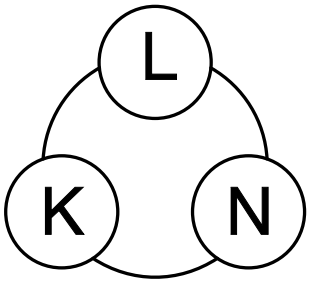


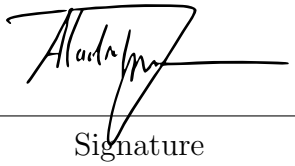
Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Research Internship

Community Detection in Large-Scale Communication Networks using Stochastic Block Modeling

Author: Djuhera, Aladin
Matriculation Number: 03671760
Supervisor: Kraemer, Patrick

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.



Munich, June 22 2021 — Place, Date

Signature

Munich, June 22 2021 — Place, Date

Signature

Abstract

In community detection, stochastic block modeling is used as a versatile modeling scheme which can be applied to obtain different types of network clusterings, dependent on the respective model implementation. However, a corresponding parameter inference is oftentimes extensive and requires the application of complex inference algorithms such as Expectation Maximization or Stochastic Variational Inference which are highly efficient but not easy to implement by hand. With the increasing capabilities of probabilistic programming languages though, the question arises whether stochastic block modeling can be efficiently implemented using the built-in inference algorithms of such frameworks and thus further integrated into a potential prototyping pipeline. In this work, a comparison between three such probabilistic frameworks, namely *edward1*, *Pyro* and *NumPyro*, and a novel, however proprietary approach to stochastic block model inference using *Natural Evolution Strategies*, is conducted. In particular, the work aims to demonstrate that although being versatile and in principle quite capable, most of the investigated probabilistic frameworks either fail to successfully and/or efficiently implement the most simple version of the stochastic block model. Thus, the focus is shifted towards the *Natural Evolution Strategies* implementation which is eventually used to effectively detect community structures in a large-scale campus network. The resulting clustering could then be used in a subsequent step to further classify the respective network traffic.

Keywords: community detection, inference, natural evolution strategies, network traffic classification, stochastic block modeling

Kurzfassung

In der Klassifizierung von Gruppenstrukturen werden Stochastische Block Modelle verwendet, um abhängig von der jeweiligen Modellimplementierung verschiedene Typen von Netzwerk Clustern zu identifizieren. Die damit verbundene Parameterschätzung (Inference) ist jedoch oft umfangreich und bedingt der Anwendung von komplexen Inferece Algorithmen wie Expectation Maximization oder Stochastic Variational Inference, welche hoch effizient, jedoch nicht einfach händisch zu implementieren sind. Durch die Weiterentwicklung von probabilistischen Programmiersprachen und den damit verbundenen Implementierungsmöglichkeiten, stellt sich zudem die Frage, ob solche Frameworks durch ihre internen Inference Algorithmen eine effiziente Realisierung von Stochastischen Block Modellen erlauben, welche im Anschluss in bestehende Prototyping Pipelines integriert werden könnten. Vor diesem Hintergrund wird in dieser Arbeit ein Vergleich zwischen drei solcher probabilitsischen Programmiersprachen, nämlich *edward1*, *Pyro* und *NumPyro*, und einem neuen, jedoch eigenimplementierten Ansatz mit *Natural Evolution Strategies*, durchgeführt. Inbesondere wird gezeigt, dass obwohl obige Frameworks vielseitige Stärken aufweisen, sie letztendlich entweder daran scheitern die einfachste Form eines stochastischen Block Modells zu implementieren oder diese nicht effizient implementieren können. Deswegen werden *Natural Evolution Strategies* weiter in den Vordergrund gestellt und damit Gruppenstrukturen eines großskaligen Campusnetzwerkes identifiziert. Diese könnten in einem weiteren Schritt benutzt werden, um den Datenverkehr des Netzwerkes zu klassifizieren.

Contents

Abbreviations

AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
DoE	Design of Experiments
ELBO	Evidence Lower Bound
EM	Expectation Maximization
ES	Evolution Strategies
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HMC	Hamiltonian Monte Carlo
IP	Internet Protocol
JIT	Just-In-Time
JSON	JavaScript Object Notation
KL	Kullback-Leibler
MAP	Maximum A Posteriori
MCMC	Markov Chain Monte Carlo
MLE	Maximum Likelihood Estimation
NES	Natural Evolution Strategies
PCAP	Packet Capture
PPL	Probabilistic Programming Language
SBM	Stochastic Block Model
SVI	Stochastic Variational Inference
TFP	TensorFlow Probability
TPU	Tensor Processing Unit

Chapter 1

Introduction

As communication networks are becoming larger and more complex at an ever-increasing rate, network engineers are faced with continuous challenges to properly analyze, maintain and most importantly secure the underlying networks w.r.t. loopholes and other malicious threats. To this, a comprehensive understanding of the network topology and its corresponding dynamics is of greater importance. This is also most certainly the case for traffic classification and resource allocation where useful insights can be obtained by applying community detection methods which allow for a restructurization of the network into several coherent clusters [TPGK03]. The considered network in this case is then a representation of nodes and edges which correspond to clients and servers and their data flows, respectively. It is thus most convenient to model data flows based on IP addresses and to define IP related grouping criteria such as the used network protocols, for example. Such an implementation greatly reduces the network analysis complexity and thus motivates community detection for traffic classification tasks where IP-to-IP graphs are already a common analysis tool [IKF+11].

The main purpose of community detection algorithms is to identify distinct groups, communities or clusters within a specified distributed network. Examples for such networks can be of different nature and thus community detection methods have been introduced in various fields of research. As such, some interesting applications can be found for the synthesis of protein-protein interactions in biological networks [NGSG11], for the analysis of community structures in social networks [AR+14] as well as for financial engineering problems where community detection can be used as a tool for the analysis of stock markets [Kap08]. Yet, the core detection goal remains the same which is to most accurately infer clustering structures that might allow for a further network analysis, identification of group relationships and other tasks specific to the respective field of application. The result of an example community detection as detailed in [Abb16] is depicted in below Figure 1.1 where the left network graph has been transformed into the grouped representation on the right using community detection methods. Note however that the right graph does not alter the original evidence and is thus just another valid representation of the network.

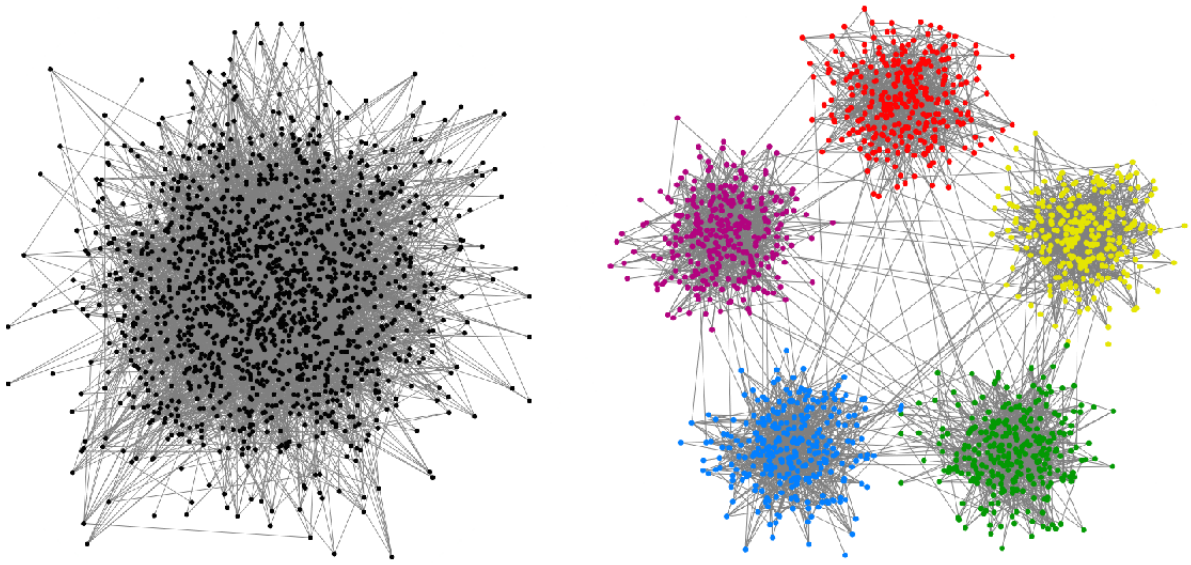


Figure 1.1: Community Identification Result (right) using an SBM Model for a Synthetically Generated Network Graph (left) with 1000 Vertices and 5 Balanced Communities. From *Community Detection and the Stochastic Block Model*, by E. Abbe, 2016, page 5.

However, as the set of problems to which community detection can be applied varies greatly, implementation of appropriate detection models and algorithms becomes an increasingly custom objective. Consequently, a manual implementation oftentimes requires a higher effort in the realization of accurate working models. As such, frameworks that incorporate - or allow for - community detection models are highly desired in order to obtain an automated discovery of relevant network properties such as node relations and other structures. This would further allow for a less extensive fast prototyping and initial analysis purposes and thus highly desired from the perspective of the network analyst. As most community detection models rely on a stochastic mathematical framework, *probabilistic programming languages* (PPLs) and other black box optimization techniques seem to provide a promising environment for the implementation of such automated detection variants. Furthermore, as for the case of PPLs for example, they represent independent and supported platforms which are general purpose in their nature and thus equipped with various already efficient optimization algorithms.

In order to eventually perform above described community detection in large-scale communication networks, the widely researched approach using *Stochastic Block Models* (SBMs) is investigated here. However, the focus of this work does not primarily lie on a successful implementation of a specific SBM variant and its subsequent community detection, but also on the performance comparison of SBM modeling between ready-to-use probabilistic inference frameworks - such as Pyro [BCJ+19] or NumPyro [PPJ19] - and proprietary modeling approaches such as a novel implementation attempt using

Natural Evolution Strategies (NES). The main contribution of this work is thus to answer the question whether off-the-shelf PPLs - using the example of NumPyro - are already powerful and versatile enough to be considered for network traffic classification tasks.

The corresponding analysis and results of this work are presented as follows.

Chapter 2 introduces the mathematical SBM modeling framework as well as some suitable inference approaches that are considered for the community detection task. Chapter 3 encompasses the corresponding materials and methods including the derivation, preprocessing and analysis of the underlying communication network data. Furthermore, a summary regarding the probabilistic frameworks that are being worked with is given. Chapter 4 is devoted to the evaluation of the respective SBMs on both benchmark data as well as the processed network data which represent a real-world use case in this regard. Eventually, Chapter 5 concludes the elaborated results and findings and gives an outlook on further prospects of SBM modeling for network traffic classification.

Chapter 2

Stochastic Block Modeling

This chapter introduces the essential terminologies, concepts and algorithmic approaches in community detection using SBMs. It can be regarded as the knowledge base of the thesis and covers the general nature of probabilistic generative models, the stochastic framework of the standard SBM and the corresponding inference problem that origins from the SBM model description. Furthermore, two candidate inference algorithms are presented which are subsequently evaluated in Chapter 4.

2.1 Probabilistic Generative Models

Before introducing SBMs as a tool to identify communities for relational data, first the probabilistic mathematical framework around it is briefly addressed to understand the important relation to PPLs. In general, SBMs are generative probabilistic models, that is models that are based on the random graph model approach and can thus be defined by a probability distribution which depends on a structural set of well-defined parameters θ and the underlying set of observed (network) data A. The θ-parametric probability distribution can thus be consequently denoted as

$$P(A|\theta) \tag{2.1}$$

The choice of such a probabilistic modeling approach allows for some important advantages as thorougly discussed in [Cla]:

- The definition of discrete model parameters can be used to (graphically) interpret their values.
- Different model variants can be easily and fairly compared by the use of well known likelihood scores.
- By the assignment of probabilities to the observed data, estimation of unobserved structures can be performed.

Most importantly though, obtaining a generative model not only allows to generate synthetic data - which can be used for various testing purposes - but also allows for an easier inference scheme as backpropagation can be used. In particular, inference in probabilistic graphical models assigns probabilities to edges with a pre-defined score functional [Cla]. With these main aspects in mind, it becomes very clear as to why we want to perform SBM modeling in a closed PPL.

2.2 Standard Stochastic Block Model

A generative probabilistic model which aims to model the relationships between network nodes - and with that the respective community associations of those - is called a stochastic block model. To be precise, one distinguishes between multiple variants of SBMs such as mixed membership models - where nodes are allowed to be part of multiple communities at the same time - and many others. However, the simplified unitary membership model is oftentimes also referred to as the *Standard Stochastic Block Model*, *Bernoulli Stochastic Block Model* or simply *Stochastic Block Model*. Other model variants are always denoted explicitly such as the *Poisson* or *Degree-Corrected* SBM. In this section, the standard SBM is defined according to the general structure described by Kevin P. Murphy in [Mur12].

In general, the standard SBM represents one of the simplest forms of previously described probabilistic generative models. For a given number of communities K it is mainly defined by three entities:

- Community Association Z
  Describes the respective community/cluster to which each node belongs to.
- Stochastic Block Matrix η
  Describes the probabilities of possible inter-group connections.
- Network Adjacency Matrix A
  Describes the relationships between the network nodes.

Their respective mathematical frameworks are discussed in the sections below.

2.2.1 Community Association Z

For every network node i, a latent block q_i is sampled as follows and denotes the respective community association.

$$q_i \sim Cat(\pi) \tag{2.2}$$
$$\text{with } \pi = [\pi_1 \dots \pi_K] \tag{2.3}$$

The distinct π_i represent the group membership priors and as such describe the probability of choosing clusters 1 to K. Consequently, this prior follows a Dirichlet distribution.

$$\pi_i \sim \text{Dir}(\alpha) \tag{2.4}$$

with concentration parameter α. The respective community association for each node is then given as

$$Z = [q_1 \dots q_N] \tag{2.5}$$

2.2.2 Stochastic Block Matrix η

The entries $\eta_{i,j}$ of the block matrix denote the probability of respective group i being connected to group j. These distinct probabilities follow a Beta distribution

$$\eta_{i,j} \sim \text{Beta}(\alpha, \beta) \tag{2.6}$$

with concentration parameters α and β. Consequently, the block matrix η is of form $K \times K$.

2.2.3 Network Adjacency Matrix A

Eventually, each edge of the adjacency matrix A is generated given the block matrix η using the following model where Ber denotes the Bernoulli distribution. This is also the reason as to why the standard SBM is oftentimes referred to as the Bernoulli variant of the SBM.

$$p(A_{i,j} = r | q_i = a, q_j = b, \eta) = \text{Ber}(r|\eta_{a,b}) \tag{2.7}$$

Correspondingly, the generative probabilistic SBM model can be described by the following distribution analogously to Equation (2.1)

$$P(A|\theta) = P(A|\eta, Z) \tag{2.8}$$

2.2.4 Directed Graphical Model

With above definitions, we can alternatively describe the mathematical structure of the SBM using a compact directed graphical model as shown in below Figure 2.1 from [Mur12], where the respective conditional (in-)dependencies can be conveniently described using plate structures. Note that $R_{i,j}$ corresponds to $A_{i,j}$ in our case.

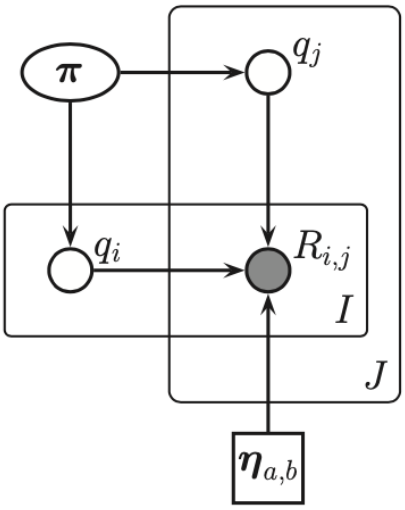


Figure 2.1: Directed Graphical Model of the Standard SBM. From *Machine Learning: A Probabilistic Perspective*, by Kevin P. Murphy, 2012, page 973.

2.2.5 Summary

In summary, the standard SBM can be precisely parameterized without a general loss of interpretability w.r.t. the meaning of the model parameters. In fact, above definitions show that each parameter denotes a specific interpretable purpose in the graphical model and can thus be very well understood as they are each assigned a probability based on the underlying data set. However, as only the network adjacency matrix A can be observed, the parameter set θ and its corresponding probabilistic priors are distinctly latent. Accordingly, inference (optimization) takes place over the latent space, as to why SBMs are also referred to as latent variable models. The corresponding maximization approach to which the SBM eventually alludes to can be formulated as follows and solved using various inference techniques.

$$\max_{\eta,Z} P(A|\eta, Z) \tag{2.9}$$

In other words, the optimization objective encompasses the maximization of the posterior probability - as described in Equation 2.8 - based on the respective parameter set $\theta = [\eta, Z]$ which is to be optimized.

2.3 Inference Problem

Once again, the main goal of the SBM is to derive a model which accurately describes the network community structure based on a given data set. The respective community structure is described by each node's community association q_i in above defined vector Z and the supplied data set comprises the observed network adjacency matrix A. The mathematical challenge in this problem thus is to accurately infer the latent variables of the optimization problem in Equation (2.9). To this, variational inference can be used to optimize previously defined variational (latent) parameters, which eventually leads to a learning process in which the community structure Z is inferred. This general algorithmic procedure and its corresponding framework according to [Bleb] and [Blea] are briefly described as follows.

In general, above defined probabilistic model describes a joint probability distribution of latent variables L and observed values X

$$p(L, X) \tag{2.10}$$

where we have

$$L = \{Z, \eta\} \tag{2.11}$$
$$X = A \tag{2.12}$$

Assume that we can sufficiently approximate the latents L by a family of probability distributions

$$q(L; \mu) \tag{2.13}$$

where L and μ describe the latent variable set and some corresponding latent variational parameters, respectively.

The inference problem can then be rephrased as an optimization objective for which we want to minimize the *Kullback-Leibler* (KL) divergence - which quantifies the similarity of two distributions - between the posterior and the variational family of distributions

$$KL(q(L; \mu) \mid\mid p(L, X)) \tag{2.14}$$

This eventually leads to an optimization of $q(L; \mu)$ by means of adjusting the distribution family to be as close as possible to the posterior $p(L, X)$.

In order to solve this optimization problem efficiently, many variational approaches - such as *Stochastic Variational Inference* (SVI) - can be applied. The resulting optimized parameters can then be used to sample the desired community structure Z from above defined probabilistic framework.

Before diving deeper into various inference approaches, it is worthwhile to first explain the need for an efficient optimization. Let us recall above maximization objective from Equation (2.9)

$$\max_{\eta,Z} P(A|\eta, Z) \tag{2.15}$$

As discussed, inference aims at minimizing the KL divergence by approximating a family of distributions to be as close as possible to the original posterior. In order to find such best approximation, we need to optimize over the respective family parameters which is done by means of classical *Bayesian Inference* using a *Maximum A Posteriori* (MAP) approach. Here we combine the prior and likelihood distribution in order to obtain the so-called posterior. During *Maximum Likelihood Estimation* (MLE), we can then use the posterior distribution to maximize the posterior probability as follows

$$P(\theta|X) = \frac{P(X|\theta) \cdot P(\theta)}{P(X)} = \frac{P(X|\theta) \cdot P(\theta)}{\int_\theta P(X|\theta) \cdot P(\theta) d\theta} \propto P(X|\theta) \cdot P(\theta) \tag{2.16}$$

where $P(\theta)$, $P(X|\theta)$ and $P(\theta|X)$ denote the prior parameter distribution, the likelihood and the posterior, respectively. Note that the denominator expression $P(X)$ only serves as a normalization constant [KF09], however, this is exactly where where MAP becomes intractable for higher dimension problems as we have to optimize over many different parameter values. In order to solve this problem, *Expectation Maximization* (EM) algorithms, *Monte Carlo* techniques and other methods can be used.

In the following sections, two possible optimization techniques are discussed: *Markov Chain Monte Carlo* (MCMC) inference and *Natural Evolution Strategies* (NES). A reason as to why above SVI inference approach was not further considered is given in detail in below remarks on the PPL framework Pyro in Chapter 3.

2.4 MCMC Inference

MCMC methods can be summarized as sampling methods as they aim to draw new samples from the posterior distribution. As such, we can eliminate the complex computation of above integral in Equation 2.16 and simply average over the generated samples. Accordingly, we can predict the probability of observing a new sample x' by marginalizing over all parameter values θ as described in [Arb] for the following expression.

$$P(x'|X) = \int_\theta P(\theta|X) \cdot P(x'|\theta) d\theta \tag{2.17}$$

Still, a closed analytical solution might not be possible for higher dimensions. However, if we can take samples from the posterior distribution, we can give an approximation to

Equation (2.17) as follows.

$$\int_{\theta} P(\theta|X) \cdot P(x'|\theta) d\theta \simeq \frac{1}{n} \sum_{1 \leq i \leq n} P(x'|\theta^i) \tag{2.18}$$

This is the general Monte Carlo estimation method where the θ^i denote the samples from the posterior.

$$\theta^i \sim P(\theta|X)$$

As such, Monte Carlo methods are very helpful for tasks where complex (intractable) integrals need to be computed. Furthermore, Monte Carlo estimators are unbiased. However, in order to efficiently sample from $P(X)$, Monte Carlo relies on many different sampling techniques such as *Rejection Sampling* and *MCMC*. The latter sampling alternative can in particular be used to draw dependent samples as they rely on the memoryless markov property which is in fact very similar to above previous considerations where we draw a sample from a proposal distribution that now is dependent on the current state as follows [Sal].

$$P(x^i|x^1, \ldots, x^{i-1}) = P(x^i|x^{i-1}) \tag{2.19}$$

In summary, Monte Carlo algorithms are very easy to implement general purpose methods. MCMC in particular defines a dynamical system from which we can effectively sample trajectories as often needed in practice. To this, there are many different realization variants such as Gibbs, Metropolis-Hastings, Hamiltonian Monte Carlo and many others that each come with their respective advantages and disadvantages. However, these methods suffer from uncertainty w.r.t. convergence often described as *burn in* [KF09] that needs to be considered during inference. In general though, oftentimes good approximations with a less intensive parameter tuning can be achieved using MCMC.

2.5 Natural Evolution Strategies

In contrast to MCMC, NES has its origins in reinforcement learning and promises - although being a decades old idea - to achieve comparable results to usual reinforcement strategies, whilst being simpler to dimensionize as well as highly parallelizable [SHC+17]. Thus, it belongs to the spectrum of black box stochastic optimization techniques and does not require a strict probabilistic framework as introduced in above inference problem. As such, NES represents an alternative inference approach to MCMC and SVI.

In general, NES is based on the simple idea of *Evolution Strategies* (ES) where optimization is achieved by iterative updating of a specified search distribution. In ES, each iteration represents one generation with a corresponding population of parameters that is being

mutated (perturbed). This way, we can ensure a certain randomness in the distribution of the parameters. Within each generation, the respective model fitness is evaluated based on a pre-defined objective function after which the best parameter vectors are recombined and essentially form the basis for the next generation with updated parameter values. Using this scheme, new generations are formed until the objective is fully optimized [SHC+17].

The evolution of ES to NES implements an additional iterative updating using stochastic gradient descent. In a first step, the parameterized search distributions generate corresponding search points for which a local fitness function is evaluated, respectively. Furthermore, the search distribution parameters include certain strategy parameters that allow to capture the local fitness function structure. Subsequently, when generating new samples, NES performs a gradient estimation on the parameters which eventually surge into a higher expected fitness area. Correspondingly, NES follows the natural gradient by performing gradient ascent steps along it. Additionally, by normalizing the natural gradients, further local oscillations can be prevented [WSPS08].

The mathematical framework corresponding to the work presented in [SHC+17] is briefly defined as follows. We denote the objective functional over the parameters θ as F. In contrast to ES, NES represents the population using a distribution over parameters

$$p_\phi(\theta) \tag{2.20}$$

which itself is parameterized by a set ϕ. The objective functional is then maximized over the population using stochastic gradient ascent which delivers a suitable set of ϕ.

$$\mathbb{E}_{\theta\sim\phi}F(\theta) \tag{2.21}$$

In particular, the algorithm performs gradient steps on this set using the following estimator

$$\nabla_\phi \mathbb{E}_{\theta\sim p_\phi}F(\theta) = \mathbb{E}_{\theta\sim p_\phi}\{F(\theta)\nabla_\phi log(p_\phi(\theta))\} \tag{2.22}$$

Oftentimes, p_ϕ is observed to be a factored Gaussian for which the resulting gradient estimator is called a *simultaneous perturbation stochastic approximation* [S+92].

To finally link NES with SBMs, we have to consider the nature of this black box optimization approach. In general, we treat the SBM parameters as mutable parameter sets that are being optimized throughout the process. The resulting parameters should thus in fact approximate the underlying network data w.r.t. the configurable SBM parameters. In particular, the population size can be interpreted as the number of parallel parameter sets that are being compared to each other during a generation. The learned parameter sets thus do not stem from an inherently probabilistic inference framework. In contrast to the MCMC approach, NES is however sensitive to its initial parameter specification. As such, it requires a more extensive parameter optimization in form of a well-defined grid search followed by a specified *Design of Experiments* (DoE) in order to obtain the best performing set of parameters.

Chapter 3

Materials and Methods

In this chapter, first two independent benchmark data sets are introduced which are used for initial evaluation purposes of the implemented SBMs in Chapter 4. Next, the corresponding generation, pre-processing and subsequent analysis of the real-world communication network data are discussed. This chapter eventually concludes with a detailed analysis of the PPL frameworks Pyro and NumPyro which were selected for the automated community detection task and explains some of the problems which were encountered during the implementation of the SBM.

3.1 Benchmark Data Sets

For an initial performance comparison between the PPL framework and NES realization of the SBM, two different data sets are used.

3.1.1 Zachary's Karte Club

The widely known karate club data is a very popular set of observations for benchmarking in SBM development. It consists of 34 members, where two explicit groups are formed during a dispute:

- Group A: Members who support Mr. Hi
- Group B: Members who support the Officer

The respective data contains two information sets:

- General friendship connections between the respective members
- Association/Grouping for Mr. Hi or the Officer

As such, the data describes an undirected, simple graph as seen in below Figure 3.1 from [HSSC08], for which a standard SBM can be implemented for community detection purposes. In particular, the adjacency matrix A and community association Z are of dimensions:

- A: $[34 \times 34]$
- Z: $[1 \times 34]$

This renders the karate club example a very simple and lightweight data set which ultimately explains its popularity among initial benchmarking in the SBM community.

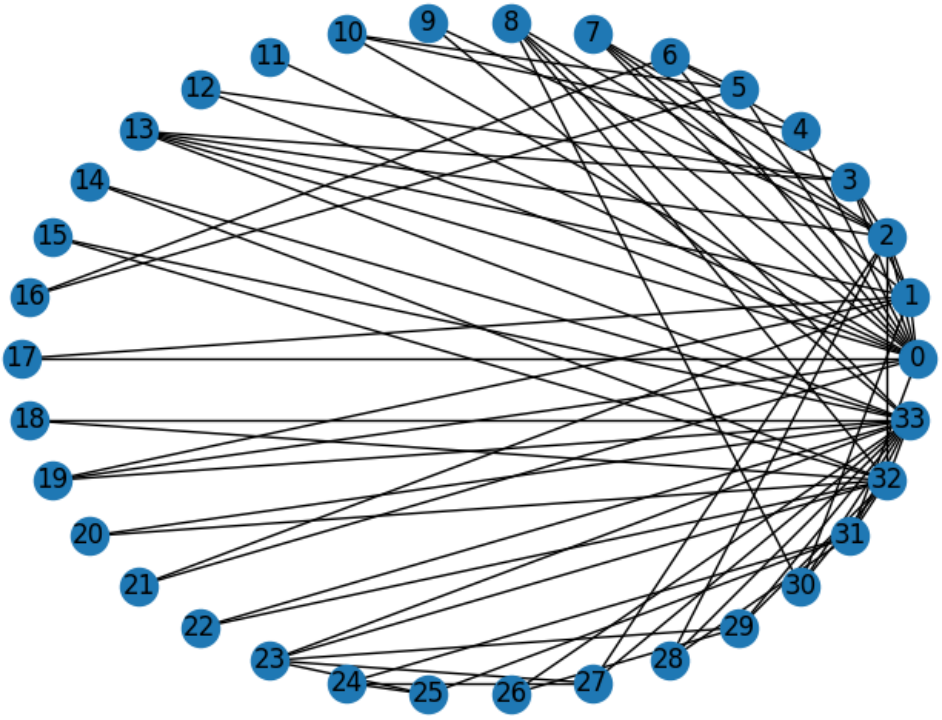


Figure 3.1: Zachary's Karate Club Graph with 34 Nodes and 78 Edges. From *Exploring Network Structure, Dynamics and Function using NetworkX*, by Hagberg et al., 2008.

3.1.2 Simplified Toy Network

Another very generic example for a fully intra-group-connected graph is given in below Figure 3.2. As can be seen, all of the nodes in the respective two groups are fully connected to each other and the groups themselves are only connected via one edge. This represents one of the most simple graph structures to be identified using SBM. It has been mostly used for testing purposes during the implementation of the NumPyro SBM.

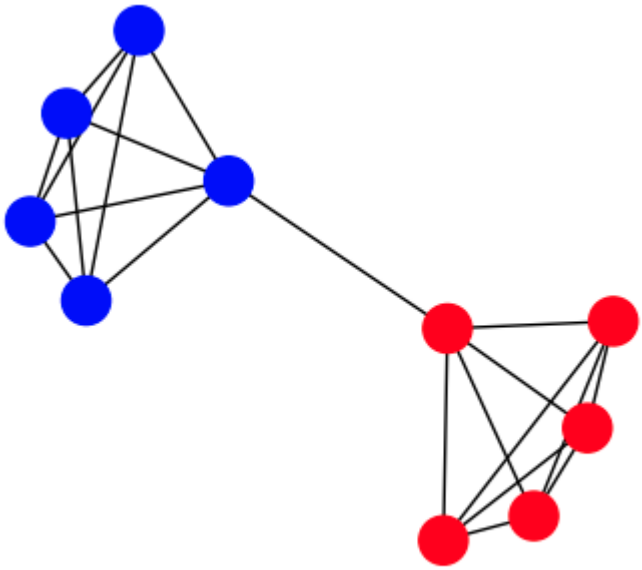


Figure 3.2: Simplified Toy Network with 10 Nodes and two Groups (Red and Blue).

3.2 Communication Network Analysis

The underlying data that is being worked with consists of about 200 GB of network traces which were recorded between August 12 and August 14 of 2020 at the TUM chair of communication networks. As such, it represents a real-world network of researchers, students and assistants for which it is interesting to investigate group clusterings inside the communication structure as well as test the performance of the implemented (Bernoulli) SBM on non-artificial data.

For a fully comprehensive analysis of these traces, both a general and more granular investigation is conducted. The former concentrates on a broader analysis of the network traces in regards to common network metrics - such as the number of nodes, edges and sent packets - over the whole time span, while the latter investigates the same metrics for significantly smaller time windows of 15 minutes of duration. This allows for a more thorough temporal insight into the network dynamics.

3.2.1 Graph Generation and Data Pre-Processing

Most of above described common network metrics can be easily extracted from the underlying graph structure where nodes and edges represent IP addresses and valid communication paths between those, respectively. However, in order to obtain such a graph, the corresponding network traces first need to be pre-processed, accordingly. Since Python's *NetworkX* library offers powerful graph analysis tools, the graph will be generated in a corresponding *NetworkX* format as follows.

1. First, we convert the traces from their original packet capture format (PCAP) into a more suitable and most importantly parseable CSV format.
2. Next, for each trace file, we remove irrelevant entries for traffic classification such as connections below the transport layer and we also mask the destination ports to a fixed value, accordingly.
3. Then, we group the data by source and destination IP addresses and accumulate the corresponding traffic size and number of sent packets between the respective source and destination.
4. Finally, for each such grouped IP address tuple, we create a directed graph by adding the corresponding tuples as nodes and edges to a *NetworkX* graph structure.

In order to avoid a data overload caused by the 200 GB of files, the graph is generated iteratively, that is the traces are loaded in batch-wise and the respective nodes and edges are added accordingly. The resulting graph structure can hereby be reduced to a JSON file format of around 500 MB of size.

3.2.2 Common Network Metrics

Using above constructed graph structure, we extract the number of nodes, edges, traffic size, exchanged packets as well as additional graph properties such as the graph density and number of (weakly) connected components as shown in below Table 3.1.

Nodes	13.651
Edges	4.689.940
Traffic Size [GB]	199,615
Exchanged Packets	573.089.330
Graph Density	0.025
Connected Graph Components	5

Table 3.1: Common Network Metrics for all Traces

From this, we both conclude and verify that the resulting graph is of a low density considering the significantly higher number of edges as compared to the number of nodes. Another very interesting property of the network is the number of (weakly) connected components. In this case we have 5 individual clusterings that are not connected to each other. As such, they can be each seen as a mini group/cluster, which however does by no means imply a categorical grouping as derived by an SBM, for example.

Furthermore, since the traces cover a time span of more than 2 days, it is reasonable to also investigate the variations of above common network metrics between day and night time. By doing so, we obtain the averages depicted in Table 3.2 for different times of the day, respectively. From this analysis, we see that indeed the number of nodes decreases significantly during the night time, implying a lower general network activity. However, the number of edges, exchanged packets and most importantly traffic size do not decrease at the same rate during the night and rather remain comparable to the noon and evening statistics. Thus, the implication of a less busy network during night time cannot necessarily be made from these metric averages and a more detailed analysis is needed as performed in the subsequent sections.

Averages	Morning	Noon	Evening	Night
Nodes	4603	4.218	2.940	1.803
Edges	461.545	367.634	382.884	364.255
Exchanged Packets	56.128.679	44.271.326	47.174.989	49.897.848
Traffic Size [GB]	20,86	16,49	16,23	17,24

Table 3.2: Network Metrics Averages over Different Times of the Day

3.2.3 Node Degree Distribution

Regarding above low graph density, a degree analysis is subsequently performed in order to determine the degree distribution amongst the nodes. In particular, we analyze the complementary CDF (CCDF) for the network which is depicted in Figure 3.3. This allows us to investigate the relationship between high and low node degrees and their respective probabilities.

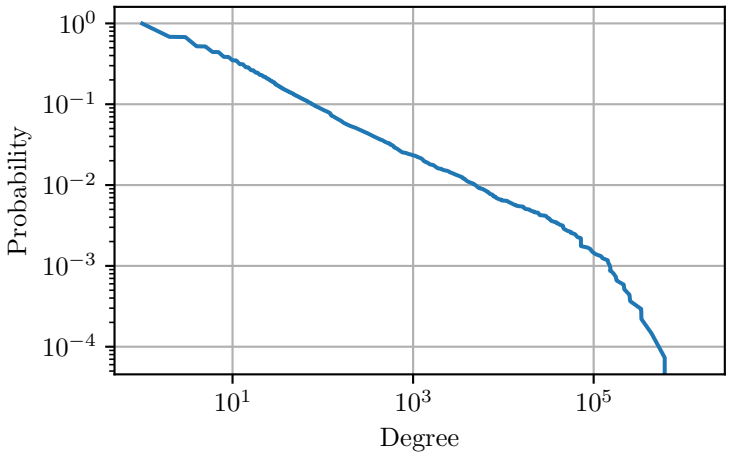


Figure 3.3: Power Law CCDF for Node Degree Distribution

From this power law distribution, we can conclude that indeed, most of the node degrees are in a small to medium sized range up to a couple of hundred connections per node. However, we also observe some nodes with extremely high degrees, though with a very low probability. This becomes especially clear when using above logarithmic scaling. Consequently, this gives motivation to further investigate the ten highest degree nodes in the network which range from a degree of 1.4 million to 180 thousand. By retracing the corresponding IP addresses, we identify the three highest degree nodes to belong to the network infrastructure whilst the others are mainly DNS related or do not exist anymore. The particular nature of these specific nodes thus reasonably explains the respective high node degrees.

3.2.4 Dominant Protocols and Services

Apart from above essential network metrics analysis, it is also important to investigate the dominant protocol types throughout the traces. As expected, most of the traffic relies on UDP and TCP which are the most dominant protocol types as shown in Figure 3.4. Correspondingly, we also need to investigate the dominant services in the same fashion. Their respective ratios with regard to traffic allocation are given in Figure 3.5. Interestingly enough, we observe that apart from https - which relates to most of the traffic - snmp seems to also dominate over the recorded two day time span.

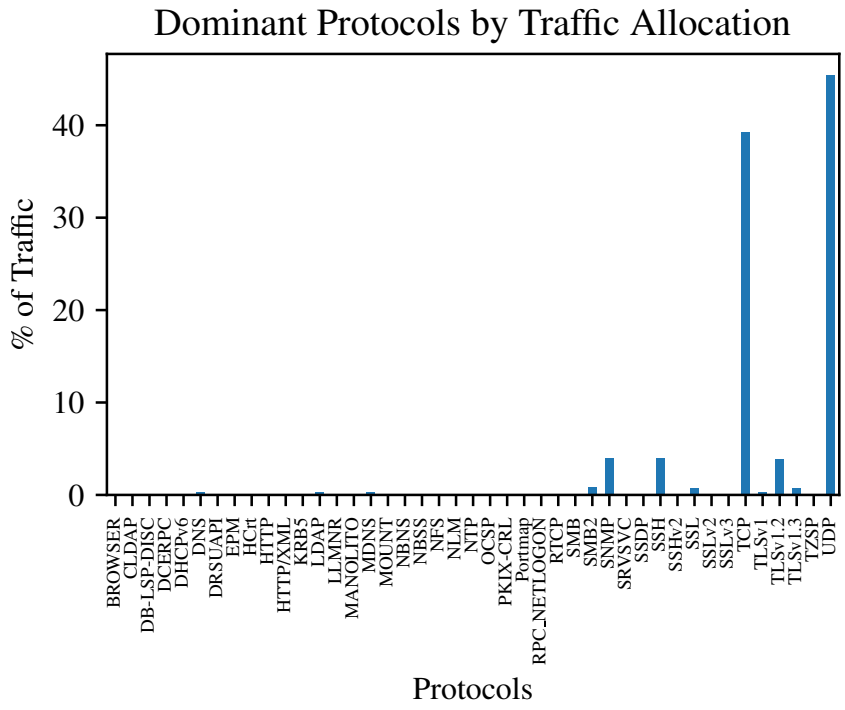


Figure 3.4: Dominant Protocol Landscape by Traffic Allocation

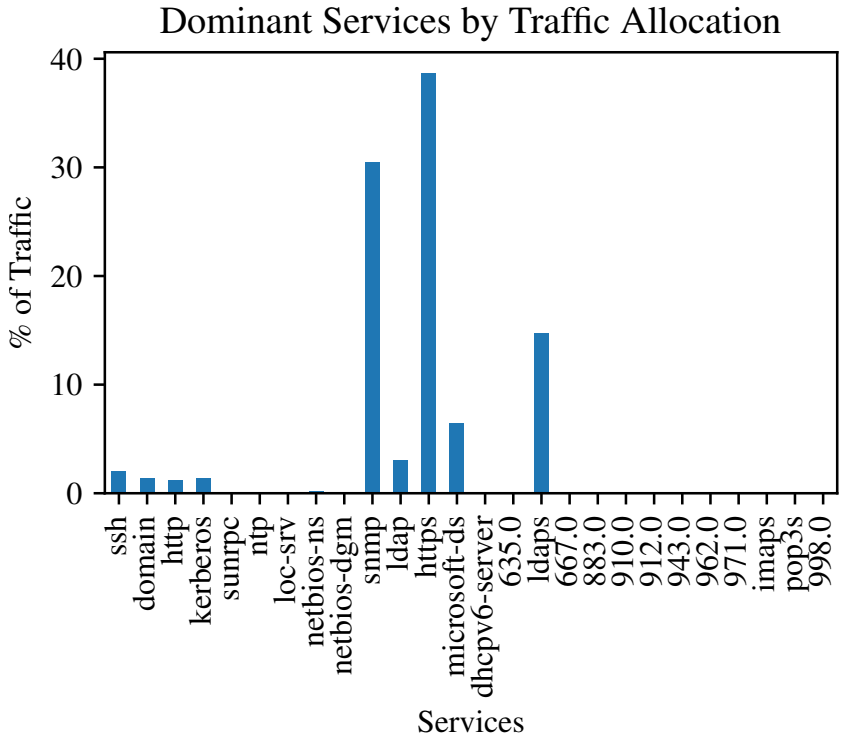


Figure 3.5: Dominant Services Landscape by Traffic Allocation

3.2.5 Node and Edge Progression

In order to obtain a more thorough insight into the network dynamics, we investigate the same metrics as above, but concentrate on significantly smaller time windows of around 15 minutes of duration. By doing so, we can observe time-dependent features as well as obtain a detailed temporal overview of the respective metrics.

For the analysis of nodes and edges, we extract their respective values for each time window and give an overview of their temporal progression in below Figures 3.6 and 3.8. From Figure 3.6, we can clearly observe the differences between day and night time for which the number of active nodes in the network increases and decreases, respectively. In particular, we see a significant increase in nodes in the morning hours between 7am and 11am and a decrease in the evening around 7-8pm. Moreover, we see higher than usual spikes on August 14 with a significant dip around the noon time. This increased activity also becomes clear when investigating the node degree progression over time as depicted in figure 3.7 where we observe the same spike again.

From the edge progression in above Figure 3.8, we observe - as expected - parallels to above node dynamics for which we see the same pattern in edge increase and decrease at roughly the same times. This coincides with the notion that an increase in network

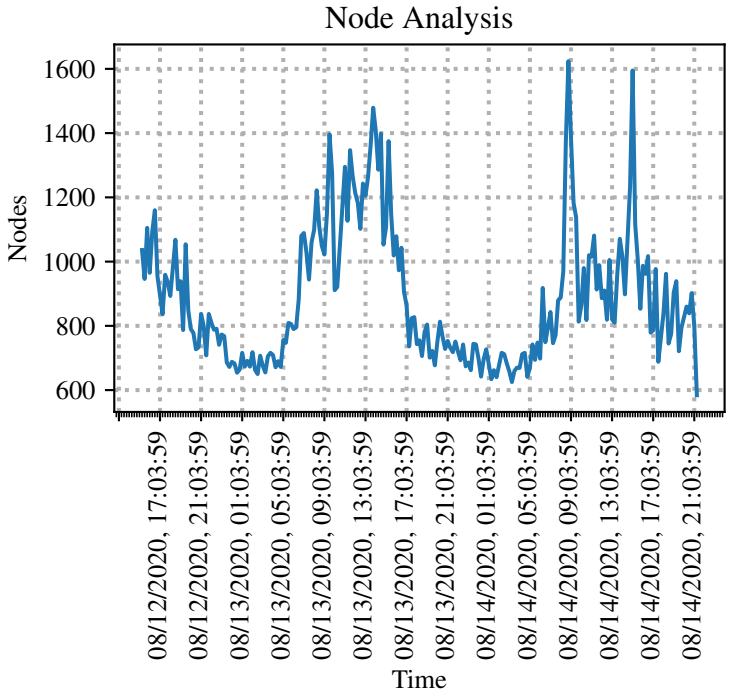


Figure 3.6: Node Progression over Time

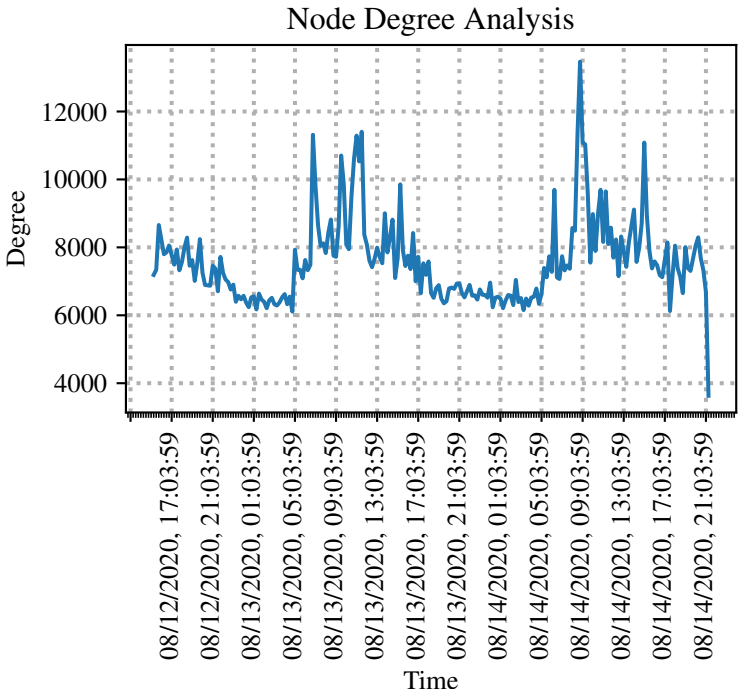


Figure 3.7: Node Degree Progression over Time

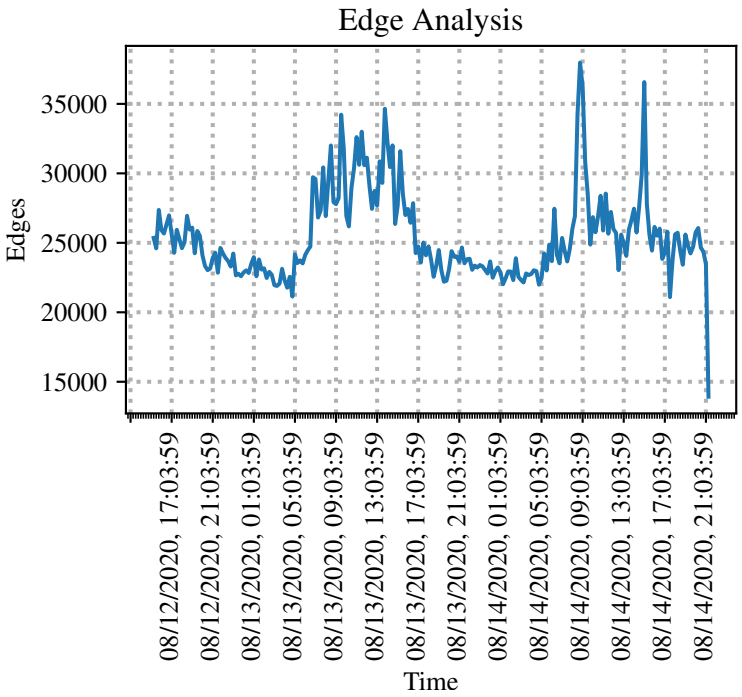


Figure 3.8: Edge Progression over Time

activity evidently motivates a surge in both nodes and edges and especially so at times where usually most of the (computing) work is done at the faculty chair. This too means that we should expect a similar pattern in the distribution of the respective node degrees which is indeed the case as can be seen in above Figure 3.7, where we have the maximal node degree at the first spike on August 14. Furthermore, as the number of edges increases and is in general significantly larger than the number of nodes - recall the aforementioned node degree discussion - we expect a very low graph density overall. This is indeed the case as depicted in below Figure 3.9.

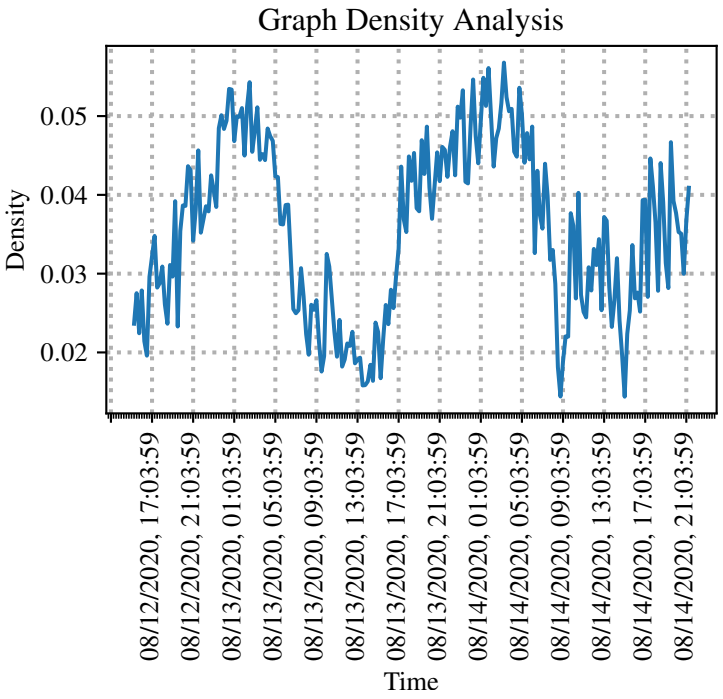


Figure 3.9: Graph Density Progression over Time

3.2.6 Traffic and Packet Analysis

Above pattern however is not necessarily ported over to the analysis of the network traffic and the corresponding number of sent packets. In fact, in both metrics we cannot observe a significant covariance between traffic and packets and above node and edge structure, which is also implied in the previously conducted general analysis where we compared metric averages for day and night times, for which the accumulated traffic stayed roughly the same in size. Nevertheless though, traffic and packet progression over time do resemble a similar pattern between themselves - as expected - as they are both closely connected to each other. We can observe this by inspecting below Figures 3.10 and 3.11.

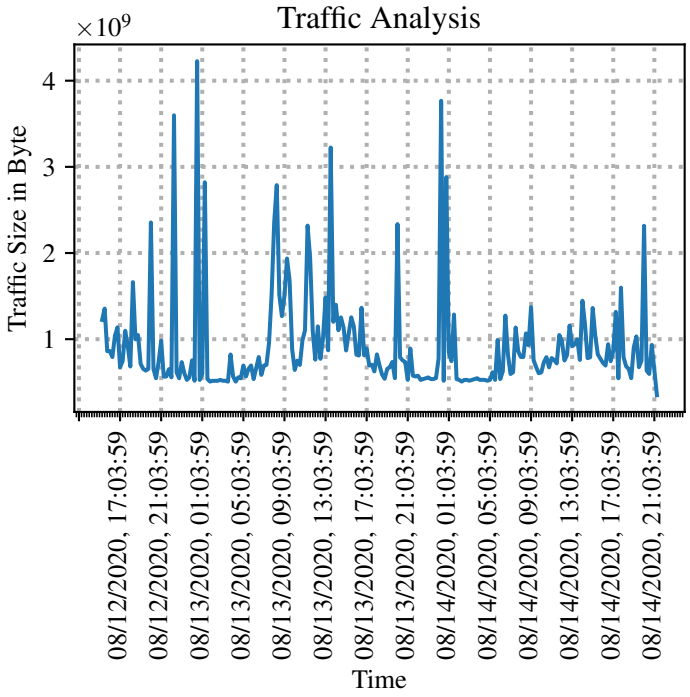


Figure 3.10: Traffic Progression over Time

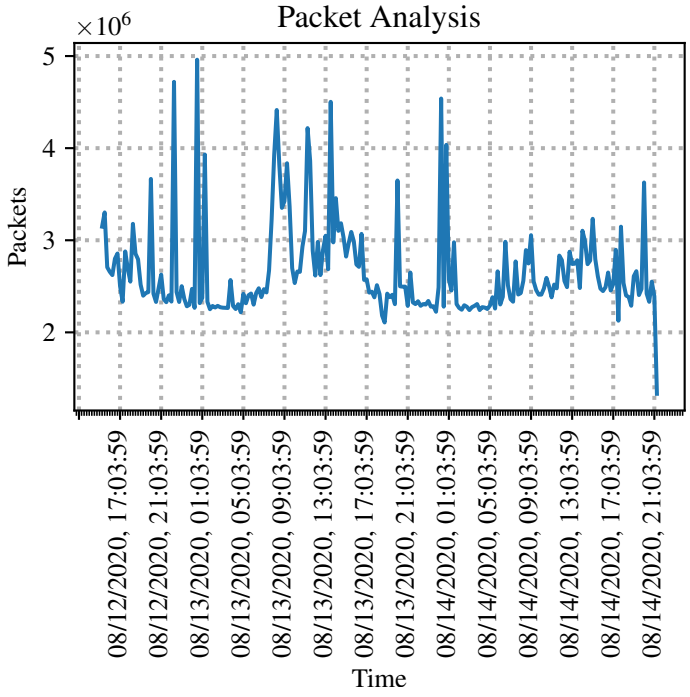


Figure 3.11: Sent Packets Progression over Time

3.3 PPL Framework Selection

In order to efficiently implement the SBM model, PPLs were considered. Amongst those, *TensorFlow Probability* (TFP), *Pyro* and *NumPyro* were further investigated. Eventually, NumPyro was chosen over Pyro and TFP because of the reasons discussed in below sections.

3.3.1 Pyro

In a first analysis, Pyro was chosen over TFP as it promised to provide a very simple yet powerful structure as well as a smaller learning curve. Furthermore, TFP already provided an SBM implementation example in its *edward1* library [TKD+16], however for a deprecated version of *TensorFlow* (TF), only. Consequently, since TFP is built upon the mother framework TF, most of the compact and easy-to-use functions for SBM modeling in edward1 are not available for newer instances of TF anymore, as TFP has significantly changed its code base and edward's successor *edward2* [THM+18] within it has become more low-level. This lack of framework support ultimately disqualified the implementation approach using TFP as the learning curve for the newer framework version seemed to be significantly steeper as compared to before. Thus, Pyro appeared to provide a more high-level approach to probabilistic SBM modeling, especially so as it works with probabilistic plate structures and is also built on a comparable and - most importantly - supported mother framework (PyTorch).

However, even as Pyro seemed to be quite capable of implementing an SBM in the first place, the actual implementation led to several severe problems. In summary, a fully-functioning SBM model using solely the tools and methods provided by the Pyro PPL itself could not be obtained. The main issues that hindered the implementation shall thus be briefly explained in this section.

Enumeration in Pyro Models
Following the implementation examples of the Pyro documentation, inference using a discrete latent variable model - as defined by the SBM - is usually implemented using enumeration strategies. In this case, sampling is not random, but rather realized as a full enumeration of the latent variables. The applied inference algorithms can then sum out the enumerated variable values [BCJ+19] as done in sum-product inference or belief propagation approaches [KF09]. As such, enumeration represents an efficient structure for probabilistic inference. However, while this might be easy to realize for simpler probabilistic models with few latent variables, for the SBM we run into two different types of problems. Firstly, the SBM inference model is realized - following the suggestions in the tutorials and guides - in a plated model structure where each adjacency matrix entry is treated as an enumerable variable. The plated structure (including nested plates) already realizes a simplification regarding dependencies in the model, but is ultimately only able to enumerate 25 variables in CUDA on a single PyTorch tensor. Regarding the real-world data we intend to apply the model to, the limited dimensionality of 25 variables is quickly exceeded. In order to overcome this problem, a different model structure (e.g. using a markov plate)

is needed [BCJ+19]. Secondly, even if applying an adapted model structure, enumeration for the SBM adjacency matrix model relies on a Pyro specific indexing using the primitive *Vindex*. This method however leads to an indexing of the adjacency matrix that causes an exponential growth in complexity when enumerating. In other terms, enumeration theoretically works, but the memory allocation is simply to high and thus unbearable for the network data and even for the simple karate club data set. Once again, while enumeration might work for other models, the structure of the adjacency matrix model inside the SBM simply cannot be efficiently realized using Pyro's enumeration strategies. Here, we do not blame the general Pyro approach, but we rather see this as a structural incompatibility to our model definition.

High Variance using SVI
In general, SVI aims to maximize the log evidence of the defined probabilistic model by taking stochastic gradient steps [BCJ+19]. In order to do this, unbiased estimates of the ELBO gradients are needed. This becomes particularly difficult w.r.t. expectation values of arbitrary and non-continuous (discrete) Pyro models. The reason why is because of inter-dependencies between variational parameter sets of the outer and inner expectation expressions in the mathematical computation of the ELBO [SHWA15]. In some cases - most prominently for continuous distributions such as the normal distribution - reparameterization can be used to eliminate said dependencies [SVI].

However, for many variational distribution families, reparameterization is not possible. This is often the case for custom models and most importantly for discrete distributions which we have in the definition of the SBM. In order to eliminate the dependencies nevertheless, the gradient needs to be rewritten to eventually take the form of a so-called reinforce/likelihood ratio estimator [SHWA15]. This way, we can ensure unbiased estimates. In order to do so, surrogate objective functions are used which are specifically modified to these requirements.

The so-obtained variational distribution however leads to the estimator having a rather high variance due to the discrete nature of the underlying problem. In some cases, it becomes so significant, ultimately rendering the estimator useless. In order to mitigate the resulting high variance issues, Pyro generally offers two solutions [BCJ+19]:

1. Reducing Variance via Dependency Structure

   Different terms in the cost function have different dependencies on the random latent variables. In order to leverage these out and thus decrease variance, terms are removed in the cost function that are specifically not downstream w.r.t. the dependency structure of the model. This can be done as these terms have a zero expectation, anyways. Such a term reduction is referred to as Rao-Blackwellization [SHWA15]. In Pyro, dependency structure tracking is taken care of by using SVI's *TraceGraph_ELBO* loss function. The dependencies are then being tracked within the execution traces of the model, which leads to the construction of a reduced surrogate objective function. However, this requires increased computations [BCJ+19].

2. Reducing Variance via Data-Dependent Baselines

   This method inverts above strategy and instead of removing unnecessary terms rather adds more (zero-expectation) terms to the cost function. These are specifically chosen such that they minimize the model variance in a control variate strategy [SHWA15]. In order to do so, the terms are specified using a baseline such as decaying average and neural baselines [BCJ+19]. For some model variants, this method has shown to reduce variance.

Even though both above approaches were incorporated in the SBM model definition, SVI inference did not lead to a satisfactory result, even for the simple karate club data set. In particular, variance could only be slightly reduced in the sense that fluctuations between the iterative inference steps were not as high anymore. In this regard, variance reduction merely resembled a less significant noise reduction in the loss function and the model still proceeded to converge to a uniform distribution for the community associations. Therefore, variance continued to persist in the model regardless of which data was being worked with. This important observation eventually once again seems to confirm that Pyro's probabilistic inference strategy might not be compatible to the SBM model structure.

Adapting the SBM to a Gaussian Mixture Model

As discussed above, Pyro's SVI approach seemed to be incompatible with the exact model architecture of the SBM, ultimately continuing to incorporate a high variance during inference. However, as SBMs strongly resemble the character of *Gaussian Mixture Models* (GMMs), the already provided GMM implementation example in Pyro's documentation was used as a motivation to adapt the SBM to its structure. By doing so, we hoped to see an improvement w.r.t. variance and general inference without completely changing the nature of the SBM. To this, we split the model into a more traditional parameter learning part for the stochastic block matrix and into a community association inference part as we observed that membership inference worked well if and only if the stochastic block matrix was already initialized properly. Thus, various approaches were considered to learn the stochastic block matrix parameters using the tools provided by Pyro, however without much success as Pyro's strengths lie in inference and not in parameter learning. For example, different initialization methods using MAP approaches were considered, the ELBO loss function was altered to above less noise-susceptible *TraceGraph_ELBO* and variations in multinomial/categorical realizations of the SBM were implemented. Eventually, neither approach yielded a desired improvement.

In summary, a closed-form solution for the SBM model using Pyro's SVI inference could not be implemented due to various reasons including strucutral bounds of the PPL framework, high variance during inference and a general incompatibility of the SBM model architecture to the structural probabilistic plate realization which was initially hoped to be a great advantage of Pyro. However, model implementation using NumPyro [PPJ19], the successor to the Pyro framework, was suggested by a Pyro developer and is addressed in the following section.

3.3.2 NumPyro

NumPyro can be seen as the evolution of Pyro to an improved approach to probabilistic programming using NumPy and Google JAX [BFH+18] instead of PyTorch. As such, NumPyro provides a NumPy backend for its predecessor Pyro and pushes its autograd and *just-in-time* (JIT) compilations optionally to the GPU, TPU and CPU using Google JAX. By doing so, NumPyro aims to build upon a faster and more future-proof framework for automatic differentiation and JIT compilation.

In general, NumPyro is designed to not differ significantly from Pyro w.r.t. its core syntax and object oriented structures. It thus adds additional features and algorithms to Pyro's primitives, inference approaches and distributions. As such, many approximate inference solutions such as MCMC using various sampling methods have been incorporated.

In order to implement the SBM using NumPyro, above discussed MCMC inference approach using a discrete *Hamiltonian Monte Carlo* (HMC) kernel was chosen as SVI could not be considered for the reasons stated in the previous section. The particular choice of this HMC kernel was motivated by the additional Metropolis updating which it performs on the discrete latent sites of the SBM model. Although being an approximate inference approach as compared to SVI, a working solution was implemented for both the simplified toy network and karate club data set. Note that neither of both could be realized using Pyro's SVI.

Consequently, only the SBM implementation using NumPyro can be considered for a subsequent comparison with the proprietary NES realization of the SBM. However, it is to be noted though that NumPyro is still in a development state and thus not fully stable. Furthermore, the working SBM examples could only be implemented for a CPU based computation scenario as NumPyro failed to enable GPU computations on our machine due to incompatibilities between the Google JAX framework and NVIDIA's CUDA toolkit. Thus, a comparison can only be reasonable for a CPU based inference scenario.

3.3.3 NES Alternative

The NES implementation that is pursued in this work relies on the theoretical foundations which were elaborated in [WSPS08] and [SHC+17] and briefly summarized in above Chapter 2. The corresponding realization of the optimization algorithms and its corresponding probabilistic distributions is done using PyTorch tensor structures similar to the fashion followed in Pyro. This does not only enable us access to the large library of PyTorch's probabilistic features but also allows us to perform computations on a GPU as PyTorch tensors can be natively transformed into GPU tensors. The alternative that this exact NES implementation thus offers compared to above NumPyro implementation is that we can not only compare their respective CPU performances but also perform inference to a larger data set on a GPU which could not be realized for the NumPyro case because of above discussed incompatibility issues.

Chapter 4

Evaluation

In this chapter, the performance and community detection results of the NumPyro MCMC and NES inference approach of Chapter 3 are discussed. First, community detection of both variants is compared using the simple benchmark data sets introduced in the previous chapter. Second, NES inference is applied to the real-world communication network data including a grid search, parameter tuning and subsequent loss analysis where a suitable number of communities has been chosen. The chapter concludes with a summary of the results.

4.1 Benchmark Data

As thoroughly discussed in Chapter 3, NumPyro's MCMC inference approach could not be implemented with a GPU support. Thus, a comparison between the PPL framework and the proprietary NES inference realization is conducted for a CPU based scenario only. To this, the performance is compared by means of computation times for applying both methods to the karate club and simplified toy network data, respectively.

4.1.1 Setup

To be able to fairly compare both inference methods, an initial parameter tuning has to be conducted in order to obtain the respective set of parameter values for which the inference algorithms converge with a minimal time. Please note that the number of communities K is already known a priori for these example sets and thus evidently not optimized in this case.

NumPyro MCMC
The set of variable parameters for the NumPyro framework consists of only two configurables:

1. Number of warmup steps n_{warmup}
   Denotes the number of samples to be drawn for the warmup phase. As detailed in Chapter 2, MCMC is a random sampling technique that suffers from the burn in effect. As such, it requires a warmup phase only after which drawn samples can be considered for inference purposes.

2. Number of samples n_{samples}
   Denotes the total number of samples to be drawn after the warmup phase.

In order to find above optimal (time minimal) model parameters n_{warmup} and n_{samples}, a DoE has been conducted where both parameters are varied between ranges of 10 and 3.000 in a full factorial fashion, that is every combination of those parameter values is investigated. To this, the upper bound was specifically chosen to be 3.000 as the working example has been obtained for both data sets with $n_{\text{warmup}} = 1.500$ and $n_{\text{samples}} = 2.500$. Furthermore, the full factorial DoE might indicate a ratio between both parameters.

After analyzing the obtained results of the DoE, we see that indeed a certain rule of thumb can be defined for the ratio of both parameters. In general, group memberships are successfully inferred when choosing n_{warmup} to be at least 1/10-th the size of n_{samples}. By applying this rule, we denote the optimal parameter set for either benchmark data set in Table 4.1.

Data Set	n_{warmup}	n_{samples}
Karate Club	10	10
Toy Network	10	100

Table 4.1: Time Minimal NumPyro MCMC Parameter Values for both Benchmark Data Sets.

Interestingly enough, the ratio can be chosen 1:1 for the karate club data set. Additionally, no further improvements were noticed when n_{samples} exceeded the number of edges in this scenario.

NES
Analogously to above NumPyro framework, we state the configurables for the NES inference approach:

1. Population size p
   Denotes the number of equivalent models with different parameter values that are compared to each other in a batch during each generation.

2. Learning rate r
   Denotes the step size at each iteration and corresponds to the learning rate in other machine learning settings.

3. Number of iterations n_{iters}
   Denotes the number of iteration steps that are to be performed during optimization.

Similarly to above NumPyro case, we apply a full factorial DoE and define the following parameter ranges based on the parameter values $p = 10.000$, $r = 1$ and $n_{\text{iters}} = 600$ from the working example.

$$\begin{aligned} p &\in [10, 100, 1.000, \ldots, 10.000, \ldots, 100.000] \\ r &\in [0.1, 0.5, 1, 1.5, 2] \\ n_{\text{iters}} &\in [10, 50, 100, 250, 500, 1.000] \end{aligned}$$

After analyzing the resulting group memberships, we obtain the time minimal parameters in Table 4.2 for each data set, respectively.

Data Set	p	r	n_{iters}
Karate Club	10	1	100
Toy Network	5	1	100

Table 4.2: Time Minimal NES Parameter Values for both Benchmark Data Sets.

From the conducted DoE analysis, we follow that the learning rate r does not significantly contribute to the model behavior and the resulting membership inference and is therefore left at $r = 1$. Thus, the population size p and the number of performed iterations n_{iters} are of a greater importance w.r.t. the parameter optimization. Furthermore, membership inference converges to a uniform distribution when decreasing p as fewer parallel models are evaluated in each iteration batch. From this observation though, a general rule of thumb cannot be constructed. While decreasing p to the values depicted in Table 4.2 might work for the underlying data, the usual notion would be to increase p for a larger amount of data as will become clear when working with the real-world communication network data.

4.1.2 Performance Analysis

With the obtained time minimal parameter values in the previous section, the actual computation times for each data set and each inference approach can now be further investigated. Their resulting numerical values are given in below Table 4.3. They represent the mean duration times until convergence, averaged for a total number of 10 inference runs. Note that the computations were performed on an Intel Core i7-7820X CPU, clocked at 3.6 GHz. Additionally, since MCMC and NES perform several iterations per time instant, the respective iterations per second are given for either approach.

Inference Approach	Data Set	Mean Duration [s]	Mean Iterations/s
NumPyro MCMC	Karate Club	4.4	4.7
	Toy Network	4.2	27
NES	Karate Club	0.07	479
	Toy Network	0.03	1195

Table 4.3: CPU Computation Times for the Benchmark Data Sets

From above CPU computation times we can conclude that NES seems to considerably outperform NumPyro's MCMC approach. In general, NES computation times only represent a small fraction as compared to NumPyro's statistics. Regarding the karate club data set for example, NumPyro's MCMC inference was around 63 times slower than NES. However, it has to be noted that a comparison solely based on CPU computation times does not seem very representative for real time applications. As such, we cannot evaluate the overall performance gain in detail. Nevertheless, the CPU comparison leaves to conclude that NES might show similar performance advantages when operating on a GPU.

4.1.3 Inference Results

After having analyzed the computational performance statistics for either inference approach, the resulting community detection results shall be visualized in this section. Note that both methods necessarily lead to the same group memberships as we already know the outcomes of the benchmark data sets for which the models have been validated.

The respective graphs are depicted in below Figures 4.1 and 4.2 for each data set, respectively. Group membership is implied using a red or blue coloring of the respective nodes. Please note that the standard SBM model as implemented here is prone to both a *label switching* and a preference for *higher-* or *lower-splits*. As can be seen from below karate club inference result in Figure 4.2, the model prefers the selection of higher degree nodes over lower ones and constructs the group accordingly. This behavior is ascribed to the specific nature of the standard SBM model and can be adjusted using degree-corrected and other variants of the SBM as discussed in Chapter 2 [DKMZ11].

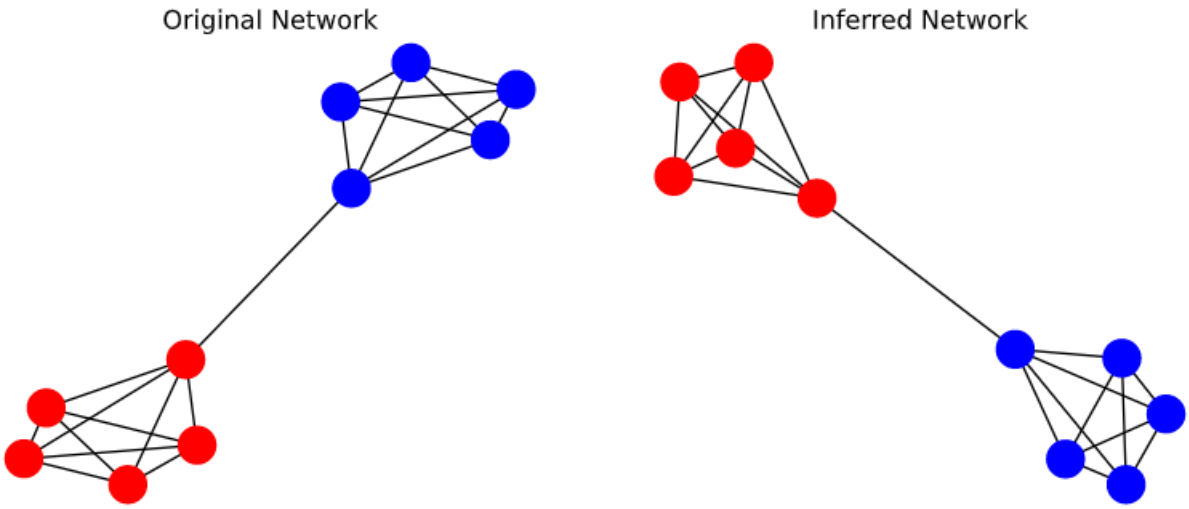


Figure 4.1: Artificial Toy Network Community Detection Result

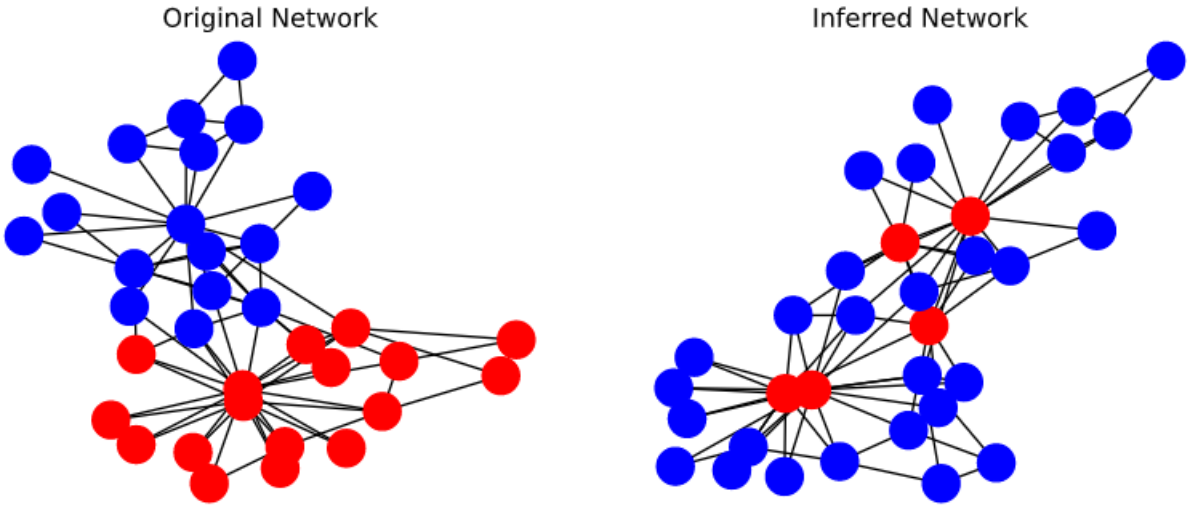


Figure 4.2: Karate Club Community Detection Result

4.2 Communication Network Data

In order to eventually perform community detection using NES on the real-world data graph of the communication network analyzed in Chapter 3, a corresponding hyperparameter tuning has to be conducted first to obtain a reasonable number of communities K that describes the network split best. Then, subsequent inference can be performed in full length to obtain the best detection result by minimizing the specified loss function.

4.2.1 Hyperparameter Tuning

As discussed in above benchmark data comparison for the karate club and toy network example, NES parameters are mostly sensitive w.r.t. the population size p and the number of iterations n_{iters} that are to be performed on the respective data set. Additionally, since we do not know how many possible clusters the real-world network incorporates, the corresponding number of communities K has to be also taken into account. Evidently, inference is only reasonable if we can specify K beforehand. However, the exact identification of this hyperparameter is not well-defined. Some approaches as described in [Yan16] argue to evaluate information criteria such as Akaike (AIC) and Bayes (BIC). To this, we perform a DoE in which we variate p, n_{iters} and K in a similar fashion as described in the previous section and calculate the corresponding AIC to make a statistical decision for a suitable number of communities K.

The respective AIC [Aka74] can be calculated as follows

$$AIC = 2 \cdot k - 2 \cdot ln(\hat{L}) \tag{4.1}$$

where k and $\hat{L}$ denote the corresponding number of parameters to be estimated and maximum log-likelihood, respectively. In our case of group membership inference, k thus translates to

$$k = K \cdot n_{\text{nodes}} \tag{4.2}$$

where n_{nodes} describes the number of network nodes since we aim to estimate a corresponding group membership probability for each node in the network.

After performing above described DoE, we fix the relevant model parameters p and n_{iters} to

$$p = 10.000$$
$$n_{iters} = 1.000$$

as those values seemed to provide good initial inference results for several variations of K. This can be examined in below Figure 4.3 where we observe that the loss functions for each value of K decrease steadily. Therefore, we subsequently proceed to calculate the respective AIC values using above fixed p and n_{iters} and variate K for a number of communities between 2 and 10. The corresponding AIC results are depicted in Figure 4.4.

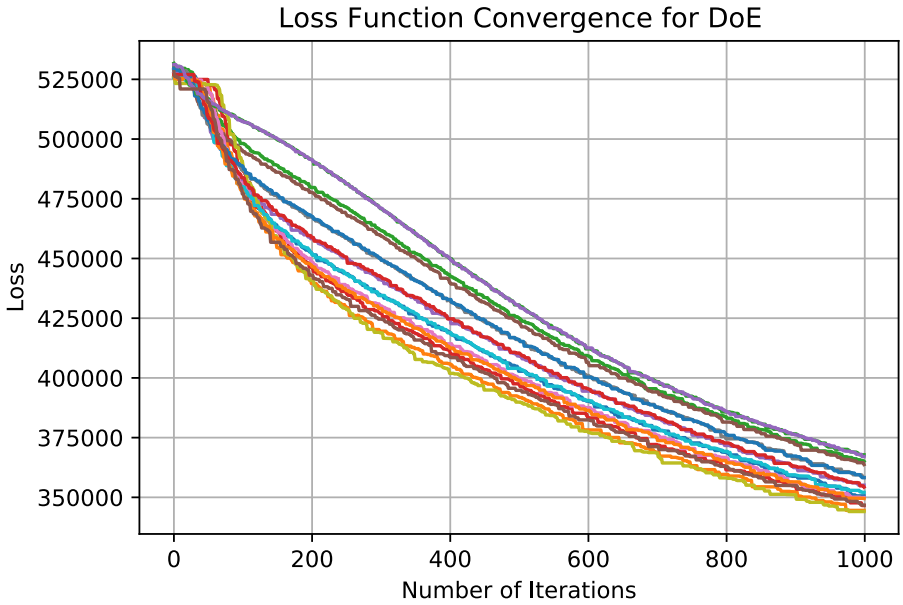


Figure 4.3: Loss Function Convergence for Different Values of K with $p = 10.000$ and $n_{iters} = 1.000$.

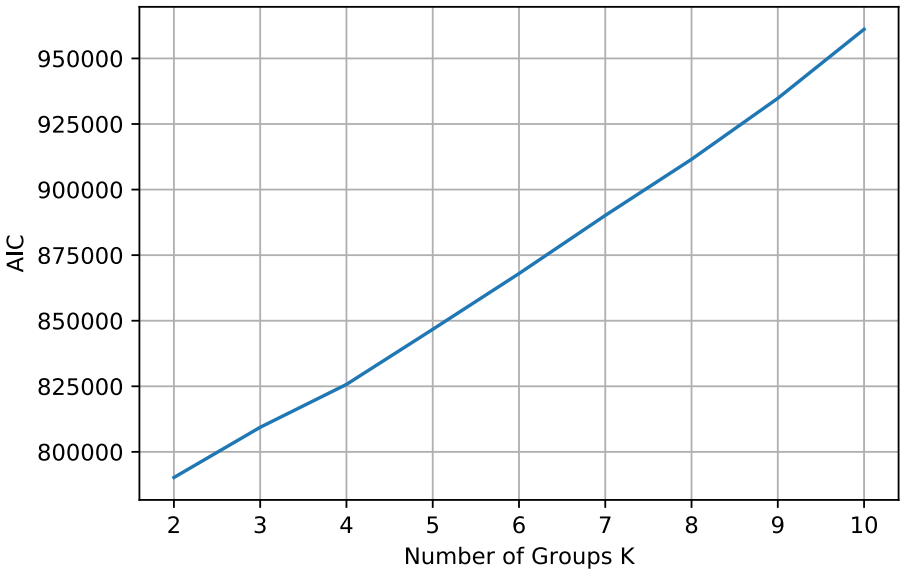


Figure 4.4: AIC Values for Different Numbers of Community Groups K

The best parameter set corresponds to the lowest AIC value which occurs for a community split with only two groups, that is $K = 2$. Thus, we concentrate on further investigations using the following set of hyperparameters:

$$\begin{aligned} K &= 2 \\ p &= 10.000 \\ n_{iters} &= 5.000 \end{aligned}$$

Note that we deliberately decide to increase the number of iterations n_{iters} in order to further minimize the loss function as compared to above DoE where the main goal was to obtain a reasonable value for K and not the best performing model.

4.2.2 Inference Results

Community detection using NES is now performed with the previously defined set of hyperparameters. In particular, the effect of increasing the number of iterations n_{iters} can be observed when comparing the corresponding AIC values to the previous ones where $n_{iters} = 1.000$. This can be seen in below Figure 4.5 where $K = 2_{1000}$ denotes the AIC for $K = 2$ and $n_{iters} = 1.000$, while all other AIC values refer to the respective K configuration with $n_{iters} = 5.000$. Indeed, we observe a significant decrease for higher n_{iters} which can be easily explained by examining the respective loss functions in Figures 4.6 and 4.7. Here it can be seen that the SBM with $K = 2$ and $n_{iters} = 5.000$ converges to a lower final loss due to a larger number of iterations for which more optimization steps can take place.

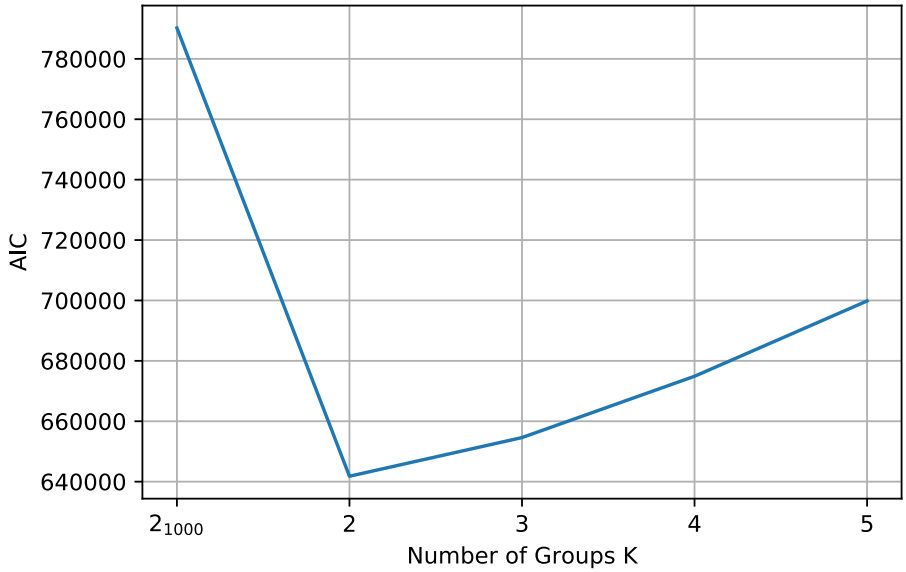


Figure 4.5: AIC values for $K = 2, 3, 4, 5$ with $n_{iters} = 5.000$. Additionally, the AIC value for $K = 2$ and $n_{iters} = 1.000$ denoted by $K = 2_{1000}$ is given as a comparison.

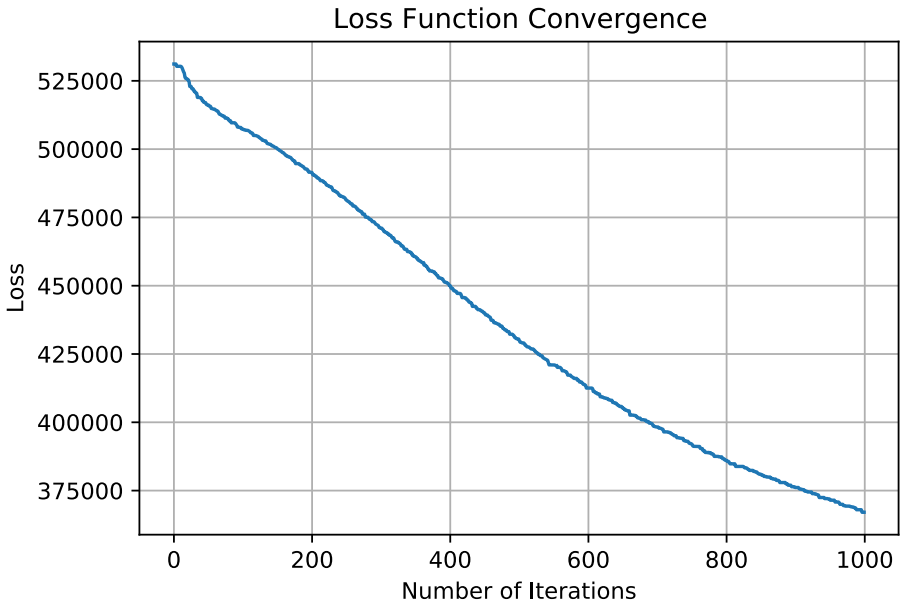


Figure 4.6: Loss Function Convergence for the SBM Setup with $K = 2$ and $n_{iters} = 1.000$.

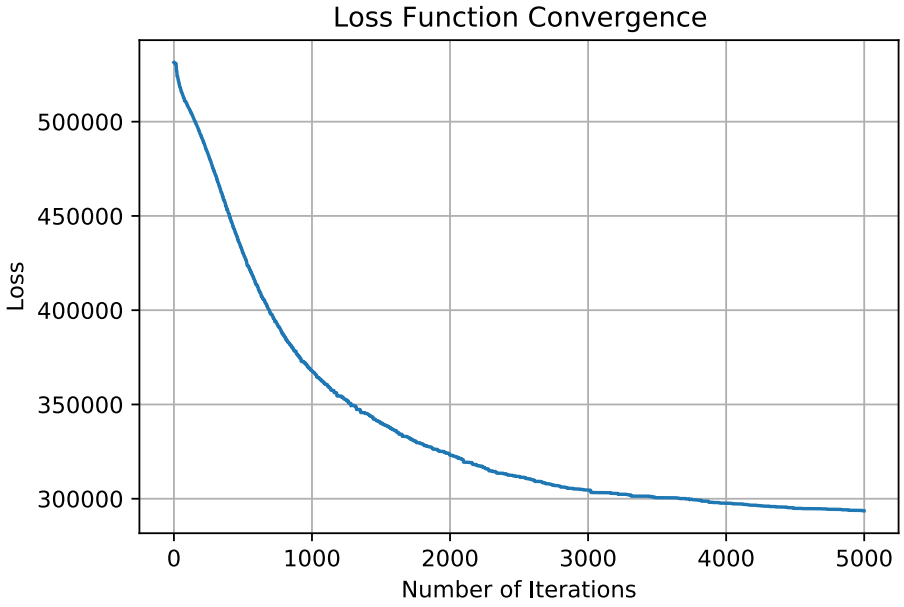


Figure 4.7: Loss Function Convergence for the SBM Setup with $K = 2$ and $n_{iters} = 5.000$.

Note that the loss function of the NES implementation pursued here is defined as the negative inverse of the log-likelihood. As such, convergence of the loss function implies convergence of the log-likelihood function. Once convergence is reached - as is done for $n_{iters} = 5.000$ in this case - the model has consequently converged to its optimal output for the underlying set of hyperparameters. Thus, we interpret above convergence in Figure 4.7 as a successful model application to community detection. This is especially the case as we have already defined the best value for K in the previous section.

Furthermore, since we deal with a large-scale network with 13.651 nodes and 67.690 edges, it is reasonable to investigate the respective time progression of the algorithm as depicted in Figure 4.8. In particular, we are interested in the computation time per iteration. From below figure we clearly observe that the NES algorithm exhibits a linear time complexity w.r.t. each subsequent iteration. This behavior is highly desirable for most inference algorithms. The corresponding time difference between the respective iteration steps can be easily determined and is in this case $\triangle t \approx 13s$. Moreover, we can determine the total computation time for the respective parameter setup defined above to be around 17.85 hrs. Thus, computation still takes a long time even with the fast NES approach which is mainly contributed to the sheer size of the network that is being worked with.

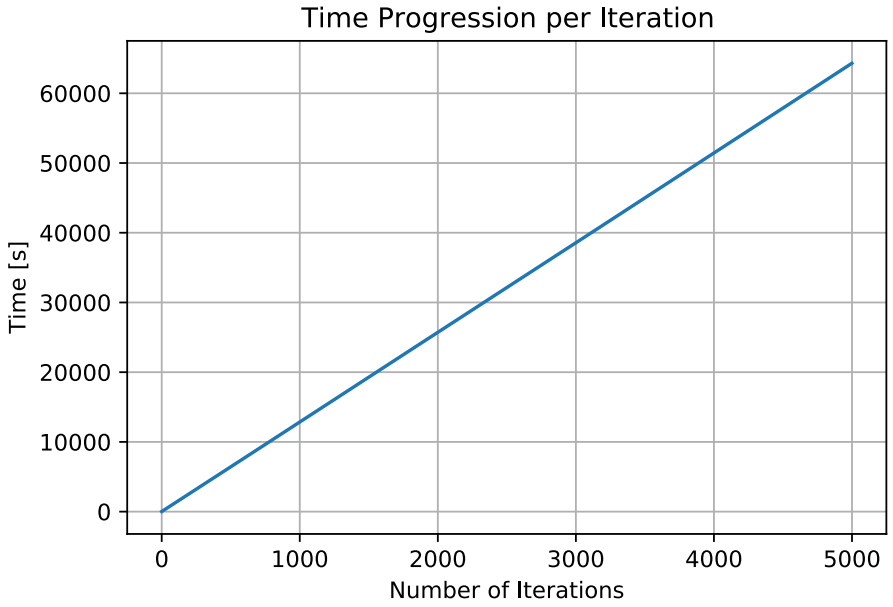


Figure 4.8: Time Complexity Analysis for the NES Setup

Chapter 5

Conclusion

The goal of this work was threefold.

First, an overview of some core aspects of stochastic block modeling and its corresponding inference algorithms was given in order to introduce the unfamiliar reader to the subject of community detection using SBMs and its corresponding use case for network traffic classification tasks, for which a thorough analysis of a real-world communication network was performed in Chapter 3. Furthermore, various PPL frameworks were analyzed which incorporate most of the common inference algorithms that can be applied to the SBM. To this, edward1, Pyro and NumPyro were further investigated as suitable candidates. However, after an extensive framework analysis following several SBM implementation attempts, edward1 and Pyro were eventually discarded due to a lack of platform support and several model compatibility issues, respectively. Thus, only NumPyro was left to be considered for a successful SBM implementation which was achieved using MCMC inference for two independent benchmark data sets. Furthermore, a novel yet proprietary approach to SBM inference using NES was thoroughly investigated and a corresponding working example was implemented for the same set of benchmark data sets.

Second, a comparison between the off-the-shelf NumPyro framework and the proprietary NES solution was to be conducted. However, due to the inability to run NumPyro on a GPU, a subsequent comparison could only be made on a CPU basis for the simple benchmark data sets of the working examples. To this, a parameter tuning was performed to create a common ground basis for which both inference approaches converged in a minimal time. This was done in order to be able to fairly compare the two different approaches for which the NES implementation eventually outperformed NumPyro's MCMC solution by far.

Third, NES was applied to the communication network data in order to perform a subsequent community detection and thus obtain a further insight into the network dynamics. To this, first a hyperparameter tuning was performed in order to identify a model optimal set of parameters including a reasonable number of community splits K. Then, the model

was re-estimated using a higher number of iterations for better model convergence, resulting in a doubly split network. Note that the computations were performed on a GPU in contrast to the capabilities of the NumPyro framework.

In conclusion, by comparing the NumPyro MCMC and NES approach, the question whether off-the-shelf PPLs can be used for automated network traffic classification cannot be directly answered. This is mainly due to the fact that the NumPyro approach could not be realized for a GPU based scenario and therefore was not applied to the large-scale communication network. Thus, a direct performance comparison was only possible for a CPU operation and only for the insignificant benchmark data sets. Furthermore, only NumPyro qualified from the selection of above three PPL frameworks to be considered for further implementations, as thoroughly discussed in Chapter 3. However, indirectly we can draw the conclusion that PPLs definitely do not represent an off-the-shelf solution for automated community detection tasks. Following the implementation approaches and the corresponding encountered issues, stochastic block modeling could not be efficiently automated using either of above selected approaches. Although being a commonly applied model and especially in probabilistic programming, no working SBM examples were found for any of the toolboxes except for edward1 which cannot be considered due to its deprecated dependencies. Furthermore, since the proprietary NES approach not only outperformed NumPyro's MCMC solution regarding the CPU based scenario, but was also easier to implement and in the end also able to efficiently perform community detection on the large-scale communication network, the question further arises whether PPLs should be considered in the first place as compared to black box optimization approaches such as NES.

Regarding above remarks, we thus answer the main question that was investigated in this work as follows. For the comparison of the selected PPL frameworks with the NES approach, no improvement but rather a deterioration was concluded. Consequently, we cannot suggest any of the PPLs considered here to be applied for community detection purposes and subsequent network traffic classification. Nevertheless, the investigated PPLs only represent a portion of available frameworks. In particular, TFP was not further investigated due to the fact that both the model and the inference algorithm needed to be manually implemented. Thus, future work on this issue could investigate whether SBMs can be efficiently realized using the methods and tools provided by TFP. Furthermore, more attention should be given to black box optimization principles as they - at least in this work - show a significant increase in optimization performance and are easy to maintain for network traffic classification tasks.

Bibliography

[Abb16] E. Abbe. Community detection and the stochastic block model. 2016.

[Aka74] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[AR+14] Andry Alamsyah, Budi Rahardjo, et al. Community detection methods in social network analysis. *Advanced Science Letters*, 20(1):250–253, 2014.

[Arb] Aryan Arbabi. Bayesian inference and mcmc csc412.

[BCJ+19] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.

[BFH+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[Blea] David Blei. Probabilistic topic models.

[Bleb] David Blei. Variational inference.

[Cla] Aaron Clauset. Network analysis and modeling csci 5352, fall 2017.

[DKMZ11] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106, 2011.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[IKF+11] Marios Iliofotou, Hyun-chul Kim, Michalis Faloutsos, Michael Mitzenmacher, Prashanth Pappu, and George Varghese. Graption: A graph-based p2p traffic classification framework for the internet backbone. *Computer Networks*, 55(8):1909–1920, 2011.

[Kap08] Todd D Kaplan. *Evolving software traders and detecting community structure in financial markets*. University of New Mexico, 2008.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[NGSG11] Tejaswini Narayanan, Merril Gersten, Shankar Subramaniam, and Ananth Grama. Modularity detection in protein-protein interaction networks. *BMC research notes*, 4(1):1–6, 2011.

[PPJ19] Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

[S+92] James C Spall et al. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[Sal] Ruslan Salakhutdinov. Approximate inference using mcmc.

[SHC+17] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[SHWA15] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *arXiv preprint arXiv:1506.05254*, 2015.

[SVI] Pyro tutorial, svi part iii: Elbo gradient estimators. `https://pyro.ai/examples/svi_part_iii.html`. Accessed: 2020-06-17.

[THM+18] Dustin Tran, Matthew D. Hoffman, Dave Moore, Christopher Suter, Srinivas Vasudevan, Alexey Radul, Matthew Johnson, and Rif A. Saurous. Simple, distributed, and accelerated probabilistic programming. In *Neural Information Processing Systems*, 2018.

[TKD+16] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

[TPGK03] Godfrey Tan, Massimiliano Poletto, John V Guttag, and M Frans Kaashoek. Role classification of hosts within enterprise networks based on connection patterns. In *USENIX Annual Technical Conference, General Track*, pages 15–28, 2003.

[WSPS08] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computa-*

tion (IEEE World Congress on Computational Intelligence), pages 3381–3387. IEEE, 2008.

[Yan16] Xiaoran Yan. Bayesian model selection of stochastic block models. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 323–328. IEEE, 2016.