Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Rodrigo Ribeiro Santos de Carvalho

Algoritmos para Estimação de Modelos Gráficos

São Paulo
Novembro de 2018

Algoritmos para Estimação de Modelos Gráficos

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisora: Profª. Drª. Florencia Graciela Leonardi

Durante a elaboração deste trabalho o autor recebeu apoio financeiro do CNPq

São Paulo
Novembro de 2018

Agradecimentos

Agradeço à professora Florencia Leonardi pela solicitude ao longo do ano.

Aos meus pais, Carlos Augusto e Célia, pelo incondicional apoio durante a minha graduação.

Agradeço também aos meus amigos do IME-USP pelas boas conversas e conselhos.

Resumo

O modelo gráfico probabilístico é uma representação em grafo das dependências condicionais de um vetor aleatório. Um problema de pesquisa atual é a reconstrução, a partir de amostras, do grafo original. Neste trabalho estudamos dois algoritmos presentes na literatura. Realizamos diversas simulações para verificar empiricamente a consistência dos estimadores. Por fim, aplicamos um dos algoritmos em problemas com dados reais.

Palavras-chave: Campo aleatório de Markov, Estimador de máxima verossimilhança penalizada, algoritmo de Chow-Liu.

Abstract

The probability graphical model is a representation in graph of the conditional dependences of a random vector. A current research problem is the reconstruction of the original graph from samples. In this monograph, we study two algorithms present in the literature, and we performed several simulations to check empirically the consistency of the estimators. Finally, we apply one of the algorithms in problems with real data.

Keywords: Markov random Field, Maximum penalized likelihood estimator, Chow-Liu algorithm.

Sumário

Capítulo 1

Introdução

O modelo gráfico probabilístico é uma representação em forma de grafo de uma distribuição conjunta de probabilidade. Ele pode ser descrito por um grafo direcionado, como nos modelos de redes Bayesianas ou não direcionado, como nos campos aleatórios de Markov (Lauritzen, 1996). Cada vértice do grafo é uma variável aleatória do vetor e seus arcos ou arestas são definidos conforme a independência condicional entre eles. Suas aplicações se encontram em áreas como a biologia, o aprendizado de máquinas e as ciências sociais, onde os modelos podem possuir um grande número de variáveis (Frondana, 2016; Hastie *et al.*, 2009).

Do ponto de vista estatístico e computacional, um problema de pesquisa atual é a reconstrução, a partir de amostras, do grafo que expressa as independências condicionais entre as variáveis. Neste trabalho estudamos dois algoritmos existentes na literatura para reconstrução do campo aleatório de Markov discreto; isto é, as variáveis assumem apenas valores num conjunto finito. O primeiro é um algoritmo proposto por Frondana (2016) e é baseado no calculo da máxima verossimilhança penalizada em cada nó do grafo. O outro, chamado de algoritmo de Chow-Liu, estima um grafo de tipo árvore que mais se aproxima, em relação à divergência de Kullback-Leibler, da distribuição original.

Apresentamos simulações em diversos cenários para verificar empiricamente a consistência dos algoritmos, inclusive testamos em altas dimensões, testando para um grafo esparso de 500 vértices.

Por fim, realizamos duas aplicações envolvendo dados reais. Na primeira, analisamos um conjunto de microarrays de pacientes com câncer de mama e de pacientes normais. O objetivo é investigar se há diferença entre as dependências entre os genes para os pacientes saudáveis e com a doença. Na segunda aplicação estudamos uma série histórica de índices de bolsas de valores de 6 países e a intenção é verificar as dependências condicionais entre as bolsas de valores.

Capítulo 2

Algoritmos

2.1 Definições Básicas

Seja $G = (V, E)$ um grafo onde V é o conjunto de vértices e E o conjunto de arestas. Dizemos que G é um grafo *não-direcionado* se para cada $(v, u) \in E$ implica que $(u, v) \in E$. Em um grafo *direcionado*, a implicação não existe. O *grau* de um vértice v de um grafo não-direcionado é definido como o número de elementos do conjunto $\{u \in V : (v, u) \in E\}$. Para um grafo direcionado o *grau de entrada* e o *grau de saída* são definidas pela cardinalidade de $\{u \in V : (u, v) \in E\}$ e $\{u \in V : (v, u) \in E\}$, respectivamente.

Sejam $G = (V, E)$ um grafo não-direcionado finito e A um conjunto finito que chamaremos de *alfabeto*. Um *campo aleatório* (em relação ao grafo G) sobre A^V é uma família de variáveis aleatórias indexadas pelo conjunto de vértices $\{X_v : v \in V\}$, onde $X_v \in A$.

Seja $\{X_v : v \in V\}$ um campo aleatório, dizemos que $\{X_v : v \in V\}$ é um *campo aleatório de Markov* se para cada $v \in V$ existir um conjunto $ne(v) \subseteq V \setminus \{v\}$ tal que para todo $W \subseteq V \setminus \{v \cup ne(v)\}$

$$\mathbb{P}(X_v = a_v | X_{ne(v)} = a_{ne(v)}) = \mathbb{P}(X_v = a_v | X_{ne(v)} = a_{ne(v)}, X_W = a_W)$$

onde $a_v \in A$, $a_{ne(v)} \in A^{ne(v)}$, $a_W \in A^W$ e $\mathbb{P}(X_{ne(v)} = a_{ne(v)}, X_W = a_W) > 0$. O conjunto $ne(v)$ é chamado *vizinhança Markoviana* de v. Temos que se $w \in ne(v)$ então $(w, v) \in E$.

Para simplificar a notação, denotaremos $\mathbb{P}(X_v = a_v | X_W = a_W)$ por $p(a_v | a_W)$.

Seja $x_V^{(1:n)}$ uma amostra independente de tamanho n de um campo aleatório de Markov $\{X_v, v \in V\}$. x_v^i denotará a i-ésima observação do vértice v. Sejam $v \in V$ e $W \subseteq V \setminus \{v\}$. O operador $N(a_v, a_W)$, que conta o número de eventos $\{X_v = a_v, X_W = a_W\}$ na amostra, é definido por

$$N(a_v, a_W) = \sum_{i=1}^{n} \mathbf{1}\{x_v^i = a_v, x_W^i = a_W\}$$

.

A *divergência de Kullback-Leibler* entre duas distribuições de probabilidade P e Q sobre A^V é definida por

$$D_{KL}(P||Q) = -\sum_{a_V \in A^V} P(a_V) \log \frac{Q(a_V)}{P(a_V)}$$

onde $Q(a_v) = 0$ implica $P(a_V) = 0$ e se $P(a_V) = 0$ então $P(a_V) \log \frac{Q(a_V)}{P(a_V)} = 0$.

2.2 Algoritmo baseado em máxima verossimilhança penalizada

O algoritmo proposto por Frondana (2016) estima cada vizinhança dos vértices através do cálculo da máxima log-verossimilhança penalizada por um fator que depende diretamente do tamanho do conjunto da vizinhança. A ideia da penalização é evitar o sobreajuste (*overfitting*) dos dados, que surge através da estimação de um modelo complexo pelo algoritmo. Um modelo complexo se ajusta bem aos dados de treinamento, mas falha em generalizar para novos dados, especialmente em altas dimensões (Giraud., 2014). A função de log-verossimilhança penalizada é definida como

$$M(\theta) = \log \mathcal{L}(\theta|x) - cP(\theta)$$

onde

1. $\mathcal{L}(\theta|x)$ é a função de verossimilhança de parâmentro θ dada uma amostra x
2. $P(\theta)$ é uma função de penalidade que em geral é crescente em relação ao aumento da complexidade do parâmetro θ
3. c é um número real chamado de *regularizador*, que determina o peso que o penalizador terá no cálculo da estimação

A ideia, portanto, é encontrar uma vizinhança para cada vértice balanceando os fatores de verossimilhança e de penalidade. Sejam $\{X_v : v \in V\}$ um campo aleatório de Markov sobre A^V e uma amostra $x_V^{(1:n)}$ desse campo aleatório de tamanho n. A função de verossimilhança para uma vizinhança W de v é

$$\mathcal{L}_v(W; x_V^{(1:n)}) = \prod_{i=1}^{n} \mathbb{P}(X_v = x_v^i | X_W = x_W^i)$$

Para uma simplicação em termos computacionais, podemos reescrever a função acima como

$$\mathcal{L}_v(W; x_V^{(1:n)}) = \prod_{a_v \in A} \prod_{a_W \in A^W} p(a_v|a_W)^{N(a_v, a_W)}.$$

A função de probabilidade $p(a_v|a_W)$, para todo $a_v \in A$ e $a_W \in A^W$, é desconhecida, portanto a estimamos por

$$\widehat{p}(a_v|a_W) = \frac{N(a_v, a_W)}{N(a_W)}.$$

onde $N(a_W) = \sum_{a_v \in A} N(a_v, a_W)$. Assim, a verossimilhança da distribuição condicional de X_v dado X_W utilizada no algoritmo é

$$\widehat{\mathbb{P}}(x_v^{(1:n)}|x_W^{(1:n)}) = \prod_{a_v \in A} \prod_{a_W \in A^W} \widehat{p}(a_v|a_W)^{N(a_v,a_W)}.$$

Utilizando as propriedades do logaritmo, o log-verossimilhança será

$$\log \widehat{\mathbb{P}}(x_v^{(1:n)}|x_W^{(1:n)}) = \sum_{a_v \in A} \sum_{a_W \in A^W} N(a_v, a_W) \log \widehat{p}(a_v|a_W).$$

Frondana (2016) propôs o uso de um regularizador $c > 0$ e uma função de penalidade

$$P(W) = c|A|^{|W|} \log_{|A|} n.$$

Observe que a função é exponencial em relação à cardinalidade do conjunto W.

Agora estamos prontos para definir a *vizinhança markoviana empírica* de v. Ela é dada por

$$\widehat{ne}(v) = \arg\max_{W \subseteq V \setminus \{v\}} \{\log \widehat{\mathbb{P}}(X_v^{(1:n)}|X_W^{(1:n)}) - c|A|^{|W|} \log_{|A|} n\}$$

Para reconstruir o grafo a partir das vizinhanças de cada nó, podemos utilizar duas abordagens. Na primeira definimos o conjunto de arestas *conservativo*

$$\widehat{E}^- = \{(v,u) \in V^2 : v \in \widehat{ne}(u) \text{ e } u \in \widehat{ne}(v)\}$$

Com esse conjunto de arestas construímos o grafo estimado conservativo $\widehat{G}^- = (V, \widehat{E}^-)$. A outra abordagem é o método não conservativo, cujo conjunto de arestas é

$$\widehat{E}^+ = \{(v,u) \in V^2 : v \in \widehat{ne}(u) \text{ ou } u \in \widehat{ne}(v)\}$$

e o seu grafo estimado não-conservativo é definido por $\widehat{G}^+ = (V, \widehat{E}^+)$.

O seguinte teorema mostra que de fato o estimador reconstroi o grafo do campo aleatório de Markov com probabilidade 1, para um número suficientemente grande de amostra.

Teorema 1. *Para qualquer $v \in V$ e regularizador $c > 0$, $\widehat{ne}(v) \xrightarrow{q.c.} ne(v)$ quando $n \to \infty$.*

Demonstração. É um prova longa, que faz uso da lei do logaritmo iterado e da lei forte dos grandes números. Ela se encontra em Frondana (2016). □

Na próxima página apresentamos os pseudo-códigos dos algoritmos conservativo e não-

conservativo e a respectiva análise de complexidade.

Algoritmo 1: PML - não conservativo

Entrada: amostra $i.i.d$ de tamanho n de um campo aleatório de Markov
Saída: um conjunto de arestas E

início
  $E = \emptyset$
  para *cada* $v \in V$ **faça**
    $\widehat{ne}(v) = \arg\max_{W \subseteq V \setminus \{v\}} \{\log \hat{\mathbb{P}}(x_v^{(1:n)} | x_W^{(1:n)}) - c|A|^{|W|} \log_{|A|} n\}$
  fim
  para *cada* $(w, u) \in V^2$ **faça**
    $E = E \cup \{(w, u) \in V^2 : w \in \widehat{ne}(u) \text{ ou } v \in \widehat{ne}(w)\}$
  fim
  retorna E
fim

Algoritmo 2: PML - conservativo

Entrada: amostra $i.i.d$ de tamanho n de um campo aleatório de Markov
Saída: um conjunto de arestas E

início
  $E = \emptyset$
  para *cada* $v \in V$ **faça**
    $\widehat{ne}(v) = \arg\max_{W \subseteq V \setminus \{v\}} \{\log \hat{\mathbb{P}}(x_v^{(1:n)} | x_W^{(1:n)}) - c|A|^{|W|} \log_{|A|} n\}$
  fim
  para *cada* $(w, u) \in V^2$ **faça**
    $E = E \cup \{(w, u) \in V^2 : w \in \widehat{ne}(u) \text{ e } v \in \widehat{ne}(w)\}$
  fim
  retorna E
fim

Análise de tempo: Para cada $O(|V|)$ vértices, realizamos $O(2^{|V|-1})$ testes de tempo $O(n)$. Portanto a complexidade do algoritmo é $O(|V|2^{|V|-1}n)$

2.3 Algoritmo de Chow-Liu

Para um campo aleatório de Markov cujo grafo é uma árvore, existe um algoritmo eficiente para a sua reconstrução. É um algoritmo de complexidade $O(n|V|^2 + |V|^2 \log_2 |V|)$, onde V é o conjunto de vértices do grafo e n é o tamanho da amostra. A ideia é aproximar a distribuição conjunta P por uma distribução conjunta de segunda ordem P_t de forma que elas tenham a menor divergência de Kullback-Leibler possível.

Definição 1. *Seja* $(X_1, ..., X_n)$ *um vetor aleatório com distribuição de probabilidade P. Dizemos que* P_t *é uma distribuição de* segunda ordem *(em relação à P) se*

$$P_t(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | X_{j(i)})$$

onde $0 \leq j(i) \leq n$, $j(i) \neq i$ *e que exista apenas um* $k \in \{1, ..., n\}$ *tal que j(k) = 0. Por conversão,* $P(X_j | X_0) = P(X_j)$

Definição 2. *Para duas variáveis aleatórias discretas X e Y, assumindo valores em um alfabeto finito A, defina-se a* medida de informação mútua *entre essas variáveis como*

$$I(X;Y) = \sum_{y\in A}\sum_{x\in A} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

Para toda aresta (X_i, X_j) do grafo, podemos calcular sua medida de informação mútua $I(X_i;X_j)$ e dizer que esse valor é o peso da aresta. Notando que a representação em grafo de uma distribuição conjunta de segunda ordem é sempre árvore, a seguir temos um teorema central para o funcionamento do algoritmo

Teorema 2. *Considere um vetor aleatório $(X_1, X_2, ..., X_n)$ com duas funções de probabilidade conjunta P_t, de segunda ordem, e P. A função P_t é uma aproximação ótima de P em relação à divergência de Kullback-Leibler se e somente se a árvore de P_t for a árvore geradora máxima considerando todas as possíveis árvores formadas pelos vértices $(X_1, X_2, ..., X_n)$.*

Demonstração. De acordo com Behsaz e Rahmati (2006),

$$D_{KL}(P||P_t) = -\sum_{i=1}^{n} I(X_i, X_{j(i)}).$$

Portanto se queremos minimizar a divergência de Kullback-Leibler, teremos que encontrar uma árvore que maximiza a soma das medidas de informação mútua das arestas.

□

Da mesma forma que no algoritmo de Frondana (2016) não dispomos da probabilidade P para o cálculo da medida de informação mútua. Considerando um campo aleatório de Markov em A^V com uma amostra de tamanho n, a estimativa de máxima verossimilhança de $I(X_v;X_w)$ é

$$\widehat{I}(X_v;X_w) = \sum_{a_v\in A}\sum_{a_w\in A}\frac{N(a_v,a_w)}{n}[\log\frac{N(a_v,a_w)}{N(a_v)N(a_w)} + \log n]$$

Calculando todas as estimativas de medida de informação mútua de todos os pares de vértices, aplicamos um algoritmo para encontrar a árvore geradora máxima (por exemplo, algoritmo de Krushal) e reconstruir o grafo.

Algoritmo 3: Chow-Liu

Entrada: amostra *i.i.d* de tamanho n de um campo aleatório de Markov
Saída: Um conjunto de arestas E

```
início
    para cada conjunto {v, w} ∈ V² faça
        Î(X_v, X_w) = Σ_{a_v∈A} Σ_{a_w∈A} N(a_v,a_w)/n [log N(a_v,a_w)/(N(a_v)N(a_w)) + log n]
    fim
    E = Kruskal(Î)
    retorna E
fim
```

Análise de tempo: Para cada $O(|V|^2)$ pares de vértices, calculamos a medida de informação mútua com tempo $O(n)$. E depois, utilizamos o algoritmo de Kruskal de complexidade $O(|V|^2 \log_2 |V|)$ para encontrar as arestas da árvore. Portanto a complexidade é $O(n|V|^2 + |V|^2 \log_2 |V|)$

O leitor poderá consultar Behsaz e Rahmati (2006) para verificar as demostrações da consistência e que de fato o algoritmo calcula a máxima verossimilhança para a distribuição de segunda ordem P_t

Capítulo 3

Simulações

3.1 Redes bayesianas e grafo moral

Para evitar a criação manual do grafo para a realização de simulações, resolvemos criar um gerador de campos aleatórios de Markov. A teoria por trás desse gerador envolve as redes bayesianas e o seu grafo moral.

Definição 3. *Seja $G = (V, E)$ um grafo direcionado aciclíco (DAG* - Directed Acyclic Graph*) e $X = \{X_v : v \in V\}$ uma família de variáveis aleatórias indexadas pelo conjunto de vértices V. Dizemos que $\{X_v : v \in V\}$ é uma* rede bayesiana *com relação a G se*

$$\mathbb{P}(X_V = x_V) = \prod_{v \in V} \mathbb{P}(X_v = x_v | X_{pa(v)} = x_{pa(v)})$$

onde $pa(v)$ é o conjunto de vértices pais de v, isto é se $w \in pa(v)$ então $(w, v) \in E$.

Exemplo. Considere a rede bayesiana na Figura 3.1, que tem vértices nomeados de A, B, C, D e E:

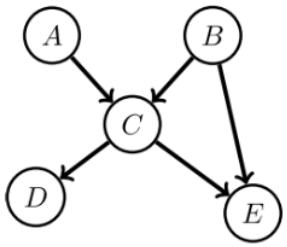

Figura 3.1: *Exemplo de uma rede bayesiana com vértices A, B, C, D e E. Disponível em https://cs-cheatsheet.readthedocs.io/en/latest/subjects/bayesian/bayesian_network.htm. Acessado em 25 de setembro de 2018.*

Seguindo a definição de rede bayesiana, a distribuição conjunta de X_A, X_B, X_C, X_D, X_E pode ser fatorada como

$$\mathbb{P}(X_A, X_B, X_C, X_D, X_E) = \mathbb{P}(X_A)\mathbb{P}(X_B)\mathbb{P}(X_C|X_A, X_B)\mathbb{P}(X_D|X_C)\mathbb{P}(X_E|X_C, X_B)$$

Agora construímos o campo aleatório de Markov a partir de redes bayesianas utilizando o conceito de *grafo moral* de um DAG.

Definição 4. *Seja $G = (V, E)$ um DAG. A partir dela, formamos um grafo não-direcionado $\mathcal{M}_G$ da seguinte forma*

1. *Para todo arco $(v, w) \in E$, (v, w) será uma aresta de $\mathcal{M}_G$.*
2. *Para todo $w \in pa(v)$ e $u \in pa(v)$, $w \neq v$, haverá uma aresta (w, u) em $\mathcal{M}_G$.*

$\mathcal{M}_G$ será chamado de grafo moral *de G.*

Teorema 3. *Sejam $G = (V, E)$ um DAG, $\mathcal{M}_G$ seu grafo moral e $X = \{X_v : v \in V\}$ uma rede bayesiana em relação à G. Então $X = \{X_v : v \in V\}$ será um campo aleatório de Markov em relação à $\mathcal{M}_G$*

Demonstração. Temos que mostrar que $\mathcal{M}_G$ consegue representar um subconjunto das independências condicionais de G. A prova detalhada se encontra em Koller *et al.* (2009) □

3.2 Gerador de distribuição condicional e de amostras

A partir de um DAG gerado aleatoriamente, temos uma distribuição conjunta fatorada em produtos de condicionais. Agora é necessário de alguma forma gerar a distribuição de probabilidade condicional para X_v dado X_W. Lembremos que a distribuição de probabilidade condional tem as mesmas propriedades da distribuição de probabilidadade usual, isto é

$$\mathbb{P}(X_v = a | X_W = a_w) \geq 0 \text{ e}$$

$$\sum_{a \in A} \mathbb{P}(X_v = a | X_W = a_W) = 1, \text{ para qualquer } a_W \in A^W.$$

Uma forma simples de gerar a distribuição condicional no computador é utilizar a distribuição de Dirichlet. A distribuição de Dirichlet é uma multivariada cujos parâmetros são um inteiro k maior que 1 e um vetor de termos positivos de tamanho k. A Dirichlet devolve um vetor de tamanho k cuja soma dos termos é 1. No nosso trabalho, utilizamos o vetor de pesos unitário, isto é, qualquer vetor é equiprovável.

3.3 Resultados

Para este trabalho, geramos diversos grafos, com diferentes números de vértices, para testar a convegência dos algoritmos estudados. Utilizamos as medidas de erro subestimado, erro sobrestimador e erro total que encontramos em Frondana (2016). Seja $G = (V, E)$ um grafo não-direcionado e $G = (V, \hat{E})$ seu grafo estimado por algum dos algoritmo. O erro de subestimação é definido por

$$esub = \frac{\sum_{v \in V} \sum_{v < w} \mathbf{1}\{(v, w) \in E \text{ e } (w, v) \notin \widehat{E}\}}{\sum_{v \in V} \sum_{v < w} \mathbf{1}\{(v, w) \in E\}}$$

O erro de subestimação pode ser entendido como o percentual de arestas que existem em G mas acabaram não sendo estimadas por $\widehat{G}$. Erro de sobrestimação é dada por

$$esob = \frac{\sum_{v \in V} \sum_{v<w} \mathbf{1}\{(v,w) \notin E \text{ e } (v,w) \in \widehat{E}\}}{\sum_{v \in V} \sum_{v<w} \mathbf{1}\{(v,w) \notin E\}}$$

O erro de subestimação pode ser entendido como o percentual de arestas que foram estimados em $\widehat{G}$ mas que não existem em G. O erro total é definido por

$$et = \frac{\sum_{v \in V} \sum_{v<w} (\mathbf{1}\{(v,w) \in E \text{ e } (v,w) \notin \widehat{E}\} + \mathbf{1}\{(v,w) \notin E \text{ e } (v,w) \in \widehat{E}\})}{\frac{|V|(|V|-1)}{2}}$$

Em nosso repositório [1], o leitor poderá acessar as informações de cada simulação: os grafos reconstruídos, a distribuição de probabilidade utilizada, a DAG gerada para construir a campo aleatório de Markov e outras informações. O código-fonte dos programas e instrução de como compilar e executar também está presente no repositório.

3.3.1 Avaliação do algoritmo de máxima verossimilhança penalizada

Primeiramente consideramos grafos pequenos, de até 6 vértices. Como padrão utilizamos o valor de regularização c = 1. Nesta parte, testamos para amostras de tamanho 10 até 5000, com intervalos de tamanho 50, fazendo 10 testes para cada tamanho de amostra e utilizando o alfabeto $A = \{0, 1, 2\}$. Com esses testes fazemos a média dos erros de subestimação, sobrestimação e total. Primeiro exibimos o grafo e logo abaixo sua respectiva simulação.

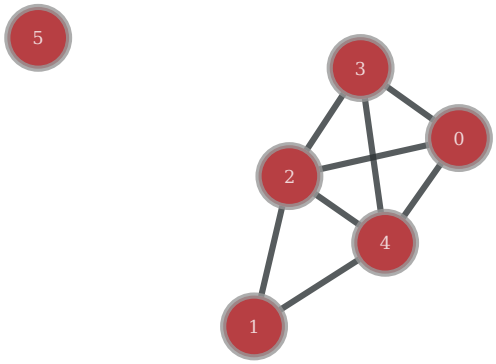

Figura 3.2: *Um grafo de 6 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.3*

[1] https://gitlab.com/rodrigorsdc/tcc

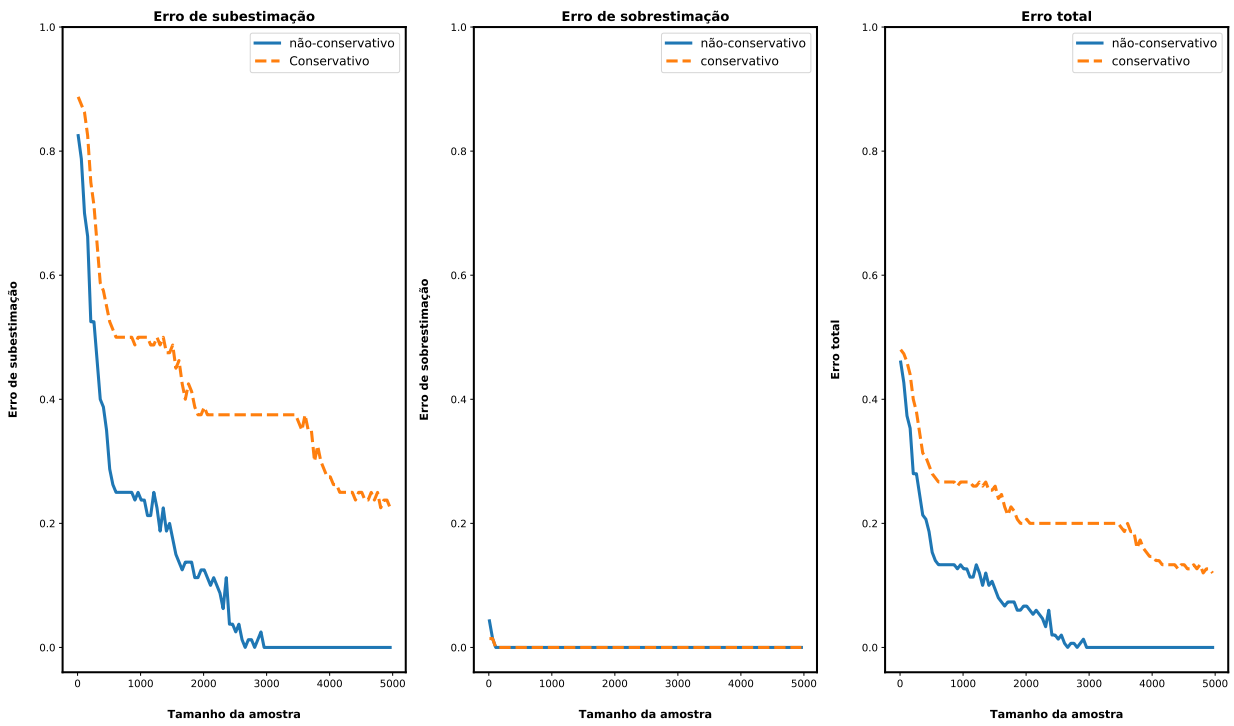

Figura 3.3: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.2 com tamanho de amostra de 10 até 5000, com intervalos de tamanho 50, e fazendo 10 testes para cada tamanho de amostra, com valor de regularização* $c = 1$.

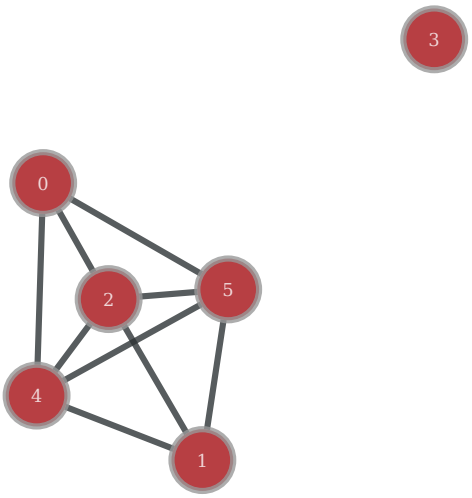

Figura 3.4: *Um grafo de 6 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.5.*

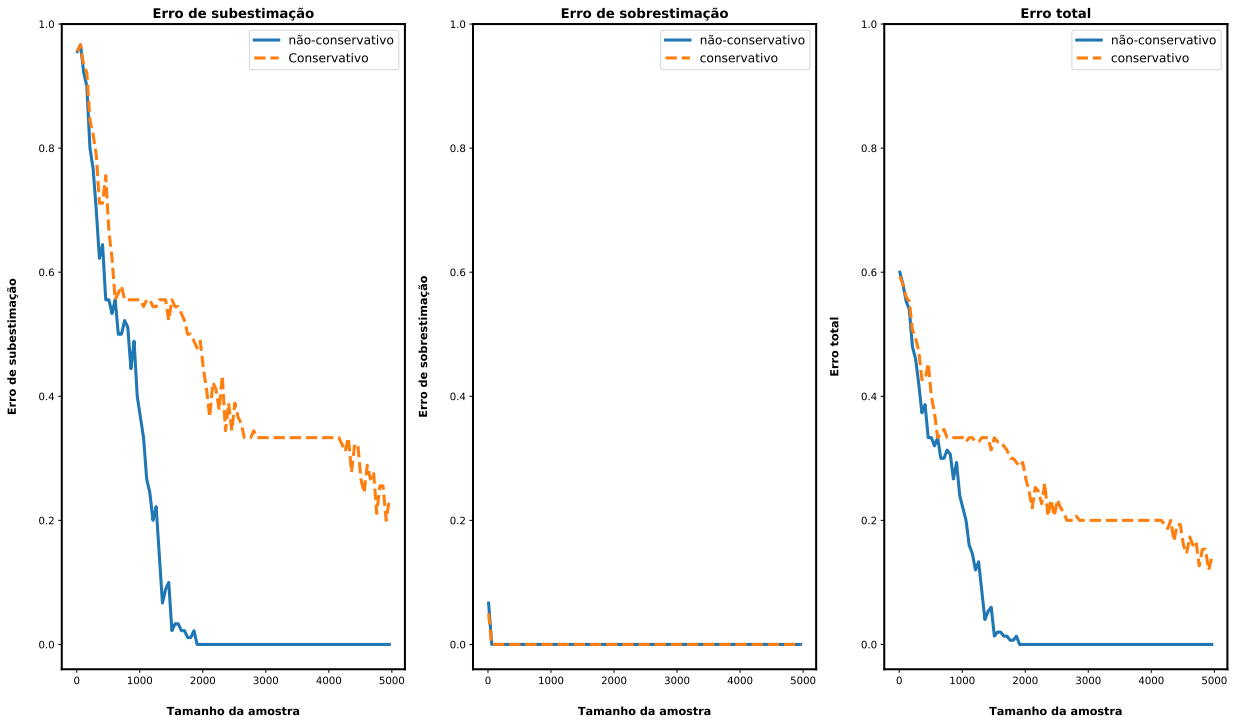

Figura 3.5: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.4 com tamanho de amostra de 10 até 5000, com intervalos de tamanho 50, e fazendo 10 testes para cada tamanho de amostras, com valor de regularização* $c = 1$.

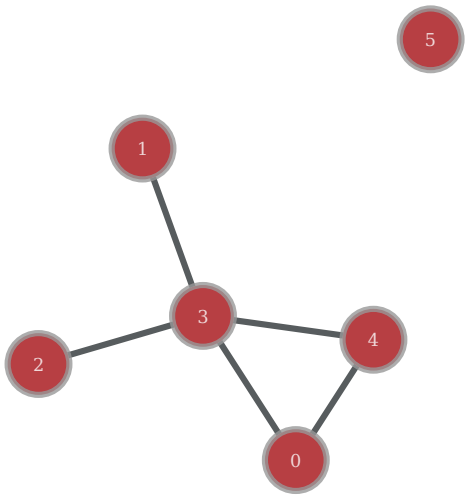

Figura 3.6: *Um grafo de 6 vértices. A avaliação erros desta simulação encontra-se na Figura 3.7.*

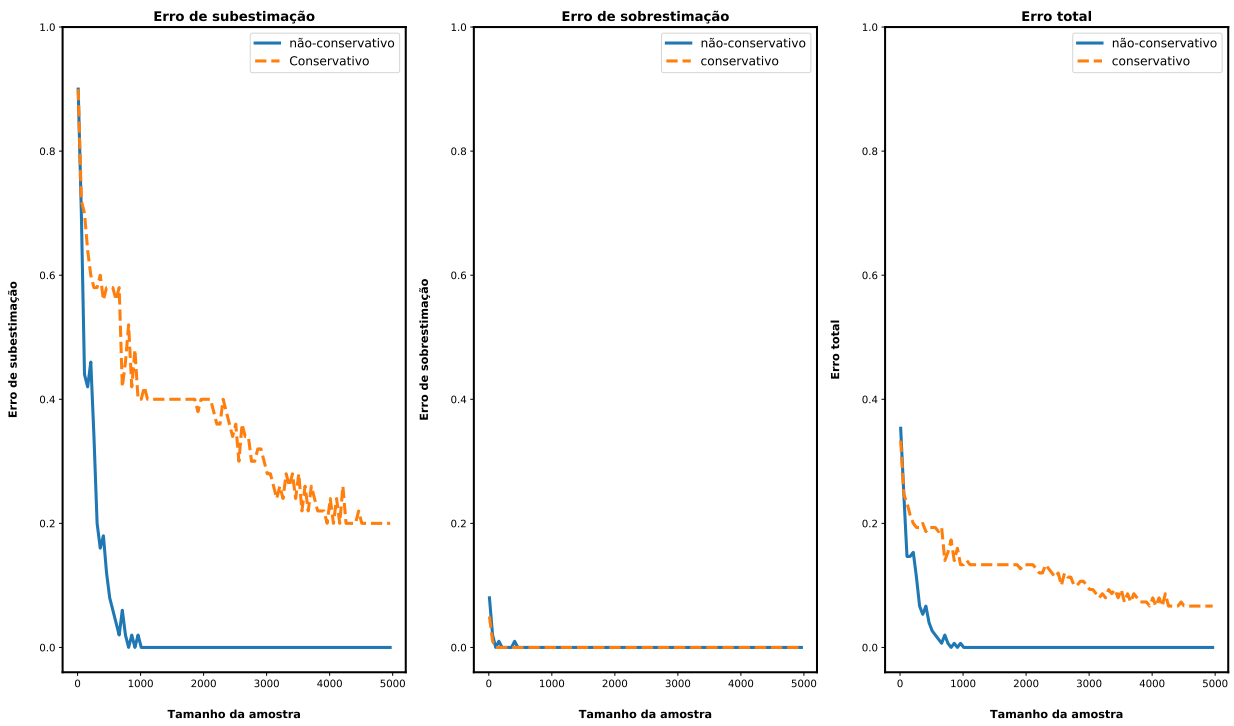

Figura 3.7: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.6 com tamanho de amostra de 10 até 5000, com intervalos de tamanho 50, e fazendo 10 testes para cada tamanho de amostra, com valor de regularização* $c = 1$.

Nas simulações notamos a superioridade do método não-conservativo como já percebido no trabalho de Frondana (2016). Vamos agora apresentar 4 simulações de grafos com 10 vértices. Da mesma forma das simulações de 6 vértices, usamos o regularizador $c = 1$, com alfabeto $A = \{0, 1, 2\}$, amostras de tamanho indo de 10 até 3500, com acréscimos de tamanho 50 e fazendo 10 simulações para cada tamanho de amostra e calculando a média de erros. Para viabilizar as simulações em termos de tempo, tivemos que limitar o grau máximo para 6.

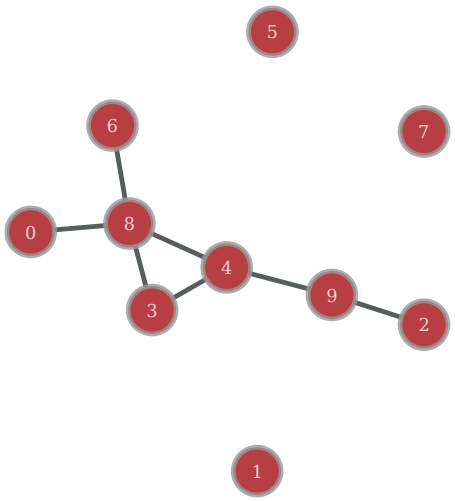

Figura 3.8: *Um grafo de 10 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.9.*

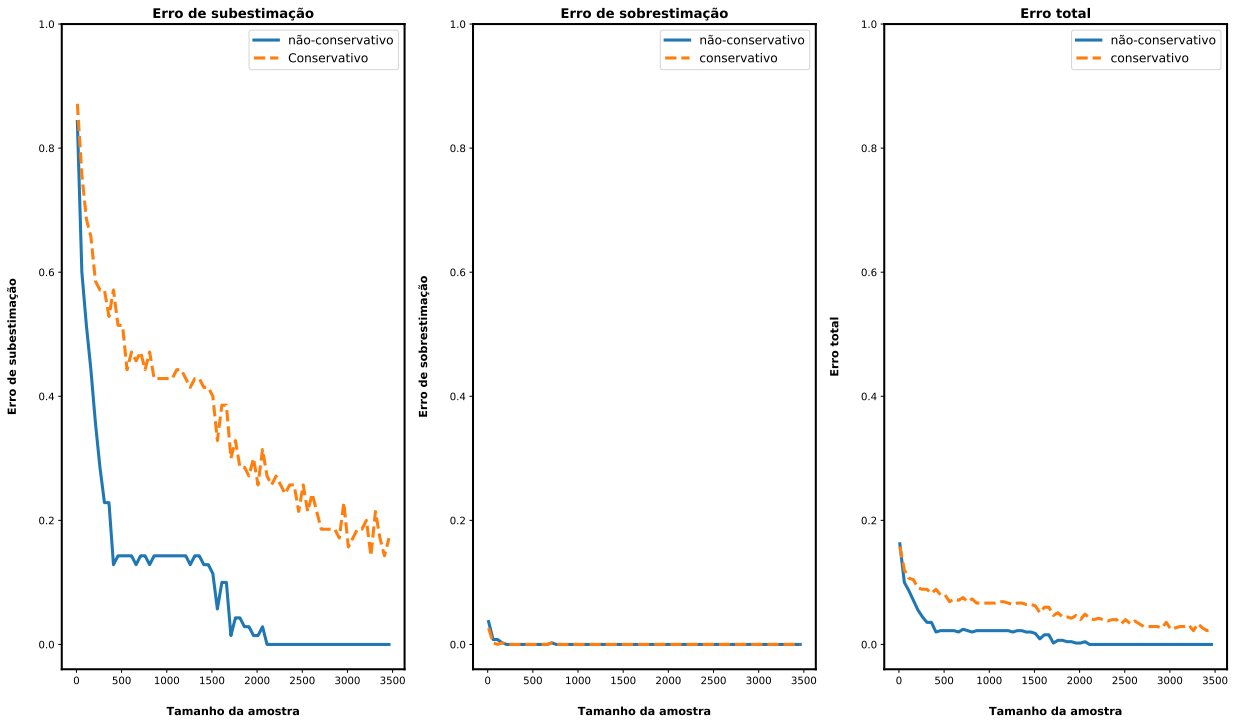

Figura 3.9: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.8 com tamanho de amostra de 10 até 3500, com intervalos de tamanho 50, e fazendo 10 testes para cada tamanho de amostra, com valor de regulização* $c = 1$

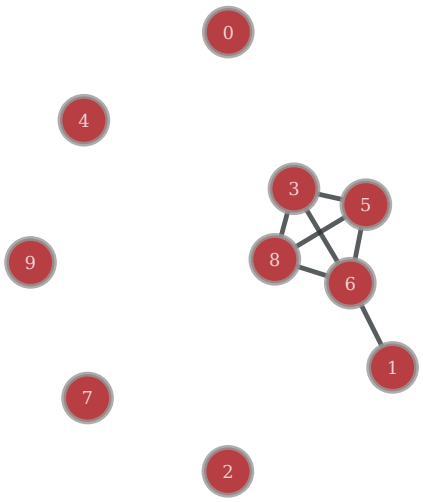

Figura 3.10: *Um grafo de 10 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.11.*

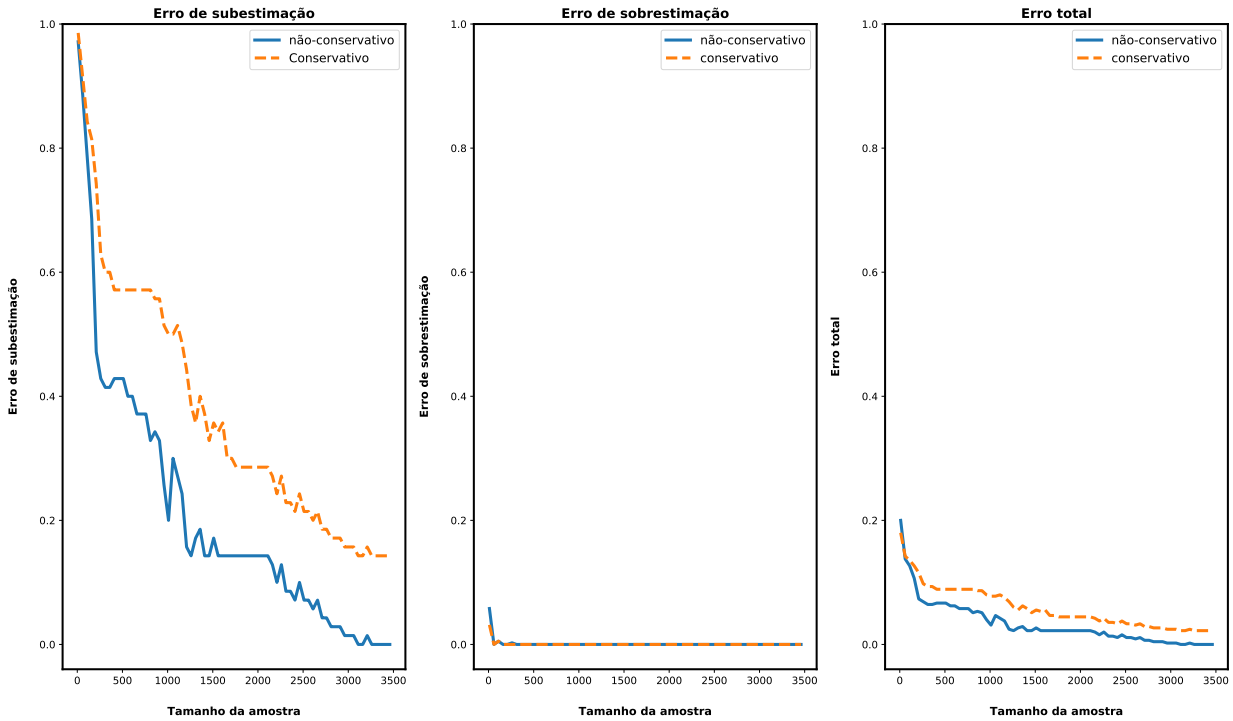

Figura 3.11: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.10 com tamanho de amostra de 10 até 5000, com intervalos de tamanho 50, e fazendo 10 testes para cada tamanho de amostra, com valor de regulização* $c = 1$

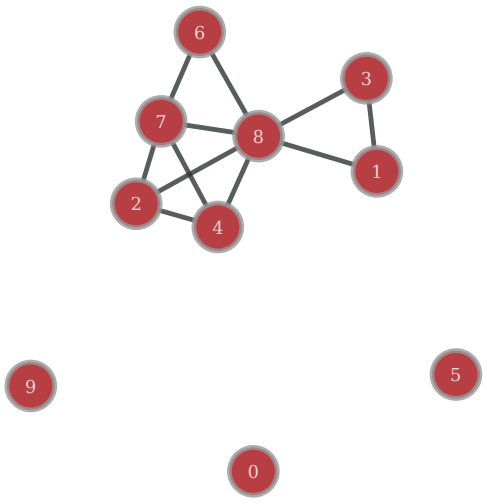

Figura 3.12: *Um grafo de 10 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.13.*

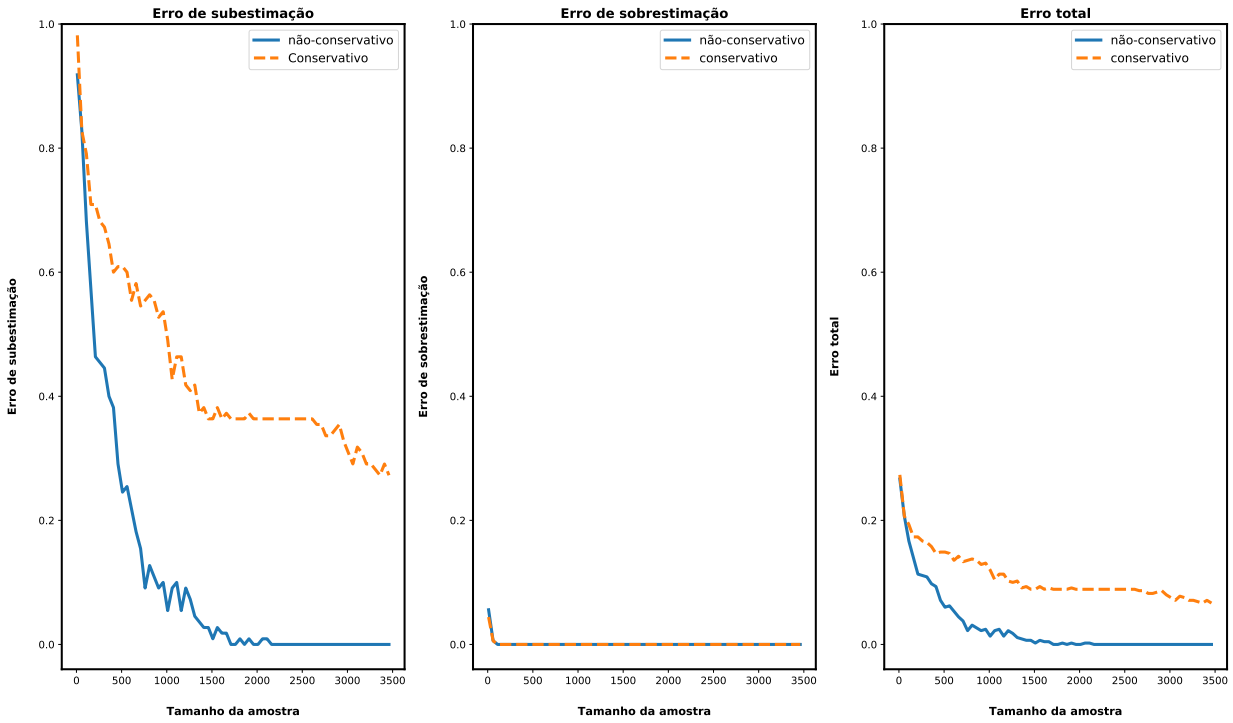

Figura 3.13: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.12 com tamanho de amostra de 10 até 5000, com intervalos de tamanho 50, e fazendo 10 testes para cada tamanho de amostra, com valor de regulização* $c = 1$

Na Figura 3.15 temos uma simulação de um campo aleatório de Markov com 25 vértices (Figura 3.14), com grau máximo 2, utilizando regularizador $c = 1$, com amostras de tamanho 10 até 5000 e alfabeto $A = \{0, 1, 2\}$.

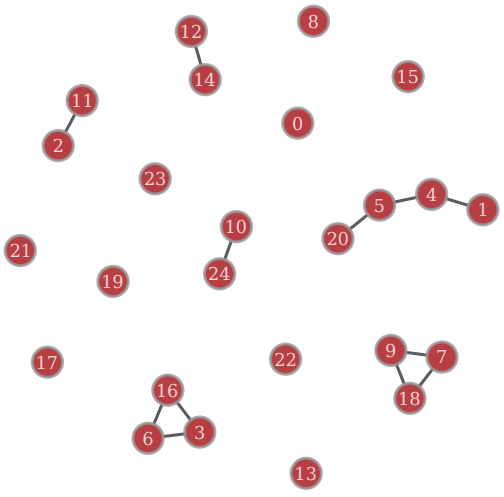

Figura 3.14: *Um grafo de 25 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.15.*

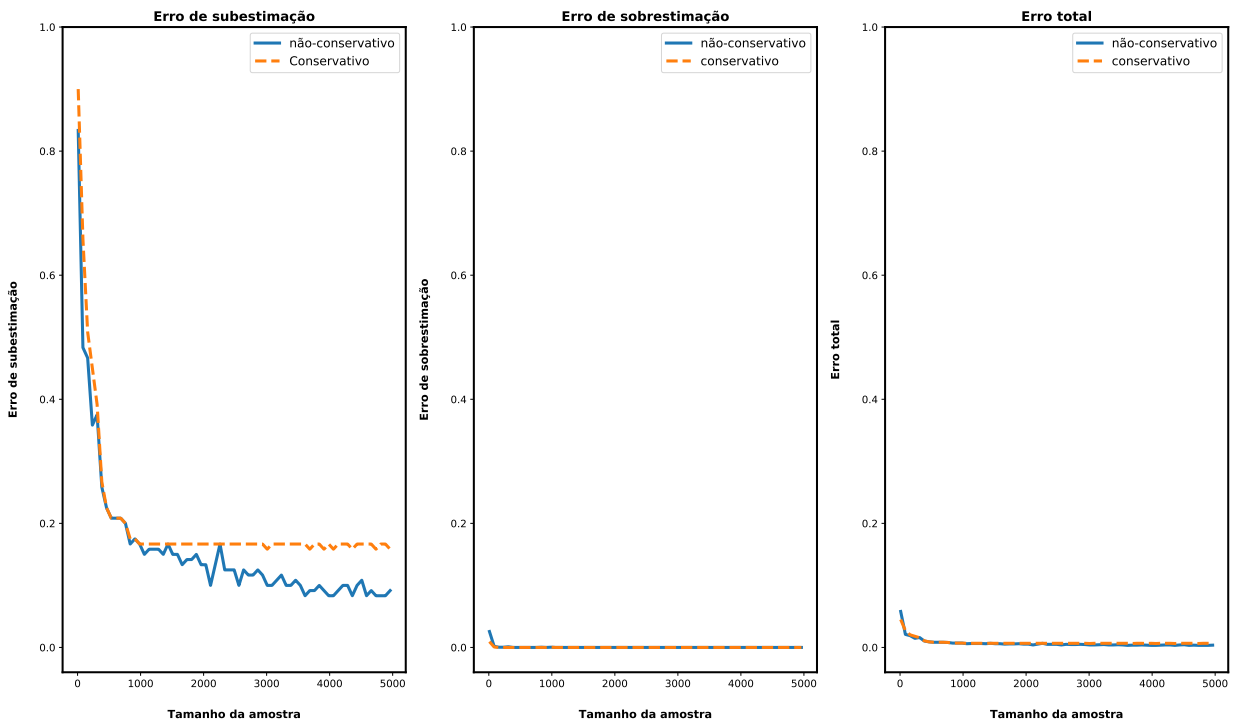

Figura 3.15: *Erros de subestimação, sobrestimação e total do grafo da Figura 3.14.*

Diferentemente das simulações com grafos de 6 e 10 vértices, o grafo de 25 vértices não consegue reconstruir corretamente o grafo com amostra de tamanho 5000.

Vamos agora testar em altas dimensões. A Figura 3.16 mostra uma simulação de um grafo de 500 vértices com grau máximo 1, utilizando regularizador $c = 1$, com amostras de tamanho 5 até 500 com invervalos de tamanho 5 e com alfabeto $A = \{0, 1\}$. A Figura 3.17 repete a simulação anterior, com a diferença que o regularizador adotado foi de $c = 0.01$, esperando obter um resultado melhor visto que o grafo é extremamente esparso.

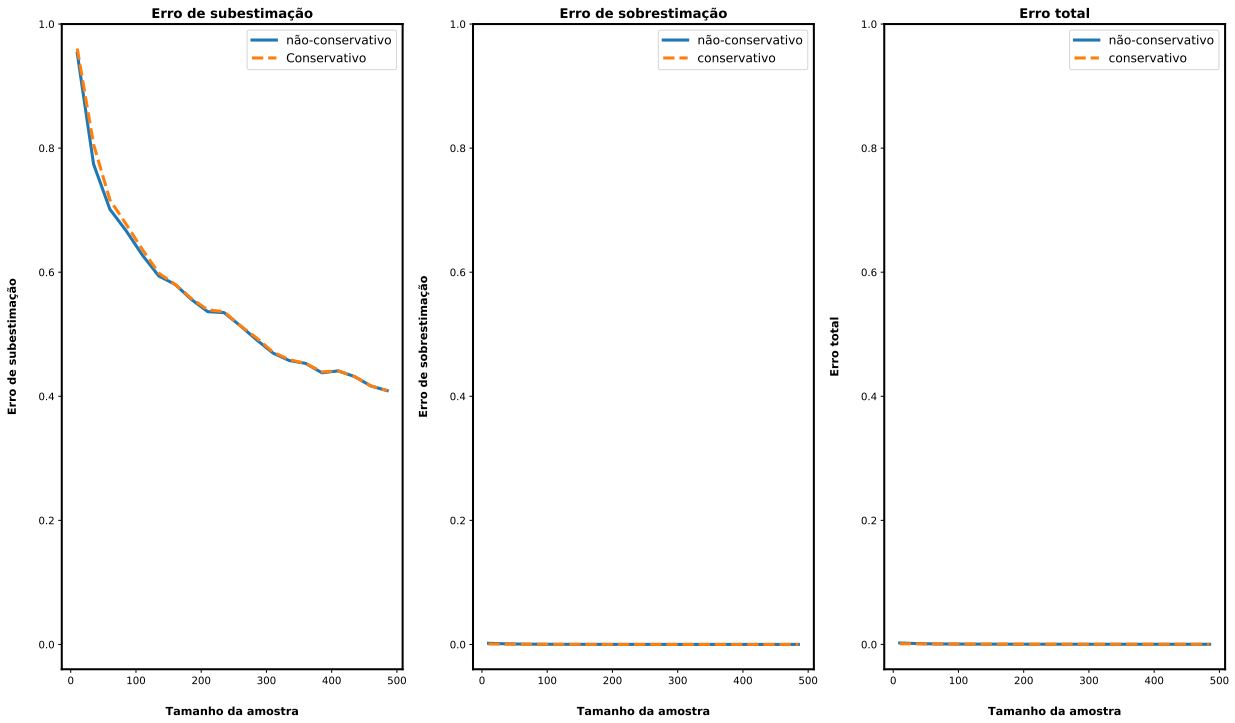

Figura 3.16: *Erros de subestimação, sobrestimação e total, com amostras de tamanho até 500, de um grafo de 500 vértices com grau máximo 1, usando regulizador* $c = 1$.

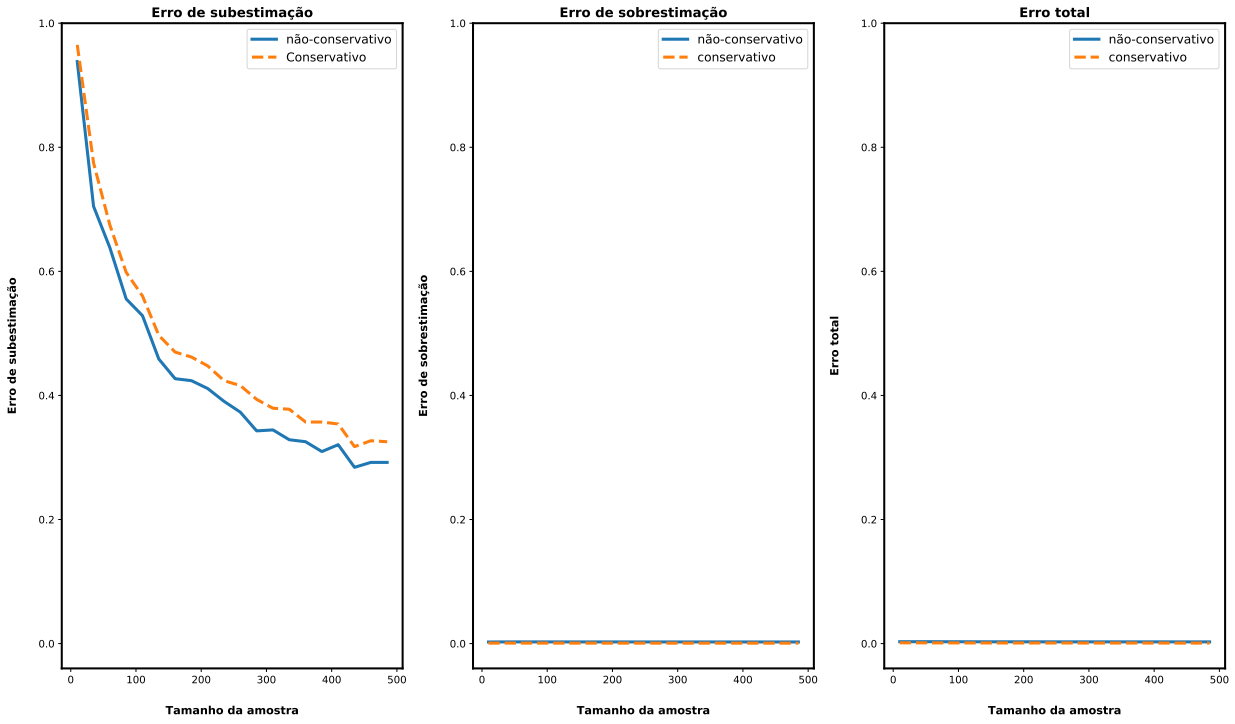

Figura 3.17: *Erros de subestimação, sobrestimação e total, com amostras de tamanho até 500, de um grafo de 500 vértices, com grau máximo 1, usando regularizador* $c = 0.01$

Não obtivemos bons resultados com amostras de tamanho até 500, porém com o regularizador $c = 0.01$, o erro teve um ligeira queda. Agora vamos simular para tamanho de amostra de 10000. A Figura 3.18 mostra esse teste, que foi feito com regularizador $c = 1$.

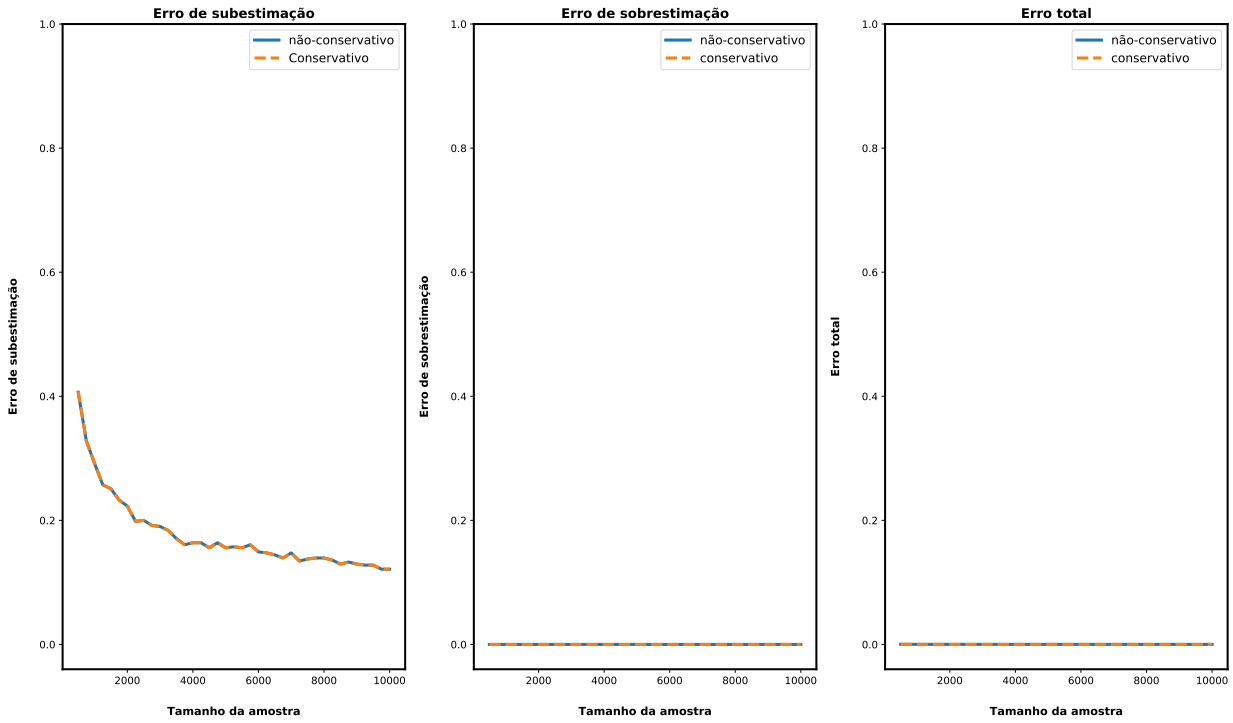

Figura 3.18: *Erros de subestimação, sobrestimação e total, com amostras de tamanho até 10000, de um grafo de 500 vértices, com grau máximo 1, usando regularizador* $c = 1$.

3.3.2 Avaliação do algoritmo de Chow-Liu

O algoritmo de Chow-Liu apenas devolve uma árvore, por isso tivemos que desenvolver um mecanismo para gerar DAG cujo grafo moral seja uma árvore, ou seja, que tenha $n - 1$ arestas, onde n é o número de vértices, e seja acíclico. Para gerar os campos aleatórios de Markov de tipo árvore, utilizamos o seguinte procedimento:

1. Tomamos um grafo direcionado completo $G = (V, E)$, ou seja, para todos vértices $v, w \in V$ e $v \neq w$, temos que $(v, w) \in E$.

2. A partir do grafo G, geramos uma arborecência, isto é, existe apenas um vértice que tem grau de entrada 0 e nenhum vértice desse grafo possui grau de entrada maior do que 1.

3. Com o grafo moral dessa arborecência, temos o campo aleatório que precisamos.

As simulações, com árvores de 10, 25 e 100 vértices, foram feitas com amostras de tamanho de 10 até 5000, com intervalos de tamanho 50 e utlizando alfabeto $A = \{0, 1, 2\}$.

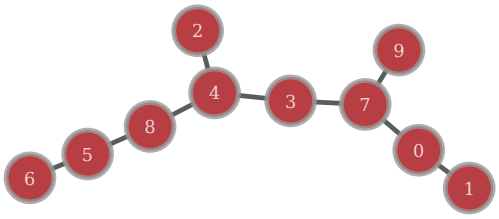

Figura 3.19: *Uma árvore de 10 vértice. A avaliação dos erros desta simulação encontra-se na Figura 3.20.*

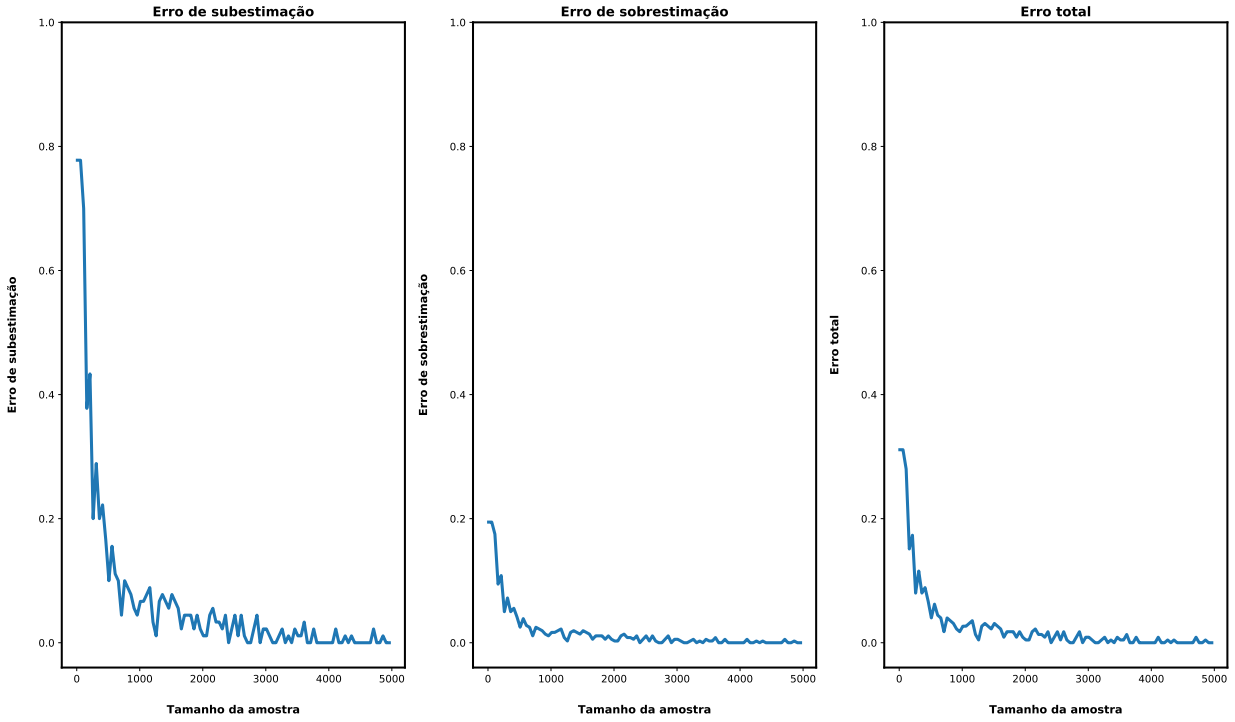

Figura 3.20: *Erros de subestimação, sobrestimação, e total para diferentes tamanhos de amostra da árvore da Figura 3.19.*

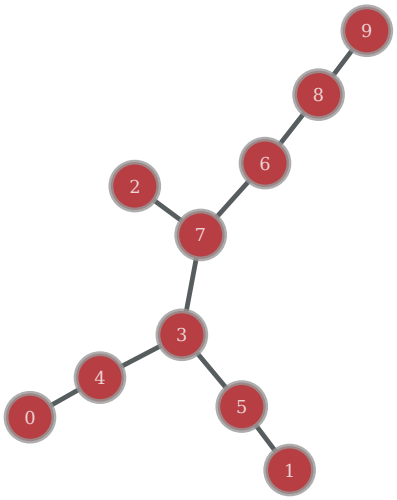

Figura 3.21: *Uma árvore de 10 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.22.*

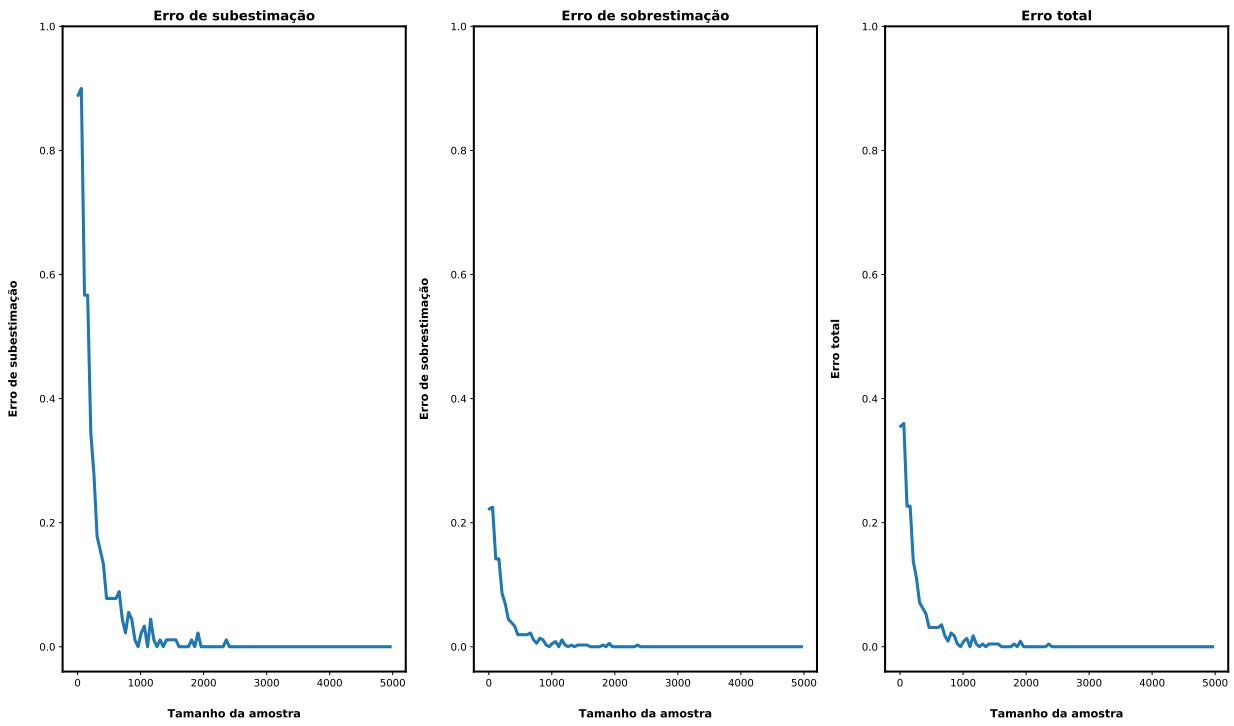

Figura 3.22: *Erros de subestimação, sobrestimação e total de diferentes tamanhos de amostra da árvore da Figura. 3.21*

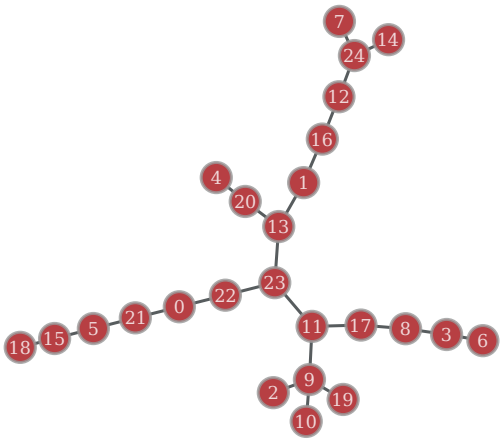

Figura 3.23: *Uma árvore de 25 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.24.*

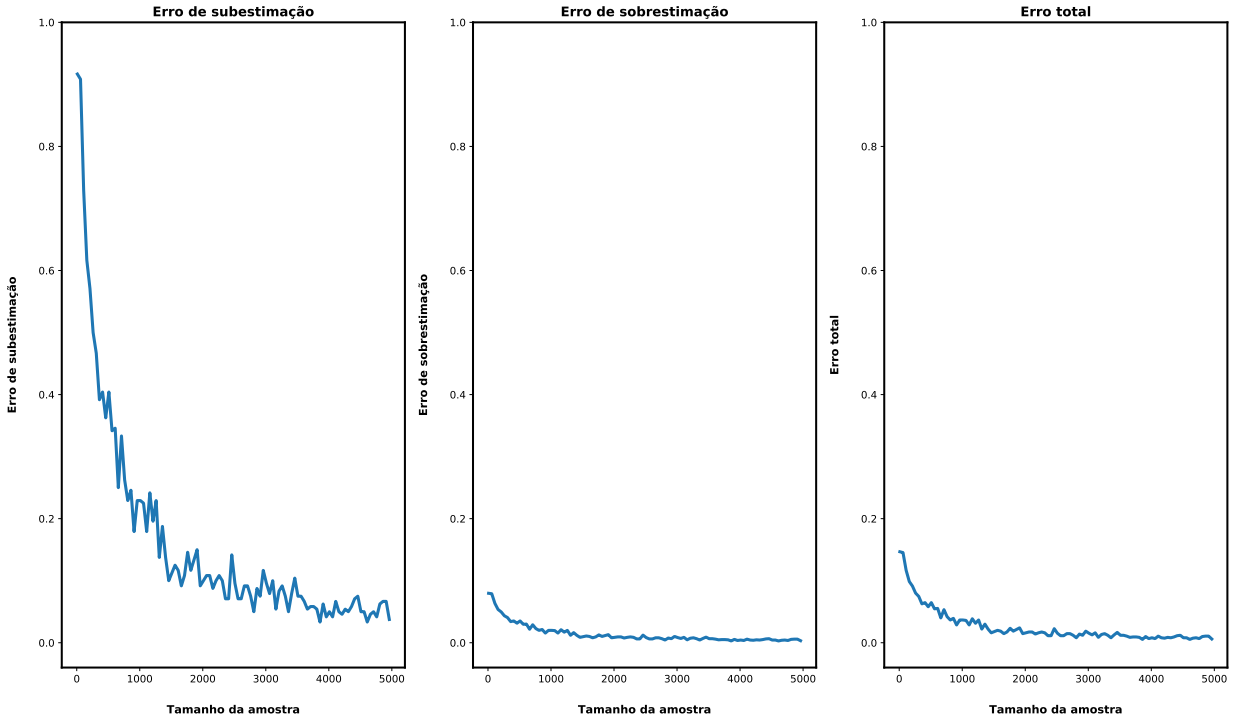

Figura 3.24: *Erros de subestimação, sobrestimação e total de diferentes tamanhos de amostra da árvore da Figura 3.23.*

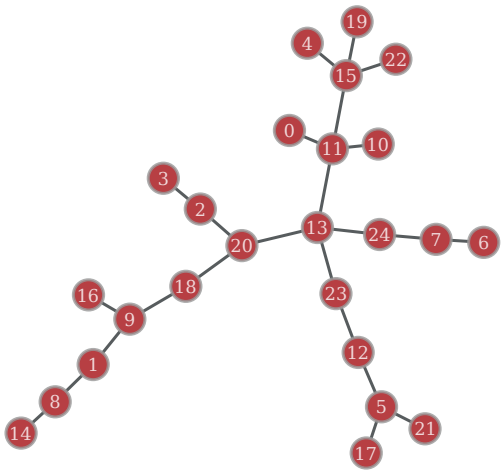

Figura 3.25: *Uma árvore de 25 vértices. A avaliação dos erros desta simulação encontra-se na Figura 3.26.*

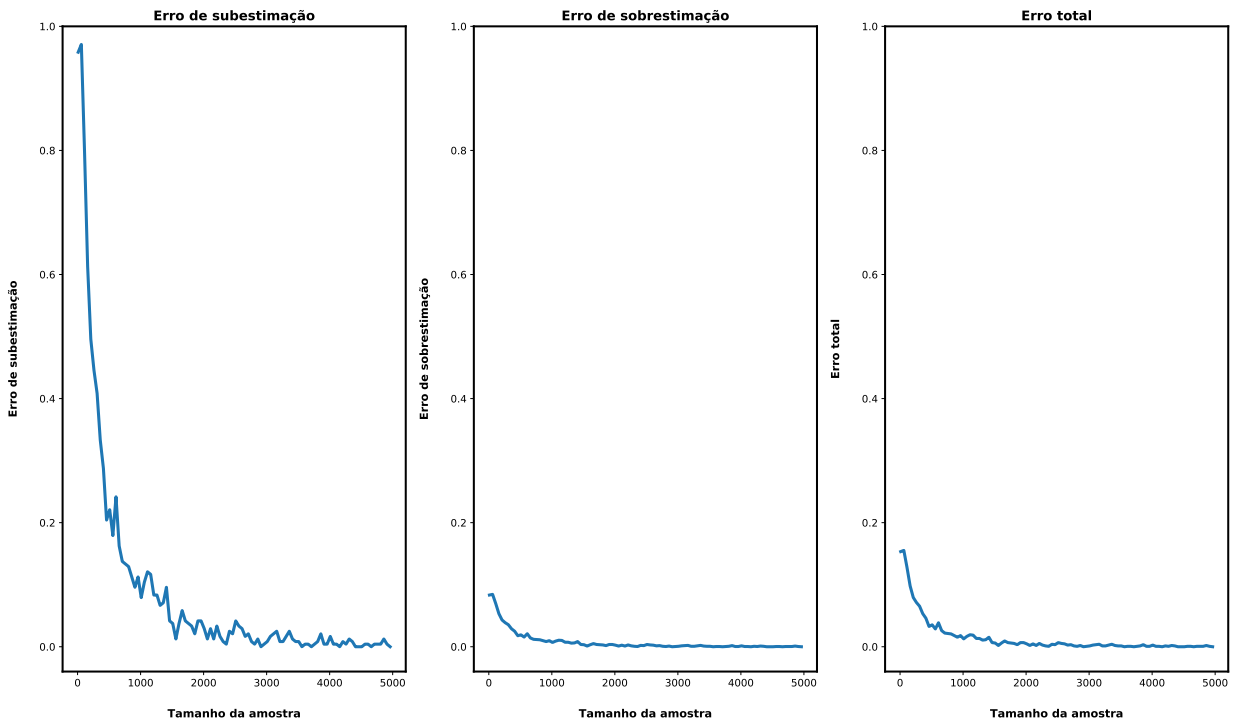

Figura 3.26: *Erros de subestimação, sobrestimação e total de diferentes tamanhos de amostra da árvore da Figura 3.25.*

Na Figura 3.27, temos a simulação de uma árvore de 100 vértices. Por uma questão de espaço, não podemos colocar a imagem desse grafo aqui no texto, mas o leitor poderá consultá-lo no repositório do trabalho.

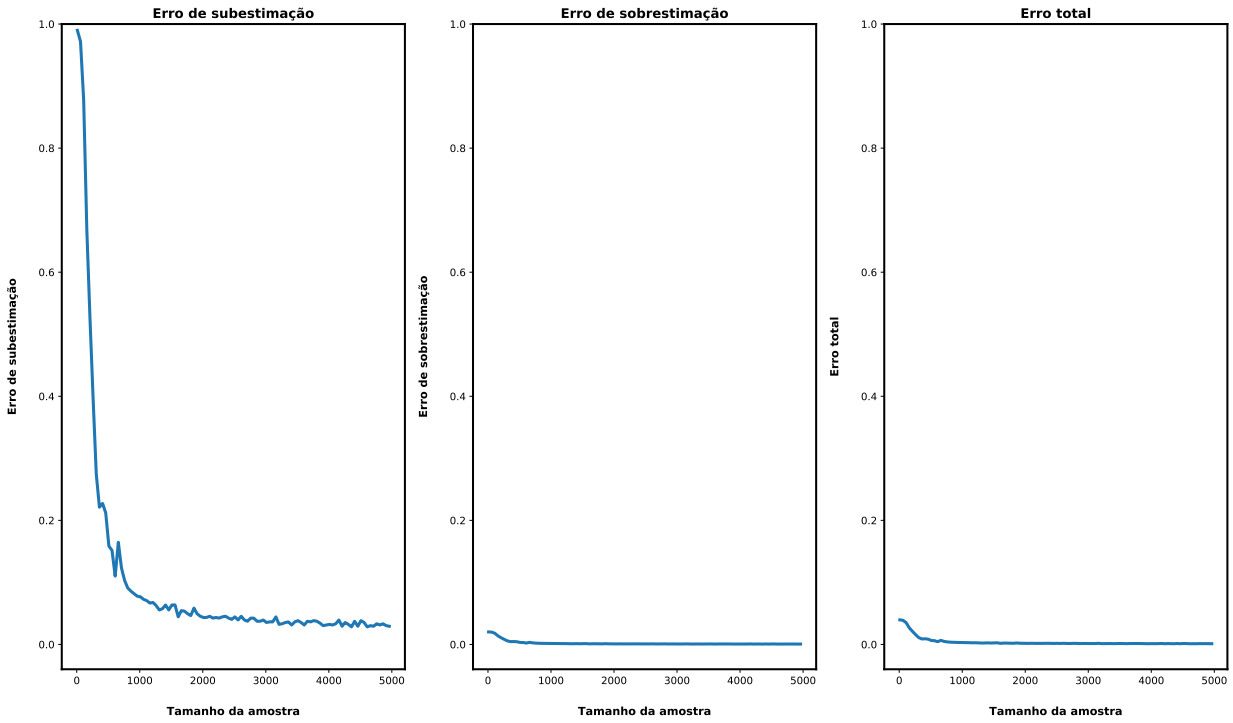

Figura 3.27: *Erross de subestimação, sobrestimação e total de diferentes tamanhos de amostras de uma árvore de 100 vértices.*

As simulações revelaram que a quantidade de vértices não alteram de modo significativo a velocidade de convergência; todas as simulações apresentam comportamento semelhante. Em nossos testes, tamanho de amostra 1000 já reconstrói a árvore com taxa de erro menor do que 0.15.

Capítulo 4

Aplicações

4.1 Validação cruzada

O estimador do algoritmo de máxima verossimilhança penalizada converge *quase-certamente* para o grafo correto quando o tamanho de amostras tende ao *infinito*, para qualquer valor de regularizador $c > 0$, porém em aplicações do mundo real não dispomos dessa quantidade infinita de observações. Em duas simulações desse algoritmo foi constatado que há diferença quando se utilizam valores de regularizador diferentes. Um bom método para encontrar um regularizador se chama *validação cruzada*.

A validação cruzada é um procedimento para estimar o *erro de generalização* de um modelo treinado e consiste em dividir a amostra em um conjunto de *treinamento* e um conjunto de *validação*. O método avalia qual(is) o(s) parâmetro(s) do modelo, que no nosso caso é o regularizador c, consegue(m) o melhor resultado (Hastie *et al.*, 2009).

O tipo de validação cruzada que utilizaremos nesse trabalho será o k-fold. Nessa aborgagem, a amostra é dividida em k subconjuntos disjuntos de tamanhos parecidos. O processo é divido em k etapas: na i-ésima etapa, é usado o i-ésimo subconjunto para a validação e os $k-1$ subconjuntos restantes para o treinamento; ao fim das k etapas, temos k medidas de estimação e fazemos a média para obter o valor da validação cruzada. Seja c o parâmetro do modelo, a fórmula da validação cruzada é

$$CV(c)_k = \frac{1}{k}\sum_{i=1}^{k} E_i(c)$$

onde $E_i(c)$ é a medida de erro calculado no subconjunto i, utilizando o modelo treinado com os $k-1$ subconjuntos restantes.

Em abordagens clássicas como em regressão linear a medida usual é o erro quadrático médio. Para o algoritmo de máxima verossimilhança penalizada utilizamos o log da verossimilhança condicional; assim a medida de validação cruzada da i-ésima etapa para o regulizador c é

$$\log \mathbb{P}(c)_i = \sum_{v\in V}\sum_{x_{\widehat{ne}(v)}\in A^{\widehat{ne}(v)}}\sum_{a\in A} N_i(a, x_{\widehat{ne}(v)}) \log(\alpha p + (1-\alpha)\hat{p}(a|x_{\hat{ne}(v)}))$$

onde a vizinhança $\widehat{ne}(v)$ e a probabilidade condicional $\hat{p}(a|x_{\widehat{ne}(v)})$ são calculadas usando os $k-1$ subconjuntos para treinar o modelo, a contagem $N_i(a, x_{\widehat{ne}(v)})$ é obtida do subconjunto

i para validar o modelo e α e p positivos bem próximos de 0. $\alpha p + (a - \alpha)\hat{p}(a|x_{\widehat{ne}(v)})$ é uma mistura para evitar que a validação cruzada adquira o valor $-\infty$. Assim a medida de validação cruzada para o parâmetro c fica

$$CV(c)_k = \frac{1}{k}\sum_{i=1}^{k} \log \mathbb{P}(c)_i.$$

4.2 Índice de bolsa de valores

O trabalho de Frondana (2016) apresenta uma aplicação que utiliza dados de índices de bolsas de valores de Brasil, Espanha, Alemanha, Japão e Austrália. O objetivo é estimar um grafo que expressa as dependências entre as bolsas de valores. Neste trabalho, escolhemos alguns países diferentes: Brasil, EUA, Reino Unido, França, Índia e Japão com índices Bovespa, NASDAQ, FTSE 100, CAC 40, Nifty 50 e Nikkei 250, respectivamente. Coletamos dados de 05 de janeiro de 2001 até 22 de outubro de 2018 de um site de investimento financeiro [1]. A Figura 4.1 mostra a serie temporal dos dados de 3822 dias.

Os valores de índices pertencem ao conjunto dos reais, portanto de alguma forma temos que discretizá-los. Na nossa aplicação, criamos um novo conjunto de dados que assume valores no alfabeto $A = \{0, 1\}$ onde o índice d recebe valor 1 se o valor do pregão de fechamento do dia $d + 1$ for maior do que o valor de fechamento do dia d, e 0 caso contrário. Com essa transformação, temos dados de 3821 dias e não 3822.

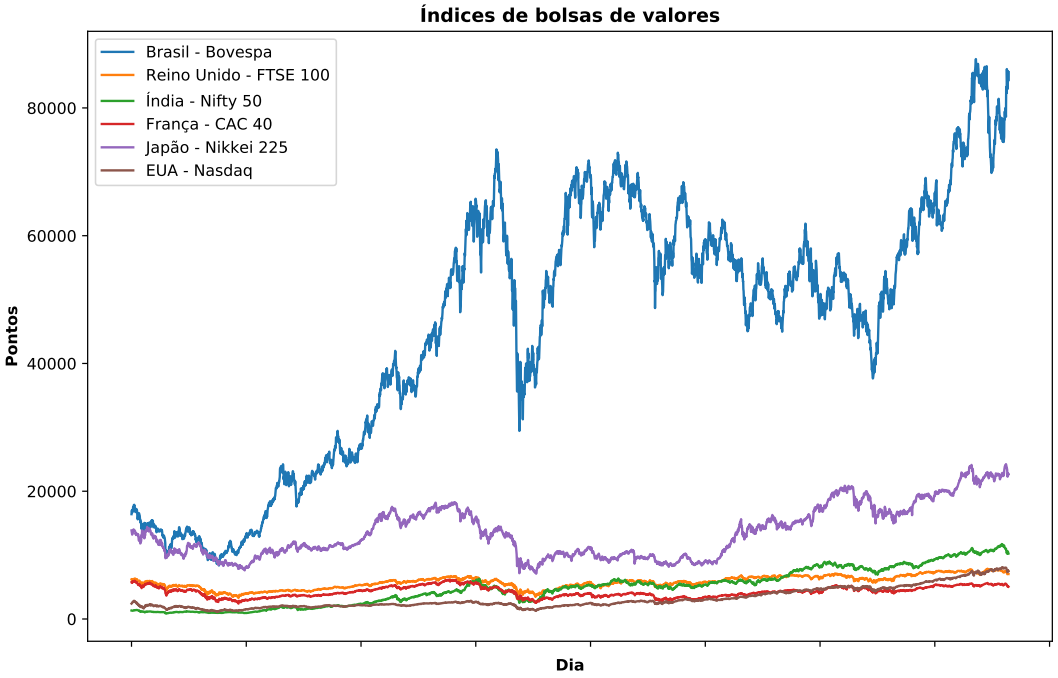

Figura 4.1: *Índices de bolsa de valores da Bovespa, NASDAQ, FTSE 100, CAC 40, Nifty 50 e Nikkei 250 de 05 de janeiro de de 2001 até 22 de outubro de 2018.*

[1] https://br.investing.com/indices/world-indices

O algoritmo de máxima verossimilhança penalizada supõe que a amostra seja independente, que nesse caso seria supor que os índices das bolsas sejam independentes de um dia para o outro. Mas isso pode não ser verdade. Para diminuir a correlação entre os elementos da amostra, selecionamos os dados com intervalos de 5 dias, diminuindo o número de dias utilizados para 764.

Atráves da validação cruzada *10-fold*, o valor ótimo obtido foi de -252.1 com regularizador no intervalo $[0.36, 0.43]$, como mostra a Figura 4.2.

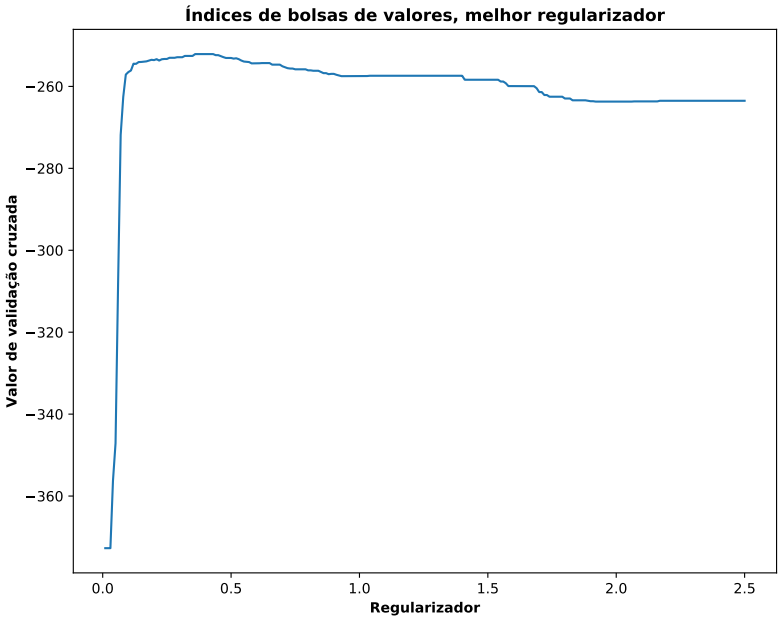

Figura 4.2: *Valores de validação cruzada com fold* $k = 10$. *O maior valor é atingido no intervalo* $[0.36, 0.43]$

Aplicando o algoritmo de máxima verossimilhança penalizada utilizando regularizador $c = 0.36$, reconstruímos o campo aleatório de Markov, mostrado na Figura 4.3. As vizinhanças Markovianas de cada nó foram àqueles que estão mais próxima do ponto de vista geográfico.

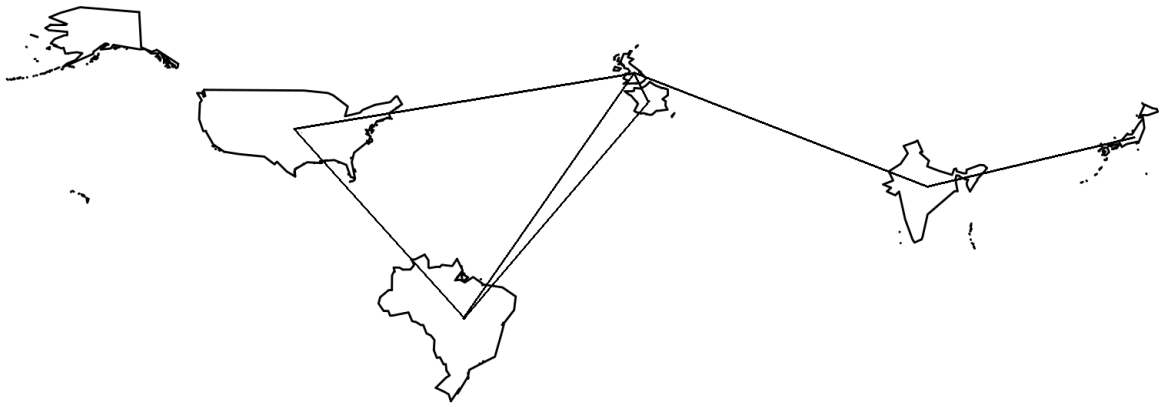

Figura 4.3: *Mapa representando o grafo de dependências entre os países.*

4.3 Microarrays de pacientes com câncer

Uma outra aplicação interessante é usar os dados de microarrays para averiguar se as dependências condicionais entre os genes mudam de um paciente saudável e um paciente com câncer de mama. O conjunto disponibilizado[2] mostra dados de 529 pacientes com câncer de mama e 61 pacientes sem a doença com a expressão gênica de 17.814 genes.

O cálculo de 17.814 vértices em nosso algoritmo de máxima verossimilhança penalizada é totalmente inviabilizado. Um procedimento que adotamos foi selecionar aqueles genes com maior variabilidade. Para discretizar os valores de expressão gênica, selecionamos os genes de cada paciente, normalizamos os valores para a Normal(0, 1) e atribuímos para o alfabeto $A = \{0, 1, 2\}$ da seguinte forma: se o valor normalizado for até -0.415 consideramos que é uma expressão fraca e é valorado com 0, se estiver entre -0.415 até 0.415 consideramos de expressão média e fica com valor 1 e por fim se for maior que 0.415 é considerado um gene bastante expressivo e portanto adquire valor 1.

Para essa aplicação, escolhemos trabalhar com os 8 genes de maior variância no conjunto de dados: EFH, RGS1, CD163, FCGR3A, FABP1, MUCL1, MAGEC2 e CTCFL. E limitamos o grau máximo para 3.

As Figuras 4.4 e 4.5 mostram a validação cruzada para os dados de paciente saudáveis e de pacientes com câncer. Estimando os grafos de pacientes normais e doentes com $c = 1.87$ e $c = 0.91$, respectivamente, obtemos os grafos das figuras 4.6 e 4.7.

[2] https://data.mendeley.com/ datasets/v3cc2p38hb/1

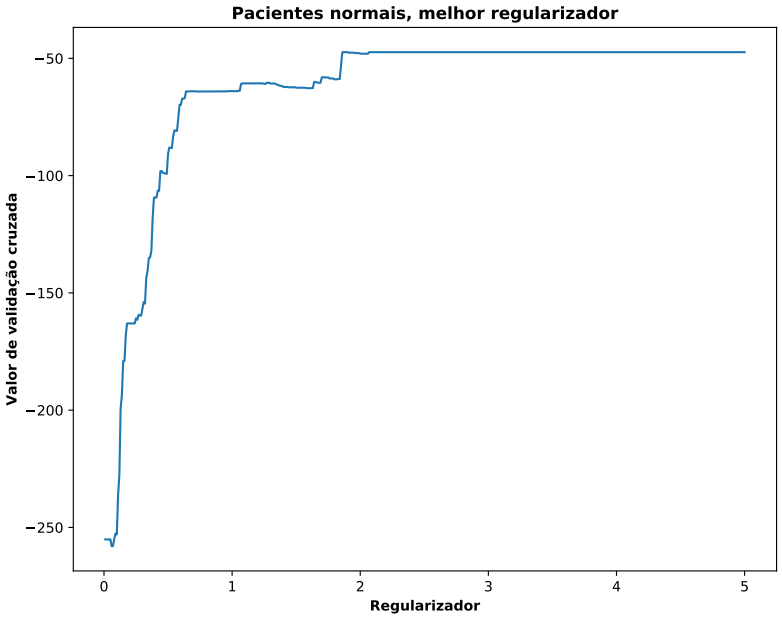

Figura 4.4: *Valores da validação cruzada de 10-fold. Maior valor atinge no intervalo* $[1.87, 1.9]$

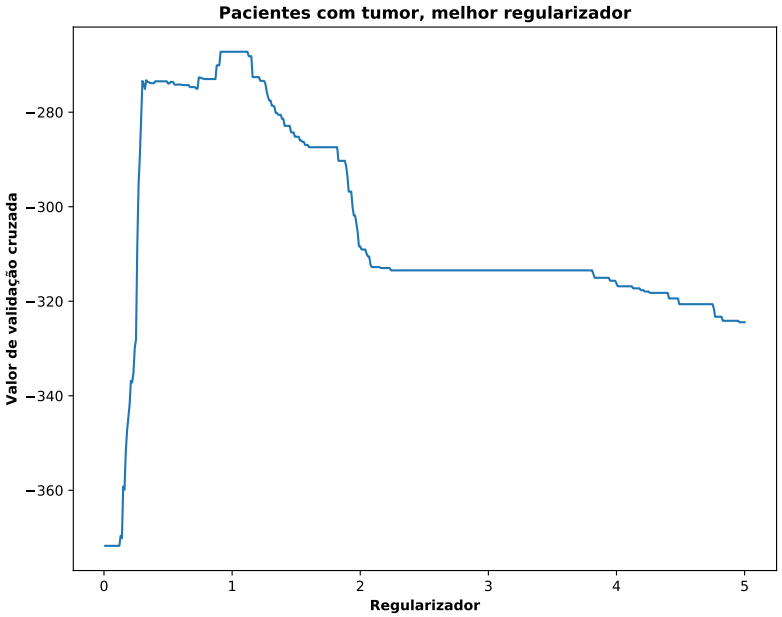

Figura 4.5: *Valores da validação cruzada 10-fold. O maior valor atinge no invervalo* $[0.91, 1.12]$

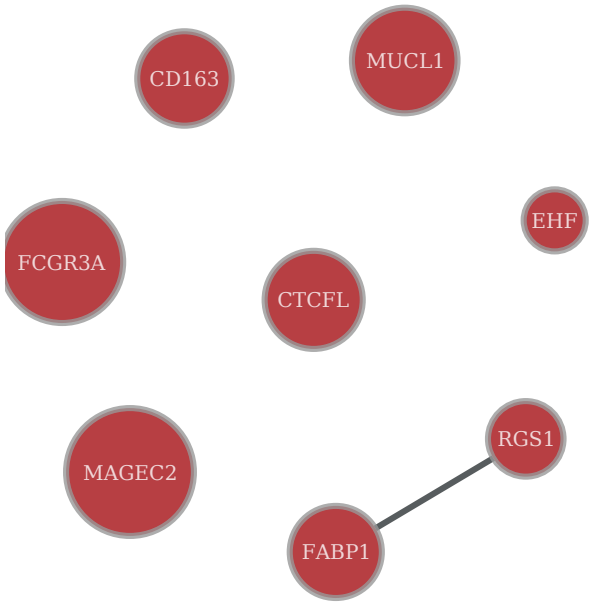

Figura 4.6: *Grafo estimado dos dados de pacientes saudáveis.*

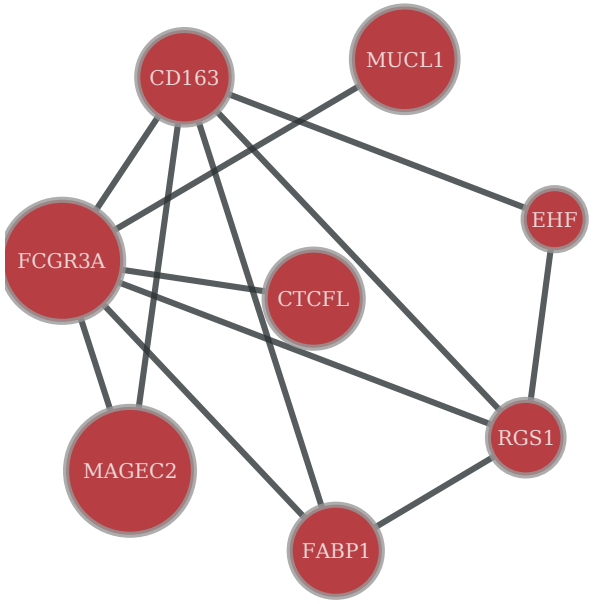

Figura 4.7: *Grafo estimado dos dados de pacientes com câncer de mama.*

Há uma notável diferença entre os grafos estimados. Uma ponderação que devemos ter sobre a aplicação é que há uma substancial diferença do tamanho da amostra entre os pacientes normais e com câncer (69 contra 529), o que deve ter levado ao grafo de pacientes normais a um alto erro de subestimação.

Capítulo 5

Conclusão

Neste trabalho estudamos dois algoritmos para reconstrução de um campo aleatório de Markov, apresentadas no Capítulo 2. O algoritmo de Chow-Liu estima uma distribuição de segunda ordem que possui a menor divergência de Kullback-Leibler em relação à distribuição correta. O estimador proposto por Frondana (2016), pode reconstruir qualquer tipo de grafo.

Em nossas simulações no Capítulo 3, notamos a rápida convergência do algoritmo de Chow-Liu para árvores de 10, 25 e 100 vértices, com praticamente a mesma velocidade de reconstrução correta. Na caso do algoritmo de máxima verossimilhança penalizada, obtemos bons resultados para grafos de 10 vértices. Em situações de alta dimensão, o tamanho de amostra de 5000 não foi suficiente para obtenção do grafo estimado com erro zero. Porém, em uma simulação com um grafo de 500 vértices esparso, obtivemos um erro de subestimação de apenas 0.13 para um tamanho de amostra de 10.000.

Com relação às aplicações apresentadas no Capítulo 4, propusemos uma análise em dados de microarrays de câncer de mama e de índices de bolsas de valores. Os testes mostraram uma diferença significativa entre os grafos de dependências entre os pacientes normais e os pacientes com câncer. Com relação aos dados de bolsa de valores, o grafo estimado de países mostrou uma forte dependência em relação à aproximação geográfica.

Referências Bibliográficas

Behsaz e Rahmati(2006) Babak Behsaz e Mohammad Rahmati. Estimation of probability density function by dependence tree methods for pattern recognition systems. *Tech. Rep. U. Alberta.* Citado na pág. 7, 8

Frondana(2016) Iara Moreira Frondana. *Model selection for discrete Markov random fields on graphs.* Tese de Doutorado, Universidade de São Paulo. Citado na pág. 1, 4, 5, 7, 10, 14, 28, 33

Giraud.(2014) C. Giraud. *Introduction to High-Dimensional Statistics.* Chapman & Hall, 1st edição. Citado na pág. 4

Hastie *et al.***(2009)** T. Hastie, R. Tibshirani e Friedman J. *The Elements of Statistical Learning.* pringer-Verlag. Citado na pág. 1, 27

Koller *et al.***(2009)** Daphne Koller, Nir Friedman e Francis Bach. *Probabilistic Graphical Models: Principles and Techniques.* The MIT Press. Citado na pág. 10

Lauritzen(1996) Steffen L. Lauritzen. *Graphical Models.* Clarendon Press. Citado na pág. 1