plot where the points on the horizontal plane represent elements of the search space (in our case, adjacency matrices) and the vertical axis represents the value of the objective function (the fitness or the loss) of each of those elements (see Figure 3.1 for an illustration). The optimization problem then translates to finding the deepest valley (in the case of minimization problems) or the highest peak (in the case of maximization problems) on this landscape. The shape of the loss surface largely dictates the difficulty of the optimization problem — if it is convex, for example, simple gradient-based techniques can be used to find the global optimum, whereas optimization algorithms often tend to get stuck in suboptimal regions of the search space if the surface is riddled with local minima. Knowledge of the properties of the optimization landscape is therefore valuable because it facilitates the assessment of the difficulty of the problem at hand and consequently enables us to choose an optimization method that is adequate for the complexity of the problem. Of course, for high-dimensional search spaces it is not possible to visually plot the full optimization landscape. Luckily, in our case we can still visualize it by projecting the search space into a single dimension, as we will see soon.

3.1 Experimental setup

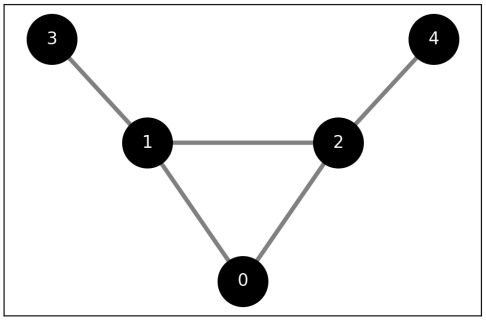

Figure 3.2: The "bull graph" used in our brute force experiments.

We perform all of our brute force experiments on the simple 5-node undirected network commonly called the "bull graph" shown in Figure 3.2. Thus, the ground truth adjacency matrix for all experiments in this chapter is given by

$$A^* = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

In total, there are $2^{5^2} = 33554432$ possible 5-node network structures, since the adjacency matrix has 5^2 binary entries. By restricting ourselves to $\mathscr{A}_5$, the set of undirected 5-node graphs with no self-loops (i.e. setting the diagonal entries to zero), we can reduce the size of the search space to $2^{(5-1)*5/2} = 2^{10} = 1024$, which makes brute force evaluation feasible. Ignoring self-loops is justified by the fact that the dynamics learner uses the node's own state for prediction *whether there is a*

self-connection or not (see Section 2.5).

In every one of our experiments, a number of dynamics time series will be simulated on the bull graph. Depending on the subject of the experiment, these time series will differ in some way from each other — for example, they will differ in the length when studying the impact of dataset size on the loss landscape. For each of these time series, we will then calculate the complete loss landscape in a brute force manner, which means we evaluate the average prediction loss that can be achieved with every possible network structure $A \in \mathscr{A}_5$ (the details of this will be described in the next subsection). Then, we analyze the resulting loss landscapes by visualizing one-dimensional projections of them and calculating the number of local minima (the details will be described in Section 3.1.2).

3.1.1 Evaluating a candidate structure

To evaluate the quality of a network structure A on a given dynamical time series X generated on the true network structure A^*, we must first train the dynamics learner. First, all of the parameters of the dynamics learner are initialized randomly. Then, the dynamics learner is trained for some fixed[1] amount of epochs using X as training data and A as the internal network structure for message passing. For the parameter updates, the well-known Adam algorithm [33] is used. We refer to the set of trained GNN parameters as Φ and to the dynamics predictor itself as $f_{A,\Phi}$.

To find out how well the dynamics learner has learned the dynamics, we then perform an "evaluation epoch", i.e. we iterate through the whole dataset once again and calculate the average prediction loss across all time steps. In learning theory, this quantity is typically called the *empirical risk* and is denoted with the letter R. For our application, we can formulate it mathematically as

$$R_X(A,\Phi) = \frac{1}{(T-1)N} \sum_{t=1}^{T-1} \mathbf{1}_N^\top L\big(f_{A,\Phi}(X^t), X^{t+1}\big), \tag{3.1}$$

where $\mathbf{1}_N^\top$ is a N-dimensional vector containing only ones, and L is a loss function that is applied element-wise for each node (i.e. we get an N-dimensional vector that contains the loss for each node in the network). In the training phase as well as in the evaluation epoch, we use cross entropy loss

$$L_{CE}\big(f_{A,\Phi}(X^t), X^{t+1}\big) = -\Big(X^{t+1} \log\big(f_{A,\Phi}^{(1)}(X^t)\big) + (1-X^{t+1}) \log\big(f_{A,\Phi}^{(0)}(X^t)\big)\Big)$$

for binary dynamics, where $f_{A,\Phi}^{(k)}(X^t)$ represents the N-dimensional post-softmax output of the dynamics predictor for class k (see Section 2.5), and for continuous dynamics we use the L_1 loss,

$$L_1\big(f_{A,\Phi}(X^t), X^{t+1}\big) = \big|f_{A,\Phi}(X^t) - X^{t+1}\big|.$$

For completeness, it should be said that we use multi-step predictions for coupled map dynamics, as was done in [9]. This means that from each sample X^t, one not only predicts the dynamics one step into the future; rather, one predicts P steps by iteratively applying the dynamics leaner with the

[1] An alternative to using a fixed number of iterations would be to automatically detect when the training process has converged. We experimented in this direction by terminating the training phase as soon as the loss changes by less than some threshold ϵ in 10 successive epochs. This works in principle, but it is hard to choose ϵ appropriately, since the loss can have different scales in different settings. There might be ways to work around this issue, but in our rudimentary experiments, automatic convergence detection did not seem to improve the accuracy of our methods (although it can save some run time), so we chose not to pursue this idea any further in this work.

previous output as input: For each p with $1 \leq p \leq P$, the prediction of the dynamics learner is given by $\hat{X}^{t+p} = f^p_{A,\Phi}(X^t)$. Since this technique is used both in the training and the evaluation phase, we get the following variation of Equation (3.1):

$$R_X(A,\Phi) = \frac{1}{(T-1)NP} \sum_{t=1}^{T-P} \sum_{p=1}^{P} \mathbf{1}_N^\top L\big(f^p_{A,\Phi}(X^t), X^{t+p}\big). \tag{3.2}$$

In practice, we use $P = 9$ prediction steps for coupled map dynamics, as was done in [9]. We do not use multi-step predictions for voter and SIS dynamics.

3.1.2 Analyzing the loss landscape

After evaluating every matrix $A \in \mathscr{A}_N$, we want to analyze the resulting loss landscape. To visualize the loss landscape, the search space is projected into a single dimension by grouping the matrices by their distance to the true adjacency matrix A. This procedure gives us eleven groups of matrices: Group 0 contains only the ground truth adjacency matrix A^*. Group 1 contains the 10 one-hop neighbors, i.e. all structures which differ from the ground truth by exactly one edge. Group 2 contains the 45 two-hop neighbors, and so on. Group 10 contains just one matrix again, namely the matrix A with $A_{ij} = 1 - A^*_{ij}\ \forall i \neq j$. In general, group k contains $\binom{10}{k}$ matrices. For each of these groups, we plot the mean prediction loss as well as the minimal and maximal prediction loss (measured in the evaluation epoch) in the group.

Additionally, we calculate the number of local minima in the search space. Here, we consider two matrices A, $B \in \mathscr{A}_5$ neighbors if and only if $\mathrm{dist}(A,B) = 1$, which means that they differ by exactly one edge. Thus, any network structure that lies in a local minimum cannot be improved (in terms of prediction loss) by simply adding or removing an edge. The number of local minima in the surface is calculated before grouping and serves as another indicator for the smoothness of the loss surface.

It is important to notice that our objective function contains stochastic noise — the prediction loss in the evaluation epoch depends on the initial weights of the graph neural network, which are randomly sampled from a probability distribution with mean 0. Thus, the loss surface can actually change in repeated runs of the same experiment, including the number of local minima. This is the reason why all numbers given in this section can only serve as *estimates* of the quantities they intend to specify. Later, when we actually optimize the prediction loss, we *could* of course fix the random seed used for initialization of network parameters to make the evaluation function deterministic, but it is not entirely clear whether this is really desirable; it could happen that we fix the initial weights to values that are simply suboptimal for certain candidates, which could be discovered in repeated evaluations with different initial weighs.

3.2 Experiments

3.2.1 Effect of dynamical models

First, we compare the loss surfaces generated by the three different dynamical models introduced in Section 2.7 (voter, SIS, coupled map). We generate datasets of 5000 time steps on the bull graph for each model. As hyperparameters we choose $\beta = \mu = 0.2$ for the SIS model and $r = 4$, $s = 0.2$ for the coupled map to obtain weakly coupled chaotic dynamics. On each candidate network structure, we train the dynamics predictor for 40 epochs using a batch size of 100. The results are shown in

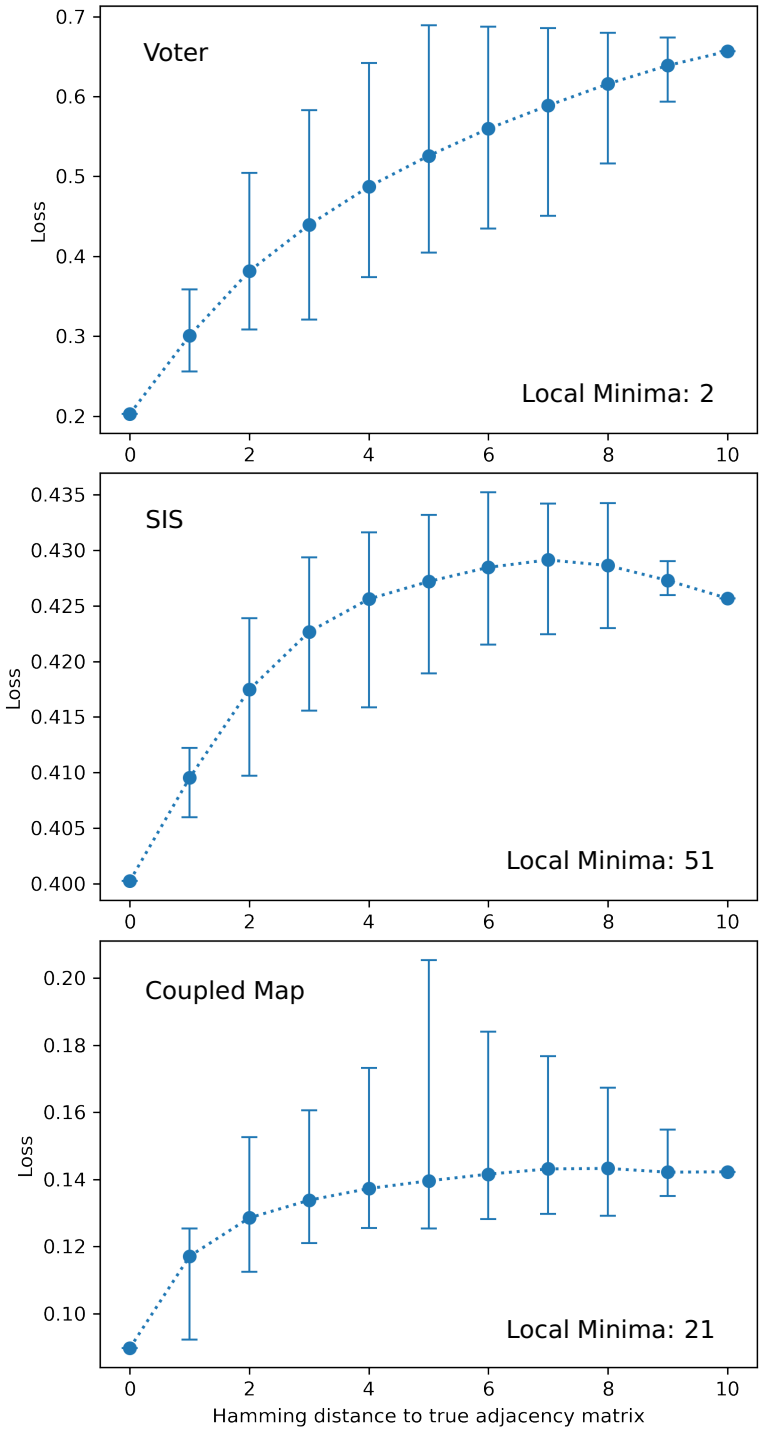

Figure 3.3: Loss surfaces generated by the three dynamical models (Voter, SIS, Coupled Map) after grouping by distance to the true matrix. Blue dots represent the mean loss in the group. Error bars show the highest and lowest loss in the group.

Figure 3.3.

First of all, we can see that the global minimum of the prediction loss lies at the ground truth network for all three dynamical models. This is good, because it means that minimizing the prediction loss will in principle guide us to the real network structure, assuming that we do not get stuck in local minima on the way. While a fully connected network (belongs to group 5) also has access to all the necessary information for its predictions, it always performs worse than the true network. This is due to the fact that the dynamics predictor is incapable of learning which neighbors are relevant or irrelevant for each individual node, it is forced by its architecture to treat all neighbors equally. So, adding more edges to the true network structure is indeed harmful for prediction performance.

Secondly, we can see that there are quite some differences in the shape of the loss surface between the different dynamical models, even in the one-dimensional projection which does not reflect the full complexity of the search space.

Regarding the voter model, we can see that there is a clear monotonically increasing relationship between the distance to the true network and the prediction performance in the minimum, maximum (up to distance 5) as well as the mean prediction loss: The larger the distance between a candidate and the true network, the higher its expected loss (in general). The low number of local minima (2) is another indicator that this surface is well-shaped.

Moving on to the SIS model, we observe that the monotonically increasing relationship between the mean loss and the distance to the ground truth only holds up to group 7, the mean loss drops off slightly when approaching group 10. What is more concerning, though, is the increase in the number of local minima in the search space. This number (51) suggests that the optimization surface is much less well-shaped than the one generated by the voter model, which could lead to difficulties when minimizing the prediction loss, since naive optimization techniques are prone to getting stuck in local minima. Note also that the ground truth matrix has a loss of roughly 0.4, which is nowhere close to zero. This raises the question whether the dynamics predictor can really learn SIS dynamics properly, even with the correct network structure given. To see why this would be a wrong conclusion, we have to recall that SIS is a stochastic model. It is not possible to achieve a cross entropy loss of zero, because that would mean that the predictor has to assign the probability 1 to every observation across all time steps and nodes (since $-\log(1) = 0$). In other words, zero loss means perfect prediction of the future, which is only possible for deterministic dynamics (even there, it is really only possible in theory). In fact, the author has verified qualitatively that the dynamics learner is able to learn SIS dynamics correctly by feeding a number of manually created dynamical states to the dynamics learner trained on the ground truth network structure and checking that the outputs of the predictor closely approximate the infection and recovery probabilities dictated by the model parameters.

With regards to the coupled map, we see that the mean loss curve is noticeably more flat across the different groups than in the other two plots, but there is still a steep drop near the ground truth matrix. We also see that the curve of group maxima has a different shape, with a peak at group 5. The matrix that generates this peak is the zero matrix, which represents a structure that contains no edges at all. This means that even false links are better than no links at all for coupled logistic map dynamics. The number of local minima in the surface (21) lies between the two other dynamics models.

3.2.2 Effect of dataset size

In real-world settings, observing the dynamics for a prolonged period of time is often costly, difficult or impossible, so an ideal reconstruction method should be able to reconstruct the network as accurately as possible even when dealing with sparse data. To find out if prediction-based approaches seem promising for small datasets, we study the effect of different dataset sizes on the resulting optimization surface. To do so, we generate SIS datasets containing 50, 200, 1000 and 5000 samples and carry out a brute force analysis on each of them. We intend to keep the batch size constant at 100 (except for the dataset with size 50, where we use a batch size of 50). However, this means that the fewer samples are contained in the dataset, the fewer gradient steps will be performed per epoch by the Adam optimizer that trains the parameters of the dynamics predictor (since one epoch contains $\frac{samples}{batchsize}$ batches). For a fair comparison focusing only on the effect of data sparsity, we thus increase the number of training epochs adequately when decreasing the dataset size, leading to the following combinations: 5000 samples and 40 epochs; 1000 samples and 200 epochs; 200 samples and 1000 epochs; 50 samples and 2000 epochs. In all cases, 2000 gradient steps are performed in total by the parameter optimizer.

The results of this experiment are shown in Figure 3.4. We can see clearly that the more data is used, the more the ground truth matrix sets itself apart from the other candidates. Also, the error bars, which represent the range of losses within any particular group, tend to shrink with more data. Both of these effects are of course desirable from an optimization perspective. Still, we can see that 200 samples are sufficient to ensure the "minimum requirement" for exact prediction-based reconstruction is met, which is that the global minimum of the prediction loss lies at the ground truth network structure. For the very small dataset of consisting of 50 time steps, this requirement is not fulfilled — this means that any optimization algorithm using this dataset would face an identifiability problem: Finding the global minimum of the prediction loss is no longer equivalent to reconstructing the network exactly. However, note that the minimum lies in group 1, so even in the case of an identifiability issue, it can happen that the global minimum at least lies *near* the true network structure. Note also that increasing the amount of data does not seem to have a significant impact on the amount of local minima in the optimization landscape in our experiment. This suggests that at least some part of complexity in the optimization surface of SIS dynamics is indeed inherent to the dynamical model itself and not just caused by insufficient data.

It might also seem surprising that many candidate matrices in the experiment with 50 samples achieve lower losses than even the ground truth matrix in the other experiments. However, this does not mean that the dynamics learner can learn SIS dynamics better with 50 samples than with 5000 samples — rather, it is a sign of overfitting. In the evaluation epoch, the same samples that were already used in the training phase are used again to evaluate the candidate. If this set is very small, the network can fit those data points well *without* learning the general laws driving the dynamical process. Since this seems to be a problem only on the smallest of datasets, we do not concern ourselves with the problem of overfitting any further in this work.[2]

[2] In fact, the author has experimented with splitting datasets into separate training and evaluation sets, but in general this approach produced worse optimization surfaces than using all samples for training and evaluation. The 50 sample dataset is simply too small — splitting it into 40 training and 10 evaluation samples cannot change this fact, instead it amplifies the problem.

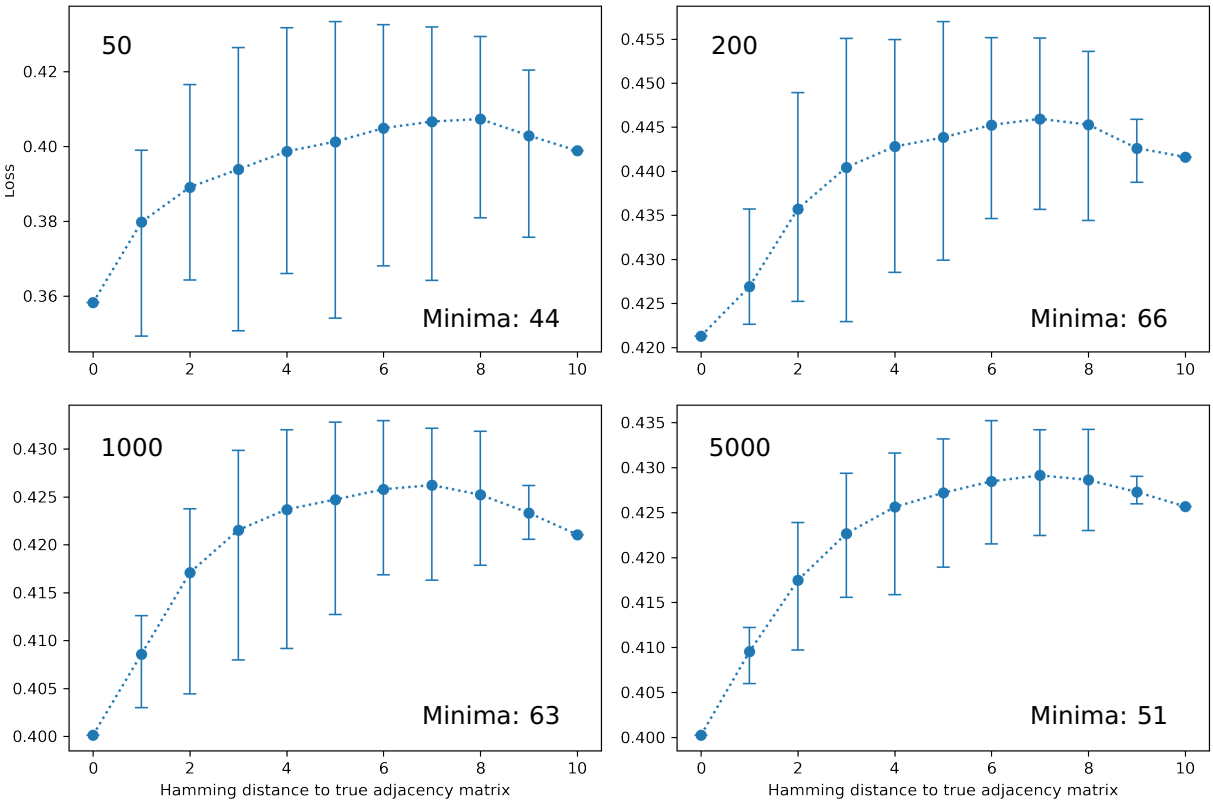

Figure 3.4: Effect of dataset size on the loss surface. The number in the top left corner indicates the number of samples in the training set.

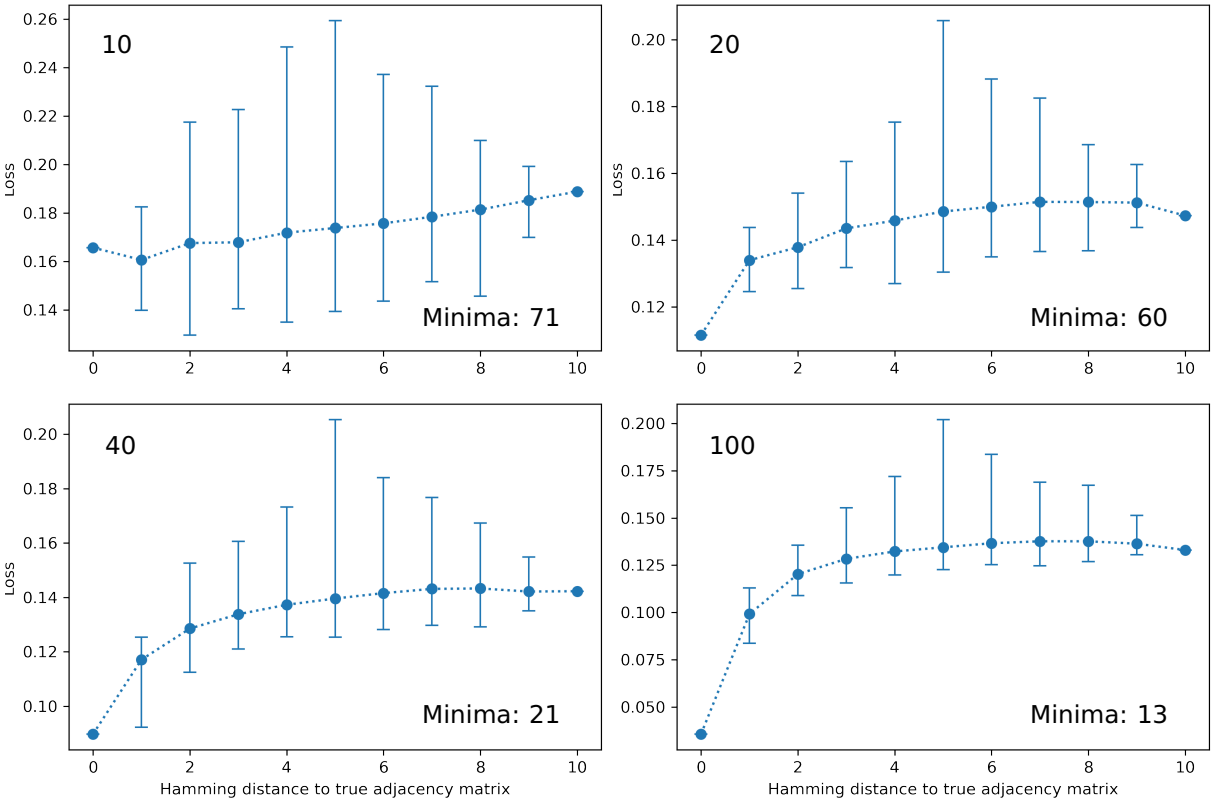

Figure 3.5: Effect of training budget on the loss surface. The number in the top left corner indicates the number of training epochs.

3.2.3 Effect of training epochs

Next, we want investigate the question at what point in the training process good candidates set themselves apart performance-wise from candidates that are far away from the true matrix — do good matrices already perform better after just a few training epochs, or does their quality only come to light in later epochs? This question is relevant because it directly relates to how much time an optimization algorithm needs to invest per matrix in order to find the true network structure.

Again, we approach this question experimentally. This time, we use coupled map dynamics and repeat the experiment four times using the same dataset but with different numbers of training epochs for evaluation. The results are shown in Figure 3.5. We can see that the more epochs are used for training, the less local minima are present in the resulting loss surface. This is a strong indicator that increasing the training time can help smoothen the loss surface, which makes the task of finding the global minimum easier for any optimization algorithm operating on the surface. In the plots, you can also see that for groups 2 to 10, the shape of the curve of the mean losses in the groups does in fact not change much with more epochs. The biggest change happens near the ground truth matrix: While for 10 epochs, the ground truth matrix does not perform better than most of the other candidates, with increasing training duration, a "valley" begins to form around the ground truth matrix. After 100 epochs, we can see that the ground truth matrix has set itself apart by a large margin.

3.3 Concluding remarks

Although all of our experiments were conducted on a very small graph, we could see that all three of the factors that we investigated (dynamical model, dataset size, training budget) play an important role in determining the shape and smoothness of the loss landscape. We can conclude that in practical applications, it will be beneficial to collect as much data as possible and train the dynamics learner until it is close to convergence on all candidate structures. Still, it would be good to have a practical way of assessing the shape of the optimization landscape for larger networks, for example to know (before applying any particular optimization algorithm) whether one has collected enough data for accurate reconstruction of the network.

For larger networks, conducting a brute force analysis of the loss landscape is not feasible, since the size of the search space grows exponentially with the number of nodes in the network. However, various techniques have been devised to characterize optimization landscapes without evaluating the whole search space. Examples of such measures include the autocorrelation of random walks on the surface [34] and characteristics based on information theory [35]. While these techniques do not provide information about the total number of local minima in the search space or the global minimum of the loss, they can be used to estimate the ruggedness of the surface. In real-world network reconstruction problems, these measures could provide a way to assess whether one has collected enough data to generate a sufficiently smooth loss surface for effective optimization. We leave the study of NR optimization landscapes on large networks using various measures of ruggedness for future research.

Chapter 4

Applying Local Search

Now, we will start thinking about how to solve the network reconstruction problem from a combinatorial optimization perspective. Because of the exponential size of the search space, $|\mathscr{A}_N| = 2^{\frac{1}{2}(N-1)N}$, and the lack of an efficient exact algorithm for network reconstruction, our work will focus on approximate methods that do not guarantee to find the global minimum of the loss surface. More specifically, we will look at a number of methods all based on the concept of *local search*. These algorithms start with some initial solution and iteratively improve the solution by replacing it with a better candidate from an appropriately defined neighborhood around the current solution. We start by addressing some general questions and design choices which are relevant for all of the methods that we are going to use. Afterwards, we will present the particular algorithms based on local search that will be evaluated in the next chapter.

4.1 Impact of neighborhood size

A crucial question when applying local search to any problem is how to define the local neighborhood in which improvements are searched in each step. In other words, we need to choose how big the steps that the algorithm is allowed to take should be. In our case, where the elements of the search space are adjacency matrices, the most straightforward choice would probably be $\text{Neighbors}(A) := \{A' \in \mathscr{A}_N \mid \text{dist}(A,A') = 1\}$. This is the same definition that we used for the analysis of local minima ín the search space in Chapter 3. From an algorithmic perspective, it means that the algorithm can add or remove one edge from the network structure in each step. However, as we have seen in Chapter 3, most optimization landscapes are riddled with local minima. Therefore, a simple local search algorithm with this neighborhood definition would most likely get stuck in suboptimal regions of the search space (except possibly on voter dynamics).

One way to alleviate this problem is to increase the step size of the algorithm — formally, this means $\text{Neighbors}_k(A) := \{A' \in \mathscr{A}_N \mid 0 < \text{dist}(A,A') \leq k\}$ for some maximum step size k. Intuitively, this allows the algorithm to step out of local optima in the optimization landscape whose basins of attraction are smaller than the step size. Thus, in theory, one can escape every local optimum (except for the one global optimum) with a sufficiently large step size. Another advantage of larger step sizes is that they allow the algorithm to traverse the search space faster. This is highly relevant for our problem, because in the worst case (that is, when the algorithm starts at the matrix A^0 with $A^0_{ij} = 1 - A^*_{ij}\ \forall i \neq j$), the length of the shortest path to the global optimum A^* grows quadratically with network

size ($O(N^2)$), since every entry of the adjacency matrix has to be flipped to reach the global optimum. A similar argumentation also holds for the average case over all possible random initializations, since the expected number of falsely initialized entries in the matrix grows quadratically as well. With a step size of 1, this means that the number of steps that the algorithm needs to reach the target structure is also $O(N^2)$. By allowing steps of size N, for instance, we could traverse the same distance with $O(N)$ steps, with obvious benefits for the run time of the algorithm.

However, these considerations only apply when assuming that the algorithm chooses the ideal neighbor in every step, i.e. it takes the shortest path to the global optimum. This will hardly be the case in practice; the *actual* amount of steps needed largely depends on the algorithm's ability to choose good neighbors.

This leads us to the main downside of allowing larger step sizes: It becomes increasingly hard to search the neighborhood for good candidates. To get a better understanding, let us take a look at the number of neighbors for different maximum step sizes: For networks with N nodes, there are $\frac{1}{2}(N-1)N$ possible undirected edges, disregarding self-loops, whose presence or absence are all specified by the upper (and also the lower) triangular parts of the adjacency matrix. For mutations of exactly k edges, there are thus $\binom{\frac{1}{2}(N-1)N}{k}$ possibilities. In total, we have

$$\left|\text{Neighbors}_k(A)\right| = \sum_{k'=1}^{k} \binom{\frac{1}{2}(N-1)N}{k'}$$

	N=5	N=10	N=15	N=20
k=1	10	45	105	190
k=2	55	1035	5565	1.8e5
k=3	175	1.5e5	1.9e6	1.1e7
k=4	385	1.6e6	5.0e7	5.4e8

Table 4.1: Numbers of elements in the k-hop neighborhood of a N×N adjacency matrix.

After recalling that an evaluation of the objective function involves the computationally expensive procedure of training a neural network, it should be clear that evaluating the whole neighborhood, or even just substantial parts of it, is not a feasible strategy for medium to large size networks when $k > 1$. This rules out classic methods for neighbor selection such as *best improvement* [36, 37], where the whole neighborhood is evaluated and the neighbor with the best objective value is chosen. Instead, we will need fast heuristics to find promising candidates for evaluation.

4.2 Heuristics for local search

In this section, we will look at two heuristics for finding promising next steps in a neighborhood around the current candidate in local search. These heuristics will calculate a so-called "mutation score matrix" $S \in \mathbb{R}^{N\times N}$, which has the same size as the adjacency matrix. Each entry S_{ij} contains an measure of how "promising" a mutation (i.e. a bit flip) of the entry A_{ij} in the candidate matrix seems; $S_{ij} > 0$ means that the heuristic expects an improvement in the loss when mutating A_{ij}, whereas $S_{ij} < 0$ means that it expects a degradation of prediction quality.

4.2.1 Gradient

The first heuristic is based on the gradient of the prediction loss in the evaluation epoch with respect to the adjacency matrix A. This approach can be motivated by the fact that gradient information was also used in the previous works on prediction-based network reconstruction [8, 9] to update the parameters of the network generator. The main difference, of course, is that we operate directly on the space of binary matrices without the probability matrix as an additional layer. This means that we cannot use gradient *descent* to change the network structure — but we can still calculate the gradient *itself* and use it to guide our discrete mutations of the matrix.

The exact procedure to calculate the mutation score matrix S^{grad} is as follows: When evaluating the prediction quality of a matrix A, we calculate the gradient $\frac{\partial R_X(A,\Phi)}{\partial A}$ of the empirical risk (= average prediction loss, see Section 3.1.1) with respect to the adjacency matrix via backpropagation after the evaluation epoch. Now, we can calculate an asymmetric mutation score matrix S^{asym} as follows:

$$S^{\text{asym}} = (1-A) * \left(-\frac{\partial R_X(A,\Phi)}{\partial A} \right) + A * \frac{\partial R_X(A,\Phi)}{\partial A}$$

where $*$ represents *element-wise* multiplication of matrix entries. This means that

$$S^{\text{asym}}_{ij} = \begin{cases} \frac{\partial R_X(A,\Phi)}{\partial A_{ij}}, & \text{if } A_{ij} = 1 \\ -\frac{\partial R_X(A,\Phi)}{\partial A_{ij}}, & \text{if } A_{ij} = 0 \end{cases}.$$

The intuition behind this formula is that the partial derivative $\frac{\partial R_X(A,\Phi)}{\partial A_{ij}}$ estimates the change in the empirical risk when taking a small step in the direction A_{ij}. Therefore, if there is a false positive at A_{ij}, this partial derivative should have a positive value, indicating that taking a step in the direction $-A_{ij}$ (i.e. a move towards zero in that entry) would decrease the prediction loss. So, if $A_{ij} = 1$, a large partial derivative is an indicator that a mutation of this position seems promising. Analogously, one can argue that when A_{ij} is a false negative, the partial derivative should have a negative value, indicating that a move in the direction A_{ij} (i.e. towards one) would decrease the prediction loss. For this reason, we flip the sign of the partial derivative in this case, so that a positive value in S^{asym}_{ij} always corresponds to a promising mutation, no matter whether $A_{ij} = 0$ or $A_{ij} = 1$.

In applications with directed graphs, one could use S^{asym} directly to guide mutations. Since our search space is only composed of symmetric matrices, the mutation score matrix must be symmetric as well, because when mutating an entry A_{ij}, we always also mutate A_{ji} to stay in our search space. We thus calculate the symmetric mutation score matrix S^{grad} as follows:

$$S^{\text{grad}} = S^{\text{asym}} + (S^{\text{asym}})^\top$$

Written element-wise, this means that

$$S^{\text{grad}}_{ij} = \begin{cases} \frac{\partial R_X(A,\Phi)}{\partial A_{ij}} + \frac{\partial R_X(A,\Phi)}{\partial A_{ji}}, & \text{if } A_{ij} = 1 \\ -\frac{\partial R_X(A,\Phi)}{\partial A_{ij}} - \frac{\partial R_X(A,\Phi)}{\partial A_{ji}}, & \text{if } A_{ij} = 0 \end{cases}.$$

This step is justified theoretically by the fact that it calculates the *directional derivative* of the risk in the direction of the symmetric mutation under the assumption that $R_X(\,\cdot\,,\Phi)$ is totally differentiable (see Section 4.2.3).

The fly in the ointment here is that the gradient (and also the directional derivative) only really

gives information about *infinitesimally* small steps, whereas we intend to take crude binary steps, so as to stay in our binary search space. While the principle of using gradient information to guide non-infinitesimal steps is a common one — this is done in all gradient-based optimization algorithms —, we will have to verify experimentally that the gradient can still guide steps as large as the ones we intend to take.

4.2.2 Risk evaluation

The second heuristic involves directly testing each possible mutation by performing an additional evaluation epoch of the dynamics learner on the given dataset. First, note that it is possible to *train* the dynamics learner on some network structure A and then *apply* it together with another network structure A': As discussed in Section 2.5, all of the model parameters Φ lie inside the functions f_{node}, f_{edge} and f_{output}, so we can just use those same trained functions together with the changed network structure A'. While these parameters probably won't be optimal for A', it is reasonable to assume that we can still obtain useful predictions as long as the new structure A' is not radically different from A, as is the case when mutating just a single entry of A.

We use this idea to evaluate mutations of the current candidate A as follows: During evaluation of the candidate A, we store its trained GNN parameters Φ. Then, for each one-hop neighbor $A^{\sim(i,j)} \in \text{Neighbors}_1(A)$, where $A^{\sim(i,j)}$ denotes the matrix where the entries (i,j) and (j,i) have been flipped and all other entries are identical to A, we calculate the empirical risk $R_X(A^{\sim(i,j)}, \Phi)$ *without retraining the dynamics learner* (this last part is what differentiates this heuristic from a full evaluation of the neighbor). The entries of the mutation score matrix S^{eval} are then calculated as the difference between the risk of the current candidate and the risk of the mutated matrix:

$$S^{\text{eval}}_{ij} = R_X(A, \Phi) - R_X(A^{\sim(i,j)}, \Phi)$$

Note that since this criterion is inherently symmetric, we only need to calculate the upper triangular part of S^{eval} in practice. Also, we can speculate that this heuristic may be a bit pessimistic — it only regards a mutation as "promising" (i.e. $S^{\text{eval}}_{ij} > 0$) if it already performs better than the current candidate without re-training the dynamics predictor.

4.2.3 Theoretical relationship

In the following, we will investigate the relationship between the two heuristics. In particular, we will show that the risk evaluation heuristic is a *finite difference* approximation of the gradient heuristic.

For this part, we interpret the risk $R_X(\,\cdot\,, \Phi) : \mathbb{R}^{N\times N} \to \mathbb{R}$ as a function on a continuous domain. Furthermore, we assume that it is totally differentiable.[1] We write the canoncial basis matrices of $\mathbb{R}^{N\times N}$ as E^{ij} (i.e. E^{ij} is a matrix that has a one in position (i,j) and zeros in all other entries). We define $U^{ij} := \frac{E^{ij}+E^{ji}}{\sqrt{2}}$ — this normalized matrix represents the direction of a symmetric matrix mutation.

We now establish the relationship between the evaluation heuristic and the *directional derivative*

[1] This is not fulfilled in our implementation, e.g. because we use *ReLU* activation functions, but one could make it differentiable by using differentiable activation and loss functions. Anyway, this section serves the purpose of illustrating the theoretical relationship between the two heuristics, so it does not need to be directly applicable to the details of our implementation.

$\frac{\partial R_X(A,\Phi)}{\partial U^{ij}}$ of the risk with respect to U^{ij}, which is defined as follows:

$$\frac{\partial R_X(A,\Phi)}{\partial U^{ij}} = \lim_{h\to 0} \frac{R_X(A+hU^{ij},\Phi) - R_X(A,\Phi)}{h} \quad (*)$$

This directional derivative describes the change in the empirical risk when taking an infinitesimal step in the direction U^{ij}. Note that using the directional derivative instead of a partial derivative is necessary because we use symmetric mutations in our application. We approximate this limit (very crudely) with a difference quotient:

Assuming $A_{ij} = 0$:

$$\begin{aligned}(*) &\approx \frac{R_X(A+\sqrt{2}U^{ij},\Phi) - R_X(A,\Phi)}{\sqrt{2}} \\ &= \frac{1}{\sqrt{2}}\big(R_X(A+E^{ij}+E^{ji},\Phi) - R_X(A,\Phi)\big) \\ &= \frac{1}{\sqrt{2}}\big(R_X(A^{\sim(i,j)},\Phi) - R_X(A,\Phi)\big) \\ &= -\frac{1}{\sqrt{2}} S_{ij}^{\text{eval}}\end{aligned}$$

Assuming $A_{ij} = 1$:

$$\begin{aligned}(*) &\approx \frac{R_X(A-\sqrt{2}U^{ij},\Phi) - R_X(A,\Phi)}{-\sqrt{2}} \\ &= -\frac{1}{\sqrt{2}}\big(R_X(A-E^{ij}-E^{ji},\Phi) - R_X(A,\Phi)\big) \\ &= -\frac{1}{\sqrt{2}}\big(R_X(A^{\sim(i,j)},\Phi) - R_X(A,\Phi)\big) \\ &= \frac{1}{\sqrt{2}} S_{ij}^{\text{eval}}\end{aligned}$$

Combining both approximations, we get:

$$\begin{aligned}S_{ij}^{\text{eval}} &\approx \begin{cases} \sqrt{2}\,\frac{\partial R_X(A,\Phi)}{\partial U^{ij}}, & \text{if } A_{ij} = 1 \\ -\sqrt{2}\,\frac{\partial R_X(A,\Phi)}{\partial U^{ij}}, & \text{if } A_{ij} = 0 \end{cases} \\ &= \sqrt{2}\,(-1)^{(1-A_{ij})}\frac{\partial R_X(A,\Phi)}{\partial U^{ij}}\end{aligned}$$

In words, this means that the evaluation heuristic is a finite difference approximation of the directional derivative (modulo sign flips and a constant factor). Now, we continue by showing that the last expression is in fact identical to the gradient heuristic:

$$\begin{aligned}&\sqrt{2}\,(-1)^{(1-A_{ij})}\frac{\partial R_X(A,\Phi)}{\partial U^{ij}} \\ \overset{\text{(I)}}{=}\ &\sqrt{2}\,(-1)^{(1-A_{ij})}\left\langle \frac{\partial R_X(A,\Phi)}{\partial A}, U^{ij}\right\rangle_F \\ =\ &(-1)^{(1-A_{ij})}\left\langle \frac{\partial R_X(A,\Phi)}{\partial A}, E^{ij}+E^{ji}\right\rangle_F \\ =\ &(-1)^{(1-A_{ij})}\left(\frac{\partial R_X(A,\Phi)}{\partial A_{ij}} + \frac{\partial R_X(A,\Phi)}{\partial A_{ji}}\right) \\ =\ &S_{ij}^{\text{grad}}\end{aligned}$$

where we have used in (I) that $R_X(\,\cdot\,,\Phi)$ is totally differentiable, and $\langle\,\cdot\,,\,\cdot\,\rangle_F$ is the Frobenius inner product on matrices, defined as $\langle A,B\rangle_F = \sum_{i=1}^N \sum_{j=1}^N A_{ij}B_{ij}$.

We have now seen two things: (1) Disregarding the sign flips and the constant factor $\sqrt{2}$, the gradient heuristic S_{ij}^{grad} is equal to the directional derivative of the risk in the direction of the symmetric mutation U^{ij} under the assumption that $R_X(\cdot,\Phi)$ is totally differentiable. (2) Disregarding the sign flips and the constant factor, the risk evaluation heuristic S^{eval} is essentially a finite difference approximation of the same directional derivative. Putting (1) and (2) together, we can say that the risk evaluation heuristic is a finite difference approximation of the gradient heuristic.

Note, however, that calling the risk evaluation heuristic a crude approximation of the gradient heuristic should not directly lead us to think that the former is inferior to the latter — after all, we are indeed interested in exactly those large discrete steps that the risk evaluation heuristic uses, and not some infinitesimally small steps for which the gradient heuristic would be more accurate. In that sense, it should be equally valid to say that the gradient heuristic is an approximation of the finite difference quotient that we calculate when using an additional evaluation epoch. Then again, one could argue in favor of the gradient heuristic that since the parameters of the dynamics predictor remain unchanged, the large discrete steps might suffer from more uncertainty than the infinitesimal step estimated by the gradient heuristic. We will need experimental verification to determine which argument outweighs the other.

4.3 Choosing a neighbor

Now that we have some heuristics to inform us about our neighborhood, we still need to specify how exactly we use the heuristic to search the neighborhood $\text{Neighbors}_k(A)$. In the following, we will consider a *first improvement* approach, where the algorithm evaluates neighbors one by one and accepts the first neighbor that has a lower empirical risk than the current candidate. There are two ways that a heuristic S could be used in this setting:

1. S could be used to determine the order of evaluation of neighbors.
2. S could be transformed into a probability distribution over the neighbors, from which promising candidates could be sampled.

Option 1 would work as follows: For $k = 1$, one would first try out the mutation $A^{\sim(i,j)}$ with the highest mutation score S_{ij}. If that mutation does not improve the objective function after a full evaluation, one would proceed with the second highest S_{ij}, and so on. For $k > 1$, one could choose the indices with the top-k highest mutation scores for mutation. However, it is not entirely clear how one should proceed when this candidate is rejected; one could of course replace the worst of the k mutations with the next best one, but if we proceed with this method, we would potentially try out many candidates in a row that all have the $k-1$ mutations in common that were rated highest by the heuristic and only replace the last one. Another method would be to vary the step size, for example by trying out the neighbor where only $\frac{k}{2}$ matrix entries with the highest scores are flipped, then the one with $\frac{k}{4}$ flips, and so on.

However, such an "ordered search" approach is actually suboptimal for our application, because as was mentioned in Section 3.1.2, our evaluation function is non-deterministic. This means that it can happen that a "good" neighbor, i.e. one which has a better prediction performance on average, performs worse than the current candidate simply because of bad parameter initialization.[2] In a first improvement setting, this means that the neighbor gets rejected and never evaluated again, which increases the risk of getting stuck at suboptimal points in the search space. It is possible to somewhat mitigate this risk by performing multiple evaluations per neighbor and averaging or taking the minimum of the empirical risk across all runs before deciding whether to accept or reject, but this of course multiplies the run time of the algorithm.

[2] The author has verified in rudimentary experiments on synthetic datasets that this problem indeed occurs.

4.3.1 Sampling from heuristics

We will now consider option 2, which is also the one that we use in our experiments: We transform the mutation score matrix S into a probability distribution over the neighbors, and then repeatedly sample from this distribution until we find a neighbor that improves our objective function. An advantage of this approach is that it alleviates the problem of noisy evaluation; if a neighbor is rejected unjustly, it will sooner or later be evaluated again (how soon depends on the probability that was assigned to it, of course). This approach also places more weight on the heuristic itself: If the heuristic assigns a probability of 0 to a "good" neighbor, the algorithm will never even give it a chance, whereas in the evaluation order approach, any neighbor is evaluated at some point.

Single step

For $k = 1$, we proceed as follows: First, all entries S_{ij} with $S_{ij} < 0$ as well as all entries on or below the main diagonal are set to zero. We denote the resulting upper triangular matrix as $\hat{S}$. Then, we normalize the matrix by dividing it element-wise by the sum of all entries: $P^{k=1} = \frac{1}{\sum_{i,j} \hat{S}_{ij}} \hat{S}$. Now, since the sum of all entries is 1, we can treat $P^{k=1}$ as a categorical distribution over mutations. We sample an entry (i, j) (since we set all entries on or below the main diagonal to zero, we have $j > i$ in any case) from $P^{k=1}$ and then mutate both the entry (i, j) and the entry (j, i) of the candidate matrix A. Note that each neighbor is chosen with a probability that is proportional to its heuristic score, and only neighbors with a positive heuristic score have a positive probability.

For $k > 0$, one could generate samples by repeatedly sampling mutations from $P^{k=1}$ without replacement (which could be achieved by setting the probability of all previously drawn mutations to 0 and renormalizing) and afterwards evaluating the matrix where all of the sampled mutations have been applied.

Dynamic steps

A different, but interesting approach would be to not place any rigid bounds on the step size and instead choose the step size dynamically based on the heuristic matrix S along with the mutations themselves. This makes sense intuitively, because if there are a lot of high mutation scores, we would like to try a neighbor with more mutations than for example when the heuristic only makes out a single promising mutation, where it would be sensible to try out the 1-hop neighbor that is only mutated in that particular position. Formally, this means we choose neighbors from the set $\text{Neighbors}_\infty(A) = \mathscr{A}_N$, i.e. from the whole search space. One could do this as follows:

We first set all negative entries as well as all entries on or below the main diagonal of S to zero, denoting the result as $\hat{S}$. Then, we normalize the entries of $\hat{S}$ to the range $[0, 1]$ by dividing all entries by $\max_{(i,j)} \hat{S}_{ij}$, denoting the result as $P^{k=\infty}$. Now, we interpret the entries of $P^{k=\infty}$ as mutation probabilities; to sample a neighbor, we perform each symmetric mutation of the entries (i, j) and (j, i) (here, $j > i$) with probability $P^{k=\infty}_{ij}$.

Now, if $\hat{S}$ contains just a single positive value $\hat{S}_{kl} > 0$, the procedure above deterministically returns the neighbor $A^{\sim(k,l)}$ since $P^{k=\infty}_{kl} = 1$ and $P^{k=\infty}_{ij} = 0 \ \forall i \neq k, \ j \neq l$. On the other hand, if there are many positive entries in $\hat{S}$, we should expect the sampled neighbors to contain many more mutations, because the entry with the highest mutation score is mutated certainly and the other entries are mutated with a probability proportional to their relative mutation score.

While in principle, this technique could hardly be called *local* search anymore when the neighborhood is the whole search space, we observe in our experiments that it tends to mutate only a fraction of all matrix entries in practice, which justifies the classification as a local search method again.

One of the aspects that motivated our discussion of larger step sizes was that they can allow us to step out of local minima. Now that we have seen the details, let us revisit this idea. The key realization here is that in order to step out of local minima, it is not enough to just increase the theoretical step size; it only works when our heuristics actually find those candidates in the extended neighborhood that lie outside of the basin of attraction of the local minimum. For the risk evaluation heuristic, we can argue that this might not be the case, since only the one-hop neighbors are evaluated to get the mutation scores. In a local minimum, all of those scores might well be negative, leading to an undefined P, since we divide by the sum or the maximum of all positive entries of S, which would be 0 in this case.[3] Thus, larger step sizes alone will most likely not solve the problem of local minima. As a consequence, we will combine the techniques discussed until now with different search algorithms to improve robustness.

4.4 Search algorithms

4.4.1 Hill climbing

Probably the simplest of all local search algorithms is the *hill climbing* algorithm. Starting with a random element of the search space, it repeatedly samples neighbors of this element until it finds one with a better objective function value (in our case, contrary to what the name of the algorithm suggests, this means a *lower* prediction loss). It then accepts that neighbor as the next step and the process is repeated until some termination criterion — in our case, a fixed number of steps — is reached.

4.4.2 Simulated annealing

Simulated annealing [38] is a local search technique that differs from hill climbing by the fact that it can also accept solutions that are *worse* than the current solution as the next step, which can allow the algorithm to escape from local minima. The probability of accepting worse candidates is gradually decreased during execution of the algorithm. Therefore, the algorithm tends to *explore* the search space in the beginning and then gets more and more selective with its next steps as the search progresses.

The simulated annealing algorithm is shown in Algorithm 1. The value which controls the selectivity is called the *temperature* T. A high temperature means that the algorithm is likely to accept solutions even if they are worse than current solution, and a temperature of 0 means that only better solutions will be accepted. In our implementation, we decrease the temperature linearly, that is $\textit{get\_temperature(step, NUM\_STEPS)} = \frac{1}{100}(1 - \frac{\textit{step}}{\textit{NUM\_STEPS}})$. (The factor $\frac{1}{100}$ was chosen by hand to make the resulting probabilities appropriate for the loss functions used in our work.)

[3] We could try to solve this problem by falling back to the gradient heuristic when the evaluation heuristic returns only negative entries, or by somehow shifting the mutation scores to a positive range. In our implementation, however, we simply accept that we are stuck when we get an all-negative S, since our goal is primarily to see which heuristic is better, and not to build a complicated algorithm that optimizes performance.

Algorithm 1 Simulated annealing

```
A ← random initialization
for step = 1, ..., NUM_STEPS do
    T ← get_temperature(step, NUM_STEPS)
    R ← eval(A)
    A' ← sample_neighbor(A)
    R' ← eval(A')
    if random(0,1) < get_acceptance_prob(R, R', T) then
        A ← A'
    end if
end for
return A
```

The function $get\_acceptance\_prob(R, R', T)$ determines the actual acceptance probability of a candidate given the scores (in our case, the empirical risks) of the candidate and the current solution and the temperature. Classically, the function

$$get\_acceptance\_prob(R, R', T) = \begin{cases} 1, & \text{if } R' < R \\ \exp(-\frac{R'-R}{T}), & \text{if } R' \geq R \end{cases}$$

is used for this (see [39]), and we use this function in our implementation as well. In the case $T = 0$, we define the second case to be 0.

In the *sample_neighbor* function, the previously discussed heuristics are used to sample promising next candidates. Note that when using the risk evaluation heuristic, it also takes the parameters of the dynamics learner trained on A as input (the training, of course, is part of the previous line where A is evaluated); we abstain from including this in the pseudocode so as to not make it too technical.

It is worth pointing out that the simulated annealing algorithm is equivalent to the hill climbing algorithm when the temperature is 0.

4.4.3 Population-based local search

In contrast to single-candidate search methods such as hill climbing, population-based algorithms maintain a whole population of candidates. The basic idea is to combine information from multiple search trajectories to guide the search towards the most promising parts of the search space [40]. Population-based search methods originate from the field of evolutionary computation, a research area in which principles inspired by biological evolution and natural selection are used to solve complex optimization problems [41]. This origin is also reflected in the terminology commonly used in this context (*individual* = a candidate solution, *population* = a collection of candidate solutions, *mutation* = the process of sampling a neighbor of an individual by applying small permutations).

The pseudocode of the PBLS as we use it is given in Algorithm 2. In short, the algorithm works as follows: In the beginning, the population is initialized with *POP_SIZE* random matrices. Then, in each generation, a number (*NEW_POP_SIZE*) of new candidates is created by mutating elements of the current population using the same heuristics we discussed previously. The current population as well as the newly generated individuals are evaluated, and the *POP_SIZE* best individuals are chosen as the next population (this is called *truncation selection*). When choosing the parameters *POP_SIZE* and *NEW_POP_SIZE*, the principle "the more, the better" applies, but of course, increasing these parameters increases the computational workload per generation, so that the limiting factors

Algorithm 2 Population-based local search

```
pop ← list of POP_SIZE randomly initialized matrices
for gen = 1, ..., NUM_GENS do
    new_pop ← empty list                          ▷ Generate new population
    for i = 1, ..., NEW_POP_SIZE do
        A ← random individual from pop
        A' ← sample_neighbor(A)
        new_pop[i] ← A'
    end for
    scores ← empty dictionary                     ▷ Evaluate old and new population
    for all A ∈ pop ∪ new_pop do
        scores[A] ← eval(A)
    end for
    pop ← select(pop, new_pop, scores)            ▷ Select best individuals
end for
return best individual in pop
```

are the computational resources and the time budget of the user.

To expand a little more on the relation of our PBLS to existing research in evolutionary computation, we want to point out the main differences to a commonly used class of algorithms, namely *genetic algorithms* (GAs). The key differences are:

- We do not use *random* mutations in the same sense of the word as it is used in GAs — due to the heuristics we use, our mutations are *targeted*: They are heavily biased towards certain regions of the neighborhood, whereas in GAs, the mutations are completely (uniformly) random.
- We use smaller population sizes (~ 10) than what is usual for GAs (> 100 [42]). We are forced to do this because the fitness evaluation is such a costly procedure that performing several hundreds of evaluations per generation did not seem practical with the limited computational resources available to us (we perform our experiments on a single GPU).
- GAs use so-called *crossover* operators to generate new offspring from the existing population, whereas we rely on mutation only. Crossover is usually performed by exchanging parts of the genetic representation (in our case, the adjacency matrix) of the two individuals. Adding crossover to our method is an opportunity for future research.

Chapter 5

Experiments

In this chapter, we put the methods discussed in the last chapter into practice. For this, we conduct a number of experiments on synthetic datasets. We compare the heuristics for neighbor selection to random search and study the effect of allowing dynamic step sizes. Then, we compare the different local search methods in combination with the heuristics on multiple datasets. We further compare these combinatorial optimization methods to Zhang et al.'s Gumbel graph network (GNN) and statistical approaches.

5.1 Data generation

In this section, we will go over the datasets that we used for evaluating our approach.

Barabási-Albert graphs

For our experiments, we used randomly generated *Barabási-Albert* graphs [43] as the underlying structure for the dynamical processes. The Barabási-Albert model uses a *preferential attachment* mechanism for generating random graphs that share similarities with many real-world networks. The algorithm takes two parameters: $n \in \mathbb{N}$ is the desired number of nodes in the network and $m \in \mathbb{N}$ determines the connectivity of the network.

The algorithm starts with a set of m nodes without connections, and then iteratively adds new nodes to the network until the network has reached the desired size n. For each new node, it adds m edges that connect the new node to the existing network. When adding these edges, the probability that an edge connects the new node to any existing node i in the network is proportional to the degree of the existing node, $\deg(i)$. This leads to highly connected nodes becoming ever more connected, while nodes with low degree will likely not be chosen for new edges ("the rich get richer, the poor get poorer").

The key property that Barabási-Albert graphs have is that they are *scale-free*, which means that their degree-distribution follows a power law: The fraction $P(k)$ of nodes with degree k is given (approximately) as $P(k) \sim k^{-3}$. The term scale-free refers to the fact that this degree distribution occurs independently of the network size n. Many real-world networks have been reported to exhibit such a scale-free degree distribution, for example the internet or networks of scientific publications connected via citations [44].

Figure 5.1 shows the two Barabási-Albert graphs that we use in our experiments.

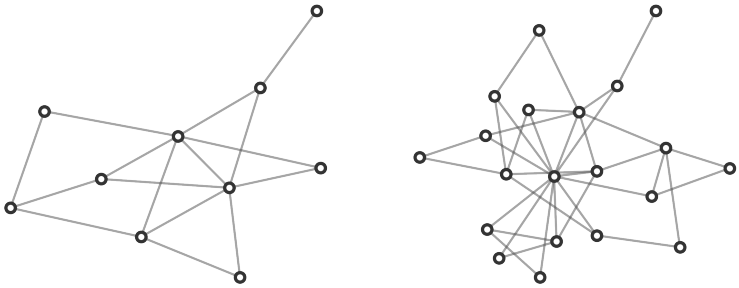

Figure 5.1: Two Barabási-Albert graphs with $n = 10$, $m = 2$ (left) and $n = 20$, $m = 2$ (right).

Simulations of dynamical models

To generate datasets, we simulate the dynamical models introduced in Section 2.7 on Barabási-Albert graphs. As mentioned in the background section, in some cases we need to observe multiple independent simulations to make network reconstruction possible. We handle this in the following way:

- **Voter:** We observe the dynamics until consensus is reached. Then, we reinitialize[1] the node states to random values, simulate until consensus, and repeat until the desired number of samples is reached.
- **SIS:** Here, we ensure that the infection does not die out by choosing suitable infection and recovery rates for the simulations. We use $\beta = \mu = 0.15$ the 10-node BA graph and $\beta = \mu = 0.1$ for the 20-node BA graph.
- **Coupled map:** For non-chaotic dynamics ($r = 3.5$), we observe in our simulations that the dynamical process usually "converges" to regular oscillations after very few steps (<10). Thus, we simulate coupled map dynamics for a fixed duration of 10 steps and then reinitialize. For chaotic dynamics ($r = 4$), we could in principle use one continuous simulation for network reconstruction, since the dynamical process does not converge to a steady state or regular oscillations. But in order to make the dataset comparable to the non-chaotic one, we use the same reinitialization procedure here as well (after every 10 steps). Note that in contrast to [9], we use $r = 4$ for chaotic dynamics, whereas they use $r = 3.6$. While $r = 3.6$ does lie above the critical threshold (see Section 2.7.3), we observe that the resulting dynamical time series is still much more "orderly" compared to $r = 4$. We thus choose $r = 4$ to see if prediction-based network reconstruction can still work with highly chaotic dynamics.

In the following, we use the nomenclature [dynamics model]-[parameter (optional)]-[graph]-[dataset size] for our datasets, e.g. CM-3.5-BA20-1k for 1000 time steps of non-chaotic coupled map dynamics on a 20-node Barabási-Albert graph, or SIS-BA10-5k for 5000 time steps of SIS dynamics on a 10-node BA graph.

[1] When we say that we "reinitialize" the dynamics, we don't just concatenate the old time series with the new one, because in that case the dynamics learner would regard the transition from the consensus state to the randomly initialized one as part of the dynamics model itself. Instead, we use a data structure that keeps all of the continuous time series separate.

5.2 Heuristics and step size

First, we want to study the effectiveness of the heuristics and the effect of using dynamic steps. We execute the simple hill climbing algorithm combined with both heuristics and the step size either fixed to $k = 1$ or using a dynamic step size on Voter and CM-3.5 datasets. Experimental details for all experiments in this chapter can be found in Appendix A.2.

Voter

Figure 5.2 shows the average loss (empirical risk) and network accuracy curves of single executions of the hill climbing algorithm on the Voter-BA20-5k dataset. The network accuracy is calculated as the number of correctly classified entries of the adjacency matrix divided by the total number of matrix entries. We can see that as the hill climbing algorithm minimizes the prediction loss, the reconstructed network gets closer and closer to the true network structure, which again confirms the underlying assumption of prediction-based network reconstruction.

In Figure 5.2a, we can see that both heuristics are highly effective in searching the 1-hop neighborhood for good candidates compared to random search — the risk evaluation heuristic finds the ground truth network structure after 111 steps, the gradient heuristic finds it after 163 steps, whereas random search only achieves a network accuracy of roughly 0.64 after 200 steps. While we can expect random search to keep improving its solution very slowly, it is clear that using any of our heuristics accelerates the search by a large factor. Consider also that the closer random search gets to the ground truth matrix, the smaller its chances of choosing a "good" mutation get, since there will be fewer incorrect entries in the adjacency matrix.

Figure 5.2b shows the same experiment, but using dynamic steps as described in Section 4.3. We observe that with dynamic step size, we get another strong improvement in the efficiency of local search — the hill climbing algorithm finds the ground truth matrix after 10 steps using the risk evaluation heuristic and after 27 steps using the gradient heuristic.

When comparing the efficiency of the heuristics, one must keep in mind that the risk evaluation heuristic has a higher run time per step because of the additional passes over the training data (see Section 4.2.2) - on average, it takes roughly 1.6 times longer on this dataset. This is the reason why there is no clear winner here in terms of measured run time — in the executions shown in the plots, the gradient heuristic was slightly faster for steps of size 1, whereas for dynamic step size the evaluation heuristic was faster. In any case, the exact run time can vary significantly in repeated runs due to (1) the fact that our algorithm still uses some degree of randomness to make its steps and (2) the random initialization of the network structure at the start of execution.

Coupled map

Figure 5.3 shows the same experiment, but on the CM-3.5-BA10-1k dataset. Again, we can see that the general premise that network accuracy increases as prediction loss is minimized is fulfilled. The relatively good performance of random search on this dataset is simply due to the smaller underlying network, which makes it possible to traverse the search space relatively quickly even with random mutations. What is noteworthy about this experiment is that the evaluation heuristic gets stuck in a local optimum (step 21 onwards in plot (a) with 0.9111 accuracy and step 8 onwards in plot (b) with 0.9778 accuracy). As we have speculated in the previous chapter, what happens at these points is that all entries of the mutation score matrix S^{eval} are negative, which means that no mutations

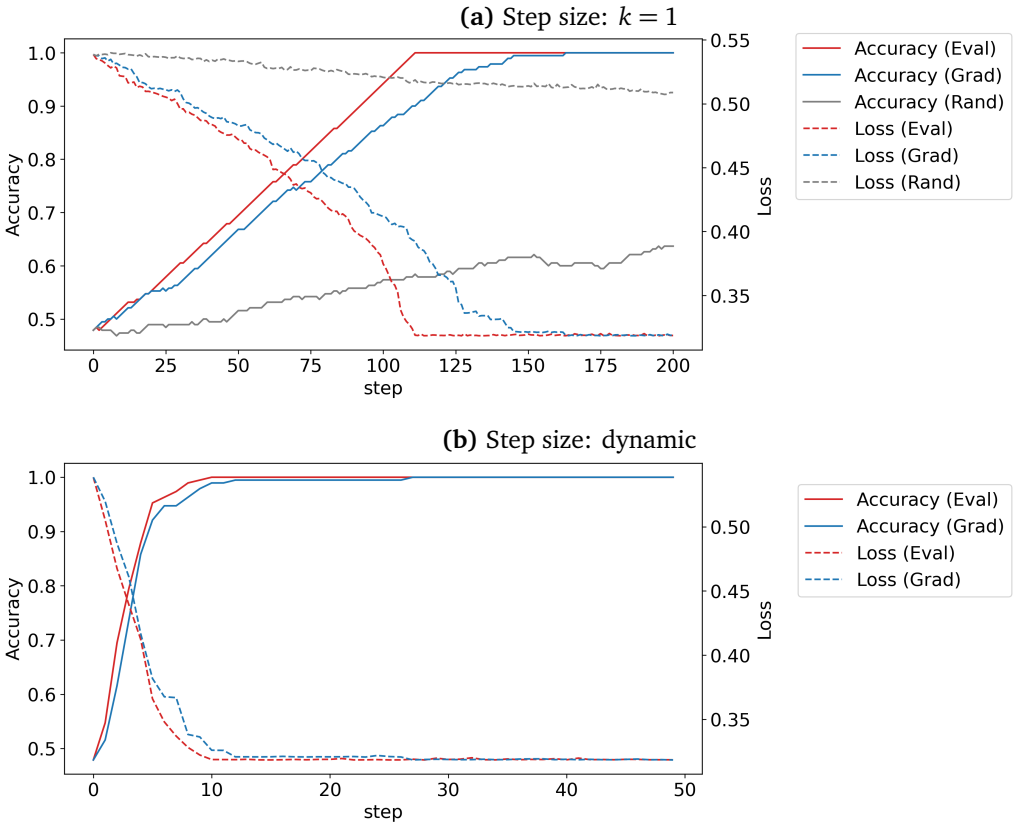

Figure 5.2: Comparison of heuristics and step size on the Voter-BA20-5k dataset.

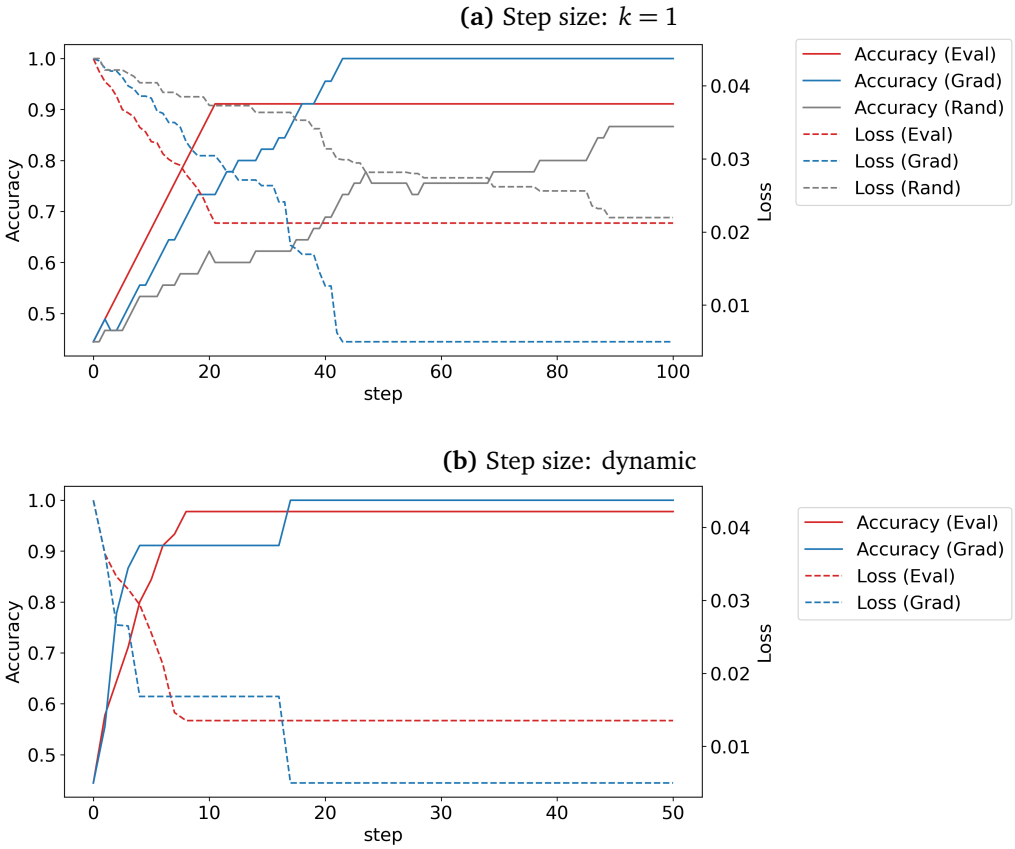

Figure 5.3: Comparison of heuristics and step size on the CM-3.5-BA10-1k dataset.

are executed, so the algorithm has no chance to escape from the local minimum (c.f. footnote on Page 30).

In further experiments (not shown) on the same dataset with different random number generator seeds, the author has observed that the gradient heuristic can get stuck in local optima as well. In contrast to the evaluation heuristic, though, it is not the case that all entries of S^{grad} are negative. This means that the algorithm keeps trying out candidates from the neighborhoods, but these candidates do not improve the loss compared to the local minimum.

To conclude this section, we can say that we have verified that the heuristics work in general, but can still lead to local minima. It is not clear yet if one heuristic is better than the other. We have also seen that using dynamic step sizes can drastically speed up the time until convergence, so we will use this sampling technique in all experiments from now on.

5.3 Comparison of reconstruction quality

Now, we will conduct an in-depth comparison of the different search algorithms in combination with the heuristics.

5.3.1 Baselines

We compare our methods to three baselines, namely Zhang et al.'s GGN method and two statistical methods, which treat each observation X^t as a sample from a time-invariant joint probability distribution over all nodes $\mathbf{X}_1, \ldots, \mathbf{X}_N$.

- **Correlation:** This method uses the correlation matrix, which is a matrix that contains Pearson's correlation coefficient between the observations for each pair of nodes:

$$C_{ij} = Corr(X_i, X_j) = \frac{Cov(X_i, X_j)}{\sqrt{Var(X_i)\,Var(X_j)}} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}} \quad \forall 1 \leq i, j \leq N$$

  where Σ represents the covariance matrix. The correlation coefficient does not distinguish between direct and indirect effects, so it would be reasonable to expect this to be a fairly weak technique for network reconstruction.

- **Partial Correlation:** This method uses the partial correlation between each pair of nodes. As stated in [45], the partial correlation coefficient quantifies the correlation between two variables conditioned on one or several other variables. In our case, the two variables are some pair of nodes and we condition on *all* other nodes, so that only direct interactions should be discovered and any indirect effects should be filtered out. The partial correlation between the nodes $\mathbf{X}_i$ and $\mathbf{X}_j$ conditioned on all other nodes can be written as:

$$PC_{ij} = \frac{\Sigma^{-1}_{ij}}{\sqrt{\Sigma^{-1}_{ii}\Sigma^{-1}_{jj}}} \quad \forall 1 \leq i, j \leq N$$

  where Σ^{-1} is the inverse of the covariance matrix [15]. In previous works, it has been demonstrated that partial correlation is an effective tool to recover unknown network structures in the context of gene regulatory networks [45] and in the context of brain networks [15].

Note that neither of these methods uses the ordering of the observations, so the time information is neglected — correlation and partial correlation can only discover dependencies *within* snapshots, as opposed to dependencies over consecutive time steps. Also, these methods yield a matrix with real instead of binary entries. Thus, we need to apply a threshold in order to be able to compare them with the same measures that we use for the prediction-based methods. We do this by providing the correct number of edges $|E|$ that are present in the respective ground truth network and choosing the top-$|E|$ largest entries (disregarding the main diagonal) of the (partial) correlation matrix as edges and the rest as non-edges (we set them to zero).

5.3.2 Results

The results of our experiments are shown in Table 5.1 (voter), Table 5.2 (SIS) and Table 5.3 (coupled map). We show the mean accuracy (⌀ACC), true positive rate (⌀TPR) and false positive rate (⌀FPR) as well as the maximum accuracy (MAX ACC) over 5 independent executions of the prediction-based methods (statistical methods only need to be evaluated once because they are deterministic). Like before, the accuracy is calculated as the number of correctly classified entries of the adjacency matrix divided by the total number of matrix entries. The TPR specifies the number of matrix entries correctly classified as edges (true positives) divided by the total number of edges (ones) in the ground truth matrix. The FPR specifies the number of matrix entries falsely classified as edges (false positives) divided by the total number of non-edges (zeroes) in the ground truth matrix. In any case, the main diagonal is not taken into account.

Statistical methods vs. prediction-based methods

On voter dynamics, statistical methods fail completely: The false positive rates are greater than the true positive rates for both methods on both datasets. This may seem surprising. Upon further inspection, it turns out that this has to do with the *synchronous* updates that we use for the voter dynamics, which means that we calculate the probabilities for the time step $t + 1$ from the time step t for all nodes, as opposed to updating the nodes one by one, each time already including the previously changed node states. To illustrate why this would lead to the failure of correlation-based methods, imagine a two-dimensional square grid graph with the node states arranged in a checkerboard pattern. With synchronous voter dynamics, all states would simply be flipped in each time step, because each node has the opposite state of all of its neighbors, which results in the probability one for changing its state. This patterns would repeat indefinitely. Since in each observation, each node has the opposite state of its direct neighbors, correlation-based methods will yield negative correlations between direct neighbors, and positive correlations between two-hop neighbors (and also 4-hop neighbors, 6-hop neighbors and so on). Since the largest entries are chosen as edges, these methods would return completely incorrect structures in this example. Of course, we use Barabási-Albert graphs in our experiments, so this extreme case does not occur in our dataset, but the general problem of negative correlations between neighbors appears also on Barabási-Albert graphs. The author has verified this theory by simulating dynamics with asynchronous updates, in which case the statistical methods work much better.

On SIS dynamics, a different picture presents itself: The statistical methods work much better on these datasets. The reason is that the SIS model generally produces linear dependencies between nodes, in the sense that the more infections are present in a node's neighborhood, the more likely the node is to be infected itself. Partial correlation works especially well, recovering the ground truth

Method	Voter-BA20-100				Voter-BA20-1k			
	⌀ACC	⌀TPR	⌀FPR	MAX ACC	⌀ACC	⌀TPR	⌀FPR	MAX ACC
Corr.	0.6737	0.1389	0.2013	0.6737	0.6737	0.1389	0.2013	0.6737
Partial Corr.	0.6842	0.1666	0.1948	0.6842	0.6737	0.1389	0.2013	0.6737
GGN	0.9184	0.7278	0.0370	0.9289	0.9974	0.9861	**0.0**	0.9974
HC (Grad)	0.9200	0.9944	0.0974	0.9316	0.9895	0.9833	0.0091	0.9947
HC (Eval)	**0.9958**	0.9944	0.0039	**1.0**	**1.0**	**1.0**	**0.0**	**1.0**
SA (Grad)	0.9084	0.9944	0.1117	0.9316	0.9926	0.9889	0.0065	**1.0**
SA (Eval)	0.9905	0.9889	0.0091	**1.0**	**1.0**	**1.0**	**0.0**	**1.0**
PBLS (Grad)	0.9442	0.9444	0.0675	0.9526	0.9989	**1.0**	0.0013	**1.0**
PBLS (Eval)	0.9926	**1.0**	0.0091	0.9947	**1.0**	**1.0**	**0.0**	**1.0**
HC* (Grad)	0.9863	**1.0**	0.0169	**1.0**	0.9989	**1.0**	0.0013	**1.0**

Table 5.1: Comparison of different reconstruction methods on voter dynamics. The mean (⌀) and maximum are taken over 5 independent executions.

Method	SIS-BA10-1k				SIS-BA10-5k			
	⌀ACC	⌀TPR	⌀FPR	MAX ACC	⌀ACC	⌀TPR	⌀FPR	MAX ACC
Corr.	0.7333	0.6250	0.2069	0.7333	0.8667	0.8125	0.1034	0.8667
Partial Corr.	0.9111	0.875	0.0690	0.9111	**1.0**	**1.0**	**0.0**	**1.0**
GGN	0.6467	0.7500	0.4103	0.7333	0.8822	0.7625	0.0517	0.9556
HC (Grad)	0.6533	0.7125	0.3793	0.9333	0.8222	0.8625	0.2000	0.9556
HC (Eval)	0.5867	0.6500	0.4483	0.8222	0.6133	0.6625	0.4138	0.9778
SA (Grad)	0.7200	0.8125	0.3310	0.8667	0.7733	0.8874	0.2897	0.9333
SA (Eval)	0.6480	0.7250	0.3931	0.8222	0.7022	0.7000	0.2966	**1.0**
PBLS (Grad)	**0.9689**	**0.9750**	**0.0345**	**1.0**	**1.0**	**1.0**	**0.0**	**1.0**
PBLS (Eval)	0.7600	0.8500	0.2897	**1.0**	0.8311	0.8625	0.1862	**1.0**
HC* (Grad)	0.7822	0.7375	0.1931	**1.0**	0.9955	**1.0**	0.0069	**1.0**

Table 5.2: Comparison of different reconstruction methods on SIS dynamics. The mean (⌀) and maximum are taken over 5 independent executions.

Method	CM-3.5-BA20-1k				CM-4.0-BA20-1k			
	⌀ACC	⌀TPR	⌀FPR	MAX ACC	⌀ACC	⌀TPR	⌀FPR	MAX ACC
Corr.	0.7053	0.2222	0.1818	0.7053	0.7053	0.2222	0.1818	0.7053
Part. Corr.	0.6737	0.1389	0.2013	0.6737	0.6737	0.1389	0.2013	0.6737
GGN	0.3516	**1.0**	0.8000	**1.0**	0.1916	**1.0**	0.9974	0.1921
HC (Grad)	0.8842	0.9944	0.1416	0.9368	0.4210	0.8167	0.6714	0.4737
HC (Eval)	0.7031	0.9111	0.3455	**1.0**	0.3789	0.9055	0.7441	0.4632
SA (Grad)	0.8853	**1.0**	0.1416	0.9421	0.4284	0.7278	0.6416	0.5368
SA (Eval)	0.5947	0.8778	0.4714	0.6737	0.2895	0.9278	0.8597	0.3526
PBLS (Grad)	0.9926	0.9889	0.0065	**1.0**	0.9958	0.9944	0.0039	**1.0**
PBLS (Eval)	**1.0**	**1.0**	**0.0**	**1.0**	**1.0**	**1.0**	**0.0**	**1.0**
HC* (Grad)	**1.0**	**1.0**	**0.0**	**1.0**	0.2063	0.9833	0.9753	0.2526

Table 5.3: Comparison of different reconstruction methods on coupled map dynamics. The mean (⌀) and maximum are taken over 5 independent executions.

matrix with 5000 samples, which is in line with previous research on its capability of filtering out spurious network effects [15, 45].

On coupled map dynamics, like on voter dynamics, correlation-based methods fail completely at reconstructing the network. Here, this should not be too surprising because of the highly non-linear nature of the logistic map.

In summary, correlation-based methods can work well when the dynamical model produces clear linear dependencies between neighboring nodes. Out of the three dynamical models we discuss, SIS is the only one where this is the case. Now, are our combinatorial prediction-based methods more universally applicable? In general, the answer is yes — you can see that for all datasets, there are one or more prediction-based methods with a strong mean accuracy of more than 96%. However, the details are important, as there are considerable differences between the performance of the different search methods and heuristics. In the following, we discuss some of those differences.

Gradient vs. evaluation heuristics

In this subsection, we will not consider or discuss the variation of the hill climbing algorithm which is listed as "HC* (Grad)" because we have not introduced it yet.

On voter dynamics, the evaluation heuristic works better than the gradient heuristic across all search algorithms, but the difference is only sizable on the tiny 100-time step dataset; on the 1000-time step dataset, the gradient heuristic achieves ∼99% accuracy as well. The difference on the 100-time step dataset seems to be related to step size: The gradient heuristic is in general more "optimistic" than the evaluation heuristic (i.e. S^{grad} usually has more positive entries than S^{eval}), which results in the gradient attempting larger steps. In the later stages of execution, when the accuracy is already greater than 90%, we observe that the evaluation heuristic typically takes steps of size 1-3, whereas the gradient heuristic keeps evaluating neighbors with 15 mutations or more, which do not lead to improvements and thus get rejected. With more data, this effect seems to be less prominent: On the 1000-time step dataset, the gradient heuristic usually attempts steps of size 5-10, which more often lead to improvements (even though they still get rejected more often than the candidates suggested by the evaluation heuristic).

On SIS and coupled map data, in contrast, we can see that the gradient heuristic performs better than the evaluation heuristic across all search methods (except PBLS on CM data, where they work equally well). Like in the CM experiment in Figure 5.3, the problem we observe with the evaluation heuristic is that during the search, the algorithm often comes to points where all of the mutation scores (= entries of S^{eval}) are negative. At this point, the mutation probabilities for all entries of the adjacency matrix are zero, so that the current candidate itself is evaluated again instead of one of its neighbors. For hill climbing and simulated annealing, this means that the search algorithm gets stuck. For PBLS, it depends on the other individuals in the population — if all of them have this problem, PBLS gets stuck as well, otherwise it still has a chance to escape. The author has verified that in the runs where PBLS (Eval) gets stuck on SIS dynamics, it is indeed the case that all of the individuals in the population have all-negative mutation score matrices S^{eval}.

It is also worth looking at the run time of the two heuristics in combination with hill climbing and population-based search.
In this table, the difference in run time between HC and PBLS is due to the additional candidate evaluations and calculations of the heuristics needed for PBLS. The reasons for the longer run time on the SIS dataset are that we use a much larger number of epochs to train the dynamics learner when

Run time	Voter-BA20-1k	SIS-BA10-1k
HC (Grad)	2min 21s	14min 41s
HC (Eval)	3min 38s	17min 26s
PBLS (Grad)	29min 31s	5h 8min
PBLS (Eval)	64min 24s	5h 19min

Table 5.4: Average run time of HC and PBLS on two of our datasets.

evaluating the candidates (200 compared to 30) as well as the larger population size used for PBLS (see Appendix A.2). What is more interesting, though, is that the evaluation heuristic has a longer run time than the gradient heuristic in all cases. This is due to the fact that the evaluation heuristic uses an evaluation epoch for each possible mutation of the candidate, which are $\frac{1}{2}(N-1)N$ mutations in total, whereas the gradient heuristic can be calculated in a single backward pass. Naturally, this quadratic amount of evaluations drives up the run time especially on the 20-node graph used for the Voter-BA20-1k dataset, where PBLS (Eval) takes more than twice as long as PBLS (Grad), and this would be even more problematic on larger graphs. The scalability of the two heuristics (as well as our methods in general) is further discussed from a more theoretical perspective in Section 6.1.

To conclude, because of the better run time and the fact that the gradient heuristic works on all dynamical models, whereas the evaluation heuristic gets stuck on SIS data due to negative mutation scores, the gradient heuristic is preferable to the evaluation heuristic overall.

Comparison of search methods

We now discuss the effectiveness of the different search methods (HC, SA and PBLS).

On voter dynamics, we can see that all of our combinatorial methods work very well when combined with the evaluation heuristic, outperforming all baselines. We can see that HC (Eval) has an accuracy of 99.58% / 100% on average, which means that there is a direct path from each of the randomly initialized matrices to a very small region around the ground truth matrix where each step on the path improves the prediction loss (here, we mean a path with varying step size). Thus, it is not needed to cross any suboptimal regions of the search space to reach the ground truth network. Therefore, the added exploration capabilities of SA and PBLS do not constitute a significant advantage on these datasets.

In contrast, most of the prediction-based approaches seem to have major difficulties with SIS dynamics. Hill climbing and simulated annealing mostly get stuck in areas of the search space that are far from optimal, although they *can* be successful in some individual executions (see maximum accuracy on SIS-BA10-5k). In general, this is in line with our brute force experiments where SIS dynamics had a lot of local minima in the loss surface. We also observe that population-based search can deal with these local minima much better than the other search methods, so that this method combined with the gradient heuristic has the best average performance on both datasets (tied with partial correlation on SIS-BA10-5k).

On coupled map data, most of the prediction-based methods fail or are inconsistent. For example, Zhang et al.'s GGN converges to a fully connected matrix 4 out of 5 times and to the ground truth matrix one time on the non-chaotic dataset ($r = 3.5$), and on the chaotic dataset ($r = 4$) it always converges near the fully connected matrix. Similar instabilities also occur with hill climbing and simulated annealing. Population-based local search is again the only search method that reliably recovers the true matrix (or a close approximation), and it does so irrespective of which heuristic is

used.

At this point, the reader may have noticed that simulated annealing did not produce any significant improvements over hill climbing in any of our experiments. This is because the main purpose of simulated annealing is to enable more exploration of the search space at the start of the execution, but our heuristics are made for exploitation and not exploration: Since the gradient and evaluation criteria aim at recommending mutations that directly reduce the loss, they will always (at least try to) guide the algorithm directly towards the ground truth matrix. Allowing worse candidates to be accepted is therefore not a sufficient technique to enable more exploration.

Node-level loss evaluation

Motivated by the observation that the gradient heuristic often produces candidates with too many mutations, we implemented an additional variant of the hill climbing method which attempts to solve this problem. In this variant, listed in the tables as HC* (Grad), the average loss is not just evaluated as the mean over all nodes, but rather on the level of individual nodes.[2] This node-level loss information enables us to take more targeted steps: After evaluating a neighbor, we can calculate for each mutation individually whether it improved the loss of the nodes that it affects. We do this as follows: To decide whether to accept the symmetric mutation of the indices (i, j) and (j, i) or not, we calculate the sum of the losses for node i and j, once for the current candidate and once for the neighbor. If this sum is lower for the neighbor, we accept the mutation. If it is lower for the current candidate, we reject the mutation, i.e. the entries (i, j) and (j, i) remain as they are.

Ideally, this should lead to the algorithm accepting only those individual mutations which improve the network structure. In other words, it should alleviate the problem of "over-stepping": If a neighbor has 10 "bad" and 5 "good" mutations, the mean loss of the neighbor is probably worse than that of the original matrix, so that it would get rejected with the graph-level loss that we used before. With node-level loss evaluation, we should be able to filter out the five "good" mutations and accept them. Of course, this can only work to a degree: If there are so many "bad" mutations that the dynamics learner cannot properly learn the dynamical model anymore, then the loss will be worse for all nodes, including those with "good" mutations.

In our experiments, we find that using node-level loss does indeed have a positive effect on the performance of the hill climbing algorithm, for example improving from 82,22% to 99,55% mean accuracy on the SIS-BA10-5k dataset. This confirms that the dynamic step size is often chosen too large when using the gradient heuristic and that the node-level losses can indicate which specific mutations the algorithm should accept. Note also that HC* (Grad) now outperforms the competing GGN method on all datasets. However, we find that the reconstruction performance is still inconsistent on the SIS-BA10-1k dataset. Also, on the CM-4.0-BA20-1k dataset, HC* (Grad) converges near a fully connected network in all executions (similar to GGN). Thus, PBLS (Grad) remains our most successful reconstruction method overall, even though it naturally has a much higher run time than hill climbing (roughly 20 times higher with our hyperparameter settings).

[2] In the notation introduced in Section 3.1.1, this means removing the vector $\mathbf{1}_N^\top$ and the factor $\frac{1}{N}$ from the empirical risk calculation, so that the result is an N-dimensional vector.

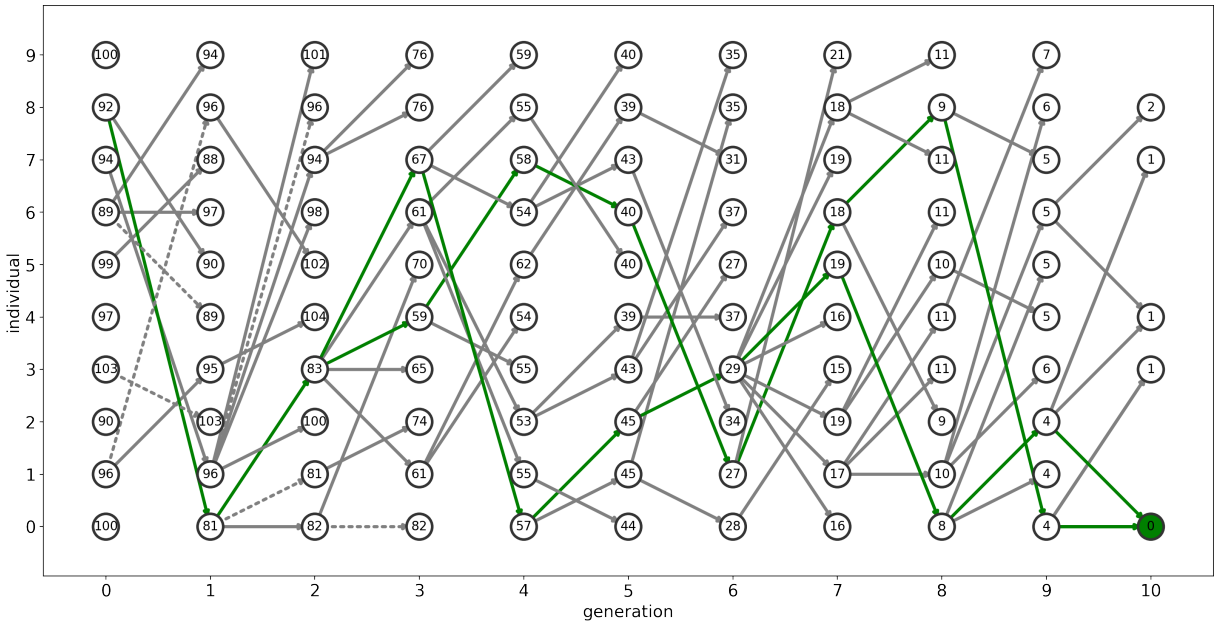

Figure 5.4: Visualization of PBLS (Eval) on the CM-4.0-BA20-1k dataset. Each circle represents an adjacency matrix, and the numbers specify the distance to the ground truth network. The individuals in each generation are sorted by loss (0: lowest loss, 9: highest loss). Arrows indicate the parent of an individual (solid: mutated; dotted: non-mutated, i.e. the parent has been accepted into the next generation without changes). The green arrows show the path(s) to the ground truth matrix.

Final remarks

To sum up, we have seen that two of our combinatorial methods, HC* (Grad) and PBLS (Grad), outperform Zhang et al.'s indirect optimization approach (GGN) on all datasets and that these methods are also more broadly applicable than correlation-based methods. PBLS (Grad) is the more robust one of the two, at the cost of a higher run time. Of course, all datasets used in this chapter were generated on fairly small graphs, which raises questions about the scalability of such methods. We discuss this topic is Chapter 6.

5.4 PBLS visualization

In the previous section, we have seen that prediction-based methods have the capability to reconstruct networks when correlation-based methods fail due to complex interactions between nodes. However, simple hill climbing algorithms and gradient-based continuous optimization techniques (GGN) can get stuck in local minima on the way to the global minimum. Population-based search seems to be able to deal with this problem much better based on our results. The following visualizations and discussion should give us some idea about why this is the case.

Figure 5.4 shows one run of PBLS (Eval) on the CM-4.0-BA20-1k dataset, where all methods except for PBLS fail. We display each individual as a circle labeled with its distance to the target matrix and use an arrow to indicate the parent of the individual. Note that when an individual has no children in this plot, it does not necessarily mean that this individual is a dead end (meaning that there is no path from this individual to the target); instead, it simply means that none of its descendants (of which there are three on average) have achieved a top-10 loss in the next generation (including the matrix itself, since we also reevaluate the existing population (see Section 4.4.3)) and

thus none of the descendants have survived the selection operator.

In the following, we use the notation (x,y) to refer to individual y from generation x. We observe several things:

- In general, the average distance to the target decreases with every generation, confirming that the underlying assumption of prediction-based network reconstruction holds even on this dataset where the simpler optimization methods fail.
- In the first generations, there are multiple candidates that move *away* from the target, for example the path from individual (0,7) (94 → 96 → 100) or the paths from individual (0,1) (96 → 95 → 104, 96 → 96 → 102). This partly explains the observation that HC and SA also tend to move away from the target (towards a fully connected matrix) on this dataset.
- In generations 3 and 4, the descendants of the individual (0,8) take over the whole population. At the same time, the paths that lead away from the ground truth matrix drop out of the population. As a consequence, the average distance to the target in the population decreases drastically (gen. 2: 94.1 — gen. 3: 69.1 — gen. 4: 56.2). The PBLS has successfully avoided the paths that lead to its demise (that is, a fully connected matrix).

At this point, critics may argue that if all PBLS does is help select the most promising candidate out of the 10 randomly initialized matrices, then the same thing could also be achieved by running a single trajectory algorithm multiple times and choosing the best result over all executions (i.e. the final matrix with the lowest loss). Thus, we performed some additional experiments to demonstrate that PBLS has advantages that go beyond having 10 matrices in its initial population:

We ran hill climbing and simulated annealing (5 runs each), starting from different individuals on the paths to the ground truth matrix in Figure 5.4. First, when starting from individual (0,8) (distance 92), we observe that both single candidate algorithms fail in all runs, that is, they move away from the ground truth matrix by accepting mutations with many false positives. The same thing also happens when starting from individual (2,3) (distance 83). When starting with individual (5,2) (distance 45), hill climbing failed 5 out of 5 times, and simulated annealing failed 4 out of 5 times and converged close to the target 1 out of 5 times. When starting with individual (7,5) (distance 19), hill climbing succeeded 5 out of 5 times and simulated annealing succeeded 4 out of 5 times.

These failures of the single-candidate algorithms from the earlier starting points demonstrate that the benefit of PBLS lies not just in picking the "right" one of the 10 randomly initialized matrices to explore. Rather, we can see that there exist many paths that lead away from the target even in later stages of execution, which the PBLS manages to avoid. Unlike the single-candidate methods, the population-based search algorithm does not immediately "commit" to a step as soon as it finds a neighbor with a lower loss. Instead, it always evaluates all newly generated candidates (3 per parent on average) before selection, giving it a better picture of its surroundings. Also, it has the capability to keep multiple options open, which is clearly important — the matrices which lead to the target are usually not the ones which perform best in their respective generations (since the individuals are sorted by loss, the matrix with the lowest loss is always shown as individual 0 in the bottom row). The author considers these two factors to be the main reason why PBLS is more robust than single-candidate methods.

More visualizations can be found in Appendix B.

Chapter 6

Discussion

In the first section of this chapter, we discuss the scalability of our combinatorial prediction-based methods. In the second section, we then mention a few ideas to improve scalability, as well as other opportunities for future research.

6.1 Scalability

Until now, we have only evaluated our methods on small networks with 10 or 20 nodes. Of course, there are many real-world networks with hundreds or thousands of nodes. Can our combinatorial prediction-based methods work at these scales? There are two aspects to this question, one is the computational complexity (time and memory) and the second is reconstruction accuracy.

Let us start with complexity. In general, the time and space complexity of MPGCNs grows linearly with the number of edges in the network, $O(E)$, due to the *locality* of the message-passing step [46]: Each node's hidden representation only depends on its neighbors, which means that the total number of messages to be computed is $|E|$ (one for each edge). In addition, the message-passing step is highly parallelizable, which can be exploited to further improve the efficiency of implementations of MPGCNs.

Since the complexity of MPGCNs is $O(E)$, the connectivity of the network plays a significant role in practical applications: For fully connected networks, a complexity of $O(E)$ would be equivalent to $O(N^2)$. In contrast, just like many networks found in the real world, the Barabási-Albert graphs used in our work are *sparse*[1] [47]: In BA graphs, the number of edges scales linearly with the number of nodes, since for each node, a constant number of links is added to the network (see Section 5.1). Consequently, the time and memory complexity of an MPGCN grows linearly with network size, $O(N)$, on BA graphs.

However, all of the prediction-based NR methods presented so far actually have a complexity that is worse than $O(E)$. The reasons for this are as follows:

- Many of the methods presented so far rely on the gradient of the loss with respect to *all entries* of the adjacency matrix. This includes our gradient heuristic as well as the previous works by Kipf et al. [8] and Zhang et al. [9]. To compute the gradient with respect to non-existing links, it is first necessary to compute the messages between all non-connected nodes in addition to

[1] Informally speaking, this means that the number of edges present in the network is much smaller than the number of links in a fully connected network with an equal number of nodes.

the messages between connected nodes. These "non-edge messages" do not actually influence the result of the forward pass, since they are multiplied with the corresponding entry of the adjacency matrix (which is 0) afterwards; their only purpose is to enable gradient calculation with respect to non-edges, which can then be done automatically via backpropagation.[2] As a result, one needs quadratically many messages when calculating the gradient, which results in a time and space complexity of $O(N^2)$, irrespective of network connectivity. This complexity was also stated (but not explained) in Zhang et al.'s work.

- When we use the risk evaluation heuristic for neighborhood search, there is no need to calculate the gradient, and thus also no need to compute the non-edge messages. However, we need one evaluation epoch for each one-hop neighbor of the candidate, which are $\frac{1}{2}N(N-1)$ epochs in total. Thus, the time complexity of the evaluation phase is $O(EN^2)$, which is even worse than calculating the gradient.[3] The space complexity is $O(N^2)$, which is needed to store the resulting mutation score matrix S^{eval}.

Thus, gradient-based methods are better than explicit risk evaluations in terms of time complexity, but they still have quadratic complexity *per step or generation*. It should be pointed out, however, that when using our combinatorial methods, the quadratic complexity is only needed in the phase where the heuristic is actually calculated and not during the training phase of the dynamics learner. To update the weights of the dynamics learner, only the gradient with respect to these weights is needed; there is no need to compute the gradient with respect to the entries of the adjacency matrix. An advanced implementation could therefore train the dynamics learner without computing non-edge messages (complexity: $O(E)$) and only compute non-edge messages during the final evaluation epoch (complexity: $O(N^2)$) when using the gradient heuristic. For our experiments so far, we used a simpler implementation which always calculates all $O(N^2)$ messages. This also means the time complexity of the evaluation heuristic is $O(N^4)$ in our current implementation.

Now, what we have considered so far pertains to the cost *per step or generation*. In addition, the size of the search space grows with increasing network size. Thus, as we have discussed in Section 4.1, the number of steps required to reach the target may also grow. To reiterate our thoughts quickly, when using random initialization (as we did so far), the expected distance to the target is $O(N^2)$. Under the strong assumption that the algorithm performs only correct mutations, the number of steps needed to reach the target only depends on the step size. With a step size of one, the time complexity of the combinatorial methods with the gradient heuristic would be $O(N^4)$. With the evaluation heuristic, it would be $O(EN^4)$. Of course, we use dynamic steps, so the time complexity should be better than that, but it is hard to quantify this effect exactly. To give a rough idea, on the Voter-BA20-1k dataset, HC* (Grad) performs steps of size 15-35 in the beginning. On a Voter-BA100-1k dataset, HC* (Grad) performs steps of size 200-550 in the beginning. In both cases, the step size is reduced when the algorithm approaches the ground truth matrix.[4] While it is of course not possible to draw

[2] For ease of exposition, we did not show these calculations in the model description in Section 2.5. The interested reader is referred to [9], particularly the equations on page 6, which show the MPGCN in vectorized form. In equation (5) on page 6, the hidden representation is multiplied element-wise with the adjacency matrix, so that non-edge messages are multiplied with 0.

[3] Of course, the run time also depends on the size of the training set in practice, but we focus on the effect of network size here, so we do not include this in the big-O notation. Other neglected factors include the number of prediction steps and the number of training epochs for the dynamics learner.

[4] Step sizes for one particular execution on each dataset:
Voter-BA20-1k: 22, 27, 33, 25, 18, 5, 2, 1, 2, 0, 1, 0, 3, 3, 4, 0, 1, 1, 0, ...
Voter-BA100-1k: 258, 237, 216, 270, 557, 449, 358, 384, 277, 157, 133, 63, 37, 41, 26, 28, 5, 15, 12, 7, 17, 6, 1, ...

any definite conclusions from this experimental data, it can serve as an indicator that the dynamic step size that we use can indeed adjust to different network sizes. Nevertheless, in this regard, our combinatorial approach is inherently at a disadvantage in comparison to Zhang et al.'s method, since they optimize all parameters of the probability matrix P at the same time, whereas we can always only (sensibly) mutate a fraction of the matrix entries in each step.

Summing up what we have discussed so far, the time complexity of our combinatorial methods with the gradient heuristic is somewhere between $O(N^2)$ and $O(N^4)$. With the evaluation heuristic, it is somewhere between $O(EN^2)$ and $O(EN^4)$. Within this span, the actual complexity depends on how the dynamic step size scales with network size, which is hard to quantify theoretically. Remember that the considerations regarding the number of steps needed to reach the target only apply when making the strong assumption that the algorithm takes optimal steps, that is, all of the performed mutations bring the algorithm closer to the target.

It may be possible to reduce the number of steps required with better initialization under additional assumptions to the graph structure. For example, it is clear that when the graph is sparse, initializing with less edges reduces the expected distance to the target. We leave an investigation of the effect of different initialization strategies for future work. Still, the general disadvantage of not being able to optimize all entries of the matrix in parallel would remain. Therefore, the overall time complexity would likely remain worse than $O(N^2)$, but at this point, it is hard to make more specific statements.

In addition to the time and memory needed to evaluate a candidate, one must also consider that the growing size of the search space makes finding the target structure more difficult; the heuristics may suggest more and more suboptimal steps. More concretely, the size of the search space grows exponentially with the number of nodes ($|\mathscr{A}_N| = 2^{\frac{1}{2}(N-1)N}$), which increases the risk for local search methods to get stuck in suboptimal regions of the search space [48]. Because an in-depth study of the effect of search space size on reconstruction quality would go beyond the scope of this thesis, we only mention some rudimentary results that we obtained when experimenting with larger graphs: On a Voter-BA100-1k dataset, HC* (Grad) works without problems, achieving an average accuracy of 99.72% over 5 runs, which is notable because we did not need to increase the size of the dataset going from a 20-node graph to a 100-node graph and still achieved a similarly high accuracy as in our original experiment (Table 5.1). On datasets generated with more complex dynamical models, such as CM-4.0, network size seems to be a bigger problem: Both HC* (Grad) and PBLS (Grad) perform poorly on a CM-4.0-BA50-1k dataset when leaving all search hyperparameters unchanged compared to the experiment on the BA20 graph, with HC* (Grad) achieving a mean accuracy of 0.30 and PBLS (Grad) achieving accuracies of 0.67 and 0.81 in two runs[5]. More research is required to find out if such problems can be overcome with more data or by using a larger population size or more offspring in PBLS.

To sum up, both the time complexity as well as the reconstruction accuracy present problems when applying our combinatorial prediction-based network reconstruction approach to large graphs. One may say that we pay the improved performance on small networks with the lack of scalability, limiting the applicability of our approach to small to medium-sized networks.

[5] The author could not perform an evaluation over 5 runs here due to time constraints; one execution of PBLS on this dataset takes more than two days.

6.2 Outlook

Implementation

First off, the efficiency of our implementation could be improved. As mentioned before, it is possible to implement the training phase of the dynamics learner with $O(E)$ instead of $O(N^2)$ complexity. Also, one could take advantage of the natural potential for parallelization in population-based search: Instead of evaluating all individuals in the population and the offspring sequentially, one could parallelize these evaluations across multiple GPUs, given access to the required hardware.

With such an advanced implementation, it should become feasible to use larger population sizes and evaluate more offspring per generation. These factors have proven to be important in previous research on evolutionary computation [48, 42, 49]: As mentioned in [48], the quality of solutions generally increases with larger population sizes, and the required population size increases with the size of the search space. Therefore, using larger population sizes and generating more offspring could help to apply PBLS to larger networks with more complex dynamical models.

Pre-training the dynamics learner

The following realization might lead to a different approach for improved scalability: For the setting we discussed, the dynamical model for a single node is independent of network size; individual nodes behave the same no matter the size of the network. It therefore seems possible to train the dynamics learner on a small network and then use it for network reconstruction on a larger network with the same dynamical process operating on it. This means that the dynamics learner would remain fixed through the whole network reconstruction process, removing the need to retrain it before each evaluation of prediction quality. With this cheaper evaluation process, new search methods which are infeasible in the current setting could be applied. Drawbacks are that (1) this approach would require access to an instance of the same dynamical process on a smaller network, which may not be easy to obtain in practice, and (2) in practical applications, the assumption that individual nodes behave identically in small and large networks may not hold, so that the transferability of the learned model from small to large networks may be limited.

Neuroevolution

Another idea that might scale better would be to encode the parameters of the dynamics learner as part of the genetic representation; in other words, to include the weights of the GNN in the search space. With such an approach, the weights of the dynamics learner would be evolved jointly with the network structure, saving all of the run time that we currently use to train the dynamics learner afresh for each individual. Also, it would remove the need to specify the number of training epochs to be performed before evaluation. Such an approach could be classified as belonging to the field of *neuroevolution*, which concerns itself with finding optimal architectures and parameters of neural networks using evolutionary methods [50, 51]. While these methods radically differ from traditional deep learning approaches which rely on gradient descent to find optimal parameters, they have recently raised attention especially in the field of reinforcement learning [50]. In [52], a deep neural network with over 4 million parameters is trained using a genetic algorithm, showing that neuroevolutionary methods can be competitive with, or in some cases better than, gradient-based optimization techniques in optimizing large-scale deep learning models in terms of run time and performance.

Modifying the dynamics predictor

Another opportunity for future research would be to relax the assumptions about the dynamical model that are dictated by the architecture of the dynamics predictor (see also Section 2.5). For example, one could use recurrent units in the graph neural network to learn dynamics that are not Markovian in the time domain, similar to Kipf et al. [8]. Furthermore, one could relax the assumption that the dynamical model is identical for all nodes in the network, as was done in Zhang et al.'s follow-up paper [53], where the authors use node-specific GNN weights instead of sharing the weights between all nodes in order to learn heterogeneous gene regulatory network dynamics.

To make more fundamental changes, one could also replace the dynamics learner with an entirely different module, for example one that uses a different task as a proxy to evaluate the quality of a candidate network: Instead of predicting the dynamics one time step into the future, one could measure how well a neural network can predict a node's state given the states of its neighbors in the same time step. With this approach, one would no longer need contiguous observations of the dynamics and could instead use snapshots of the dynamics taken at arbitrary points in time. Such an approach would be applicable to practical settings where observing the dynamic with a sufficiently high time resolution to treat the samples as contiguous is not possible.

Chapter 7

Summary

In this work, we have addressed the problem of inferring the hidden structure of a network from observations of a dynamical process operating on the network, without knowledge of the rules that govern the dynamical process. We have built upon recent work on what we have called prediction-based network reconstruction, where the quality of predictions about future states of the dynamical process made with a certain candidate structure is used as a measure of the accuracy of the structure itself.

We have analyzed the optimization landscapes given by the prediction losses of a graph neural network trained on each individual in the search space of small problem instances, finding that the ground truth network does generally indeed have the best prediction performance out of all candidates in the search space. We have also identified differences between the loss surfaces generated by datasets with different dynamical models and found that the amount of data and the training budget for the graph neural network play an important role in determining whether the resulting loss surface is well-shaped or riddled with local minima.

Tackling the NR problem from a combinatorial optimization perspective, we have introduced two heuristics for fast neighborhood search (based on gradients and finite differences), which have proven effective at speeding up the search for the global optimum of the prediction loss compared to random search, especially when using dynamic step sizes determined by the heuristic itself. We have combined these heuristics with different local search methods (hill climbing, simulated annealing, population-based search) and evaluated them on six datasets with binary and continuous dynamics on 10-node and 20-node graphs. In these experiments, we found that prediction-based methods can successfully reconstruct networks in cases where correlation-based methods fail due to the highly non-linear interactions produced by some dynamical models. Two of our methods, hill climbing with node-level loss evaluations and population-based search combined with the gradient heuristic, outperformed Zhang et al.'s original GGN method on all six datasets. We have discussed scalability issues related to our approach such as the time complexity of the heuristics and the deteriorating reconstruction accuracy due to the exponential size of the search space on larger networks, which limit the applicability of the methods presented in this thesis. Lastly, we have identified multiple avenues for future research, which might improve the scalability and other aspects of our methods.

Bibliography

[1] Javier Orlandi et al. “First Connectomics Challenge: From Imaging to Connectivity”. In: *Neural Connectomics Challenge*. Ed. by Demian Battaglia et al. Springer International Publishing, 2017, pp. 1–22. DOI: 10.1007/978-3-319-53070-3_1.

[2] Niklas Boers et al. “Complex networks reveal global pattern of extreme-rainfall teleconnections”. In: *Nature* 566 (2019). DOI: 10.1038/s41586-018-0872-x.

[3] Vân Anh Huynh-Thu and Guido Sanguinetti. *Gene regulatory network inference: an introductory survey*. 2018. arXiv: 1801.04087 [q-bio.QM].

[4] Richard Stein et al. “Ecological Modeling from Time-Series Inference: Insight into Dynamics and Stability of Intestinal Microbiota”. In: *PLoS Computational Biology* 9, e1003388 (2013). DOI: 10.1371/journal.pcbi.1003388.

[5] Michael Kohanski, Daniel Dwyer, and James Collins. “How Antibiotics Kill Bacteria: From Targets to Networks”. In: *Nature Reviews. Microbiology* 8 (2010), pp. 423–435. DOI: 10.1038/nrmicro2333.

[6] John Tyson, Kathy Chen, and Bela Novak. “Network dynamics and cell physiology”. In: *Nature Reviews. Molecular Cell Biology* 2 (2002), pp. 908–916. DOI: 10.1038/35103078.

[7] S. Arianos et al. “Power grid vulnerability: A complex network approach”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 19.1, 013119 (2009). ISSN: 1089-7682. DOI: 10.1063/1.3077229.

[8] Thomas Kipf et al. *Neural Relational Inference for Interacting Systems*. 2018. arXiv: 1802.04687 [stat.ML].

[9] Zhang Zhang et al. “A general deep learning framework for network reconstruction and dynamics learning”. In: *Applied Network Science* 4.1 (2019). ISSN: 2364-8228. DOI: 10.1007/s41109-019-0194-4.

[10] Ildefons Magrans de Abril, Junichiro Yoshimoto, and Kenji Doya. “Connectivity inference from neural recording data: Challenges, mathematical bases and research directions”. In: *Neural Networks* 102 (2018), pp. 120–137. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.02.016.

[11] Bastian Prasse and Piet Van Mieghem. *Maximum-Likelihood Network Reconstruction for SIS Processes is NP-Hard*. 2018. arXiv: 1807.08630 [cs.CC].

[12] Bastian Prasse and Piet Van Mieghem. “Exact Network Reconstruction from Complete SIS Nodal State Infection Information Seems Infeasible”. In: *IEEE Transactions on Network Science and Engineering* 6.4 (2019), pp. 748–759. DOI: 10.1109/TNSE.2018.2872511.

[13] Clive W. J. Granger. “Investigating Causal Relations by Econometric Models and Cross-spectral Methods”. In: *Econometrica* 37.3 (1969), pp. 424–438. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1912791.

[14] Thomas Schreiber. “Measuring Information Transfer”. In: *Physical Review Letters* 85.2 (2000), pp. 461–464. DOI: 10.1103/PhysRevLett.85.461.

[15] Antonio Sutera et al. *Simple connectome inference from partial correlation statistics in calcium imaging*. 2014. arXiv: 1406.7865 [stat.ML].

[16] Lukasz Romaszko. “Signal Correlation Prediction Using Convolutional Neural Networks”. In: *Proceedings of the Neural Connectomics Workshop at ECML 2014*. Ed. by Demian Battaglia et al. Vol. 46. Proceedings of Machine Learning Research, 2015, pp. 45–56. URL: http://proceedings.mlr.press/v46/romaszko15.html.

[17] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML].

[18] Zonghan Wu et al. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. ISSN: 2162-2388. DOI: 10.1109/tnnls.2020.2978386.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[20] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[21] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314. DOI: 10.1007/BF02551274.

[22] Thomas M. Liggett. “Voter Models”. In: *Stochastic Interacting Systems: Contact, Voter and Exclusion Processes*. Springer Berlin Heidelberg, 1999, pp. 139–208. ISBN: 978-3-662-03990-8. DOI: 10.1007/978-3-662-03990-8_3.

[23] V. Sood and S. Redner. “Voter Model on Heterogeneous Graphs”. In: *Physical Review Letters* 94.17 (2005). ISSN: 1079-7114. DOI: 10.1103/physrevlett.94.178701.

[24] Mehmet Yildiz et al. “Voting models in random networks”. In: *2010 Information Theory and Applications Workshop (ITA)*. 2010, pp. 1–7. DOI: 10.1109/ITA.2010.5454090.

[25] Roni Parshani, Shai Carmi, and Shlomo Havlin. “Epidemic Threshold for the Susceptible-Infectious-Susceptible Model on Random Networks”. In: *Physical Review Letters* 104.25 (2010). ISSN: 1079-7114. DOI: 10.1103/physrevlett.104.258701.

[26] Tatsuo Yanagita. “Phenomenology of boiling: A coupled map lattice model”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 2.3 (1992), pp. 343–350. DOI: 10.1063/1.165877.

[27] Kunihiko Kaneko. “Overview of coupled map lattices”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 2.3 (1992), pp. 279–282. DOI: 10.1063/1.165869.

[28] Robert M. May. “Simple mathematical models with very complicated dynamics”. In: *Nature* 261.5560 (1976), pp. 459–467. ISSN: 1476-4687. DOI: 10.1038/261459a0.

[29] Robert M. May and George F. Oster. "Period doubling and the onset of turbulence: An analytic estimate of the Feigenbaum ratio". In: *Physics Letters A* 78.1 (1980), pp. 1–3. ISSN: 0375-9601. DOI: 10.1016/0375-9601(80)90788-4.

[30] Geoff Boeing. "Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction". In: *Systems* 4 (2016), p. 37. DOI: 10.3390/systems4040037.

[31] *Creative Commons Attribution 4.0 International (CC BY 4.0)*. URL: https://creativecommons.org/licenses/by/4.0/ (visited on 04/01/2021).

[32] Sewall Wright. "The roles of mutation, inbreeding, crossbreeding, and selection in evolution". In: *Proceedings of the sixth international congress of Genetics* 1 (1932). URL: https://www.blackwellpublishing.com/ridley/classictexts/wright.pdf.

[33] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[34] Edward Weinberger. "Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference". In: *Biological Cybernetics* 63 (1990), pp. 325–. DOI: 10.1007/BF00202749.

[35] Vesselin K. Vassilev, Terence C. Fogarty, and Julian F. Miller. "Information Characteristics and the Structure of Landscapes". In: *Evolutionary Computation* 8.1 (2000), pp. 31–60. DOI: 10.1162/106365600568095.

[36] Pierre Hansen and Nenad Mladenović. "First vs. best improvement: An empirical study". In: *Discrete Applied Mathematics* 154.5 (2006). IV ALIO/EURO Workshop on Applied Combinatorial Optimization, pp. 802–817. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2005.05.020.

[37] Gabriela Ochoa, Sebastien Verel, and Marco Tomassini. "First-Improvement vs. Best-Improvement Local Optima Networks of NK Landscapes". In: *Parallel Problem Solving from Nature, PPSN XI*. Vol. 6238. Springer Berlin Heidelberg, 2010, pp. 104–113. ISBN: 978-3-642-15843-8. DOI: 10.1007/978-3-642-15844-5_11.

[38] Dimitris Bertsimas and John Tsitsiklis. "Simulated Annealing". In: *Statistical Science* 8 (1993). DOI: 10.1214/ss/1177011077.

[39] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. "Optimization by Simulated Annealing". In: *Science (New York, N.Y.)* 220 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[40] John H. Holland. "Genetic Algorithms". In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939139.

[41] Thomas Bäck and Hans-Paul Schwefel. "Evolutionary Computation: An Overview". In: (1999), pp. 20–29. DOI: 10.1109/ICEC.1996.542329.

[42] Gotshall Stanley and Bart Rylander. "Optimal population size and the genetic algorithm". In: *Population* 100.400 (2002), p. 900. URL: http://www.wseas.us/e-library/conferences/mexico2002/papers/215.pdf.

[43] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Reviews of Modern Physics* 74 (1 2002), pp. 47–97. DOI: 10.1103/RevModPhys.74.47.

[44] Albert-Laszlo Barabasi. "Scale-Free Networks: A Decade and Beyond". In: *Science (New York, N.Y.)* 325 (2009), pp. 412–413. DOI: 10.1126/science.1173299.

[45] Alberto de la Fuente et al. “Discovery of Meaningful Associations in Genomic Data Using Partial Correlation Coefficients”. In: *Bioinformatics (Oxford, England)* 20 (2005), pp. 3565–3574. DOI: 10.1093/bioinformatics/bth445.

[46] Vijay Prakash Dwivedi et al. *Benchmarking Graph Neural Networks*. 2020. arXiv: 2003.00982 [cs.LG].

[47] Charo Genio, Thilo Gross, and Kevin Bassler. “All Scale-Free Networks Are Sparse”. In: *Physical Review Letters* 107.178701 (2011). DOI: 10.1103/PhysRevLett.107.178701.

[48] Ruhul Sarker and M.F.A. Kazi. “Population size, search space and quality of solution: An experimental study”. In: *CEC 2003 - Proceedings* 3 (2004), pp. 2011–2018. DOI: 10.1109/CEC.2003.1299920.

[49] Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki. “Influence of the population size on the genetic algorithm performance in case of cultivation process modelling”. In: *2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013* (2013), pp. 371–376. URL: https://ieeexplore.ieee.org/document/6644027.

[50] Kenneth Stanley et al. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1 (2019). DOI: 10.1038/s42256-018-0006-z.

[51] Risto Miikkulainen et al. *Evolving Deep Neural Networks*. 2017. arXiv: 1703.00548 [cs.NE].

[52] Felipe Petroski Such et al. *Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning*. 2018. arXiv: 1712.06567 [cs.NE].

[53] Yan Zhang et al. *Automated Discovery of Interactions and Dynamics for Large Networked Dynamical Systems*. 2021. arXiv: 2101.00179 [physics.soc-ph].

[54] URL: https://github.com/julianzimmerlin/network-reconstruction (visited on 04/01/2021).

[55] URL: https://github.com/bnusss/GGN/ (visited on 04/01/2021).

Appendices

Appendix A

Implementation & Experimental Details

A.1 Implementation

All of the code for this thesis was written in Python and can be found on GitHub [54]. We use the PyTorch deep learning library for automatic differentiation.

The code for the model definition of the graph neural network for dynamics prediction, which was described in Section 2.5, is shared among all search algorithms and is mostly identical to the one used by Zhang et al. [9], which is also on GitHub [55].

To simulate coupled map dynamics, we use a modified version Zhang et al.'s CML data generation script, which uses the NetworkX python package. For voter and SIS dynamics, we wrote our own simulation scripts using NetworkX and NumPy in Jupyter notebooks.

To compute the (partial) correlation baselines for network reconstruction, we use the netrd python package.

For the GGN baseline in our experiments, we do not use Zhang et al.'s original implementation, but a slightly modified one. Differences are: (1) We do not separate the dataset into training, validation and test sets (we also do not do this for any of our own methods). (2) We integrate our GGN version into the functionalities for loading datasets and tracking experiments which we also use for the other methods. Thus, we can for example be sure that the calculation of the network accuracy is done in exactly the same way for all methods. (3) There are a number of technical differences; e.g. our implementations can handle different input formats and mostly uses the same code for binary and continuous dynamics (in Zhang et al.'s code, there are basically separate implementations for binary and continuous dynamics).

For the plots in this thesis, we use Matplotlib and NetworkX' built-in graph visualization functionalities.

A.2 Experimental Details

We execute our experiments on a single NVIDIA Tesla P100 GPU (16GB memory). In the following, we will give the values of the hyperparameters for all experiments in Chapter 5.

First, a general thought on the "training budget" parameter NUM_DYN_EPOCHS, which corresponds to the number of training epochs performed when evaluating a candidate matrix. As men-

tioned in Section 3.2.2, because we keep the batch size constant, the number of weight updates per epoch during the training phase of the dynamics learner varies with the size of the dataset. For this reason, we will generally scale this parameter antiproportionally when running experiments with datasets of different sizes. Of course, the number of required epochs also depends on the complexity of the dynamical model; we observe that voter dynamics can be learned significantly faster than SIS and coupled map dynamics.

In all experiments, we use a batch size of 100 during the dynamics learner training phase. The learning rate for the Adam optimizer is set to 0.0001 for continuous dynamics and 0.001 for discrete dynamics. We use a size of 128 for the hidden layers in the dynamics learner (i.e. the messages and hidden node representations, see Section 2.5). For the final layer f_{output} we use a single linear layer. We do not use skip connections in any of our experiments.

Heuristics and step size

Experiments on Voter-BA20-5k (Figure 5.2):
NUM_DYN_EPOCHS = 40
NUM_STEPS = 200 for step size 1, 50 for dynamic step size

Experiments on CML-3.5-BA10-1k (Figure 5.3):
NUM_DYN_EPOCHS = 250
NUM_STEPS = 100 for step size 1, 50 for dynamic step size

Comparison of reconstruction methods

All statistics given in the tables are recorded at the final step/generation of the respective algorithm.
Experiments on voter dynamics (Table 5.1):
GGN:

- NUM_CYCLES = 100 (What we call a "cycle" is called an epoch in [9]. But we use the term epoch for a complete pass over the training data, so we call it a cycle instead. A cycle contains S_d epochs in which the dynamics learner is trained and S_n epochs in which the network generator is trained. We also tried 200 cycles but found no further improvement.)
- $S_d = 10$, $S_n = 20$ (These numbers identical to what Zhang et al. used for binary dynamics in [9]. We also experiment with different values for S_n and S_d, but we found no significant influence on reconstruction results.)
- INITIAL_GUMBEL_TEMPERATURE = 10
- TEMP_DROP_FACTOR = 0.95 (The temperature for the Gumbel softmax sampling is multiplied with this factor after every cycle to gradually make samples "more discrete". We also tried different values (0.9, 0.99) but found only minor influence on performance.)

Hill climbing:

- NUM_DYN_EPOCHS = 300 for Voter-BA20-100
  NUM_DYN_EPOCHS = 30 for Voter-BA20-1k
- NUM_STEPS = 45 (The number of dynamic (!) steps performed.)

Simulated annealing:

- NUM_DYN_EPOCHS = 300 for Voter-BA20-100
  NUM_DYN_EPOCHS = 30 for Voter-BA20-1k

- NUM_STEPS = 45 (30 steps while gradually decreasing the temperature + 15 steps with temperature 0.)

Population-based local search:

- NUM_DYN_EPOCHS = 300 for Voter-BA20-100
  NUM_DYN_EPOCHS = 30 for Voter-BA20-1k
- POP_SIZE = 8
- NEW_POP_SIZE = 16
- NUM_GENERATIONS = 45

Experiments on SIS dynamics (Table 5.2):

GGN:

- Same as for voter dynamics. Again, we also tried more training cycles, but this did not lead to improvements.

Hill climbing, simulated annealing:

- NUM_DYN_EPOCHS = 200 for SIS-ba10-1k
  NUM_DYN_EPOCHS = 40 for SIS-ba10-5k
- NUM_STEPS = 45

Population-based local search:

- NUM_DYN_EPOCHS = 200 for SIS-ba10-1k
  NUM_DYN_EPOCHS = 40 for SIS-ba10-5k
- NUM_GENERATIONS = 45
- POP_SIZE = 10
- NEW_POP_SIZE = 30

Experiments on coupled map dynamics (Table 5.3):

GGN:

- $S_d = 30$, $S_n = 5$ (Zhang et al. used the same values for CM dynamics in [9].)
- TEMP_DROP_FACTOR = 0.98 (seems to converge slightly faster than when using .95, but not a huge difference)
- Other parameters identical to previous experiments:
  NUM_CYCLES = 100, INITIAL_GUMBEL_TEMPERATURE = 10.

Hill climbing, simulated annealing:

- NUM_DYN_EPOCHS = 200
- NUM_STEPS = 45

Population-based local search:

- NUM_DYN_EPOCHS = 200
- NUM_GENERATIONS = 45
- SELECTION_SIZE = 10
- POPULATION_SIZE = 30

Appendix B

Additional PBLS Visualizations

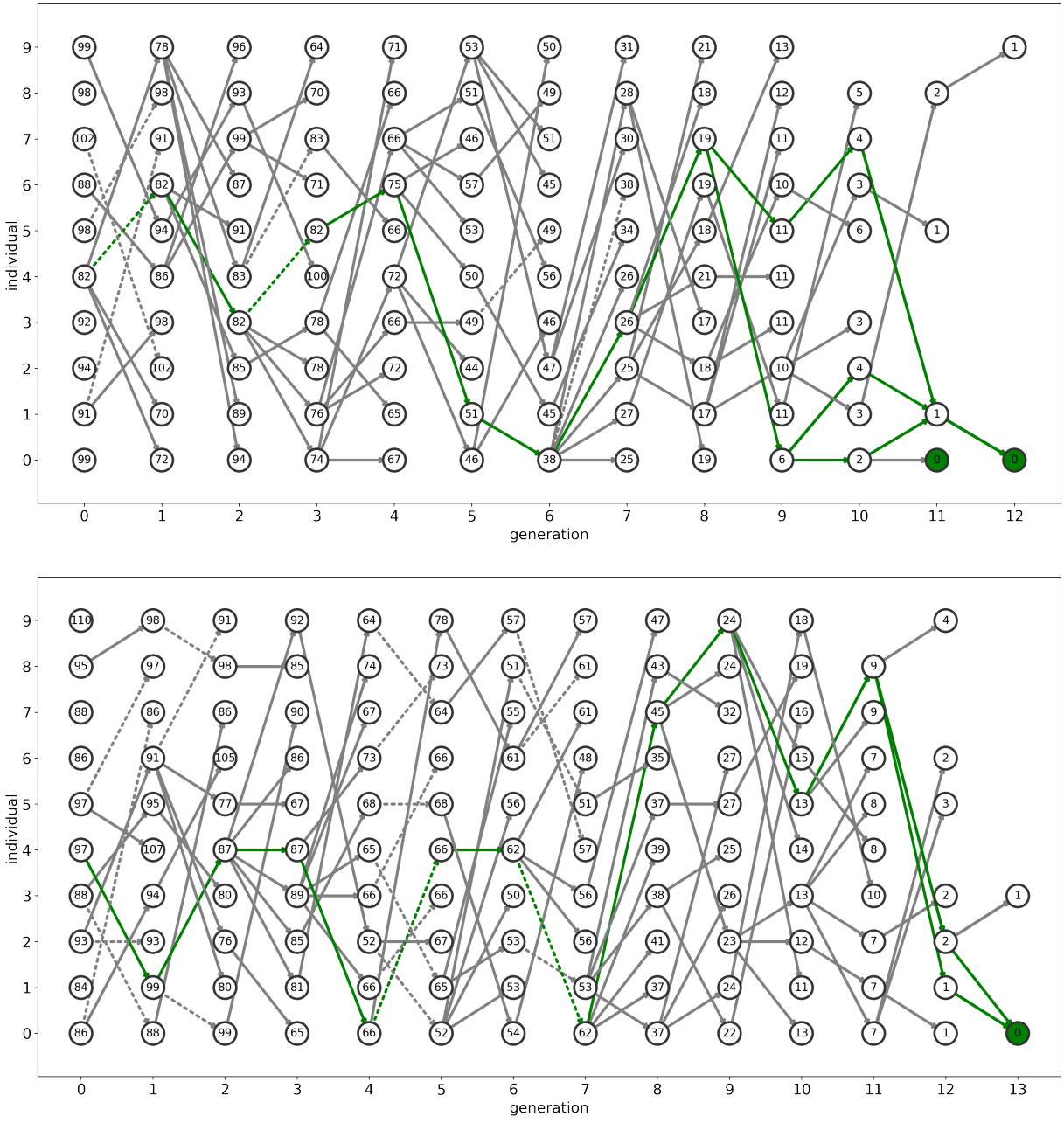

Figure B.1: Visualizations of PBLS (Eval) on the CML-4.0-BA20-1k dataset.

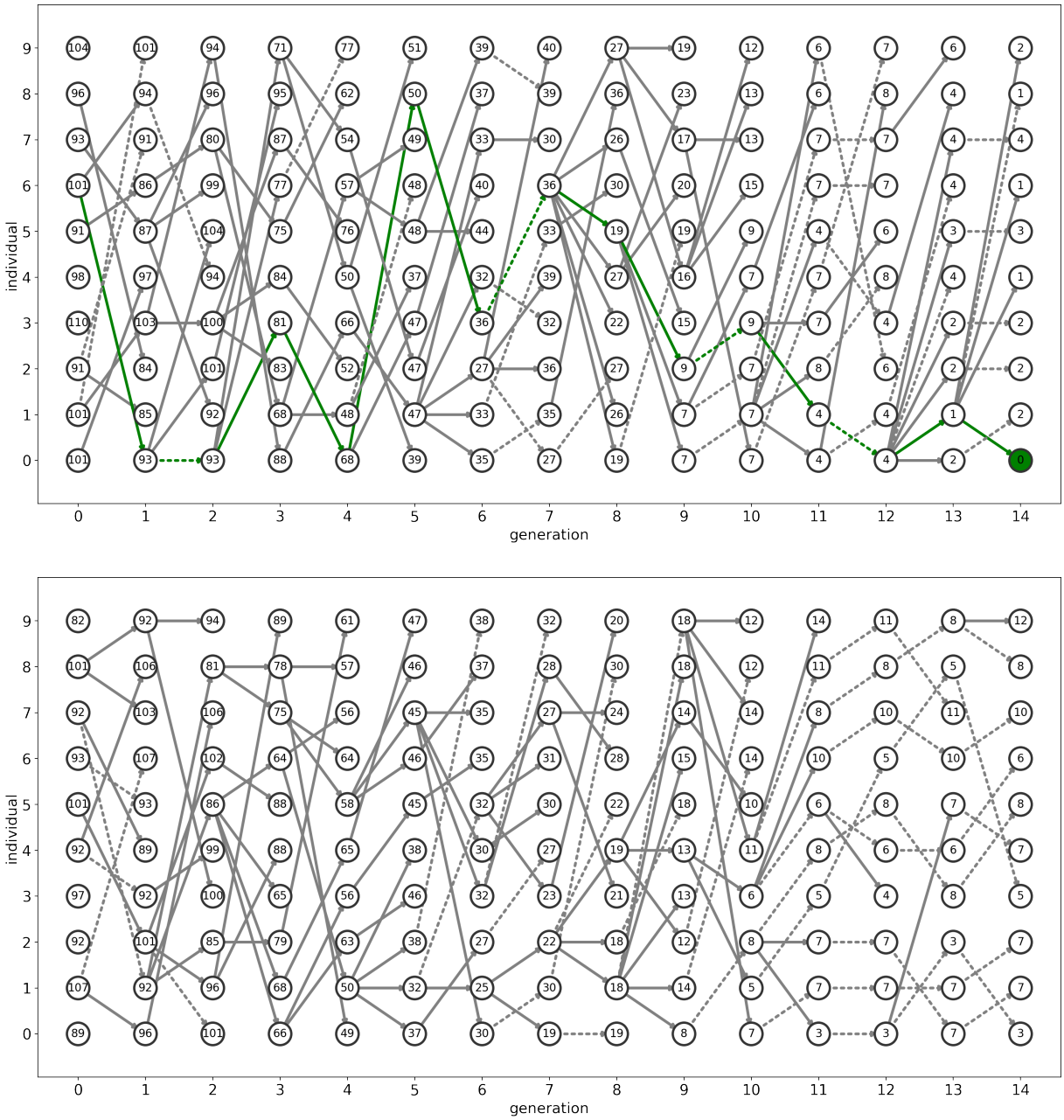

Figure B.2: Visualizations of PBLS (Grad) on the CML-4.0-BA20-1k dataset.

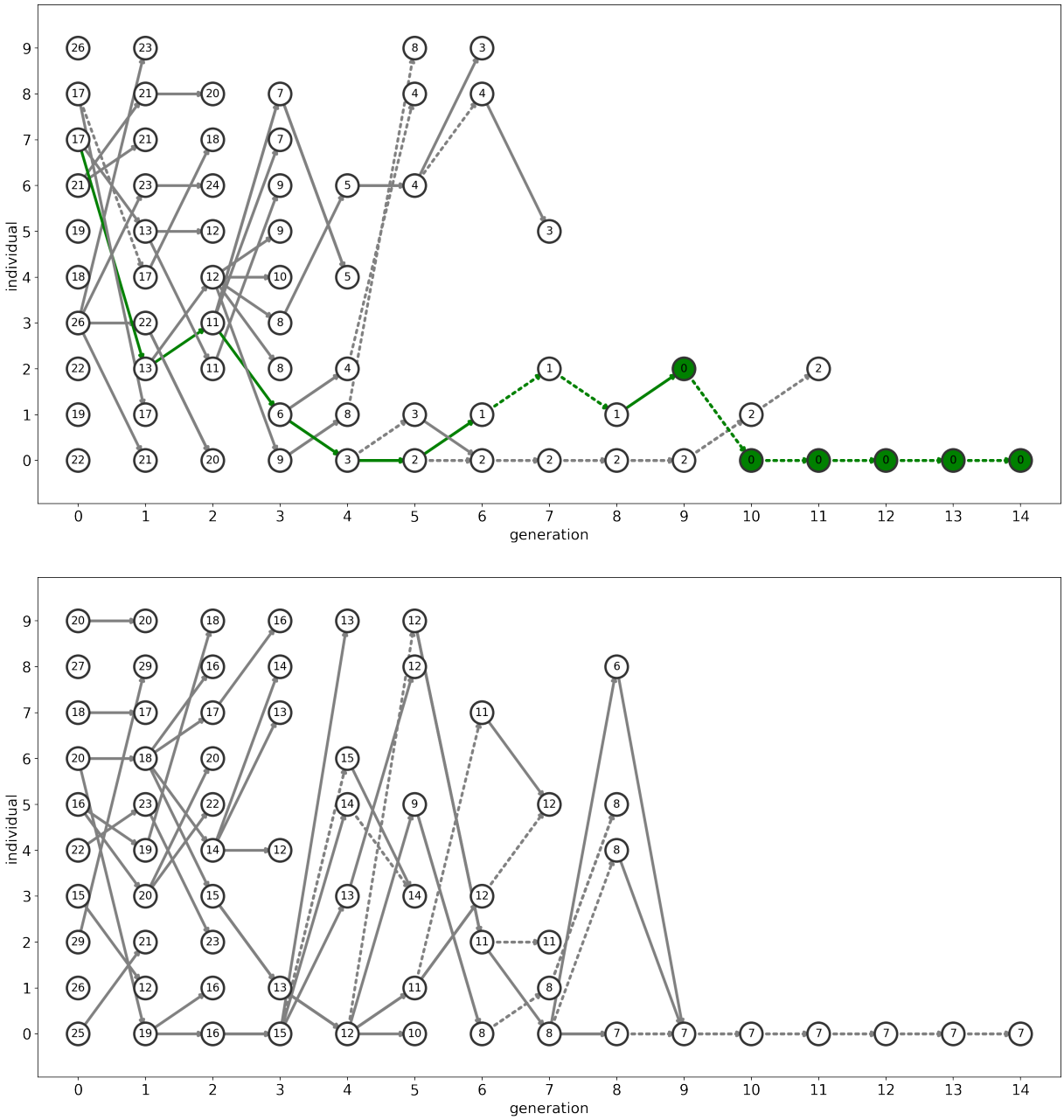

Figure B.3: Visualizations of PBLS (Eval) on the SIS-BA10-1k dataset.

Note that we merge identical individuals in each generation into one node in our visualizations. This is the reason why there is just a single node displayed in the later generations in Figure B.3; there are in fact still 10 individuals, but they are all identical (i.e. they represent the same network structure). This also explains for example why individual (6,0) in the first plot appears to have two parents — the two incoming edges actually belong to two elements of the population which have been merged. We observe "convergence" to a single individual only when using the evaluation heuristic. This is caused by the evaluation heuristic returning all-negative mutation score matrices in local minima, so that all offspring of the individual are just the individual itself again (instead of mutated versions) which can then quickly take over the whole population in a few generations.

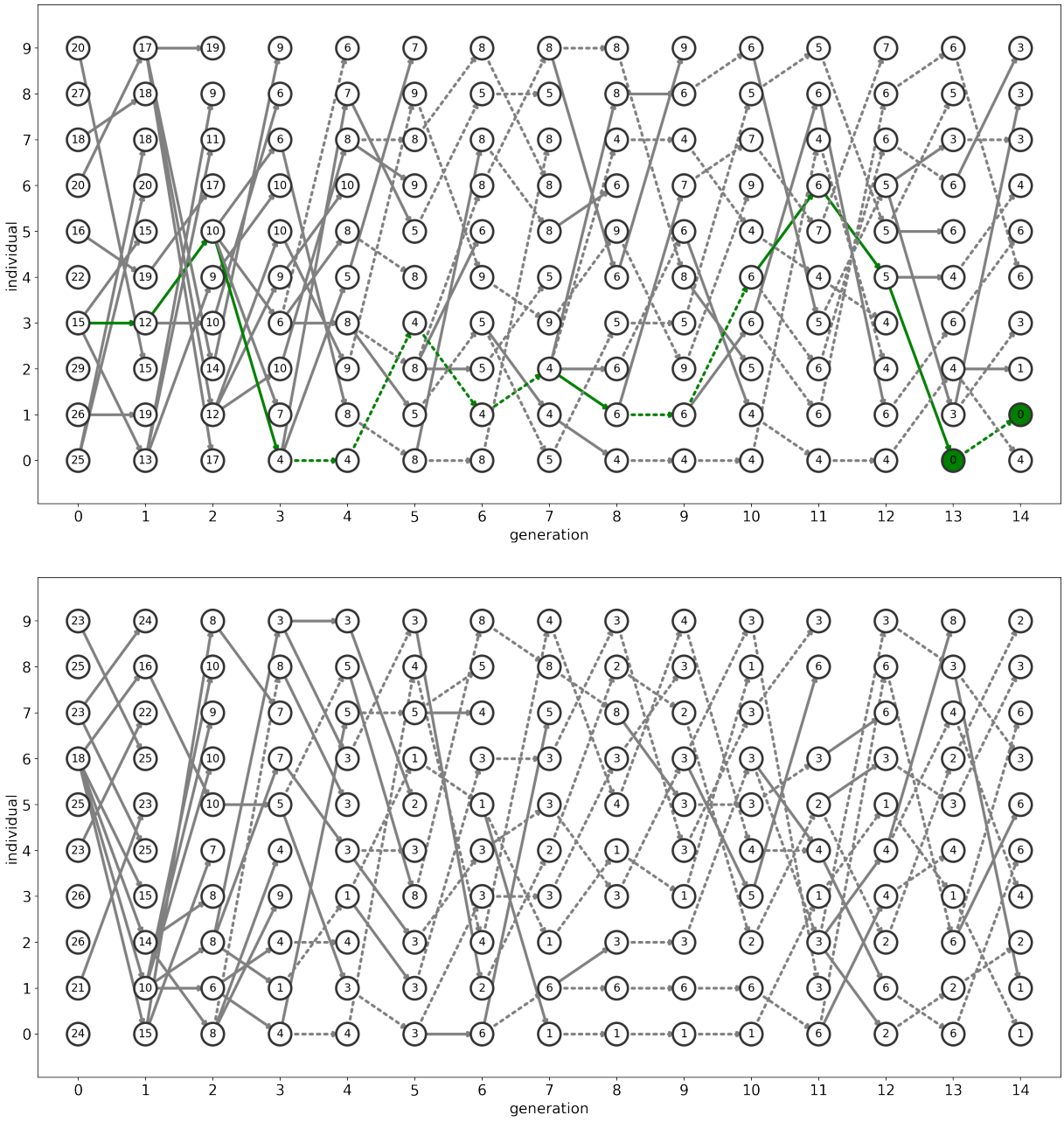

Figure B.4: Visualizations of PBLS (Grad) on the SIS-BA10-1k dataset.