Estimation and model selection for graphical models under mixing conditions

Magno Tairone de Freitas Severino

PhD Thesis presented
to
Institute of Mathematics and Statistics
of
University of São Paulo
to
obtain the title
of
Doctor in Science

Program: Probability and Statistics

Advisor: Prof. Florencia Graciela Leonardi

This work was produced as part of the activities of the Research, Innovation and Dissemination Center for Neuromathematics (grant FAPESP 2013/07699-0). It also was financed in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

São Paulo, March 2024

Estimação e seleção de modelos para modelos gráficos sob condições de mixing

Magno Tairone de Freitas Severino

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Probabilidade e Estatística

Orientadora: Profa. Florencia Graciela Leonardi

Este trabalho foi produzido como parte das atividades do Centro de Pesquisa, Inovação e Disseminação em Neuromatemática (bolsa FAPESP 2013/07699-0). Também foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

São Paulo, março 2024

Estimation and model selection for graphical models under mixing conditions.

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 05/04/2024. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profª. Drª. Florencia Graciela Leonardi (orientadora) - IME-USP
- Profª. Drª. Aline Duarte de Oliveira - IME-USP
- Prof. Dr. Daniel Yasumasa Takahashi - UFRN
- Prof. Dr. Guilherme Ost de Aguiar - IM-UFRJ
- Profª. Drª. Mariela Sued - IC-UBA

Dedico esta tese à memória de minha avó Therezinha, cujo carinho e incentivo moldaram meu caminho acadêmico.

Agradecimentos

Meus sinceros agradecimentos à Profa. Florencia pela sua orientação e apoio durante toda essa jornada. Sua maneira de me incentivar e motivar me deram força para realizar esta tese e todos os outros projetos que desenvolvemos juntos nos últimos anos, e não foram poucos!

Sou muito grato aos meus pais, Magno e Scheila, que sempre acreditaram que a educação transforma. Sem o seu apoio incondicional, eu jamais teria chegado aqui.

Agradeço aos meus avós Francisco e Therezinha (in memoriam), que sempre torceram pelo meu sucesso nos estudos.

Aos amigos que fiz no IME, dentro da sala 154B, em especial Cátia, Felipe, Gabriela, Joan, Luísa e Luiza, com quem compartilhei minhas incertezas e inseguranças, mas também muitos momentos de alegria.

Agradeço ao Gustavo pelo companheirismo e suporte genuínos que foram essenciais para a conclusão desta jornada.

Agradeço também à CAPES, ao CNPq e à Fapesp pelo financiamento de parte deste trabalho.

“Eu sou o sonho dos meus pais
Que eram sonhos dos avós
Que eram sonhos dos meus ancestrais!”

Abstract

Severino, M. T. F. **Estimation and model selection for graphical models under mixing conditions**. PhD Thesis - Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

This thesis introduces a novel approach for estimating the graph of conditional dependencies in a random vector based on finite sample data. We define this approach as a global model selection criterion, which means optimizing a function across the entire set of potential graphs, removing the need to estimate and combine individual neighborhoods as commonly proposed in the literature. Our results establish the strong convergence of this graph estimator, provided that the multivariate stochastic process satisfies a mixing condition. To the best of our knowledge, these results represent a pioneering demonstration of the consistency of a model selection criterion for Markov random fields on graphs when dealing with non-independent data. Additionally, we propose efficient algorithms for graph estimation and complement our theoretical results with simulation studies. To illustrate the practical applicability of our approach, we present two real-world examples: a study of the dependence structure among water flow measurements gauges located in the course of the São Francisco River in Brazil; and a daily stock market index performance analysis in order to identify the conditional dependence among the stock markets around the world.
Keywords: Model selection, regularized estimator, structure estimation, mixing processes.

Resumo

Severino, M. T. F. **Estimação e seleção de modelos para modelos gráficos sob condições de mixing**. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Esta tese apresenta uma nova abordagem para estimar o grafo de dependências condicionais em um vetor aleatório com base em dados de amostras finitas. Definimos essa abordagem como um critério global de seleção de modelos, o que significa otimizar uma função em todo o conjunto de possíveis grafos, eliminando a necessidade de estimar e combinar vizinhanças individuais como é comumente proposto na literatura. Nossos resultados estabelecem a convergência deste estimador, desde que o processo estocástico multivariado satisfaça uma condição de mistura. Até onde sabemos, esses resultados representam uma demonstração pioneira da consistência de um critério de seleção de modelos para campos aleatórios de Markov em grafos ao lidar com dados não independentes. Além disso, propomos algoritmos eficientes para a estimativa do grafo e complementamos nossos resultados teóricos com um estudo de simulação. Para ilustrar a aplicabilidade prática de nossa abordagem, apresentamos dois exemplos do mundo real: estudo da estrutura de dependência entre as medições de fluxo de água nos medidores localizados no curso do rio São Francisco, no Brasil; e análise do desempenho diário de índices de mercado de ações, com o objetivo de identificar a dependência condicional entre os diferentes mercados de ações ao redor do mundo.

Palavras-chave: seleção de modelos, estimador regularizado, estimação de estruturas de grafo, processos de mistura.

Contents

List of Figures viii

List of Tables xii

1 Introduction 1

2 Graph Estimator on Mixing Processes 5
- 2.1 Graphical Models . . . 5
- 2.2 Vector-Valued Mixing Processes . . . 7
- 2.3 Empirical Probabilities . . . 11
- 2.4 Regularized Maximum Pseudo-Likelihood Graph Estimator . . . 17

3 Algorithms for Computing the Estimator 26
- 3.1 Exact Algorithm . . . 27
- 3.2 Simulated Annealing Algorithm . . . 28
- 3.3 Stepwise Edge Selection Algorithm . . . 29
- 3.4 The `MixingGraph` R package . . . 30

4 Simulation Studies 31
- 4.1 Fixed Graph Scenario . . . 32
  - 4.1.1 Data Generation . . . 33
  - 4.1.2 Estimation . . . 35
  - 4.1.3 Overall Performance in Several Replications . . . 41
  - 4.1.4 The Choice of the Penalizing Constant c . . . 43
  - 4.1.5 Additional Scenario . . . 46
- 4.2 Random Graphs with Varying Edge Number . . . 47
  - 4.2.1 Data Generation . . . 47
  - 4.2.2 Estimation . . . 49
- 4.3 Comparative Analysis . . . 52

5 Applications 53
- 5.1 São Francisco River Data . . . 53
- 5.2 Stock Exchange Data . . . 58

6 Conclusion and Future Work **63**

A Theoretical complementary results **66**

B Simulations' complementary results **69**

B.1 Supplementary Information for Example 12 69

B.1.1 Results 70

B.2 Supplementary Information for Example 13 74

C R functions **76**

C.1 Estimator 76

C.2 Generating samples 77

Bibliography **83**

List of Figures

2.1 Representation of a realization of the process $\mathbf{X}$, with set of vertices $V = \{1, \ldots, d\}$ observed from time 1 to n. The subscript indicates the vertex, and the superscript indicates the time at which the observation was taken. The highlighted rectangle indicates the observed slice $x_{\{1,2\}}^{(2:3)}$. 8

2.2 Decomposition of the event $\{\widehat{G} \neq G^*\}$ in two cases: (a) G^* is strictly contained in G (non-overfitting), and (b) G^* is not a subset of G (non-underfitting). Furthermore, in the second case we consider that $\{G^* \setminus \widehat{G}\} \neq \emptyset$, otherwise it is similar to the first case. 19

3.1 An example of (a) a graph with five nodes, denoted as G, and two possible neighbors: (b) a neighboring graph of G, without the edge linking nodes 1 and 4, (c) another possible neighbor of graph G, now adding an edge between nodes 3 and 5. 28

4.1 Graph of Example 11, with vertices $X_1, \ldots, X_5$. For simplicity, instead of X_i the figure shows only the node index, i.e., i, $1 \leq i \leq 5$. 33

4.2 Scheme showing how the subsamples of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ were extracted from the initial chain of size 10,000. 35

4.3 The effect of the penalization term in the log-likelihood function in terms of the number of edges, considering the Exact algorithm $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$, in a sample of size 5,000. The dotted horizontal lines highlight the value of $H(G)$ when the number of edges in the estimated graph equals five, the value in the true graph. 36

4.4 Exact algorithm results for Example 11, considering several scenarios, namely $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing term $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. 37

4.5 Simulated annealing algorithm results for Example 11, considering several scenarios, with samples of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant $c \in \{0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. 38

4.6 Steps of the Backward Stepwise selection algorithm. Commencing with the full graph, it successively removes edges to increase the penalized pseudo-log-likelihood, stopping when further improvements become unattainable. These steps were obtained using a sample of size 5,000 and $c = 0.5$. 39
4.7 Backward Stepwise algorithm results for Example 11, considering several scenarios, with a sample of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. 39
4.8 Stages of the Forward Stepwise selection algorithm. Starting with the null graph, it sequentially incorporates edges that increase the penalized pseudo-log-likelihood, stopping when further increases are unattainable. These steps were obtained considering a sample of size 5,000 and $c = 0.5$. 40
4.9 Penalized log-likelihood function $H(G)$ in terms of the number of edges, considering a sample of size $N = 5{,}000$ and $c = 0.50$ for the Forward Stepwise algorithm. The highlighted orange dot represents the graph at which the maximum of this function is attained. 40
4.10 Forward Stepwise algorithm results for Example 11, considering several scenarios, namely $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. 41
4.11 Mean error metrics (underestimation, overestimation, and total error) for the algorithms considering ten samples of each size. Sizes considered are $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant value $c \in \{0.1, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. The color in the tile's background indicates the size of the error metric; the darker, the bigger. In general, small penalizing constant values tend to cause overestimation. Conversely, bigger c values tend to cause underestimation, especially when the sample size is small. 43
4.12 5-fold cross-validation error for Example 11, considering a sample of size 5,000 and set of penalizing constant $\{0.01, 0.05, 0.10, 0.15, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00, 1.25, 1.50, 2.00, 2.50\}$. 45
4.13 Graph of Example 12, with vertices $X_1, \ldots, X_8$. For simplicity, instead of X_i, the figure shows only the node index, i.e., i. 46
4.14 Randomly generated graphs with varying numbers of edges, considering a graph with five nodes. For each number of edges (from 1 to 10, displayed in columns), ten random graphs were generated (represented in rows). The node arrangement in the figure corresponds to that of Figure 4.1. 48
4.15 Outcomes of the Forward Stepwise algorithm for graph G_1 considering a range of penalization constant $c \in \{0.001, 0.010, 0.050, 0.100, 0.150, 0.200, 0.250\}$. . 49
4.16 Average error metrics outcomes of the Forward Stepwise algorithm. Each subfigure represents the result of a specific c value and displays the averaged errors as a function of the number of edges in the graph. 50

4.17 Effect of varying the penalizing constant c on the average error metrics outcomes from the Forward Stepwise algorithm. Each subfigure is the result given by graphs with a fixed number of edges, from 1 to 10. 51

5.1 (a) Geographical boundaries of Brazil and its state limits. The highlighted rectangle indicates the region where the São Francisco River is situated; (b) An enlarged view of the boxed area in (a), encompassing the São Francisco River. Gray points correspond to the ten streamflow gauges included in our analysis, numbered in ascending order from bottom to top. 54
5.2 Availability of measurements (gray spots) and data gaps (white spots) for each station in the SFR dataset. The vertical dashed black line delineates the time frame of this study, a period with reduced data gaps, spanning from January 1977 to January 2016. 55
5.3 Frequency of discrete levels within random variables representing discretized streamflow values from the original dataset. The x-axis corresponds to the random variables (stream flow gauges), while the y-axis displays the corresponding observation counts for each level. Numeric values inside each tile indicate the observation count for specific levels within the random variables. 56
5.4 5-fold cross-validation error as function of $c \in \{0.01, 0.10, 0.20, 0.30, 0.40, 0.50, 0.75, 1.00\}$ for the SFR data. The text box highlights the penalizing value at which the maximum is attained: 0.40 and 0.50. 56
5.5 The underlying graph estimated by our proposed model for the SFR, considering $c^* \in \{0.40, 0.50\}$. The station's arrangement mirrors the geographical locations shown in Figure 5.1 (b). 57
5.6 Stream flow volume measured at the ten stations in the SFR (logarithmic scale). 57
5.7 Working hours of the 15 stock exchanges on business days corresponding to the market indices under consideration. The time zone used as a reference is UTC+0. 59
5.8 Stock indices from May 18th 2010 to September 20th 2023. Each subfigure shows the indices grouped by continent. 60
5.9 5-fold cross-validation error as function of $c \in \{0.01, 0.05, 0.10, 0.12, 0.14, 0.16, 0.17, 0.18, 0.19, 0.20, 0.25, 0.30, 0.40, 0.50, 0.75, 1.00, 1.50, 2.00\}$ for the stock indices data. The text box highlights the penalizing value at which the maximum is attained: $c^* = 0.16$. 61
5.10 The underlying graph estimated by our proposed model for the stock indices data, considering $c^* = 0.16$. 62

B.1 Simulated Annealing algorithm results for Example 12, considering several scenarios, with samples of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$. The node structure resembles that of Figure 4.13. 71

B.2 Backward Stepwise algorithm results for Example 12, considering several scenarios, with sample of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$. The node structure resembles that of Figure 4.13. 72
B.3 Forward Stepwise algorithm results for Example 12, considering several scenarios, with sample of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$. The node structure resembles that of Figure 4.13. 72
B.4 Underestimate error, overestimate error, and total error of algorithms considered in the estimation of Example 12. Mean error metrics (underestimation, overestimation, and total error) for the algorithms considering ten samples of each size in Example 12. Sizes considered are $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ and penalizing constant value $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.0, 2\}$. The color in the tile's background indicates the size of the error metric; the darker, the bigger. In general, small penalizing constant values tend to cause overestimation. Conversely, bigger c values tend to cause underestimation, especially when the sample size is small. 73
B.5 Average error metrics outcomes of the Backward Stepwise algorithm for Example 12. Each sub-figure represents the result of a specific c value and displays the averaged errors as a function of the number of edges in the graph. 74
B.6 Effect of varying the penalizing constant c on the average error metrics outcomes from the Backward Stepwise algorithm for Example 12. Each sub-figure is the result given by graphs with a fixed number of edges, from 1 to 10. . . 75

List of Tables

4.1 Marginal distribution of X_3 for Example 11. 34
4.2 Conditional distribution of $X_1|X_3$ for Example 11. 34
4.3 Conditional distribution of $X_4|X_3$ for Example 11. 34
4.4 Conditional distribution of $X_5|X_3$ for Example 11. 34
4.5 Conditional distribution of $X_2|X_1, X_3$ for Example 11. 34
4.6 Marginal distribution of X_1 in graph G_1 of Example 13. 49
4.7 Conditional distribution of $X_2|X_5$ in graph G_1 of Example 13. 49
4.8 Conditional distribution of $X_3|X_4$ in graph G_1 of Example 13. 49
4.9 Conditional distribution of $X_4|X_3$ in graph G_1 of Example 13. 49
4.10 Conditional distribution of $X_5|X_2$ in graph G_1 of Example 13. 49

Chapter 1

Introduction

A *graph* is an ordered pair $G = (V, E)$, where V is the set of vertices (or nodes) and $E \subseteq V \times V$ is the set of edges connecting pairs of vertices. We say a graph is *undirected* if $(v_i, v_j) \in E$ implies that $(v_j, v_i) \in E$, $\forall (v_i, v_j) \in E$, for $v_i, v_j \in V$. Moreover, a graph is said to be *simple* if $(v, v) \notin E \ \forall v \in V$. In this work, we focus our attention solely on undirected simple graphs, which will be referred to simply as *graph*.

Let $X = (X_1, \ldots, X_d)$ denote a d-dimensional random vector, with each component $X_j \in A$, a finite alphabet, for $1 \leq j \leq d$. We include a superscript to indicate that this vector $X^{(i)}$, $i = 0, 1, \ldots$, is observed over a discrete-time, constituting a multivariate stochastic process. We assume the process is stationary, with invariant distribution π. In such processes, not all random variables within the vector X are inherently dependent on each other. Therefore, there exists a field of research dedicated to studying the relationships among these random variables by estimating the underlying graph of conditional dependencies inherent in this multivariate stochastic process. Denote by G^* the graph encoding the conditional dependencies in π. In this work, we are interested in estimating G^* as well as the associated conditional probability distributions.

In the case where we assume that the sample $X^{(1)}, \ldots, X^{(n)}$ is independent and identically distributed (IID), we reduce to the classical model selection for discrete graphical models or Markov random fields on graphs. Extensive research has been conducted on these models, including, but not limited to, Lauritzen [1996], Divino *et al.* [2000], Koller and Friedman [2009], Lerasle and Takahashi [2016], Pensar *et al.* [2017], Leonardi *et al.* [2023]. Furthermore, these models have found applications in various fields, including Biology [Shojaie and Michailidis, 2010], Social Sciences [Strauss and Ikeda, 1990] or Neuroscience [Duarte *et al.*, 2019]. Up to this moment, the most studied model has been the binary graphical model with pairwise interactions where structure estimation can be addressed by using standard logistic regression techniques [Ravikumar *et al.*, 2010, Strauss and Ikeda, 1990], distance-based approaches between conditional probabilities Bresler *et al.* [2018], Galves *et al.* [2015] and maximization of the ℓ_1-penalized pseudo-likelihood [Atchade, 2014, Höfling and Tibshirani, 2009]; see also

Santhanam and Wainwright [2012].

In the case of bigger discrete alphabets or general types of interactions, to our knowledge, the only works addressing the structure estimation problem are Loh and Wainwright [2013] and Leonardi *et al.* [2023]. In Loh and Wainwright [2013], the authors obtain a characterization of the edges in the graph with the zeros in a generalized inverse covariance matrix. Then, this characterization is used to derive estimators for restricted classes of models, and the authors prove the consistency in probability of these estimators. In the work Leonardi *et al.* [2023], a penalized criterion is proposed to estimate the neighborhood of each vertex, and the results are then combined to construct the model's graph. Markov random fields on graphs have also been proposed for continuous random variables, where the structure estimation problem has been addressed by ℓ_1-regularization for Gaussian Markov random fields [Meinshausen and Bühlmann, 2006] and also extended to non-parametric models [Lafferty *et al.*, 2012, Liu *et al.*, 2012] and general conditional distributions from the exponential family [Yang *et al.*, 2015].

From another perspective, graphical models can be seen as non-homogeneous versions of general random fields or Gibbs distributions on lattices, classical models in stochastic processes, and statistical mechanics theory [Georgii, 2011]. In such a setting, the number of variables increases despite having only one observation within the sample. Given the regularity of the graph (each node has the same neighborhood), inference and model selection can be done based on the unique observation. The statistical inference for Markov random fields and Gibbs distributions under this setting has been addressed in Comets [1992] and Comets and Gidas [1992], for example. More recently, model selection criteria, such as the BIC proposed by Schwarz [1978], have been proven consistent under this regular setting [Csiszár and Talata, 2006, Ji and Seymour, 1996]; see also Tjelmeland and Besag [1998] and Löcherbach and Orlandi [2011].

From an applied point of view, the assumption of independence of the observations in the non-homogeneous Markov random fields setting is often too restrictive. Consider, for example, the task of estimating interaction graphs from EEG time series data [Cerqueira *et al.*, 2017], river stream flow data [Leonardi *et al.*, 2021] or daily stock market indices [Leonardi *et al.*, 2023]. In these scenarios, the independence assumption does not hold, and the methods commonly used for graphical models serve only as approximations to the true underlying distribution. While such approximations can be practical from an applied point of view, from a theoretical perspective, it is interesting to consider the problem of estimation and model selection in a scenario with dependence, as, for example, in the case of mixing processes considered in this thesis.

Conventional model selection techniques for graphical models often involve estimating the neighborhoods of individual nodes and constructing the graph based on these neighborhoods, as exemplified by Ravikumar *et al.* [2010]. As mentioned above, the approach adopted by Leonardi *et al.* [2023] involves a penalized pseudo-likelihood criterion for estimating the graph of conditional dependencies within partially observed discrete Markov random fields.

This technique is based on estimating each node's neighborhood in the graph. The authors have established the almost sure convergence of the estimator in cases where the number of variables is finite or countably infinite, and the process is independent and identically distributed. Moreover, the method imposes minimal assumptions on the probability distribution, eliminating the need for the usual positivity condition present in other approaches in the literature. Once the neighborhood of each vertex is estimated, they are aggregated to construct the estimated graph itself. However, as discussed by the authors, depending on the rule to combine the neighborhoods, the final estimated graph can drastically underestimate or overestimate the set of edges in the graph.

Conversely, Leonardi *et al.* [2021] presents a different perspective by proposing a model selection criterion for estimating points of independence within a random vector that satisfies a mixing condition. This results in decomposing the vector's distribution function into distinct independent blocks. The method, based on a general estimator of the distribution function, can be applied to both discrete and continuous random vectors, as well as IID data or dependent time series. The authors have proved the consistency of the approach under general conditions on the estimator of the distribution function and show that the consistency holds for IID data and discrete time series with mixing conditions.

This thesis is mainly motivated by the works of Leonardi *et al.* [2021] and Leonardi *et al.* [2023] and can be viewed as a combination and generalization of both. We aim to overcome the limitations of the previously mentioned works. While the estimator introduced by Leonardi *et al.* [2023] is only applicable to IID data and the estimation is done for each vertex, the method proposed by Leonardi *et al.* [2021] assumes that the random vector can only be decomposed into subvectors. In response to these limitations, we propose a penalized pseudo-likelihood criterion for estimating the entire graph G, which consists of the set of edges E connecting the nodes V, particularly for multivariate stochastic processes satisfying a mixing condition. The primary advantage of our approach is its ability to handle non-IID data and its global estimation approach. This means that the entire set of edges E is estimated as a whole, eliminating the need to estimate the neighborhood of each node separately and then combine them to obtain the estimated graph. We provide a proof of convergence, showing that the estimator almost surely converges to the actual underlying graph in cases of finite graphical models, provided a mixing condition holds for the generating process.

This thesis is organized as follows. Chapter 2 reviews the theoretical framework of multivariate stochastic processes, particularly the definition of the mixing condition. The chapter concludes by proposing a method to estimate the underlying graph of conditional dependencies of a multivariate process satisfying a mixing condition.

Chapter 3 presents algorithms implementing the proposed graph estimator. Specifically, the exact algorithm that performs an exhaustive search for the graph. Additionally, the chapter covers the simulated annealing algorithm. This stochastic optimization technique iteratively explores potential edge additions or deletions while gradually reducing the acceptance probability of unfavorable changes to find an optimal structure that maximizes the

pseudo-log-likelihood function. Furthermore, the chapter discusses the forward and backward stepwise algorithms. This iterative model selection technique incrementally add or remove edges to maximize the pseudo-log-likelihood function while considering both the forward and or backward directions in the estimation process.

Chapter 4 conducts a detailed evaluation of the algorithms introduced in Chapter 3 for graph estimation. The assessment utilizes simulation studies to evaluate the convergence and performance of the proposed graph estimators, focusing on each algorithm individually. The evaluations are conducted in two distinct settings. First, the performance is analyzed in scenarios where a graph of conditional dependencies is defined, and data is sampled from a distribution with this structure. This allows for the assessment of the estimator's stability. Second, the investigation shifts to analyzing performance concerning the number of edges in the graph. This is achieved by generating random graphs, each with a different number of edges, and assessing the estimator's accuracy in recovering these graphs considering specific metrics. These comprehensive evaluations highlight the effectiveness and robustness of the proposed estimator in real-world applications.

Chapter 5 presents two distinct applications of the proposed method, primarily emphasizing practical implementation rather than introducing new modeling approaches. The analysis centers on stream flow data from the São Francisco River in Brazil, aiming to uncover the dependency structure among measurements from various gauges along the river. Also we present a study about stock market indices, examining the dependencies among the global markets in terms of their fluctuations, particularly focusing on their upward and downward movements.

Finally, Chapter 6 presents the main discussions, highlights the contributions of this work and the directions for future research. Additional and auxiliary content regarding theoretical results, complementary simulation results, and algorithms are presented in Appendices A, B, and C, respectively.

Chapter 2

Graph Estimator on Mixing Processes

This chapter is dedicated to presenting the fundamental aspects of multivariate stochastic processes with mixing conditions. Once again, this thesis aims to perform estimation and model selection for mixing graphical models. The purpose of this chapter is to establish the theoretical background for this work. We begin with the definition of graphical models and the concepts regarding this field of study in Section 2.1. Subsequently, we present the definition of vector-valued stochastic processes that satisfy a mixing condition in Section 2.2. Following this, we discuss the estimation and rate of convergence of the empirical probabilities in a stationary stochastic process with exponential mixing rate in Section 2.3. Finally, in Section 2.4 we discuss and propose a model for the estimation of the underlying graph of the process.

2.1 Graphical Models

Consider a graph $G = (V, E)$, with $V = \{1, \ldots, d\}$, for $d \in \mathbb{N}$, and assume we observe at each vertex $v \in V$ a random variable X_v, which is discrete and takes values in A, a finite alphabet. Moreover, let $X = (X_1, \ldots, X_d)$ be the vector of all variables observed on the graph's vertices. Denote by $\mathbb{P}$ the joint probability distribution of the vector X. For any $W \subset V$ and any configuration $a_W \in A^{|W|}$ we write

$$\pi(a_W) = \mathbb{P}(X_W = a_W).$$

Moreover, if $\pi(a_W) > 0$ then we denote by

$$\pi(a_U | a_W) = \mathbb{P}(X_U = a_U | X_W = a_W),$$

the corresponding conditional probability distributions, for $a_U \in A^{|U|}$ and $a_W \in A^{|W|}$, where U and W are subsets of V.

For a given vertex $v \in V$, any set $W \subset V$, with $v \notin W$, is a neighborhood of v.

Furthermore, W is called a *Markov neighborhood* of v if

$$\pi(a_v|a_U) = \pi(a_v|a_W),$$

for all $U \supset W$, $v \notin U$ and all $a_U \in A^{|U|}$, with $\pi(a_U) > 0$. The definition of a Markov neighborhood W of v is equivalent to request that for all $U' \subset V \setminus \{v\}$ (not containing v) such that $U' \cap W = \varnothing$, $X_{U'}$ is conditionally independent of X_v, given X_W. That is,

$$X_v \perp\!\!\!\perp X_{U'}|X_W,$$

for all U' with $U' \cap W = \varnothing$, where $\perp\!\!\!\perp$ is the usual symbol denoting independence of random variables.

As discussed in Leonardi *et al.* [2023], if W is a Markov neighborhood of $v \in V$, then any finite set $U \supset W$ is also a Markov neighborhood of v. In contrast, W_1 and W_2 being Markov neighborhoods of v does not imply in general that $W_1 \cap W_2$ is a Markov neighborhood of v, however this property is satisfied by some probability measures. This fact leads to the following definition.

Definition 1 (Markov intersection property)**.** *We say that π satisfies the Markov intersection property if for all $v \in V$ and all W_1 and W_2 Markov neighborhoods of v, the set $W_1 \cap W_2$ is also a Markov neighborhood of v.*

The Markov intersection property is desirable in this context to define the smallest Markov neighborhood of a node and its structure estimation. This property is guaranteed under the usually assumed positivity condition defined below.

Definition 2 (Positivity condition)**.** *We say that π satisfies the positivity condition if for all finite $W \subset V$ and all $a_W \in A^{|W|}$ we have $\pi(a_W) > 0$.*

The positivity condition implies the Markov intersection property [see Lauritzen, 1996]. For this reason, in the literature on Markov random fields, it is generally assumed that the positivity condition holds. Leonardi *et al.* [2023] discuss that there are distributions satisfying the Markov intersection property that are not strictly positive. Thus, we assume in this thesis that the distribution π satisfies the Markov intersection property in Definition 1. Therefore, it is worth to name the intersection of all Markov neighborhoods of a given node $v \in V$, as in Definition 3.

Definition 3 (Basic neighborhood)**.** *For $v \in V$, let $\mathcal{W}(v)$ be the set of all subsets of V which are Markov neighborhoods of v. The* basic neighborhood *of v is defined as*

$$\mathrm{ne}(v) = \bigcap_{W \in \mathcal{W}(v)} W. \tag{2.1}$$

When the Markov intersection property holds, $\mathrm{ne}(v)$ is the smallest Markov neighborhood of $v \in V$. Based on these basic neighborhoods, define the graph $G^* = (V, E^*)$ as

$$(v, w) \in E^* \text{ if and only if } w \in \mathrm{ne}(v), \tag{2.2}$$

where $E^* \subseteq V \times V$. The graph G^* with edges defined in (2.2) is *undirected*, as proved by Leonardi *et al.* [2023]. It means that,

$$(v, w) \in E^* \iff (w, v) \in E^*.$$

For any given graph $G = (V, E)$, let $G(v)$ be the set of all neighbors of vertex $v \in V$ in graph G, that is

$$G(v) = \{u \in V : (u, v) \in E\}. \tag{2.3}$$

Note that for $G = G^*$, $G(v) = \mathrm{ne}(v)$. This set $G(v)$ will be used in the estimation process presented in Section 2.4.

2.2 Vector-Valued Mixing Processes

In this work, we consider a vector-valued stationary and ergotic stochastic process $X^{(1)}, X^{(2)}, \ldots$, where each variable $X^{(i)}$ is a vector of d components, belonging to the set A^d, with A being a finite alphabet. We denote by $((A^d)^\infty, \mathcal{F}, \mathbb{P})$ the probability space for the process $\{X^{(i)} : i \in \mathbb{Z}\}$. We use superscript indices in this process to indicate the time at which the observation is taken. Therefore $X^{(i)} = (X_1^{(i)}, \ldots, X_d^{(i)})$ is the d-dimensional random vector observed at time i, for $i = 1, \ldots, n$, and more specifically $X_v^{(i)}$ is the random variable observed at time i on vertex $v \in V$. Further, we assume that the process $\{X^{(i)} : i \in \mathbb{Z}\}$ has an underlying graph G^*, which we aim to estimate this graph given a sample of the process.

In order to make inference, we shall consider *slices* of the entire realization of $X^{(1)}, \ldots, X^{(n)}$ on both dimensions, i.e. time and space. To avoid misleading notations, we use superscripts to denote the indices in *time* (ranging from 1 to n) and subscripts to represent indices on *space* (a subset of $V = \{1, \ldots, d\}$). For any set $U \subset V$ and any integer interval $i : j$, we denote by $X_U^{(i:j)}$ the sequence $X_U^{(i)}, \ldots, X_U^{(j)}$ with $X_U^{(k)} = (X_u^{(k)} : u \in U)$, $k = i, \ldots, j$. When $U = V$, we avoid the subscript and simply write $X^{(i:j)}$. The same notation is used for *realizations* of the process, denoted in lower case $x_U^{(i:j)}$ instead of the notation for the random variables $X_U^{(i:j)}$. As an illustration, Figure 2.1 shows an example of realization of the process X: the x-axis represents *time* whereas the y-axis represents *space*, i.e., the vertices. Moreover, the rectangle highlights the slice of the process denoted by $x_U^{(2:3)}$, where $U = \{1, 2\}$, or straightforwardly, $x_{\{1,2\}}^{(2:3)}$.

As mentioned before, we want to estimate the graph G^* given a sample $\{x^{(i)} : i = 1, \ldots, n\}$. The estimation of G^* is presented in Leonardi *et al.* [2023] specifically for the case when the observations are independent, that is, for IID processes. We consider

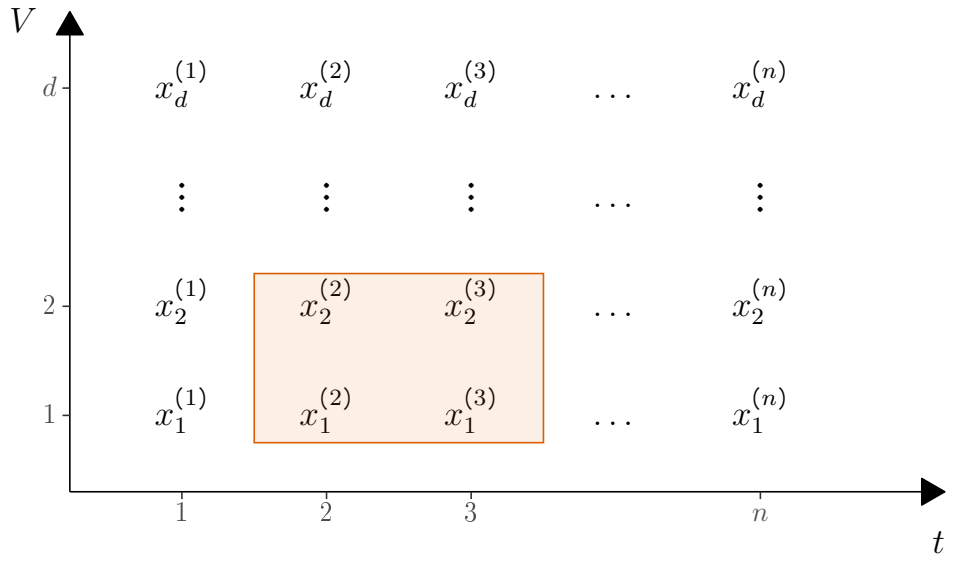


Figure 2.1: *Representation of a realization of the process* $\mathbf{X}$*, with set of vertices* $V = \{1, \dots, d\}$ *observed from time* 1 *to* n*. The subscript indicates the vertex, and the superscript indicates the time at which the observation was taken. The highlighted rectangle indicates the observed slice* $x_{\{1,2\}}^{(2:3)}$.

a less restrictive case where the sample satisfies the mixing condition below.

Definition 4 (Mixing condition)**.** *For* $i < j$*, let* $X^{(i:j)}$ *denote the sequence of vectors* $X^{(i)}, X^{(i+1)}, \dots, X^{(j)}$*. We say the process* $\mathbf{X} = \{X^{(i)} : -\infty < i < \infty\}$ *satisfies a* mixing condition *with rate* $\{\psi(\ell)\}_{\ell \in \mathbb{N}}$ *if for each* $k, m \in \mathbb{N}$ *and each* $x^{(1:k)} \in (A^d)^k$*,* $x^{(1:m)} \in (A^d)^m$ *with* $\mathbb{P}(X^{(1:m)} = x^{(1:m)}) > 0$*, we have that*

$$\left|\mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)} \mid X^{(1:m)} = x^{(1:m)}\big) - \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)}\big)\right| \leq \psi(\ell)\mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)}\big), \tag{2.4}$$

for $\ell \geq n - m$*.*

In the estimation process that we propose in Section 2.4, we will consider the sample data restricted to some nodes rather than the entire process. Specifically, a subprocess $\{X_W^{(i)} : i \in \mathbb{N}\}$, $W \subset V$, where $X_W^{(i)} = (X_w^{(i)} \in A : w \in W)$. Lemma 5 states that any sub-process of a multivariate mixing process is also mixing.

Lemma 5. *If the process* $\{X^{(i)} : i \in \mathbb{Z}\}$ *satisfies the mixing condition* (2.4) *with rate* $\{\psi(\ell)\}$*, then for* $W \subset V$*, the sub-process* $\{X_W^{(i)} : i \in \mathbb{Z}\}$ *is also mixing with the same rate* $\{\psi(\ell)\}$*.*

Proof. Let $W^c = V \setminus W$, and let $a_{W^c}^{(1:k)} \in (A^{|W^c|})^k$. Observe that for any $a_W^{(1:k)} \in \left(A^{|W|}\right)^k$, and any $a_W^{(1:m)} \in \left(A^{|W|}\right)^m$ we have that

$$\begin{aligned}
&\left|\mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)} \mid X_W^{(1:m)} = a_W^{(1:m)}) - \mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)})\right| \\
&= \Big| \sum_{a_{W^c}^{(1:k)}} \mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)}, X_{W^c}^{(n:(n+k-1))} = a_{W^c}^{(1:k)} \mid X_W^{(1:m)} = a_W^{(1:m)})
\end{aligned}$$

$$
\begin{aligned}
&\quad - \sum_{a_{W^c}^{(1:k)}} \mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)}, X_{W^c}^{(n:(n+k-1))} = a_{W^c}^{(1:k)})\Big| \\
&\leq \sum_{a_{W^c}^{(1:k)}, a_{W^c}^{(1:m)}} \Big|\mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)}, X_{W^c}^{(n:(n+k-1))} = a_{W^c}^{(1:k)} | X_W^{(1:m)} = a_W^{(1:m)}, X_{W^c}^{(1:m)} = a_{W^c}^{(1:m)}) \\
&\quad - \mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)}, X_{W^c}^{(n:(n+k-1))} = a_{W^c}^{(1:k)})\Big| \\
&\leq \sum_{a_{W^c}^{(1:k)}} \psi(n-m)\mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)}, X_{W^c}^{(n:(n+k-1))} = a_{W^c}^{(1:k)}) \\
&\leq \psi(n-m)\mathbb{P}(X_W^{(n:(n+k-1))} = a_W^{(1:k)}) .
\end{aligned}
$$

Then the process $\{X_W^{(i)} : i \in \mathbb{N}\}$ with $W \subset \{1, \dots, d\}$ is mixing with rate $\{\psi(\ell)\}$. □

The following lemma states that a given function of a mixing process is also mixing, as long as this function is bounded and takes values from A^d into the real line.

Lemma 6. *Let the process $\{X^{(i)} : i \in \mathbb{Z}\}$ satisfy the mixing condition (2.4) with rate $\{\psi(\ell)\}_{\ell \in \mathbb{N}}$ Since A^d is finite, $f : A^d \to \mathbb{R}$ is a bounded function. Then, the process $\{f(X^{(i)}) : i \in \mathbb{Z}\}$ is also mixing with rate $\psi(\ell)_{\ell \in \mathbb{N}}$.*

Proof. Consider $Y^{(i)} = f(X^{(i)})$ and let $\mathbb{P}_Y$ denote the distribution of the process $\{Y^{(i)} : i \in \mathbb{Z}\}$. Note that

$$
\begin{aligned}
\mathbb{P}_Y(Y^{(i)} = y) &= \mathbb{P}_Y(f(X^{(i)}) = y) \\
&= \sum_{a : f(a) = y} \mathbb{P}_X(X^{(i)} = a),
\end{aligned}
$$

where $a \in A^d$ and $\mathbb{P}_X$ denotes the probability distribution of the process $\{X^{(i)} : i \in \mathbb{Z}\}$. Also note that the notation above can be generalized for vectors of the form $Y^{(1:m)} = (Y^{(1)}, \dots, Y^{(m)})$, for $m > 1$, as

$$
\begin{aligned}
\{Y^{(1:m)} = y^{(1:m)}\} &= \bigcap_{i=1}^{m} \{f(X^{(i)}) = y^{(i)}\} \\
&= \bigcap_{i=1}^{m} \Big\{ \bigcup_{a : f(a) = y^{(i)}} \{X^{(i)} = a\} \Big\}.
\end{aligned}
\tag{2.5}
$$

Now, for $i < j$, define the set

$$
C(y^{(i:j)}) = \Big\{ (a^{(i)}, \dots, a^{(j)}) \in A^{d \times (j-i+1)} : f(a^{(k)}) = y^{(k)}, k = i, \dots, j \Big\},
$$

which denotes all configurations of $(a^{(i)}, \dots, a^{(j)})$ such that $\{Y^{(i:j)} = y^{(i:j)}\}$ holds. Then, we can write

$$
\mathbb{P}_Y(Y^{(1:m)} = y^{(1:m)}) = \sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}_X(X^{(1:m)} = x^{(1:m)})
\tag{2.6}
$$

for each sequence $y^{(1:m)}$.

Analogously to (2.5), the conditional event $\{Y^{(n:(n+k-1))} = y^{(1:k)} \big| Y^{(1:m)} = y^{(1:m)}\}$ can be written as

$$\left\{ \bigcup_{x^{(1:k)} \in C(y^{(1:k)})} \{X^{(n:(n+k-1))} = x^{(1:k)}\} \middle| \bigcup_{x^{(1:m)} \in C(y^{(1:m)})} \{X^{(1:m)} = x^{(1:m)}\} \right\},$$

for each sequence $y^{(1:k)}$ given $y^{(1:m)}$. Then

$$\begin{aligned}
&\mathbb{P}_Y\big(Y^{(n:(n+k-1))} = y^{(1:k)} \big| Y^{(1:m)} = y^{(1:m)}\big) \\
&\quad = \mathbb{P}\left(\bigcup_{x^{(1:k)} \in C(y^{(1:k)})} \{X^{(n:(n+k-1))} = x^{(1:k)}\} \middle| \bigcup_{x^{(1:m)} \in C(y^{(1:m)})} \{X^{(1:m)} = x^{(1:m)}\} \right) \\
&\quad = \sum_{x^{(1:k)} \in C(y^{(1:k)})} \mathbb{P}\left(X^{(n:(n+k-1))} = x^{(1:k)} \middle| \bigcup_{x^{(1:m)} \in C(y^{(1:m)})} \{X^{(1:m)} = x^{(1:m)}\} \right) \\
&\quad = \sum_{x^{(1:k)} \in C(y^{(1:k)})} \frac{\mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}, \cup_{x^{(1:m)} \in C(y^{(1:m)})} \{X^{(1:m)} = x^{(1:m)}\})}{\mathbb{P}(\cup_{x^{(1:m)} \in C(y^{(1:m)})} X^{(1:m)} = x^{(1:m)})} \\
&\quad = \sum_{x^{(1:k)} \in C(y^{(1:k)})} \frac{\sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}, X^{(1:m)} = x^{(1:m)})}{\sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}(X^{(1:m)} = x^{(1:m)})}.
\end{aligned} \tag{2.7}$$

Observe that by the mixing property (2.4), for each $x^{(1:m)} \in C\big(y^{(1:m)}\big)$, we obtain that

$$\begin{aligned}
[1 - \psi(n-m)]\mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}) \leq \frac{\mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}, X^{(1:m)} = x^{(1:m)})}{\mathbb{P}(X^{(1:m)} = x^{(1:m)})} \\
\leq [1 + \psi(n-m)]\mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}).
\end{aligned} \tag{2.8}$$

By combining (2.7) with (2.8), we obtain that

$$\begin{aligned}
&[1 - \psi(n-m)]\mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}) \\
&\qquad \leq \frac{\sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}, X^{(1:m)} = x^{(1:m)})}{\sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}(X^{(1:m)} = x^{(1:m)})} \\
&\qquad \leq [1 + \psi(n-m)]\mathbb{P}(X^{(n:(n+k-1))} = x^{(1:k)}).
\end{aligned} \tag{2.9}$$

Finally, to prove that the process $\{Y^{(i)}, i \in \mathbb{N}\}$ is mixing, we need to show that the absolute difference

$$\left| \mathbb{P}\big(Y^{(n:(n+k-1))} = y^{(1:k)} \big| Y^{(1:m)} = y^{(1:m)}\big) - \mathbb{P}\big(Y^{(n:(n+k-1))} = y^{(1:k)}\big) \right| \tag{2.10}$$

is bounded, similarly as in Expression (2.4). Using (2.6), (2.7), and (2.9) we can rewrite

expression (2.10) as

$$
\begin{aligned}
&\Bigg| \sum_{x^{(1:k)} \in C(y^{(1:k)})} \Bigg[\mathbb{P}\Bigg(X^{(n:(n+k-1))} = x^{(1:k)} \Bigg| \bigcup_{x^{(1:m)} \in C(y^{(1:m)})} \{X^{(1:m)} = x^{(1:m)}\} \Bigg) \Bigg] \\
&\qquad\qquad - \sum_{x^{(1:k)} \in C(y^{(1:k)})} \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)} \big) \Bigg| \\
&\leq \sum_{x^{(1:k)} \in C(y^{(1:k)})} \Bigg| \mathbb{P}\Bigg(X^{(n:(n+k-1))} = x^{(1:k)} \Bigg| \bigcup_{x^{(1:m)} \in C(y^{(1:m)})} \{X^{(1:m)} = x^{(1:m)}\} \Bigg) \\
&\qquad\qquad - \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)} \big) \Bigg| \\
&= \sum_{x^{(1:k)} \in C(y^{(1:k)})} \Bigg| \frac{\sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)}, X^{(1:m)} = x^{(1:m)} \big)}{\sum_{x^{(1:m)} \in C(y^{(1:m)})} \mathbb{P}\big(X^{(1:m)} = x^{(1:m)} \big)} \\
&\qquad\qquad - \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)} \big) \Bigg| \\
&\leq \sum_{x^{(1:k)} \in C(y^{(1:k)})} \psi(n-m) \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)} \big) \\
&= \psi(n-m) \sum_{x^{(1:k)} \in C(y^{(1:k)})} \mathbb{P}\big(X^{(n:(n+k-1))} = x^{(1:k)} \big) \\
&= \psi(n-m) \mathbb{P}\big(Y^{(n:(n+k-1))} = y^{(1:k)} \big).
\end{aligned}
$$

Thus,

$$
\begin{aligned}
&\Big| \mathbb{P}\big(Y^{(n+1):(n+k)} = y^{(1:k)} \big| Y^{(1:m)} = y^{(1:m)} \big) - \mathbb{P}\big(Y^{(1:k)} = y^{(1:k)} \big) \Big| \\
&\qquad\qquad \leq \psi(n-m) \mathbb{P}\big(Y^{(n:(n+k-1))} = y^{(1:k)} \big)
\end{aligned}
$$

and the process $\{Y^{(i)}, i \in \mathbb{Z}\}$ is mixing with rate $\{\psi(\ell)\}$. □

2.3 Empirical Probabilities

Assume we observe a sample of size n of the process, denoted by $\{x^{(i)} : i = 1, \ldots, n\}$. In this section, we state and prove Proposition 7 and Proposition 8 that show upper bounds for the rate of convergence of $\widehat{\pi}(a_W)$ into $\pi(a_W)$ and $\widehat{\pi}(a_v|a_W)$ into $\pi(a_v|a_W)$, respectively. These are auxiliary results needed in the proof of Theorem 10, the main contribution of this thesis.

Since the stationary distribution of the process π is unknown, we must estimate it from the data. For any $W \subset V$ and any $a_W \in A^W$ denote by

$$
\widehat{\pi}(a_W) = \frac{N(a_W)}{n},
$$

where $N(a_W)$ denotes the number of times the configuration a_W appears in the sample $\{x^{(i)}: i = 1, \dots, n\}$. If $\widehat{\pi}(a_W) > 0$, we can also define the conditional probabilities

$$\widehat{\pi}(a_W | a_{W'}) = \frac{\widehat{\pi}(a_{W \cup W'})}{\widehat{\pi}(a_{W'})}, \tag{2.11}$$

for two disjoint subsets $W, W' \subset V$ and configurations $a_W \in A^W, a_{W'} \in A^{W'}$.

Based on results in Csiszár [2002], we can state and prove the following two propositions showing the rate of convergence of the empirical probabilities in a stationary stochastic process with exponential mixing sequence.

Proposition 7 (Typicality). *Assume the process $\{X^{(i)}: i \in \mathbb{Z}\}$ satisfies the mixing condition (2.4) with rate $\psi(\ell) = O(1/\ell^{1+\epsilon})$, for some $\epsilon > 0$. Then, for any $W \subset V$ and $\delta > 0$,*

$$\left| \widehat{\pi}(a_W) - \pi(a_W) \right| < \sqrt{\frac{\delta \log n}{n}},$$

eventually almost surely as $n \to \infty$.

Proof. Let $a_W \in A^W$ be fixed and let

$$Y^{(i)} = f(X^{(i)}) = \mathbb{1}\{X_W^{(i)} = a_W\} - \pi(a_W), \tag{2.12}$$

for $i = 1, 2, \dots, n$, be random variables that are functions of the process $\{X_W^{(i)} : i \in \mathbb{N}, W \subset V\}$. Since $\{X^{(i)} : i \in \mathbb{Z}\}$ is mixing with rate $\psi(\ell)$ and the function $f(X^{(i)})$ defined in (2.12) is bounded, then by Lemmas 5 and 6 the sub-process $\{Y^{(i)} : i \in \mathbb{Z}\}$ is also mixing with the same rate.

Also, note that

$$\begin{aligned} \mathbb{E}(Y^{(i)}) &= \big[1 - \pi(a_W)\big]\pi(a_W) - \pi(a_W)\big[1 - \pi(a_W)\big] \\ &= 0, \end{aligned}$$

and

$$\begin{aligned} \mathbb{E}\big[(Y^{(i)})^2\big] &= \big[1 - \pi(a_W)\big]^2 \pi(a_W) + \pi(a_W)^2\big[1 - \pi(a_W)\big] \\ &= \pi(a_W) - 2\pi(a_W)^2 + \pi(a_W)^3 + \pi(a_W)^2 - \pi(a_W)^3 \\ &= \pi(a_W)\big[1 - \pi(a_W)\big] \\ &\leq \frac{1}{4}. \end{aligned} \tag{2.13}$$

For $Y^{(i)}$ defined in (2.12) and $\delta > 0$, we have that

$$\begin{aligned}
\mathbb{E}\left|Y^{(i)}\right|^{2+\delta} &= \sum_{y^{(i)}} |y^{(i)}|^{2+\delta} \pi(Y^{(i)} = y^{(i)}) \\
&= \left|1 - \pi(a_W)\right|^{2+\delta} \pi(a_W) + \left| - \pi(a_W)\right|^{2+\delta} \left[1 - \pi(a_W)\right] \\
&= \left[\left|1 - \pi(a_W)\right|^{2+\delta} - \left|\pi(a_W)\right|^{2+\delta}\right] \pi(a_W) + \left|\pi(a_W)\right|^{2+\delta} \\
&\leq \left[\left(1 - \pi(a_W)\right)^{2+\delta} - \pi(a_W)^{2+\delta}\right] \pi(a_W) + \pi(a_W)^{2+\delta} \\
&< \infty.
\end{aligned}$$

Now, define the sum of the first n elements $Y^{(i)}$,

$$Z_n = \sum_{i=1}^{n} Y^{(i)}.$$

By Theorem 17 (see Appendix A), for some $\epsilon > 1/(1+\delta)$, the process $\{Y^{(i)} : i \in \mathbb{N}\}$ satisfies the Law of the Iterated Logarithm. That is, for any $\epsilon > 0$,

$$|Z_n| < (1+\epsilon)(2\sigma^2 n \log\log n)^{1/2},$$

eventually almost surely as $n \to \infty$, where

$$\sigma^2 = \mathbb{E}\left[(Y^{(1)})^2\right] + 2\sum_{j=1}^{n} \mathbb{E}\left[Y^{(1)} Y^{(j)}\right].$$

Now, by Lemma 19, since $E\left[Y^{(1)}Y^{(j)}\right] = 0$, we have that $\sigma^2 = E\left[\left(Y^{(1)}\right)^2\right]$. And therefore, by Expression (2.13), we have that

$$\sigma^2 \leq \frac{1}{4}.$$

Thus, for any $\epsilon > 0$,

$$|Z_n| < (1+\epsilon)(n \log\log n)^{1/2},$$

eventually almost surely as $n \to \infty$. Since $Z_n = N(a_W) - n\pi(a_W)$, we obtain

$$\left|\frac{Z_n}{n}\right| = \left|\hat{\pi}(a_W) - \pi(a_W)\right| < \sqrt{\frac{2\log\log n}{n}},$$

eventually almost surely as $n \to \infty$. Now, for any $\delta > 0$ we have that

$$2\log\log n < \delta \log n.$$

for all n sufficiently large. Therefore,

$$\left|\hat{\pi}(a_W) - \pi(a_W)\right| < \sqrt{\frac{\delta \log n}{n}},$$

eventually almost surely as $n \to \infty$. □

Proposition 8 (Conditional typicality)**.** *Then for any $\delta > 0$, any disjoint sets $W, W' \subset V$ and any $a_W \in A^W$ and $a_{W'} \in A^{W'}$ we have that*

$$\left|\widehat{\pi}(a_W|a_{W'}) - \pi(a_W|a_{W'})\right| < \sqrt{\frac{\delta \log n}{N(a_W)}},$$

eventually almost surely as $n \to \infty$.

Proof. This proof uses similar arguments to those presented to prove Proposition 7 with slight adaptations since here we seek for the estimator's rate of convergence to the conditional distribution $\pi(a_W|a_{W'})$.

For $W, W' \in V$ two disjoint subsets, let $a_W \in A^W$ and $a_{W'} \in A^{W'}$ be two fixed configurations. Define the process $\{Y^{(i)} : i \in \mathbb{Z}\}$ by

$$Y^{(i)} = \mathbb{1}\{X_W^{(i)} = a_W, X_{W'}^{(i)} = a_{W'}\} - \pi(a_W|a_{W'})\mathbb{1}\{X_W^{(i)} = a_{W'}\}, \tag{2.14}$$

for $i = 1, 2, \ldots, n$. Note that $Y^{(i)}$ can only assume three values,

$$Y^{(i)} = \begin{cases} 1 - \pi(a_W|a_{W'}), & \text{with probability } \pi(a_W, a_{W'}), \\ 0, & \text{with probability } 1 - \pi(a_{W'}), \\ -\pi(a_W|a_{W'}), & \text{with probability } \sum_{\tilde{a}_W \neq a_W} \pi(\tilde{a}_W, a_{W'}) = \pi(a_{W'}) - \pi(a_W, a_{W'}), \end{cases}$$

where $\pi(a_{W'}) = \sum_{a \in A^{|W|}} \pi(a, a_{W'})$. Since $\{X^{(i)} : i \in \mathbb{Z}\}$ is mixing with rate $\psi(\ell)$ and $Y^{(i)}$ defined in (2.14) is a bounded function of this process, then, according to Lemmas 5 and 6 the sub-process $\{Y^{(i)} : i \in \mathbb{N}\}$ is also mixing with the same rate.

Moreover,

$$\begin{aligned}
\mathbb{E}(Y^{(i)}) &= \big[1 - \pi(a_W|a_{W'})\big]\pi(a_W, a_{W'}) - \pi(a_W|a_{W'})\big[\pi(a_{W'}) - \pi(a_W, a_{W'})\big] \\
&= \pi(a_W, a_{W'}) - \pi(a_W|a_{W'})\pi(a_W, a_{W'}) - \pi(a_W|a_{W'})\pi(a_{W'}) + \pi(a_W|a_{W'})\pi(a_W, a_{W'}) \\
&= \pi(a_W, a_{W'}) - \frac{\pi(a_W, a_{W'})}{\pi(a_{W'})}\pi(a_W, a_{W'}) - \frac{\pi(a_W, a_{W'})}{\pi(a_{W'})}\pi(a_{W'}) + \frac{\pi(a_W, a_{W'})}{\pi(a_{W'})}\pi(a_W, a_{W'}) \\
&= 0,
\end{aligned}$$

and

$$\mathbb{E}\big[(Y^{(i)})^2\big] = \big[1 - \pi(a_W|a_{W'})\big]^2\pi(a_W, a_{W'}) + \big[-\pi(a_W|a_{W'})\big]^2\big[\pi(a_{W'}) - \pi(a_W, a_{W'})\big]$$

$$
\begin{aligned}
&= \big[1 - 2\pi(a_W|a_{W'}) + \pi(a_W|a_{W'})^2\big]\pi(a_W, a_{W'}) \\
&\qquad + \big[-\pi(a_W|a_{W'})\big]^2\big[\pi(a_{W'}) - \pi(a_W, a_{W'})\big] \\
&= \pi(a_W, a_{W'}) - 2\pi(a_W|a_{W'})\pi(a_W, a_{W'}) + \pi(a_W|a_{W'})^2\pi(a_W, a_{W'}) \\
&\qquad + \pi(a_W|a_{W'})^2\pi(a_{W'}) - \pi(a_W|a_{W'})^2\pi(a_W, a_{W'}) \\
&= \pi(a_W, a_{W'}) - 2\pi(a_W|a_{W'})\pi(a_W, a_{W'}) + \pi(a_W|a_{W'})^2\pi(a_{W'}) \\
&= \pi(a_W, a_{W'})\left[1 - 2\frac{\pi(a_W, a_{W'})}{\pi(a_{W'})} + \frac{\pi(a_W, a_{W'})}{\pi(a_{W'})}\right] \\
&= \pi(a_W, a_{W'})\left[1 - \frac{\pi(a_W, a_{W'})}{\pi(a_{W'})}\right] \\
&= \frac{\pi(a_W, a_{W'})}{\pi(a_{W'})}\big[\pi(a_{W'}) - \pi(a_W, a_{W'})\big] \\
&= \pi(a_W|a_{W'})\big[\pi(a_{W'}) - \pi(a_W|a_{W'})p(a_{W'})\big] \\
&= \pi(a_W|a_{W'})\big[1 - \pi(a_W|a_{W'})\big]\pi(a_{W'}) \\
&\leq \frac{1}{4}\pi(a_{W'}).
\end{aligned} \tag{2.15}
$$

Also note that, for some $\delta > 0$, we have

$$
\begin{aligned}
\mathbb{E}\big|Y^{(i)}\big|^{2+\delta} &= \sum_{y^{(i)}} |y^{(i)}|^{2+\delta}\pi(Y^{(i)} = y^{(i)}) \\
&= \big|1 - \pi(a_W|a_{W'})\big|^{2+\delta}\pi(a_W, a_{W'}) + \big|-\pi(a_W|a_{W'})\big|^{2+\delta}\big[\pi(a_{W'}) - \pi(a_W, a_{W'})\big] \\
&= \Big[\big|1 - \pi(a_W|a_{W'})\big|^{2+\delta} - \big|\pi(a_W|a_{W'})\big|^{2+\delta}\Big]\pi(a_W, a_{W'}) + \big|\pi(a_W|a_{W'})\big|^{2+\delta}\pi(a_{W'}) \\
&\leq \Big[\big(1 - \pi(a_W|a_{W'})\big)^{2+\delta} - \pi(a_W|a_{W'})^{2+\delta}\Big]\pi(a_W, a_{W'}) + \pi(a_W|a_{W'})^{2+\delta}\pi(a_{W'}) \\
&\leq \Big[\big(1 - \pi(a_W|a_{W'})\big)^{2+\delta} - \pi(a_W|a_{W'})^{2+\delta}\Big] + \pi(a_W|a_{W'})^{2+\delta} + \pi(a_W|a_{W'})^{2+\delta} \\
&< \infty.
\end{aligned} \tag{2.16}
$$

Now, define

$$
Z_n = \sum_{i=1}^{n} Y^{(i)}. \tag{2.17}
$$

Analogously to the proof of Proposition 7, since (2.16) holds, $\mathbb{E}\big|Y^{(i)}\big|^{2+\delta} < \infty$ with rate $\psi(\ell) = O(1/\ell^{1+\epsilon})$, then by Theorem 17 and Remark 18 (see in Appendix A), we obtain that

$$
|Z_n| < (1 + \epsilon)(2\sigma^2 n \log\log n)^{1/2},
$$

eventually almost surely as $n \to \infty$, where $\sigma^2 = \mathbb{E}\big[(Y^{(0)})^2\big] + 2\sum_{j=1}^{n}\mathbb{E}\big[Y^{(0)}Y^{(j)}\big]$. Again, by Lemma 19, since $E\big[Y^{(0)}Y^{(j)}\big] = 0$ and from (2.15), we have that

$$
\sigma^2 \leq \frac{1}{4}\pi(a_{W'}).
$$

Therefore, for any $\epsilon > 0$,

$$|Z_n| < (1+\epsilon)\left[\frac{1}{2}\pi(a_{W'})n\log\log n\right]^{1/2},$$

eventually almost surely as $n \to \infty$. In particular, by taking $\epsilon = \sqrt{2}-1$ we have that

$$|Z_n| < \left[\pi(a_{W'})n\log\log n\right]^{1/2}, \tag{2.18}$$

eventually almost surely as $n \to \infty$.

Note that Z_n defined in (2.17) can be written as

$$Z_n = N(a_W, a_{W'}) - \pi(a|a_{W'})N(a_{W'}).$$

If we divide Z_n by $N(a'_W)$ and together with (2.18), we get that

$$\left|\widehat{\pi}(a_W|a_{W'}) - \pi(a_W|a_{W'})\right| < \sqrt{\frac{\pi(a_{W'})n\log\log n}{N(a_{W'})^2}},$$

eventually almost surely as $n \to \infty$. By Proposition 7, for any $\alpha > 0$ we have that

$$N(a_{W'}) \;>\; n\pi(a_{W'}) - \sqrt{\delta n\log n} \;>\; (1-\alpha)n\pi(a_{W'}),$$

eventually almost surely as $n \to \infty$. Then, we obtain that

$$\left|\widehat{\pi}(a_W|a_{W'}) - \pi(a_W|a_{W'})\right| \;<\; \sqrt{\frac{\log\log n}{(1-\alpha)N(a_{W'})}}\,.$$

As before, for any $\delta > 0$ we have that

$$\frac{\log\log n}{1-\alpha} \;< \delta\log n$$

for sufficiently large n, and therefore for all $\delta > 0$

$$\left|\widehat{\pi}(a_W|a_{W'}) - \pi(a_W|a_{W'})\right| \;<\; \sqrt{\frac{\delta\log n}{N(a_{W'})}},$$

eventually almost surely as $n \to \infty$. □

2.4 Regularized Maximum Pseudo-Likelihood Graph Estimator

In this work, we take a regularized pseudo maximum likelihood approach to estimate the graph G^*, given a sample $x^{(1)}, \ldots, x^{(n)}$ of the stochastic process. Instead of estimating each neighborhood and then combining the results, as is done in several works, in particular in Leonardi *et al.* [2023], we globally estimate the graph G^* by optimizing a function over the set of all simple graphs over V.

Given any graph G, we define the pseudo-likelihood function by

$$L(G) = \prod_{i=1}^{n} \prod_{v \in V} \pi(x_v^{(i)} | x_{G(v)}^{(i)}),$$

where $G(v)$, defined in (2.3), represents the set of all neighbors of node v in graph G. Since the conditional probabilities of π are unknown, we can estimate them from the data, obtaining the maximum pseudo-likelihood estimator given by

$$\widehat{L}(G) = \prod_{i=1}^{n} \prod_{v \in V} \widehat{\pi}(x_v^{(i)} | x_{G(v)}^{(i)}), \tag{2.19}$$

with $\widehat{\pi}(x_v^{(i)} | x_{G(v)}^{(i)})$ being defined as in (2.11) taking $W = \{v\}$ and $W' = \{G(v)\}$. Applying the logarithm and taking into account the number of occurrences of each configuration in the sample, we can write the log pseudo-likelihood function as

$$\log \widehat{L}(G) = \sum_{v \in V} \sum_{a_v \in A} \sum_{a_{G(v)} \in A^{|G(v)|}} N(a_v, a_{G(v)}) \log \widehat{\pi}(a_v | a_{G(v)}), \tag{2.20}$$

where the sum is taken over all $v \in V$ and all configurations $a_v \in A$, $a_{G(v)} \in A^{|G(v)|}$ such that $N(a_v, a_{G(v)}) > 0$. Here $|G(v)|$ denotes the cardinal of the set $G(v)$. The graph estimator is defined below.

Definition 9 (Graph estimator). *Given a sample $\{x^{(i)} : i \in \{1, \ldots, n\}\}$ of a process that satisfies the mixing condition (2.4) and λ_n a non-negative decreasing sequence, we define the estimator of G^* as*

$$\widehat{G} = \arg\max_{G} \Big\{ \log \widehat{L}(G) - \lambda_n \sum_{v \in V} |A|^{|G(v)|} \Big\}. \tag{2.21}$$

Denote by $G_{\max}$ the complete graph over V, that is

$$G_{\max} = (V, E_{\max}),$$

where

$$E_{\max} = \{(u, v) \in V : u \neq v\}.$$

Note that in particular we have that $G_{\max}(v) = V \setminus \{v\}$ for all $v \in V$. For any $v \in V$, denote by

$$\alpha(v) = \min_{G(v) \subset V \setminus \{v\} \,:\, G^*(v) \not\subseteq G(v)} \left\{ \sum_{a_{G_{\max}(v)}} \pi(a_{G_{\max}(v)}) D\big(\pi(\cdot_v | a_{G^*(v)}); \pi(\cdot_v | a_{G(v)})\big) \right\} \tag{2.22}$$

where $\pi(\cdot_v | a_{G^*(v)})$ denotes the probability distribution over A given by $\{\pi(a_v | a_{G^*(v)})\}_{a_v \in A}$ and similarly for $\pi(\cdot_v | a_{G(v)})$, and D denotes the Küllback-Leibler divergence (see Definition 14 in Appendix A). By Definition 3 of basic neighborhood and Lemma 20 (presented in Appendix A) we must have $\alpha(v) > 0$. A formal proof can be found in Leonardi *et al.* [2023]. Thus, by taking the sum of $\alpha(v) > 0$ over all $v \in V$ we get

$$\sum_{v \in V} \alpha(v) > 0.$$

The quantity defined in (2.22) is required to prove the results in the sequel.

The theorem presented below is the main theoretical contribution of this work. Moreover, as mentioned before, Propositions 7 and 8 are auxiliary results used to prove the consistency of our proposed estimator.

Theorem 10. *Assume the process* $\{X^{(i)} : i \in \mathbb{Z}\}$ *satisfies the mixing condition* (2.4) *with* $\psi(\ell) = O(1/\ell^{1+\epsilon})$ *for some* $\epsilon > 0$. *Then, taking* $\lambda_n = c \log n$ *we have that* $\widehat{G}$ *defined in* (2.21) *satisfies* $\widehat{G} = G^*$ *eventually almost surely as* $n \to \infty$.

Proof. First, note that we can decompose the event $\{\widehat{G} \neq G^*\}$ as the union of two events:

$$\{G^* \subsetneq \widehat{G}\} \cup \{G^* \not\subseteq \widehat{G}\}, \tag{2.23}$$

for all graph $\widehat{G}$. The event on the left-hand side signifies overfitting, occurring when $\widehat{G}$ is a strict superset of G^*, meaning that $\widehat{G}$ has at least one more edge than G^*. On the other hand, the event on the right-hand side denotes underfitting, where $\widehat{G}$ has at least one fewer edge than G^*. In this proof, we consider the elements in the union (2.23) separately, i.e., we treat them in two cases

(a) $\{G^* \subsetneq \widehat{G}\}$,

(b) $\{G^* \not\subseteq \widehat{G}\}$,

proving that eventually almost surely as $n \to \infty$, neither of them can happen, which implies that $\widehat{G} = G^*$.

First, case (a), where $G^* \subsetneq \widehat{G}$, is graphically shown in Figure 2.2 (a). We show that $\{G^* \subsetneq \widehat{G}\}$ does not hold eventually almost surely as $n \to \infty$. To show that, we prove that for all graph G such that $G^* \subsetneq G$,

$$\log \widehat{L}(G) - \lambda_n \sum_{v \in V} |A|^{|G(v)|} < \log \widehat{L}(G^*) - \lambda_n \sum_{v \in V} |A|^{|G^*(v)|}, \tag{2.24}$$

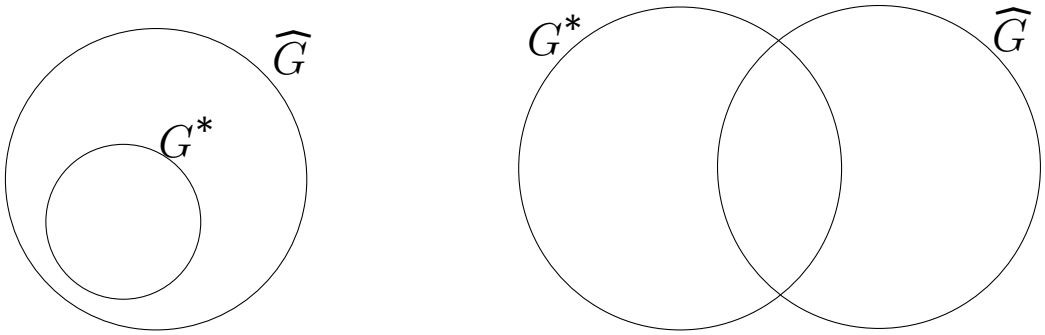


Figure 2.2: *Decomposition of the event* $\{\widehat{G} \neq G^*\}$ *in two cases: (a)* G^* *is strictly contained in* G *(non-overfitting), and (b)* G^* *is not a subset of* G *(non-underfitting). Furthermore, in the second case we consider that* $\{G^* \setminus \widehat{G}\} \neq \emptyset$*, otherwise it is similar to the first case.*

eventually almost surely as $n \to \infty$, proving that $\widehat{G} \neq G$ for all $G \supsetneq G^*$. Note that (2.24) is equivalent to say that

$$\log \hat{L}(G) - \log \hat{L}(G^*) < \lambda_n \left(\sum_{v \in V} |A|^{|G(v)|} - \sum_{v \in V} |A|^{|G^*(v)|} \right), \tag{2.25}$$

eventually almost surely as $n \to \infty$.

Moreover, from (2.19) we can write

$$\log \hat{L}(G) = \sum_{v \in V} \sum_{a_v, a_{G(v)}} N(a_v, a_{G(v)}) \log \hat{\pi}(a_v | a_{G(v)}), \tag{2.26}$$

and $\log \hat{L}(G^*)$ can be expressed similarly as in Equation (2.26). Thus, since $G^* \subsetneq G$ (i.e., G has more edges than G^*), the difference $\log \hat{L}(G) - \log \hat{L}(G^*)$ can be written in terms of the sum over the edges of G, that is,

$$\log \hat{L}(G) - \log \hat{L}(G^*) = \sum_{v \in V} \sum_{a_v, a_{G(v)}} N(a_v, a_{G(v)}) \log \frac{\hat{\pi}(a_v | a_{G(v)})}{\hat{\pi}(a_v | a_{G^*(v)})}.$$

By the definition of maximum likelihood estimators, for a fixed $v \in V$, we have that

$$\begin{aligned} \sum_{a_v, a_{G(v)}} N(a_v, a_{G(v)}) \log \hat{\pi}(a_v | a_{G^*(v)}) &\geq \sum_{a_v, a_{G(v)}} N(a_v, a_{G(v)}) \log \pi(a_v | a_{G^*(v)}) \\ &= \sum_{a_v, a_{G(v)}} N(a_v, a_{G(v)}) \log \pi(a_v | a_{G(v)}), \end{aligned}$$

and $\pi(a_v | a_{G^*(v)}) = \pi(a_v | a_{G(v)})$ holds since G^* is the true graph and G is considered to have more edges than G^*.

Therefore, the left-hand side of (2.25) can be upper-bounded by

$$\sum_{v\in V}\sum_{a_v,a_{G(v)}} N(a_v,a_{G(v)})\log\frac{\widehat{\pi}(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} = \sum_{v\in V}\sum_{a_{G(v)}} N(a_{G(v)})D\big(\widehat{\pi}(\cdot_v|a_{G(v)})\,;\,\pi(\cdot_v|a_{G(v)})\big),$$

where D denotes the Küllback-Leibler divergence (see Definition 14 in Appendix A). Thus, by Lemma 15,

$$\begin{aligned}\sum_{v\in V}\sum_{a_{G(v)}} N(a_{G(v)})D(\widehat{\pi}(\cdot_v|a_{G(v)})\,;\,\pi(\cdot_v|a_{G(v)}))&\\ \le \sum_{v\in V}\sum_{a_{G(v)}} N(a_{G(v)})\sum_{a_v\in A}\frac{[\,\widehat{\pi}(a_v|a_{G(v)})-\pi(a_v|a_{G(v)})\,]^2}{\pi(a_v|a_{G(v)})}.&\end{aligned} \tag{2.27}$$

Now, by Proposition 8 and Expression (2.27), for $\delta>0$, we have that

$$\begin{aligned}&\sum_{v\in V}\sum_{a_{G(v)}} N(a_{G(v)})D(\widehat{\pi}(\cdot_v|a_{G(v)})\,;\,\pi(\cdot_v|a_{G(v)}))\\ &\qquad\le \sum_{v\in V}\sum_{a_{G(v)}} N(a_{G(v)})\sum_{a_v\in A}\frac{\delta\log n}{N(a_{G(v)})\pi(a_v|a_{G(v)})}\\ &\qquad\le \sum_{v\in V}\sum_{a_{G(v)}}\sum_{a_v\in A}\frac{\delta\log n}{\pi(a_v|a_{G(v)})}\\ &\qquad\le \frac{\delta\log n}{\pi_{\min}}\sum_{v\in V}|A|^{G(v)}|A|\\ &\qquad\le \frac{\delta\log n}{\pi_{\min}}|A|\sum_{v\in V}|A|^{G(v)},\end{aligned} \tag{2.28}$$

and this holds eventually almost surely for $n\to\infty$, where

$$\pi_{\min}=\min_{v\in V}\Big\{\pi(a_v|a_{G(v)}):\pi(a_v|a_{G(v)})>0,a_v\in A,a_{G(v)}\in A^{|G(v)|}\Big\}. \tag{2.29}$$

Finally, in order to show (2.25), one can see from Expression (2.28) that it is enough to show that for $\lambda_n=c\log n$, there exists $\delta>0$ such that

$$\frac{\delta\log n}{\pi_{\min}}|A|\sum_{v\in V}|A|^{G(v)} < c\log n\Big(\sum_{v\in V}|A|^{|G(v)|}-\sum_{v\in V}|A|^{|G^*(v)|}\Big),$$

which is equivalent to

$$c\log n\Big(\sum_{v\in V}|A|^{|G(v)|}-\sum_{v\in V}|A|^{|G^*(v)|}\Big)-\frac{\delta\log n}{\pi_{\min}}|A|\sum_{v\in V}|A|^{G(v)}>0. \tag{2.30}$$

Note that, since $G^* \subsetneq G$, we have that $|E^*| < |E|$ and

$$\begin{aligned}\sum_{v\in V} |A|^{|G(v)|} - \sum_{v\in V} |A|^{|G^*(v)|} &= \sum_{v\in V}\left[\ |A|^{|G(v)|}\left(1 - \frac{\sum_{v\in V} |A|^{|G^*(v)|}}{\sum_{v\in V} |A|^{|G(v)|}}\right)\right] \\ &\geq \sum_{v\in V}\left[\ |A|^{|G(v)|} c'\right] \\ &\geq c' \sum_{v\in V} |A|^{|G(v)|},\end{aligned}$$

for $0 < c' < 1$. This inequality holds because

$$\frac{\sum_{v\in V} |A|^{|G^*(v)|}}{\sum_{v\in V} |A|^{|G(v)|}} < 1.$$

Thus in (2.30), δ can be chosen such that

$$|A|\frac{\delta}{\pi_{\min}} < cc',$$

that is, we take

$$\delta < cc'\pi_{\min}\frac{1}{|A|}$$

and therefore, we have that

$$\max_{G\supset G^*}\left\{\ \log \hat{L}(G) - \lambda_n \sum_{v\in V} |A|^{|G(v)|}\right\} < \log \hat{L}(G^*) - \lambda_n \sum_{v\in V} |A|^{|G^*(v)|}, \tag{2.31}$$

eventually almost surely as $n \to \infty$. This completes the proof of case (a), i.e., that $\{G^* \subsetneq \widehat{G}\}$ does not hold with probability converging to 1 as $n \to \infty$.

Now, we move to prove case (b). In this case, shown in Figure 2.2 (b), we need to prove that

$$\{G^* \not\subset \widehat{G}\},$$

where $\{G^* \setminus G\} \neq \varnothing$, does not hold eventually almost surely as $n \to \infty$. To prove this, we need to demonstrate that for any graph G such that $G \not\supset G^*$,

$$\log \hat{L}(G) - \lambda_n \sum_{v\in V} |A|^{|G(v)|} < \log \hat{L}(G^*) - \lambda_n \sum_{v\in V} |A|^{|G^*(v)|}, \tag{2.32}$$

eventually almost surely as $n \to \infty$. In order to prove that (2.32) holds, first we prove that

$$\log \hat{L}(G) - \lambda_n \sum_{v\in V} |A|^{|G(v)|} < \log \hat{L}(G_{\max}) - \lambda_n \sum_{v\in V} |A|^{|G_{\max}(v)|},$$

where $G_{\max}$ denotes the complete graph in V. Then, this inequality, together with the arguments presented in case (a), see Expression 2.31, will imply the desired result.

Note that we have

$$\begin{aligned}
\log \widehat{L}(G_{\max}) - \lambda_n \sum_{v\in V} |A|^{|G_{\max}(v)|} - \log \widehat{L}(G) + \lambda_n \sum_{v\in V} |A|^{|G(v)|} \\
&= \sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} N(a_v, a_{G_{\max}(v)}) \log \frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\widehat{\pi}(a_v|a_{G(v)})} \\
&\qquad - \lambda_n \sum_{v\in V} \left(|A|^{|G_{\max}(v)|} - |A|^{|G(v)|} \right) \\
&= n\Bigg[\sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} \frac{N(a_v, a_{G_{\max}(v)})}{n} \log \frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\widehat{\pi}(a_v|a_{G(v)})} \\
&\qquad - \frac{\lambda_n}{n} \sum_{v\in V} \left(|A|^{|G_{\max}(v)|} - |A|^{|G(v)|} \right) \Bigg]. \qquad (2.33)
\end{aligned}$$

One can see that for $\lambda_n = c \log n$, with $c > 0$, the second term in the brackets in (2.33) vanishes when $n \to \infty$, i.e.,

$$\frac{\lambda_n}{n} \sum_{v\in V} \left(|A|^{|G_{\max}(v)|} - |A|^{|G(v)|} \right) \xrightarrow[n\to\infty]{} 0.$$

Now, by adding

$$\frac{N(a_v, a_{G_{\max}(v)})}{n} \log \frac{\pi(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} = 0$$

into the first term of the sum in (2.33), we can write it as

$$\begin{aligned}
&\sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} \frac{N(a_v, a_{G_{\max}(v)})}{n} \log \frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\widehat{\pi}(a_v|a_{G(v)})} \\
&\quad = \sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} \frac{N(a_v, a_{G_{\max}(v)})}{n} \left(\log \frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\widehat{\pi}(a_v|a_{G(v)})} + \log \frac{\pi(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} \right) \\
&\quad = \sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} \frac{N(a_v, a_{G_{\max}(v)})}{n} \left(\log \frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\pi(a_v|a_{G(v)})} - \log \frac{\widehat{\pi}(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} \right) \\
&\quad = \underbrace{\sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} \frac{N(a_v, a_{G_{\max}(v)})}{n} \log \frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\pi(a_v|a_{G(v)})}}_{(1)} \\
&\qquad - \underbrace{\sum_{v\in V} \sum_{a_v, a_{G_{\max}(v)}} \frac{N(a_v, a_{G_{\max}(v)})}{n} \log \frac{\widehat{\pi}(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})}}_{(2)}. \qquad (2.34)
\end{aligned}$$

As highlighted in (2.34), we analyze this expression in two parts. The second term in the

right-hand side of (2.34) can be written as

$$\begin{aligned}
&\sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}} \frac{N(a_v,a_{G_{\max}(v)})}{n}\log\frac{\widehat{\pi}(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} \\
&\qquad = \sum_{v\in V}\sum_{a_v,a_{G(v)}} \frac{N(a_{G(v)})}{n}\widehat{\pi}(a_v|a_{G(v)})\log\frac{\widehat{\pi}(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} \\
&\qquad = \sum_{v\in V}\sum_{a_{G(v)}} \frac{N(a_{G(v)})}{n}\sum_{a_v}\widehat{\pi}(a_v|a_{G(v)})\log\frac{\widehat{\pi}(a_v|a_{G(v)})}{\pi(a_v|a_{G(v)})} \\
&\qquad = \sum_{v\in V}\sum_{a_{G(v)}} \frac{N(a_{G(v)})}{n} D\big(\widehat{\pi}(\cdot_v|a_{G(v)});\pi(\cdot_v|a_{G(v)})\big).
\end{aligned}$$

Further, observe that, by Lemma 15 and Proposition 8, for $\delta > 0$, we have

$$\begin{aligned}
&\sum_{v\in V}\sum_{a_{G(v)}} \frac{N(a_{G(v)})}{n} D(\widehat{\pi}(\cdot_v|a_{G(v)})\,;\,\pi(\cdot_v|a_{G(v)})) \\
&\qquad \le \sum_{v\in V}\sum_{a_{G(v)}} \frac{N(a_{G(v)})}{n}\sum_{a_v\in A}\frac{[\,\widehat{\pi}(a_v|a_{G(v)})-\pi(a_v|a_{G(v)})\,]^2}{\pi(a_v|a_{G(v)})} \\
&\qquad \le \sum_{v\in V}\sum_{a_{G(v)}} \frac{N(a_{G(v)})}{n}\sum_{a_v\in A}\frac{\delta\log n}{N(a_{G(v)})\pi(a_v|a_{G(v)})} \\
&\qquad \le |V||A|^{|V|}\frac{\delta}{\pi_{\min}}\frac{\log n}{n}\to 0.
\end{aligned}$$

as $n\to\infty$. Remember that π_{min} is defined in (2.29).

Since $\widehat{\pi}(a_v|a_{G_{max}(v)})$ are the maximum likelihood estimators of $\pi(a_v|a_{G_{max}(v)})$ and $G^*\subseteq G_{max}$, the first term in the right-hand side of (2.34) can be lower-bounded by

$$\begin{aligned}
&\sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}} \frac{N(a_v,a_{G_{\max}(v)})}{n}\log\frac{\widehat{\pi}(a_v|a_{G_{\max}(v)})}{\pi(a_v|a_{G(v)})} \\
&\qquad \ge \sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}} \frac{N(a_v,a_{G_{\max}(v)})}{n}\log\frac{\pi(a_v|a_{G_{\max}(v)})}{\pi(a_v|a_{G(v)})} \\
&\qquad = \sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}} \frac{N(a_v,a_{G_{\max}(v)})}{n}\log\frac{\pi(a_v|a_{G^*(v)})}{\pi(a_v|a_{G(v)})}.
\end{aligned} \tag{2.35}$$

By Proposition 7,

$$\frac{N(a_v,a_{G_{\max}(v)})}{n} = \widehat{\pi}(a_v,a_{G_{\max}(v)}) > \pi(a_v,a_{G_{\max}(v)}) - \sqrt{\frac{\delta\log n}{n}},$$

eventually almost surely as $n \to \infty$. Then, one can see that (2.35) can be lower-bounded by

$$\begin{aligned}
&\sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}}\left[\pi(a_v,a_{G_{\max}(v)})-\sqrt{\frac{\delta\log n}{n}}\right]\log\frac{\pi(a_v|a_{G^*(v)})}{\pi(a_v|a_{G(v)})}\\
&\qquad=\sum_{v\in V}\sum_{a_{G_{\max}(v)}}\pi(a_{G_{\max}(v)})\sum_{a_v}\pi(a_v|a_{G_{\max}(v)})\log\frac{\pi(a_v|a_{G^*(v)})}{\pi(a_v|a_{G(v)})}\\
&\qquad\qquad-\sqrt{\frac{\delta\log n}{n}}\sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}}\log\frac{\pi(a_v|a_{G^*(v)})}{\pi(a_v|a_{G(v)})}\\
&\qquad=\sum_{v\in V}\sum_{a_{G_{\max}(v)}}\pi(a_{G_{\max}(v)})D\big(\pi(\cdot_v|a_{G^*(v)});\pi(\cdot_v|a_{G(v)})\big)\\
&\qquad\qquad-\sqrt{\frac{\delta\log n}{n}}\sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}}\log\frac{\pi(a_v|a_{G^*(v)})}{\pi(a_v|a_{G(v)})}\\
&\qquad\geq\frac{1}{2}\sum_{v\in V}\alpha(v),
\end{aligned}\tag{2.36}$$

eventually almost surely as $n \to \infty$. Revisit Expression (2.22) for the definition of $\alpha(v)$. The last inequality above comes from the fact that

$$\sqrt{\frac{\delta\log n}{n}}\sum_{v\in V}\sum_{a_v,a_{G_{\max}(v)}}\log\frac{\pi(a_v|a_{G^*(v)})}{\pi(a_v|a_{G(v)})}\to 0$$

eventually almost surely as $n \to \infty$.

Therefore, since $\sum_{v\in V}\alpha(v) > 0$, we have from (2.36) that

$$\log\hat{L}(G)-\lambda_n\sum_{v\in V}|A|^{|G(v)|}\ \leq\ \log\hat{L}(G_{\max})-\lambda_n\sum_{v\in V}|A|^{|G_{\max}(v)|},$$

eventually almost surely as $n \to \infty$. Now, since $G^* \subset G_{\max}$, by case (a) we have that

$$\log\hat{L}(G)-\lambda_n|G|\ \leq\ \log\hat{L}(G_{\max})-\lambda_n\sum_{v\in V}|A|^{|G_{\max}(v)|}<\log\hat{L}(G^*)-\lambda_n\sum_{v\in V}|A|^{|G^*(v)|}$$

eventually almost surely as $n \to \infty$, and this concludes the proof for case (b). Thus, combining the two cases leads to

$$\max_{G:\{G\neq G^*\}}\ \log\hat{L}(G)-\lambda_n\sum_{v\in V}|A|^{|G(v)|}<\log\hat{L}(G^*)-\lambda_n\sum_{v\in V}|A|^{|G^*(v)|},$$

eventually almost surely as $n \to \infty$ and we conclude that $\widehat{G} = G^*$ eventually almost surely as $n \to \infty$ which concludes the proof. □

In summary, this chapter served as the fundamental groundwork for our study of multi-

variate stochastic processes satisfying the mixing condition (2.4). Additionally, we presented our proposed estimator of the underlying graph of this process and proved its consistency.

Chapter 3

Algorithms for Computing the Estimator

This chapter introduces and discusses the algorithms employed in this work to estimate the graph G^* (refer to Expression 2.21). The estimation involves finding the maximal value of

$$\log \widehat{L}(G) - \lambda_n \sum_{v \in V} |A|^{|G(v)|} \tag{3.1}$$

over the set of graphs G over V. For clarity, let us define

$$H(G) = \log \widehat{L}(G) - \lambda_n \sum_{v \in V} |A|^{|G(v)|}. \tag{3.2}$$

Notice that our focus is on determining the maximal value of $H(\cdot)$ and identifying the argument at which this maximum occurs. As previously mentioned, an existing method in the literature [see Leonardi *et al.*, 2023] involves estimating the neighborhood of each node and constructing the graph based on these estimated neighborhoods, adopting either a conservative or non-conservative approach. Alternatively, a direct but computationally demanding approach is the Exact algorithm outlined in Section 3.1, which evaluates Expression (3.1) for all possible graphs and selects the one maximizing this quantity. However, it is important to note that this algorithm suffers from exponential time complexity.

In Section 3.2, we introduce the Simulated Annealing algorithm, a heuristic optimization technique. This method begins with an initial graph solution and iteratively explores the solution space by randomly perturbing the current solution, accepting perturbations based on a probability function. As discussed later, this approach may require a substantial number of iterations to find the solution, i.e., the estimated graph. To mitigate this challenge, we also explore a greedy algorithm in Section 3.3, employing stepwise model selection. The Stepwise algorithm, whether Forward or Backward, begins with a graph without edges (or a complete graph) and selectively adds (or removes) edges based on their contributions to maximize the penalized pseudo-loglikelihood function.

The primary goal of this chapter is to present each algorithm mentioned above. In Chapter 4, we explore different simulation scenarios, providing a detailed discussion of each implementation's specifics and unique aspects. Additionally, we conduct a comparative study

Algorithm 1 Exact algorithm

$\mathcal{G} \leftarrow \{G = (V, E) : E \subseteq \{V \times V\} \setminus \{(v, v) : v \in V\}\}$
$\ell_{\max} \leftarrow -\infty$
for $G \in \mathcal{G}$ **do**
    $\ell \leftarrow \log \hat{L}(G) - \lambda_n \sum_{v \in V} |A|^{|G(v)|}$
    if $\ell > \ell_{\max}$ **then**
        $\ell_{\max} = \ell$
        $\hat{G} \leftarrow G$
    end if
end for
return $\hat{G}$

among these algorithms to evaluate their performance over different penalizing constant values.

3.1 Exact Algorithm

Given the task of maximizing Expression (3.1), the primary method that comes to mind involves an exhaustive search across all potential graphs $G = (V, E)$ within the defined setting, where $E \subseteq \{V \times V\} \setminus \{(v, v) : v \in V\}$. The algorithm starts by defining the set $\mathcal{G}$ encompassing all graphs with $d = |V|$ nodes, specified as

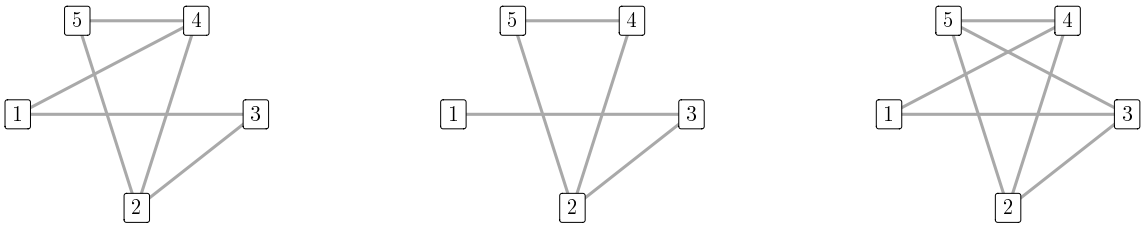
$$\mathcal{G} = \{G = (V, E) : E \subseteq \{V \times V\} \setminus \{(v, v) : v \in V\}\}, \tag{3.3}$$

and initializes the maximum pseudo-likelihood value $\ell_{\max}$ to $-\infty$.

Then, it calculates the penalized pseudo-likelihood for each $G \in \mathcal{G}$ and sets $\widehat{G}$ as the graph at which Expression (3.1) achieves its maximum; let us refer to this value as $\ell_{\max}$. These steps are shown in Algorithm 1.

Note that the number of candidates to seek for the one that maximizes (3.1) is $2^{d(d-1)/2}$. Its computational complexity is a significant drawback, specifically as the number of nodes in the graph increases. Hence, the exhaustive search performed by the Exact algorithm for the best subset becomes computationally prohibitive, leading to impractical runtimes for datasets with a large number of variables. Additionally, the algorithm is susceptible to overfitting, particularly in scenarios with small sample sizes, potentially resulting in poor generalization. Another concern is the risk of selection bias, as the algorithm may favor subsets that perform well on the training data but might not accurately represent the true relationships in the broader population. See more details in James *et al.* [2021].

Despite the computational inefficiency of the Exact algorithm, we will consider it in our simulation studies presented in Chapter 4. This is because the Exact algorithm ensures that the estimated graph maximizes the penalized pseudo-log-likelihood function. Thus, we will use its results for comparison with alternative methods discussed in the following sections: the Simulated Annealing algorithm and the Stepwise edge selection algorithm.



(a) **(b)** **(c)**

Figure 3.1: *An example of (a) a graph with five nodes, denoted as G, and two possible neighbors: (b) a neighboring graph of G, without the edge linking nodes 1 and 4, (c) another possible neighbor of graph G, now adding an edge between nodes 3 and 5.*

3.2 Simulated Annealing Algorithm

As discussed earlier, the Exact algorithm lacks numerical efficiency. Therefore, we introduce an alternative approach, the Simulated Annealing algorithm. Referring back to Expression (3.2), let

$$H^* = \max_{G \in \mathcal{G}} H(G)$$

represents the maximum penalized pseudo-likelihood among all possible graphs in the set $\mathcal{G}$ and let

$$\mathcal{H} = \{G \in \mathcal{G} : H(G) = H^*\}$$

be the set of all graphs for which the penalized pseudo-likelihood attains the maximum value H^*. Our goal is to determine H^* as well as an element of the set $\mathcal{H}$. In this section, we demonstrate how to achieve this.

Before presenting the algorithm itself, let us define the concept of a graph's neighbor. Two graphs, G and G', are considered neighbors if they differ by only a single edge. Figure 3.1 provides two examples of neighbors for the graph in the left panel. The central panel shows a neighboring graph obtained by removing the edge between nodes 1 and 4 from the original graph. In the right panel, nodes 3 and 5 were connected, adding an edge to the original graph.

The Simulated Annealing is an iterative algorithm and operates as follows. It starts by defining an initial state $G_1 \in \mathcal{G}$ for the Markov chain. This state can be chosen randomly or based on prior knowledge about the graph being estimated. For each $n \in \{1, 2, 3 \dots m\}$, a

Algorithm 2 Simulated annealing algorithm

1: Initialize $G_1 \in \mathcal{G}$.
2: Randomly choose a neighbor G_1' of G_1.
3: Transition to G_1' with probability $\min\left\{1, \frac{\exp\{\eta_n H(G_2)\}/|N(G_2)|}{\exp\{\eta_n H(G_1)\}/|N(G_1)|}\right\}$, where $\eta_n = C\log(1+n)$, for $n \geq 1$, where $C > 0$.
4: Repeat steps 2 and 3 m times.

neighbor G_n' of G_n is randomly selected, here we consider that any neighbor of G_n is equally likely to be chosen. The next state of the chain is either G_n' with probability

$$\min\left\{1, \frac{\exp\{\eta_n H(G_n')\}/|N(G_n')|}{\exp\{\eta_n H(G_n)\}/|N(G_n)|}\right\}, \tag{3.4}$$

or it remains G_n, where $|N(G)|$ is the number of neighbors of G, and η_n, for $n \geq 1$, is a prescribed set of values that start small, which leads to a large number of changes in state, and then grow. This process is repeated m successive times to generate the states $G_1, G_2, \ldots, G_m$. We can then estimate H^* by

$$\max_{i=1,\ldots,m} H(G_i)$$

and if the maximum occurs at G_i^*, then it is taken as the estimated point in $\mathcal{H}$. These steps are outlined in Algorithm 2, and more details are available in Ross [2006], Section 10.4.

A computationally practical choice of η_n that also mathematically results in convergence is to let $\eta_n = C\log(1+n)$, where $C > 0$ is a fixed constant [Ross, 2006]. For further details and justification, refer to Besag *et al.* [1995] and Diaconis and Holmes [1995]. Note that Expression (3.4) can be simplified if each graph has the same number of neighbors. Then when state is G_n, one of its neighbors, G_n' for instance, is randomly chosen, the chain moves to state G_n' with probability $\exp\{\eta_n[H(G_n') - H(G_n)]\}$ or remains in state G_n otherwise.

As one can see, some choices must be made before executing Algorithm 2. The first is the number m of iterations of the Simulated Annealing, the second is the value of constant C, and the third is the initial state G_1 of the chain. These values can interfere in the final graph estimate. We discuss in more detail these aspects in Chapter 4 considering simulated data in several scenarios.

3.3 Stepwise Edge Selection Algorithm

The Stepwise edge selection criteria is a computationally efficient alternative to the Exact algorithm presented in Section 3.1. While the Exact algorithm (also known as best subset selection procedure) considers all $2^{d(d-1)/2}$ possible graphs, the Stepwise approach considers a much smaller set of models. Remember that $d = |V|$.

The Backward Stepwise edge selection begins with the complete graph containing edges linking all nodes and then removes edges from the graph, one at a time until no more edge is left. In particular, at each step, the edge that, upon its removal, results in the greatest improvement to the fit, is removed from the graph. The stopping condition of these steps occurs when no improvement is made in maximizing the penalized pseudo-likelihood. More formally, the Backward Stepwise selection procedure is outlined in Algorithm 3.

Alternatively, the Forward Stepwise edge selection is initiated with a graph containing no edges. Then, edges are added to the graph one by one until all possible edges are included.

Algorithm 3 Backward Stepwise edge selection

1: Let G_d denote the complete graph.
2: **for** $k = d, d-1 \ldots, 1$ **do**
3: Consider all $k(k-1)/2$ graphs that contain all but one of the edges of graph G_k.
4: Choose the *best* among these $k(k-1)/2$ models, and call it G_{k-1}. Here *best* is defined as having the highest penalized pseudo-likelihood (3.1).
5: **if** No improvement is made in maximizing the penalized pseudo-likelihood **then**
6: **break**
7: **end if**
8: **end for**
9: Select among $G_0, G_1, \ldots, G_d$ the graph with maximum penalized pseudo-likelihood.

Algorithm 4 Forward Stepwise edge selection

1: Let G_0 denote the empty graph.
2: **for** $k = 0, \ldots, d$ **do**
3: Consider all $(d-k)(d-k-1)/2$ graphs that adds exactly one edge in graph G_k
4: Choose the *best* among these $(d-k)(d-k-1)/2$ models, and call it G_{k+1}. Here *best* is defined as having the highest penalized pseudo-likelihood (3.1).
5: **if** No improvement is made in maximizing the penalized pseudo-likelihood **then**
6: **break**
7: **end if**
8: **end for**
9: Select among $G_0, G_1, \ldots, G_d$ the graph with maximum penalized pseudo-likelihood.

Specifically, at each step, the edge that provides the most significant additional enhancement to the fit is incorporated into the graph. The stopping condition of these steps occurs when no improvement is made in maximizing the penalized pseudo-likelihood. Algorithm 4 gives a formal version of the Forward Stepwise selection procedure.

3.4 The `MixingGraph` R package

The algorithms for estimation introduced in this chapter have been implemented in the R package `MixingGraph`. You can access the package through the development version, which is currently accessible at github.com/magnotairone/MixingGraph. Examples demonstrating the package's usage are also provided on this page.

These estimation algorithms will be evaluated in Chapter 4 not only in terms of quality of estimation but also computationally in terms of complexity considering practical scenarios with simulated data.

Chapter 4

Simulation Studies

This chapter is dedicated to comprehensively evaluating the algorithms introduced in Chapter 3 concerning their estimation performance. We use a simulation scheme to show the convergence of the proposed graph estimator presented in Expression (2.21) considering each algorithm individually. By thoroughly evaluating the performance of the proposed estimator through simulation studies, we aim to provide evidence of its effectiveness and robustness in real-world applications. To conduct this simulation study, we utilize the R language [see R Core Team, 2022], which provides a flexible and powerful tool for statistical computing.

Our exploration of the algorithms' performance encompasses two distinct settings. Initially, we address the scenario where a fixed true graph exists. Conditional probabilities are defined on the basis of this graph, leading to the generation of synthetic data that are used to assess how close our estimation is to the "true" graph. The primary objective of this initial scenario is to analyze the proposed estimator's performance under stability conditions. This investigation is detailed in Section 4.1.

Subsequently, our attention shifts to an investigation of performance relative to the number of edges within the graph. To undertake this study, the number of vertices d is held constant while a series of random graphs is generated, varying the number of edges over a range from 1 to $d(d-1)/2$. Based on each generated graph, we define conditional probabilities and then generate sample data, which is then subjected to the algorithms that implement our proposed method in order to recover the graph. This process is repeated several times for each scenario, permitting the consolidation of results in terms of the selected metrics (including underestimation error, overestimation error, and overall error for each case), which are defined ahead. In Section 4.2, we present the findings obtained from this exploration.

Overall, the synthetic graph examples presented in this section allowed us to evaluate our proposed estimator's performance comprehensively. The results showcase not only its effectiveness but also its potential utility in real-world applications.

4.1 Fixed Graph Scenario

Within this section, we introduce two illustrative instances of graphical models. In both cases, the joint probabilities are established by the multiplication of conditional probabilities $p(x_i|x_W)$ where the set W does not have to be the set of neighbors of vertex i. Nevertheless, the selection of the joint probability's factorization delineates the structures of conditional dependencies—essentially shaping the graph's arrangement. This factorization facilitates a more straightforward methodology for generating samples, as discussed in Section 4.1.1.

The first example (Example 11) presented in this section functions as a straightforward scenario to demonstrate the estimation algorithms under examination within this thesis. Based on this example, we discuss specific characteristics associated with each algorithm presented in Section 4.1.2. Shifting to the second example (Example 12), presented in Section 4.1.5, the graph structure is motivated by the application of our method to the context of São Francisco river water flow measurements (see Section 5.1).

Example 11. *Let us consider a scenario involving five distinct random variables denoted by $X_1, \ldots, X_5$, where each variable assumes values in the set $A = \{0, 1\}$. The joint probability function of these variables is expressed as follows:*

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_3)p(x_1|x_3)p(x_2|x_1, x_3)p(x_4|x_3)p(x_5|x_3). \tag{4.1}$$

This equation highlights the dependencies among the variables through conditional probabilities. From (4.1) we can compute $p(x_1|x_2, x_3, x_4, x_5)$ in order to identify the neighborhood of node 1. That is

$$\begin{aligned}
p(x_1|x_2, x_3, x_4, x_5) &= \frac{p(x_1, x_2, x_3, x_4, x_5)}{\sum_{x_1 \in A} p(x_1, x_2, x_3, x_4, x_5)} \\
&= \frac{p(x_3)p(x_1|x_3)p(x_2|x_1, x_3)p(x_4|x_3)p(x_5|x_3)}{p(x_4|x_3)p(x_5|x_3)\sum_{x_1 \in A} p(x_3)p(x_1|x_3)p(x_2|x_1, x_3)} \\
&= \frac{p(x_3)p(x_1|x_3)p(x_2|x_1, x_3)}{\sum_{x_1 \in A} p(x_3)p(x_1|x_3)p(x_2|x_1, x_3)} \\
&= \frac{p(x_1, x_3)p(x_2|x_1, x_3)}{\sum_{x_1 \in A} p(x_1, x_3)p(x_2|x_1, x_3)} \\
&= \frac{p(x_1, x_2, x_3)}{\sum_{x_1 \in A} p(x_1, x_2, x_3)} \\
&= p(x_1|x_2, x_3).
\end{aligned}$$

Therefore, the set of neighbors of vertex 1 is $\{2, 3\}$. We can use this very same argument to

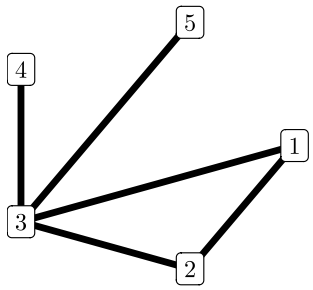


Figure 4.1: *Graph of Example 11, with vertices $X_1, \ldots, X_5$. For simplicity, instead of X_i the figure shows only the node index, i.e., i, $1 \leq i \leq 5$.*

find out that

$$p(x_2|x_1, x_3, x_4, x_5) = p(x_2|x_1, x_3),$$
$$p(x_4|x_1, x_2, x_3, x_5) = p(x_4|x_3),$$
$$p(x_5|x_1, x_2, x_3, x_4) = p(x_5|x_3).$$

That is, vertex 2 is the neighbor of vertices $\{1, 3\}$, vertex 4 is the neighbor of vertex $\{3\}$, vertex 5 is also the neighbor of vertex $\{3\}$ and since vertex 3 is the neighbor of all other nodes, its neighborhood is the set $\{1, 2, 4, 5\}$. Figure 4.1 shows the graph of conditional dependencies for this example, with edges linking all neighbors of each node.

It is necessary to establish theoretical values for the marginal and conditional distributions described above to generate sample data for this example using the Gibbs sampler algorithm. In our illustrative scenario, we assigned specific numerical values to the marginal distribution of X_3, which are outlined in Table 4.1. Moreover, the conditional probabilities for $X_1|X_3$, $X_4|X_3$, $X_5|X_3$, and $X_2|X_1, X_3$ are presented in Tables 4.2, 4.3, 4.4, and 4.5, respectively. These particular values have been chosen randomly and can be adjusted as necessary.

4.1.1 Data Generation

We use the Gibbs sampler algorithm, originated in the studies of Geman and Geman [1984] and Gelfand and Smith [1990], to generate the samples. The Gibbs sampler is a widely used Monte Carlo Markov Chain (MCMC) method that provides a way to estimate complex joint distributions by iteratively drawing samples from the conditional distribution of each variable given the current values of all the other variables. By sampling from the conditional distributions, the Gibbs sampler can generate a sequence of samples whose distribution converges to the target distribution.

Remember that the joint probability function of Example 11 can be factorized as shown in (4.1). We consider the following steps to use the Gibbs sampler algorithm to sample from (4.1):

x_3	0	1
$p(x_3)$	1 / 2	1 / 2

Table 4.1: *Marginal distribution of X_3 for Example 11.*

x_1	0	1
$p(x_1\|x_3 = 0)$	1 / 2	1 / 2
$p(x_1\|x_3 = 1)$	1 / 3	2 / 3

Table 4.2: *Conditional distribution of $X_1|X_3$ for Example 11.*

x_4	0	1
$p(x_1\|x_3 = 0)$	1 / 5	4 / 5
$p(x_1\|x_3 = 1)$	3 / 5	2 / 5

Table 4.3: *Conditional distribution of $X_4|X_3$ for Example 11.*

x_5	0	1
$p(x_1\|x_3 = 0)$	2 / 3	1 / 3
$p(x_1\|x_3 = 1)$	1 / 2	1 / 2

Table 4.4: *Conditional distribution of $X_5|X_3$ for Example 11.*

x_2	0	1
$p(x_2\|x_1 = 0, x_3 = 0)$	1 / 2	1 / 2
$p(x_2\|x_1 = 1, x_3 = 0)$	3 / 4	1 / 4
$p(x_2\|x_1 = 0, x_3 = 1)$	1 / 4	3 / 4
$p(x_2\|x_1 = 1, x_3 = 1)$	1 / 3	2 / 3

Table 4.5: *Conditional distribution of $X_2|X_1, X_3$ for Example 11.*

1. Initialize the variables

$$
\begin{aligned}
X_3^0 &\sim p(X_3),\\
X_1^0 &\sim p(X_1|X_3^0),\\
X_2^0 &\sim p(X_2|X_1^0, X_3^0),\\
X_4^0 &\sim p(X_4|X_3^0),\\
X_5^0 &\sim p(X_5|X_3^0).
\end{aligned}
$$

2. For $j = 1, 2 \ldots$, generate

$$
\begin{aligned}
X_1^j &\sim p(X_1|X_2^{j-1}, X_3^{j-1}),\\
X_2^j &\sim p(X_2|X_1^j, X_3^{j-1}),\\
X_4^j &\sim p(X_4|X_3^{j-1}),\\
X_5^j &\sim p(X_5|X_3^{j-1}),\\
X_3^j &\sim p(X_3|X_1^j, X_2^j, X_4^j, X_5^j).
\end{aligned}
$$

Step 2 above should be repeated accordingly to generate a sample size with the size needed.

In the case of this example, the Gibbs sampler was set to perform 15,000 iterations, with a burn-in period of 5,000 iterations and thus 10,000 observations forming the final

sample $s_1, \ldots, s_{10,000}$. The R function generating a sample for Example 11 can be found in Section C.2 of Appendix C. From this initial sample of size 10,000, smaller samples of size N were extracted, where $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$. Each subsample S_N was taken from the initial state of the chain to the N-th state, represented as $(s_1, \ldots, s_N)$. Figure 4.2 illustrates this sampling scheme.

4.1.2 Estimation

In this section, we evaluate the performance of the algorithms presented in Chapter 3, with a focus on the quality of estimation considering the data generated following the steps described in Section 4.1.1. Specifically, we work with five sets of sample data, each with different sizes, namely $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$. For each algorithm (Exact, Simulated Annealing, Forward Stepwise, and Backward Stepwise), we execute the procedures with various values for the penalizing term $\lambda = c \log N$, where $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$ and N represents the sample size.

Exact algorithm

In the context of the Exact algorithm, which computes the penalized log-pseudo-likelihood for all possible graphs, the computational burden remains manageable as we are dealing with a relatively small number of vertices ($d = 5$) – yielding a total of 1,024 potential graphs.

To begin with an illustration, we fixed the sample size in $N = 5{,}000$ and computed the penalized pseudo-log-likelihood for all possible graphs, considering a set of penalizing constant values c. Then, we fix the number of edges and choose the graph with the maximum value of $H(G)$. Figure 4.3 shows the effect of the penalization in the log pseudo-likelihood function (refer to (2.20)), that is, the penalized logarithm of the pseudo-likelihood function $\log L(\hat{G})$ concerning the number of edges present in the graph (ranging from 0 to 10). The absence of penalization is illustrated by the solid line at $c = 0$, which is an increasing function. It is evident that, in this case, the true graph (Figure 4.1) cannot be recovered. However, note that the rate of increase rapidly diminishes when the number of edges in the graph exceeds five, the number of edges in the true graph.

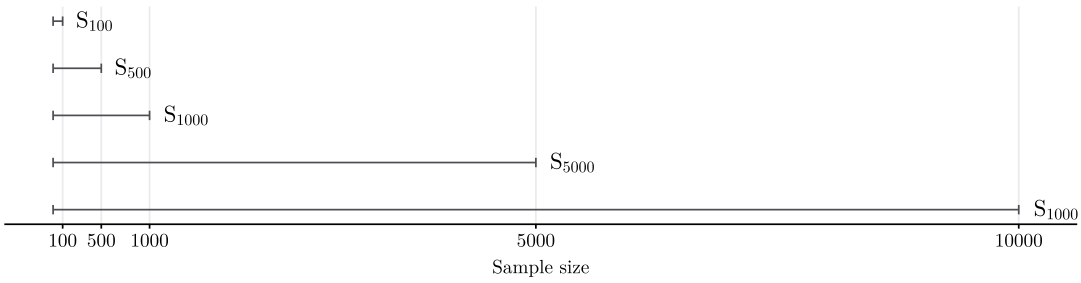


Figure 4.2: *Scheme showing how the subsamples of size $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ were extracted from the initial chain of size 10,000.*

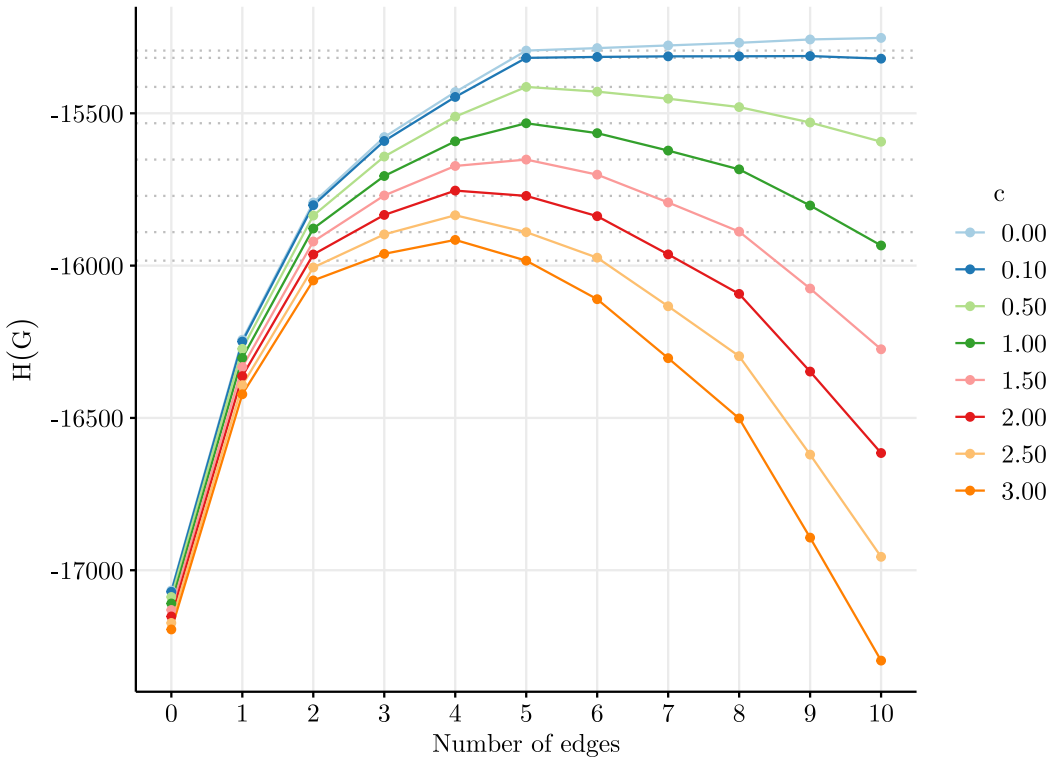


Figure 4.3: *The effect of the penalization term in the log-likelihood function in terms of the number of edges, considering the Exact algorithm* $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$, *in a sample of size 5,000. The dotted horizontal lines highlight the value of* $H(G)$ *when the number of edges in the estimated graph equals five, the value in the true graph.*

Furthermore, Figure 4.3 illustrates the effect of the penalization term in Expression (3.2) across various values of $c \in \{0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. The dotted horizontal lines highlight the value of $H(G)$ defined in (3.2) when the number of edges in the estimated graph equals five, consistent with the true graph. With this aid, we observe that for $c \in \{0.1, 0.5, 1.0, 1.5\}$, the estimation outcomes are accurate (the corresponding estimated graphs are shown in Figure 4.4). However, for $c \in \{2.0, 2.5, 3.0\}$, the correct recovery of the true graph was not achievable, as the maximum penalized pseudo-likelihood was obtained for a graph with four edges.

Illustrated in Figure 4.4, each subgraph is an outcome of the exact algorithm corresponding to a pair (N, c) of values of sample size and penalizing constant. For simplicity, we remove the node number in each graph, but the node arrangement in the figure mirrors that of Figure 4.1.

Considering the terms of the sum in the log pseudo-likelihood function (2.20), we can say that sample sizes of 100 and 500 are small, and thus, the results may not lead to good estimation. In the first column of Figure 4.4, we can clearly see an example of an overestimation problem (when the set of estimated edges is greater than the set of edges in the true graph) for $c = 0.10$. As the penalizing term increases, the underestimation problem arises (when the set of estimated edges is smaller than the true graph's set of edges). Evidently, smaller values of the penalizing term tend to yield graphs characterized

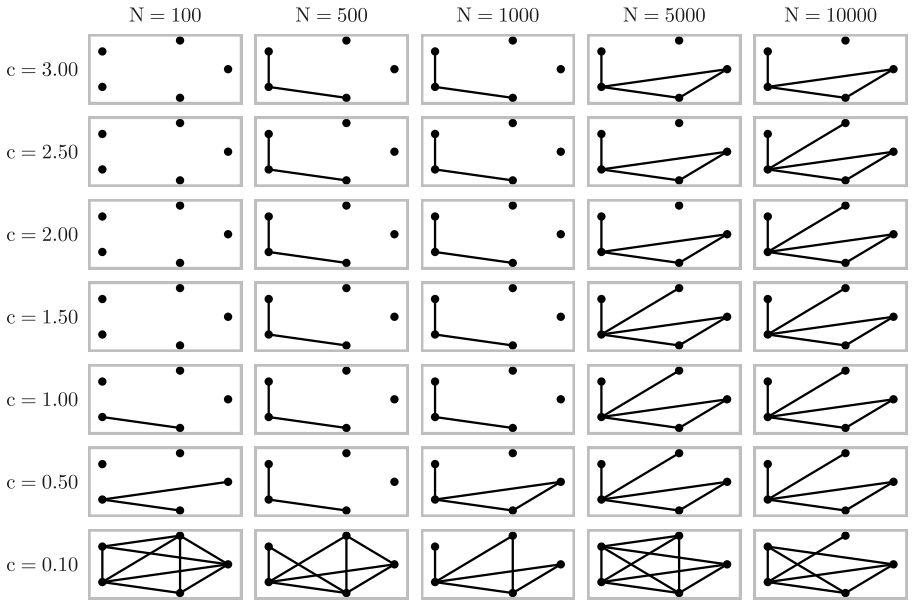


Figure 4.4: *Exact algorithm results for Example 11, considering several scenarios, namely* $N \in \{100, 500,\ 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing term* $c \in \{0, 0.1, 0.5, 1.0,\ 1.5, 2.0, 2.5, 3.0\}$.

by a substantial number of edges. In this particular example, regardless of the sample size, a penalizing constant of $c = 0.10$ consistently yields an overestimated graph.

The case where $N = 1{,}000$ shows an interesting result: no correct estimation for all the c values considered. In this case, one could refine the grid in the search for the correct penalizing constant value.

Conversely, the graph is correctly estimated for a sample of size 5,000 and for penalizing constants $c \in \{0.5, 1.0, 1.5\}$, and for a sample of size 10,000 coupled with penalizing constants $c \in \{0.5, 1.0, 1.5, 2.0, 2.5\}$. Nevertheless, when $c = 3.0$, all scenarios yield an erroneous underestimation.

Simulated Annealing

Presented in Section 3.2, the Simulated Annealing algorithm involves iteratively exploring potential solutions while accepting less favorable ones with a decreasing probability over time to eventually converge to an optimal or near-optimal solution. This algorithm has some hyperparameters, which are parameters that are not learned from the data but are set prior to training a statistical learning model. These parameters are essential for controlling the model's learning process and behavior.

In this context, the hyperparameters consist of the initial state of the Markov chain, denoted as G_1; the cooling rate, denoted as C; and the total number of iterations m for the algorithm.

Various combinations of these hyperparameter values were evaluated, but none yielded satisfactory results. Consequently, we present the outcomes based on one specific configuration that correctly recovered the true graph in one specific case: the initial state of the

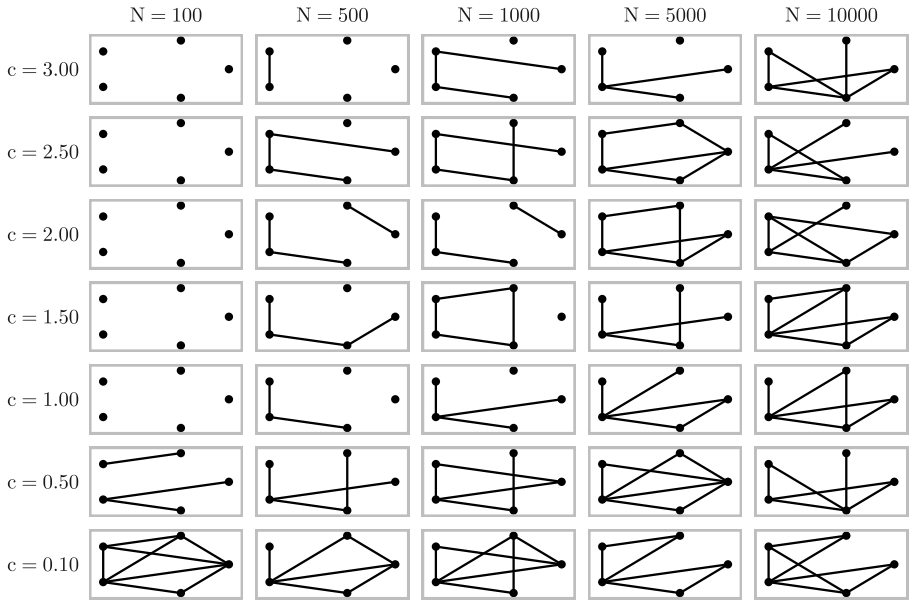


Figure 4.5: *Simulated annealing algorithm results for Example 11, considering several scenarios, with samples of size* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant* $c \in \{0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$.

chain is set as the empty graph (with no edges), the cooling rate is fixed at $C = 0.001$, and the total number of iterations is set to $m = 100$. We selected this specific chain size value because opting for a larger one would not be logical, as it would necessitate precisely 1024 steps to the exhaustive algorithm to explore the entire range of potential graphs.

Figure 4.5 displays the outcomes of the Simulated Annealing algorithm following the described configuration. It is noteworthy that, in general, for small sample sizes (specifically $N = 100$ and $N = 500$), the estimated graphs do not closely resemble the true graph, except for instances when $c \in \{0.10, 0.50\}$ and $N = 500$, where some edges are correctly estimated while others are overestimated. As the sample size increases, more of the estimated graphs start to resemble the true one. However, unlike the Exact algorithm, only one of them has fully captured the correct underlying graph, for $c = 1.00$ and $N = 5{,}000$. This result led us to consider the Stepwise algorithms, the results of which are presented ahead.

Backward Stepwise

As defined in Section **??**, the Backward Stepwise algorithm starts with the complete graph as its initial guess. It progressively eliminates the edges that help increase the penalized pseudo-log-likelihood, halting when further increases are unattainable. Depicted in Figure 4.6, this algorithm's progression is shown for a sample of size 5,000 and penalizing constant $c = 0.5$. The process encompasses six successive steps before reaching the halting criterion.

The outcomes of each scenario are presented in Figure 4.7. Each subgraph represents the estimated graph corresponding to a particular pair of values (N, c). In general, it is possible to observe that smaller values of c lead to substantial overestimation errors. As the value of c

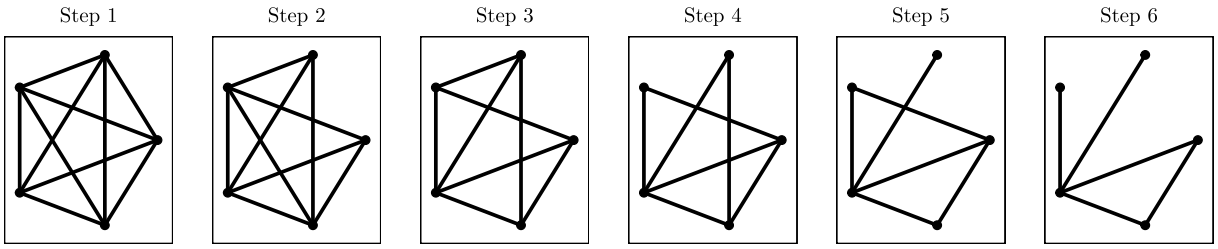


Figure 4.6: *Steps of the Backward Stepwise selection algorithm. Commencing with the full graph, it successively removes edges to increase the penalized pseudo-log-likelihood, stopping when further improvements become unattainable. These steps were obtained using a sample of size* 5,000 *and* $c = 0.5$.

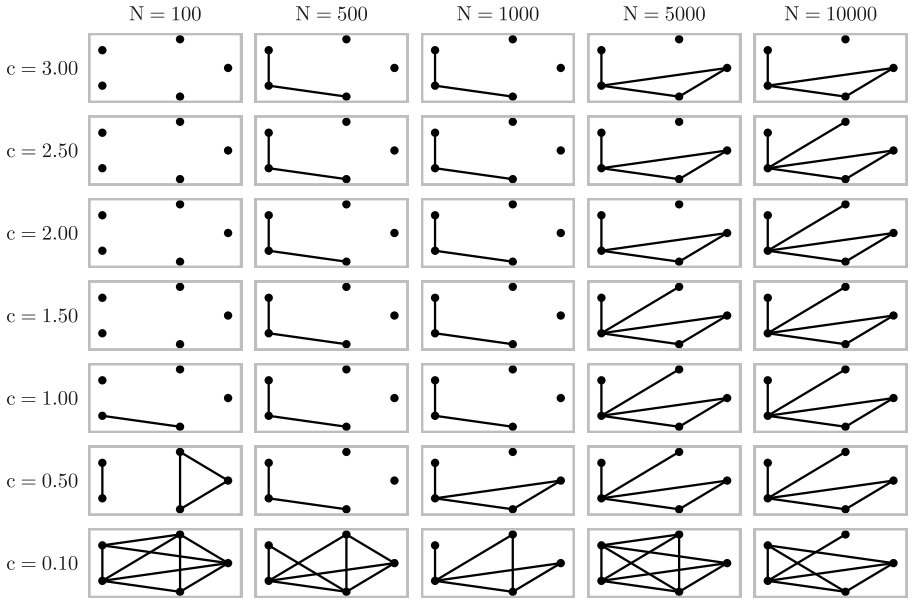


Figure 4.7: *Backward Stepwise algorithm results for Example 11, considering several scenarios, with a sample of size* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant* $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$.

increases, the overestimation error diminishes; however, in this progression, underestimation errors arise, specially for small values of N and large values of c.

Now, for a sample of size $N \in \{5{,}000, 10{,}000\}$, the Backward Stepwise algorithm produces accurate estimations for some values of c, but presents increasing underestimation error as c increases and overestimation when $c = 0.1$.

Forward Stepwise

This algorithm starts with the null graph (with no edges) and progressively adds edges that increase the penalized pseudo-log-likelihood function $H(G)$ defined in (3.2), see details in Section **??**. The iteration stops when further increases are unattainable. Figure 4.8 shows the progression of the Forward Stepwise algorithm for a sample of size $N = 5{,}000$ and considering $c = 0.5$. The process encompasses six successive steps before satisfying the termination criterion.

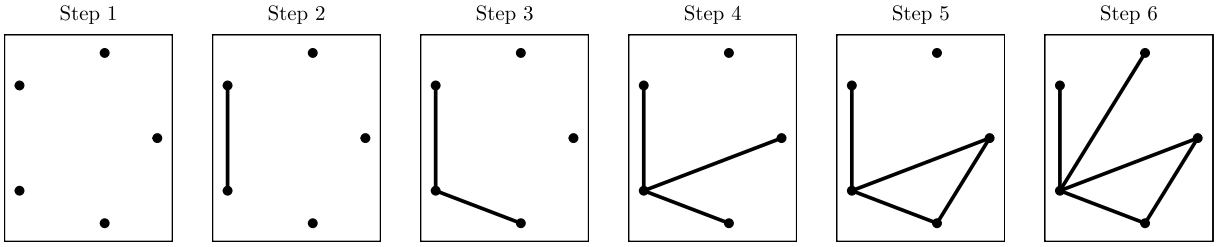


Figure 4.8: *Stages of the Forward Stepwise selection algorithm. Starting with the null graph, it sequentially incorporates edges that increase the penalized pseudo-log-likelihood, stopping when further increases are unattainable. These steps were obtained considering a sample of size 5,000 and* $c = 0.5$.

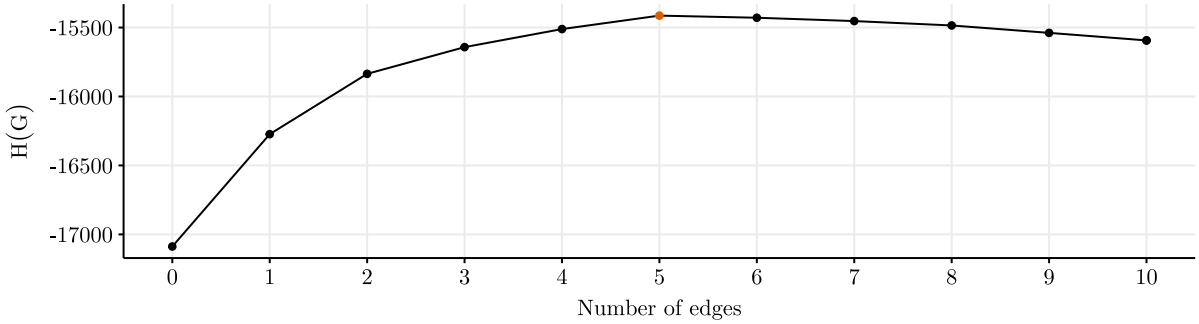


Figure 4.9: *Penalized log-likelihood function* $H(G)$ *in terms of the number of edges, considering a sample of size* $N = 5{,}000$ *and* $c = 0.50$ *for the Forward Stepwise algorithm. The highlighted orange dot represents the graph at which the maximum of this function is attained.*

Figure 4.9 shows the penalized log pseudo-likelihood function, defined in Expression (3.2), considering the Forward Stepwise algorithm, a sample of size $N = 5{,}000$, and penalizing constant $c = 0.5$. The highlighted orange dot represents the graph at which the maximum of this function is attained, which is exactly the same number of edges as in the true graph. This estimated graph is shown in Figure 4.10 (second row from bottom to the top, fourth column), note that it has the same structure as the true graph.

In Figure 4.10, each subgraph depicts the outcome for a specific pair of values (N, c). For $N = 100$, smaller c values lead to substantial overestimation errors. As c increases, overestimation errors vanish; however, underestimation errors emerge for all cases considered. This pattern holds the same for $N = 500$, and $N = 1{,}000$ with underestimation errors rising for $c \geq 0.5$.

Note that as the sample size grows, algorithmic errors diminish for certain values of c. For a sample of size $N = 5{,}000$, the Forward Stepwise algorithm produces accurate estimations for the cases considered where $0.5 \leq c \leq 1.5$, but underestimation arises as c increases. In the case of $N = 10{,}000$, the overestimation error displays a diminishing trend as c increases, vanishing when $c \geq 0.5$, however when $c = 3{,}0$, underestimation error arises.

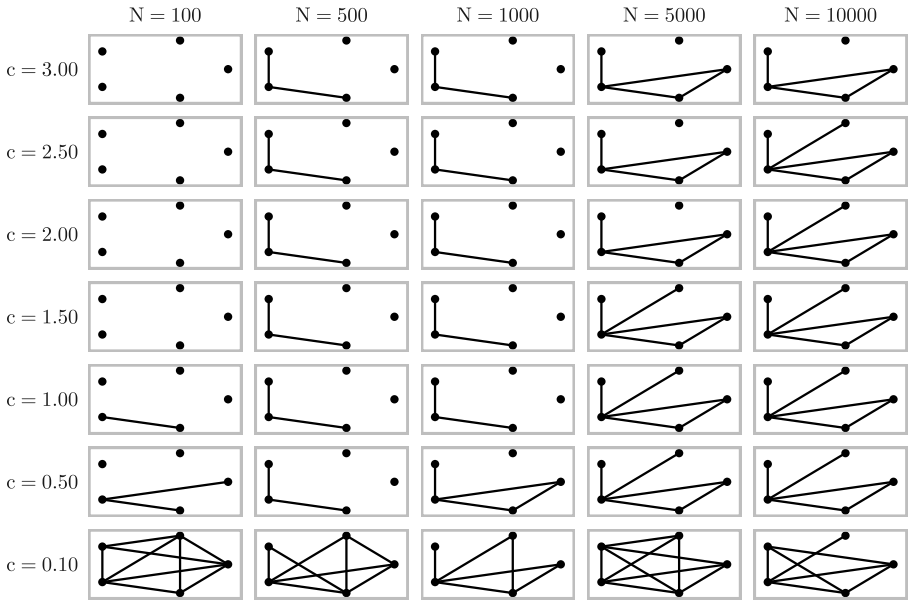


Figure 4.10: *Forward Stepwise algorithm results for Example 11, considering several scenarios, namely* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant* $c \in \{0, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$.

4.1.3 Overall Performance in Several Replications

Applying the data generation process outlined in Section 4.1.1, we proceed to assess the algorithms across samples of different sizes: $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$. Ten distinct samples are generated for each sample size N, 50 in total. Subsequently, these 50 samples undergo evaluation through the Exact, Simulated Annealing, Forward Stepwise, and Backward Stepwise algorithms. This evaluation considers a range of penalizing constant values $c = \{0.1, 0.5,\ 1.0, 1.5, 2.0, 2.5, 3.0\}$, yielding a result comprising 350 scenarios for each algorithm considered (five sample sizes, ten replications, and seven penalizing values). The primary motivation to perform this study is to not only rely on a single realization as presented in Section 4.1 but to consider different repetitions of the same scenario to observe the average performance of each algorithm.

To facilitate this comprehensive analysis, we define metrics to quantify two distinct errors: *underestimation error* (when the estimated graph does not contain all edges of the true graph), *overestimation error* (when the estimated graph has edges that are not in the true graph), and a weighted average of both, referred to as *total error*.

Let $G = (V, E)$ be the true graph and $\hat{G} = (V, \hat{E})$ an estimate. Also, let $v, w \in V$. We define the underestimation error as

$$ue(G, \hat{G}) = \frac{\sum_{(v,w)} \mathbb{1}\{(v,w) \in E \text{ and} (v,w) \notin \hat{E}\}}{\sum_{(v,w)} \mathbb{1}\{(v,w) \in E\}}. \tag{4.2}$$

This equation computes the underestimation error by comparing the set E of edges in the true graph with the set $\hat{E}$ of edges in the estimated graph. The numerator of the equation

counts all the edge (v, w) present in the true graph E but not in the estimated graph $\hat{E}$. The denominator counts the number of edges present in the true graph E. Thus, the fraction in (4.2) gives the proportion of edges that are present in the true graph but not correctly estimated in the estimated graph.

The overestimation error is defined as

$$oe(G, \hat{G}) = \frac{\sum_{(v,w)} \mathbb{1}\{(v, w) \notin E \text{ and} (v, w) \in \hat{E}\}}{\sum_{(v,w)} \mathbb{1}\{(v, w) \notin E\}}. \tag{4.3}$$

The numerator sums the edges in $\hat{E}$ but not in E. The denominator sums over pairs (v, w) where edges are absent in E. By computing the fraction of edges present in $\hat{E}$ but absent in E, the equation provides insights into the extent of overestimation. It offers an understanding of how the estimation process has overpredicted the graph's edge structure.

Finally, the total error provides a quantitative measure of the overall accuracy of the estimated graph and is calculated by weighting the underestimation error with the number of edges missing in G and the overestimation error with the number of edges in G and. It is given by

$$te(G, \hat{G}) = \frac{ue \sum_{(v,w)} \mathbb{1}\{(v, w) \notin E\} + oe \sum_{(v,w)} \mathbb{1}\{(v, w) \in E\}}{|V|(|V| - 1)/2}. \tag{4.4}$$

Subsequently, we calculated the error metrics for each combination of algorithm, sample size, and penalizing constant using the ten replicates and then averaged these outcomes. The synthesized results are visually depicted in Figure 4.11. The initial row in the figure illustrates the mean underestimation error (ue), overestimation error (oe), and total error (te) values attained from the Exact algorithm's performance.

As anticipated, this row echoes the behavior evident in the individual sample scenario, as illustrated in Figure 4.4. Specifically, larger values of c predominantly lead to underestimation, especially notable for smaller sample sizes, whereas smaller values of c induce significant overestimation errors. These errors diminish as the sample size is bigger. Overall, the most accurate estimations are achieved with larger sample sizes and penalizing constants in the interval $(0.5, 2.0)$.

The results from the Forward Stepwise notably resemble those depicted in Figure 4.10 for the individual sample case. In general, the Forward Stepwise algorithm's performance aligns with that of the Exact algorithm. The third row in Figure 4.11 presents the outcomes from the Backward Stepwise algorithm. These results also resemble those displayed in Figure 4.7. The Backward Stepwise algorithm generally mirrors the behavior of the Exact algorithm. Lastly, the fourth row of Figure 4.11 illustrates the outcomes of the Simulated Annealing algorithm. As demonstrated in Figure 4.5 and discussed above, the algorithm's behavior, as considered in this thesis, presents an unpredictable behavior even in instances with larger sample sizes.

In conclusion, the investigation presented in this section thoroughly explores the algo-

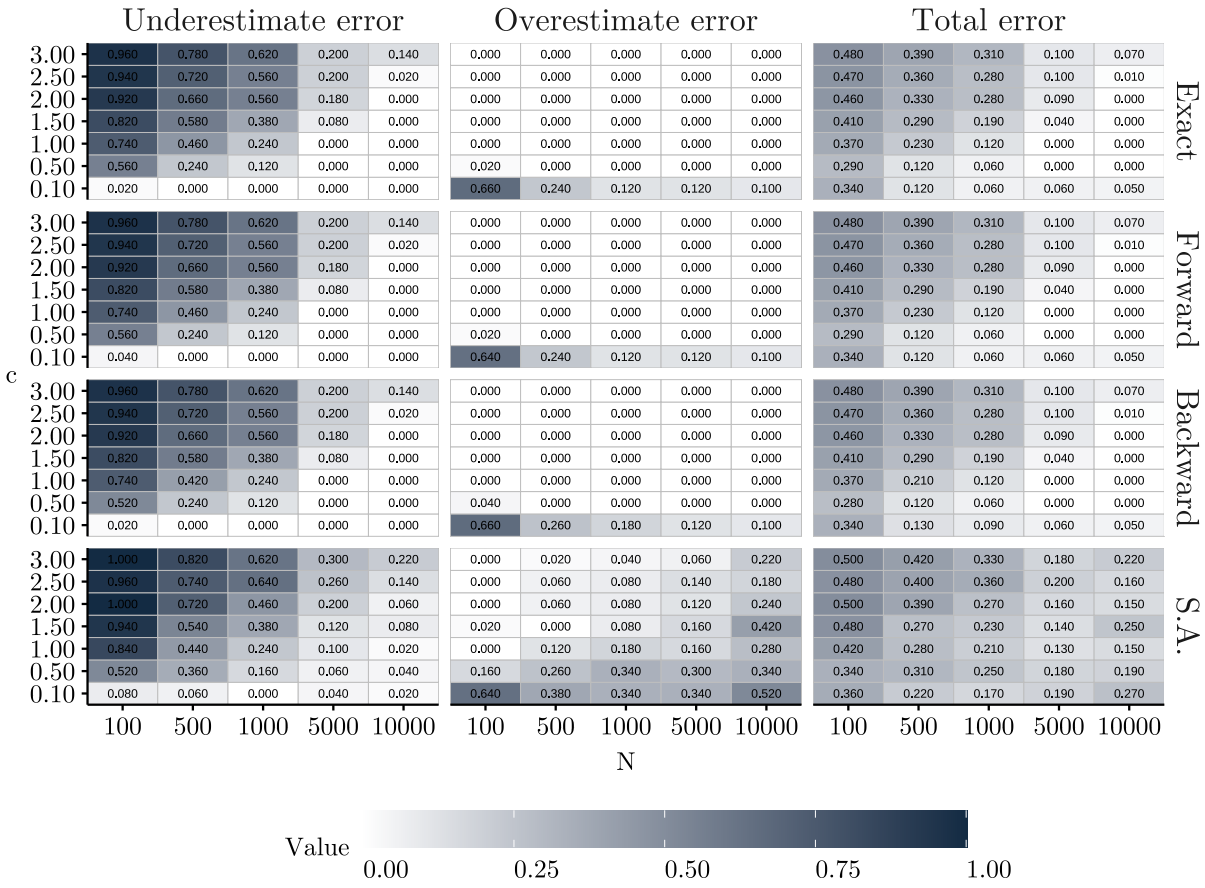


Figure 4.11: *Mean error metrics (underestimation, overestimation, and total error) for the algorithms considering ten samples of each size. Sizes considered are* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant value* $c \in \{0.1, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. *The color in the tile's background indicates the size of the error metric; the darker, the bigger. In general, small penalizing constant values tend to cause overestimation. Conversely, bigger c values tend to cause underestimation, especially when the sample size is small.*

rithms' performance across diverse sample sizes. As mentioned before, we used the Exact algorithm as a comparison basis since it is not feasible even for graphs with a small number of vertices. For this particular example with five edges, it was possible to compare the results with the Backward and Forward Stepwise, and the Simulated Annealing. This detailed evaluation enables us to draw valuable insights into the algorithms' efficacy across a range of scenarios, offering a robust foundation for understanding their performance and guiding practical applications. The Stepwise algorithms have shown an appealing alternative to the Exact algorithm. Meanwhile, the Simulated Annealing did not yield good results. Thus, we shall consider either the Forward Stepwise or the Backward Stepwise algorithms for practical applications.

4.1.4 The Choice of the Penalizing Constant c

In the sections above, we examined the effect of various penalization values over the penalized pseudo-likelihood function (refer to Expression 2.21), always assessing the results

against the true graph, which is known, in each considered scenario. However, when working with real data, the underlying graph is unknown. In order to use any algorithm that incorporates a penalizing term, a range of possible values should be considered. The challenging task then becomes selecting the appropriate value for penalization.

Multiple methods can be used to select the penalizing constant value that yields the best result. Here, we consider the k-fold cross-validation approach – a widely used technique in machine learning and statistical modeling to evaluate a model's performance and generalization capability. In summary, in this approach, the available dataset is randomly divided into k subsets, or *folds*, of roughly equal size. The model is then estimated and evaluated k times, each time using a different fold as the validation set and the remaining $k-1$ folds as the training set. This process helps mitigate the risk of overfitting and provides a more reliable estimate of the model's performance by utilizing different portions of the data for training and validation. The final evaluation metrics are typically averaged over all k runs, yielding a more robust assessment of the model's effectiveness across different subsets of the data. K-fold cross-validation is particularly beneficial when the dataset is limited or when the goal is to better understand how the model performs on various portions of the data. More details can be found in James *et al.* [2021] and Hastie *et al.* [2009].

Generally, the metric used for evaluation is the mean squared error (for regression problems) and the accuracy (for classification problems). However, in our case, we will assess the pseudo log-likelihood of the model over each of the k-folds.

Let P_i, $i = 1, \ldots, k$ denote a partition of the observed data set $\mathbf{X} = \{X^{(1)}, \ldots, X^{(N)}\}$ and let $P_{-i} = \bigcup_{j \neq i} P_j$. Each P_i is a fold for the k-fold cross-validation method. For each fixed value of c and for each i, we estimate the underlying graph using the estimator in (2.21) considering the set P_{-i} as training data. After this procedure, we have an estimate $\hat{G}$ of the true graph and corresponding probabilities for $\pi(a_v | a_{\widehat{G}(v)})$. These estimates are used to compute the pseudo-log-likelihood over the validation set, i.e., P_i, as

$$\mathrm{CV}_k^{(i)}(c) = \sum_{v \in V} \sum_{(a_v \in A)} \sum_{a_{\hat{G}(v)} \in A^{|\hat{G}(v)|}} N_i(a_v, a_{\widehat{G}(v)}) \log \widehat{\pi}(a_v | a_{\widehat{G}(v)}),$$

where $N_i(a_v, a_{\hat{G}(v)})$ is computed over the validation set P_i and c is the fixed penalizing term.

Then, the cross-validation metric is averaged over the results from all folders,

$$\mathrm{CV}_k(c) = \frac{1}{k} \sum_{i=1}^{k} \mathrm{CV}_k^{(i)}(c). \tag{4.5}$$

This procedure is repeated for a set of different values of constant c, and the best c is then

$$c_k^* = \max_c \mathrm{CV}_k(c).$$

Once this value is chosen, the final graph is estimated using the entire sample data with

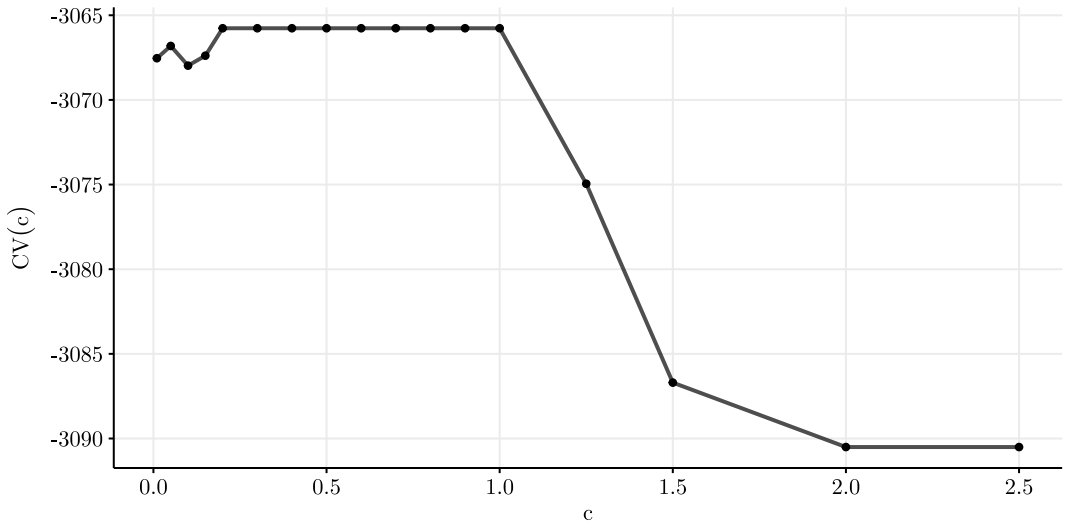


Figure 4.12: 5-*fold cross-validation error for Example 11, considering a sample of size* 5,000 *and set of penalizing constant* $\{0.01, 0.05, 0.10, 0.15, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00, 1.25, 1.50, 2.00, 2.50\}$.

penalizing constant c_k^*.

For Example 11, we performed the 5-fold cross-validation approach considering a sample of size 5,000 and a set of penalizing values $\{0.01, 0.05, 0.10, 0.15, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00, 1.25, 1.50, 2.00, 2.50\}$. Figure 4.12 shows the cross-validation metric (4.5) as a function of the penalizing constant c. The penalizing values considered were chosen in an iterative process, manually refining the search grid when necessary, for example, when $c \in [0.2, 1.0]$. Note that the highest value of $\text{CV}_5(c)$ is attained for $c_5^* \in \{0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00\}$. The next step is to estimate the graph considering the complete data set and the values of c_5^*. Considering the Forward Stepwise algorithm, the graph was correctly estimated in all possible values for c_5^*; revisit Figure 4.1.

An issue that can arise in validation methods, such as cross-validation, with real data occurs when a configuration a_v observed in the validation set is absent in the training set. This mismatch leads to $\hat{\pi}(a_v | a_{\hat{G}(v)}) = 0$, resulting in $\text{CV}_k^{(i)}(c) = -\infty$. To prevent such cases, we introduce a hyperparameter γ and redefine

$$\hat{\pi}(a_v | a_{\hat{G}(v)}) = \gamma, \tag{4.6}$$

for cases where this probability is zero. Then, the distribution $\pi(\cdot | a_{\hat{G}(v)})$ should be rescaled to accommodate this change. We recommend the hyperparameter γ to be set to a small value to prevent numerical instability. Following this, the distribution $\hat{\pi}(\cdot | a_{\hat{G}(v)})$ is rescaled for configurations not observed in the training data. This adjustment ensures stability in the cross-validation method for real data sets. Examples of this issue happening in real data sets are shown and examined in Section 5.1 and Section 5.2.

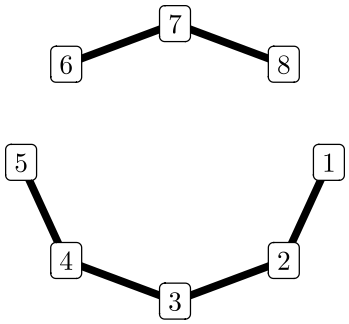


Figure 4.13: *Graph of Example 12, with vertices $X_1, \ldots, X_8$. For simplicity, instead of X_i, the figure shows only the node index, i.e., i.*

4.1.5 Additional Scenario

We also consider another illustrative example, presented below.

Example 12. *Let us consider a scenario involving eight discrete random variables, each taking values in $A = \{0, 1, 2\}$ with a joint probability function given by*

$$p(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = p(x_1|x_2)p(x_2|x_3)p(x_3|x_4)p(x_4)\times \\ p(x_5|x_4)p(x_6|x_7)p(x_8|x_7)p(x_7). \tag{4.7}$$

The graphical representation of this example is depicted in Figure 4.13. Notably, the graph structure exhibits a certain linearity in the arrangement of te nodes. Again, it was intentionally designed to resemble the graph structure expected in the context of streamflow data from the São Francisco river; see details in Section 5.1.

For the sake of conciseness and to prevent redundancy in this chapter, we offer a concise overview of the outcomes from Example 12 in Appendix B, briefly discussing the data generation process and relevant particular aspects of each algorithm. This approach helps us avoid unnecessary repetition and the inclusion of similar figures within the main text.

In Section B.1, we present the conditional distributions as well as the theoretical values chosen for the probability distributions. For this example, we also evaluate the performance of the algorithms in two settings: first, considering a single sample of size $N \in \{100, 500, 1000, 5{,}000, 10{,}000\}$; second, generating ten different samples of each size N. We then compute and analyze the estimation errors; see Expressions (4.2), (4.3), and (4.4).

For this relatively simple eight-node graph, employing the Exact algorithm becomes impractical due to the vast number of possible graphs in the search space, totaling 268,435,456. As a result, we exclude this algorithm from consideration in this context. Despite the erratic behavior exhibited by the Simulated Annealing algorithm in the results of Example 11, we opt to include it in this scenario. Additionally, we also consider the forward and Backward Stepwise algorithms, as outlined in detail in Section B.1 of Appendix B. In summary, we observe that the forward and Backward Stepwise algorithms demonstrate effective performance when the sample size is not *small* and the penalizing constant value is appropriately chosen.

As in Example 11, here results of the Simulated Annealing algorithm were not satisfying.

4.2 Random Graphs with Varying Edge Number

We use the Example 13 presented below to study the performance of the proposed algorithms in arbitrarily chosen simulated scenarios.

Example 13. *Let the number of nodes in the graph be fixed at* $d = 5$*. Thus, this graph with five nodes might have at most ten edges, which would be the complete graph. The discrete random variables in the nodes of the graph are binary, taking value in* $A = \{0, 1\}$*. In this example, we fix the number of edges* n_{edges} *in the graph and generate ten random graphs with* d *nodes and exactly* n_{edges}*. This procedure is executed for* $1 \leq n_{\text{edges}} \leq 10$*.*

In this example, we iterate over the number of edges in a graph with a fixed number of nodes. For each configuration of a fixed number of edges, we generate ten random graphs, each having the same number of edges. Figure 4.14 displays the assortment of randomly generated graphs used in this example. These graphs present varying numbers of edges within the scope of graphs comprising five vertices (ranging from 1 to 10 edges). We show the ten different graphs considered for each possible number of edges. The node arrangement in the figure mirrors that of Figure 4.1.

4.2.1 Data Generation

We illustrate the data generation procedure used for each replicate in Example 13 through a single specific case. Taking a graph comprising two edges, let us focus on the replication number 1. This specific graph is visualized in the final row of the second column within Figure 4.14. From now on, within the scope of this section, we will refer to this specific graph simply as G_1. In this graph, the edges link nodes 2 and 5, as well as nodes 3 and 4. With the edges of G_1 defined, the subsequent step involves determining the neighborhood of each node, which serves as the basis for generating random values for each node's marginal and conditional distributions. The resulting probability distributions for this illustrative scenario are presented in Tables 4.6, 4.7, 4.8, 4.9, and 4.10. Using the Gibbs sampler algorithm (similar to the one introduced in Section 4.1.1), we generate sample data from the multivariate stochastic process with a dependence structure defined by the graph G_1. This procedure considers a burn-in period of 5,000 iterations and results in a sample of size 5,000.

The procedure described above to generate sampled data from graph G_1 was replicated across each of the 100 graph scenarios depicted in Figure 4.14. Details regarding the implementation of this data-generating process can be found in Section C.2 of Appendix C.

Figure 4.14: *Randomly generated graphs with varying numbers of edges, considering a graph with five nodes. For each number of edges (from 1 to 10, displayed in columns), ten random graphs were generated (represented in rows). The node arrangement in the figure corresponds to that of Figure 4.1.*

x_1	0	1
$p(x_1)$	0.4934	0.5066

Table 4.6: *Marginal distribution of X_1 in graph G_1 of Example 13.*

x_2	0	1
$p(x_2\|x_5 = 0)$	0.2649	0.2649
$p(x_2\|x_5 = 1)$	0.8535	0.1465

Table 4.7: *Conditional distribution of $X_2|X_5$ in graph G_1 of Example 13.*

x_3	0	1
$p(x_3\|x_4 = 0)$	0.4185	0.4185
$p(x_3\|x_4 = 1)$	0.2216	0.7784

Table 4.8: *Conditional distribution of $X_3|X_4$ in graph G_1 of Example 13.*

x_4	0	1
$p(x_4\|x_3 = 0)$	0.4708	0.5292
$p(x_4\|x_3 = 1)$	0.0676	0.9324

Table 4.9: *Conditional distribution of $X_4|X_3$ in graph G_1 of Example 13.*

x_5	0	1
$p(x_5\|x_2 = 0)$	0.3612	0.6388
$p(x_5\|x_2 = 1)$	0.4292	0.5708

Table 4.10: *Conditional distribution of $X_5|X_2$ in graph G_1 of Example 13.*

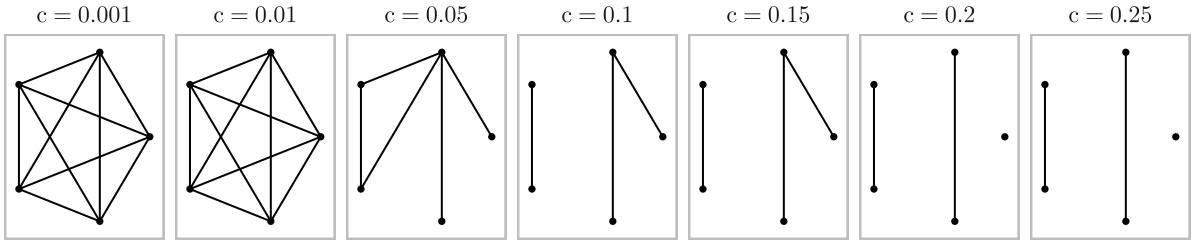


Figure 4.15: *Outcomes of the Forward Stepwise algorithm for graph G_1 considering a range of penalization constant $c \in \{0.001, 0.010, 0.050, 0.100, 0.150, 0.200, 0.250\}$.*

4.2.2 Estimation

Considering the superior outcomes of the Stepwise algorithms in Examples 11 and 12 over the Simulated Annealing algorithm, we opted to exclusively use this algorithm for estimation within this example, considering both forward and Backward approaches. Then, for each sampled data set, we executed the algorithm across a range of penalization constants, specifically $c \in \{0.001, 0.010, 0.050, 0.100, 0.150, 0.200, 0.250\}$.

To illustrate the impact of the penalizing constant on the estimation process, Figure 4.15 showcases the estimated graphs generated by the Forward Stepwise algorithm for all c values considered in this case study. As expected, lower values of penalizing constant result in higher overestimation errors. In particular, when $c \in \{0.001, 0.010\}$, the estimated graph is the complete graph, leading to an overestimation error of 1. As c increases, the number of overestimated edges decreases for $c \in \{0.005, 0.100, 0, 150\}$, and finally, the algorithm correctly recovers the true structure of graph G_1 for $c \in \{0.200, 0.250\}$.

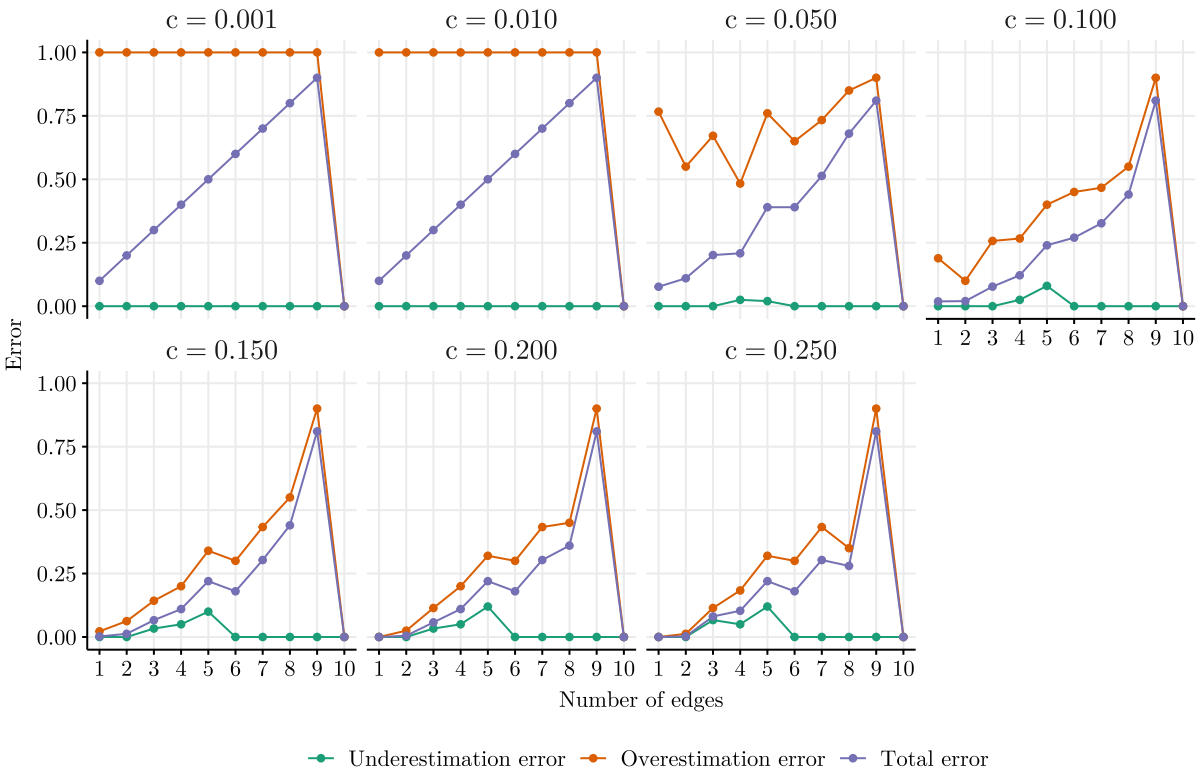


Figure 4.16: *Average error metrics outcomes of the Forward Stepwise algorithm. Each sub-figure represents the result of a specific c value and displays the averaged errors as a function of the number of edges in the graph.*

Again, for each possible number of edges in the graph, ranging from 1 to 10, ten distinct random graphs and, consequently, ten samples were generated. Error metrics were computed for each of these samples and then averaged. An overall summary of the results is depicted in Figure 4.16, where each sub-figure corresponds to a specific value of c and shows the average error metrics as a function of the number of edges in the graph. Regardless of the penalizing constant, the average of all errors vanishes when the number of edges in the graph is 10 (the complete graph in this case). In particular, by construction, the overestimation error is zero. On the other hand, the underestimation error in the Forward Stepwise algorithm is low for all values of c, as shown in all sub-figures of Figure 4.16.

Furthermore, we can break down the analysis of the results in Figure 4.16 into two distinct parts. The first one encompasses outcomes for small values of the penalizing constant c, specifically $\Lambda_1 = \{0.001, 0.010\}$. Here, the mean overestimation error is consistently high for all numbers of edges (except in the complete graph case), and the underestimation error is zero. This behavior is expected, as small values of the penalizing constant are known to lead to overestimation.

In contrast, the second group corresponds to larger values of c, denoted as $\Lambda_2 = \{0.050, 0.100, 0.150, 0.200, 0.250\}$, and exhibits a different trend. The mean overestimation error is relatively smaller for $c \in \Lambda_2$. However, it increases as the number of edges in the graphs grows. In general, the underestimation error remains low for all values of c. However, for

$c \in \Lambda_2$, some instances exhibit underestimation error, particularly when the number of edges in the graphs falls between 2 and 5.

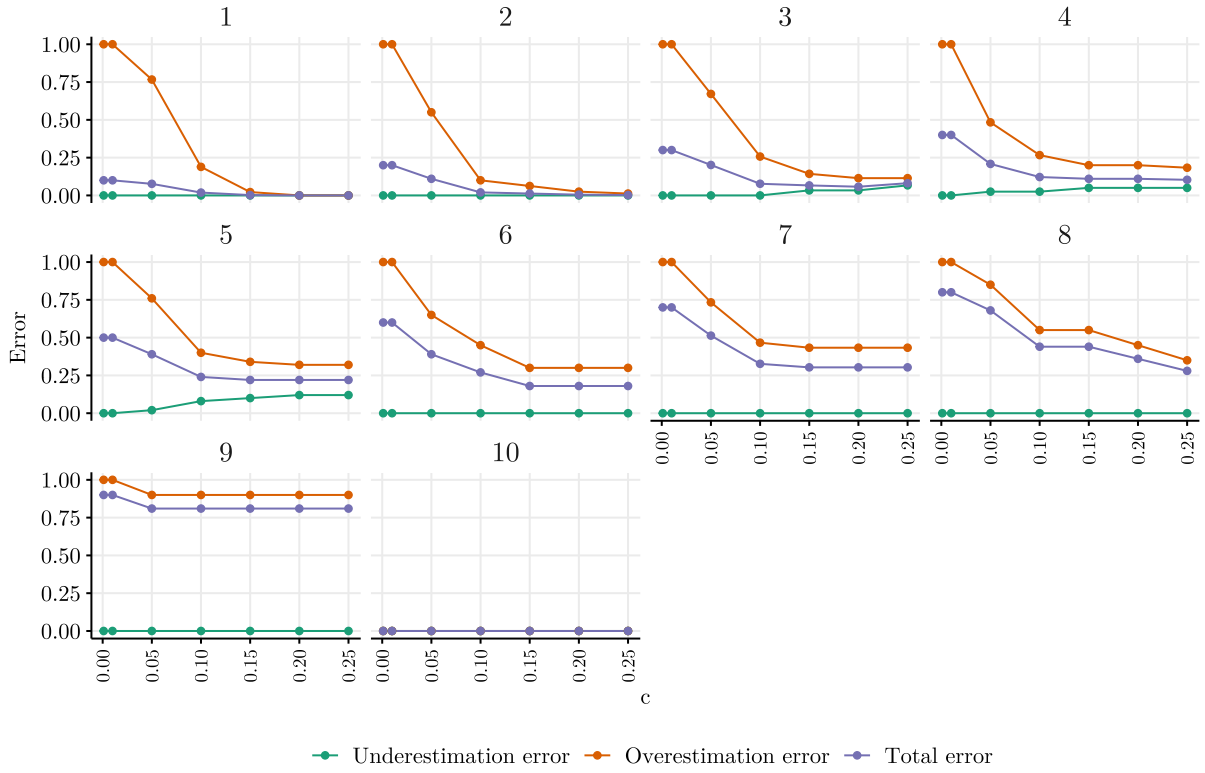


Figure 4.17: *Effect of varying the penalizing constant c on the average error metrics outcomes from the Forward Stepwise algorithm. Each subfigure is the result given by graphs with a fixed number of edges, from 1 to 10.*

Additionally, Figure 4.17 shows the result from a different perspective: the impact of different penalizing constant c values on the average error metrics from the Forward Stepwise algorithm results. Each subfigure showcases the result while maintaining constant the number of edges in the graphs, ranging from 1 to 10. As expected, the error metrics decrease as the penalizing constant value increases. The exception is the case where the number of edges is 10, and the error metrics are somewhat constant. This is due to the definition of the overestimation error and the total error. When the number of edges is 10, the complete graph, all the errors vanish regardless of the penalizing constant value. More specifically, in this case, the overestimation error is always zero.

We also considered the Backward Stepwise algorithm for estimation in this scenario. In terms of estimation, the results were similar to those of the Forward Stepwise algorithm. For this reason, we leave the corresponding figures and additional comments to Section B.2 of Appendix B.

4.3 Comparative Analysis

In this chapter, we presented an extensive study aimed at validating the proposed algorithms for estimating the underlying graph of a multivariate stochastic process that satisfies a mixing condition. The entire suite of R functions utilized in this simulation study can be accessed in the author's repository at github.com/magnotairone/phd_codes.

As shown, the Exact algorithm was used solely as a means of comparison among the other methods, as it is unfeasible in computational terms even for graphs with a small number of nodes (see Example 12).

The Simulated Annealing algorithm was the first alternative to the Exact algorithm that we took into consideration. However, despite our efforts performing hyperparameter tuning, this method did not yield satisfying results. This was not expected, as this method was used in Leonardi *et al.* [2023] to estimate the neighborhood of each node separately. In this work, the underlying graph is estimated by combining these neighborhoods. However, the Simulated Annealing was not a good approach when used to estimate the entire graph, as proposed in this thesis.

Therefore, we considered the Stepwise algorithms. Both the forward and Backward approaches presented satisfying results, comparable to the Exact algorithm as shown in the scenarios of Example 11. Since the forward and Backward had a good performance in all scenarios considered, we suggest using the Forward Stepwise in cases where it is previously known that the underlying graph has few edges. On the other hand, the Backward Stepwise should be used in cases where the number of edges in the underlying graph is known to be high.

Chapter 5

Applications

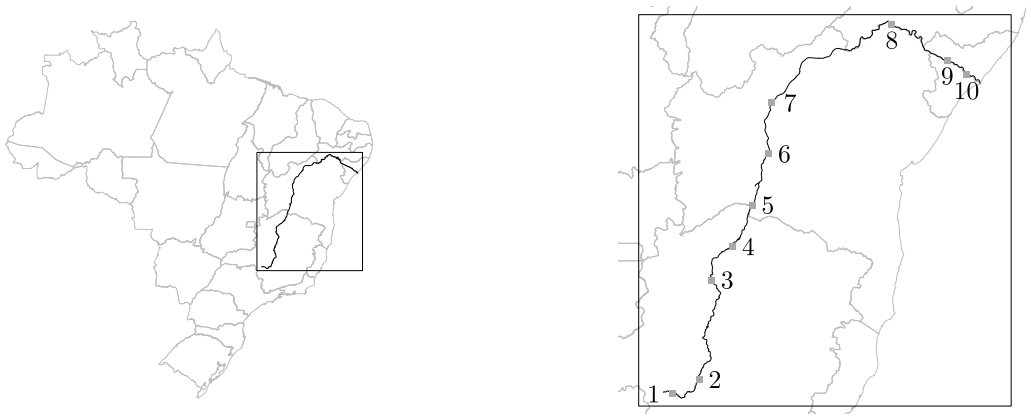
This chapter presents two distinct applications. It is important to emphasize that our intention here is not to introduce novel modeling strategies. Instead, we aim to demonstrate the practical utilization of the proposed method across different contexts.

In Section 5.1, we present the stream flow data recorded along the São Francisco river in Brazil over time. Our goal is to study the underlying dependency structure among the measurements taken across the gauges positioned throughout the river's course. Moving forward to Section 5.2, we shift our focus to stock market indices. Here, we investigate the dependency relationship concerning the fluctuations of these indices in different markets, specifically analyzing their daily rises and falls, and the aim is to understand the dependency relationship across global markets.

5.1 São Francisco River Data

The measurement of water volume flowing within a river's course relies on streamflow stations strategically positioned along its length. Each station denoted as X_u, where $u = 1, \ldots, d$, captures the flow characteristics at specific points. These individual measurements collectively constitute the random vector $\mathbf{X} = (X_1, \ldots, X_d)$ containing the records from the d stations. This random vector is also observed at distinct discrete time intervals, such as daily, weekly, or monthly instances. We represent the vector observed on the i-th day as $\mathbf{X}^{(i)} = (X_1^{(i)}, \ldots, X_d^{(i)})$. Then, we can consider the process $\mathbf{X}^n = \{\mathbf{X}^{(i)} : 1 \leq i \leq n\}$, where $\mathbf{X}^{(i)} \in \mathbb{R}^d$, and employ techniques for modeling multidimensional stochastic processes.

In scenarios such as river water flow, the dynamic nature of the system can lead to independence in behavior at specific points along the river's course. For example, constructing a hydroelectric dam or diverting a watershed can introduce independence between observations made before and after these interventions. To illustrate, consider the scenario where $(X_1, \ldots, X_u)$ is independent of $(X_{u+1}, \ldots, X_d)$. This independence allows us to model the records from the initial u stations separately from the remaining stations. Consequently, we reduce the dimensionality of the problem, enabling the exploration of independent processes



(a) **(b)**

Figure 5.1: *(a) Geographical boundaries of Brazil and its state limits. The highlighted rectangle indicates the region where the São Francisco River is situated; (b) An enlarged view of the boxed area in (a), encompassing the São Francisco River. Gray points correspond to the ten streamflow gauges included in our analysis, numbered in ascending order from bottom to top.*

with u and $d - u$ dimensions, respectively.

One direct application of this method lies in monitoring streamflow data in Brazilian rivers. Daily measurements of meteorological and hydrological variables, including river levels, flow rates, precipitation, climatology, and water quality in Brazil, are publicly accessible through the Brazilian National Water Agency (SNIRH [2023]). The analysis's main focus will be the flow rate at various points along the course of São Francisco River (SFR). Spanning a distance of over 2,914 kilometers, the SFR originates in the Canastra mountain range in the central region of Brazil, flowing northwards into the northeastern part of the country. Figure 5.1 (a) visually represents the river's path within Brazil.

The SFR course can be segmented into four sections: the upper section (including stations 1 and 2), extending from its source to Pirapora city; the upper middle section (stations 3, 4, 5, 6, and 7), spanning from Pirapora to Sobradinho dam, the navigable part; the lower middle section, covering the stretch from Sobradinho Dam to Itaparica Dam (station 8); and the lower section, encompassing the path from the Itaparica dam to the river mouth (stations 9 and 10). The locations of these selected stations are depicted in Figure 5.1 (b). The river's flow at various points can also be influenced by the time of year. The wet season, which accounts for nearly 60% of the annual precipitation, occurs from November to January, while the dry season lasts from June to August.

The data used in this study case were collected at the beginning of 2019 and contain daily measurements dating back to the first record on October 1st, 1924, with the most recent recorded on February 28th, 2019. According to the National Hydrometeorological Network Inventory by Agência Nacional de Águas (ANA), there are 29 gauging stations positioned along the SFR. Due to the considerably large amount of missing data, as each gauge was constructed and started operating at different times, we focused on ten stations with the least amount of missing data. The data wrangling process, performed using the `tidyverse`

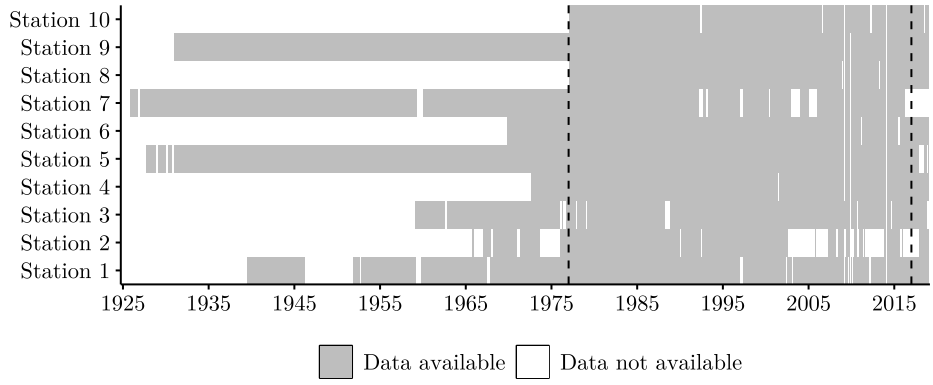


Figure 5.2: *Availability of measurements (gray spots) and data gaps (white spots) for each station in the SFR dataset. The vertical dashed black line delineates the time frame of this study, a period with reduced data gaps, spanning from January 1977 to January 2016.*

R package (Wickham *et al.* [2019]), took a considerable amount of time due to the format in which the data were stored and organized.

Despite this selection, among the stations there is a notable amount of missing data persists, as evident in Figure 5.2. This figure illustrates the days of available measurements (gray tiles) and those with missing data (white tiles) for each station. The vertical dashed black lines indicate a period with fewer gaps in data, spanning from January 1977 to December 2016. This timeframe is the scope of analysis for this study. Instead of working with the daily measurements, we opted to consider 10-day average values to reduce the amount of missing data.

This data was first analyzed by Leonardi *et al.* [2021], which introduces a model selection criterion designed to identify points of independence within a random vector, leading to a decomposition of the vector distribution function into independent blocks. In their analysis, the authors found out that the random vector $\mathbf{X}$ could be decomposed into two subvectors, $(X_1, X_2, X_3, X_4, X_5, X_6, X_7)$, and (X_8, X_9, X_{10}). It is important to mention that Leonardi *et al.* [2021] focused on monthly averaged streamflow data, whereas our current study utilizes a 10-day average approach. An inherent limitation of this approach is that it restricts the dependence of any given random variable X_i to its neighboring variables in the vector in the same block.

Our proposed estimator is designed for discrete random processes. However, the stream flow measurement data comprises continuous random variables. To accommodate this, we introduce the process $\mathbf{Y}^n = \{\mathbf{Y}^{(i)} : 1 \leq i \leq n\}$, where $\mathbf{Y}^{(i)} \in |A|^d$, $A \in \{0, 1, 2, 3, 4\}$. To work with this process effectively, we discretized the data into five levels based on quantiles. Figure 5.3 provides a visual representation of the count of levels observed in the discretized streamflow values derived from the original dataset. The horizontal x-axis displays these individual random variables, while the vertical y-axis represents each level of these random variables. Each tile within the figure contains a numeric value, indicating the count of observations associated with a particular level of the respective random variable. This graphical

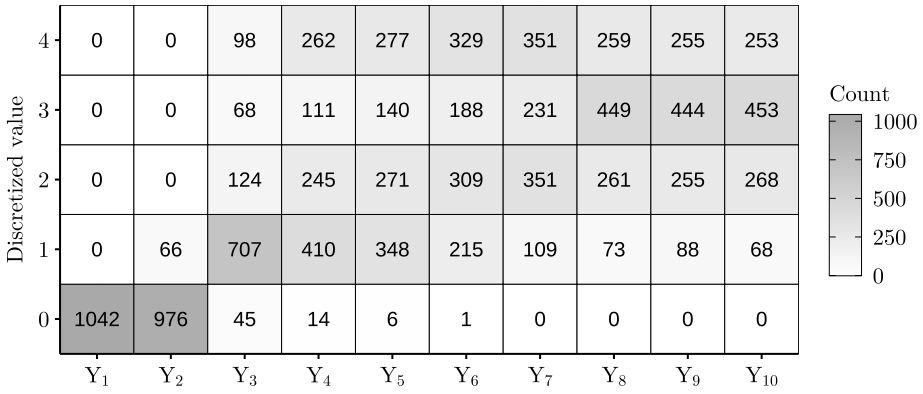


Figure 5.3: *Frequency of discrete levels within random variables representing discretized streamflow values from the original dataset. The x-axis corresponds to the random variables (stream flow gauges), while the y-axis displays the corresponding observation counts for each level. Numeric values inside each tile indicate the observation count for specific levels within the random variables.*

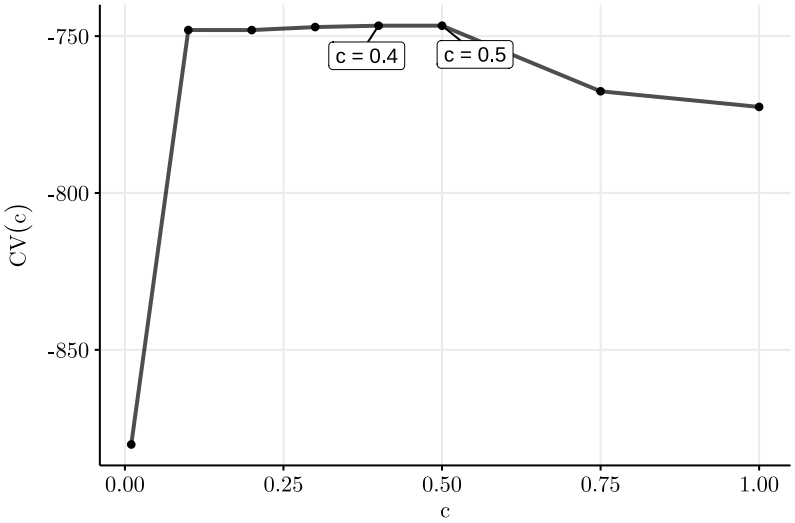


Figure 5.4: *5-fold cross-validation error as function of* $c \in \{0.01, 0.10, 0.20, 0.30, 0.40, 0.50, 0.75, 1.00\}$ *for the SFR data. The text box highlights the penalizing value at which the maximum is attained:* 0.40 *and* 0.50.

representation offers insights into the distribution and variability of the discretized streamflow values across different levels.

Altogether, the dataset encompasses a total of 1042 observations. It is worth noting that variable Y_1, corresponding to station 1, assumes values only within the first level in the alphabet, $Y^{(i)} \in \{0\}$. This singular behavior could challenge estimation and will be discussed ahead. In contrast, the remaining random variables assume values across at least two levels of the alphabet.

We employed our proposed estimator using the forward stepwise algorithm and a 5-fold cross-validation approach, as presented in Section 4.1.4, to determine the optimal value of the penalizing constant c. The range considered for c was the set $\{0.01, 0.10, 0.20, 0.30, 0.40, 0.50, 0.75, 1.00\}$. Figure 5.4 displays the average log pseudo-likelihood computed over the validation set for each value of c. The highest value of $CV_5(c^*)$ was achieved when $c^* = 0.40$

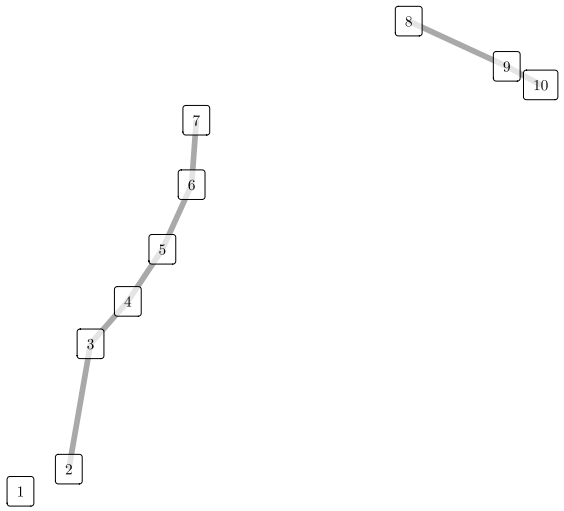


Figure 5.5: *The underlying graph estimated by our proposed model for the SFR, considering* $c^* \in \{0.40, 0.50\}$. *The station's arrangement mirrors the geographical locations shown in Figure 5.1 (b).*

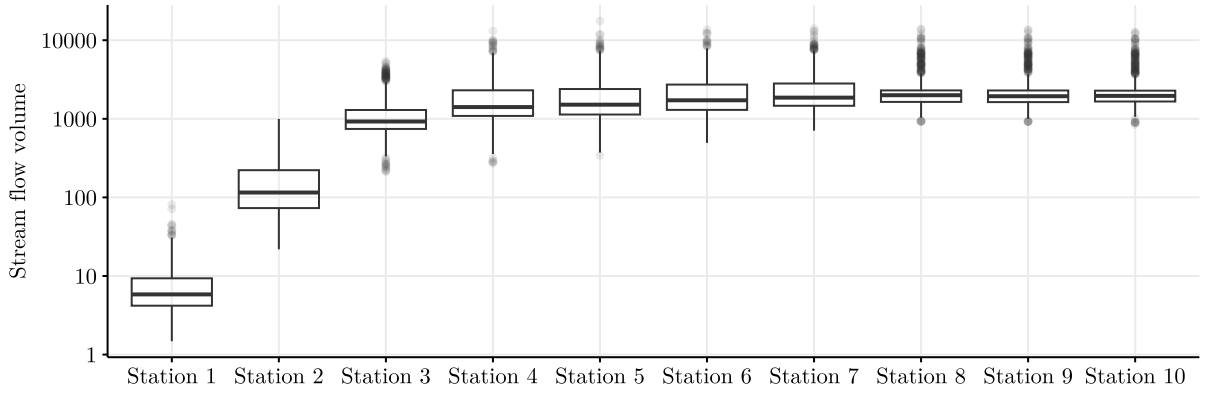


Figure 5.6: *Stream flow volume measured at the ten stations in the SFR (logarithmic scale).*

and $c^* = 0.50$.

It is important to note that during cross-validation for this data, certain configurations a_v observed in the validation set were not present in the training set. As discussed in Section 4.1.4, this situation can cause instability in the cross-validation method, and a hyper-parameter γ is introduced to cope with this problem. Thus, we considered here $\gamma = 0.0001$ (refer to Equation (4.6) for its definition).

Finally, using the complete sample, we estimate the underlying graph with c^* values of 0.4 and 0.5. The resulting estimated graph is visualized in Figure 5.5. By the natural spatial configuration of the river, the sequential structure seems adequate for this problem. It consists of three disconnected sub-graphs with nodes $\{1\}$, $\{2, 3, 4, 5, 6, 7\}$ and $\{8, 9, 10\}$, respectively, with each node being linked with the previous and subsequent stations in each component, as expected for a dataset with a Markovian spatial dependence.

The measurements recorded at Station 1 significantly differ from those at the other stations, which might explain why this node is estimated to be independent of the others.

Moreover, the break in the sequential structure can be explained by the presence of the Sobradinho hydroelectric dam, the largest dam along the course of the SFR, situated between stations 7 and 8. This influence is evident in the boxplots of streamflow volume measurements shown in Figure 5.6, represented on a logarithmic scale. There is a qualitative change in regime in the boxplots after Station 7, likely attributed to the hydroelectric dam's impact on the river flow. This aligns with the independence detected by the algorithm at this location and is consistent with the results obtained by Leonardi *et al.* [2021].

5.2 Stock Exchange Data

A stock, also known as a share or equity, represents ownership in a company. Individuals or institutions become shareholders when they own stocks, holding a portion of the company, as the total ownership is divided into shares. Investors can buy or sell these shares in the stock market.

A stock exchange, commonly known as a stock market, serves as a centralized platform facilitating the trading of various financial instruments, with a primary focus on stocks and securities. This marketplace enables investors, from individuals to institutions, to buy and sell shares in publicly listed companies. Within stock exchanges, the essential infrastructure and regulatory framework are in place to ensure the efficiency and transparency of these financial transactions.

On the other hand, a stock index is a statistical measure that represents the performance of a specific group of stocks or the broader stock market. It is often used to gauge the overall health and direction of the financial markets. Stock indices are constructed using a weighted average of the prices or market capitalizations of the component stocks. Popular examples include the Índice Bovespa (Ibovespa) in Brazil and the Dow Jones Industrial Average (DJIA) in the United States.

The relationship between the performance of stock indices in different countries is of significant importance for several reasons. Firstly, it reflects the interconnectedness of global financial markets. As economies become increasingly globalized, the performance of one country's stock market can have a ripple effect on others. Investors and analysts closely monitor these relationships to assess potential risks and opportunities.

Secondly, it provides valuable insights into broader economic trends. When multiple stock indices from different countries move in a similar direction, it can signal global economic trends, such as periods of growth or recession. Conversely, diverging performances may indicate regional disparities or unique economic factors at play. Lastly, the correlation between stock indices can impact investment strategies. Diversifying investments across countries with low or negative correlations can help reduce portfolio risk.

In this scenario, another challenging problem arises when dealing with multiple time zones due to different opening times of global stock exchanges, as depicted in Figure 5.7.

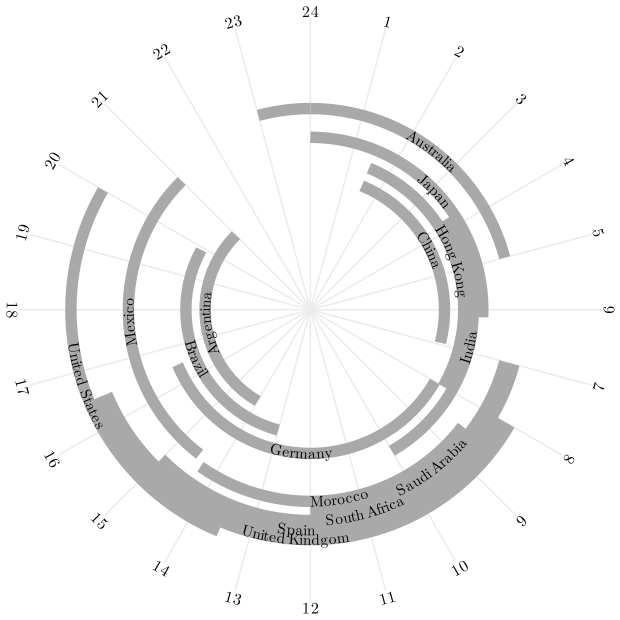


Figure 5.7: *Working hours of the 15 stock exchanges on business days corresponding to the market indices under consideration. The time zone used as a reference is UTC+0.*

Here the analysis focuses on data from 15 stock indices of different countries: S&P Merval (Argentina), S&P ASX 200 (Australia), Ibovespa (Brazil), Shanghai Composite (China), DAX (Germany), FTSE China 50 (Hong Kong), Nifty 50 (India), Nikkei 225 (Japan), S&P BMV IPC (Mexico), Moroccan All Shares (Morocco), Tadawul All Share (Saudi Arabia), South Africa Top 40 (South Africa), IBEX 35 (Spain), FTSE 100 (United Kingdom), Dow Jones Industrial Average (United States). The analysis covers the period from May 18th, 2010, to September 20th, 2023, with daily measurements taken from Investing.com [2023], which provides free access to this data.

Several data processing steps were implemented to address missing data, particularly on holidays and weekends in certain countries. In all the countries under consideration, stock exchanges operate from Monday to Friday on business days. However, the stock exchange in Saudi Arabia follows a different schedule, operating from Sunday to Thursday. A workaround was applied to maintain consistency and avoid the need to exclude all Friday data from the dataset. Specifically, the data observed on each Thursday was replicated for Fridays in all records pertaining to the Tadawul All Share index. A similar approach was applied to account for holidays, ensuring data continuity. After implementing these adjustments, the final dataset comprises 2,654 rows.

Figure 5.8 displays the daily variations in the stock indices grouped by their respective continents. Analyzing these variations for correlations is a complex task. As an alternative to address this, we utilize our proposed graph estimator model to represent conditional dependencies among these random variables. However, a preliminary step involves transforming the continuous random variables into discrete ones. We achieve this by considering the daily

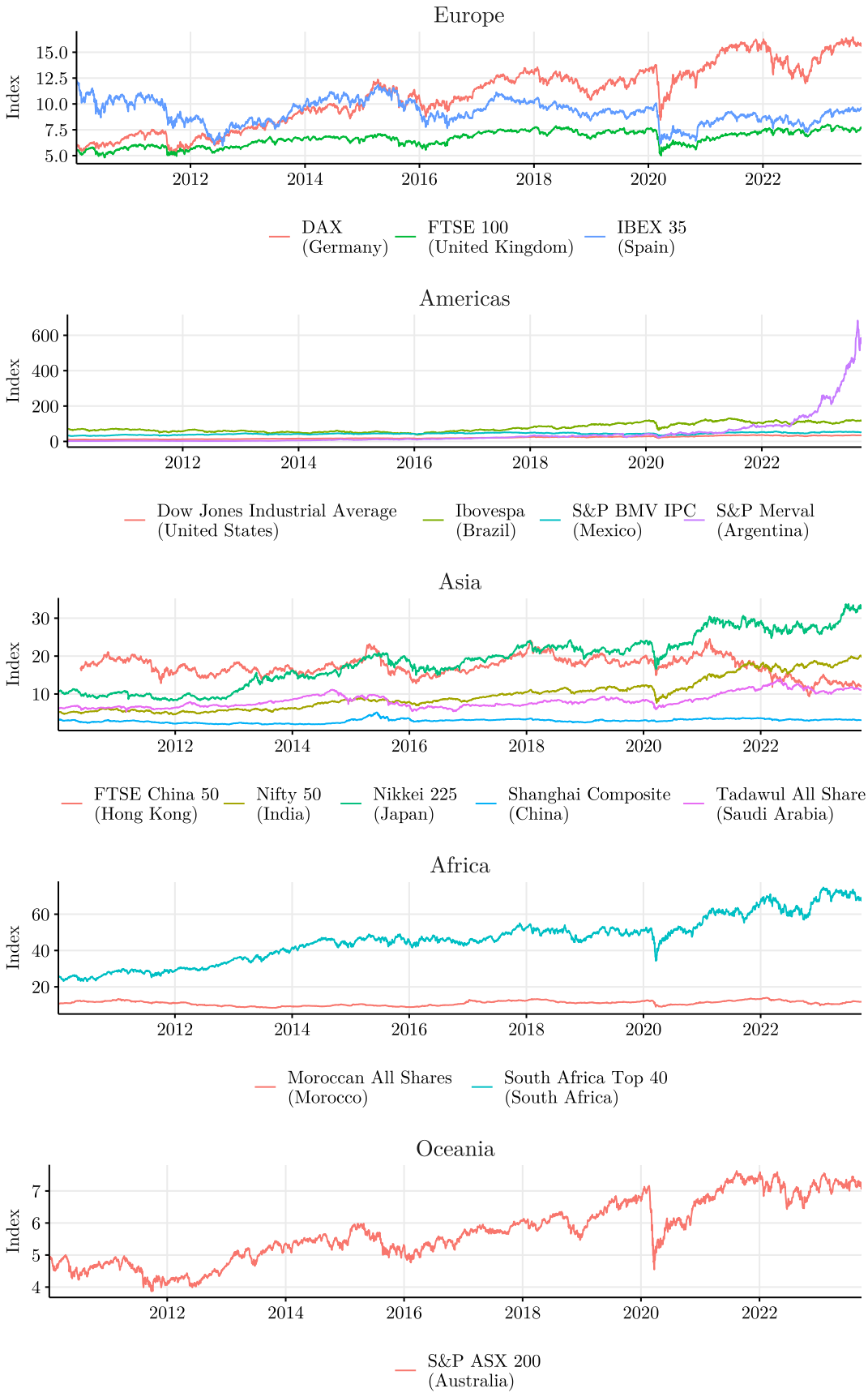


Figure 5.8: *Stock indices from May 18th 2010 to September 20th 2023. Each subfigure shows the indices grouped by continent.*

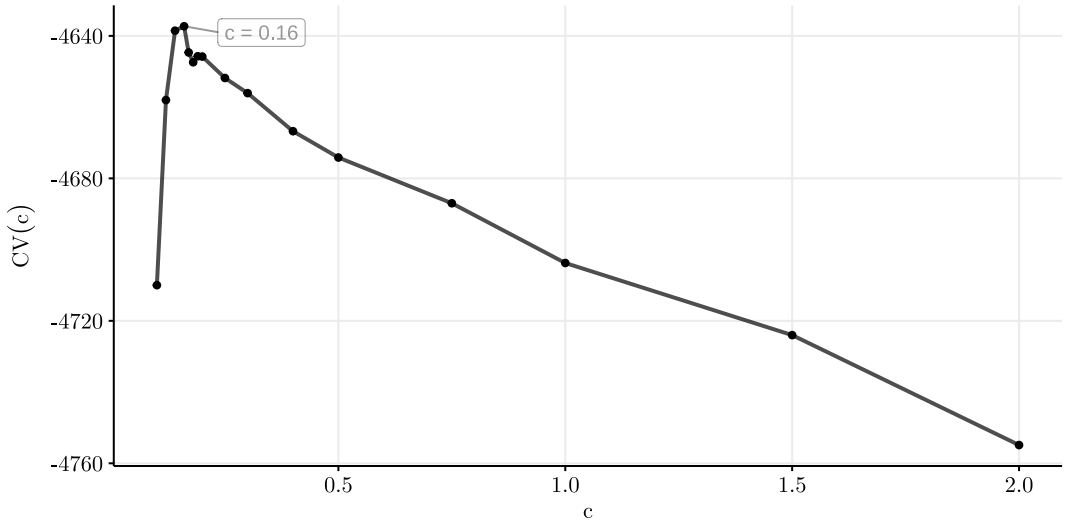


Figure 5.9: *5-fold cross-validation error as function of* $c \in \{0.01, 0.05, 0.10, 0.12, 0.14, 0.16, 0.17, 0.18, 0.19, 0.20, 0.25, 0.30, 0.40, 0.50, 0.75, 1.00, 1.50, 2.00\}$ *for the stock indices data. The text box highlights the penalizing value at which the maximum is attained:* $c^* = 0.16$.

indicator of an increase in the index rate. Specifically, consider two dates, D_0 as the reference date and D_{-1} as the previous date. If the index price at D_0 exceeds that of D_{-1}, the random variable takes the value 1; otherwise, it takes 0. Thus, for each stock index, we create a discrete random variable taking value in $A = \{0, 1\}$ with a sample of size 2,653, one observation less than the original data due to the lagged data used in this transformation.

We employed our proposed graph estimator through a 5-fold cross-validation approach, exploring a range of potential values for the penalizing constant c. The considered values for c were iteratively chosen and included $\{0.01, 0.05, 0.10, 0.12, 0.14, 0.16, 0.17, 0.18, 0.19, 0.20, 0.25, 0.30, 0.40, 0.50, 0.75, 1.00, 1.50, 2.00\}$. The results, illustrated in Figure 5.9, showcase the average pseudo-log-likelihood computed over the validation set for each c value. The highest $CV_{10}(c^*)$ value occurred at $c^* = 0.16$.

Again, as in the previous application for the São Francisco river data, we also define the hyperparameter $\gamma = 0.0001$ to avoid possible cross-validation problems, specifically when certain configurations a_v observed in the validation set were not present in the training set. See Equation (4.6) for this hyperparameter definition.

We used this optimal c^* value of 0.16 to estimate the underlying graph based on the complete sample, as visualized in Figure 5.10. This graph represents the relationship between the stock indices fluctuations. In general, the estimated edges are linking stock markets in the same geographical area, except for specific indices that play a global influence, such as that of the United Kingdom and the United States. An interesting finding is the connections with South Africa across all continents: Mexico, United Kingdom, Germany, and Hong Kong, while Morocco was not connected to any other country.

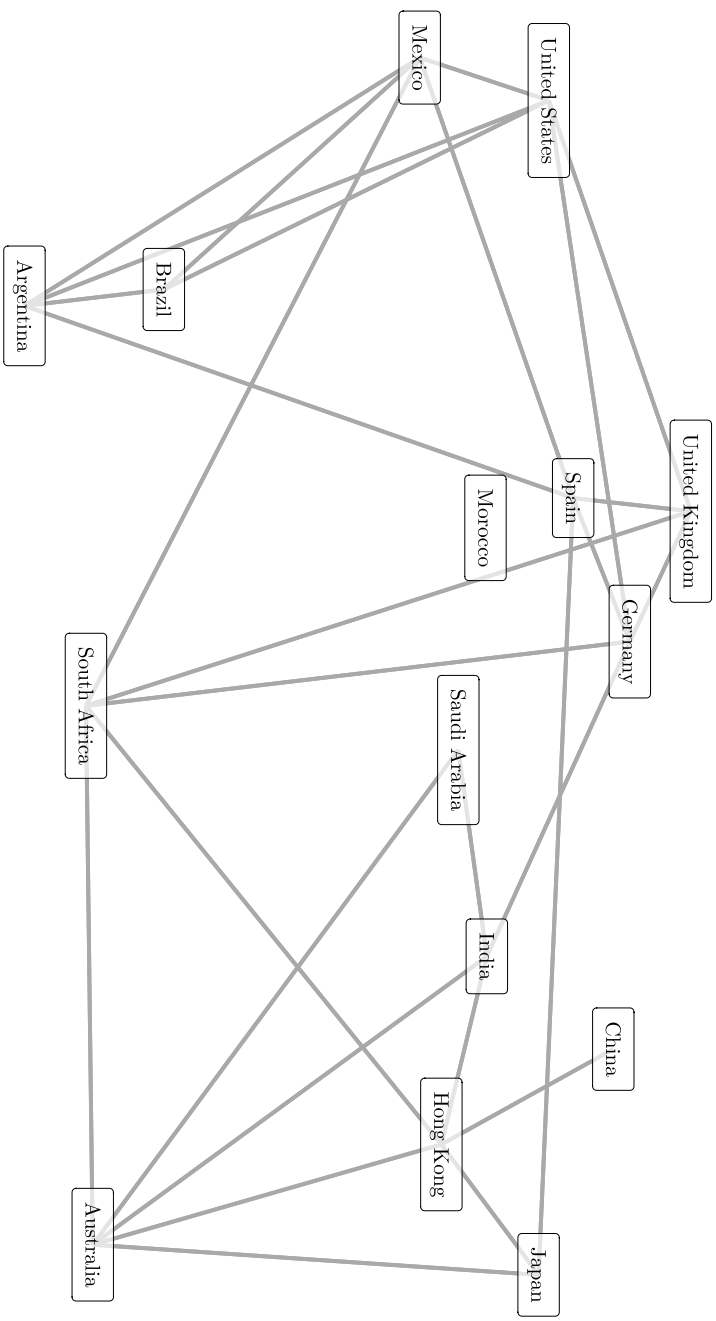


Figure 5.10: *The underlying graph estimated by our proposed model for the stock indices data, considering* $c^* = 0.16$.

Chapter 6

Conclusion and Future Work

The study presented in this thesis was mainly motivated by two key sources: Leonardi *et al.* [2023] and Leonardi *et al.* [2021]. The former introduced a penalized pseudo-likelihood criterion for estimating the graph of conditional dependencies within a partially observed discrete Markov random field. Their approach involved estimating the neighborhood of each node in the graph and subsequently aggregating these neighborhoods to construct the graph itself. The latter proposed a model selection criterion for identifying points of independence within a random vector, resulting in a decomposition of the vector's distribution function into independent blocks. This method, based on a general estimator of the distribution function, is applicable to both discrete and continuous random vectors, as well as independent identically distributed (IID) data and dependent time series data.

This thesis addresses certain limitations of the previously mentioned works. While the estimator introduced by Leonardi *et al.* [2023] is restricted to independent and identically distributed data, the method proposed by Leonardi *et al.* [2021] assumes that the random vector can only be decomposed into subvectors, limiting the dependence structure. In this scenario, a random variable in the vector can only be dependent on the random variables within its block of dependence.

In response to these limitations, we have developed a penalized pseudo-likelihood criterion for estimating the entire graph G, which consists of the set of edges E connecting the nodes V, particularly for multivariate stochastic processes satisfying a mixing condition. The primary advantage of our approach is its ability to handle non-IID. data and its global estimation approach. This means that the entire set of edges E is estimated as a whole, eliminating the need to estimate the neighborhood of each node separately and then combine them to obtain the estimated graph. Another advantage is that our proposed method allows for any node to be dependent on any other node in the graph, eliminating restrictions on the dependence structure.

We proved the consistency of our proposed estimator as well as defined the rate of convergence. This is the main theoretical result of this thesis (refer to Theorem 10). Transitioning to practical implementation, we explored several approaches for implementing the estima-

tor: the exact algorithm, the simulated annealing algorithm, and the forward and backward stepwise algorithms. We also discussed and highlighted the advantages and disadvantages associated with each method.

We conducted a comprehensive simulation study to evaluate the algorithms' effectiveness in estimation. First, we evaluated the estimation of each algorithm in a specific scenario, fixing the graph structure and then generating a random vector with a joint probability function being factorized according to the graph arrangement. The results showed that once the penalizing constant c is fixed, we observed a convergence of the estimated graph as the sample size increases, confirming the consistency of the estimator. We also assessed the impact of the value of c in estimation: small values of c lead to a nonconservative estimation (an estimated graph with many edges), while higher values of c lead to less conservative graphs (with fewer edges linking the nodes).

In the second part of the simulation study, we focused on evaluating the algorithms' performance concerning the number of edges in a graph. The random vector's size remained constant, and we generated multiple random graph structures, each with a different number of edges. Subsequently, random samples were generated based on each created graph, and we applied the proposed algorithms to recover the graph structure.

In summary, while the exact algorithm produced favorable results, it becomes impractical due to its exponential complexity on the number of edges in the graph. Although effective in neighbor estimation according to Leonardi *et al.* [2023], the simulated annealing algorithm did not yield satisfactory results in our considered scenarios. On the other hand, the stepwise algorithms provided a viable alternative, exhibiting satisfactory performance. We recommend utilizing the forward stepwise method when prior knowledge suggests a small number of graph edges. Conversely, the backward stepwise approach is more suitable when dealing with graphs known to have a larger number of edges. Furthermore, it is essential to consider that several factors influence the estimation process, including the number of nodes in the problem, the size of the alphabet, and the underlying graph structure. These factors can significantly affect the computational performance of the algorithms.

Furthermore, we extend our analysis to real-world applications, demonstrating the practical performance of the estimator in scenarios where understanding the conditional dependence structure is of particular interest. The two datasets used in this context comprise observations of continuous random variables. Given that the proposed method is designed for discrete random variables, an appropriate discretization method relevant to each case was necessary.

Moreover, one notable challenge encountered when working with real data is the choice of the penalizing constant c. To address this issue, we designed a k-fold cross-validation approach specifically for the scope of this thesis, which aids in determining the most suitable value for the penalization constant.

Directions for future work

A straightforward extension of this work involves generalizing the approach to accommodate continuous multivariate stochastic processes. This extension would significantly broaden the range of data types that can be effectively analyzed using the estimator, making it even more versatile and widely applicable in fields such as signal processing, finance, and environmental monitoring.

Another line of research for extending this work involves the generalization of the methodology to infinite vertex sets and unbounded estimators, where the size of the estimated graph is permitted to grow proportionally with the sample size. Such an approach could find applications in diverse fields, including but not limited to biology, social sciences, and finance, where complex systems with evolving structures are prevalent.

From another point of view, we can adapt the model to be able to estimate the direction of the edges, i.e., to cope with directed graphs. Extending the model to capture the asymmetry inherent in directed graphs would significantly broaden its applicability. Directed graphs are prevalent in various domains, such as causal inference in epidemiology, information flow in communication networks, and regulatory interactions in biological systems.

Appendix A

Theoretical complementary results

In this Appendix, we present a collection of theoretical results that complement and support the findings discussed in the main body of the thesis. We begin with the definition of the Küllback-Leibler divergence.

Definition 14 (Küllback-Leibler divergence)**.** *The Küllback-Leibler divergence between two probabilities distributions P and Q over A is defined as*

$$D(P;Q) = \sum_{a\in A} P(a)\log\frac{P(a)}{Q(a)}. \tag{A.1}$$

By convention, $P(a)\log\frac{P(a)}{Q(a)} = 0$ if $P(a) = 0$ and $P(a)\log\frac{P(a)}{Q(a)} = +\infty$ if $P(a) > Q(a) = 0$.

The next lemma is presented and proved in Lemma A.7 of Csiszár and Talata [2006].

Lemma 15. *For any P and Q we have*

$$D(P;Q) \le \sum_{a\in A:\, Q(a)>0} \frac{\big[P(a)-Q(a)\big]^2}{Q(a)}.$$

The following definition is adapted from Oodaira and Yoshihara [1971] for the context of this thesis.

Definition 16 (Oodaira and Yoshihara 1971, Section 1)**.** *Let $\mathbf{Y} = \{Y^{(i)}, -\infty < i < \infty\}$ be an univariate process which is strictly stationary and let $X^{(i:j)}$ denote the sequence $Y^{(i)}, Y^{(i+1)}, \ldots, Y^{(j)}$, for $i < j$. We say the process $\mathbf{Y}$ satisfies the uniformly strong mixing condition with rate $\{\psi(\ell)\} \downarrow 0$ as $\ell \to \infty$ if*

$$\Big|\mathbb{P}\big(Y^{(n:(n+k-1))} = y^{(1:k)} \mid Y^{(1:m)} = y^{(1:m)}\big) - \mathbb{P}\big(Y^{(n:(n+k-1))} = y^{(1:k)}\big)\Big| \le \psi(n-m), \tag{A.2}$$

for each $k, m \in \mathbb{N}$ and each $y^{(1:k)} \in \mathbb{R}^k$, $y^{(1:m)} \in \mathbb{R}^m$ with $\mathbb{P}(Y^{(1:m)} = y^{(1:m)}) > 0$.

The following theorem is also adapted from Oodaira and Yoshihara [1971].

Theorem 17 (Oodaira and Yoshihara 1971, Theorem 3). *The strictly stationary univariate process* $\{Y^{(i)}, -\infty < i < \infty\}$, *satisfying the mixing condition* (A.2), *obeys the law of the iterated logarithm if the following requirements are fulfilled:*

1. $E|Y^{(i)}|^{2+\delta} < \infty$ *for some* $\delta > 0$;
2. $\psi(n) = O(1/n^{1+\epsilon})$ *for some* $\epsilon > 1/(1+\delta)$.

Definition 18. *To say that the process* $\{Y^{(i)}, -\infty < i < \infty\}$, *in Theorem 17, obeys the law of iterated logarithm means that, for any* $\epsilon > 0$,

$$P\big(|Z_n| > (1+\epsilon)\chi(n) \ \ i.o.\big) = 0,$$
$$P\big(|Z_n| > (1-\epsilon)\chi(n) \ \ i.o.\big) = 1,$$

where $Z_n = \sum_{i=1}^{n} Y^{(i)}$, $\chi(n) = (2\sigma^2 n \log\log n)^{1/2}$, *and*

$$\sigma^2 = E\big[\big(Y^{(1)}\big)^2\big] + 2\sum_{j=2}^{n} E\big(Y^{(1)}Y^{(j)}\big). \tag{A.3}$$

Lemma 19. *Under the same assumptions of Theorem 17, if* $E(Y^{(j)}) = 0$ *for all* j, *then* $\sigma^2 = E\big[\big(Y^{(1)}\big)^2\big]$.

Proof. For the process $\{Y^{(i)}, -\infty < i < \infty\}$ defined in Theorem 17, we have that

$$\begin{aligned}
\big|E\big(Y^{(1)}Y^{(j)}\big)\big| &= \sum_{Y^{(1)}}\sum_{Y^{(j)}} Y^{(1)}Y^{(j)}\mathbb{P}\big(Y^{(1)} = Y^{(1)}, Y^{(j)} = Y^{(j)}\big) \\
&\le \sum_{Y^{(1)}}\sum_{Y^{(j)}} Y^{(1)}Y^{(j)}\mathbb{P}\big(Y^{(j)} = Y^{(j)} | Y^{(1)} = Y^{(1)}\big)\mathbb{P}\big(Y^{(1)} = Y^{(1)}\big) \\
&\le \sum_{Y^{(1)}}\sum_{Y^{(j)}} Y^{(1)}Y^{(j)}\big[\mathbb{P}\big(Y^{(j)} = Y^{(j)}\big) + \psi(j)\big]\mathbb{P}\big(Y^{(1)} = Y^{(1)}\big) \\
&= \Big[\sum_{Y^{(1)}} Y^{(1)}\mathbb{P}\big(Y^{(1)} = Y^{(1)}\big)\Big]\Big[\sum_{Y^{(j)}} Y^{(j)}\mathbb{P}\big(Y^{(j)} = Y^{(j)}\big)\Big] \\
&\qquad + \psi(j)\sum_{Y^{(j)}} Y^{(j)}\sum_{Y^{(1)}} Y^{(1)}\mathbb{P}\big(Y^{(1)} = Y^{(1)}\big) \\
&= E\big(Y^{(1)}\big)E\big(Y^{(j)}\big) + \psi(j)\sum_{Y^{(j)}} Y^{(j)}E\big(Y^{(1)}\big) \\
&= 0.
\end{aligned}$$

Since $E\big(Y^{(j)}\big) = 0$ for all j, by (A.3), $\sigma^2 = E\big[\big(Y^{(1)}\big)^2\big]$. □

The lemma below is presented and proved in Leonardi *et al.* [2023].

Lemma 20. *If a neighborhood* W *of* $v \in V$ *satisfies*

$$\pi(a_v|a_W) = \pi(a_v|a_{\mathrm{ne}(v)})$$

for all $a_v \in A$*, and all* $a_{W \cup \mathrm{ne}(v)} \in A^{W \cup \mathrm{ne}(v)}$ *with* $p(a_{W \cup \mathrm{ne}(v)}) > 0$ *then* W *is a Markov neighborhood.*

Appendix B

Simulations' complementary results

We present the corresponding results in this Appendix to avoid redundancy with figures from other scenarios examined in the main text. We begin with the outcomes of the considered algorithms for Example (12) in Section B.1. Subsequently, Section B.2 provides supplementary results regarding Example (13), specifically focusing on the Backward Stepwise algorithm.

B.1 Supplementary Information for Example 12

In this section, we provide additional details concerning Example 12. The formulation of the joint probability distribution in Example 12 mirrors that of Example 11, and certain details have been excluded for conciseness. We reproduce Expression (4.7) below:

$$p(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = p(x_1|x_2)p(x_2|x_3)p(x_3|x_4)p(x_4) \\ p(x_5|x_4)p(x_6|x_7)p(x_8|x_7)p(x_7).$$

To generate a sample from the joint probability distribution described by this scheme, we calculate the conditional probability distribution for each node, as outlined below.

$$\begin{aligned}
p(x_1|x_2, x_3, x_4, x_5, x_6, x_7, x_8) &= p(x_1|x_2), \\
p(x_2|x_1, x_3, x_4, x_5, x_6, x_7, x_8) &= \frac{p(x_1|x_2)p(x_2|x_3)}{\sum_{x_2 \in A} p(x_1|x_2)p(x_2|x_3)}, \\
p(x_3|x_1, x_2, x_4, x_5, x_6, x_7, x_8) &= \frac{p(x_2|x_3)p(x_3|x_4)}{\sum_{x_3 \in A} p(x_2|x_3)p(x_3|x_4)}, \\
p(x_4|x_1, x_2, x_3, x_5, x_6, x_7, x_8) &= \frac{p(x_3|x_4)p(x_4)p(x_5|x_4)}{\sum_{x_4 \in A} p(x_3|x_4)p(x_4)p(x_5|x_4)},
\end{aligned}$$

$$
\begin{aligned}
p(x_5|x_1,x_2,x_3,x_4,x_6,x_7,x_8) &= p(x_5|x_4),\\
p(x_6|x_1,x_2,x_3,x_4,x_5,x_7,x_8) &= p(x_6|x_7),\\
p(x_7|x_1,x_2,x_3,x_4,x_5,x_6,x_8) &= \frac{p(x_6|x_7)p(x_8|x_7)p(x_7)}{\sum_{x_7\in A} p(x_6|x_7)p(x_8|x_7)p(x_7)},\\
p(x_8|x_1,x_2,x_3,x_4,x_5,x_6,x_7) &= p(x_8|x_7).
\end{aligned}
$$

The R function for generating a sample in Example 12 is detailed in Section C.2 of Appendix C. The code within the `ex2initialize` function, found in the same section, supplies the theoretical values for the marginal and conditional distributions presented above. These theoretical values serve as inputs for the Gibbs sampler algorithm to generate a sample of the data, following the same approach executed in Example 11.

To assess the proposed algorithm's performance, we initially evaluate them using a single sample. Subsequently, we repeat the process to generate ten distinct samples, calculating average error metrics as defined in Equations (4.2), (4.3), and (4.4) for further analysis of the estimator's performance.

B.1.1 Results

Due to the considerable computational complexity involved in evaluating all potential graphs (amounting to a total of 268,435,456), we omit consideration of the Exact algorithm in this scenario. Instead, we focus on presenting results derived from the Simulated Annealing and Stepwise algorithms, encompassing both Forward and Backward variations.

The outcomes achieved using the Simulated Annealing algorithm closely resemble those observed in Example 11. Following the algorithm specifications outlined in Section 4.1.2, we present the corresponding graphs in Figure B.1. This illustration is for a single sample with varying sizes ($N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$) and a range of penalizing constant values $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$.

For context on this scenario's setup, refer to Figure 4.2. Despite assessing 40 scenarios, the Simulated Annealing algorithm consistently failed to accurately recover the graph structure. In contrast, the Stepwise algorithms displayed favorable outcomes for certain combinations of sample size and penalizing constant. Figure B.2 depicts the results of the Backward Stepwise algorithm. As observed in Example 11, these algorithms correctly capture the actual graph structure for the cases where the sample size is equal to 5,000 and $c \in \{0.25, 0.50, 0.75, 1.00\}$. The same occurs when $N = 10{,}000$ and $c = 0.25$ or $c = 0.50$. For all other combinations of N and c, there were either under or overestimation problems.

Figure B.3 shows the results of the Forward Stepwise algorithm, which exhibited slightly better performance than the Backward Stepwise algorithm. The Forward Stepwise algorithm correctly recovered the true graph in a few more scenarios, specifically when $N = 1{,}000$

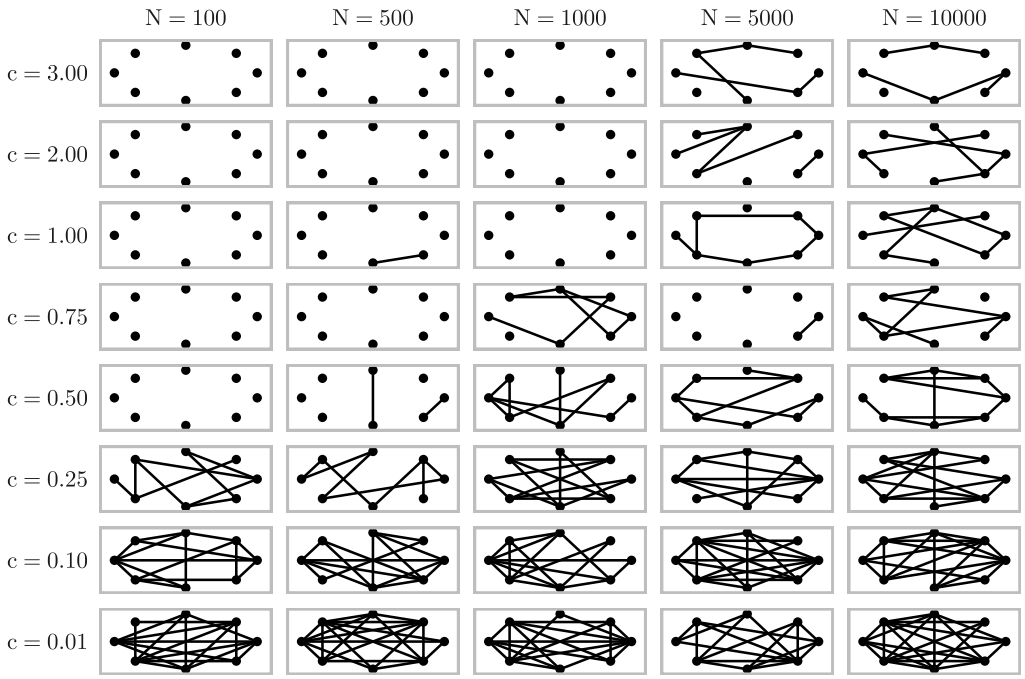


Figure B.1: *Simulated Annealing algorithm results for Example 12, considering several scenarios, with samples of size* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant* $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$. *The node structure resembles that of Figure 4.13.*

and $c = 0.50$, $N = 10{,}000$ and $c \in \{0.25, 0.50, 0.75, 1.00\}$, and $N = 10{,}000$ and $c =\in \{0.25, 0, 50, 0, 75, 1.00, 2.00\}$.

As expected, the overestimation error is considerably high for small penalizing values ($c = 0.01$ or $c = 0.10$). In all other scenarios, an increase in the penalizing constant leads to the emergence of underestimation error.

We then executed the considered algorithms for ten replications of each scenario (combination of sample size and penalizing constant), with a total of 40 combinations. The average of underestimation error, overestimation error, and total error were then computed. The results are visually presented in Figure B.4. The first row of the figure represents the mean underestimation error (ue), overestimation error (oe), and total error (te) values obtained from the performance of the Forward Stepwise algorithm. The second and third rows correspond to the Backward and Simulated Annealing algorithms.

In general, these outcomes closely mirror those obtained for the individual sample case, as illustrated in Figures B.1, B.2, and B.3. Except for the Simulated Annealing algorithm, the combination of relatively large sample sizes and specific penalizing constant values leads to low error rates.

In conclusion, we observe that the Forward and Backward Stepwise algorithms demonstrated effective performance when the sample size is not *small* and when the penalizing constant value was appropriately chosen. Here, the sample size being *small* depends on the number of nodes in the graph.

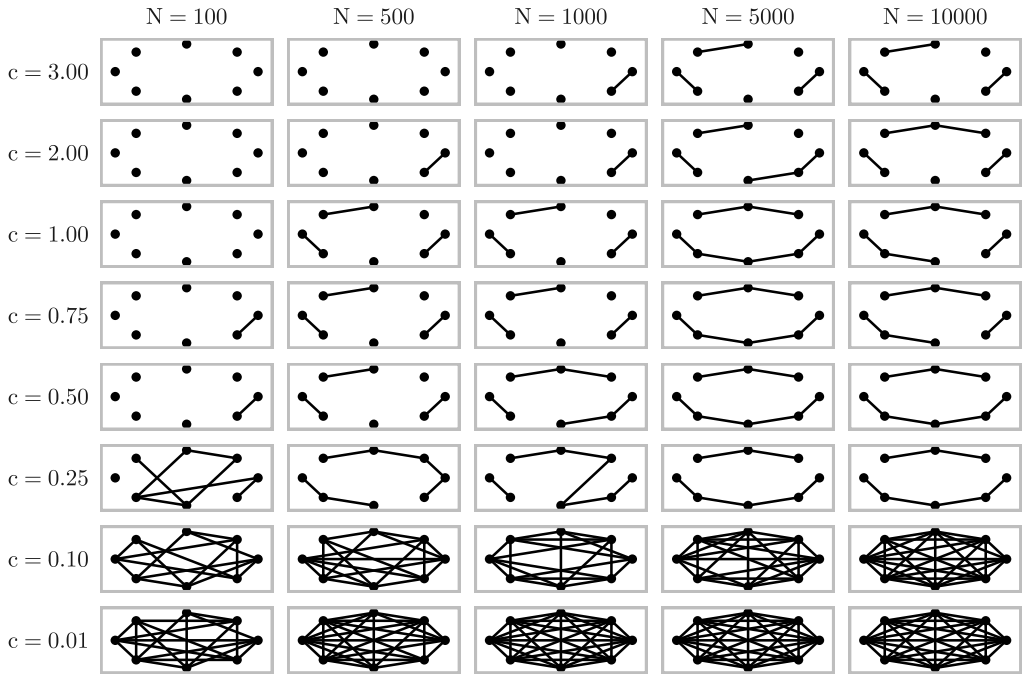


Figure B.2: *Backward Stepwise algorithm results for Example 12, considering several scenarios, with sample of size* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant* $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$. *The node structure resembles that of Figure 4.13.*

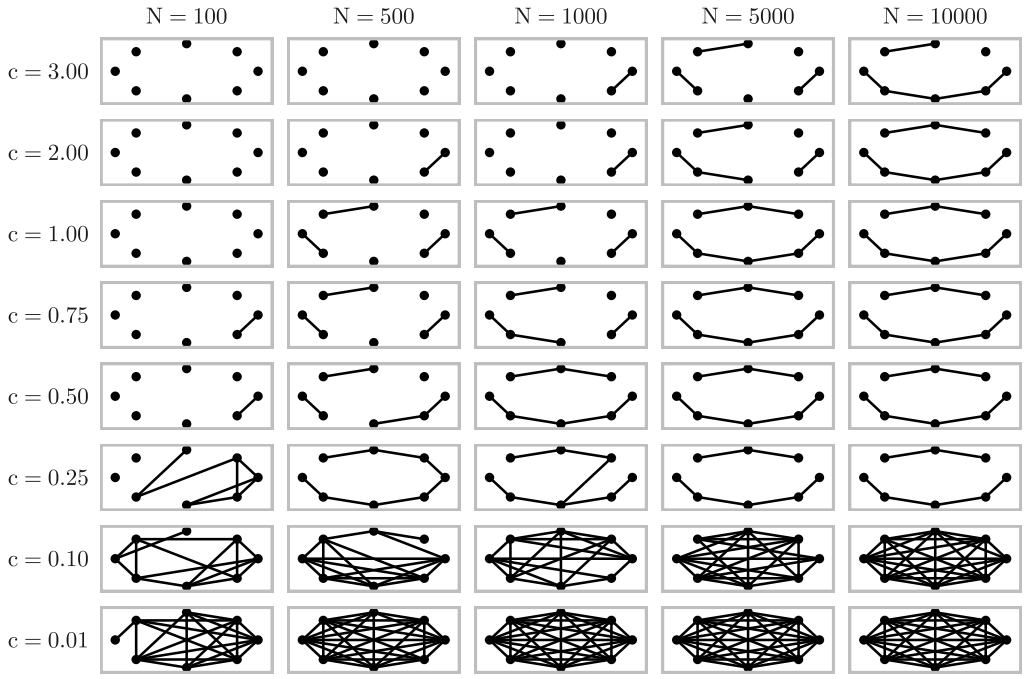


Figure B.3: *Forward Stepwise algorithm results for Example 12, considering several scenarios, with sample of size* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant* $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 3.00\}$. *The node structure resembles that of Figure 4.13.*

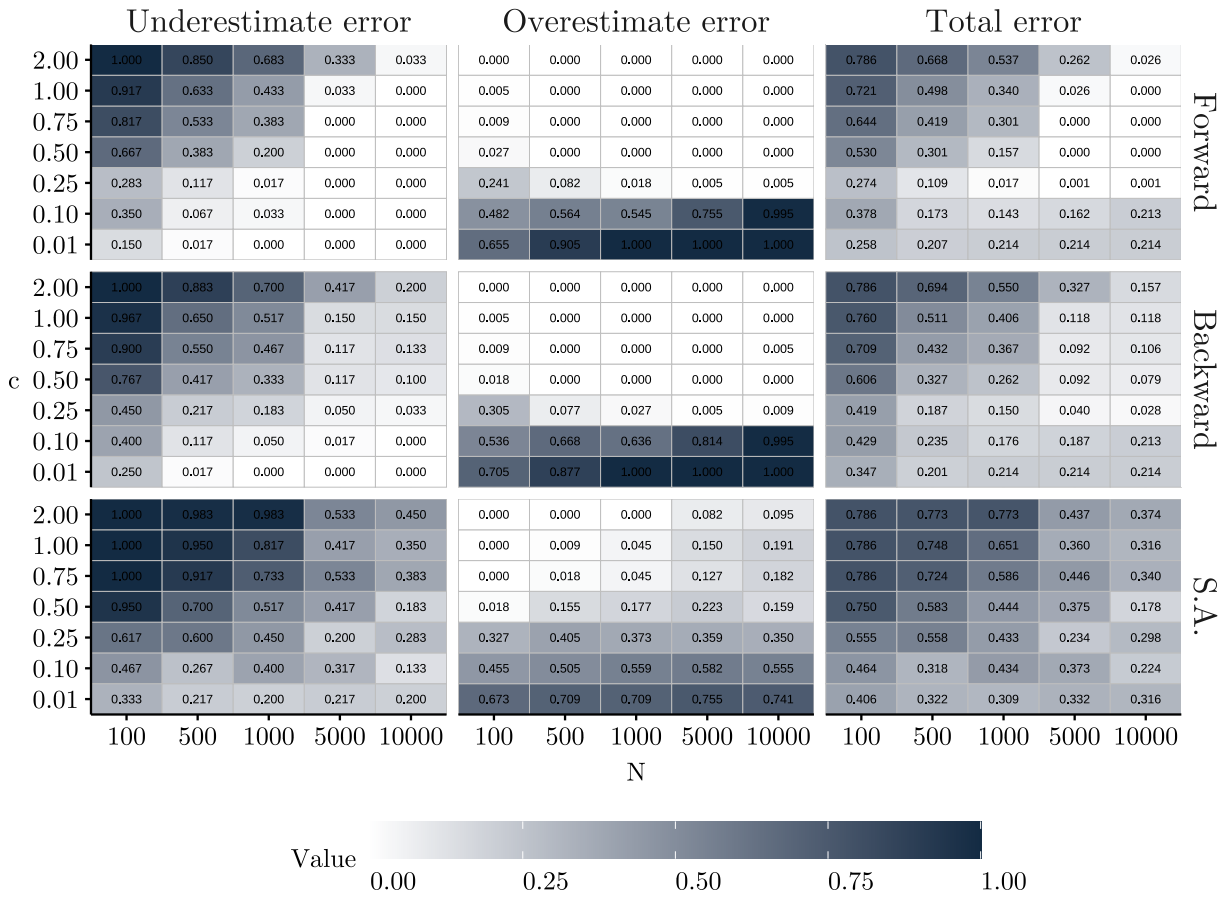


Figure B.4: *Underestimate error, overestimate error, and total error of algorithms considered in the estimation of Example 12. Mean error metrics (underestimation, overestimation, and total error) for the algorithms considering ten samples of each size in Example 12. Sizes considered are* $N \in \{100, 500, 1{,}000, 5{,}000, 10{,}000\}$ *and penalizing constant value* $c \in \{0.01, 0.10, 0.25, 0.50, 0.75, 1.0, 2\}$. *The color in the tile's background indicates the size of the error metric; the darker, the bigger. In general, small penalizing constant values tend to cause overestimation. Conversely, bigger c values tend to cause underestimation, especially when the sample size is small.*

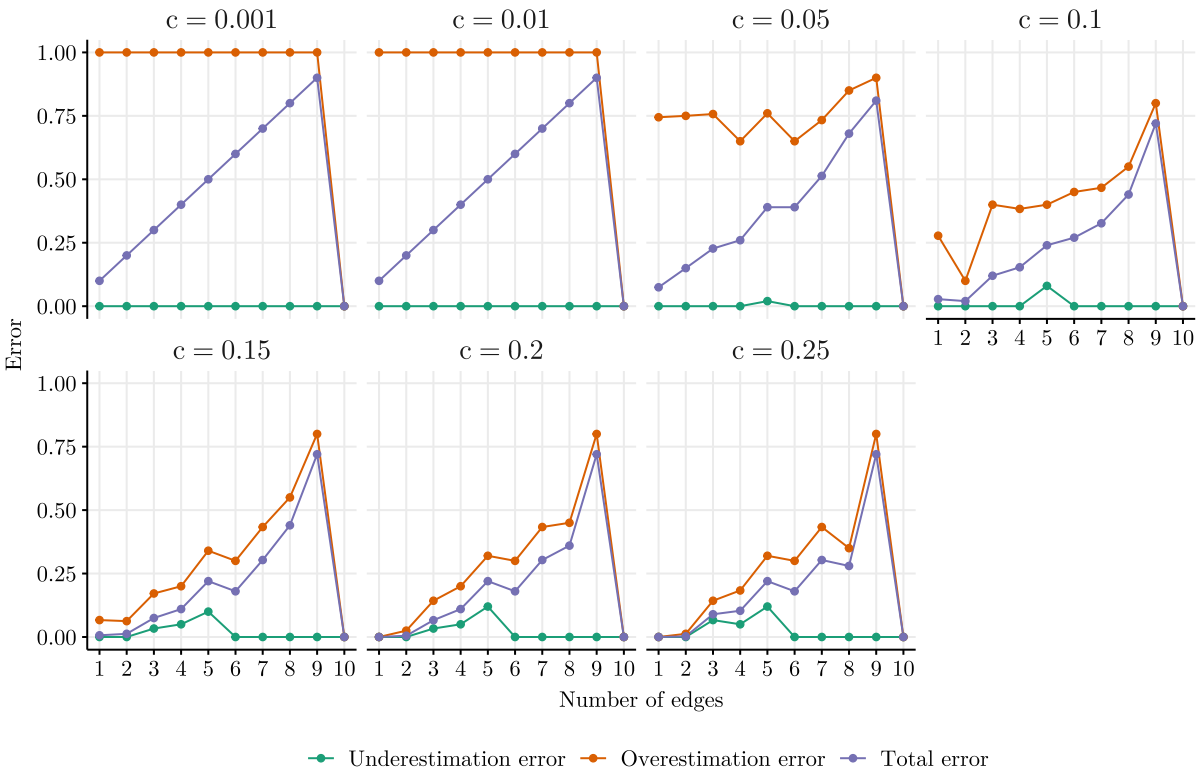


Figure B.5: *Average error metrics outcomes of the Backward Stepwise algorithm for Example 12. Each sub-figure represents the result of a specific c value and displays the averaged errors as a function of the number of edges in the graph.*

B.2 Supplementary Information for Example 13

In this section, we provide the results of the Backward Stepwise algorithm when applied to Example 13, which consists of evaluating 100 scenarios, as shown in Figure 4.14.

An overall summary of the results is depicted in Figure B.5, where each sub-figure corresponds to a specific value of c and shows the average error metrics as function of the number of edges in the graph. This result can be analyzed in two groups. The first one regards small values of the penalizing constant, specifically $\Lambda_1 = \{0.001, 0.010\}$. In this case, the mean overestimation error is high regardless of the number of edges in the graph (except in the complete graph case), and the underestimation error is zero. This is expected as small values of the penalizing constant lead to overestimation.

The second group corresponds to larger values of c, denoted as $\Lambda_2 = \{0.050, 0.100, 0.150, 0.200, 0.250\}$, and showcases a different behavior. The mean overestimation error is relatively smaller for $c \in \Lambda_2$, but it increases with the growth in the number of edges in the graphs. Generally, the underestimation error remains low across all c values. However, for $c \in \Lambda_2$, certain instances exhibit underestimation error, particularly when the number of edges in the graphs ranges between 2 and 5.

Finally, note that regardless of the value of the penalizing constant when the number of edges in the graph is 10 (the complete graph in this case), the average of all errors vanishes.

In particular, by construction, the overestimation error is zero here, and the underestimation error was zero in all scenarios.

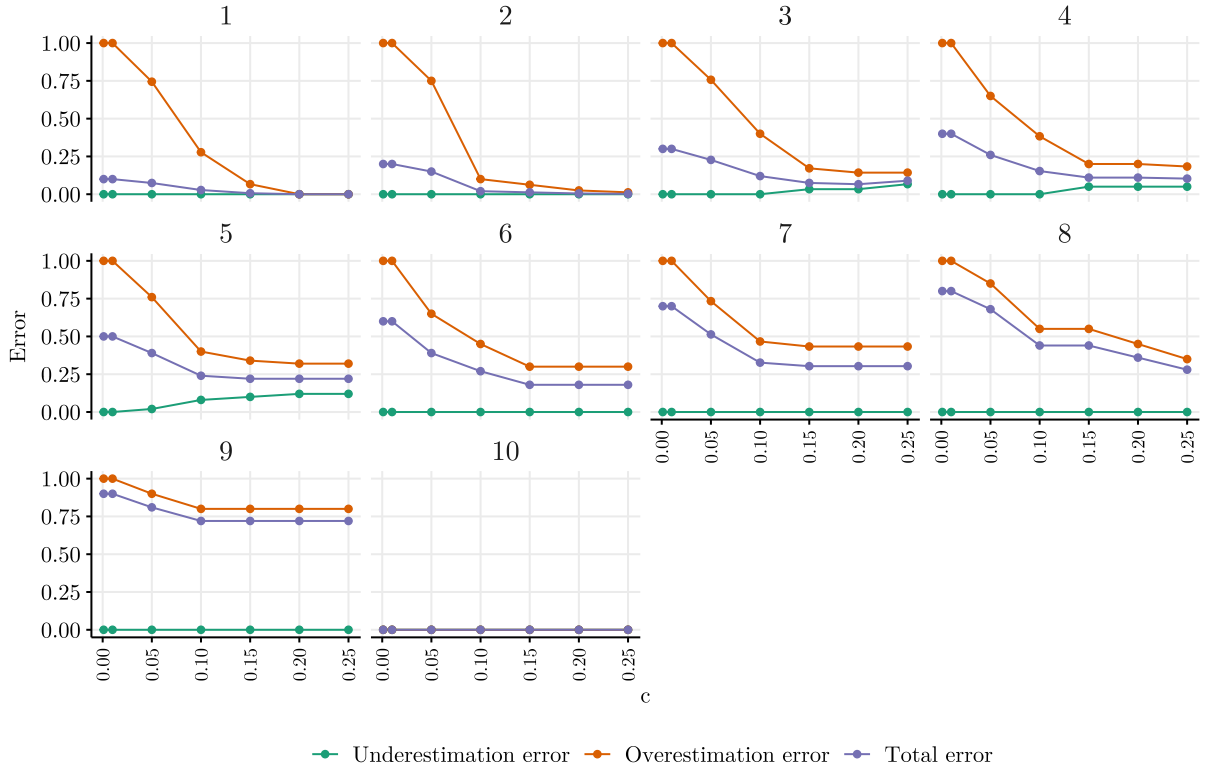


Figure B.6: *Effect of varying the penalizing constant c on the average error metrics outcomes from the Backward Stepwise algorithm for Example 12. Each sub-figure is the result given by graphs with a fixed number of edges, from 1 to 10.*

Additionally, Figure B.6 illustrates the impact of different values of the penalizing constant c on the average error metrics resulting from the Backward Stepwise algorithm. Each subfigure presents the results while maintaining a constant number of edges in the graphs, ranging from 1 to 10. As expected, the error metrics decrease as the penalizing constant value increases. The exception is the case where the number of edges is 9, and the error metrics remain somewhat constant. This phenomenon is attributed to the definition of the overestimation error and the total error. When the number of edges is 10, representing the complete graph, all errors vanish regardless of the penalizing constant value.

Appendix C

R functions

In this Appendix, we provide a collection of R functions that play an important role in implementing various aspects of this thesis. Section C.1 presents the scripts regarding the implementation of our proposed estimator in (2.21) and Section C.2 encompasses the scripts used for generating samples for the examples presented in Chapter 4. The entire set of functions used in this thesis is available in the author's GitHub repository (github.com/magnotairone/phd_codes).

C.1 Estimator

In Listing C.1, we show the function that takes as argument the sample dataset (in a `dataframe` structure) and the graph's adjacency matrix and implements the computation of the terms in the pseudo-likelihood function defined in (2.20). The terms are returned in a list format containing the elements of the sum indexed by the vertices.

```
get_sum_terms <- function(sample_data, G) {
  result <- list()
  for (v in 1:ncol(G)) {
    if (length(which(G[v, ] == 1)) == 0) { # W is empty
      V <- paste("V", v, sep = "")
      a_v_W <- sample_data %>%
        dplyr::group_by_at(c(V)) %>%
        dplyr::count(name = "N_a_v_W") %>%
        dplyr::ungroup() %>%
        dplyr::mutate(N_a_w = nrow(sample_data),
               pi = N_a_v_W * log(N_a_v_W / N_a_w))
      result[[v]] <- a_v_W
      next
    }
    W <- paste("V", which(G[v, ] == 1), sep = "")
    V <- paste("V", v, sep = "")
    a_v_W <- sample_data %>%
      dplyr::group_by_at(c(V, W)) %>%
```

```
      dplyr::count(name = "N_a_v_W") %>%
      dplyr::ungroup()
    a_v <- a_v_W %>%
      dplyr::group_by_at(W) %>%
      dplyr::summarize(N_a_w = sum(N_a_v_W)) %>%
      dplyr::ungroup()
    result[[v]] <- a_v_W %>%
      dplyr::left_join(a_v) %>%
      dplyr::mutate(pi = N_a_v_W * log(N_a_v_W / N_a_w))
  }
  return(result)
}
```

Listing C.1: *Computes the terms in the sum of the log pseudo-likelihood function.*

In Listing C.2, we present the implementation of the penalizing term (2.21). The input arguments are the penalizing factor (number), the cardinality of the alphabet (number), and the adjacency matrix of the graph (matrix).

```
penalty <- function(lambda, card_A, G) {
  return(lambda * sum(card_A ^ colSums(G)))
}
```

Listing C.2: *Computes the estimator's penalty term.*

In Listing C.3, we showcase the function that computes and returns the value of the penalized pseudo-likelihood function (2.21) given the sample data frame and the graph's adjacency matrix.

```
get_penalized_pseudo_log_likelihood <- function(sample_data, G){
  get_sum_terms(sample_data, G) %>%
    sum_terms()
}
```

Listing C.3: *Computes the penilized log-likelihood function.*

C.2 Generating samples

This section presents the functions used to generate samples for each example presented in Chapter 4.

Example 11

We start by presenting the scripts utilized to generate samples for Example 11. The function in Listing C.4 generates the initial state for the five variables in the vector.

```
ex1_initialize <- function(){
  alphabet <- c(0, 1)
```

```
  p1_3 <- matrix(c(1/2, 1/2,
                   1/3, 2/3), nrow = 2, ncol = 2, byrow = T)
  p2_13 <- matrix( c(1/2, 1/2,
                     3/4, 1/4,
                     1/4, 3/4,
                     1/3, 2/3), nrow = 4, ncol = 2, byrow = T)
  p4_3 <- matrix(c(1/5, 4/5,
                   3/5, 2/5), nrow = 2, ncol = 2, byrow = T)
  p5_3 <- matrix(c(2/3, 1/3,
                   1/2, 1/2), nrow = 2, ncol = 2, byrow = T)
  p3 <- c(1/2, 1/2)
  x <- vector(length = 5)
  x[3] <- sample(alphabet, 1, prob = p3)
  x[1] <- sample(alphabet, size = 1, prob = p1_3[x[3]+1,])
  x[4] <- sample(alphabet, size = 1, prob = p4_3[x[3]+1,])
  x[5] <- sample(alphabet, size = 1, prob = p5_3[x[3]+1,])
  x[2] <- sample(alphabet, size = 1, prob = p2_13[2*x[3]+x[1]+1,])
  return(x)
}
```

Listing C.4: *Generation of an initial state for the chain in Example 11.*

Listing C.5 presents the function that is used in the Gibbs sampler algorithm. It generates a new state for the random vector based on the chain's current state, taken as input (vector) of the function.

```
ex1_new_state <- function(x){
  p1_3 <- matrix(c(1/2, 1/2,
                   1/3, 2/3), nrow = 2, ncol = 2, byrow = T)
  p2_13 <- matrix( c(1/2, 1/2,
                     3/4, 1/4,
                     1/4, 3/4,
                     1/3, 2/3), nrow = 4, ncol = 2, byrow = T)
  p4_3 <- matrix(c(1/5, 4/5,
                   3/5, 2/5), nrow = 2, ncol = 2, byrow = T)
  p5_3 <- matrix(c(2/3, 1/3,
                   1/2, 1/2), nrow = 2, ncol = 2, byrow = T)
  p3 <- c(1/2, 1/2)
  y <- vector(length = 5)
  q1 <- p3[x[3]+1] * p1_3[x[3]+1,] * p2_13[2*x[3]+1:2, x[2]+1] /
    sum(p3[x[3]+1] * p1_3[x[3]+1,] * p2_13[2*x[3]+1:2,x[2]+1])
  y[1] <- sample(c(0,1), 1, prob = q1)
  q2 <- p3[x[3]+1] * p1_3[x[3]+1,y[1]+1] * p2_13[2*x[3]+y[1]+1,] /
    sum(p3[x[3]+1] * p1_3[x[3]+1,y[1]+1] * p2_13[2*x[3]+y[1]+1,])
  y[2] <- sample(c(0,1), 1, prob = q2)
  q3 <- p3 * p1_3[,y[1]+1] * p2_13[c(0,2)+y[1]+1,y[2]+1] /
    sum(p3 * p1_3[,y[1]+1] * p2_13[c(0,2)+y[1]+1,y[2]+1])
  y[3] <- sample(c(0,1), 1, prob = q3)
  q4 <- p3[y[3]+1] * p4_3[y[3]+1,] / sum(p3[y[3]+1] * p4_3[y[3]+1,])
```

```
  y[4] <- sample(c(0,1), 1, prob = q4)
  q5 <- p3[y[3]+1] * p5_3[y[3]+1,] / sum(p3[y[3]+1] * p5_3[y[3]+1,])
  y[5] <- sample(c(0,1), 1, prob = q5)
  return(y)
}
```

Listing C.5: *Generation of a new state for the chain in Example 11.*

As an illustrative example, the code snippet below in Listing C.6 demonstrates how to generate a Gibbs sample of size 10,000.

```
n <- 10000
n_nodes <- 5
data <- matrix(ex1_initialize(), ncol = n_nodes)
for(i in 2:n) data <- rbind(data, ex1_new_state(data[i-1,]))
```

Listing C.6: *Gibbs sampler for Example 11*

Example 12

Listing C.7 presents the function to generate an initial state for the random variables in the vector of Example 12. It also showcases the randomly generated empirical probabilities used in this example.

```
ex2_initialize <- function(){
  alphabet <- c(0, 1, 2)
  p1_2 <- matrix(c(1/3, 1/3, 1/3,
                   1/5, 2/5, 2/5,
                   4/9, 4/9, 1/9), nrow = 3, ncol = 3, byrow = TRUE)
  p2_3 <- matrix(c(2/9, 3/9, 4/9,
                   1/3, 1/3, 1/3,
                   1/5, 1/5, 3/5), nrow = 3, ncol = 3, byrow = TRUE)
  p3_4 <- matrix(c(2/8, 2/8, 4/8,
                   4/10, 3/10, 3/10,
                   3/8, 2/8, 3/8), nrow = 3, ncol = 3, byrow = TRUE)
  p4 <- c(1/3, 1/3, 1/3)
  p5_4 <- matrix(c(2/10, 5/10, 3/10,
                   2/6, 3/6, 1/6,
                   2/9, 4/9, 3/9), nrow = 3, ncol = 3, byrow = TRUE)
  p6_7 <- matrix(c(2/7, 2/7, 3/7,
                   2/6, 1/6, 3/6,
                   3/9, 4/9, 2/9), nrow = 3, ncol = 3, byrow = TRUE)
  p7 <- c(3/6, 2/6, 1/6)
  p8_7 <- matrix(c(3/7, 2/7, 2/7,
                   3/8, 3/8, 2/8,
                   2/5, 1/5, 2/5), nrow = 3, ncol = 3, byrow = TRUE)
  x <- vector(length = 8)
  x[4] <- sample(alphabet, 1, prob = p4)
  x[3] <- sample(alphabet, 1, prob = p3_4[x[4] + 1,])
```

```
  x[2] <- sample(alphabet, 1, prob = p2_3[x[3] + 1,])
  x[1] <- sample(alphabet, 1, prob = p1_2[x[2] + 1,])
  x[5] <- sample(alphabet, 1, prob = p5_4[x[4] + 1,])
  x[7] <- sample(alphabet, 1, prob = p7)
  x[6] <- sample(alphabet, 1, prob = p6_7[x[7] + 1,])
  x[8] <- sample(alphabet, 1, prob = p8_7[x[7] + 1,])
  return(x)
}
```

Listing C.7: *Generation of an initial state for the chain in Example 12.*

The function in Listing C.8 generates a new state for the random vector based on the current state (vector) taken as the function's input.

```
ex2_new_state <- function(x){
  alphabet <- c(0, 1, 2)
  p1_2 <- matrix(c(1/3, 1/3, 1/3,
                   1/5, 2/5, 2/5,
                   4/9, 4/9, 1/9), nrow = 3, ncol = 3, byrow = TRUE)
  p2_3 <- matrix(c(2/9, 3/9, 4/9,
                   1/3, 1/3, 1/3,
                   1/5, 1/5, 3/5), nrow = 3, ncol = 3, byrow = TRUE)
  p3_4 <- matrix(c(2/8, 2/8, 4/8,
                   4/10, 3/10, 3/10,
                   3/8, 2/8, 3/8), nrow = 3, ncol = 3, byrow = TRUE)
  p4 <- c(1/3, 1/3, 1/3)
  p5_4 <- matrix(c(2/10, 5/10, 3/10,
                   2/6, 3/6, 1/6,
                   2/9, 4/9, 3/9), nrow = 3, ncol = 3, byrow = TRUE)
  p6_7 <- matrix(c(2/7, 2/7, 3/7,
                   2/6, 1/6, 3/6,
                   3/9, 4/9, 2/9), nrow = 3, ncol = 3, byrow = TRUE)
  p7 <- c(3/6, 2/6, 1/6)
  p8_7 <- matrix(c(3/7, 2/7, 2/7,
                   3/8, 3/8, 2/8,
                   2/5, 1/5, 2/5), nrow = 3, ncol = 3, byrow = TRUE)
  y <- vector(length = 8)
  q4 <- p4 * p3_4[x[4]+1,] * p5_4[x[4]+1,] / sum(p4 * p3_4[x[4]+1,] * p5_
    4[x[4]+1,])
  y[4] <- sample(alphabet, 1, prob = q4)
  q5 <- p5_4[y[4]+1,] / sum(p5_4[y[4]+1,])
  y[5] <- sample(alphabet, 1, prob = q5)
  q3 <- p2_3[x[3]+1] * p3_4[y[4]+1,] / sum(p2_3[x[3]+1] * p3_4[y[4]+1,])
  y[3] <- sample(alphabet, 1, prob = q3)
  q2 <- p1_2[x[2]+1,] * p2_3[y[3]+1,] / sum(p1_2[x[2]+1,] * p2_3[y[3]+1,])
  y[2] <- sample(alphabet, 1, prob = q2)
  q1 <- p1_2[y[2]+1,] / sum(p1_2[y[2]+1,])
  y[1] <- sample(alphabet, 1, prob = q1)
  q7 <- p7 * p6_7[x[7]+1,] * p8_7[x[7]+1,] / sum(p7 * p6_7[x[7]+1,] * p8_
```

```
    7[x[7]+1,])
  y[7] <- sample(alphabet, 1, prob = q7)
  q6 <- p6_7[y[7]+1,] / sum(p6_7[y[7]+1,])
  y[6] <- sample(alphabet, 1, prob = q6)
  q8 <- p8_7[y[7]+1,] / sum(p8_7[y[7]+1,])
  y[8] <- sample(alphabet, 1, prob = q8)
  return(y)
}
```

Listing C.8: *Generation of a new state for the chain in Example 12.*

As the data generation process is very similar to the code chunk presented in Listing C.6, we omit the respective code chunk for Example 12.

Example 13

As this example involves generating random scenarios, in Listing C.9 we define the function that randomly creates a symmetric adjacency matrix to represent an undirected graph with a specified number of vertices (`n_vertices`) and edges (`n_edges`). It ensures that the number of edges does not exceed the maximum possible for a symmetric matrix and employs a while loop to randomly assign edges between vertices until the desired number of edges is reached.

```
generate_adj_matrix <- function(n_vertices, n_edges){
  if (n_edges > n_vertices * (n_vertices - 1) / 2) {
    stop("The number of edges cannot exceed the maximum possible number of
    edges in a symmetric matrix.")
  }
  adj_matrix <- matrix(0, nrow = n_vertices, ncol = n_vertices)
  edge_counter <- 0
  while (edge_counter < n_edges) {
    row <- sample(1:n_vertices, 1)
    col <- sample(1:n_vertices, 1)

    if (row != col && adj_matrix[row, col] == 0) {
      adj_matrix[row, col] <- 1
      adj_matrix[col, row] <- 1
      edge_counter <- edge_counter + 1
    }
  }
  return(adj_matrix)
}
```

Listing C.9: *Generation of a random adjacency matrix.*

Listing C.10 presents the function that takes the graph's adjacency matrix as input and produces random conditional probability distributions for each vertex based on its neighbors in the graph. It initializes a list to store the distributions, iterates over each vertex, identifies

its neighbors, and calculates the number of possible combinations of states for the neighbors. The function generates random conditional probabilities for each combination and organizes them into a matrix. Finally, these matrices are stored in a list, and the function returns this list of conditional probability distributions for each vertex in the graph.

```
generate_conditional_distributions <- function(adjacency_matrix) {
  num_vertices <- nrow(adjacency_matrix)
  distributions <- vector("list", num_vertices)
  for (i in 1:num_vertices) {
    neighborhood <- adjacency_matrix[i, ]
    neighbors <- which(neighborhood == 1)
    num_neighbors <- length(neighbors)
    num_combinations <- 2^num_neighbors
    conditional_probs <- matrix(0, nrow = num_combinations, ncol = 2)
    for (j in 1:num_combinations) {
      probabilities <- runif(2)
      probabilities <- probabilities / sum(probabilities)
      conditional_probs[j, ] <- probabilities
    }
    distributions[[i]] <- conditional_probs
  }
  return(distributions)
}
```

Listing C.10: *Generation of random conditional distribution functions.*

The complete functions used in implementing the algorithms proposed in this thesis can be found in the repository in the author's Github profile.

Bibliography

Atchade(2014) Yves F. Atchade. Estimation of high-dimensional partially-observed discrete markov random fields. *Electron. J. Statist.*, 8(2):2242–2263. Cited on page 1

Besag *et al.***(1995)** Julian Besag, Peter Green, David Higdon and Kerrie Mengersen. Bayesian computation and stochastic systems. *Statistical science*, páginas 3–41. Cited on page 29

Bresler *et al.***(2018)** Guy Bresler, David Gamarnik and Devavrat Shah. Learning graphical models from the Glauber dynamics. *IEEE Trans. Inform. Theory*, 64(6):4072–4080. ISSN 0018-9448. Cited on page 1

Cerqueira *et al.***(2017)** Andressa Cerqueira, Daniel Fraiman, Claudia D. Vargas and Florencia Leonardi. A test of hypotheses for random graph distributions built from eeg data. *IEEE Transactions on Network Science and Engineering*, 4(2):75–82. Cited on page 2

Comets(1992) Francis Comets. On Consistency of a Class of Estimators for Exponential Families of Markov Random Fields on the Lattice. *The Annals of Statistics*, 20(1):455 – 468. Cited on page 2

Comets and Gidas(1992) Francis Comets and Basilis Gidas. Parameter Estimation for Gibbs Distributions from Partially Observed Data. *The Annals of Applied Probability*, 2 (1):142 – 170. Cited on page 2

Csiszár(2002) I. Csiszár. Large-scale typicality of Markov sample paths and consistency of MDL order estimators. *IEEE Trans. Inform. Theory*, 48(6):1616–1628. ISSN 0018-9448. Special issue on Shannon theory: perspective, trends, and applications. Cited on page 12

Csiszár and Talata(2006) I. Csiszár and Z. Talata. Consistent estimation of the basic neighborhood of Markov random fields. *The Annals of Statistics*, 34(1):123–145. ISSN 0090-5364. Cited on page 2, 66

Diaconis and Holmes(1995) Persi Diaconis and Susan Holmes. Three examples of monte-carlo markov chains: at the interface between statistical computing, computer science, and statistical mechanics. Em *Discrete probability and algorithms*, páginas 43–56. Springer. Cited on page 29

Divino *et al.*(**2000**) Fabio Divino, Arnoldo Frigessi and Peter J. Green. Penalized pseudo-likelihood inference in spatial interaction models with covariates. *Scandinavian Journal of Statistics*, 27(3):445–458. Cited on page 1

Duarte *et al.*(**2019**) Aline Duarte, Antonio Galves, Eva Löcherbach and Guilherme Ost. Estimating the interaction graph of stochastic neural dynamics. *Bernoulli*, 25(1):771–792. ISSN 1350-7265. Cited on page 1

Galves *et al.*(**2015**) Antonio Galves, Enza Orlandi and Daniel Y. Takahashi. Identifying interacting pairs of sites in Ising models on a countable set. *Braz. J. Probab. Stat.*, 29(2): 443–459. Cited on page 1

Gelfand and Smith(1990) Alan E Gelfand and Adrian FM Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409. Cited on page 33

Geman and Geman(1984) Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741. Cited on page 33

Georgii(2011) Hans-Otto Georgii. *Gibbs measures and phase transitions*, volume 9 of *de Gruyter Studies in Mathematics*. Walter de Gruyter & Co., Berlin, second edição. Cited on page 2

Hastie *et al.*(**2009**) Trevor Hastie, Robert Tibshirani, Jerome H Friedman and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer. Cited on page 44

Höfling and Tibshirani(2009) Holger Höfling and Robert Tibshirani. Estimation of sparse binary pairwise markov networks using pseudo-likelihoods. *Journal of Machine Learning Research*, 10(32):883–906. Cited on page 1

Investing.com(2023) Investing.com. Major World Market Indices, 2023. URL https://www.investing.com/indices/major-indices. Accessed: 2023-12-30. Cited on page 59

James *et al.*(**2021**) Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. *An introduction to statistical learning*. Springer. Cited on page 27, 44

Ji and Seymour(1996) C. Ji and L. Seymour. A consistent model selection procedure for Markov random fields based on penalized pseudolikelihood. *The Annals of Applied Probability*, 6(2):423 – 443. Cited on page 2

Koller and Friedman(2009) Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press. ISBN 0262013193, 9780262013192. Cited on page 1

Lafferty *et al.***(2012)** John Lafferty, Han Liu and Larry Wasserman. Sparse Nonparametric Graphical Models. *Statistical Science*, 27(4):519 – 537. Cited on page 2

Lauritzen(1996) Steffen L Lauritzen. *Graphical models*, volume 17. Clarendon Press. Cited on page 1, 6

Leonardi *et al.***(2021)** Florencia Leonardi, Matías Lopez-Rosenfeld, Daniela Rodriguez, Magno TF Severino and Mariela Sued. Independent block identification in multivariate time series. *Journal of Time Series Analysis*, 42(1):19–33. Cited on page 2, 3, 55, 58, 63

Leonardi *et al.***(2023)** Florencia Leonardi, Rodrigo Carvalho and Iara Frondana. Structure recovery for partially observed discrete markov random fields on graphs under not necessarily positive distributions. *Scandinavian Journal of Statistics (accepted)*. Cited on page 1, 2, 3, 6, 7, 17, 18, 26, 52, 63, 64, 67

Lerasle and Takahashi(2016) Matthieu Lerasle and Daniel Y. Takahashi. Sharp oracle inequalities and slope heuristic for specification probabilities estimation in discrete random fields. *Bernoulli*, 22(1):325–344. ISSN 1350-7265. Cited on page 1

Liu *et al.***(2012)** Han Liu, Fang Han, Ming Yuan, John Lafferty and Larry Wasserman. High-dimensional semiparametric Gaussian copula graphical models. *The Annals of Statistics*, 40(4):2293 – 2326. Cited on page 2

Löcherbach and Orlandi(2011) Eva Löcherbach and Enza Orlandi. Neighborhood radius estimation for variable-neighborhood random fields. *Stochastic Process. Appl.*, 121(9): 2151–2185. ISSN 0304-4149. Cited on page 2

Loh and Wainwright(2013) Po-Ling Loh and Martin J. Wainwright. Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. *Ann. Statist.*, 41(6):3022–3049. Cited on page 2

Meinshausen and Bühlmann(2006) N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *Ann. Statist.*, 34(3):1436–1462. Cited on page 2

Oodaira and Yoshihara(1971) Hiroshi Oodaira and Ken-ichi Yoshihara. The law of the iterated logarithm for stationary processes satisfying mixing conditions. Em *Kodai Mathematical Seminar Reports*, volume 23, páginas 311–334. Department of Mathematics, Tokyo Institute of Technology. Cited on page 66, 67

Pensar *et al.***(2017)** Johan Pensar, Henrik Nyman and Jukka Corander. Structure learning of contextual markov networks using marginal pseudo-likelihood. *Scandinavian Journal of Statistics*, 44(2):455–479. Cited on page 1

R Core Team(2022) R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. Cited on page 31

Ravikumar *et al.***(2010)** Pradeep Ravikumar, Martin J. Wainwright and John D. Lafferty. High-dimensional Ising model selection using l_1-regularized logistic regression. *Ann. Statist.*, 38(3):3022–1319. Cited on page 1, 2

Ross(2006) Sheldon M Ross. *Simulation*. Statistical modeling and decision science. Academic Press, fourth edição. ISBN 9780125984102. Cited on page 29

Santhanam and Wainwright(2012) Narayana P. Santhanam and Martin J. Wainwright. Information-theoretic limits of selecting binary graphical models in high dimensions. *IEEE Trans. Inform. Theory*, 58(7):4117–4134. ISSN 0018-9448. Cited on page 2

Schwarz(1978) G. Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6:461–464. Cited on page 2

Shojaie and Michailidis(2010) A. Shojaie and G. Michailidis. Penalized likelihood methods for estimation of sparse high-dimensional directed acyclic graphs. *Biometrika*, 97(3): 519–538. Cited on page 1

SNIRH(2023) SNIRH. Sistema Nacional de Informações sobre Recursos Hídricos, Portal HidroWeb, 2023. URL http://www.snirh.gov.br/hidroweb/. Accessed: 2023-12-30. Cited on page 54

Strauss and Ikeda(1990) David Strauss and Michael Ikeda. Pseudolikelihood Estimation for Social Networks. *Journal of the American Statistical Association,*, 85(409):204–212. Cited on page 1

Tjelmeland and Besag(1998) Håkon Tjelmeland and Julian Besag. Markov random fields with higher-order interactions. *Scandinavian Journal of Statistics*, 25(3):415–433. Cited on page 2

Wickham *et al.***(2019)** Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo and Hiroaki Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686. Cited on page 55

Yang *et al.***(2015)** Eunho Yang, Pradeep Ravikumar, Genevera I. Allen and Zhandong Liu. Graphical models via univariate exponential family distributions. *Journal of Machine Learning Research*, 16(115):3813–3847. Cited on page 2