Estudo de técnicas de paralelização de métodos computacionais de fatoração de matrizes esparsas aplicados à redes bayesianas e redes credais

Viviane Teles de Lucca Maranhão

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

Programa: Matemática Aplicada
Orientador: Prof. Dr. Julio Michael Stern

Durante o desenvolvimento deste trabalho a autora recebeu auxílio financeiro do CNPq

São Paulo, maio de 2013

Estudo de técnicas de paralelização de métodos computacionais de fatoração de matrizes esparsas aplicados à redes bayesianas e redes credais

Esta dissertação trata-se da versão original
da aluna Viviane Teles de Lucca Maranhão.

Agradecimentos

A Deus;

Ao meu orientador, Prof. Dr. Julio Michael Stern, pela sua dedicação e atenção, pelos conselhos fornecidos e principalmente pela confiança depositada;

Ao meu marido, Flamínio, por todo apoio, enorme ajuda e incentivo nesta empreitada;

Ao Dr. Ernesto Coutinho Colla por gentilmente fornecer sua versão de desenvolvimento da biblioteca que foi complementada neste trabalho e por sua disponibilidade em conversar sobre ela;

Ao Prof. Dr. Fabio Gagliardi Cozman pelas conversas que motivaram parte do estudo desenvolvido neste trabalho;

À CNPq, pelo fomento fornecido;

Ao Instituto de Matemática e Estatística, por conceder os recursos utilizados ao longo deste período.

Resumo

Neste trabalho demos continuidade ao estudo desenvolvido por Colla (2007) que utilizou-se do arcabouço de álgebra linear com técnicas de fatoração de matrizes esparsas aplicadas à inferência em redes Bayesianas. Com isso, a biblioteca computacional resultante possui uma separação clara entre a fase simbólica e numérica da inferência, o que permite aproveitar os resultados obtidos na primeira etapa para variar apenas os valores numéricos.

Aplicamos técnicas de paralelização para melhorar o desempenho computacional, adicionamos inferência para Redes Credais e novos algoritmos para inferência em Redes Bayesianas para melhor eficiência dependendo da estrutura do grafo relacionado à rede e buscamos tornar ainda mais independentes as etapas simbólica e numérica.

Palavras-chave: Redes Bayesianas, Redes Credais, Matrizes Esparsas, Computação Paralela.

Abstract

In this work we continued the study by Colla (2007), who used the framework of linear algebra techniques with sparse matrix factorization applied to inference in Bayesian networks. Thus, the resulting computational library has a clear separation between the symbolic and numerical phase of inference, which allows you to use the results obtained in the first step to vary only numeric values.

We applied parallelization techniques to improve computational performance, we add inference to Credal Networks and new algorithms for inference in Bayesian networks for better efficiency depending on the structure of the graph related to network and seek to become more independent symbolic and numerical steps.
Keywords: Bayesian Networks, Credal Networks, Sparse Matrixes, Parallel Computing.

Sumário

Lista de Abreviaturas **xi**

Lista de Símbolos **xiii**

Lista de Figuras **xv**

Lista de Tabelas **xvii**

1 Introdução **1**
1.1 Objetivos . . . 1
1.2 Contribuições . . . 2
1.3 Organização do Trabalho . . . 2

2 Introdução à Teoria das Redes Bayesianas **3**
2.1 Redes Bayesianas . . . 3
2.1.1 Independência condicional . . . 4
2.1.2 Evidência . . . 5
2.1.3 Inferência em Redes Bayesianas . . . 6
2.1.4 Determinação das variáveis requisitadas . . . 7
2.1.5 Algoritmos de inferência em redes Bayesianas . . . 10
2.1.6 Exemplo . . . 12

3 Redes Credais **15**
3.1 Definições . . . 15
3.1.1 Inferência em Redes Credais . . . 16

4 Matrizes Esparsas e Decomposição de Matrizes **19**
4.1 Classificação de Matrizes . . . 19
4.2 Decomposição de Matrizes . . . 20
4.2.1 Permutações e Operações Elementares . . . 20
4.2.2 Método de Eliminação de Gauss . . . 21
4.2.3 Fatoração LU . . . 21
4.2.4 Fatoração de Cholesky . . . 22
4.3 Preservação da Esparsidade . . . 22
4.3.1 Escolha da Ordem de Eliminação . . . 24
4.4 Estruturas . . . 26

4.4.1 Fatoração em Matriz de Estrutura Angular Blocada 26
4.4.2 Obtendo a forma blocada de uma matriz 28

5 Conceitos de Computação Paralela 31
5.1 Definições 32
5.2 Memória Compartilhada 32
5.2.1 POSIX Threads 32
5.2.2 OpenMP 33
5.3 Memória Distribuída 33
5.3.1 MPI 34
5.4 Speed up e desempenho 35
5.4.1 Avaliando a possibilidade de paralelizar ou não 36

6 Inferência Paralela em Redes Bayesianas e Credais 39
6.1 Algoritmo de Eliminação de Variáveis Paralelo 39
6.1.1 Fase Simbólica 39
6.1.2 Fase Numérica 43
6.2 Algoritmo de Inferência Paralela para Redes Credais 44
6.3 Exemplo de Inferência Credal Paralela 44
6.4 Exemplo de Fatoração Simbólica Blocada 54

7 Implementação Computacional da biblioteca 59
7.1 Bibliotecas e Estruturas de Dados 61
7.1.1 Organização das Bibliotecas 61
7.1.2 Principais Estruturas de Dados 65
7.2 Estimação dos Tempos para Avaliar a necessidade de Paralelizar 71
7.2.1 FLOPS 72
7.2.2 PThreads 73
7.2.3 MPI 74
7.2.4 Comparação de desempenhos 74
7.3 Entrada dos dados para Inferência 76
7.4 Fatoração Simbólica da Rede Bayesiana 78
7.5 Criação das Evidências da Rede Bayesiana a partir da Rede Credal 80
7.6 Fatoração Numérica da Rede Bayesiana 81
7.7 Inferência na Rede Credal 83

8 Testes e Resultados 85
8.1 Testes de Precisão dos Resultados 85
8.2 Testes de Performance 86
8.2.1 Fatoração simbólica blocada 86
8.2.2 Inferência paralela em redes Credais 90
8.2.3 Fatoração numérica paralela 91
8.2.4 Previsão de desempenho 94

9 Conclusões **97**
9.1 Sugestões para Pesquisas Futuras 98

A Teoria dos Grafos **99**

B Teoria das Probabilidades **103**
B.1 Potenciais 105
B.1.1 Exemplos 106

C Redes dos Exemplos **109**
C.1 Rede Credal utilizada para o exemplo de inferência 109
C.1.1 Rede com limites inferiores 109
C.1.2 Rede com limites superiores 112
C.1.3 Consulta 115
C.1.4 Evidência 115
C.2 Rede Bayesiana utilizada para o exemplo de fatoração blocada 116
C.2.1 Rede 116
C.2.2 Consulta 120
C.2.3 Evidência 120
C.2.4 Parâmetros do Simulated Annealing 120

D Diagrama das Redes dos Testes **123**

Referências Bibliográficas **127**

Índice Remissivo **131**

Lista de Abreviaturas

BEL	Busca em Largura
CPU	*Central Processing Unit*
ENN	Elemento Não Nulo
FAB	Forma Angular Blocada
FABC	Forma Angular Blocada por Colunas
FABL	Forma Angular Blocada por Linhas
FLOP	*Floating-point Operation*
FLOPS	*Floating-point Operations Per Second*
GDA	Grafo Direcionado Acíclico
MPI	*Message Passing Interface*
PThreads	POSIX *threads*
RB	Rede Bayesiana

Lista de Símbolos

$A \wedge B$	Intersecção de A e B: $\{w \in A \ e \ w \in B\}$
$A \cup B$	União de A e B: $\{w \in A \ e \backslash ou \ w \in B\}$
$A \setminus B$	Diferença de conjuntos: $\{w \in A \ e \ w \notin B\}$
A^c	Complementar:$A \subseteq \Omega$: $\{w \in \Omega \ e \ w \notin A\}$
$\emptyset$	Conjunto Vazio, conjunto sem nenhum elemento
V_G	Conjunto de vértices de um grafo
A_G	Conjunto de arestas de um grafo
$G = (V_G, A_G)$	Grafo definido pelo conjunto de vértices e arestas
$H = (V_H, A_H)$	Árvore definida pelo conjunto de vértices e arestas
B_G	Matriz booleana de adjacência do grafo G
$\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$	Conjunto de variáveis aleatórias
$P(X)$	Densidade de probabilidade de uma variável X
$P(X = x)$ ou $P(x)$	Probabilidade de X assumir um valor específico x
$P(X\|Y)$	Densidade de probabilidade da variável X condicionada à variável Y
$\sum_{\mathbf{X}} f(\mathbf{X}, \mathbf{Y})$	Eliminação por marginalização de todas variáveis de $\mathbf{X}$ de f
$pa(X_i)$	Conjuntos de vértices (ou variáveis) que são pais da variável X_i
$ch(X_i)$	Conjuntos de vértices (ou variáveis) que são filhos da variável X_i
$sp(X_i)$	Conjuntos de vértices (ou variáveis) que são esposos da variável X_i
$mb(X_i)$	Conjuntos de vértices que compõe o Markov Blanket da variável X_i
$\phi(X_i)$	Potencial da variável X_i
$dom(\phi(X_i))$	Domínio do potencial da variável X_i
$\#X$	Número de Elementos do conjunto X

Lista de Figuras

2.1 Exemplo de um GDA que define uma Rede Bayesiana . . . 3
2.2 Tipos de conexões possíveis entre as variáveis A e C em um GDA . . . 5
2.3 Regras de movimentação de uma bola no bayes-ball . . . 8
2.4 Exemplo de execição do bayes-ball . . . 8
2.5 Rede Bayesiana Simples . . . 12
2.6 Eliminação de variáveis em uma Rede Bayesiana Simples . . . 13
2.7 Incorporação de evidência na eliminação de variáveis em uma Rede Bayesiana Simples 13
2.8 Grafo de Domínio de uma Rede Bayesiana Simples . . . 14
2.9 Eliminação dos vértices de uma Rede Bayesiana Simples . . . 14

3.1 Exemplo de uma rede Credal . . . 15
3.2 Dog Problem em sua versão credal . . . 17
3.3 Dog Problem em sua versão credal após a transformação CCM . . . 18

4.1 Fatoração de Cholesky . . . 22
4.2 Árvores de Eliminação para algumas matrizes de adjacência . . . 23
4.3 Exemplo de um grafo e sua respectiva árvore de dissecção . . . 24
4.4 Exemplo de árvore gerada através das Heurísticas de Gibbs e BEL . . . 26
4.5 Árvore de dissecção gerada após as heurísticas de Gibbs e BEL . . . 26

5.1 Exemplo de arquitetura de memória compartilhada (a) e distribuída (b) . . . 31
5.2 Topologias de rede: anel (a), estrela (b), árvore (c) e grafo completo (d). . . . 34

6.1 Etapas da inferência na rede . . . 40
6.2 Exemplo de processamento numérico paralelo . . . 44
6.3 Exemplo de Rede Credal para inferência paralela . . . 45
6.4 Exemplo de Rede Credal com Variáveis Transparentes para inferência paralela . . . 45
6.5 Aplicação do Bayes-Ball para definir X_R na rede de exemplo . . . 48
6.6 Grafo Moral da rede de exemplo . . . 48
6.7 Exemplo de Árvore BEL (a) e Árvore de dissecção(b) . . . 49
6.8 Fatoração simbólica do Grafo Moral . . . 50
6.9 Árvore de eliminação e Grafo Preenchido obtidos através da fatoração simbólica . . . 50
6.10 Operações com potenciais envolvidas na eliminação de variáveis . . . 51
6.11 Grafo da rede Bayesiana para o exemplo de inferência blocada . . . 54
6.12 Grafo Moral da rede Bayesiana para o exemplo de inferência blocada . . . 54
6.13 Fatoração simbólica dos subgrafos . . . 56

6.14 Árvore de eliminação obtida através da fatoração simbólica dos subgrafos 57

7.1 Organização dos Arquivos e Bibliotecas 61
7.2 Estruturas usadas para manipulação de grafos 65
7.3 Estruturas usadas para a implementação da lista ligada simples 66
7.4 Estruturas usadas para implementação do *Hash* 67
7.5 Estruturas usadas para implementação da árvore de *threads* 67
7.6 Sequência de execução de uma árvore de *threads* 68
7.7 Estruturas usadas para implementação do gerador de *ids* únicas 68
7.8 Estruturas usadas para implementação da rede Bayesiana e Eliminação de Variáveis . 69
7.9 Estruturas usadas para implementação uma rede Credal 70
7.10 Estruturas usadas para medir tempo e desempenho 71
7.11 Processos e estruturas de dados relacionados à medição de tempo 72
7.12 Processos e estruturas de dados relacionados à entrada de dados 77
7.13 Processos e estruturas de dados relacionados à fatoração simbólica 79
7.14 Processos e estruturas de dados relacionados à ordem de eliminação paralela 79
7.15 Processos e estruturas de dados relacionados à ordem de eliminação serial 80
7.16 Processos e estruturas de dados para criação de evidências a partir da Rede Credal . 81
7.17 Processos e estruturas de dados relacionados à fatoração numérica 81
7.18 Processos e estruturas de dados relacionados à inferência na rede Credal 83

8.1 Árvore de *Threads* produzida para o teste serial 01 da rede Blocada02 88
8.2 Árvore de *Threads* produzida para o teste blocado 01 da rede Blocada02 89
8.3 Árvore de *Threads* produzida para o teste serial 02 da rede Blocada02 89
8.4 Árvore de *Threads* produzida para o teste blocado 02 da rede Blocada02 89

A.1 Exemplo de um grafo 99
A.2 Exemplo de um GDA, seu grafo Moral e Markov Blanket de um vértice 100
A.3 Processo de eliminação em um grafo não direcionado e grafo preenchido 101
A.4 Processo de eliminação simplificada em um grafo não direcionado e grafo preenchido 101

D.1 Diagrama da Rede: Exemplo01 123
D.2 Diagrama da Rede: DogProblem 123
D.3 Diagrama da Rede: Asia 124
D.4 Diagrama da Rede: Hailfinder25 124
D.5 Diagrama da Rede: Exemplo02 125
D.6 Diagrama da Rede: Munin1 125
D.7 Diagrama da Rede: Pigs 126

Lista de Tabelas

5.1 Principais comandos MPI 35

6.1 Probabilidades da Rede credal para inferência paralela 46
6.2 Probabilidades da Rede Credal com Variáveis Transparentes para inferência paralela 46

8.1 Resultados dos testes para fatoração simbólica blocada 87
8.2 Resultados dos testes para inferência paralela em redes Credais 91
8.3 Resultados dos testes para fatoração numérica paralela 93
8.4 Resultados da previsão de desempenho para inferência em redes Credais 95
8.5 Resultados da previsão de desempenho para eliminação blocada 95
8.6 Resultados da previsão de desempenho para fatoração numérica paralela 96

B.1 Distribuição conjunta de filhas e filhos de uma família 105
B.2 Distribuição condicionais de filhas dados os filhos de uma família 105
B.3 Potenciais $\phi_1(A, B)$ e $\phi_2(B, C)$ 106
B.4 Produto $\phi_1(A, B) \cdot \phi_2(B, C)$ 106
B.5 $\sum_C \phi_1(A, B) \cdot \phi_2(B, C)$ 106
B.6 $\sum_C \phi_2(B, C)$ 107
B.7 $\phi_1(A, B) \sum_C \cdot \phi_2(B, C)$ 107

Capítulo 1

Introdução

As Redes Bayesianas constituem um modelo gráfico (Grafo direcional acíclico) que representa as relações de causalidade das variáveis de um sistema. Seja $X = X_1, X_2, \ldots, X_n$ o conjunto destas variáveis, uma Rede Bayesiana é composta por vértices e arcos, sendo que cada vértice i representa uma variável X_i de X e os arcos representam a existência de dependência direta entre os nós quantificada pela probabilidade condicional. Para cada variável aleatória X_i está associada uma probabilidade condicional $P(X_i|pa(X_i))$, onde $pa(X_i)$ representa o conjunto de pais $Y_1, ..., Y_m$ desta variável. Estas redes têm ampla utilização em problemas de inteligência artificial, bioinformática, medicina, engenharia, otimização e também em outras situações que envolvam matrizes esparsas.

A inferência em redes Bayesianas é feita através das probabilidades condicionais e pode ser utilizada para tomar decisões baseadas em probabilidades, fazer consultas a partir de evidências, decidir quais evidências adicionais devem ser observadas a fim de se obter informações úteis do sistema, analisar o sistema a fim de buscar os aspectos do modelo que possuem maior impacto sob as variáveis de consulta, entre outros usos.

A implementação computacional desses modelos pode permitir uma série de otimizações, gerando ganho de desempenho, devido às características das redes. Um particular problema está na fatoração de matrizes esparsas; uma matriz esparsa possui apenas um pequeno número de coeficientes não nulos de forma que identificar o padrão de esparsidade e mantê-lo ao longo de transformações minimiza o número de operações aritméticas a serem realizadas. Colla (2007) em seu trabalho mostra que este problema pode ser resumido a uma escolha eficiente para ordem da eliminação das variáveis. Para tanto, existem alguns procedimentos heurísticos como de Gibbs em uma estrutura de árvore de dissecção.

1.1 Objetivos

O objetivo de estudo neste trabalho é um aprofundamento na teoria das redes Bayesianas e redes Credais e aplicações práticas das mesmas através da implementação dos modelos, e comparando o desempenho dos algoritmos que utilizam principalmente computação paralela com algoritmos que utilizam exclusivamente programação sequencial. Sobretudo, desejamos dar continuidade ao estudo desenvolvido por Ernesto Colla em seu trabalho Colla (2007), através de suas sugestões de trabalhos futuros como a complementação dos testes por ele elaborados e a implementação de métodos que descubram, antes da execução, se é computacionalmente mais eficiente executar as operações de um determinado problema de forma serial ou paralela. Além disso implementaremos um método paralelo em memória distribuída para inferência em redes Credais e utilizaremos técnicas de programação paralela para aumentar o desempenho computacional da biblioteca, nos casos em que este tipo de procedimento for vantajoso.

1.2 Contribuições

As principais contribuições deste trabalho são as seguintes:

- Fornecer um ambiente *Open-Source* e eficiente de inferência em redes Bayesianas e redes Credais;
- Permitir escalar de maneira praticamente infinita o problema original para inferência em redes Credais;
- Estruturar o ambiente computacional de modo que seja possível aproveitar posteriormente os resultados de uma fatoração simbólica para diversas evidências;
- Fornecer um método de fatoração simbólica que obtém vantagem da topologia da rede;
- Analisar e prever o desempenho computacional dos métodos implementados.

1.3 Organização do Trabalho

No Capítulo 2, apresentamos os conceitos de redes Bayesianas que serão utilizados no desenvolvimento deste trabalho, e conceitos de redes Credais utilizados são apresentados no Capítulo 3. O Capítulo 4 apresenta conceitos de álgebra linear para a utilização de fatoração de matrizes, a fatoração de Cholesky, ideal para matrizes esparsas e também conceitos sobre fatoração de matrizes com estruturas especiais. O Capítulo 5 apresenta alguns paradigmas de computação paralela que justificam as implementações do algoritmo de eliminação apresentado. Temos no Capítulo 6 a descrição de todo o procedimento paralelo utilizado para a eliminação distribuída de variáveis em redes Bayesianas e credais. Finalmente, no Capítulo 7 apresentamos a biblioteca computacional que resulta da implementação dos algoritmos de inferência para redes descritos nas seções anteriores. No Capítulo 8 apresentamos alguns exemplos e resultados obtidos e no Capítulo 9 discutimos algumas conclusões obtidas neste trabalho.

Um breve resumo sobre os conceitos de teoria dos grafos pode ser visto no Apêndice A, assim como um resumo dos principais conceitos de teorias das probabilidades utilizados pode ser encontrado no Apêndice B para uma maior compreensão dos assuntos abordados na teoria das redes Bayesianas e Credais. O Apêndice C apresenta o conteúdo dos arquivos utilizados para as redes dos exemplos passo a passo deste trabalho e o Apêndice D apresenta os diagramas das redes utilizadas nos testes computacionais.

Capítulo 2

Introdução à Teoria das Redes Bayesianas

Apresentaremos neste Capítulo uma revisão bibliográfica de redes Bayesianas. Nosso objetivo é fornecer uma visão geral de cada tema e não esgotá-los completamente, para isso, recomendamos a leitura das referências indicadas em cada Seção. Assumiremos também um conhecimento prévio sobre grafos e probabilidades, entretanto, algumas definições principais podem ser encontradas nos Apêndices A e B, respectivamente.

2.1 Redes Bayesianas

Redes Bayesianas são uma ferramenta bastante útil para lidar com incertezas, tendo aplicações na medicina, biologia, sociologia, psicologia, economia, entre outras áreas. O termo "Rede Bayesiana" foi criado em 1985 por Judea Pearl para enfatizar três aspectos: (1) A natureza subjetiva das informações de entrada; (2) A confiança na regra de Bayes para atualização de informação; (3) A distinção entre o raciocínio causal e o baseado em evidência. Elas começaram a ser mais profundamente estudadas no fim dos anos 80 e vem se popularizando com o passar dos anos. Uma das grandes vantagens das redes Bayesianas é a facilidade com que podemos alterar algum evento com apenas poucas alterações no modelo, em particular, apenas naquelas variáveis relacionadas a ele, outra vantagem é que quando cada variável depende apenas de um conjunto pequeno de outras variáveis a representação em forma de rede Bayesiana pode ser bem econômica (Pearl, 2009).

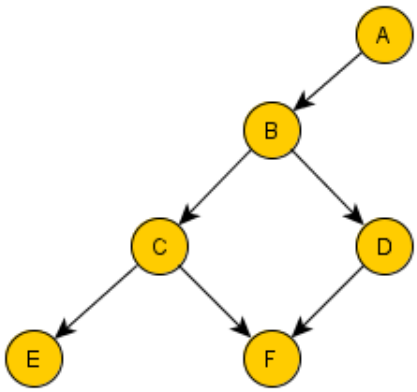


Figura 2.1: Exemplo de um GDA que define uma Rede Bayesiana

Uma rede Bayesiana *(RB)* é um grafo direcionado acíclico em que os vértices representam variáveis com conjuntos finitos de estados possíveis e associado aos vértices arestas representam as relações entre as variáveis. Adicionalmente, para cada variável X com pais Y_1, Y_2, ..., Y_n temos

associada uma tabela de probabilidade condicional $P(X|Y_1, Y_2, \ldots, Y_n, I)$, onde I representa todo o conhecimento prévio, que é um conhecimento relevante que não aparece na forma de vértices do grafo, podendo algumas vezes ser omitido da notação para simplificá-la. Se X for um vértice raiz então sua tabela se reduz a $P(X|I)$, não dependendo dos outros vértices do grafo (Taroni *et al.*, 2006).

A Figura 2.1 apresenta um exemplo de rede Bayesiana. Nela é necessário especificar as seguintes probabilidades $P(A)$, $P(B|A)$, $P(C|B)$, $P(D|B)$, $P(E|C)$, e $P(F|C, D)$. As mesmas relações entre vértices de um GDA podem ser aplicadas para redes Bayesianas, por exemplo, podemos ver que C e D são esposos, F é filho de C e D, C é pai de E, C é descendente de A, mas não de D, entre outras relações.

2.1.1 Independência condicional

Chamamos a probabilidade P que gera o GDA de uma rede Bayesiana de **probabilidade adjacente**. Dado que cada vértice X_i possui associado a ele $P(X_i|pa(X_i))$, Geiger *et al.* (1990) nos mostra como calcular a probabilidade adjacente de uma rede Bayesiana para um conjunto de variáveis $X = \{X_1, \ldots, X_n\}$:

$$P(X) = \prod_{i=1}^{n} P(X_i|pa(X_i)) \tag{2.1}$$

Uma consulta padrão em uma rede Bayesiana é verificar a crença de uma hipótese α dado um conjunto de evidências $Y = y$, ou seja, calcular $P(X|y)$ para cada um dos valores de α e uma combinação dada de valores de Y. Apesar de, tecnicamente, a Equação 2.1 ser suficiente para obter o valor desejado, o procedimento pode ser custoso em armazenamento e tempo computacionais, o que torna interessante explorar as relações de independência entre as variáveis apresentada pela rede. Para tratar melhor esta questão utilizaremos alguns conceitos que apresentaremos a seguir.

A semântica das redes Bayesianas é delimitada pela **condição de Markov**, que origina a Equação 2.1 (e faz com que $P(X)$ seja única) e incorpora a topologia do GDA para as relações entre variáveis dentro da rede. Esta condição pode ser enunciada da seguinte forma: *Toda variável é independente de seus não-descendentes não-pais dados os seus pais* (Cozman, 2000a). O conceito de d-separação em um GDA apresentado a seguir permitirá estudar mais profundamente as relações de independência entre as variáveis.

d-separação

Duas variáveis distintas X e Z em uma rede Bayesiana são **d-separadas** se para todos os caminhos entre X e Z existe uma variável intermediária V, (distinta de X e Z) tal que uma das seguintes condições é satisfeita:

- A conexão entre as variáveis é serial ou divergente e o valor de V é conhecido;
- A conexão entre as variáveis é convergente e os valores de V e de seus descendentes são desconhecidos.

Se X e Z não são d-separadas, dizemos que são d-conectadas.

A Figura 2.2 apresenta os tipos de conexões possíveis entre as variáveis em uma rede Bayesiana, ou seja, as três maneiras possíveis que uma informação pode trafegar pela rede, ilustrando a d-separação. Ding (2010) mostra que para as conexões divergentes (a) e seriais (b) se o valor de Y for conhecido, ela d-separa X e Z pois elas se tornam variáveis independentes. Por outro lado, em uma conexão convergente (c), o conhecimento do valor de Y d-conecta X e Z.

Uma propriedade fundamental das Redes Bayesianas é que se duas variáveis, representadas por vértices de um GDA, estão graficamente d-separadas então estas variáveis são independentes (Pearl, 1988). Esta propriedade é importante pois permite identificar o conjunto mínimo de variáveis que são estatisticamente relevantes para se inferir sobre uma variável ou um conjunto de variáveis da

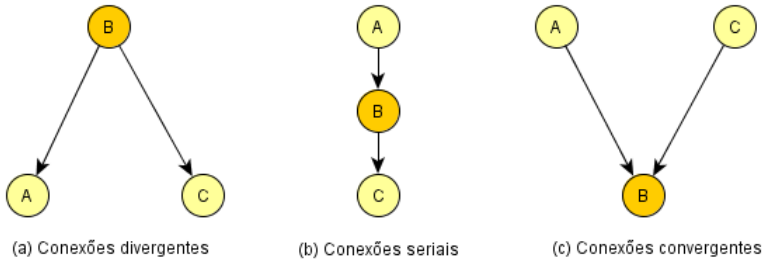


Figura 2.2: Tipos de conexões possíveis entre as variáveis A e C em um GDA

rede. Com isto é possível reduzir significativamente as dimensões do problema de inferência e, consequentemente, a necessidade obrigatória de processamento numérico.

O conceito de *Markov Blanket* de um GDA pode ser estendido para o contexto das redes Bayesianas como os pais, filhos e esposos (pais dos filhos) de uma variável. É ele o conjunto mínimo de variáveis que d-separa uma variável de todas as outras da rede, ou seja, que torna uma variável X independente do restante da rede (Neapolitan, 2003).

2.1.2 Evidência

Pode ser de nosso interesse incorporar novas informações em um modelo representado por uma rede Bayesiana, seja por ter observado o valor de alguma variável, ou por ter a informação de que alguns de seus estados não são mais passíveis de serem assumidos. Dada uma rede Bayesiana, um evento E é chamado de evidência e corresponde ao conjunto de valores observados de determinadas variáveis da rede e X_E é o conjunto de variáveis aos quais os estados ou valores são conhecidos. Podemos ter, por exemplo, $E = \{X_1 = x_{11}, X_2 = x_{22}\}$ indicando os estados assumidos pelas variáveis X_1 e X_2 na rede e, neste caso, $X_E = \{X_1, X_2\}$. Podemos ainda dizer que uma evidência torna a probabilidade do estado observado da variável igual a 1 e todos os outros iguais a zero.

Vamos supor agora que X_1 possua n estados possíveis com $P(X_1) = (x_{11}, x_{12}, \ldots, x_{1n})$ e assuma que temos a informação e de que X_1 pode assumir somente um dos estados i e j. Isso significa que todos os outros estados diferentes de i e j são impossíveis e que a probabilidade $P(X_1, e) = (0, \ldots, 0, x_i, 0, \ldots, 0, x_j, 0, \ldots, 0)$. Note que $P(X_1, e)$ é obtida através da multiplicação de $P(X_1)$ por um vetor no qual todas as posições são zero, com excessão das posições i e j (ou dos estados conhecidos), que valem 1. A esse vetor damos o nome de *finding* e o representamos por $\underline{e}$ para que não se confunda com a informação e "X_1 está no estado i ou j" (Jensen, 2001).

Seja $P(X)$ uma tabela com a probabilidade conjunta $P(X_1, X_2, \ldots, X_n)$ e $\underline{e}$ um *finding*. A tabela com a probabilidade conjunta $P(X, E)$ é a tabela resultante de $P(X)$ quando atribuímos zero à posição todos os estados que são excluídos por $\underline{e}$, ou seja através da multiplicação de $P(X)$ por $\underline{e}$:

$$P(X, E) = P(X) \cdot \underline{e}.$$

Note que $P(E)$ é a probabilidade a priori de E e pode ser escrita como $P(E) = \sum_X P(X, E) = \sum_X (P(X) \cdot \underline{e})$. Através da regra da cadeia para redes Bayesianas temos o seguinte teorema:

Teorema 2.1 (Jensen (2001)) *Seja RB uma rede Bayesiana sobre X e $\underline{e_1}$, $\underline{e_2}$, ..., $\underline{e_m}$ findings. Então:*

$$P(X, E) = \prod_{i=1}^{n} P(X_i | pa(X_i)) \cdot \prod_{j=1}^{m} \underline{e_j},$$

e para $X_i \in X$ temos:

$$P(X_i | E) = \frac{\sum_{X \setminus X_i} P(X, E)}{P(E)}$$

2.1.3 Inferência em Redes Bayesianas

Redes bayesianas podem ser utilizadas para calcular novas probabilidades quando informações são adquiridas. Inferência em uma rede Bayesiana corresponde ao processo de obter as distribuições de todas as variáveis da rede condicionais a um determinado conjunto de variáveis.

Uma vez conhecida a distribuição conjunta de probabilidades das variáveis de uma rede é possível responder a todas possíveis demandas por inferência sobre uma ou mais variáveis da mesma através do processo de marginalização de todas as variáveis da distribuição conjunta, exceto aquelas que se deseja conhecer a distribuição marginal. Chamamos as variáveis que desejamos conhecer a distribuição marginal a posteriori de **variáveis questionadas** (*query variables*). Para realizar a inferência em redes Bayesianas utilizaremos potenciais, assim, a equação da probabilidade adjacente para um conjunto de variáveis X 2.1 passa a ser dada por:

$$P(X) = \prod_{i=1}^{n} \Phi_i(P(X_i|pa(X_i))) \tag{2.2}$$

Vamos ilustrar o processo de inferência através da rede apresentada na Figura 2.1 cujo conjunto de variáveis é $\mathbf{X} = \{A, B, C, D, E, F, G\}$. Temos as seguintes probabilidades e potenciais, com os domínios entre parênteses: $P(A) = \Phi_1(A)$, $P(B|A) = \Phi_2(A, B)$, $P(C|B) = \Phi_3(B, C)$, $P(D|B) = \Phi_4(B, D)$, $P(E|C) = \Phi_5(C, E)$ e $P(F|C, D) = \Phi_6(C, D, F)$.

A distribuição conjunta das variáveis $\mathbf{X}$ é portanto:

$$P(\mathbf{X}) = P(A) \cdot P(B|A) \cdot P(C|B) \cdot P(D|B) \cdot P(E|C) \cdot P(F|C, D).$$

Ou ainda, utilizando potenciais:

$$P(\mathbf{X}) = \Phi_1(A) \cdot \Phi_2(A, B) \cdot \Phi_3(B, C) \cdot \Phi_4(B, D) \cdot \Phi_5(C, E) \cdot \Phi_6(C, D, F).$$

Caso desejemos conhecer a distribuição $P(D)$, devemos fazer a eliminação por marginalização da distribuição conjunta as demais variáveis:

$$P(D) = \sum_{A,B,C,E,F} \Phi_1(A) \cdot \Phi_2(A, B) \cdot \Phi_3(B, C) \cdot \Phi_4(B, D) \cdot \Phi_5(C, E) \cdot \Phi_6(C, D, F).$$

É possível aplicar a propriedade distributiva dos potenciais antes das operações produto de marginalização para que não sejam calculados todos simultaneamente, de forma a minimizar o número de potenciais em cada operação, ou seja:

$$P(D) = \sum_{A} \Phi_1(A) \cdot \sum_{B} \Phi_2(A, B) \cdot \Phi_4(B, D) \cdot \sum_{C} \Phi_3(B, C) \cdot \sum_{E} \Phi_5(C, E) \cdot \sum_{F} \Phi_6(C, D, F).$$

Para isso, inicialmente devemos calcular $\Phi'_6 = \sum_F \Phi_6(C, D, F)$ e multiplicar o resultado por Φ_5 para obter $\Phi'_5 \sum_E \Phi_5(C, E) \cdot \Phi'_6$, multiplicar o resultado obtido por Φ_3 e assim por diante. Por outro lado, o cálculo de Φ_5 não depende de F, portanto é possível aplicar novamente a propriedade distributiva para o cálculo de Φ'_5, o que indica que algumas operações de marginalização poderiam ocorrer em paralelo.

Nesse processo de marginalização a cada operação uma variável é eliminada da distribuição de probabilidades, por isso este processo também é chamado de **eliminação de variáveis**. Cabe ressaltar que uma ordem de eliminação pré-definida especifica a ordem de marginalização das variáveis e que ela está diretamente relacionada com o custo computacional da marginalização. A escolha de uma boa ordem de eliminação está intimamente ligada à topologia do GDA que define a rede. Graficamente, o processo de eliminação de variáveis corresponde à eliminação de vértices no Grafo Moral.

2.1.4 Determinação das variáveis requisitadas

Seja X_q um conjunto de variáveis questionadas, E uma evidência e X_E o conjunto de variáveis dessa evidência. A probabilidade a posteriori de X_q dada E é:

$$P(X_q|E) = \frac{P(X_q, E)}{P(E)} = \frac{\sum_{X \setminus \{X_q, X_E\}} P(X)}{\sum_{X \setminus X_E} P(X)} \tag{2.3}$$

E, computacionalmente, o numerador é o termo mais importante desta equação uma vez que o denominador é uma constante de normalização que pode ser facilmente obtida através do numerador, sendo deixado para o fim dos cálculos (Cozman, 2000a). Apesar do numerador não ser sozinho uma medida de probabilidade, é possível tratá-lo como um potencial cujo domínio é $dom(\Phi) = X_q \cup X_E$.

Tipicamente o cálculo de $P(X_q|E)$ envolve a densidade de apenas um conjunto de variáveis da rede. Caso a densidade $p(X_i|pa(X_i)$ seja necessária para realizar determinada consulta, então X_i é chamada de **variável requisitada** e chamamos X_R o conjunto de todas as variáveis requisitadas. As variáveis de X_q necessariamente pertencem a X_R, porém apenas as variáveis observadas de X_E que possuem pais não observados em X_R pertencem a X_R.

Poder determinar antecipadamente o conjunto de variáveis requisitadas elimina esforços computacionais desnecessários e está relacionado com uma boa manipulação da topologia da rede. Algoritmos baseados no critério de d-separação podem executar a tarefa em tempo polinomial. Neste trabalho utilizaremos o **Bayes-Ball** (Shachter, 1998) que consiste em uma bola imaginária que percorre os nós da rede seguindo o fluxo da informação. Utilizamos este algoritmo devido à sua simplicidade e aproveitamento do que já fora feito por (Colla, 2007). Outros algoritmos podem ser encontrados em (Geiger *et al.*, 1990) e (Torti e Wuillemin, 2010).

Bayes-Ball

O Bayes-Ball é um algoritmo que define as variáveis probabilisticamente irrelevantes para determinada consulta em uma RB em tempo linear sobre o tamanho da rede.

Retomando, dada uma distribuição de probabilidade conjunta para X_N, $P(X_N)$ de uma rede $G = (N, A)$, X_J é **probabilisticamente irrelevante** (também chamado de independente) para X_L dado X_K para $J, K, L \subseteq N$ se

$$P(X_J|X_K, X_L) = P(X_J|X_K)$$

X_J é dito **irrelevante** para X_L dado X_K em G, denotado por $X_J \perp_G X_L|X_K$, se X_J for probabilisticamente irrelevante para X_L dado X_K para qualquer distribuição de probabilidade conjunta $P(X_N)$ que possa ser marginalizada. Em outras palavras, X_J é irrelevante para X_L dado X_K em G se, tendo observado X_K, nada mais puder ser dito sobre X_J ao se observar X_L.

Utilizando o conceito de d-separação, fortemente utilizado para elaboração do algoritmo, temos que dado um grafo $G = (N, A)$ e $J, K, L \subseteq N$,

$$X_J \perp_G X_L|X_K \Leftrightarrow \text{K D-separa J de L em G}$$

A bola lançada para a execução do algoritmo pode ter três destinos. Ela pode:

- Passar pelo nó (De um pai para os filhos, de um filho para os pais);
- Pode rebater no nó e voltar (de um pai para os pais, de um filho para os filhos)
- Ser bloqueada e não se movimentar mais.

A Figura 2.3 ilustra as regras de movimentação da bola. Nós mais escuros representam variáveis observadas que fazem parte do conjunto de evidências E. As setas preenchidas representam os arcos direcionados que formam a rede e as setas pontilhadas representam o sentido de deslocamento da bola ao visitar o nó, sendo que uma seta com barreira indica que a bola parou. Podemos observar

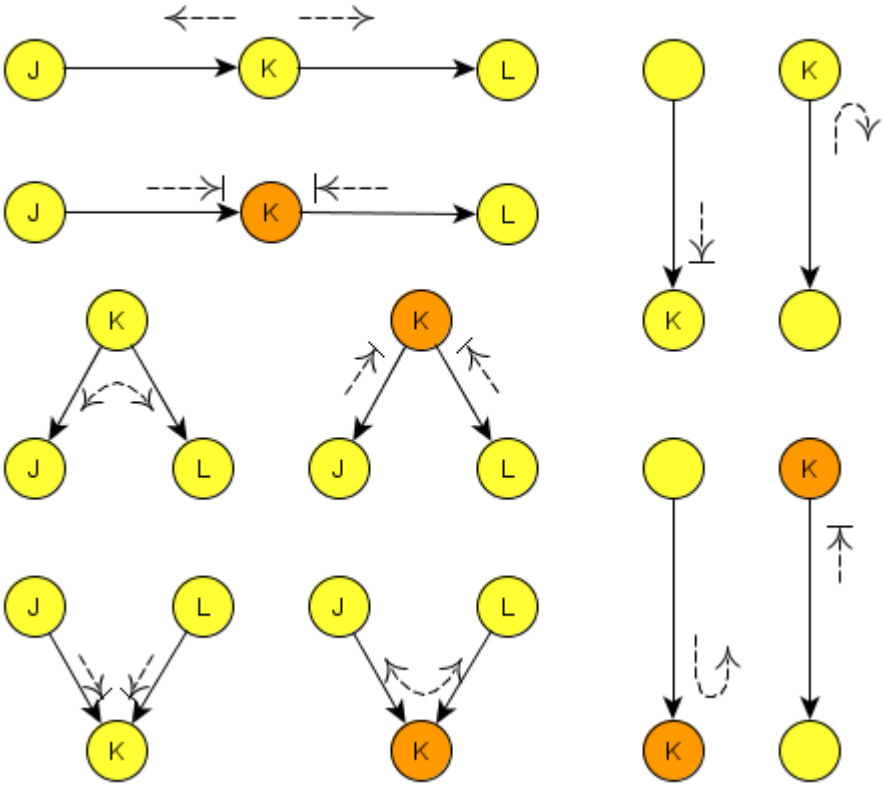


Figura 2.3: Regras de movimentação de uma bola no bayes-ball

que um nó probabilístico não observado passa a bola através de si mas rebate de volta a bola vinda de seus filhos e que um nó probabilístico observado rebate de volta a bola vinda de seus pais mas bloqueia a bola vinda de seus filhos.

A primeira versão do algoritmo considera apenas o fluxo da bola para identificar as variáveis relevantes. A segunda versão, mais refinada, mantém uma lista dos nós a serem visitados pelos pais e pelos filhos e apenas visita um mesmo arco em uma mesma direção uma vez. Isto é feito marcando o topo do nó quando a bola é enviada aos seus pais e marcando a parte inferior do nó quando a bola for enviada aos seus filhos, e assinalando os nós observados visitados. Além dos ganhos computacionais, essas melhorias evitam que o programa entre em um loop infinito.

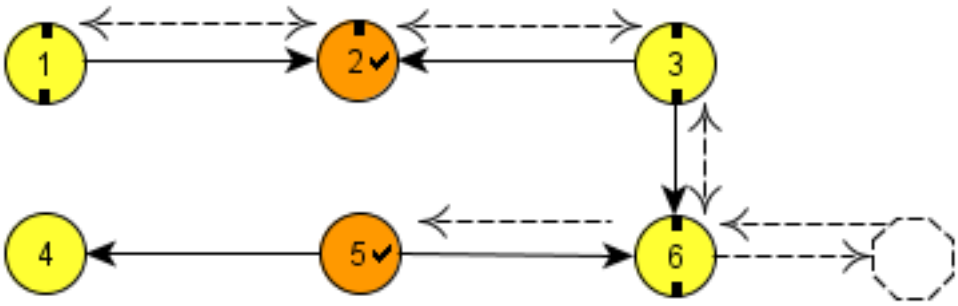


Figura 2.4: Exemplo de execição do bayes-ball

A Figura 2.4 ilustra uma execução da versão refinada do algoritmo, para uma rede Bayesiana em que $X_q = \{6\}$ e $E = \{2, 5\}$. A execução do procedimento é a seguinte:

- No início do procedimento o nó 6 é visitado, como se fosse a partir de um filho e manda a bola para seus pais e seus filhos, marcando seu topo e sua parte inferior;
- O nó 5 não manda a bola para ninguém, portanto, não é marcado, mas como é um nó observado é assinalado;
- O nó 3 é visitado através de um nó filho (6), então manda a bola para seus pais e filhos e marca seu topo e parte inferior;
- O nó 6 recebe a bola de seu pai, mas como já está marcado na parte inferior, não faz nada;
- O nó 2 recebe a bola de seu pai (3), manda para seus pais e marca seu topo;
- O nó 1 recebe a bola de seu nó filho (2), manda para seus pais e filhos e marca seu topo e parte inferior;
- Finalmente os nós 2 e 3 recebem a bola mas reconhecem que não há mais nada a ser feito por já estarem marcados.

Shachter (1998) apresenta este e outros exemplos adicionais de execução do algoritmo. A seguir apresentaremos a versão final do algoritmo.

Algoritmo Bayes-Ball

O Algoritmo explora uma rede Bayesiana com variáveis $X = \{X_1, X_2, \ldots, X_n\}$ em que se deseja calcular $P(X_q|E)$ construindo o conjunto X_R.

1. Inicialize todos os nós como não visitados e não marcados nas partes superior e inferior;
2. Crie uma lista de nós a serem visitados, inicializando cada nó em X_q como se fosse visitado por um filho;
3. Enquanto houverem nós a serem visitados na lista:
   (a) Pegue um nó j da lista e remova-o da mesma. j pode estar programado para ser visitado por um pai, um filho, ou ambos;
   (b) Marque j como visitado;
   (c) Se $j \notin E$ e a visita de j for a partir de um filho:
      i. Se o topo de j não estiver marcado, marque-o e adicione cada um de seus pais na lista de nós a serem visitados;
      ii. Se a parte inferior de j não estiver marcada, marque-a e adicione cada um de seus filhos na lista de nós a serem visitados;
   (d) Se a visita a j for a partir de um pai:
      i. Se $j \in E$ e o topo de j não estiver marcado, marque-o e adicione cada um de seus pais na lista de nós a serem visitados;
      ii. Se $j \notin E$ e a parte inferior de j não estiver marcada, marque-a e adicione cada um de seus filhos na lista de nós a serem visitados;
4. Os nós irrelevantes são aqueles não marcados na parte inferior;
5. Os nós probabilísticos requisitados são aqueles marcados no topo;
6. Os nós observados requisitados são os nós em E marcados como visitados (assinalados);
7. X_R é formado pela união dos nós probabilísticos requisitados com os nós observados requisitados.

2.1.5 Algoritmos de inferência em redes Bayesianas

A escolha ideal do algoritmo de inferência a ser utilizado está diretamente ligada à natureza da rede. Temos algoritmos exatos e aproximados, sendo que os exatos tem a vantagem de apresentar o valor da probabilidade sem erros e os aproximados são úteis para redes grandes em que o processamento exato poderia ficar muito lento.

Entre os algoritmos exatos podemos citar o *bucket elimination* (Dechter e Dechter, 1996), e eliminação de variáveis (Cozman, 2000a), que consistem basicamente em eliminar da rede as variáveis não consultadas não observadas e uma a uma acumular seus valores através de marginalização; algoritmos baseados em *polytrees* (um grafo direcionado com no máximo um caminho não-direcionado entre quaisquer outros dois vértices) que utilizam a técnica de propagação de crença (Kim e Pearl, 1983) baseado na troca de mensagens entre os vértices do grafo moral, explorando as propriedades da topologia do grafo e que pode ser útil para múltiplas inferências pois uma mesma mensagem pode contribuir para mais de um marginal e ainda algoritmos em *junction trees* (Jensen, 1996) e (Cowell, 1999) utilizados para grafos mais genéricos que trabalha com a moralização do grafo, triangularização, e explora os cliques do grafo.

Dentre os algoritmos aproximados temos a simulação estocástica que, segundo Castillo *et al.* (1997) gera uma amostra a partir da função de probabilidade conjunta das variáveis e também usa mecanismos aleatórios. Estas amostras são utilizadas para calcular valores aproximados das probabilidades de certos nós, dada a evidência. Entre esses algoritmos podemos mencionar por exemplo *logic sampling* (Henrion, 1986), também chamado de método de aceitação-rejeição que inicia a simulação em nós sem pais e atribui um valor baseado em sua probabilidade a priori e vai percorrendo os nós filhos através da atribuição de valores feita por sorteio de números aleatórios até que todos os nós da rede tenham sido visitados, e depois calculando quantas vezes cada variável foi atribuída; *likelihood weighting* (Fung e Chang, 1990) e (Shachter e Peot, 1990) que é um tipo de amostragem por importância que fixa as variáveis com evidências gerando apenas amostras nas variáveis não observadas e ponderando cada amostra pela verossimilhança de acordo com a evidência. Temos ainda métodos direcionais como *backward simulation* (Fung e Del Favero, 1994) que parte das evidências e faz a simulação na direção oposta aos arcos, e métodos baseados e métodos de Monte Carlo para cadeias de Markov como *gibbs sampling* (Casella e George, 1992) que busca amostrar estados que representem a distribuição a posteriori a partir do *Markov Blanket* das variáveis. Ide e Cozman (2002) mencionam ainda métodos de Quasi-Monte Carlo, que trabalham com a geração de números quasi-aleatórios para obter taxas de convergência superiores as do Monte Carlo tradicional.

Mostraremos agora o método utilizado por Colla (2007), fortemente baseado no método de eliminação de variáveis e *bucket elimination* cuja escolha foi motivada foi por apresentarem esquemas algébricos para inferência relativamente simples de serem entendidos e implementados e permitirem a aplicação de técnicas, heurísticas e estruturas combinatórias abstratas características da fatoração de Cholesky em matrizes esparsas para otimizar e paralelizar o processamento numérico.

Método de Eliminação de Variáveis

Dado um conjunto de variáveis questionadas X_q, as evidências E e as variáveis requisitadas X_R queremos calcular:

$$P(X_q, E) = \sum_{X_R \setminus \{X_q, X_E\}} \left(\prod_{X_i \in X_R} P(X_i | pa(X_i)) \right) \tag{2.4}$$

Seja N o número de variáveis requisitadas que não foram observadas e também não pertencem a X_q e suponha que essas variáveis estão ordenadas de uma forma específica $q = [(1), (2), \ldots, (N)]$.

Temos então:

$$P(X_q, E) = \sum_{X_{\{(N)\}}} \ldots \sum_{X_{\{(1)\}}} P(X_{(N)}|pa(X_{(N)})) \times \ldots \times P(X_{(1)}|pa(X_{(1)})) \tag{2.5}$$

Por questões computacionais podemos deixar a normalização para o final do processo, neste caso, podemos trabalhar com os potenciais ϕ:

$$\phi(X_q, E) = \sum_{X_N} \ldots \sum_{X_1} \phi_{(N)} \times \ldots \times \phi_{(1)} \tag{2.6}$$

Para eliminar uma variável aleatória X_i cujos valores possíveis são $(xi, 1, \ldots, x_{i,k})$ do potencial $\phi(X_1, \ldots, X_i, \ldots, X_n)$ devemos fazer a soma:

$$\sum_{X_i} \phi(X_1, \ldots, X_i, \ldots, X_n) = \phi(X_1, \ldots, x_{i,1}, \ldots, X_n) + \ldots + \phi(X_1, \ldots, x_{i,k}, \ldots, X_n)$$

Utilizaremos a notação $\sum_V$ para indicar a eliminação por marginalização de um determinado potencial de todas a variáveis do conjunto V, sendo que como vale a comutatividade dos potenciais essa notação não é ambígua.

Uma variável X_i aparece como variável condicional para cada um de seus filhos, aparecendo portanto nos respectivos potenciais, e aparece uma única vez como variável não condicional na densidade $P(X_i|pa(X_i))$ e portanto faz parte do potencial correspondente, $\phi_i(X_i, pa(X_i))$.

Sendo $\Phi_{X_{(i)}}$ o conjunto de todos os potenciais com $X_{(i)}$ no seu domínio e por $\prod \phi_j$ o produto de k potenciais: $\phi_1, \ldots, \phi_k$ e considerando a propriedade distributiva dos potenciais, podemos escrever a equação 2.6 de uma maneira computacionalmente mais eficiente:

$$\phi(X_q, E) = \sum_{X_{(N)}} \ldots \sum_{X_{(2)}} \phi_{(N)} \ldots \phi_{(2)} \sum_{X_{(1)}} \prod \Phi_{X_{(1)}} \tag{2.7}$$

Sistematizando esse processo para marginalizar as variáveis requisitadas não pertencentes nem a X_q quanto a X_E temos o Algoritmo de Eliminação de Variáveis, para os potenciais $\Phi = \{\phi_1, \ldots, \phi_n\}$ de um conjunto de variáveis $X = \{X_1, \ldots, X_n\}$, que definimos a seguir:

1. Gere uma ordem de eliminação para as N variáveis requisitadas não questionadas não observadas.

2. Para i de 1 a N:

   (a) Crie uma estrutura de dados B_i, chamada *bucket* contendo:
   - A variável $X_{(i)}$
   - Os potenciais que contenham a variável $X_{(i)}$

   e remova de Φ estes potenciais

   (b) Calcule o *separador* S_i de B_i como o potencial $\phi^{-X_{(i)}} = \sum_{X_{(i)}} \prod \Phi_{X_{(i)}}$. O produto $\prod \Phi_{X_{(i)}}$ é chamado de *cluster* B_i.

   (c) Adicione $\phi^{-X_{(i)}}$ em Φ. Esse resultado é indicado por $\Phi^{-X_{(i)}}$

3. Colete de $\phi^{-X_{(N)}}$ todos os potenciais que contenham variáveis pertencentes a X_q. Multiplique-as e normalize o resultado.

A sequência de operações envolvendo os *buckets* pode ser representada como uma árvore na qual os nós são os *buckets*. A raiz da árvore normalizada será a distribuição $P(X_q, E)$.

2.1.6 Exemplo

Para ilustrar o procedimento anterior e os conceitos previamente estabelecidos, vamos apresentar o exemplo fornecido por Jensen (2001). Considere o GDA correspondente à uma rede Bayesiana representado pela Figura 2.5

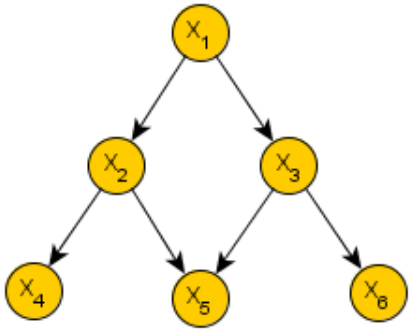


Figura 2.5: Rede Bayesiana Simples

Temos $P(X) = \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6$.

Suponha que queremos calcular $P(X_4) = \sum_{X_1,X_2,X_3,X_5,X_6} \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6$. Para evitar calcular P(X) podemos usar a propriedade distributiva:

$$P(X_4) = \sum_{X_1} \phi_1(X_1) \sum_{X_2} \phi_2(X_2, X_1)\phi_4(X_4, X_2) \sum_{X_3} \phi_1(X_3, X_1) \sum_{X_5} \phi_5(X_5, X_2, X_3) \sum_{X_6} \phi_6(X_6, X_3)$$

Primeiro devemos calcular $\phi'_6(X_3) = \sum_{X_6} \phi_6(X_6, X_3)$ e em seguida multiplicar o resultado por $\phi_5(X_5, X_2, X_3)$ e calcular $\phi'_5(X_2, X_3) = \sum_{X_5} \phi_6(X_5, X_2, X_3)\phi'_6(X_3)$, multiplicar o resultado por $\phi_3(X_3, X_1)$ e assim por diante. Note que podemos aplicar a propriedade distributiva também no cálculo de $\phi'_5(X_2, X_3)$, ou seja, não é preciso multiplicar por $\phi'_6(X_3)$ antes de marginalizar X_3.

A razão de usar a propriedade distributiva é minimizar o tamanho das tabelas com distribuições conjuntas que serão manipuladas. A Figura 2.6 ilustra o graficamente o esquema de cálculo, os nós elípticos são os *buckets* contendo os potenciais, e em um *bucket* os potenciais são multiplicados pelos que chegam, uma variável é marginalizada e o resultado é apresentado em uma caixa retangular para ser enviado para um *bucket* vizinho.

Neste exemplo a ordem de eliminação utilizada foi X_6, X_5, X_3, X_2, X_1, mas, lembrando o que foi visto nas seções anteriores, poderia ser outra, graças à propriedade comutativa dos potenciais, e lembramos também que uma boa escolha de ordem de eliminação é responsável pela eficiência e possibilidade de paralelização do algoritmo.

Considere agora as evidências $X_5 = x^5$ e $X_6 = x^6$ representadas pelos *findings* $\underline{e}_5$ e $\underline{e}_6$, respectivamente. Temos

$$P(X_4, e) = \sum_{X_1,X_2,X_3,X_5,X_6} \phi_1\phi_2\phi_3\phi_4\phi_5\phi_6\underline{e}_5\underline{e}_6$$

e, para este caso a fórmula do teorema 2.1 fica:

$$P(X_4|e) = \frac{P(X_4, e)}{\sum_{X_4} P(X_4, e)}$$

O efeito das evidências no cálculo de $P(X_4, e)$ é que os *findings* são incorporados nos *buckets* com os potenciais correspondentes, o que torna a marginalização de X_5 e X_6 redundantes uma vez que $\phi'_5 = P(X_5 = x^5|X_2, X_3)$ e $\phi'_6 = P(X_6 = x^6|X_3)$. A Figura 2.7 apresenta este novo esquema de eliminação.

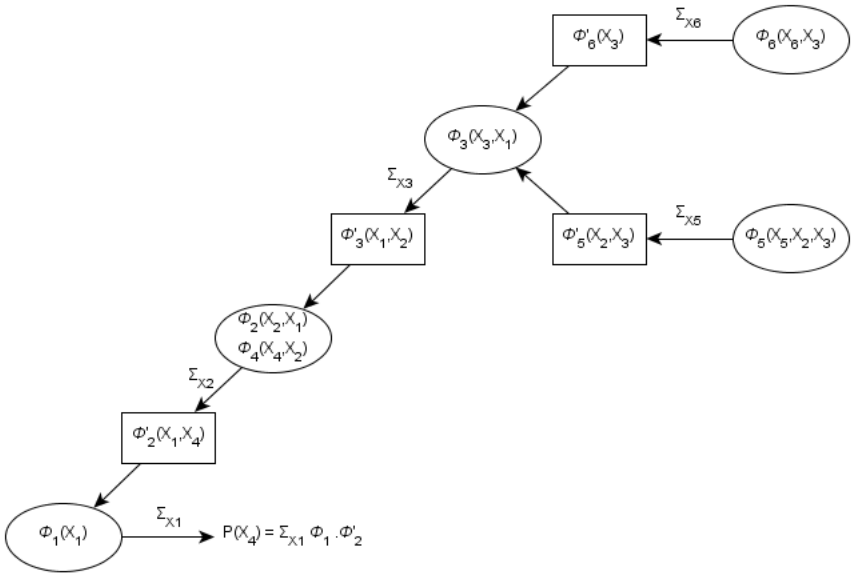


Figura 2.6: Eliminação de variáveis em uma Rede Bayesiana Simples

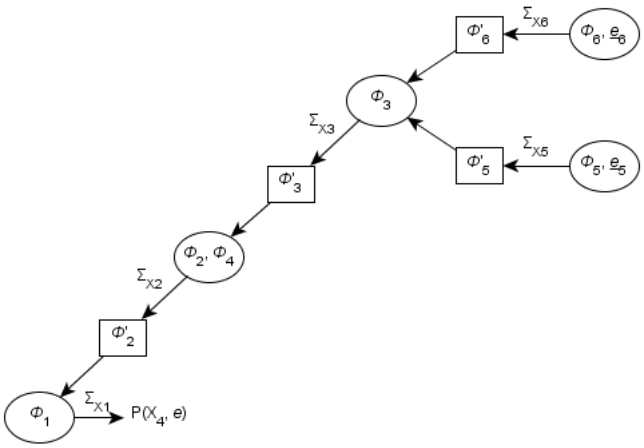


Figura 2.7: Incorporação de evidência na eliminação de variáveis em uma Rede Bayesiana Simples

Para finalizar, apresentamos o conceito de **Grafo de Domínio**. Seja $\Phi = \{1, \ldots, n\}$ um conjunto de potenciais com domínios $dom(\phi_i)$ envolvendo um conjunto de variáveis $X = \{X_1, \ldots, X_n\}$. O Grafo de Domínio para Φ é um grafo não direcionado no qual cada vértice está associado a uma variável de X e os arcos unem vértices que estejam associados a variáveis que pertençam a um mesmo domínio $dom(\phi_i)$. A Figura 2.8 apresenta o grafo de domínio para a rede Bayesiana utilizada neste exemplo. Por construção o grafo de domínio de uma rede Bayesiana é exatamente o grafo moral do GDA que define a topologia da mesma.

Graficamente a operação de eliminação de variáveis corresponde ao processo de eliminação dos vértices do Grafo Moral. A cada variável que é eliminada por marginalização da distribuição, o respectivo vértice é retirado do Grafo Moral. A sequência de eliminação para processo de cálculo da distribuição $P(X_4)$ realizado anteriormente é apresentado pela Figura 2.9 sendo que o arco tracejado é um arco de preenchimento incluído ao longo do processo de eliminação dos vértices. Convém ainda observar que o grau do vértice é igual ao número de variáveis do domínio da variável representada por ele a menos da própria variável e, quando for o caso, das variáveis sobre a quais se deseja fazer a inferência.

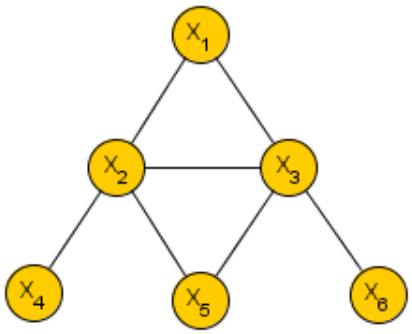


Figura 2.8: Grafo de Domínio de uma Rede Bayesiana Simples

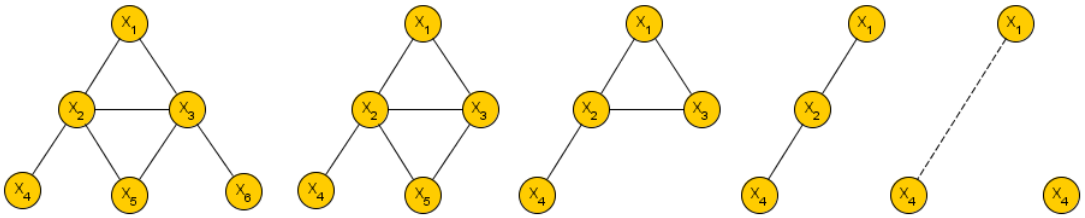


Figura 2.9: Eliminação dos vértices de uma Rede Bayesiana Simples

Capítulo 3

Redes Credais

Certas vezes pode ser útil considerar a presença de incerteza ou imprecisão relacionadas aos valores presentes nas tabelas de probabilidade condicional, seja para analisar a sensibilidade de uma rede Bayesiana, por limitação de recursos para obtenção de parâmetros com a precisão desejada, ou mesmo pela discordância sobre os valores das tabelas de probabilidade condicional pelos agentes de conhecimento envolvidos no processo. Nestes casos pode ser inadequado escolher um único conjunto de parâmetros para as tabelas de probabilidade condicional (da Rocha e Cozman, 2003).

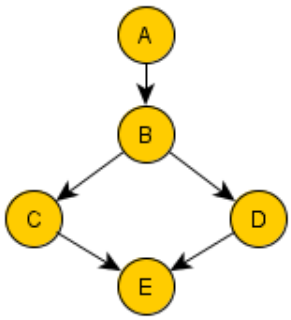
Para tratar esse tipo de situação podemos utilizar uma rede Credal. Em seu trabalho, Cozman (2000b) apresenta uma teoria completa de redes Credais. Neste Capítulo apresentaremos algumas das suas definições que serão úteis para o desenvolvimento do trabalho, entretanto, para um maior arcabouço teórico recomendamos fortemente sua leitura.

3.1 Definições

Uma rede Credal, assim como uma rede Bayesiana, consiste em um GDA em que cada vértice representa uma variável. A diferença entre elas é que na rede Credal temos associado a cada variável um conjunto fechado convexo de medidas de probabilidade, permitindo, por exemplo, que uma variável esteja associada a uma inequação probabilística.

Mais especificamente, um conjunto convexo de medidas de probabilidade K é tal que se P_1 e P_2 pertencem a ele, então $\alpha P_1 + (1-\alpha)P_2$ também pertence, para $\alpha \in [0,1]$. Um conjunto credal $K(X)$ contém medidas de probabilidade induzidas pelas densidades $p(X)$, e um conjunto credal condicional $K(X|B)$ contém medidas de probabilidade induzidas pelas densidades condicionais $p(X|B)$.

A Figura 3.1 mostra um exemplo de rede Credal.



$$
\begin{array}{ll}
0.4 \leq P(A) \leq 0.5, & \\
0.7 \leq P(B|A) \leq 0.8, & 0.1 \leq P(B|A^c) \leq 0.2, \\
P(C|B) = 0.6, & P(C|B^c) = 0.3, \\
P(D|B) = 0.8, & P(D|B^c) = 0.1, \\
P(E|C,D) = 0.3, & P(E|C,D^c) = 0.5, \\
P(E|C^c,D) = 0.6, & P(E|C^c,D^c) = 0.2
\end{array}
$$

Figura 3.1: Exemplo de uma rede Credal

Dado um conjunto credal $K(X)$ e um evento A podemos definir as probabilidades inferior e superior, respectivamente, como:

$$\underline{P}(A) = \min_{P \in K} P(A), \qquad \overline{P}(A) = \max_{P \in K} P(A). \tag{3.1}$$

Sendo que os máximos e mínimos ocorrem somente nos vértices do conjunto credal. Analogamente, podemos definir também as densidades inferior e superior através dos pontos de mínimo e máximo das densidades induzidas pelo conjunto credal. Conjuntos credais são capazes de gerar valores inferiores e superiores para qualquer funcional (esperança, variância, ...)

É possível definir também um conjunto credal condicional $K(X|A)$ em termos da probabilidade condicional $p(X|A)$ e da regra de Bayes. Quando os conjuntos credais condicionais de K são independentes de vértice para vértice da rede, dizemos que a rede é **localmente especificada**. Neste trabalho assumiremos que as redes são todas localmente especificadas.

As consultas em redes Credais podem ser vistas como problemas de otimização, pois são sempre em termos das probabilidades inferiores e superiores uma vez que a distribuição não é única.

Os conceitos de independência e irrelevância de variáveis em redes Credais funcionam exatamente da mesma forma que em redes Bayesianas, sendo que também assumimos a condição de Markov. Adicionalmente podemos definir o conceito de **extensão**, dada uma rede Credal, uma série de conjuntos de medidas de probabilidade satisfaz as restrições impostas pela rede, cada um destes conjuntos é denominado uma extensão. A **extensão forte** de uma rede Credal é o maior destes conjuntos onde os pontos extremos são redes Bayesianas formadas pela rede. O conceito de extensão forte está diretamente ligado ao de independência forte, ou seja, duas variáveis X e Y são independentes quando todo ponto extremo do conjunto credal das distribuições conjuntas $K(X,Y)$ satisfaz independência estocástica entre X e Y, ou seja, $p(X|Y) = p(X)$ e $p(Y|X) = p(Y)$.

Muitos dos métodos de inferência são feitos com base nestes modelos, inclusive os apresentados neste trabalho. A escolha por este tipo de extensão foi motivada pela similaridade com a teoria das redes Bayesianas já estudada até o momento. Nesse contexto, cada vértice de uma extensão forte fatora-se como uma rede Bayesiana, de forma que a rede Credal pode ser considerada uma coleção de redes Bayesianas. Uma discussão sobre outros tipos de extensão, suas vantagens e desvantagens pode ser vista com detalhes em (Cozman, 2000b).

3.1.1 Inferência em Redes Credais

Pode ser de interesse calcular tanto as probabilidades posteriores superiores quanto inferiores. Para o caso das probabilidades a posteriori superiores temos:

$$\bar{P}(X_q = x_q|E) = max\left(\frac{P(X_q = x_q, E)}{P(E)}\right) \tag{3.2}$$

$$= max\left(\frac{\sum_X I_{\{X_q=x_q,E\}}(X_{qE})P(X)}{\sum_X I_E(X_E)P(X)}\right) \tag{3.3}$$

$$= max\left(\frac{\sum_{X\setminus X_{qE},X_{qE}=x_{qE}} P(X)}{\sum_{X\setminus X_E,X_E=x_E}}\right) \tag{3.4}$$

As probabilidades posteriores inferiores podem ser obtidas substituindo as maximizações por minimizações.

Assumiremos também, para simplificar, que todas as combinações de variáveis possuem probabilidade inferior positiva, para evitar trabalhar com probabilidades nulas, um problema que aparece já em redes Bayesianas e é ainda mais difícil de manipular em redes Credais, Além disso, sob estas condições, as extensões fortes admitem d-separação, levando a grandes economias computacionais nos processo de inferência (Cozman, 2000b).

Por conta de um aproveitamento maior do trabalho já desenvolvido, apresentaremos uma transformação de uma rede Credal em uma rede Bayesiana comum dada por Cano *et al.* (1994), chamada transformação CCM, para que seja possível utilizar os algoritmos de inferência em redes Bayesianas já conhecidos, inclusive para eliminar as variáveis irrelevantes. Entretanto, assim como em redes Bayesianas, existem diversos algoritmos para redes Credais, entre eles podemos citar os de enumeração que examinam todos os vértices potenciais das extensões fortes para produzir os valores dos

limites superiores ou inferiores (Cano *et al.*, 1993) os de eliminação de variáveis para redes Credais, que recai em um problema de programação multilinear proposto por de Campos e Cozman (2004), *Branch-and-Bound* (da Rocha e Cozman, 2005) que se utiliza de propriedades específicas das extensões fortes para resolver o problema de programação multilinear, A/R+ (da Rocha e Cozman, 2005), um algoritmo de inferência aproximada para intervalos de probabilidades em redes com topologia de *polytrees*, 2U (Fagiuoli e Zaffalon, 1998), utilizado para redes com variáveis binárias entre outros.

Método de Transformação de uma Rede Credal em Rede Bayesiana

Em linhas gerais, o método consiste em acrescentar variáveis em cada um dos conjuntos credais locais $K(X_i|pa(X_i))$ de modo a transformá-los em uma única rede Bayesiana.

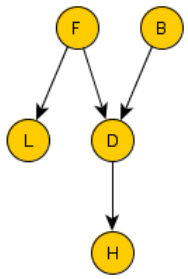
Em mais detalhes, a transformação CCM adiciona uma variável, chamada transparente, X_i' para cada uma das variáveis X_i da rede. Esta nova variável tem o vértice X_i como seu único filho e assume m valores inteiros, onde m é cada uma das possíveis maneiras que o conjunto credal local $K(X_i|pa(X_i))$ pode ser combinado, levando em conta seus valores extremos. X_i é então relacionada com uma família de densidades de probabilidade condicionais:

$$P(X_i|pa(X_i), X_i' = l) = P_l(X_i|pa(X_i)) \quad l = 1, \ldots, m.$$

Como se sabe que os valores extremos de $P(X_q = x_q|E)$ são atingidos nos vértices dos conjuntos credais locais, uma inferência tradicional sobre esta rede Bayesiana produz o resultado desejado (de Campos, 2006). Como cada variável transparente tem tantas categorias quanto número de vértices do conjunto associado pode ser um tanto inconveniente trabalhar com redes grandes, no sentido em que as redes Bayesianas geradas ficarão muito extensas. Neste caso, pode ser vantajoso trabalhar com algum método de inferência específico para (grandes) redes Credais.

Exemplo

Para ilustrar os conceitos anteriores, incluindo o processo de inferência em uma rede Credal através da transformação CCM, considere a seguinte o problema do cachorro *(Dog Problem)*, uma rede de variáveis binárias bastante utilizada em literatura de redes Bayesianas (Charniak, 1991) em uma versão credal apresentada na Figura 3.2.



$$\begin{array}{ll}
P(F) \in [0,15;\ \ 0,25], & P(B) \in [0,01;\ \ 0,05] \\
P(L|F) \in [0,8;\ \ 0,9], & P(L|F^c) \in [0,01;\ \ 0,1] \\
P(D|F,B) \in [0,2;\ \ 0,3], & P(D|F,B^c) \in [0,9;\ \ 0,95] \\
P(D|F^c,B) \in [0,05;\ \ 0,1], & P(D|F^c,B^c) \in [0,6;\ \ 0,7] \\
P(H|D) \in [0,6;\ \ 0,7], & P(H|D^c) \in [0,01;\ \ 0,05]
\end{array}$$

Figura 3.2: Dog Problem em sua versão credal

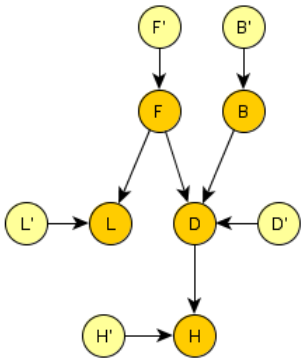
Aplicando o CCM na rede, obtemos a rede Bayesiana apresentada na Figura 3.3. Mantendo a estrutura binária da rede, consideraremos o primeiro valor das variáveis transparentes como o valor inferior do intervalo e seu complementar como o valor superior.

Podemos fazer inferências computando limites inferiores e superiores da seguinte maneira, como no *JavaBayes* (Cozman, 2001):

1. Execute algum algoritmo de redes Bayesianas, como de eliminação de variáveis na nova rede, sem especificar os valores das variáveis transparentes, de forma que o resultado seja dado condicionado aos seus valores;

2. Inicialize os limites inferior e superior das marginais posteriores com 1 e 0, respectivamente;

3. Para todas as combinações de valores das variáveis transparentes armazene os valores que forem menores que o limite inferior e maiores que o superior da distribuição posterior marginal.



$$
\begin{array}{ll}
P(F|F') = 0.15, & P(F|F'^c) = 0.25 \\
P(B|B') = 0.01, & P(B|B'^c) = 0.05 \\
P(L|F, L') = 0.8, & P(F|F, L'^c) = 0.9 \\
P(L|F^c, L') = 0.01, & P(F|F^c, L'^c) = 0.1 \\
P(D|F, B, D') = 0.2, & P(D|F, B, D'^c) = 0.3 \\
P(D|F, B^c, D') = 0.9, & P(D|F, B^c, D'^c) = 0.95 \\
P(D|F^c, B, D') = 0.05, & P(D|F^c, B, D'^c) = 0.1 \\
P(D|F^c, B^c, D') = 0.6, & P(D|F^c, B^c, D'^c) = 0.7 \\
P(H|D, H') = 0.6, & P(H|D, H'^c) = 0.7 \\
P(H|D^c, H') = 0.01, & P(H|D^c, H'^c) = 0.05
\end{array}
$$

Figura 3.3: Dog Problem em sua versão credal após a transformação CCM

Este exemplo mostra também como esta transformação implica em um aumento da quantidade de variáveis na rede Bayesiana associada, ressaltando a importância de eliminar previamente as variáveis irrelevantes para a inferência.

Capítulo 4

Matrizes Esparsas e Decomposição de Matrizes

Neste Capítulo primeiramente apresentaremos alguns conceitos de matrizes e álgebra linear de maneira bastante próxima ao apresentado em (Stern, 1994) e (Stern, 2008) e também como estes conceitos podem ser aplicados para a fatoração de matrizes esparsas como é mostrado também em (Colla, 2007) utilizando teoria dos grafos, todavia nos limitaremos a apresentar de maneira simples os resultados obtidos por esses autores, e para um maior formalismo, aprofundamento teórico e exemplos adicionais recomendamos a leitura dos originais. A segunda parte usa o que foi apresentado para mostrar um procedimento análogo que pode ser utilizado para inferência em redes Credais.

Uma matriz **esparsa** é uma matriz cuja grande maioria das entradas possui valor igual a zero, possuindo portanto uma pequena quantidade de elementos não nulos (ENNs). As vantagens computacionais de se levar em conta as características deste tipo de matriz são basicamente que existem maneiras bastante econômicas de armazená-las e também que o número de operações aritméticas necessário para operá-las fica reduzido por conta dos elementos nulos (Stern, 1994).

Notação

Para evitar ambiguidades, vamos estabelecer a notação que utilizaremos para expor os conceitos de Álgebra Linear que virão a seguir. Matrizes serão representadas por letras do alfabeto maiúsculas, $A, B, C, \ldots$ e vetores por letras minúsculas, $a, b, c, \ldots$ Em ambos, os índices de linha serão subscritos e de coluna serão sobrescritos, sempre à direita. Desta forma: A_i^j corresponde ao elemento da i-ésima linha e j-ésima coluna; b_i é o elemento da i-ésima linha do vetor coluna e c^j o elemento da j-ésima coluna do vetor linha. Índices sobrescritos ou subscritos à esquerda serão utilizados para indicar matrizes diferentes. Um vetor é uma matriz onde uma de suas dimensões é igual a 1.

4.1 Classificação de Matrizes

Alguns tipos especiais de matrizes possuem uma terminologia específica que facilitam sua identificação, como é o caso das matrizes esparsas. Apresentaremos a seguir mais alguns tipos particulares de matrizes.

Uma matriz é dita **quadrada** se tem o mesmo número de linhas e colunas, ou seja, $m = n$. Numa matriz quadrada A de ordem $n \times n$, chama-se de *diagonal principal* os elementos a_{ij} onde $i = j$, para $i = 1, \cdots, n$.

Uma matriz é dita **triangular** se os elementos acima ou abaixo da diagonal principal são zero. Caso os elementos acima da diagonal principal sejam zero, ela é chamada **triangular inferior**, caso os elementos abaixo da diagonal principal sejam zero, ela é dita **triangular superior**.

Uma matriz **diagonal** é uma matriz quadrada e que possui todos os elementos fora da diagonal principal iguais a zero. Uma matriz **identidade** I_n é uma matriz diagonal $n \times n$ e que possui todos

os elementos da diagonal principal iguais a 1.

A matriz **transposta** é obtida através da troca de linhas pelas colunas em uma determinada matriz fazendo a correspondência dos índices: $a_i^j = a_j^i$. Representaremos a matriz transposta de uma matriz M por M'. Uma matriz **simétrica** é aquela que é igual à sua transposta.

Uma matriz quadrada A^{-1} é dita **inversa** de uma matriz quadrada A se o produto entre as matrizes é a matriz identidade, ou seja, se $AA^{-1} = A^{-1}A = I$. Se A^{-1} não puder ser encontrada, A é dita não invertível, singular ou degenerada. Uma matriz quadrada é **ortogonal** quando possui sua inversa igual à transposta.

Uma matriz A é **positiva-definida** se $z'Az > 0, \forall z \in \mathbb{R}^n$.

4.2 Decomposição de Matrizes

Em algumas situações como na resolução de sistemas lineares, no cálculo de matriz inversa, etc, pode ser interessante escrever uma matriz A como o produto de duas ou mais matrizes, por exemplo $A = BC$, onde B e C possuem algumas características especiais.

Além disso, Barone Jr (1988) mostra que no contexto das transformações lineares, a escolha dos vetores da base modifica a estrutura da matriz, portanto a escolha de uma base conveniente pode levar a uma estrutura de matriz que facilite os cálculos. A forma mais simples de matriz para a base é a matriz diagonal, pois com ela a operação de multiplicação de matrizes é simplificada, devido a matriz diagonal possuir números diferentes de zero apenas na diagonal principal.

As seções a seguir mostram algumas dessas decomposições e ferramentas necessárias para elas de maneira bastante sucinta.

4.2.1 Permutações e Operações Elementares

Permutar linhas ou colunas em uma matriz identidade nos fornece uma matriz ortogonal chamada **matriz de permutação** . Dada uma matriz de permutação de linhas da matriz identidade P e uma matriz de permutação de colunas da matriz identidade Q, os vetores correspondentes de linhas e colunas permutadas são respectivamente:

$$p = \left(P \cdot \begin{bmatrix} 1 \\ 2 \\ \vdots \\ m \end{bmatrix} \right)' \quad e \quad q = \begin{bmatrix} 1 & 2 & \dots & n \end{bmatrix} \cdot Q$$

Dada uma matriz A, podemos realizar uma permutação de linhas de modo a obter a matriz permutada $\tilde{A}$ através da multiplicação à esquerda pela matriz de permutação de linhas correspondente. Analogamente realizar uma permutação de colunas equivale a multiplicar A à direita pela matriz de permutação de colunas correspondente. Dessa forma, se p e q forem respectivos vetores índices de linhas e colunas permutados teremos:

$$\tilde{A}_i^j = (PA)_i^j = A_{p(i)}^j$$

$$\tilde{A}_i^j = (AQ)_i^j = A_i^{q(j)}$$

Uma **permutação simétrica** de uma matriz quadrada A é da forma $\tilde{A} = PAP'$ onde P é uma matriz de permutação. Se A for simétrica, então o resultado dessa permutação será também uma matriz simétrica.

O exemplo a seguir ilustra os efeitos de uma permutação nos elementos de uma matriz. Sejam:

$$A = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}, P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, Q = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad e \quad p = q = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}$$

Temos:

$$PA = \begin{bmatrix} 31 & 32 & 33 \\ 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix} \quad e \quad AQ = \begin{bmatrix} 13 & 11 & 12 \\ 23 & 21 & 22 \\ 33 & 31 & 32 \end{bmatrix}$$

São **operações elementares** de uma matriz:

1. Multiplicar uma linha por um escalar não nulo;
2. Adicionar a uma linha da matriz uma outra linha da matriz;
3. Subtrair de uma linha da matriz uma outra linha da matriz multiplicada por um escalar não nulo.

Uma **matriz elementar** E é obtida através da aplicação de uma operação elementar na matriz identidade. Realizar um operação elementar sobre uma matriz qualquer A equivale a multiplicá-la à esquerda pela matriz elementar correspondente.

Veremos a seguir que permutações e operações elementares podem ser usadas para fatorar uma matriz em matrizes com características especiais, por exemplo, triangulares.

4.2.2 Método de Eliminação de Gauss

O método de Gauss utiliza o terceiro tipo de operação elementar para triangularizar (inferiormente) uma matriz, buscando utilizar os elementos de uma dada linha para anular os elementos das linhas abaixo dela que estão abaixo da diagonal principal. Em cada etapa esta linha é chamada de **linha pivô**, seu elemento diagonal é o **elemento pivô** e os fatores de multiplicação são chamados **multiplicadores**. ${}^kA$ é a matriz obtida na *k-ésima* etapa da eliminação e ${}^kM$ é a matriz de multiplicadores correspondente, sendo de ${}^kM_i^j$ é o multiplicador utilizado para anular o elemento da *i-ésima* linha e *j-ésima* coluna se $j \leq k$ e $i > j$ e vale zero caso contrário.

Como os elementos nulos em ${}^kA$ são justamente os elementos não nulos em ${}^kM$ é possível poupar espaço computacional armazenando simultaneamente essas duas matrizes em uma matriz $[{}^kA{+}{}^kM]$.

Pivoteamento

Denominamos **pivoteamento** o processo de troca de linhas para alteração do pivô a ser utilizado em determinada etapa da eliminação. Ele pode se fazer necessário caso apareça algum elemento nulo na diagonal da matriz ${}^kA$ (e nesta coluna houver algum elemento não nulo), e pode ser bastante útil para aumentar a estabilidade numérica do processo, colocando em cada etapa o elemento de máximo módulo de modo que todos os multiplicadores tenham módulo menor que 1, controlando a propagação de erros de arredondamento em uma estratégia denominada **pivoteamento parcial** (Stern, 1994). Para computacionalmente não ter que efetivamente trocar as linhas da matriz é possível armazenar um vetor de índices de pivoteamento e realizar as operações na matriz considerando os valores desse vetor.

A seguir mostraremos a matriz $[{}^kA + {}^kM]$ juntamente com o vetor de permutação nas etapas de um processo de eliminação de uma matriz 4×4 com pivoteamento.

$$\begin{bmatrix} 2 & 1 & 9 & -1 \\ 1 & 3 & 7 & 7 \\ 2 & 8 & 4 & 2 \\ 3 & 9 & 6 & 6 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \rightarrow \begin{bmatrix} 3 & 9 & 6 & 6 \\ 1/3 & 0 & 5 & 5 \\ 2/3 & 2 & 0 & -2 \\ 2/3 & -5 & 5 & -5 \end{bmatrix} \begin{matrix} 4 \\ 2 \\ 3 \\ 1 \end{matrix} \rightarrow \begin{bmatrix} 3 & 9 & 6 & 6 \\ 2/3 & -5 & 5 & -5 \\ 2/3 & -2/5 & 2 & -4 \\ 1/3 & 0 & 5 & 5 \end{bmatrix} \begin{matrix} 4 \\ 1 \\ 3 \\ 2 \end{matrix} \rightarrow \begin{bmatrix} 3 & 9 & 6 & 6 \\ 2/3 & -5 & 5 & -5 \\ 1/3 & 0 & 5 & 5 \\ 2/3 & -2/5 & 2/5 & -6 \end{bmatrix} \begin{matrix} 4 \\ 1 \\ 2 \\ 3 \end{matrix}$$

4.2.3 Fatoração LU

A decomposição LU fatora uma matriz quadrada A em um produto de uma matriz triangular inferior por uma triangular superior e é obtida através do método de eliminação de Gauss com

pivoteamento, onde L é a matriz triangularizada e U é a última matriz dos multiplicadores obtida somada com a identidade (Lema da Fatoração). Temos ainda que se A for inversível então existe uma matriz de permutação de linhas P de modo que $\tilde{A} = PA$ é triangularizável pelo método de Gauss e $\tilde{A} = LU$ (Stern, 1994). Se A for positiva-definida não será necessário realizar o pivoteamento.

4.2.4 Fatoração de Cholesky

Um caso particular da decomposição LU quando a matriz $S_{n\times n}$ é simétrica e positiva-definida é a *fatoração de Cholesky* $S = CC'$, onde C é uma matriz triangular inferior com elementos positivos na diagonal e $C = LD^{\frac{1}{2}}$, onde D é uma matriz com os elementos da diagonal de U. A vantagem da fatoração de Cholesky é que ela efetua cerca de metade das operações aritméticas quando comparada à decomposição LU.

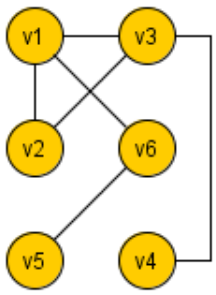
Algoritmo

O algoritmo de fatoração de Cholesky é semelhante ao da fatoração LU, com a diferença que já incorpora a simetria da matriz. Os passos são os seguintes:

$$\begin{aligned}
l_{11} &= \sqrt{a_{11}} \\
l_{j1} &= \frac{a_{j1}}{l_{11}} && j = 2, \ldots, n \\
l_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} && i = 2, \ldots, n-1 \\
l_{ji} &= \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{ii}} && i = 2, \ldots, n-1 \;\; e \;\; j = i+1, \ldots, n
\end{aligned} \tag{4.1}$$

4.3 Preservação da Esparsidade

Pela natureza das operações na eliminação de Gauss, alguns valores que eram zero na matriz original podem acabar assumindo valores diferentes de zero após a triangularização, o que chamamos de **preenchimento**. Para ilustrar, a Figura 4.1 apresenta um grafo, o resultado da fatoração de Cholesky em sua respectiva matriz de adjacência e em outras duas permutações simétricas, os números representam os vértices, 'x' representa um arco conectando dois vértices no grafo original e '*' são as posições nulas que foram preenchidas durante a fatoração.



$$\begin{bmatrix} 1 & x & x & & & x \\ x & 2 & x & & & * \\ x & x & 3 & x & & * \\ & & x & 4 & & * \\ & & & & 5 & x \\ x & * & * & * & x & 6 \end{bmatrix} \begin{bmatrix} 1 & x & x & & & \\ x & 3 & & x & x & \\ x & * & 6 & * & * & x \\ x & x & * & 2 & * & * \\ & x & * & * & 4 & * \\ & & x & * & * & 5 \end{bmatrix} \begin{bmatrix} 5 & & & x & & \\ & 4 & & & x & \\ & & 2 & & x & x \\ x & & & 6 & & x \\ & x & x & & 3 & x \\ & & x & x & x & 1 \end{bmatrix}$$

Figura 4.1: Fatoração de Cholesky para diferentes permutações de uma matriz de adjacência de um grafo

Podemos ver que o número de preenchimentos varia de acordo com a permutação escolhida. Dessa forma, é interessante buscar ao longo do processo escolhas de pivôs que minimizem esse preenchimento. Através da teoria dos grafos, em que uma abordagem inicial pode ser encontrada no

Apêndice A, incluindo os grafos de eliminação referentes à segunda matriz da Figura 4.1, é possível encontrar maneiras de preservar a esparsidade durante a fatoração.

No contexto de fatoração de matrizes, os grafos de eliminação E_i são aqueles que possuem como matriz de adjacência as submatrizes ${}^kA$ do processo de fatoração, e o grafo preenchido $P = (V, F)$ incorpora em F os arcos originais e os que surgiram na eliminação. Dessa forma, dada uma ordem de eliminação q, eliminar a *j-ésima* coluna na fatoração de Cholesky de $A_{q(i)}^{q(j)} = LL'$ preenche as posições que correspondem aos arcos preenchidos em F pela eliminação do vértice $q(j)$

Dada uma matriz de adjacência A, um grafo de eliminação $G = (V, E)$, uma ordem de eliminação q e o respectivo grafo preenchido, o conjunto de índices de linha de elementos não nulos (ENNs) na coluna j do fator de Cholesky $L^j|QAQ' = LL'$ é dado por:

$$ENN(L^j) = \{i|i > j \wedge \{q(i), q(j)\} \in F\} + \{j\} \tag{4.2}$$

Definimos a **Árvore de Eliminação** H como:

$$\Gamma_H^{-1}(j) = h(j) = \begin{cases} j, & \text{se } ENN(L^j) = \{\text{j}\} \\ \min\{i > j \mid i \in ENN(L^j)\}, & \text{caso contrário} \end{cases} \tag{4.3}$$

Ou seja, pai do nó j em H, é simplesmente o primeiro ENN e não diagonal na *j-ésima* coluna da matriz L da fatoração de Cholesky.

Para ilustrar, construímos as árvores de eliminação para cada uma das matrizes do exemplo anterior na mesma ordem em que aparecem e apresentamos na Figura 4.2.

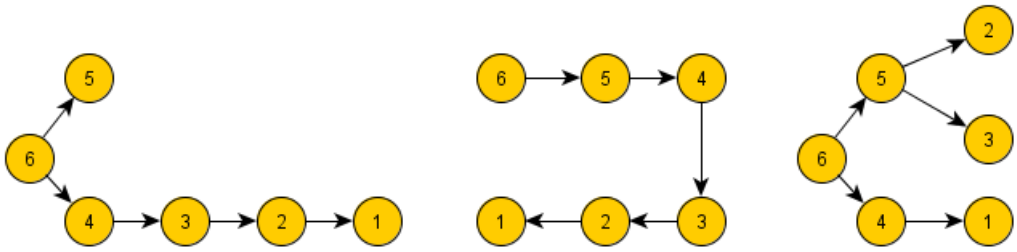


Figura 4.2: Árvores de Eliminação para algumas matrizes de adjacência

Temos ainda alguns resultados importantes apresentados em (Stern, 1994) como:

- Dado $i > j$, $i \in ENN(L^j) \Rightarrow j \in \bar{\Gamma}(i)$ onde $\bar{\Gamma}(i)$ representa todos os descendentes do nó i (função de descendência). Pondo de outra forma, qualquer índice de linha abaixo da diagonal na coluna j de L é um ascendente de j na árvore de eliminação.

- A árvore de eliminação retrata as dependências entre as colunas para o processo de fatoração da matriz, podemos eliminar a coluna j de A se e somente se já tivermos eliminado todos os descendentes de j na árvore de eliminação.

- Caso esteja disponível um processamento paralelo é possível eliminar simultaneamente todas as colunas de um mesmo nível da árvore de eliminação, das folhas até a raiz, o que permite calcular antecipadamente quantas etapas serão necessárias para a fatoração.

Temos ainda o **Teorema da Fatoração simbólica** (Stern, 1994) que garante uma forma mais eficiente de se obter o grafo preenchido $P = (V, F)$ e a árvore de eliminação H uma vez conhecidos o padrão de esparsidade da matriz original $G = (V, E)$ e a ordem de eliminação q. Usaremos a versão simplificada dos grafos de eliminação G_j^* em que ao eliminar o vértice $q(j)$ apenas os lados incidentes ao vizinho mais próximo de ser eliminado são preenchidos. Os seguintes passos garantem que $F^* = F$ e $h^* = h$:

1. $G_j^* = (V_j, E_j^*)$,

2. $E_1^* = E$,
3. $h^*(j) = min\{i > j \mid \{q(j), q(i)\} \in E_j^*\}$
4. $E_{j+1}^* = \{\{a, b\} \in E_j^* \mid \bar{q}(a), \bar{q}(b) > j\} \cup \{\{q(h(j)), v\}, v \mid \bar{q}(v) \geq j \wedge \{q(j), v\} \in E_j^*\}$
5. $F^* = \cup_1^n E_j^*$

4.3.1 Escolha da Ordem de Eliminação

Uma vez que mostramos a importância de escolher uma boa ordem de eliminação para evitar o preenchimento da matriz é preciso um bom método para escolha dessa ordem, que consideramos conhecida até o momento. Isto será feito nesta Seção.

Considere em um grafo $G = (V, E)$ um separador S que divide $V - S$ em componentes conexas $V_1, V_2, \ldots, V_k$. Podemos considerar em cada uma dessas componentes um novo separador que as reparte e assim recursivamente. Esse processo pode ser representado pela **Árvore de dissecção** D com raiz S, componentes separadas por S ou separadores dentro dele como filhas de S e com as componentes não repartidas como folhas dessa árvore.

Dada uma árvore H de raiz r, definimos um **pós-ordem** como uma ordem $\tilde{q}$ de seus vértices, que lista os vértices de cada uma das árvores de $H - r$ recursivamente em pós-ordem até chegar em r. Se $\tilde{q}$ é uma pós-ordem de uma árvore de dissecção D de G, temos uma **ordem de dissecção** q ao substituir nela cada nó d de D pelos vértices de G em d.

Considere um grafo G e sua ordem de dissecção q. A eliminação de um vértice $v \in d$ somente pode incluir arcos de preenchimento entre os vértices de G pertencentes ao nó d, entre vértices de d e vértices em nós ancestrais de d em D ou ainda entre vértices em ascendentes de d (Stern, 1994), (Colla, 2007).

Para ilustrar os conceitos apresentados considere o seguinte exemplo, a Figura 4.3 apresenta um grafo e sua respectiva árvore de dissecção.

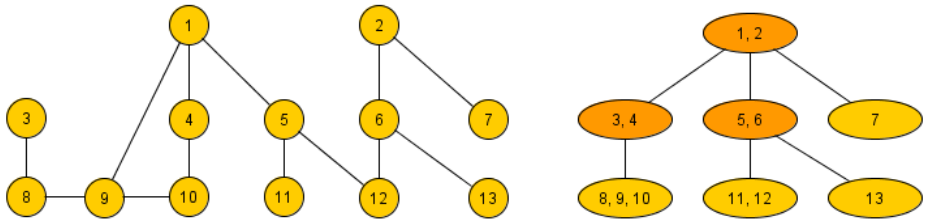


Figura 4.3: Exemplo de um grafo e sua respectiva árvore de dissecção

O procedimento de eliminação simétrica deste grafo para a seguinte ordem de eliminação: $q = [9, 8, 10, 3, 4, 11, 12, 13, 5, 6, 7, 1, 2]$, produz a matriz com as posições correspondentes à arcos de preenchimento indicadas com '*':

$$
\begin{array}{c}
1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13
\end{array}
\begin{bmatrix}
9 & x & x & & & & & & & & x & \\
x & 8 & * & x & & & & & & & * & \\
x & * & 10 & x & * & x & & & & & * & \\
 & x & * & 3 & * & & & & & & * & \\
 & & x & * & 4 & & & & & & x & \\
 & & & & & 11 & & & x & & & \\
 & & & & & & 12 & & x & x & & \\
 & & & & & & & 13 & & x & & \\
 & & & & & x & x & & 5 & * & x & \\
 & & & & & & x & x & * & 6 & * & x \\
 & & & & & & & & & & 7 & x \\
x & * & * & * & x & & & & x & * & 1 & * \\
 & & & & & & & & & x & x & * & 2
\end{bmatrix}
$$

Levando em conta os arcos de preenchimento que podem surgir em G, ou, analogamente, posições que podem ser preenchidas na matriz QAQ' para que uma ordem de eliminação minimize a região de possível preenchimento e considerando o separador S_k é desejável que ele seja "pequeno e balanceado"ou, em outras palavras:

- $\#S_k$ seja o menor possível;
- As sub-árvores tenham aproximadamente o mesmo número de vértices de G.

Heurística de Busca em Largura

Uma busca em largura, BEL, varre um grafo G partindo de uma raiz $v \in V$, em seguida visitando os seus vizinhos, e os vizinhos de seus vizinhos e assim sucessivamente até que o alvo seja encontrado. Ela particiona os vértices de G a partir da raiz em níveis $L_0, L_1, \ldots, L_k$ dados por:

$$L_0 = \{v\}, \qquad L_{i+1} = adj(L_i) - L_{i-1} \tag{4.4}$$

Podemos definir alguns conceitos como a **profundidade** do nível L_i, que é i e a **largura** do nível L_i que é $\#L_i$, a quantidade de vértices que este nível possui. A profundidade máxima e largura máxima entre os níveis da BEL definem respectivamente sua profundidade e largura.

A heurística da BEL busca um separador balanceado $S \subseteq L_i$ tomando

$$i \approx \frac{k}{2} \text{ ou } i \mid \sum_{j=1}^{i-1} \#L_j < \frac{n}{2} \wedge \sum_{j=i+1}^{n} \#L_j < \frac{n}{2}$$

É interessante que a largura da BEL seja pequena, maximizando sua profundidade, e a BEL vai justamente buscar uma raiz v que gere estas características. A **distância** em G de um vértice v a um vértice w, $dist(v, w)$, é o comprimento, ou número de lados, do caminho mais curto entre ambos os vértices. A **excentricidade** de um vértice v é a distância máxima entre ele e qualquer outro vértice, $exc(v) = max_{w \in V} dist(v, W)$. Chamamos de **periférico** um vértice de máxima excentricidade e esta é o **diâmetro** de G.

Uma BEL com raiz v tem profundidade igual à excentricidade de v, desta forma, é interessante iniciar a BEL por um vértice periférico. Encontrar este vértice é computacionalmente difícil e uma solução é utilizar a **Heurística de Gibbs** para encontrar um vértice **quase-periférico** como proposto em (Stern, 1994).

Heurística de Gibbs

Dado $G = (V, E)$:

1. Escolha como raiz um vértice v de grau mínimo;
2. Forme os níveis da BEL com raiz v, $L_0, L_1, \ldots, L_k$, particione o nível mais profundo em suas l componentes conexas, $L_k = \cup_{j=1}^{l}$, e tome um vértice de grau mínimo, v_j , em cada componente.
3. Para $j = 1 : l$

   Tome v_j como nova raiz e encontre os níveis da BEL $L_1, \ldots, L_{k'}$
   Até que $k' > k$ ou $j = l$;
4. Se o passo anterior terminou com $k' > k$ volte ao passo 2. Caso contrário a raiz atual é um vértice quase-periférico.

Para ilustrar, tomemos o grafo apresentado na Figura 4.3. A heurística de Gibbs aponta o vértice 3 como quase-periférico e sua BEL produz a árvore representada na Figura 4.4

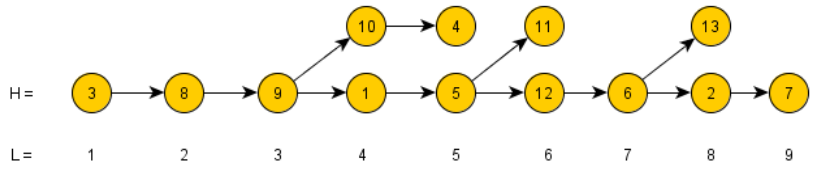


Figura 4.4: Exemplo de árvore gerada através das Heurísticas de Gibbs e BEL

Escolhendo como separador inicial $S_1 = L_5 = \{4, 5\}$ obtemos as seguintes componentes conexas: $C_1 = \{1, 3, 8, 9, 10\}$, $C_2 = \{11\}$ e $C_3 = \{2, 6, 7, 12, 13\}$. Escolhendo $S_2 = L_3 = \{9\}$ como separador para C_2 e $S_3 = L_7 = \{6\}$ como separador para C_3 obtemos a árvore de dissecção apresentada na Figura 4.5.

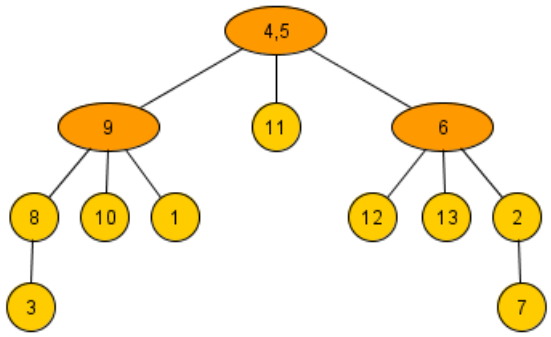


Figura 4.5: Árvore de dissecção gerada após as heurísticas de Gibbs e BEL

Listando os vértices obtemos a seguinte ordem por dissecção: $q = [3, 8, 1, 10, 9, 12, 13, 2, 7, 6, 4, 5]$.

4.4 Estruturas

Existem ainda alguns tipos particulares de matrizes esparsas que permitem obter vantagem computacional na triangularização. Estudaremos dois tipos formados por blocos de elementos não nulos dispostos de forma regular, são aquelas formadas por uma estrutura de forma **angular blocada** (FAB) por linhas e por colunas (respectivamente FABL e FABC). O bloco ${}^r_sB$ tem dimensão $m(r) \times n(s)$ e fora deles todos os elementos são nulos. Triangularizar estas matrizes corresponde a triangularizar os blocos diagonais, sendo que durante a triangularização nenhum elemento não nulo é criado fora dos blocos. A seguir apresentamos matricialmente estas duas estruturas:

$$\begin{bmatrix} {}^1_1B & 0 & \dots & 0 \\ 0 & {}^2_2B & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & & {}^h_hB \\ {}^1_hL & {}^2_hL & \dots & {}^h_hL \end{bmatrix} \begin{bmatrix} {}^1_1B & 0 & \dots & 0 & {}^h_1C \\ 0 & {}^2_2B & & 0 & {}^h_2C \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & {}^{h-1}_{h-1}B & \\ 0 & 0 & \dots & 0 & {}^h_hC \end{bmatrix}$$

4.4.1 Fatoração em Matriz de Estrutura Angular Blocada

Conforme dito anteriormente, é possível fazer a fatoração de Cholesky de uma matriz com estrutura angular blocada sem adicionar elementos não nulos fora dos blocos. Stern e Vavasis (1993) mostram ainda que, devido à essa estrutura de blocos, a fatoração não só permite preservar a esparsidade como também permite que várias operações sejam realizadas em paralelo. Vamos apresentar o algoritmo dado por eles para a fatoração blocada de Cholesky para uma rede computacional de $h + 1$ nós. Para maiores detalhes, demonstrações e informação sobre consumo computacional recomendamos a leitura do material original.

Fatoração de Cholesky Blocada

Caso não haja nenhuma linha ou coluna residual na forma blocada é possível realizar a fatoração de Cholesky de forma independente em cada um dos blocos. No caso de colunas residuais devemos calcular $A'A = U'U$, sendo que U vem da decomposição de Cholesky. Como estamos trabalhando com matrizes simétricas, não é necessário fazer a multiplicação pois colunas residuais implicam na presença de linhas residuais.

O algoritmo paralelo de fatoração de Cholesky blocado é o seguinte:

1. No nó k variando de 1 a h compute os blocos $(B^k)'B^k$, $(B^k)'C^k$ e $(C^k)'C^k$;
2. Envie $(C^k)'C^k$ do nó k para o nó 0, onde acumulamos $Z^0 = \sum_1^h (C^k)'C^k$;
3. No nó k compute a fatoração de Cholesky parcial, eliminando as primeiras $n(k)$ colunas da matriz blocada:
$$\begin{bmatrix} (B^k)'B^k & (B^k)'C^k \\ (C^k)'B^k & 0 \end{bmatrix}$$
obtendo
$$\begin{bmatrix} V^k & W^k \\ 0 & Z^k \end{bmatrix}$$
4. Envie Z^k do nó k para o nó 0, onde acumulamos s $Z = \sum_0^h Z^k$;
5. No nó 0 fatore o bloco sudeste $S = chol(Z)$ pela fatoração de Cholesky padrão.

Etapa simbólica da fatoração de Cholesky Blocada

Na Subseção anterior apresentamos um método paralelo de fatoração para matrizes blocadas. Existem alguns algoritmos para realizar a fatoração simbólica de Cholesky em paralelo, como por exemplo a metodologia do algoritmo de Grondelle (1999) que gera uma árvore de eliminação e um grafo preechido para a fase simbólica da fatoração de Cholesky percorrendo a matriz de adjacência por ondas. Para o caso de matrizes blocadas, o formato da matriz induz a maneira de paralelizar o problema, uma vez que a fatoração de um bloco não gera arcos de preenchimento no outro e é possível adaptar o algoritmo paralelo de fatoração de Cholesky blocado para uma fatoração simbólica

Vamos desconsiderar do algoritmo de fatoração de Cholesky as operações que não introduzem ou destroem elementos não nulos, ou seja, as raízes quadradas e divisões. Dessa forma, assim como na eliminação sequencial, nos preocuparemos apenas com as adições, utilizando uma notação de grafos, árvores de eliminação e o teorema da fatoração simbólica, como no caso da fatoração numérica sequencial.

Precisamos fazer a adaptação no algoritmo para calcular em cada bloco a fatoração parcial e posteriormente unificar os resultados obtidos nos elementos correspondentes às colunas residuais (ou linhas residuais, já que a matriz é simétrica) na fatoração. O método consiste basicamente nos seguintes passos:

1. Para cada bloco k considere um subgrafo com os vértices deste bloco e das colunas residuais e as ligações entre eles;
2. Para cada bloco k compute a etapa simbólica da fatoração de Cholesky, considerando na eliminação os elementos dos blocos e das colunas residuais e guardando o resultado na sub-árvore$[k]$;
3. Faça a fatoração dos nós correspondentes às colunas residuais e armazene o resultado em uma sub-árvore principal;

4. Combine os resultados adionando na sub-árvore principal os filhos das sub-árvores, respeitando as ligações já existentes.

É importante ressaltar que esse algoritmo assume que o grafo é conexo e que a matriz de adjacencia correspondente está na FAB, mas que dentro de cada bloco os elementos não precisam ser todos diferentes de zero.

4.4.2 Obtendo a forma blocada de uma matriz

Uma vez que sabemos que matrizes em forma blocada tem sua fatoração facilitada, pode ser interessante colocar uma matriz esparsa nesta forma antes de prosseguir com a fatoração (ou eliminação de variáveis, no caso de um GDA correspondente a uma rede Bayesiana). Apresentaremos a seguir um algoritmo apresentado por Stern (1992) que usa um método heurístico bastante conhecido, **Simulated Annealing**, com a adição de penalidade relacionada a temperatura na função objetivo para obter uma permutação que transforme uma matriz esparsa em matriz blocada que, dependendo da forma da matriz, pode ser mais indicado para escolher uma boa ordem de eliminação que o algoritmo apresentado anteriormente em 4.3.1.

Simulated Annealing com função de penalidade relacionada à temperatura

Annealing (recozimento) é um procedimento utilizado na indústria para fundir metal que consiste em aquecê-lo a uma temperatura elevada e depois ir resfriando-o lentamente. *Simulated Annealing* é um método de otimização heurístico que surgiu no contexto da mecânica estatística que busca simular este procedimento numericamente, simulando os níveis de temperatura no resfriamento (Haeser e Gomes-Ruggiero, 2008).

Temos uma função objetivo $f(x) : S \to \mathbb{R}$, $x \in S$, S finito que queremos minimizar. Na versão mais simples do algoritmo devemos em cada nível, dado um ponto $u \in S$, gerar vários pontos na vizinhança de u e calcular o valor da função objetivo em cada um destes pontos. Cada ponto é aceito ou não de acordo com uma certa probabilidade de aceitação que decresce de acordo com o nível do processo, ou seja, a medida que a temperatura diminui.

Uma característica deste método é que no início ele tende a aceitar uma grande quantidade de pontos para fugir de um ótimo local, e ir aceitando pontos que pioram o valor da função objetivo com probabilidade cada vez menor, sendo que no limite (temperatura valendo zero) apenas os pontos que melhoram o valor da função são aceitos. Desta forma é minimizada a necessidade de um bom ponto inicial

Stern (1992) descreve uma versão deste método que pode ser diretamente aplicada ao problema de permutar linhas e colunas de uma matriz de modo a obter uma matriz blocada e mostra que ele é bastante eficiente mesmo em comparação à outras heurísticas. Apresentaremos os aspectos principais do método a seguir:

O problema de encontrar uma FAB de uma matriz $A_{m\times n}$ a partir de permutações de linhas e colunas P e Q em b blocos retangulares, $B_1, \ldots, B_b$, mais algumas colunas residuais C pode ser visto como um problemas de grafos em que queremos pintar todos ENNs de cada vértice ou linha $i \in M = \{1, \ldots, m\}$ com uma cor $p(i) \in B = \{1, \ldots, b\}$. A cor $q(j)$ de um arco ou coluna $j \in N = \{1, \ldots, n\}$ é o conjunto de todas as cores de seus ENNs, sendo que colunas multicoloridas correspondem às residuais na FABC. A FABL permite linhas residuais ao invés de colunas. Uma forma mais geral, e aplicável ao nosso problema pode permitir ainda colunas e linhas residuais, sendo que as linhas residuais recebem uma cor especial zero, ou $b + 1$, que não é levada em conta no cálculo das colunas residuais. Em todos os casos é desejável:

- Aproximadamente o mesmo número de linhas em cada bloco;
- Poucas linhas e/ou colunas residuais.

O que nos leva à função objetivo que queremos minimizar:

$$
\begin{aligned}
f(p) &= \alpha \sum_{k=1}^{b} \left[\frac{m}{b} - s(k)\right]^2 + \beta c(p) + \gamma r(p) \\
c(p) &= |\{j \in N : |q(j)| \geq 2\}| \\
r(p) &= |\{i \in M : p(i) = 0\}| \\
s(p) &= |\{i \in M : p(i) = k\}|
\end{aligned} \tag{4.5}
$$

O primeiro termo na função objetivo avalia o quanto o tamanho do bloco se afastou do tamanho ideal, $c(p)$ é o número de colunas residuais e $r(p)$ é o número de linhas residuais.

Definimos um estado vizinho como uma nova cor para a linha i: $p(i) \to p'(i) \in B$. Podemos escolher uma entre as m linhas e então escolher uma entre as b cores fazendo com que o tamanho da vizinhança $TAMVIZ$ de qualquer estado p seja sempre de mesmo tamanho e igual a $m * b$.

O método parte de uma configuração inicial de cores e vai propondo mudanças de estado para um estado vizinho, que podem ser aceitas ou não, até que um critério de parada seja atingido, sendo que pode ser tanto por uma baixa aceitação de movimentos propostos (dizemos então que o algoritmo congelou), ou dado um número máximo de iterações.

O *simulated annealing* se comporta de maneira ruim para problemas de FAB pois muitos vizinhos de um estado de custo baixo podem ter custo alto, e também pelo fato do problema ser degenerado, tendo muitos estados próximos tenham um mesmo custo, fazendo com que a taxa de aceitação permaneça elevada, fazendo com que o algoritmo demore mais para terminar. Para contornar estas dificuldades é proposta uma função dependente da temperatura com o acréscimo de uma penalidade:

$$
\begin{aligned}
f(p, \mu(t)) &= f(p) - \mu(t) pen(p) \\
pen(p) &= \sum_{j, |q(j)|>1} (b - |q(j)|)
\end{aligned} \tag{4.6}
$$

O termo de penalidade pode ser interpretado como uma penalização heurística que recompensa as colunas residuais por utilizarem menos cores. Esta nova função objetivo é linear em μ e coincide com a função objetivo original caso μ valha zero. Por conta dessas propriedades, é razoável que seu decaimento acompanhe o da temperatura. Com isso, o método consiste nos seguintes passos:

1. Dada a matriz A, armazene uma lista de ENNs de cada linha:

$$aijs(i) = \{j \in N : A[i, j] \neq 0\}$$

2. Atribua o estado p através de uma configuração inicial de cores e compute a matriz $n \times b$ de pesos contendo o número de ENNs de cada cor por coluna:

$$W[j, k] = |\{i \in M : A[i, j] \neq 0 \wedge p(i) = k\}|$$

   e utilize esta matriz para calcular $s(k)$ e $f = f(p)$;

3. Inicialize μ com um valor $INITMU$, *temp* com a temperatura inicial $INITTEMP$ e calcule $f(p, \mu)$;

4. Escolha os valores dos parâmetros: $FATORTEMP$, $FATORMU$, $FATORTAMANHO$, $CUTOFF$, $ACEITEMIN$, $MAXCONGELA$;

5. Proponha uma mudança de estado p' através de uma proposta de mudança de cor de uma linha;

6. Calcule os custos das mudanças propostas:

$$\Delta_0 = f(p') - f(p)$$
$$\Delta_\mu = f(p', \mu) - f(p, \mu)$$

7. Aceite a proposta de mudança com probabilidade $prob(\Delta_\mu)$:

$$prob(\Delta_\mu) = \begin{cases} 1 & \text{se } \Delta_\mu \leq 0 \\ exp(-\Delta_\mu/temp & \text{se } \Delta_\mu > 0 \end{cases}$$

8. Em caso de aceitação atualize W, s, c, e $f = f + \Delta_0$;

9. Mantenha armazenado o melhor valor da função objetivo encontrado até o momento;

10. Proponha novas mudanças até que haja: $TAMANHO = FATORTAMANHO \cdot m \cdot b$ propostas ou se aceite $CUTOFF \cdot TAMANHO$ movimentos;

11. Compute a taxa de aceitação para esta temperatura:

$$txact = \frac{aceites}{propostas}$$

12. Decremente a temperatura $temp = temp \cdot FATORTEMP$ e o parâmetro de penalidade $mu = mu \cdot FATORMU$;

13. Cada vez seguida que $txact$ for menor que $ACEITEMIN$ incremente o contador de congelamento. Repita desde o passo 5 até que o numero de congelamentos seguidos ultrapasse $MAXCONGELA$ ou o número total de iterações ultrapasse algum valor previamente estabelecido.

Lembramos que devido à natureza simétrica das matrizes associadas aos grafos que trabalharemos, devemos fazer a otimização considerando linhas e colunas residuais.

Capítulo 5

Conceitos de Computação Paralela

O **paralelismo**, uma das técnicas de programação de alto desempenho (do inglês, *High Performance Computing*, HPC) busca melhorar o desempenho de uma aplicação através da execução simultânea de múltiplas tarefas. Um dos primeiros aspectos que devem ser analisados para tornar uma aplicação paralela é a possibilidade de subdividir o problema original em problemas menores independentes. Outro ponto relevante é a arquitetura do ambiente onde o programa será executado, em particular, se a memória será compartilhada ou distribuída. No primeiro caso estamos interessados em ganhar velocidade de processamento, já no segundo o objetivo é o compartilhamento de recursos e serviços, tendo bastante preocupação em como será feita a passagem de informações. Apesar dessa diferença de paradigmas, ambos os casos requerem atenção especial à sincronização das informações.

A Figura 5.1 mostra a diferença entre esses dois tipos de arquitetura para três processadores, sendo a memória representada por retângulos e cada processador representado por um círculo. Em (a) temos uma situação de memória compartilhada, em que todos os processadores conseguem acessar os mesmos endereços de memória diretamente. No segundo caso, em (b) temos um exemplo de memória distribuída, em que cada processador tem acesso a endereços de memória distintos e precisam realizar a comunicação entre si através da troca de mensagens, e, no caso da situação desta Figura, centralizando as informações em outro lugar.

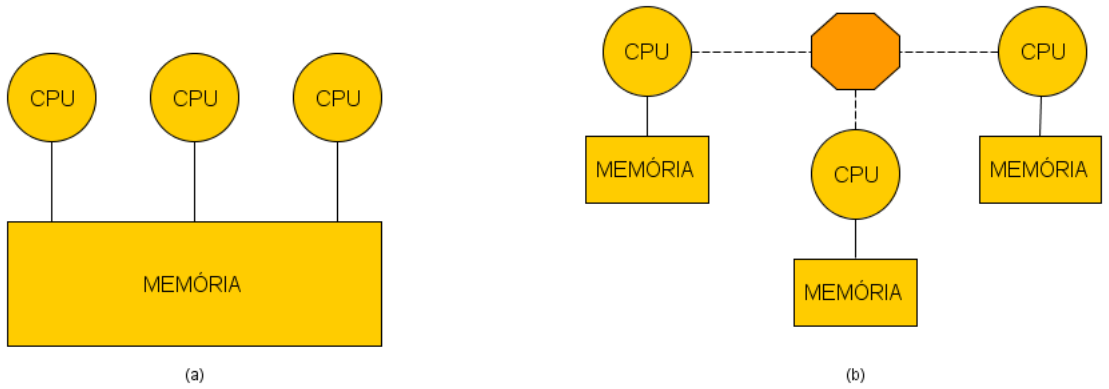


Figura 5.1: Exemplo de arquitetura de memória compartilhada (a) e distribuída (b)

Vamos apresentar brevemente algumas definições e conceitos utilizados em computação paralela e distribuída, para posteriormente aplicar esses conceitos na elaboração de um código distribuído para inferência em redes Bayesianas, partindo do código paralelo originalmente proposto por Colla (2007).

5.1 Definições

As definições e conceitos a seguir seguem de maneira bem próxima o que foi apresentado por Gubitoso (2008) e buscam fornecer uma base para o que será discutido mais adiante.

Programa sequencial: Consiste em um conjunto de instruções que devem ser executadas sequencialmente.

Programa concorrente: Conjunto de programas sequenciais que executam em paralelo (concorrentemente) e de modo cooperativo para cumprir uma tarefa.

Processo: Um programa em execução.

Contexto: Conjunto de variáveis, pilha de execução, valor dos registradores, arquivos abertos, e outros elementos associados ao processo.

Escalonamento: Escolha do próximo processo a ganhar a CPU. Utilizado para balanceamento de carga, ou seja, para evitar a ociosidade das CPUs.

Preempção: Capacidade de um processo mais prioritário interromper temporariamente a execução de outro processo para ganhar a CPU e retomar o processo original posteriormente.

Operação atômica: Operação que pode ser interrompida sem afetar o resultado final do processo.

Seção crítica: Trecho de código de um algoritmo que utiliza um recurso compartilhado que não pode ser utilizado concorrentemente por mais de uma linha de execução.

Threads (ou processos leves): Linhas de execução concorrentes que compartilham o mesmo processo; funções ou procedimentos de um mesmo programa executando concorrentemente e que compartilham o contexto.

Dependência: Dizemos que uma instrução Y depende da instrução X se X precisa ser executada antes de Y. A identificação do paralelismo é baseada fortemente no reconhecimento das dependências dentro do programa que, na medida do possível, devem ser eliminadas.

Sincronização: Garante a cooperação entre os processos e pode ser feita via memória compartilhada ou distribuída.

Embarasing Parallel: Aplicações onde não há dependência alguma entre as tarefas.

Race Condition: (condição de corrida) Quando diferentes processos ou threads em execução dependem de um estado compartilhado, e o resultado depende do escalonamento dos processos.

5.2 Memória Compartilhada

Podemos considerar ser mais fácil programar em máquinas com memória compartilhada, uma vez que o sistema se encarrega da distribuição dos processos entre os processadores e não há a necessidade de troca de mensagens. Todos os processos possuem acesso aos mesmos endereços de memória, e ao mesmo tempo. Um impeditivo para que se use muitos processos com memória compartilhada é a limitação de *hardware*, uma vez que sai caro fazer um computador com muitos processadores e é difícil aumentar sua capacidade.

Em geral se faz programação paralela com memória compartilhada com *threads* utilizando APIs como *POSIX Threads* e *OpenMP*. Operações com threads incluem sua criação, término, sincronização (através de travas e semáforos), agendamento, interação com processos e manipulação de informação.

5.2.1 POSIX Threads

As *POSIX Threads*, ou *PThreads* como são mais conhecidas, surgiram da necessidade de criar um padrão para computação paralela, mais especificamente para *multithreading*, que permitissem aos desenvolvedores programar suas aplicações independente do sistema operacional utilizado.

Pthreads são definidas como um conjunto de tipos de dados, funções e constantes em C. Todos os identificadores da biblioteca começam com o prefixo `pthread_` e estão especificados em um arquivo `pthread.h` que deve ser incluído no código fonte desenvolvido e inclui:

- Gerenciamento de *threads* como criação, união (*join*), etc;
- *Mutexes* (gerenciamento da Seção crítica para eventos mutuamente exclusivos);
- Variáveis de condição (utilizadas para controlar a entrada e saída de processos na Seção crítica);
- Sincronização entre *threads* através de travas e barreiras.

Maiores detalhes sobre especificações de *PThreads*, mecanismos de sincronização e outros algoritmos de computação paralela podem ser vistos em Nichols *et al.* (1996).

5.2.2 OpenMP

OpenMP é um padrão e biblioteca para programação com memória compartilhada e consiste em diretivas para o compilador (pragmas) e uma biblioteca de funções.

As diretivas fazem com que o compilador inclua os comandos para geração e tratamento das *threads*, distribua variáveis, faça a sincronização, etc, de modo bastante transparente. Podem ser um comentário (FORTRAN) ou diretivas de compilador (C e C++). As funções permitem a consulta e controle da execução, mas nem sempre seu uso explícito é necessário.

O uso de diretivas é vantajoso pois facilita a portabilidade uma vez que elas são automaticamente ignoradas por um compilador que não suporte OpenMP e também por serem passíveis de otimizações vindas do compilador, uma vez que são processadas por ele (Chandra, 2001).

Os elementos centrais do OpenMP são:

- Criação das *threads*;
- Distribuição de carga do trabalho;
- Gestão do ambiente de dados;
- Sincronização de *threads*;
- Funções de *runtime* e variáveis de ambiente.

A especificação completa da versão atual (3.1) do OpenMP pode ser encontrada em OpenMP Architecture Review Board (2011).

5.3 Memória Distribuída

Em memória distribuída cada processo possui sua própria memória e a informação só pode ser passada por troca de mensagens. A programação é mais difícil pois a comunicação tem que ser explicitada e deve-se pensar no conjunto com mais detalhes. Por outro lado, é mais fácil construir e estender uma máquina paralela com essa arquitetura, por exemplo os *clusters*.

A principal implicação da memória não ser compartilhada é de como o sistema reconhece o estado em que se encontra, uma vez que não existe nenhum estado global visível por nenhum processo num dado momento. Para definir um estado um processo tem apenas informações locais e as mensagens enviadas ou recebidas, o que implica também na ausência de ordenação temporal dos eventos.

Um sistema distribuído é formado pelos processos e pela rede de comunicação entre eles. Essa rede pode assumir diversas topologias, a Figura 5.2 ilustra as mais frequentes que são anel, estrela, árvore e grafo completo. Cabe ressaltar que a conexão por hardware não precisa acompanhar a topologia virtual.

Existem algumas suposições comuns aos sistemas distribuídos. São elas:

- A transmissão é feita sem duplicação de mensagens;

- A transmissão é feita sem modificação de mensagens;
- As mensagens são recebidas na mesma ordem em que foram enviadas;
- As mensagens são entregues em tempo finito;
- As mensagens são entregues em um tempo máximo conhecido.

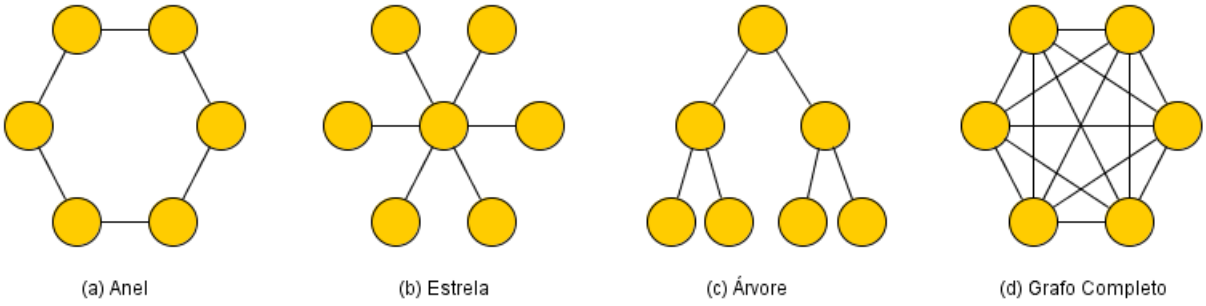


Figura 5.2: Topologias de rede: anel (a), estrela (b), árvore (c) e grafo completo (d).

É ideal que um algoritmo utilize o menor número de suposições possível.
Podemos ainda definir os níveis de simetria de acordo com o particionamento dos processos:

- **Assimetria:** Cada processo corresponde a um programa diferente (por exemplo em um modelo cliente-servidor);
- **Simetria textual:** O programa é o mesmo, a menos de referências internas ao nome. Os nomes dos processos são diferentes;
- **Simetria Forte:** Os programas e os nomes são os mesmos e o comportamento muda de acordo com as mensagens;
- **Simetria Total:** Todos os processos se comportam da mesma maneira.

5.3.1 MPI

O MPI, do inglês *Message Passing Interface*, é uma especificação de uma biblioteca que descreve métodos para troca de mensagens. Existem diversas implementações e várias funções disponíveis para o usuário. O ambiente de comunicação é usado de forma abstrata e mapeado internamente para os canais reais presentes no ambiente. Existem diversas bibliotecas que implementam essas especificações inclusive algumas *open-source* por exemplo OpenMPI e MPICH2.

No padrão MPI uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Em geral, criamos inicialmente um conjunto fixo de processos que são capazes de executar diferentes programas. Isso faz com que o padrão MPI algumas vezes seja referido como MPMD *(multiple program multiple data)*. Elementos importantes em implementações paralelas são a comunicação de dados entre processos paralelos e o balanceamento da carga. Os processos podem usar mecanismos de comunicação ponto a ponto (operações para enviar mensagens de um determinado processo a outro). Um grupo de processos pode invocar operações coletivas *(collective)* de comunicação para executar operações globais. O MPI é capaz de suportar comunicação assíncrona e programação modular, através de mecanismos de comunicadores *(communicator)* que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna (Ignácio e Ferreira Filho, 2002).

No MPI cada tarefa deve ser identificada por um id assim que um processo envolvendo aplicação MPI está sendo iniciado. O ambiente MPI atribui a cada processo um *rank* (armazenado como um

valor inteiro) e um grupo de comunicação. O *rank* é um tipo de identificador de processo para cada tarefa MPI. É contíguo, começa por zero e serve para especificar a origem e o destino de mensagens. Frequentemente é usado em condições para controle de execução que permitem o processo agir de maneiras distintas de acordo com seu *rank*. O grupo de comunicação define quais processos podem chamar comunicações ponto a ponto. Inicialmente, todos os processos MPI são associados a um grupo de comunicação padrão. Os membros de um grupo de comunicação podem mudar depois da aplicação ter iniciado. Além do envio de mensagens ponto a ponto também é possível fazer um envio coletivo de mensagens (para todos os componentes do grupo, por exemplo), o que é chamado de *broadcast*.

Para fazer uso das funções de uma biblioteca MPI o programa deve ser compilado de forma especial, por um compilador específico (em geral chamado mpicc para códigos em C compilados com gcc) e também deve ser executado através de um comando especial (mpirun ou mpiexec) para que os processos sejam devidamente atribuídos às CPUs. Nessa chamada é preciso especificar o número de processadores que serão utilizados e passar um arquivo de hosts para que sejam conhecidos os nomes ou endereços dos processadores disponíveis, além do nome do programa que será executado em paralelo.

A Tabela 5.1 apresenta os principais comandos MPI. Uma lista mais extensa e mais detalhada pode ser obtida em Aoyama e Nakano (1999).

Tipo	Subrotina
Comunicação Ponto a Ponto	MPI_SEND, MPI_RECV, MPI_WAIT
Comunicação Coletiva	MPI_BCAST, MPI_GATHER, MPI_REDUCE
Tipos de Dados	MPI_TYPE_CONTIGUOUS,MPI_TYPE_COMMIT
Topologia	MPI_CART_CREATE, MPI_GRAPH_CREATE
Comunicadores	MPI_COMM_SIZE, MPI_COMM_RANK
Grupos	MPI_GROUP_SIZE, MPI_GROUP_RANK
Gestão do Ambiente	MPI_INIT, MPI_FINALIZE, MPI_ABORT
Arquivos	MPI_FILE_OPEN, MPI_FILE_READ_AT
Informações	MPI_INFO_GET, MPI_INFO_SET
Medição de Tempo	MPI_WTIME, MPI_WTICK

Tabela 5.1: Principais comandos MPI

5.4 Speed up e desempenho

Sendo um dos principais objetivos da programação paralela o ganho de desempenho, é preciso elaborar critérios e medidas para avaliar a eventual melhora de um programa paralelo em relação à sua versão serial. A **performance** de um programa ou algoritmo está relacionada ao tempo de execução do mesmo, dizemos que quanto menor o tempo, maior a performance, ou desempenho. ***Speedup*** é uma medida para a performance definida pelo quociente dos tempos de execução das versões serial e paralela de um mesmo programa. A **eficiência** é uma medida do uso efetivo de cada processador e em geral é calculada pelo coeficiente entre o *speedup* e o número de processadores. Um valor de eficiência menor que um indica que o processador não está sendo utilizado ao máximo, enquanto valores maiores que um indicam que o problema poderia ser mais paralelizado para um melhor desempenho. Finalmente, **escalabilidade** refere-se ao *speedup* de um algoritmo dados diferentes números de processadores e pode ser avaliada através da eficiência, se esta permanecer constante à medida que o número de processadores aumenta, temos uma escalabilidade linear.

Levando em conta que o número de processadores disponíveis é limitado, que acesso à memória ou troca de mensagens através da rede levam tempo, a natureza do problema, a necessidade de sincronização presente em trechos de algoritmos, entre outros fatores, os conceitos anteriores ajudam a responder se há a necessidade real de paralelizar determinado algoritmo. Uma lei bastante utilizada

para estudar esta questão é a **Lei de Amdahl** que separa um código entre sua parte exclusivamente serial, S e a paralela P:

$$Speedup = \frac{1}{S + \frac{P}{N}} \tag{5.1}$$

Onde N é o número de processadores. Podemos ver através desta lei que à medida que o número de processadores aumenta, o *speedup* passa a ser dominado pela parte serial do programa. O denominador da equação pode ser considerado o tempo de execução do programa.

Chamamos de ***overhead*** todo o custo computacional adicionado pela paralelização do algoritmo, inclusive a alocação das *threads*. Este custo não é previsto na lei de Amdahl. Naturalmente, quanto maior o *overhead*, maior a chance de um programa não ganhar muito desempenho ao ser paralelizado, por isso, também é importante ter alguma forma de medi-lo.

5.4.1 Avaliando a possibilidade de paralelizar ou não

Muitos trabalhos foram desenvolvidos na tentativa de avaliar o comportamento de uma aplicação paralela, porém eles apresentam alguma limitação (Sanjai, 2008). Algumas destas limitações podem ser contornadas assumindo por exemplo que os nós terão dedicação exclusiva (não precisarão disputar tempo de processamento com programas externos), e que possuímos conhecimento das complexidades computacionais do algoritmo, uma vez que estamos tratando de uma biblioteca que foi implementada com código aberto.

Como nosso objetivo é estruturar uma biblioteca modularizada, apresentaremos um método que avalia para uma dada função, os ganhos computacionais de paralelização para estimar e comparar o tempo de execução do programa em sua versão serial e paralela, considerando a separação do código nas partes serial S e paralela P como na lei de Amdahl. Se o gasto com o *overhead*, $\kappa(P, N)$ da parte paralelizável P para N processadores for maior que o ganho por paralelizar uma parte do código, então é mais vantajoso executá-lo de forma serial. Em outras palavras se:

$$S + P \leq S + \frac{P}{N} + \kappa \Rightarrow P \leq \frac{P}{N} + \kappa \Rightarrow \frac{P(N-1)}{N} \leq \kappa \tag{5.2}$$

Uma vez conhecido o código fonte podemos usar FLOPs, *Floating-point Operations*, ou em português, operações de ponto flutuante, para estimar o custo computacional das operações. Como o custo de uma operação com ponto flutuante varia dependendo da arquitetura e capacidade do processador, é preciso estimá-lo de alguma forma. Além disso, precisamos calcular o custo de *overhead*. Consideraremos gastos com:

- Comunicação (exclusivo para memória distribuída)
- Operações extras de acesso em memória (para memória compartilhada)
- Sincronização

Além do número de processadores N e a parte paralelizável P, existem dados específicos que devem ser informados, que se relacionam com a natureza do problema e serão utilizados em P, por exemplo, em uma árvore de eliminação a largura da mesma. A quantidade de comunicação, acessos extras à memória e operações de sincronização podem ser estimadas pelo código fonte, restando apenas descobrir quanto tempo efetivamente é consumido por cada tipo de operação.

Cabe mencionar que o custo de todas as operações da parte paralela que não estão presentes na parte serial, como por exemplo a criação de uma árvore de threads deve ser computado também como *overhead*. Devemos perceber também que apesar do custo da execução da parte exclusivamente serial do código não entrar no cálculo final, se ela for muito grande em relação à parte paralelizável, ou seja, se o programa for pouco escalável, ainda que valha a pena paralelizar de acordo com a equação 5.2, o ganho computacional será pouco percebido uma vez que o tempo total será dominado pela parte serial. Levamos isto em conta na hora de escolher quais funções da biblioteca seriam disponibilizadas para execução paralela.

Estimando gastos de tempo com as operações

Afim de descobrir os custos computacionais realizados pelo algoritmo e por sua paralelização elaboramos algumas funções que realizam operações e contabilizam seu tempo. Essas funções são auxiliares, algumas baseadas em códigos de software livre prontos já implementados por terceiros e idealmente devem ser chamadas uma única vez antes de rodar o programa e seus valores armazenados para que sejam passados diretamente para a biblioteca em futuras execuções, isto porque a própria execução destas funções consome tempo e, dependendo do tamanho do problema, pode ficar mais caro descobrir se vale a pena paralelizar ou não que rodar o programa de forma serial, ou paralela. Além disso, cabe ressaltar que os tempos calculados serão todos com base nos testes computacionais, e que os resultados podem variar entre dias diferentes ou computadores, e que a idéia é apenas dar uma direção sobre um eventual gasto muito grande com paralelização que faça o trabalho computacional extra não valer a pena.

Para estimar o valor de FLOPs utilizamos uma adaptação da tradução do *benchmark* do LINPACK (Dongarra, 1987) para C de Bonnie Toy com modificações de Jack Dongarra e Roy Longbottom que faz a resolução de um sistema linear denso 100×100 através da fatoração LU e estima a quantidade de MFLOPS (MegaFLOPS = 10^6 FLOPS). LINPACK é uma biblioteca científica de funções de álgebra linear computacional desenvolvida em FORTRAN nos anos 70 e seu *benchmark*, que originalmente fazia parte da biblioteca, acabou tornando-se um projeto separado amplamente utilizado para comparar performance de computadores e supercomputadores. Mais detalhes sobre o método utilizado pode ser visto mais adiante na Seção 7.2.1

Para estimar os gastos de comunicação e sincronização com MPI fizemos uso de uma função da própria biblioteca MPI, $MPI_Wtime()$ que pode ser utilizada para computar o tempo decorrido entre determinadas operações. Nos baseamos no procedimento indicado por Luecke *et al.* (2009) que consiste em inicialmente descobrir um número ótimo de medições que devem ser relizadas, em seguida efetuar repetidas comunicações (em um processo conhecido como ping-pong) para dois processadores, computar o tempo apenas no processo pai após o recebimento das mensagens do filho e descontar o próprio *overhead* introduzido pelo uso da função $MPI_Wtime()$, após chamá-la sucessivas vezes sem nenhuma operação intermediária. Além do *send-receive* utilizamos também *broadcasts* para um número maior de processadores passando uma quantidade variável de caracteres (*chars* até atingir um valor máximo estabelecido pelo usuário. Ao invés de trabalhar com um número ótimo, optamos por fazer uma regressão linear dos tempos para diferentes quantidades de dados transmitidos, uma vez que testes preliminares indicaram que o tempo cresce linearmente conforme o aumento da quantidade de dados.

Para o tempo de alocação de *threads* e acesso à memória com POSIX threads fizemos testes semelhantes aos do MPI, testando basicamente a criação de threads e utilização dos *mutexes* da mesma forma que feito por de Supinski e May (1999) que faz a adaptação de um *benchmark* MPI para *PThreads* considerando suas principais funcionalidades e substituindo a função de medição de tempo para a nativa `gettimeofday`. A escolha por essa abordagem é devido à estrutura de nossa biblioteca que trabalha com esse tipo de centralização, tanto na árvore de *threads* quanto na fatoração blocada.

Finalmente, organizamos a chamada a todas essas funções em um único programa que retorna os valores de tempo calculados para cada um dos blocos de operações. Como todas as estimativas anteriores simplificam e unificam muitas etapas do processo, como generalizar o custo por operação que pode variar entre somas e divisões, por exemplo, ou a disponibilidade de carga dos processadores para comunicação distribuída realizada, o objetivo é fornecer apenas uma direção sobre a necessidade real de se paralelizar, e em casos de ganhos ou perdas muito pequenos, utilizaremos sempre o critério de minimizar o número de processadores utilizados. Além disso, é preciso considerar nos cálculos toda implementação extra feita por conta da paralelização, como é o caso da *ThreadTree* que falaremos mais adiante na implementação computacional da biblioteca paralela.

Capítulo 6

Inferência Paralela em Redes Bayesianas e Credais

Neste Capítulo combinaremos o que foi visto para redes Bayesianas e Credais com álgebra linear e computacional com o objetivo de criar algoritmos eficientes e paralelizáveis para inferência nas redes. Apresentaremos também o que já foi posto por Colla (2007) em seu trabalho, um algoritmo computacional que utiliza técnicas de Fatoração de Cholesky para matrizes esparsas para otimizar e paralelizar o processamento numérico necessário para inferência em Redes Bayesianas.

Como já é frequentemente utilizado em métodos de eliminação de variáveis, adotamos a separação dos algoritmos em uma etapa simbólica e outra numérica. A vantagem desse tipo de separação é permitir a aplicação de métodos de fatoração de matrizes esparsas que são equivalentes aos métodos de inferência que levam a ganhos computacionais no algoritmo final. Além disso, considerando a possibilidade de salvar esta etapa, algo que ocorre na versão atual da biblioteca, é possível reaproveitar as estruturas de eliminação geradas para realizar diversas consultas, mantendo constante somente a estrutura da rede.

Ao final apresentaremos dois exemplos numéricos que ilustram os conceitos apresentados até o momento e o funcionamento de cada algoritmo.

6.1 Algoritmo de Eliminação de Variáveis Paralelo

A Figura 6.1 mostra os passos presentes no algoritmo de eliminação de variáveis. Em comparação ao trabalho de Colla (2007), pode-se ver que acrescentamos a heurística do *simulated annealing* e também trabalharemos com a fatoração blocada que permite paralelização já nesta etapa. Fora isto, as duas grandes etapas permanecem iguais:

1. **Fase Simbólica:** Nesta fase é definido o conjunto de variáveis requisitadas, a ordem de eliminação e a árvore de eliminação

2. **Fase Numérica:** Nesta fase ocorre o processamento numérico com a multiplicação e marginalização das variáveis e normalização dos potenciais envolvidos na rede.

6.1.1 Fase Simbólica

Na fase simbólica ocorre todo o processamento relacionado à topologia da rede, sem trabalhar propriamente com os valores numéricos das distribuições. Isso permite ganho computacional e, como dito anteriormente, a possibilidade de reaproveitamento do processamento para diversas consultas. Mantivemos a eliminação das variáveis irrelevantes para a consulta nesta fase pois compreendemos que este procedimento também envolve apenas a topologia da rede, detalhes sobre o funcionamento do *Bayes-Ball* podem ser vistos na Seção 2.1.4 e suas referências.

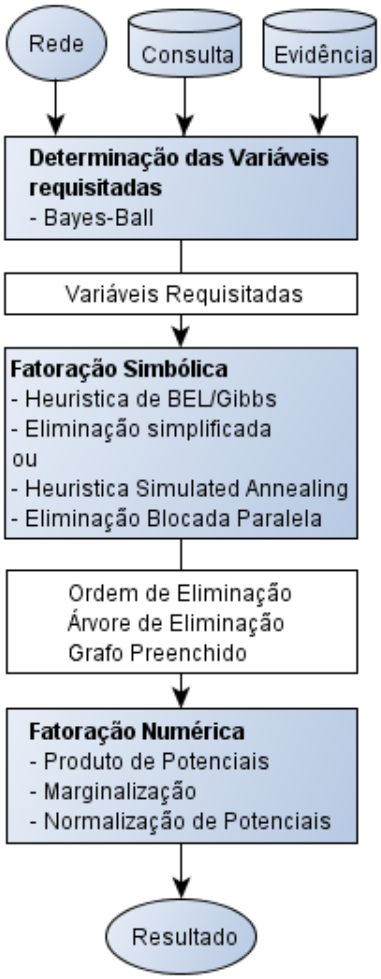


Figura 6.1: Etapas da inferência na rede

Na fase simbólica é executada a **Fatoração Simbólica**, onde são aplicadas técnicas de álgebra linear computacional para fatoração de matrizes esparsas para inferência em redes Bayesianas. A Fatoração Simbólica consiste na determinação da ordem de eliminação e construção da árvore de eliminação dos vértices do grafo da rede, ambos algoritmos que recebem como entrada um grafo não direcional. Esta fase, que busca uma otimização do processo de inferência e viabilizar a paralelização do processamento numérico será também realizada em paralelo, caso haja vantagem computacional.

O uso de uma etapa simbólica e construção de árvore de eliminação é amplamente utilizado para fatoração de Cholesky em matrizes esparsas. Esta etapa é obtida essencialmente retirando do algoritmo de fatoração todas aquelas operações que envolvem os valores da posição da matriz e efetuando somente aquelas relacionadas à topologia do grafo associado à matriz. Liu (1990) em seu trabalho mostra com detalhes a importância deste tipo de processo, em particular do uso de uma árvore de eliminação. Para a fatoração simbólica da rede Bayesiana, que visa a construção de uma árvore de eliminação também utilizamos somente a topologia do grafo moral associado à rede, sem a necessidade de conhecer os valores das probabilidades.

A primeira versão da biblioteca utiliza as Heurísticas de Busca em Largura e de Gibbs, detalhadas na Seção 4.3.1 para determinar a ordem de eliminação. Considerando a possibilidade de trabalhar com redes cujos grafos morais resultem em estruturas blocadas, adicionamos uma nova heurística, o *Simulated Annealing*, descrita na Seção 4.4.2 que também gera uma ordem de eliminação, mas com o objetivo de encontrar uma permutação que coloque a matriz de adjacência do grafo em sua forma angular blocada. A escolha das heurísticas busca contemplar dois objetivos principais:

1. Gerar uma ordem de eliminação eficiente que minimize a inserção de arcos de preenchimento durante a eliminação dos vértices;

2. Gerar uma ordem que maximize as possibilidades de paralelização do processamento numérico.

E no *Simulated Annealing* buscamos adicionalmente também gerar uma ordem que leve a uma otimização da eliminação paralela de variáveis, como na fatoração de Cholesky blocada com blocos igualmente distribuídos e minimizando as colunas residuais. Em linhas gerais, apresentamos o algoritmo de determinação da ordem de eliminação a seguir, sem preocupação com os detalhes computacionais que são fornecidos no Capítulo 7.

Determinação da Ordem de Eliminação

Dado um Grafo Moral $G = (V, E)$ associado a uma Rede Bayesiana:

1. Inicialize todos os vértices do grafo como não testados.

2. Escolha um vértice de grau mínimo v do grafo e marque-o como testado.

3. Construa a BEL adotando como raiz o vértice v e a defina como a $BEL_{corrente}$.

4. Construa uma lista de nós a serem testados com os nós do nível mais profundo da $BEL_{corrente}$.

5. Enquanto a lista de nós a serem testados não for vazia:

   (a) Para cada nó v da lista de nós a serem testados faça:
   Marque o nó v como testado.
   Construa a BEL adotando como raiz o vértice v e a defina como BEL_{nova}.
   Se, segundo os critérios de comparação, a BEL_{nova} for superior a $BEL_{corrente}$ então adote a BEL_{nova} como a nova $BEL_{corrente}$.

   (b) Se houve a substituição da $BEL_{corrente}$ construa uma nova lista de nós a serem testados com os nós não marcados como testados do nível mais profundo da $BEL_{corrente}$.

6. Para a $BEL_{corrente}$ construa a Árvore de Dissecção.

7. A partir da raiz percorra a Árvore de Dissecção recursivamente montando a ordenação dos vértices em pós-ordem.

8. Monte a ordem de eliminação percorrendo a lista dos vértices da Árvore de Dissecção listados em pós-ordem substituindo cada vértice da Árvore de Dissecção pelos vértices correspondentes do grafo original.

Determinação da Ordem de Eliminação com Blocagem

Dado um Grafo Moral $G = (V, E)$ associado a uma Rede Bayesiana:

1. Execute a heurística *Simulated Annealing* com penalidade relacionada à temperatura para a matriz de adjacência do Grafo Moral.

2. Ordene a matriz em blocos de acordo com as cores obtidas para cada coluna, obtendo os blocos ${}_i^iB$, $i = 1, \ldots, h$ e as colunas residuais agrupadas em C. A permutação das linhas (e colunas) realizada para blocar a matriz corresponde à ordem de eliminação.

3. Armazene o tamanho dos blocos para que seja possível realizar a etapa simbólica blocada.

4. Caso haja muita esparsidade dentro de um bloco ${}_i^iB$, $i = 1, \ldots, h$ é possível calcular uma nova ordem de eliminação pelos métodos de gibbs e BEL considerando o grafo ${}_i^iB \cap C$ para cada bloco.

A escolha entre os algoritmos apresentados deve ser feita com base na estrutura da matriz e disponibilidade de recursos computacionais para paralelização. Na Seção 5.4.1 discutimos um pouco sobre as situações que determinado algoritmo deve ser rodado em paralelo ou não.

Definida a ordem de eliminação, partimos para a etapa seguinte da fatoração simbólica, que resultará na árvore de eliminação e no grafo preenchido. O algoritmo utilizado será o de eliminação simplificada decorrente do teorema da fatoração simbólica visto na Seção 4.3, este algoritmo tem a preocupação de introduzir poucos arcos de preenchimento no grafo, uma vez que ao longo do processo de eliminação dos vértices adiciona ao grafo apenas os arcos de preenchimento que incidem sobre o vértice vizinho mais próximo de ser eliminado, o que o torna computacionalmente eficiente.

Colla (2007) relaciona a relação de dependência entre as variáveis da rede Bayesiana dada pelos arcos do grafo original e os de preenchimento com a a relação de dependência na eliminação das colunas da matriz dada pelos arcos na fatoração de Cholesky, que, de maneira resumida, é que o acréscimo de um elemento não nulo na matriz fatorada corresponde à inclusão de um arco de preenchimento na eliminação de variáveis. Na versão inicial do algoritmo de construção da árvore de eliminação estas dependências são analisadas para realizar o processamento numérico em paralelo. Apresentamos o algoritmo a seguir:

Fatoração Simbólica para construção da Árvore de Eliminação

Dado um Grafo Moral $G = (V, E)$ associado a uma Rede Bayesiana e uma ordem de eliminação q:

1. Inicializar com os nós de G a estrutura para armazenar a Árvore de Eliminação H, mas sem estabelecer ainda os arcos de H.
2. Inicializar com os nós e os arcos de G a estrutura para armazenar o Grafo Preenchido F, ligando em F apenas os nós que estejam ligados em G.
3. $j = 0$.
4. Enquanto $j <$ (Número de Vértices de G - 1) faça:
   (a) Definir v_j como o j-ésimo nó de q.
   (b) $i = j + 1$.
   (c) Enquanto $i <$ (Número de Vértices de G) faça:
      i. Definir v_i como o i-ésimo nó de q;
      ii. Se o existe em F o arco v_j , v_i então
      Acrescentar em H um arco direcional v_i, v_j;
      Definir v_i como o vizinho de v_j mais próximo de ser eliminado;
      Interromper o laço e as iterações de i;
      iii. Incrementar $i = i + 1$;
   (d) $k = i + 1$;
   (e) Enquanto $k <$ (Número de Vértices de G) faça:
      i. Definir v_k como o k-ésimo nó de q;
      ii. Se existir em F um arco v_j , v_k e não existir em F o arco v_i, v_k então
      Acrescentar em F o arco v_i, v_k;
      iii. Incrementar $k = k + 1$;
   (f) Incrementar $j = j + 1$;

Levando em conta uma possível estrutura angular blocada obtida pelo *Simulated Annealing* e também a possibilidade de trabalhar com vários processadores já nesta etapa, fizemos uma modificação do algoritmo anterior, baseado na Fatoração de Cholesky Blocada:

Fatoração Simbólica Paralela para construção da Árvore de Eliminação

É possível construir a Árvore de Eliminação e o grafo preenchido através de um algoritmo paralelo de fatoração em blocos, baseado na fatoração blocada de Cholesky que apresentamos na Seção 4.4.1.

Dado um Grafo Moral $G = (V, E)$ associado a uma Rede Bayesiana, uma ordem de eliminação q, h blocos de tamanho n e m linhas residuais. Sejam G_b^i o bloco i do grafo e G_c o conjunto de vértices correspondentes às colunas residuais na matriz de adjacência:

1. Crie um subgrafo considerando as variáveis presentes nas ultimas m posições da ordem de eliminação q (G_c);
2. Para cada um dos blocos i crie um subgrafo com as variáveis presentes em G_b^i e G_c, o que pode ser feito com o auxílio do vetor q e o conhecimento do valor de n;
3. No processador 0 faça a fatoração simbólica de G_c através do algoritmo sequencial visto em 6.1.1;
4. No processador i, $i = 1, \ldots, h$ faça a fatoração simbólica de $G_b^i \cap G_c$ através do algoritmo sequencial visto em 6.1.1;
5. Acumule no processador 0 todos os resultados das fatorações simbólicas parciais;
6. Faça um *merge* dos subgrafos, adicionando à arvore de eliminação de G_c os arcos necessários para incorporar cada uma das árvores de eliminação correspondentes aos blocos G_b^i, $i = 1, \ldots, h$.
7. Faça um novo *merge* dos subgrafos, adicionando ao grafo preenchido de G_c as ligações necessárias para incorporar cada um dos grafos preenchidos correspondentes aos blocos G_b^i, $i = 1, \ldots, h$.

É interessante notar que durante todo o processo não utilizamos nenhuma informação numérica relativa às variáveis da rede Bayesiana. Desta forma, mantendo constantes os elementos dos conjuntos X_q e E, e a estrutura da rede, é possível fazer diversas inferências sem ter que refazer a Fatoração simbólica. Exatamente por isso optamos por, ao fim desta etapa, permitir ao usuário salvar a estrutura da árvore de eliminação e a ordem de eliminação para que ele possa realizar inferências posteriormente, pulando esta etapa.

6.1.2 Fase Numérica

Na fase numérica são feitas as operações que envolvem números: produto entre potenciais, marginalização das variáveis e normalização dos mesmos, o que originou o termo **Fatoração Numérica**. Estas operações são as mesmas realizadas por outros métodos de inferência para modelos probabilísticos em grafos, em particular no método de eliminação de variáveis descrito na Seção 2.1.5.

Nesta fase nos aproveitamos da estrutura obtida na fatoração simbólica; a árvore de eliminação e ordem de eliminação fornecem bastante informação sobre a estrutura dos potenciais envolvidos e, em particular, sobre quais operações podem ser realizadas em paralelo. Segundo Colla (2007), estas informações podem ser utilizadas para alocar de forma estática os recursos computacionais que serão necessários antes de se iniciar o processamento, o que do ponto de vista computacional pode representar um ganho significativo de performance.

Dada uma árvore de eliminação, cada nó pode ser eliminado da árvore após a eliminação de todos seus descendentes, conforme expresso pelos arcos direcionais dos grafos. Isto implica que o processo começa a eliminação dos nós em cada folha, e que isto pode ser feito de maneira paralela em relação às folhas da árvore resultante após a eliminação de uma leva de nós terminando na raiz da árvore. Este processo é representado pela Figura 6.2, nela temos a árvore de eliminação de uma

rede Bayesiana construída na etapa da fatoração simbólica e ao seu lado o processo de fatoração numérica, com os nós processados (em paralelo) em cada etapa em destaque.

É possível observar ainda que o processamento sequencial demandaria 13 ciclos de processamento enquanto usando a topologia da árvore de eliminação e processamento paralelo esse número pode ser reduzido para 4 ciclos.

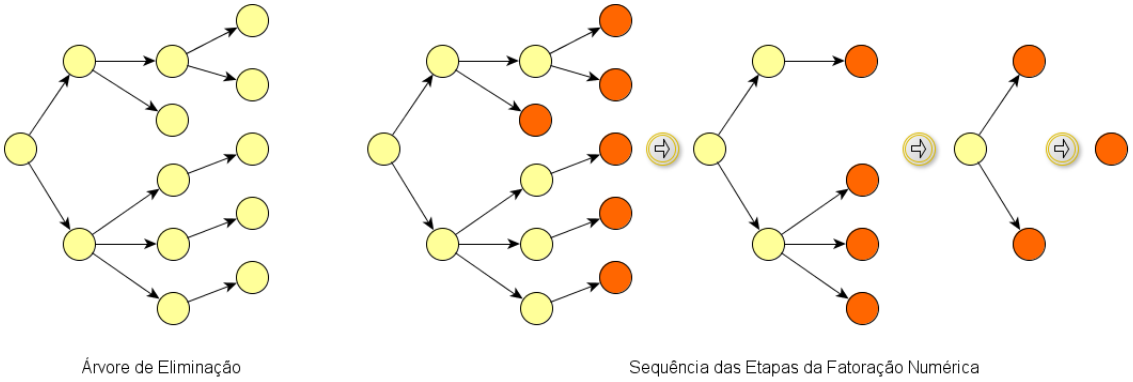


Figura 6.2: Exemplo de processamento numérico paralelo

Detalhes sobre implementação computacional das estruturas de dados e do algoritmo são dados na Seção 7.6.

6.2 Algoritmo de Inferência Paralela para Redes Credais

Uma vez que temos um algoritmo para reduzir uma rede Credal à uma rede Bayesiana, visto na Seção 3.1.1, para fazer a inferência precisamos apenas ser capazes de enumerar todas as combinações de valores das variáveis transparentes. Os valores de probabilidade inferior e superior são obtidos comparando os valores das inferências de cada rede.

Para ganhar desempenho, caso haja disponibilidade, podemos realizar a inferência em cada uma das redes paralelamente, obtendo uma economia significativa de ciclos de processamento, em especial para redes com muitas variáveis transparentes, ou variáveis transparentes com muitos estados. Conforme explicaremos mais adiante, na Seção 7.7, essa parte funciona bem em ambiente distribuído, pois existe uma quantidade grande de processamento a ser executada antes da troca de mensagens, e esse processamento pode ser realizado de forma paralela, tirando proveito de *clusters* com arquitetura híbrida (vários computadores cada um com mais de um processador).

A enumeração dos conjuntos é feita de maneira bastante simples:

1. Inicialize a lista τ de variáveis transparentes;
2. Durante o algoritmo CCM na rede Credal, a cada adição de uma variável transparente T'_i, $i = 1, \ldots, n$ adicione-a em τ;
3. Para cada variável transparente T'_i associe uma lista com seus valores possíveis;
4. Combine cada valor de uma variável T'_i com cada um dos valores possíveis das variáveis restantes T_j, $j = 1 \ldots, n \wedge j \neq i$.

6.3 Exemplo de Inferência Credal Paralela

Para ilustrar o procedimento de inferência paralela em redes Credais (e, consequentemente, em redes Bayesianas) vamos nos basear no exemplo apresentado por Colla (2007), fazendo as adaptações necessárias para transformar a rede Bayesiana em uma rede Credal.

Buscamos, assim como no exemplo original, detalhar as etapas do processo de modo a ilustrar todo o procedimento. As mesmas etapas realizadas no exemplo são feitas computacionalmente, através dos processos descritos no Capítulo 7. O formato de entrada de informações para a inferência é xml, conforme especificado na Seção 7.3 e os arquivos que geram esta rede Credal, a consulta e as evidências estão no apêndice C.1.

A Figura 6.3 apresenta o GDA de uma rede Credal cujas probabilidades estão representadas pela tabela 6.1

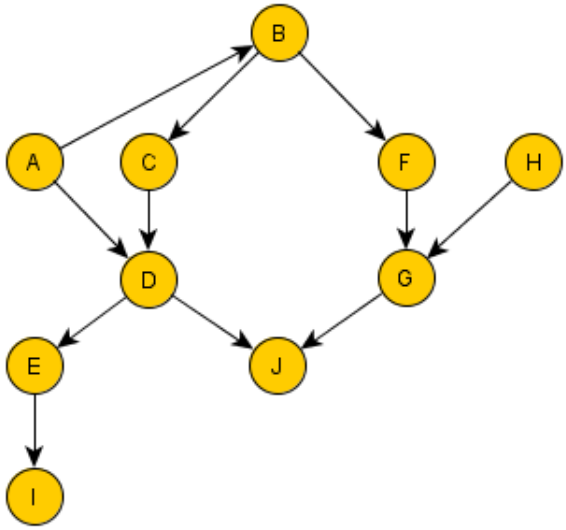


Figura 6.3: Exemplo de Rede Credal para inferência paralela

Nosso objetivo é calcular as probabilidades superiores e inferiores da variável I observados os estados das variáveis $H = H2$ e $J = J3$, ou seja, queremos obter $\overline{P}(I|H = H2, J = J3)$ e $\underline{P}(I|H = H2, J = J3)$. Assumiremos já conhecidas as evidências, a rede e a consulta.

O primeiro passo que devemos fazer é adicionar as variáveis transparentes de modo a transformar a rede Credal em uma rede Bayesiana. A rede Bayesiana obtida é apresentada pela Figura 6.4 e pela tabela 6.2.

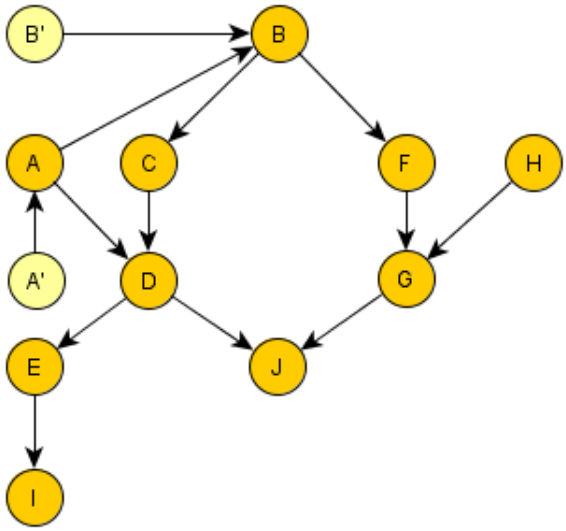


Figura 6.4: Exemplo de Rede Credal com Variáveis Transparentes para inferência paralela

P(A)	
A1	[0,58; 0,62]
A2	[0,08; 0,10]
A3	[0,27; 0,35]

P(H)	
H1	0,50
H2	0,03
H3	0,47

P(B|A)	A1	A2	A3
B1	[0,34; 0,36]	[0,03; 0,05]	[0,47;0,49]
B2	[0,48; 0,52]	[0,38; 0,42]	[0,46; 0,50]
B3	[0,12; 0,17]	[0,53; 0,59]	[0,01; 0,07]

P(D|A,C)	A1		A2		A3	
	C1	C2	C1	C2	C1	C2
D1	0,01	0,23	0,51	0,26	0,40	0,27
D2	0,47	0,05	0,23	0,32	0,46	0,38
D3	0,52	0,72	0,26	0,42	0,14	0,35

P(C|B)	B1	B2	B3
C1	0,08	0,51	0,86
C2	0,92	0,49	0,14

P(G|F,H)	F1			F2		
	H1	H2	H3	H1	H2	H3
G1	0,03	0,39	0,48	0,96	0,51	0,60
G2	0,97	0,61	0,52	0,04	0,49	0,40

P(E|D)	D1	D2	D3
E1	0,38	0,92	0,12
E2	0,62	0,08	0,88

P(I|E)	E1	E2
I1	0,42	0,79
I2	0,58	0,21

P(J|D,G)	D1		D2		D3	
	G1	G2	G1	G2	G1	G2
J1	0,43	0,21	0,07	0,14	0,33	0,35
J2	0,32	0,53	0,39	0,68	0,40	0,56
J3	0,25	0,26	0,54	0,18	0,27	0,09

P(F|B)	B1	B2	B3
F1	0,79	0,43	0,45
F2	0,21	0,57	0,55

Tabela 6.1: Probabilidades da Rede credal para inferência paralela

P(B|A, B')	A1		A2		A3	
	B'1	B'2	B'1	B'2	B'1	B'2
B1	0,34	0,36	0,03	0,05	0,47	0,49
B2	0,48	0,52	0,38	0,42	0,46	0,50
B3	0,12	0,17	0,53	0,59	0,01	0,07

P(C|B)	B1	B2	B3
C1	0,08	0,51	0,86
C2	0,92	0,49	0,14

P(H)	
H1	0,50
H2	0,03
H3	0,47

P(A|A')	A'1	A'2
A1	0,58	0,62
A2	0,08	0,10
A3	0,27	0,35

P(D|A,C)	A1		A2		A3	
	C1	C2	C1	C2	C1	C2
D1	0,01	0,23	0,51	0,26	0,40	0,27
D2	0,47	0,05	0,23	0,32	0,46	0,38
D3	0,52	0,72	0,26	0,42	0,14	0,35

P(E|D)	D1	D2	D3
E1	0,38	0,92	0,12
E2	0,62	0,08	0,88

P(G|F,H)	F1			F2		
	H1	H2	H3	H1	H2	H3
G1	0,03	0,39	0,48	0,96	0,51	0,60
G2	0,97	0,61	0,52	0,04	0,49	0,40

P(I|E)	E1	E2
I1	0,42	0,79
I2	0,58	0,21

P(J|D,G)	D1		D2		D3	
	G1	G2	G1	G2	G1	G2
J1	0,43	0,21	0,07	0,14	0,33	0,35
J2	0,32	0,53	0,39	0,68	0,40	0,56
J3	0,25	0,26	0,54	0,18	0,27	0,09

P(F|B)	B1	B2	B3
F1	0,79	0,43	0,45
F2	0,21	0,57	0,55

Tabela 6.2: Probabilidades da Rede Credal com Variáveis Transparentes para inferência paralela

Esta nova rede Bayesiana codifica uma única distribuição conjunta de probabilidades para o conjunto de variáveis aleatórias $\mathbb{X} = \{A, B, C, D, E, F, G, H, I, A', B'\}$. Como vimos, a topologia do grafo da Rede Bayesiana define as relações de dependência entre as variáveis, permitindo especificar as distribuições de probabilidades envolvidas e, portanto, os potenciais envolvidos:

$$\begin{array}{ccc} P(A|A') = \Phi_A(A, A'), & P(B|A, B') = \Phi_B(B, A, B'), & P(C|B) = \Phi_C(C, B), \\ P(D|A, C) = \Phi_D(D, A, C), & P(E|D) = \Phi_E(E, D), & P(F|B) = \Phi_F(F, B), \\ P(G|F, H) = \Phi_G(G, F, H), & P(H) = \Phi_H(H), & P(I|E) = \Phi_I(I, E), \\ P(J|D, G) = \Phi_J(J, D, G), & P(A') = \Phi_{A'}(A'), & P(B') = \Phi_{B'}(B'). \end{array}$$

Desta forma, a distribuição de probabilidades de $\mathbb{X}$ pode ser calculada por:

$$\begin{gathered} P(\mathbf{X}) = P(A|A') \cdot P(B|A, B') \cdot P(C|B) \cdot P(D|A, C) \cdot P(E|D) \cdot P(F|B) \cdot \\ P(G|F, H) \cdot P(H) \cdot P(I|E) \cdot P(J|D, G) \cdot P(A') \cdot P(B'). \end{gathered}$$

Ou ainda, utilizando potenciais por:

$$P(\mathbf{X}) = \Phi_A \cdot \Phi_B \cdot \Phi_C \cdot \Phi_D \cdot \Phi_E \cdot \Phi_F \cdot \Phi_G \cdot \Phi_H \cdot \Phi_I \cdot \Phi_J \cdot \Phi_{A'} \cdot \Phi_{B'}$$

Temos o conjunto de variáveis observadas que compõe a evidência $X_E = \{H, J\}$ e o conjunto de variáveis questionadas $X_Q = \{I\}$. Analisando as evidências podemos montar o conjunto $\mathbf{e}$ com os *findings* correspondentes às variáveis observadas que para o exemplo são: $\underline{e}_H = \{0, 1, 0\}$ e $\underline{e}_J = \{0, 0, 1\}$. De acordo com o método de inferência para redes Credais apresentado em 3.1.1 devemos incorporar as variáveis transparentes ao conjunto de evidências para, posteriormente, gerar os *findings* utilizados para representar cada uma das combinações de estado possíveis.

Com a informação da evidência temos a probabilidade conjunta dada por:

$$P(\mathbf{X}, \mathbf{e}) = \Phi_A \cdot \Phi_B \cdot \Phi_C \cdot \Phi_D \cdot \Phi_E \cdot \Phi_F \cdot \Phi_G \cdot \Phi_H \cdot \underline{e}_H \cdot \Phi_I \cdot \Phi_J \cdot \underline{e}_J \cdot \Phi_{A'} \cdot \Phi_{B'}$$

A inferência na rede Credal é calculada através dos máximos e mínimos obtidos pelo processo de inferência de cada uma das combinações de valores possíveis das variáveis transparentes da rede Bayesiana associada. Até agora não fizemos distinção de quais estados das variáveis transparentes estamos utilizando, faremos isso mais adiante, incorporando essa informação no conjunto da variáveis que compõe a evidência E. Para calcularmos a distribuição marginal a posteriori da variável I, devemos marginalizar todas as variáveis de $P(\mathbf{X}, \mathbf{e})$, exceto I.

$$P(I, \mathbf{e}) = \sum_{A,B,C,D,E,F,G,H,J,A',B'} \Phi_A \cdot \Phi_B \cdot \Phi_C \cdot \Phi_D \cdot \Phi_E \cdot \Phi_F \cdot \Phi_G \cdot \Phi_H \cdot \underline{e}_H \cdot \Phi_I \cdot \Phi_J \cdot \underline{e}_J \cdot \Phi_{A'} \cdot \underline{e}_{A'} \cdot \Phi_{B'} \cdot \underline{e}_{B'}. \tag{6.1}$$

Após o cálculo dos máximos e mínimos a distribuição desejada pode ser obtida através da normalização da distribuição:

$$P(I|\mathbf{e}) = \frac{P(I, \mathbf{e})}{\sum_I P(I|\mathbf{e})} \tag{6.2}$$

O próximo passo é determinar as variáveis estatisticamente relevantes para a inferência, o que pode resultar em uma redução da dimensão do problema. Faremos isto através do algoritmo Bayes-Ball explicado na Seção 2.1.4. Ressaltamos que, por serem auxiliares e artificiais, as variáveis transparentes não devem ser retiradas, inclusive, como vamos varrer as diferentes combinações de valores elas podem ser consideradas parte de uma evidência. A Figura 6.5 apresenta o resultado da aplicação do algoritmo para este exemplo, com os nós correspondentes à variáveis observadas ou transparentes em laranja. O conjunto obtido é $X_R = \{A, B, C, D, E, F, G, H, I, J, A', B'\}$. Podemos ver que, por conta das evidências e da consulta, todas as variáveis devem permanecer no grafo para a inferência.

Em seguida devemos construir o Grafo Moral para a Rede Bayesiana utilizando apenas as variáveis presentes em X_R. O resultado pode ser visto na Figura 6.6 com o Grafo Moral e os Moral

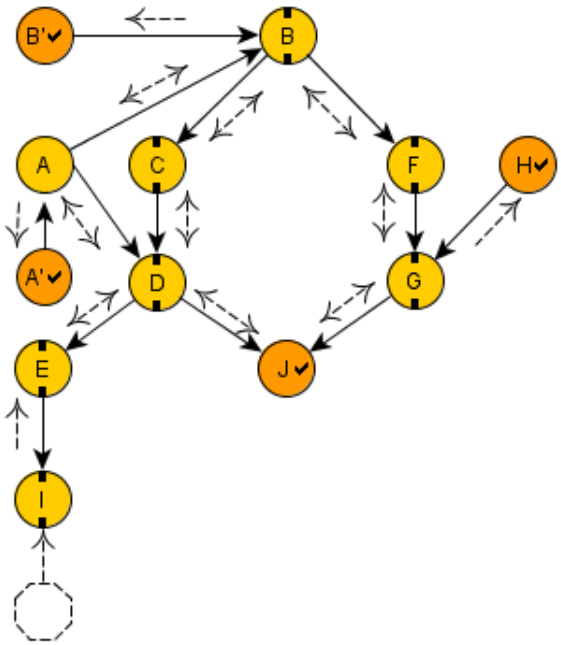


Figura 6.5: Aplicação do Bayes-Ball para definir X_R na rede de exemplo

links acrescentados para unir os vértices com filhos em comum, indicados em destaque e as variáveis transparentes em cor mais clara.

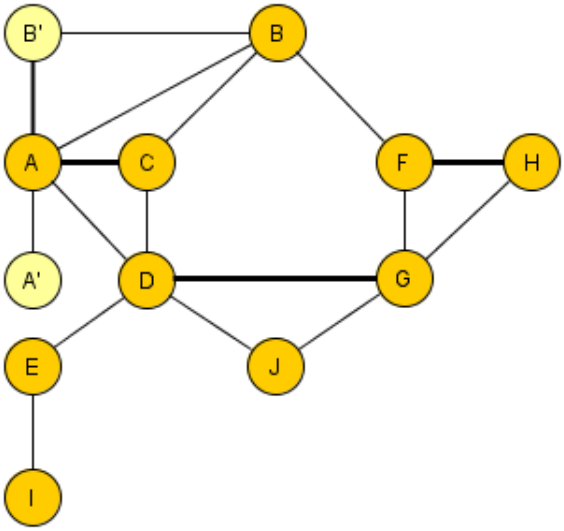


Figura 6.6: Grafo Moral da rede de exemplo

Para determinar a ordem de eliminação vamos utilizar a heurística de Gibbs/BEL passando como entrada o grafo moral da rede. Este procedimento gera uma possível árvore BEL com profundidade 5 e escolhe um nó quase periférico buscando maximizar a possibilidade de paralelização da árvore de eliminação gerada pela ordem de eliminação obtida pela heurística. Uma discussão mais detalhada sobre o assunto é apresentada em Colla (2007).

A Figura 6.7 mostra a escolha do vértice B' como vértice quase-periférico e os separadores S_1, S_2 e S_3 escolhidos como vértices da árvore de dissecção em (a) e a árvore de dissecção resultante em (b).

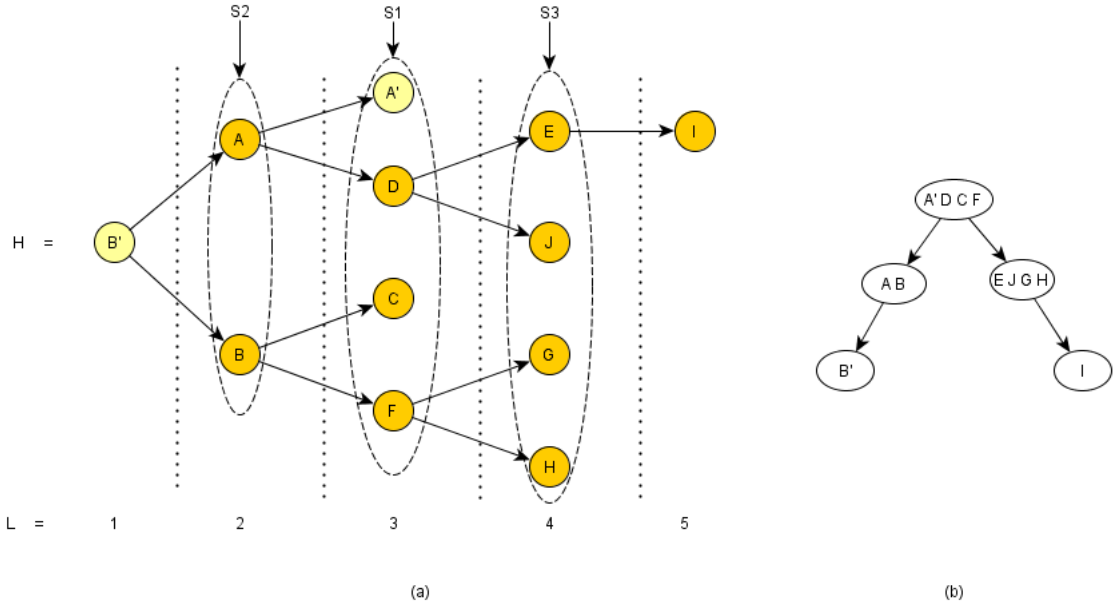


Figura 6.7: Exemplo de Árvore BEL (a) e Árvore de dissecção(b)

Enumerando os vérices da árvore de dissecção em pós ordem temos:
$\tilde{q} = [\{B'\}, \{A, B\}, \{I\}, \{E, J, G, H\}, \{A', D, F, C\}]$. A ordem de eliminação é a ordem de dissecção, que é obtida substituindo em $\tilde{q}$ cada vértice da Árvore de Dissecção pelos vértices correspondentes do Grafo Moral que a compõe. Este procedimento origina $q = [B', B, A, I, H, E, J, G, A', C, F, D]$ como uma ordem de eliminação possível.

Tomando como base a ordem de eliminação obtida e considerando a propriedade distributiva dos potenciais podemos reescrever a equação 6.1 como:

$$\begin{aligned} P(I, \mathbf{e}) = \textstyle\sum_D \sum_G \sum J\Phi_J(J, D, G)\underline{e}_J \sum_{A'} \Phi_{A'} \cdot \underline{e}_{A'} \sum_E \Phi_I(I, E)\Phi_E(E, D)\cdot \\ \cdot \textstyle\sum_F \sum_H \Phi_G(G, F, H)\Phi_H(H)\underline{e}_H \sum_C \sum_A \Phi_D(D, A, C)\Phi_A(A, A')\cdot \\ \cdot \textstyle\sum_B \Phi_F(F, B)\Phi_C(C, B) \sum_{B'} \Phi_B(B, A, B')\Phi_{B'}(B') \cdot \underline{e}_{B'}. \end{aligned} \tag{6.3}$$

É possível efetuar as operações sequencialmente, entretanto Colla, em seu trabalho (Colla, 2007) mostra que é possível realizar operações independentes de forma paralela, considerando a propriedade distributiva dos potenciais e a independência entre certas variáveis, o que pode significar em ganho computacional.

A fim de identificar as dependências de domínio entre os potenciais ou, em outras palavras, quais operações são dependentes e quais podem ser executadas em paralelo, através da ordem de eliminação obtida devemos construir a Árvore de Eliminação, principal resultado da Etapa de Fatoração Simbólica. A Árvore de Eliminação é criada através da eliminação de variáveis do grafo Moral, seguindo a ordem de eliminação. Utilizaremos o algoritmo de eliminação simplificada descrito na Seção 4.3 para economizar recursos computacionais.

A Figura 6.8 apresenta as etapas da eliminação dos vértices do grafo moral e construção da árvore de eliminação através da eliminação simplificada. O vértice circunscrito no círculo é o vértice a ser eliminado em cada etapa e o vértice inscrito no quadrado é vizinho mais próximo a ser eliminado, sobre o qual incidirão os arcos de preenchimento (em tracejado) quando necessário. Para a árvore de eliminação que vai sendo construída no interior de cada vértice está a ordem na qual o mesmo será eliminado e sobre o vértice está indicada a variável correspondente no Grafo Moral.

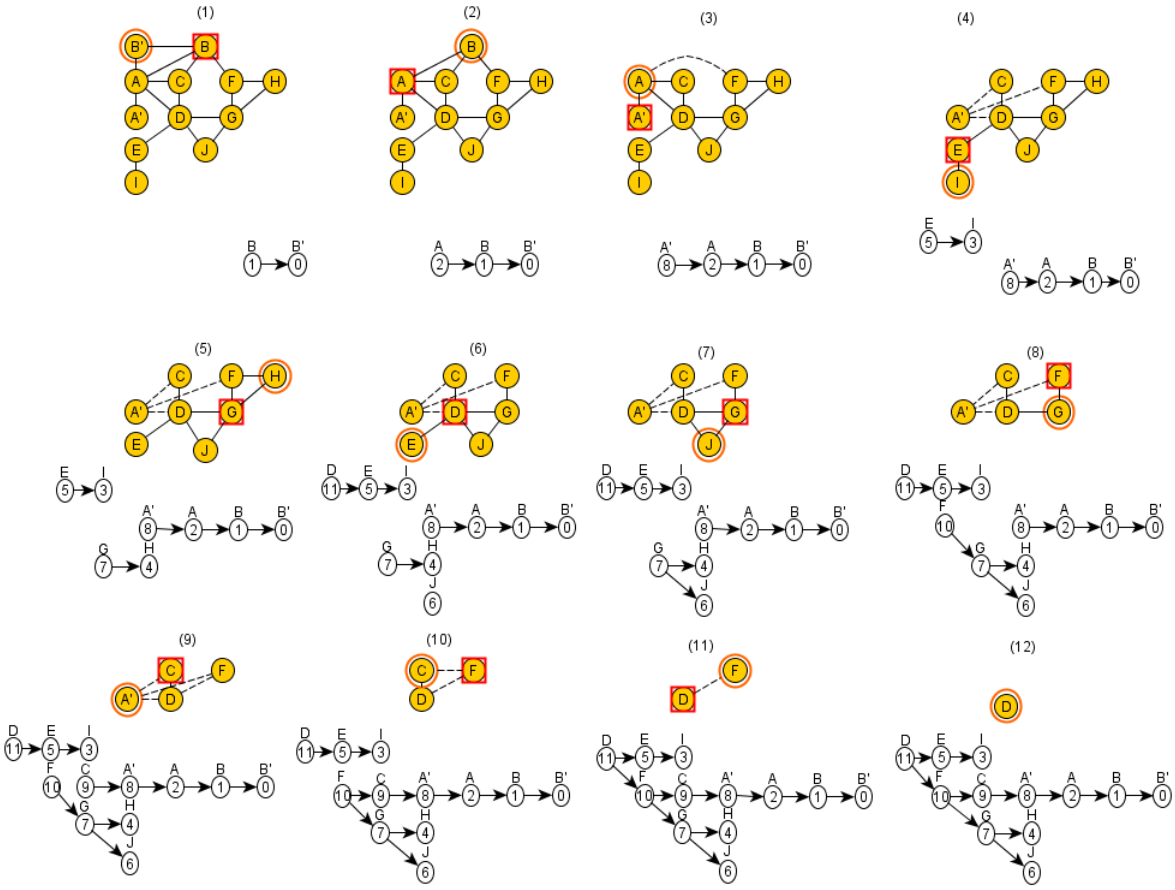


Figura 6.8: Fatoração simbólica do Grafo Moral

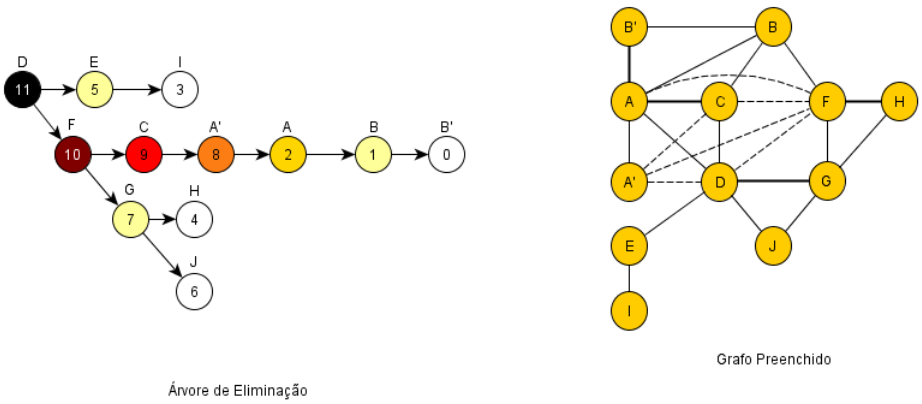


Figura 6.9: Árvore de eliminação e Grafo Preenchido obtidos através da fatoração simbólica do Grafo Moral

A árvore de eliminação e o grafo preenchido, com arcos de preenchimento em tracejado, resultantes do processo podem ser vistos na Figura 6.9. Podem ser eliminados em paralelo, ou seja, vértices que possuem eliminações independentes, aqueles que não apresentam nenhum arco, *moral link* ou arco de preenchimento que os conecte. Com isto encerramos a etapa de fatoração simbólica, ressaltando que em nenhum momento foi solicitada informação das probabilidades ou valores dos potenciais.

Utilizamos na Figura 6.9 cores iguais para indicar a possibilidade de executar os mesmos nós em paralelo na fase numérica. Como cada nó da árvore de eliminação equivale à eliminação de uma variável, cada cor indica uma fase da fatoração numérica, começando pelas folhas (nós mais claros) e conforme a execução dos nós é concluída, seguindo na direção dos pais até a raiz.

Com a ordem e a árvore de eliminação obtidas identificamos as operações que podem ser realizadas em paralelo ilustradas na Figura 6.10. Podemos observar que esta árvore possui a mesma estrutura que a árvore de eliminação.

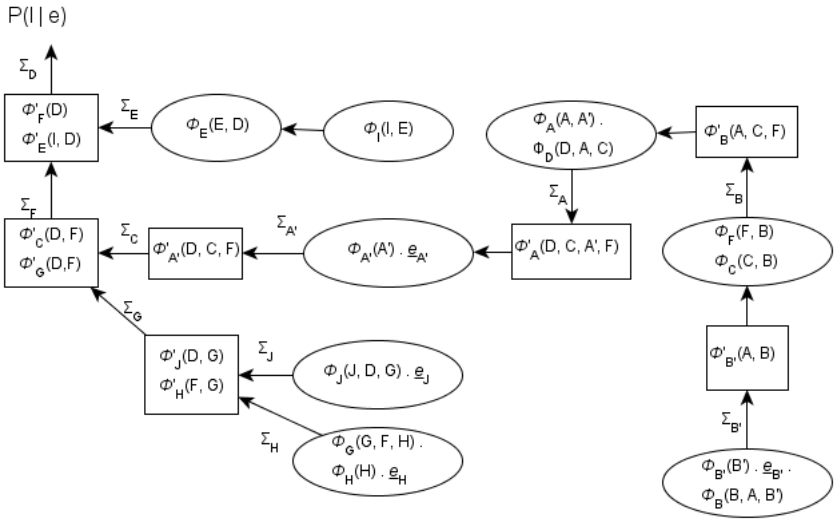


Figura 6.10: Operações com potenciais envolvidas na eliminação de variáveis

A partir deste momento precisaremos identificar sobre qual combinação de estados das variáveis transparentes estamos trabalhando. O conjunto de estados possíveis é $ev_{transp} = [\{A' = A'1, B' = B'1\}, \{A' = A'2, B' = B'1\}, \{A' = A'2, B' = B'2\}, \{A' = A'1, B' = B'2\}]$. Para cada combinação temos o conjunto de *findings* correspondente, no primeiro caso temos $\underline{e}_{A'_1} = \{1, 0\}$ e $\underline{e}_{B'_1} = \{1, 0\}$ e para os outros, respectivamente, $\underline{e}_{A'_2} = \{0, 1\}$ e $\underline{e}_{B'_2} = \{1, 0\}$, $\underline{e}_{A'_3} = \{0, 1\}$ e $\underline{e}_{B'_3} = \{0, 1\}$, $\underline{e}_{A'_4} = \{1, 0\}$ e $\underline{e}_{B'_4} = \{0, 1\}$. Vamos iniciar pelo primeiro caso.

Respeitando a sequência de operações representada na Figura 6.10 vamos iniciar o processamento numérico da rede simulando uma execução em paralelo. Observe que não utilizaremos tabelas de probabilidades para variáveis transparentes, consideraremos apenas os *findings*. Computacionalmente é possível inicializar todos os valores dos potenciais correspondentes à essas variáveis com 1 que não haverá alteração de resultado e não é preciso fazer nenhum tratamento especial.

Ciclo 1: Variáveis B', H, J e I

- Cálculo de $\Phi'_{B'}(A, B) = \sum_{B'} \Phi_B(B, A, B')\Phi_{B'}(B') \cdot \underline{e}_{B'_1}$

$\Phi_B(B, A, B')\Phi_{B'}(B') \cdot \underline{e}_{B'}$

	A1		A2		A3	
	B'1	B'2	B'1	B'2	B'1	B'2
B1	0,34	-	0,03	-	0,47	-
B2	0,48	-	0,38	-	0,46	-
B3	0,12	-	0,53	-	0,01	-

$\sum_{B'} \Phi_B(B, A, B')\Phi_{B'}(B') \cdot \underline{e}_{B'_1}$

	A1	A2	A3
B1	0,34	0,03	0,47
B2	0,48	0,38	0,46
B3	0,12	0,53	0,01

- Cálculo de $\Phi'_H(F,G)\sum_H \Phi_G(G,F,H)\Phi_H(H)\underline{e}_H$

$\Phi_H(H)\underline{e}_H$

H1	-
H2	0,03
H3	-

$\Phi_G(G,F,H)\Phi_H(H)\underline{e}_H$

	F1			**F2**		
	H1	**H2**	**H3**	**H1**	**H2**	**H3**
G1	-	0,0117	-	-	0,0153	-
G2	-	0,0183	-	-	0,0147	-

$\sum_H \Phi_G(G,F,H)\Phi_H(H)\underline{e}_H$

	F1	**F2**
G1	0,0117	0,0153
G2	0,0183	0,0147

- Cálculo de $\Phi'_J(D,G) = \sum J\Phi_J(J,D,G)\underline{e}_J$

$\Phi_J(J,D,G)\underline{e}_J$

	D1		**D2**		**D3**	
	G1	**G2**	**G1**	**G2**	**G1**	**G2**
J1	-	-	-	-	-	-
J2	-	-	-	-	-	-
J3	0,25	0,26	0,54	0,18	0,27	0,09

$\sum J\Phi_J(J,D,G)\underline{e}_J$

	G1	**G2**
D1	0,25	0,26
D2	0,54	0,18
D3	0,27	0,09

- Como nosso objetivo é calcular $P(I|e)$ a variável I não será marginalizada para fora.

Ciclo 2: Variáveis E, G, B

- Cálculo de $\Phi'_E(I,D) = \sum_E \Phi_I(I,E)\Phi_E(E,D)$

$\sum_E \Phi_I(I,E)\Phi_E(E,D)$

	D1	**D2**	**D3**
I1	0,6494	0,4496	0,7456
I2	0,3506	0,5504	0,2544

- Cálculo de $\Phi'_G(D,F)\sum_G \Phi'_H(F,G)\Phi'_J(D,G)$

$\sum_G \Phi'_H(F,G)\Phi'_J(D,G)$

	F1	**F2**
D1	0,0077	0,0076
D2	0,0096	0,0109
D3	0,0048	0,0055

- Cálculo de $\Phi'_B(A,C,F) = \sum B\Phi'_{B'}(A,B)\Phi_F(F,B)\Phi_C(C,B)$

$\sum B\Phi'_{B'}(A,B)\Phi_F(F,B)\Phi_C(C,B)$

	A1		**A2**		**A3**	
	C1	**C2**	**C1**	**C2**	**C1**	**C2**
F1	0,1732	0,3558	0,2903	0,1353	0,1345	0,4391
F2	0,2020	0,2090	0,3617	0,1527	0,1463	0,2201

Como podemos observar pela Figura 6.9 os próximos ciclos implicam em um processamento sequencial, pois a partir de agora só é possível marginalizar uma variável por vez por conta da dependência entre as variáveis restantes.

Ciclo 3: Variável A

- Cálculo de $\Phi'_A(C,D,A',F) = \sum_A \Phi_D(D,A,C)\Phi_A(A,A')\Phi'_B(A,C,F)$

$\sum_A \Phi_D(D,A,C)\Phi_A(A,A')\Phi'_B(A,C,F)$

	F1				**F2**			
	A'1		**A'2**		**A'1**		**A'2**	
	C1	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**
D1	0,0274	0,0823	0,0347	0,0958	0,0317	0,0471	0,0402	0,0546
D2	0,0693	0,0588	0,0788	0,0738	0,0799	0,0325	0,0907	0,0406
D3	0,0634	0,1946	0,0700	0,2183	0,0740	0,1132	0,0817	0,1267

Ciclo 4: Variável A'

- Cálculo de $\Phi'_{A'}(C,D,F) = \sum_{A'} \Phi'_A(D,C,A',F)\Phi_{A'}(A')\underline{e}_{A'_1}$

$\Phi'_A(D,C,A')\Phi_{A'}(A')\underline{e}_{A'_1}$

	F1				F2			
	A'1		A'2		A'1		A'2	
	C1	C2	C1	C2	C1	C2	C1	C2
D1	0,0274	0,0823	-	-	0,0317	0,0471	-	-
D2	0,0693	0,0588	-	-	0,0799	0,0325	-	-
D3	0,0634	0,1946	-	-	0,0740	0,1132	-	-

$\sum_{A'} \Phi'_A(D,C,A')\Phi_{A'}(A')\underline{e}_{A'_1}$

	F1		F2	
	C1	C2	C1	C2
D1	0,0274	0,0823	0,0317	0,0471
D2	0,0693	0,0588	0,0799	0,0325
D3	0,0634	0,1946	0,0740	0,1132

Ciclo 5: Variável C

- Cálculo de $\Phi'_C(D,F) = \sum_C \Phi'_{A'}(C,D,F)$

$\sum_C \Phi'_{A'}(C,D,F)$

	F1	F2
D1	0,1097	0,0788
D2	0,1281	0,1124
D3	0,2580	0,1872

Ciclo 6: Variável F

- Cálculo de $\Phi'_F(D) = \sum_F \Phi'_G(D,F)\Phi'_C(D,F)$

$\Phi'_C(D,F)\Phi'_G(D,F)$

	F1	F2
D1	0,0008	0,0006
D2	0,0012	0,0012
D3	0,0012	0,0010

$\sum_F \Phi'_C(D,F)\Phi'_G(D,F)$

D1	0,0014
D2	0,0025
D3	0,0023

Ciclo 7: Variável D

- Cálculo de $\Phi'_D(I) = \sum_D \Phi'_F(D)\Phi'_E(I,D)$

$\Phi'_F(D)\Phi'_E(I,D)$

	D1	D2	D3
I1	0,0009	0,0011	0,0017
I2	0,0005	0,0014	0,0006

$\sum_D \Phi'_F(D)\Phi'_E(I,D)$

I1	0,0037
I2	0,0024

Normalizando o resultado teremos $P(I|\mathbf{e_1})$

$P(I|\mathbf{e_1})$

I1	0,6050
I2	0,3950

Procedendo da mesma maneira para as outras combinações de estados para as variáveis transparentes encontramos as seguintes distribuições de probabilidade:

$P(I|\mathbf{e_2})$

I1	0,6023
I2	0,3977

$P(I|\mathbf{e_3})$

I1	0,6003
I2	0,3997

$P(I|\mathbf{e_4})$

I1	0,6030
I2	0,3970

Resta agora, para cada estado da variável I encontrar o potencial com o maior e menor valor. Temos então:

- Máximo para $I = I1$ e Mínimo para $I = I2$: $P(I|\mathbf{e_1})$,
- Mínimo para $I = I2$ e Mínimo para $I = I1$: $P(I|\mathbf{e_3})$

Com isto concluímos nosso exemplo de inferência paralela em redes Credais. A seguir apresentaremos um exemplo de processamento paralelo para a fase simbólica com a fatoração Blocada, para redes cujo grafo possui matriz de adjacência em FAB.

6.4 Exemplo de Fatoração Simbólica Blocada

Para ilustrar o procedimento de fatoração blocada utilizaremos a rede Bayesiana cujo grafo está apresentado na Figura 6.11. Uma vez que realizaremos apenas a etapa simbólica omitiremos as distribuições de probabilidade. Os arquivos que geram esta rede, as evidências, a consulta e os parâmetros para o *Simulated Annealing* estão no Apêndice C.2

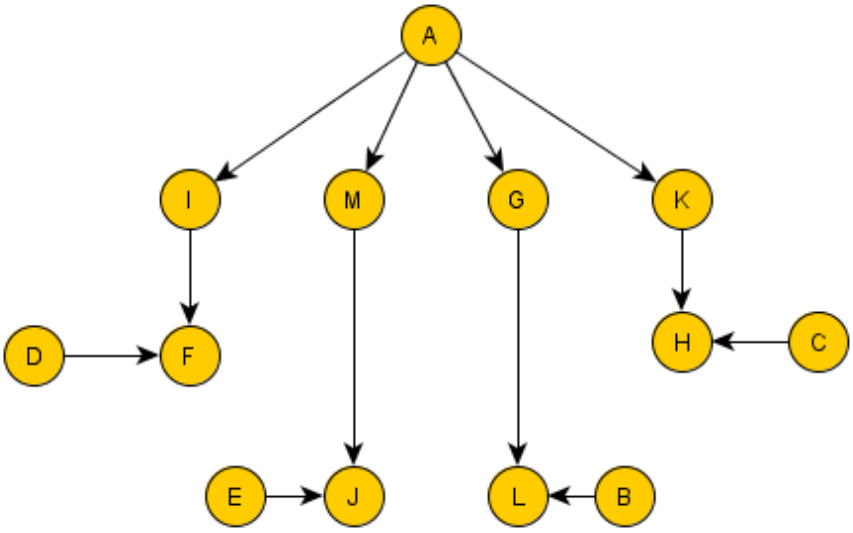


Figura 6.11: Grafo da rede Bayesiana para o exemplo de inferência blocada

Suponha que queremos calcular a probabilidade conjunta das variáveis F, H, J e L dado o estado da variável $A = A2$. Temos então $X_Q = \{F, H, J, L\}$ e $X_E = \{A\}$ com *finding* $\underline{e}_A = \{0, 1\}$. Aplicando o Bayes-Ball (2.1.4) no grafo da rede temos o conjunto de variáveis requisitadas $X_R = \{A, B, C, D, E, F, G, H, I, J, K, L, M\}$, que, mais uma vez, corresponde à todas as variáveis que compõe a rede. Particularmente neste exemplo optamos por uma consulta que levasse a isto para ilustrar mais claramente a fatoração blocada.

Em seguida construímos o grafo moral com as variáveis requisitadas que pode ser visto na Figura 6.12 com os *moral links* em destaque.

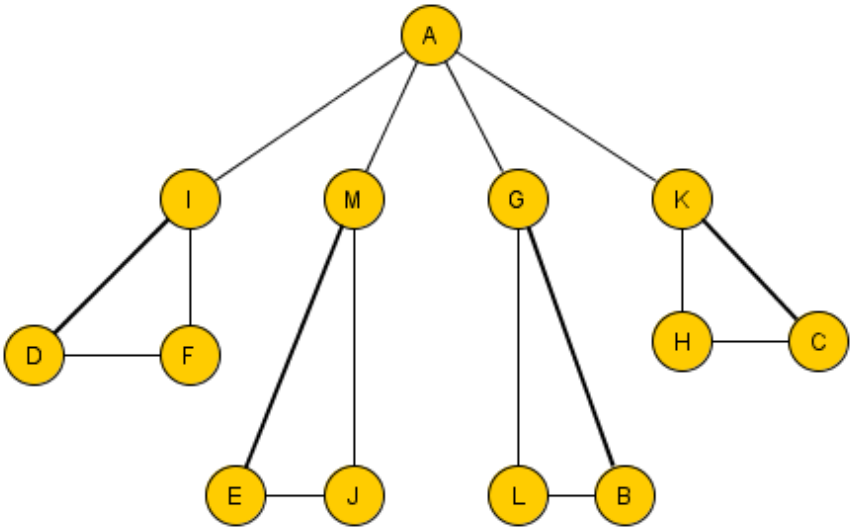


Figura 6.12: Grafo Moral da rede Bayesiana para o exemplo de inferência blocada

Para encontrar a ordem de eliminação devemos primeiramente montar a matriz de adjacência do grafo moral. Para tornar a visualização mais clara, preenchemos os elementos das diagonais com a variável correspondente:

$$M_{adj} = \begin{pmatrix} A & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & B & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & C & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & D & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & E & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & F & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & G & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & H & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & J & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & K & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & L & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & M \end{pmatrix}$$

Em seguida devemos encontrar a forma blocada da matriz de adjacência através do *Simulated Annealing*, conforme descrito em 4.4.2. A matriz blocada é obtida através de permutações de linhas e colunas em relação à matriz original. Um vetor possível de permutações é:

$$q = [I, F, D, B, G, L, K, C, H, J, M, E, A]$$

E a matriz blocada correspondente é:

$$M_{blck} = \begin{pmatrix} I & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & F & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & B & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & G & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & L & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & K & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & C & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & H & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & M & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & E & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & A \end{pmatrix}$$

Vale lembrar que esta permutação é uma das permutações possíveis que leva à matriz blocada. O desempenho do processo trocando blocos de lugar não é alterado. De posse da ordem de eliminação e da forma blocada da matriz podemos obter a árvore de eliminação de forma paralela, unido as variáveis de cada um dos blocos com as variáveis de colunas residuais (no caso somente a variável A) e fatorando cada um destes subgrafos simultaneamente. Como a fatoração de um bloco não altera a quantidade de elementos não nulos de outro, é possível antecipar o formato da árvore de eliminação com cada bloco correspondendo a um ramo. Sabemos também que a raiz sempre será necessariamente uma variável presente nas colunas residuais.

A Figura 6.13 apresenta o processo de eliminação simplificada dos vértices ocorrendo de forma paralela. Assim como na fatoração serial, o vértice circunscrito no círculo é o vértice a ser eliminado em cada etapa e o vértice inscrito no quadrado é vizinho mais próximo a ser eliminado, sobre o qual incidirão os arcos de preenchimento quando necessário. A Figura 6.13 mostra também o resultado obtido para as sub-árvores de eliminação. No interior de cada vértice da sub-árvore está a ordem na qual o mesmo será eliminado e sobre o vértice está indicada a variável correspondente no Grafo

Moral Completo.

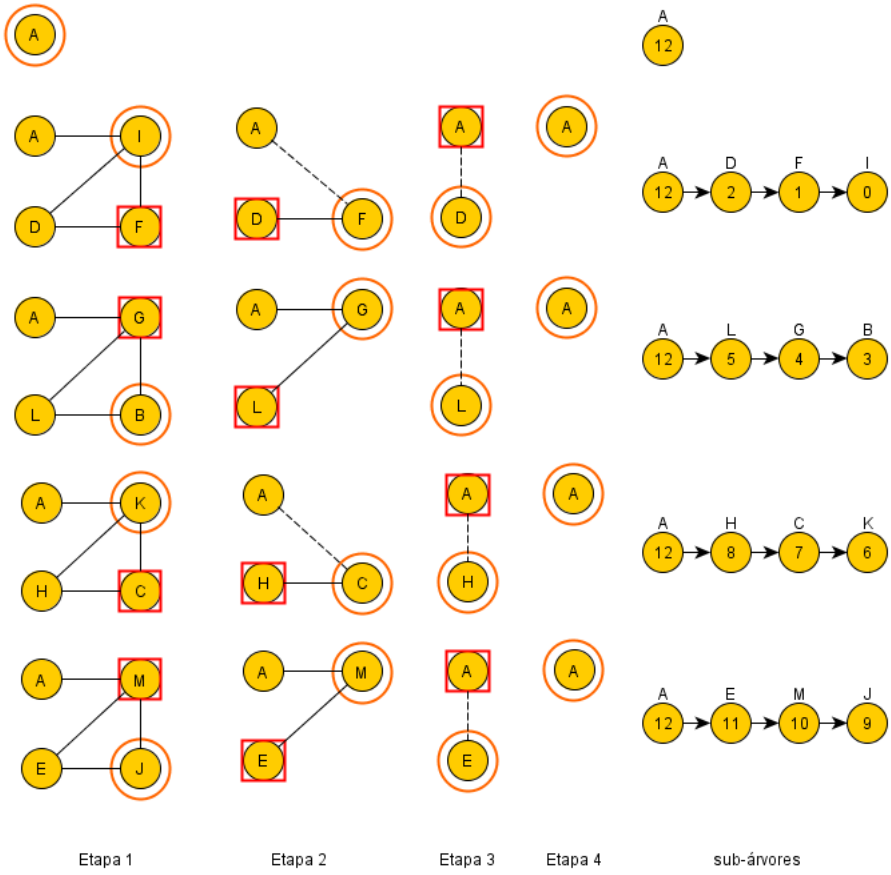


Figura 6.13: Fatoração simbólica dos subgrafos

Podemos observar que dentro de cada bloco não há tratamento para a escolha da ordem de eliminação, o que pode levar à inclusão desnecessária de arcos de preenchimento. Para blocos grandes uma solução para este problema é procurar dentro de cada um a melhor ordem de eliminação através das heurísticas de Gibbs e Busca em Largura, apenas respeitando que as variáveis das colunas residuais devem ser as últimas a serem eliminadas em cada subgrafo.

A árvore de eliminação resultante do processo pode ser vista na Figura 6.14. Utilizamos cores iguais para indicar a possibilidade de executar os mesmos nós em paralelo na fase numérica. Para montar esta árvore de eliminação basta percorrer os nós das sub-árvores resultantes de fatoração simbólica blocada buscando nós que tenham pais com variáveis de colunas residuais. Encontrando essas variáveis atualizamos a árvore resultante com um novo ramo correspondente à fatoração do bloco. Com isto termina a fase simbólica da inferência e através da árvore de eliminação é possível prosseguir com a fatoração numérica em paralelo como no exemplo anterior.

Com os dois exemplos apresentados buscamos ilustrar os passos de uma aplicação típica de inferência utilizando os algoritmos propostos.

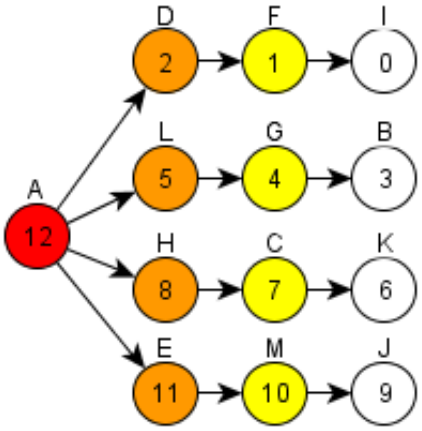


Figura 6.14: Árvore de eliminação obtida através da fatoração simbólica dos subgrafos

Capítulo 7

Implementação Computacional da biblioteca

Nosso conjunto de bibliotecas computacionais consiste em um aprimoramento de alguns métodos da versão inicial elaborada por Colla (2007), em especial na paralelização, juntamente com a adição de novas técnicas, como a inferência para Redes Credais e *Simulated Annealing* para obter a estrutura blocada de uma matriz de adjacência associada à uma rede, e alguns recursos extras como a possibilidade de salvar em arquivo o resultado obtido através da fase numérica para utilização posterior.

Buscamos aproveitar a maior parte possível da separação dos procedimentos nas bibliotecas como no trabalho original e mantivemos as mesmas premissas para o projeto e implementação das bibliotecas, a saber:

- As bibliotecas devem ser, na medida do possível, genéricas, flexíveis e com uma arquitetura que permita o desenvolvimento de aplicações com diferentes propósitos e/ou otimizadas para uma determinada etapa do processo de inferência.

- Deve haver uma separação clara e bem definida dos diferentes procedimentos e algoritmos, de forma que as bibliotecas possam ser utilizadas independentemente, ou seja, a utilização de uma biblioteca não pode estar vinculada a outra biblioteca do conjunto. Por este motivo procuramos isolar os procedimentos para que possam ser desenvolvidas ou utilizadas formas alternativas de realizar uma determinada tarefa, inclusive utilizando funções desenvolvidas por terceiros, sem inviabilizar adoção das demais rotinas que integram o conjunto de bibliotecas.

- As bibliotecas devem encapsular a complexidade dos procedimentos por meio de interfaces que tornem mais fácil e intuitivo o desenvolvimento das aplicações e possibilitem que no futuro as bibliotecas de operações mais elementares, de mais baixo nível, possam ser reescritas ou substituídas mantendo a compatibilidade com as aplicações desenvolvidas até então.

- Para facilitar a integração com aplicações e bibliotecas de terceiros, bem como para utilizar bases de testes e repositórios de dados de domínio público, sempre que possível, devem ser utilizados padrões abertos e reconhecidamente adotados pela comunidade científica.

Como no nosso sistema a inferência para redes Bayesianas é um caso particular da inferência para redes Credais, vamos explicar o funcionamento para estas últimas. Em linhas gerais, são realizadas as seguintes etapas, que serão detalhadas mais adiante:

Etapa 0 - Estimação dos Tempos para Avaliar a necessidade de Paralelizar

0.1 Cálculo do valor de MFLOPS do processador;

0.2 Cálculo do tempo utilizado em operações de memória compartilhada;

0.3 Cálculo do tempo utilizado em operações de memória distribuída;

0.4 Chamada das funções que retornam a quantidade de operações seriais e paralelas;

0.5 Execução de cálculos para escolher as etapas que serão realizadas em paralelo.

Etapa 1 - Entrada de Dados para Inferência

1.1 Leitura dos arquivos que definem a Rede Credal (ou Bayesiana);

1.2 Adição das variáveis transparentes para redes Credais (e salva a rede nova);

1.3 Leitura dos arquivos que definem a Consulta (*Query*);

1.4 Leitura dos arquivos que definem as variáveis observadas (*Evidence*);

1.5 Determinação as Variáveis Requisitadas para processar a inferência.

Etapa 2 - Fatoração Simbólica da Rede Bayesiana

2.1 Construção do Grafo Moral;

2.2 Blocagem da matriz de adjacência para permitir o processamento paralelo;

2.3 Determinação da Ordem de Eliminação;

2.4 Construção da Árvore de Eliminação e do Grafo Preenchido (em paralelo);

2.5 Criação do arquivo com o resultado da fatoração simbólica.

Etapa 3 - Criação das Redes Bayesianas a partir da Rede Credal

3.1 Enumeração das combinações de estados das variáveis transparentes como evidências;

3.2 Monitoramento das redes criadas (processo principal);

3.3 Chamada em paralelo (processamento distribuído) de inferência para cada combinação.

Etapa 4 - Fatoração Numérica da Rede Bayesiana

4.1 Carga do arquivo com as estruturas obtidas na fatoração simbólica;

4.2 Construção da Árvore de *Threads* para executar o processamento numérico;

4.3 Execução da Árvore de *Threads* para o cálculo da inferência utilizando o Método de Eliminação de Variáveis;

4.4 Compilação dos resultados obtidos para cada uma das redes para calcular os limites superior e inferior da inferência na rede Credal.

Etapa 5 - Inferência na Rede Credal

5.1 Obtenção do valor máximo das inferências em redes Bayesianas;

5.2 Obtenção do valor mínimo das inferências em redes Bayesianas.

As etapas descritas anteriormente refletem uma execução sequencial de uma inferência típica em uma rede Credal. Nada impede que sejam executadas separadamente, ou que sejam puladas, uma vez que a estrutura das bibliotecas foi feita pensando em permitir este tipo de procedimento. Por exemplo, uma vez feita a fatoração simbólica e gravada no arquivo, é possível reaproveitar estas informações para diversas evidências (útil para simulações), o cálculo dos tempos de processamento pode ser realizado somente uma vez por processador, mesmo que sejam realizadas várias inferências, algumas etapas ou subetapas podem ser eliminadas caso o usuário opte por decidir ele mesmo o que será paralelizado, eliminando a etapa 0, ou para realizar inferência em uma rede Bayesiana ao invés de rede Credal, eliminando as etapas 3 e 5.

7.1 Bibliotecas e Estruturas de Dados

Antes de detalhar o funcionamento de cada etapa presente nas bibliotecas apresentaremos a forma em que estão organizadas e também as principais estruturas de dados presentes nas mesmas. Buscaremos dar uma atenção especial aos métodos e bibliotecas novos e aos métodos presentes na biblioteca anterior que sofreram alterações na versão atual. Aconselhamos ao leitor que leia também a especificação da biblioteca original.

7.1.1 Organização das Bibliotecas

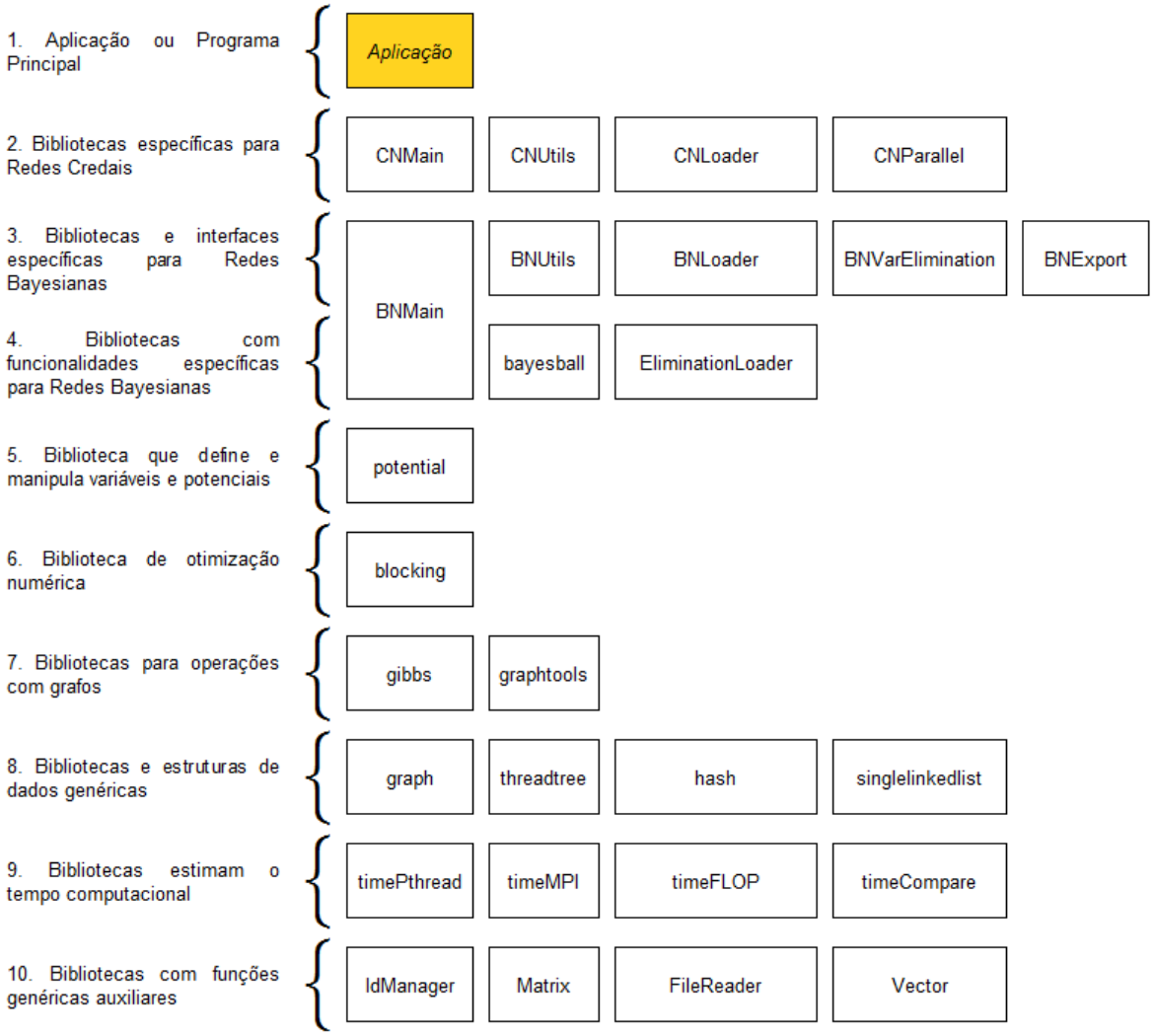


Figura 7.1: Organização dos Arquivos e Bibliotecas

Buscamos organizar as bibliotecas novas de modo similar à estrutura anterior elaborada por Colla (2007), apenas acrescentando os novos pacotes de acordo com os níveis correspondentes e níveis de acordo com a hierarquia das operações utilizadas. Tal organização visa sobretudo isolar os procedimentos relacionados à álgebra linear computacional dos utilizados para inferência, e ainda separar as Redes Credais das Redes Bayesianas, permitindo que no futuro novos métodos de inferência para redes Credais possam ser adicionados. Um diagrama com os novos níveis de separação das bibliotecas pode ser visto na Figura 7.1. Explicaremos a seguir cada um dos níveis e os arquivos presentes neles:

Nível 1 - Aplicação ou Programa Principal: A aplicação não está presente nas bibliotecas, mas é a responsável por fazer corretamente a chamada das funções de acordo com a necessidade do usuário. Ela deve ser implementada pelo desenvolvedor que for utilizar o conjunto de bibliotecas.

Nível 2 - Bibliotecas específicas para Redes Credais: Este é o nível que contém os códigos e estruturas necessários para a manipulação de redes Credais. Nele é feita a implementação das funções que representam internamente uma rede Credal e buscamos, na medida do possível, isolar estes métodos dos mais elementares, para que estas bibliotecas (e a aplicação) não precisem ser alteradas devido à alterações de níveis abaixo deste. Aqui também são definidas as funções que manipulam a rede Credal de modo a transformá-la em um conjunto de redes Bayesianas para que seja realizada a inferência.

- *CNMain*: Biblioteca que define a estrutura de dados que representa uma Rede Credal e implementa construtores, destrutores e manipuladores de informações da mesma.
- *CNUtils*: Biblioteca que implementa funções para transformar uma Rede Credal em uma rede Bayesiana com um conjunto de evidências.
- *CNLoader*: Biblioteca com as funções para carregar uma rede Credal, através de dois arquivos com as definições de uma rede Bayesiana.
- *CNParallel*: Biblioteca que realiza a inferência em uma Rede Credal de forma paralela.

Nível 3 - Bibliotecas e interfaces específicas para Redes Bayesianas: Assim como no nível anterior, este é um nível que busca abstrair suas funções dos níveis inferiores para permitir um maior reaproveitamento de código, mas desta vez para representar internamente uma Rede Bayesiana. As bibliotecas deste nível não foram alteradas substancialmente, mas há uma certa modificação em relação ao código original com pequenas melhorias de performance ou aumento de compatibilidade com ambiente Linux. Também acrescentamos funções para realizar procedimentos em paralelo e adicionamos a função para exportar informações da rede Bayesiana.

- *BNMain*: Biblioteca que define a estrutura de dados que representa uma Rede Bayesiana e implementa funções para construção, destruição e obtenção de informações da mesma.
- *BNUtils*: Interface/Biblioteca que implementa funções genéricas e úteis para o processamento de uma Rede Bayesiana, como por exemplo, a construção do Grafo Moral, fatoração simbólica ou a interface para o algoritmo de determinação das variáveis requisitadas.
- *BNVarElimination*: Biblioteca que implementa o algoritmo de inferência utilizando o método de eliminação de variáveis.
- *BNLoader*: Biblioteca com as funções para carregar uma rede Bayesiana, consultas e evidências através de arquivos com as definições das mesmas.
- *BNExport*: Biblioteca com as funções para salvar uma rede Bayesiana e evidências através de arquivos com as definições das mesmas.

Nível 4 - Bibliotecas com funcionalidades específicas para Redes Bayesianas: Neste nível se encontram arquivos para inferência em redes Bayesianas com funcionalidades específicas, no caso, a determinação das variáveis requisitadas da rede. Como testes preliminares provaram a eficiência das funções mantivemos sua estrutura intacta. As informações de uma fatoração simbólica gravadas em arquivos também são carregadas neste nível.

- *bayesball*: Implementação do algoritmo Bayes-Ball, descrito em 2.1.4, para a determinar as variáveis requisitadas para executar uma inferência, dado um conjunto de variáveis observadas.
- *EliminationLoader*: Carrega um grafo com uma árvore de eliminação e respectiva ordem de eliminação vindas de um arquivo especifico.

Nível 5 - Biblioteca que define e manipula variáveis e potenciais: Neste nível são definidas as estruturas de potenciais e variáveis utilizadas para a manipulação dos valores associados às variáveis da rede, ou seja relacionadas à Fatoração Numérica. Acrescentamos algumas funções que permitem fazer a comparação entre dois potenciais ou duas variáveis e modificamos algumas funções para garantir a unicidade dos *ids* associados a essas estruturas;

- *potential*: Define as estruturas que representam internamente uma variável e um potencial. Implementa um conjunto de operações básicas envolvendo as duas estruturas como multiplicação e marginalização de uma variável de um potencial.

Nível 6 - Biblioteca de otimização numérica: Neste nível implementamos um método de otimização que encontra a forma blocada de uma matriz, no caso da matriz de adjacência associada ao grafo da Rede Bayesiana, para posteriormente executar a eliminação de variáveis em paralelo. Caso haja vantagem, é possível substituir as bibliotecas deste nível por outros métodos de otimização sem ser necessário alterar as estruturas de dados ou funções já definidas em outros métodos.

- *blocking*: Implementa o método *Simulated Annealing* para matrizes esparsas descrito na Seção 4.4.2.

Nível 7 - Bibliotecas para operações com grafos: Implementam operações e heurísticas com grafos utilizadas em álgebra linear computacional.

- *graphtools*: Implementação da função de construção do Grafo Moral e da Fatoração Simbólica de grafos genéricos. Adicionalmente, acrescentamos uma função que gera a matriz booleana de adjacência de um grafo, e a fatoração simbólica paralela para matrizes blocadas.
- *gibbs*: Executa a Heurística de BEL/Gibbs para a determinação da ordem de eliminação dos vértices de um grafo.

Nível 8 - Bibliotecas e estruturas de dados genéricas: Este nível contém as definições e operações de estruturas genéricas de dados utilizadas pelos níveis superiores.

- *graph*: Define a estrutura e implementa operações com grafos. Adicionamos uma função para comparar dois grafos e também passamos a fazer verificação de unicidade de ids para os vértices do grafo.
- *threadtree*: Define a estrutura e implementa as funções para utilização das árvores de *threads* para o processamento numérico paralelo.
- *singlelinkedlist*: Define a estrutura e implementa operações com lista ligadas simples.

- *hash*: Implementa operações com *hash*.

Nível 9 - Bibliotecas que estimam o tempo computacional: Este nível, ausente na primeira versão, contém as bibliotecas relacionadas à estimação de tempos computacionais de operações com ponto flutuante e em paralelo, e também que avaliam a vantagem de se executar determinado procedimento de maneira paralela em relação ao mesmo procedimento de forma serial.

- *timePthread*: Estima o tempo necessário para realizar operações com *PThreads* (memória compartilhada).
- *timeMPI*: Estima o tempo necessário para realizar operações com MPI (memória distribuída)
- *timeFLOP*: Estima a quantidade de tempo necessária para se executar uma FLOP (operação de ponto flutuante).
- *timeCompare*: Com base nos tempos de processamento calculados e número de operações necessário para executar um determinado algoritmo de forma paralela e serial, avalia se é vantajoso utilizar o paralelismo.

Nível 10 - Bibliotecas com funções genéricas auxiliares: Neste nível mais inferior implementamos funções básicas com funcionalidades genéricas, úteis nos níveis superiores, mas que podem ser substituídas ou reformuladas sem necessidade de alterar as demais funções

- *IdManager*: Biblioteca para manipulação de Ids. Verifica duplicidade e fornece novos ids com valores distintos e positivos.
- *Matrix*: Biblioteca para manipulação de matrizes e resolução de sistemas lineares por mínimos quadrados.
- *FileReader*: Biblioteca para ler o conteúdo de um arquivo e colocá-lo em memória.
- *Vector*: Biblioteca para manipulação de vetores. Atualmente tem implementada a concatenação de dois vetores.

Esta separação busca isolar as estruturas de dados mais complexas e permitir um maior grau de abstração para níveis anteriores. Por exemplo, a estrutura de uma Rede Credal, definida no arquivo *CNMain.h* é conhecida somente nos níveis 1 e 2. O nível 3, que contém o arquivo *BNMain.h* com as definições da Rede Bayesiana desconhece a estrutura da rede Credal. A estrutura de rede Bayesiana, por sua vez, é conhecida pelos níveis 1, 2, 3 e 4, nos quais estão definidas estruturas e implementadas as funções específicas para trabalhar com Redes Bayesianas. Os níveis mais inferiores conhecem somente os componentes da rede Bayesiana, como o grafo, as variáveis e potenciais, ou nem isso para os níveis 8 e 9.

7.1.2 Principais Estruturas de Dados

Apresentaremos a seguir de maneira sucinta as principais estruturas de dados presentes na biblioteca. Muitas destas estruturas vieram da biblioteca original e estão mais detalhadas em (Colla, 2007) e também na própria documentação dos códigos-fonte, por isso o objetivo é apenas fornecer uma base para possibilitar o entendimento das implementações.

Grafos

Podemos ver na Figura 7.2 a representação das estruturas utilizadas para a manipulação dos grafos, e como elas se relacionam. Essas estruturas foram preservadas da implementação original da biblioteca. A principal estrutura definida é a *Graph* que define os elementos do grafo como o tipo de grafo, seu nome e seus vértices, entre outros. Esta estrutura pode ser utilizada tanto para grafos direcionais, como os utilizados em redes Bayesianas e redes Credais, como não direcionais como é o caso do grafo moral utilizado no processo de eliminação de variáveis.

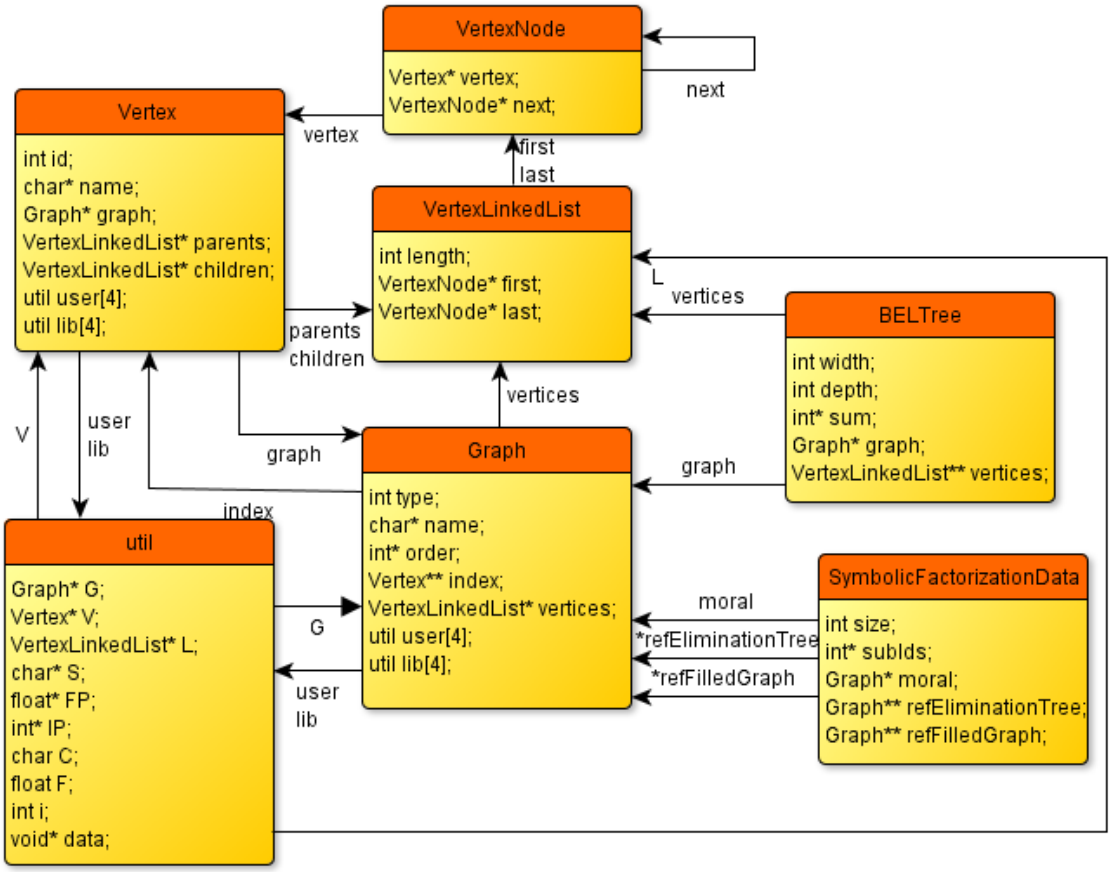


Figura 7.2: Estruturas usadas para manipulação de grafos

Um vértice é definido pela estrutura *Vertex* que contém referência para o grafo ao qual ele pertence, e também as listas de vértices pais (*parents*) e vértices filhos (*children*). Essas listas são listas ligadas simples com ponteiro para o início e fim da lista definidas como *VertexLinkedList*. No caso de um grafo não direcionado, dois vértices conectados são pais e filhos um do outro.

Internamente os vértices do grafo são indexados pelo campo *id* que permite um acesso em tempo constante aos vértices. Estes ids devem ser únicos e, por uma questão de otimização de espaço de armazenamento, preferencialmente sequenciais. Mais adiante, na Seção 7.1.2 detalharemos nossa implementação de estruturas relacionadas à garantia da geração de ids únicos, positivos e sequenciais que pode ser utilizada no campo *id* dos vértices.

Uma outra estrutura presente na Figura 7.2 é a *BELTree*. Ela é utilizada para a Busca em Largura e Heurística de Gibbs, procedimento adotado para encontrar a ordem de eliminação em grafos que não apresentam uma estrutura angular blocada. Esta árvore de busca em largura possui informação do grafo ao qual pertence (*graph*), sua profundidade (*depth*), largura (*width*) e uma lista ligada de vértices para cada um dos níveis da árvore. O campo *sum* é um vetor cujo conteúdo correspondente a cada nível da árvore é a soma acumulada do número de vértices desde o primeiro nível. Este vetor será utilizado para a escolha de uma entre as várias árvores que podem ser geradas pela Heurística de Gibbs.

Adicionamos uma nova estrutura, *SymbolicFactorizationData*, que também pode ser vista na Figura 7.2, utilizada para fazer a fatoração simbólica blocada em paralelo, através do uso de *PThreads*. Ela contém referência para a árvore de eliminação (*refEliminationTree*) e grafo preenchido (*refFilledGraph*) que serão gerados ao final da eliminação, o grafo moral (*moral*), um vetor com os ids das variáveis (*subIds*) do grafo moral envolvidas na fatoração (que é na verdade um subgrafo contendo um bloco e variáveis de colunas residuais), e a quantidade dessas variáveis (*size*). É através dessa estrutura que é possível administrar as informações de entrada e saída para as *threads*.

Lista Ligada Simples

A biblioteca *SingleLinkedList* implementa uma lista ligada simples, como comumente visto na teoria da ciência da computação. Duas características importantes desta biblioteca elaborada por Colla são a presença de ponteiros para o início e fim da lista, permitindo acesso a estes elementos em tempo constante, e a implementação de inclusão e remoção de elementos da lista de forma atômica através do uso de *mutexes* para acessar os elementos da lista, o que torna estas operações *thread-safe*, ou seja, podem ser executadas seguramente de forma paralela sem que haja problema de concorrência.

A Figura 7.3 apresenta as estruturas de dados utilizadas na implementação da lista ligada. Um exemplo de uso desta lista é para armazenar os potenciais envolvidos na operação de eliminação de variáveis.

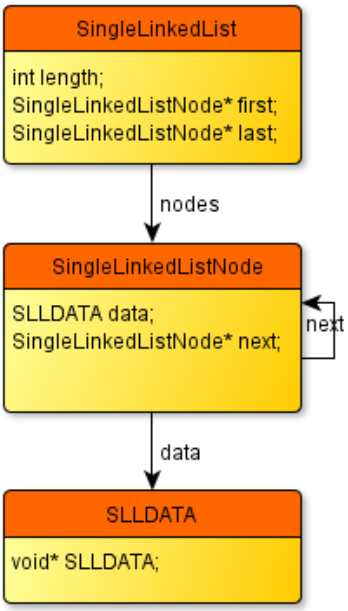


Figura 7.3: Estruturas usadas para a implementação da lista ligada simples

Hash

A Figura 7.4 mostra as estruturas de dados utilizadas para a implementação do *Hash* que é utilizado para armazenamento de pares chave-valor, no caso chaves contendo *strings* e valores contendo um *double*. Esta implementação vem da biblioteca original e segue padrões tradicionais da teoria da ciência da computação.

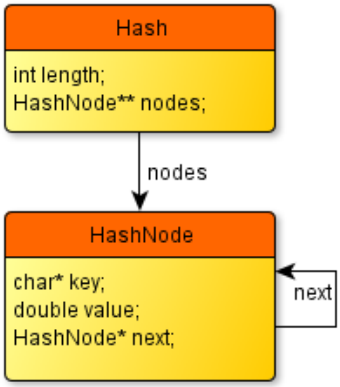


Figura 7.4: Estruturas usadas para implementação do *Hash*

Na biblioteca as tabelas de *Hash* são utilizadas para armazenar os *ids* das variáveis das redes Bayesianas e das redes Credais.

Árvore de Threads

A estratégia adotada por Colla para paralelizar o processo de eliminação de variáveis foi a criação de uma estrutura que pode ser vista como uma árvore de *threads* denominada *ThreadTree* que pode ser vista na Figura 7.5. Cada vértice desta árvore, chamado *ThreadVertex*, é uma *thread* que pode ser executada paralelamente com outros *ThreadVertex* sob a condição que os vértices filhos dele já tenham sido processados.

A topologia da árvore de *threads* é dada pela topologia da árvore de eliminação obtida na fatoração simbólica do processo de eliminação de variáveis que é instanciada como um grafo direcionado.

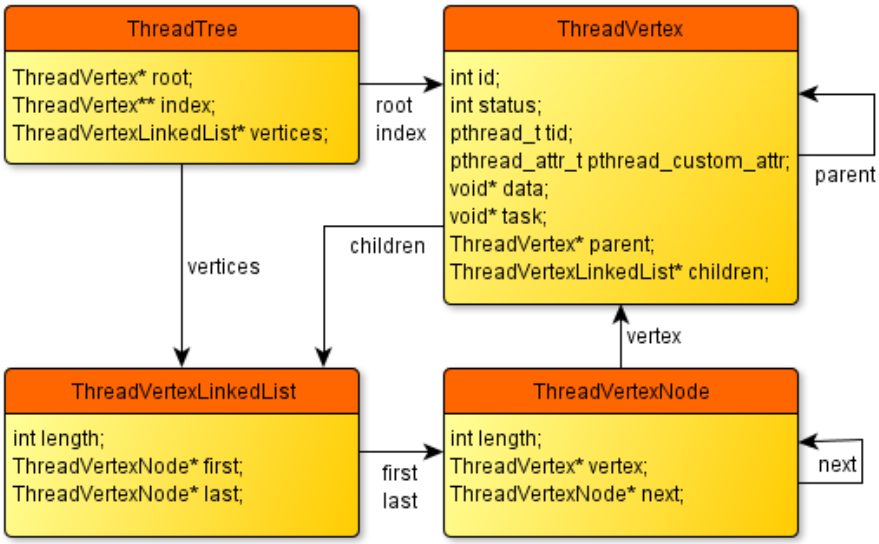


Figura 7.5: Estruturas usadas para implementação da árvore de *threads*

A eliminação dos *ThreadVertex* é feita por níveis, das folhas até a raiz, como pode ser visto na Figura 7.6. A raiz da árvore é o último *ThreadVertex* processado. Todas as *threads* são criadas no início do processamento da árvore, mas o processamento efetivo das threads é feito em etapas, de acordo com os níveis de cada vértice. Na Figura 7.6 os vértices destacados em cada passo são os vértices que estão sendo executados.

Dentro do programa utilizamos a *ThreadTree* na fatoração numérica, porém sua estrutura é genérica o suficiente para permitir que seja utilizada em outras situações que apresentem dependência entre as etapas de processamento necessárias para resolver um problema.

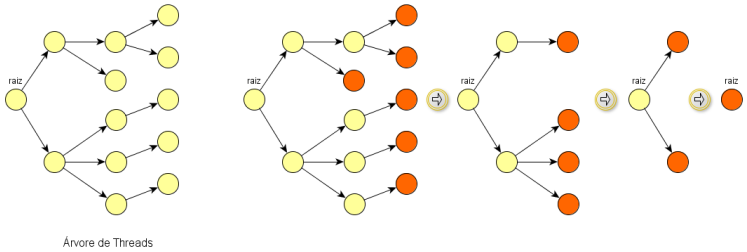


Figura 7.6: Sequência de execução de uma árvore de *threads*

Gerenciador de Ids

O gerenciamento de unicidade de *ids* nas bibliotecas é feito através da biblioteca *IdManager*. Buscando criar uma estrutura genérica o suficiente para poder ser utilizada em diferentes casos que necessitem um gerenciamento de *ids* elaboramos a estrutura que pode ser vista na Figura 7.7.

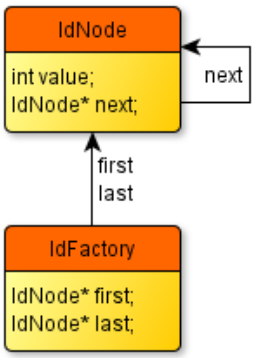


Figura 7.7: Estruturas usadas para implementação do gerador de *ids* únicas

Este gerenciador (*IdFactory*) é implementado como uma lista ligada com ponteiro para início e fim da lista (*first* e *last*, respectivamente). Cada elemento da lista é um *IdNode* que contém um valor inteiro *id* e um ponteiro para o próximo elemento da lista (*next*).

A principal razão para a implementação de uma nova lista ligada ao invés de utilizar a *SingleLinkedList* é o fato desta lista ser mais simples; os ids são sempre inteiros, dispensando a necessidade de um ponteiro para void e também a simplicidade de inserção e remoção de elementos pois não envolve operações com *threads*.

Redes Bayesianas e Eliminação de Variáveis

Uma Rede Bayesiana é definida através da estrutura *BayesNet* como pode ser visto na Figura 7.8. Ela é composta por um grafo direcionado (*Graph*), uma lista de variáveis (*Variable*) e outra de potenciais (*Potentials*) com as distribuições de probabilidade. A correspondência entre um vértice do grafo da rede, uma variável e sua tabela de potenciais é feita através do *id* que é o mesmo para todos.

Cada estado de uma variável é armazenado como uma *string* no vetor *states* e a quantidade total de estados (que são mutuamente exclusivos) é armazenada em *nstates*. O potencial possui um vetor de variáveis (*vars*), a quantidade de valores possíveis para a distribuição de probabilidade que é dada pelo produto da quantidade de valores possíveis de cada uma das variáveis associadas a ele e é armazenada em *nvalues*. Os valores da distribuição de probabilidades são armazenados no vetor *values*.

A estrutura *Elimination* define operação de eliminação de variáveis em uma rede Bayesiana. Ela está associada à uma única variável que será eliminada (*variable*), contém uma lista ligada com os potenciais que incluem esta variável (*potentials*) e também uma tabela de potenciais para armazenar o resultado da operação (*result*). O campo *marginalize* especifica se o resultado deve ser marginalizado ou não, o que varia de acordo com a consulta ou operação que estiver sendo realizada.

Para permitir o processamento paralelo da eliminação de variáveis, um *mutex* garante que a inserção dos resultados obtidos na operação sejam inseridos na lista de potenciais das *threads* dos vértices pais sem problema de concorrência entre as *threads* filhas. Este *mutex* é passado para as funções que fazem operações críticas com as estruturas de dados da *Elimination*.

Mantivemos as estruturas usadas para definir uma rede Bayesiana e operação de eliminação paralela idênticas às da biblioteca original.

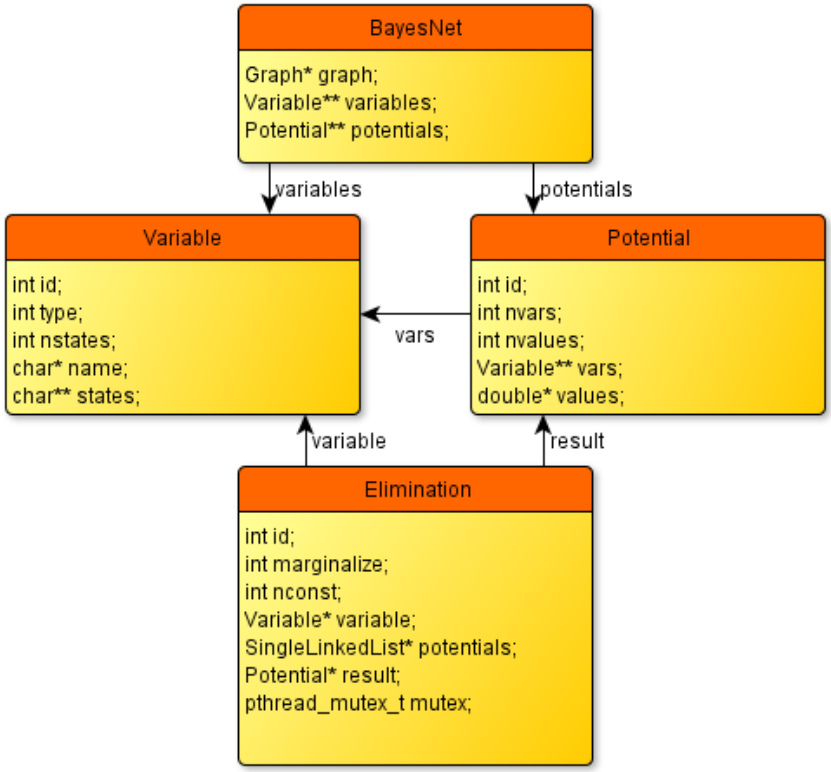


Figura 7.8: Estruturas usadas para implementação da rede Bayesiana e para o processo de Eliminação de Variáveis

Redes Credais

Buscamos implementar um estrutura para definir uma rede Credal que se aproximasse o máximo possível de uma rede Bayesiana, inclusive para entrada de dados como mostraremos mais adiante na Seção 7.3. A Figura 7.9 apresenta esta estrutura.

Uma rede Credal é composta de um grafo direcional (*graph*), uma lista de variáveis (*variables*) que podem ser do tipo discreta ou transparente e duas listas de potenciais, uma com os valores dos limites inferiores (*potentials_ lower*) e outra com os valores dos limites superiores (*potentials_ upper*). Caso determinada variável apresente um valor pontual ao invés de um intervalo basta que os limites superior e inferior possuam o mesmo valor que isto será tratado na hora da inferência.

A vantagem desta abordagem, além da entrada de dados facilitada, é que para o método de inferência escolhido, de adição de variáveis transparentes e transformação em inferência em uma rede Bayesiana com múltiplas evidências, conforme visto na Seção 3.1.1, é possível poupar processamento fazendo novas atribuições de grafos e tabelas de variáveis. É importante ressaltar que esta estrutura não cobre todos os tipos de redes Credais, outras redes, que não se aplicam ao método de inferência escolhido, podem necessitar de uma outra estrutura voltada para seu caso particular.

Uma estrutura adicional gerada desta abordagem é a *EvidCombination*, também apresentada na Figura 7.9, que contém todas as informações necessárias para gerar cada um dos conjuntos de evidências derivados da rede Bayesiana com as variáveis transparentes. Essas informações são a quantidade de conjuntos distintos (*nTotal*), a quantidade de variáveis transparentes (*nVars*), a quantidade de estados de cada umas das variáveis (*nStates*), os nomes das variáveis para cada conjunto possível de evidências (*varnames*) e os seus estados (*varstates*).

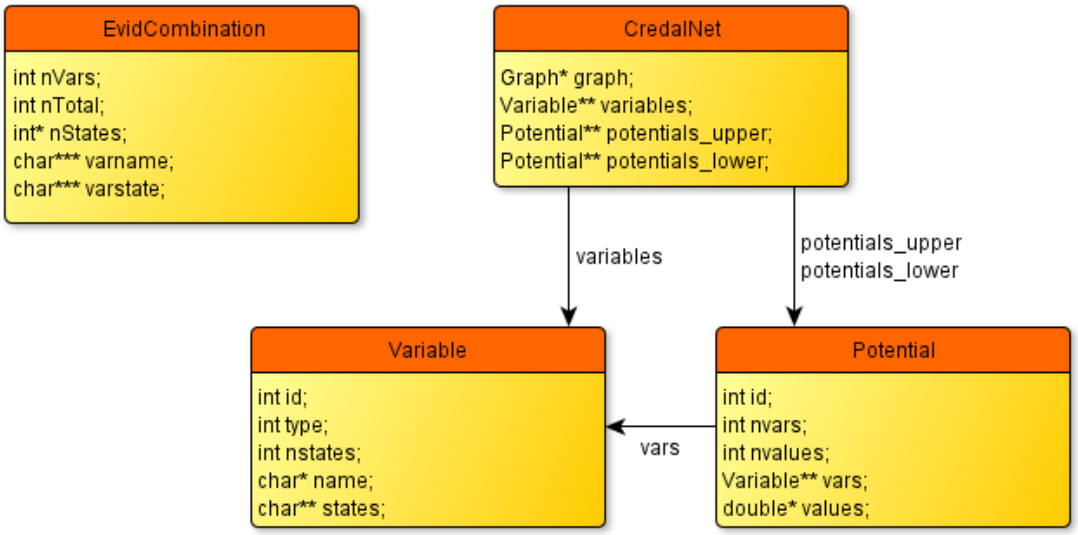


Figura 7.9: Estruturas usadas para implementação uma rede Credal

Medição de Tempo e Desempenho

Para fazer os cálculos relacionados à necessidade ou não de paralelização de algumas funções criamos algumas estruturas de dados específicas que são apresentadas na Figura 7.10.

Estas estruturas estão definidas na biblioteca *timeCompare* e são compostas somente por valores inteiros e de ponto flutuante, mas são bastante úteis para manipular as informações de medidas de tempo de *overhead* de MPI e *PThreads* e desempenho do processador (MFLOPS). Na estrutura *Flops_ Info* temos o valor de MFLOPS do processador (*average*) e outras informações relacionadas ao processo de medição que serão detalhadas mais adiante na Seção 7.2.1.

A estrutura *Pthreads_ Info* contém o número de *threads* abertas (*nthreads*), o tempo de alocação das *threads* (*overhead*), o tempo gasto para disparar e depois sincronizar a execução de uma função

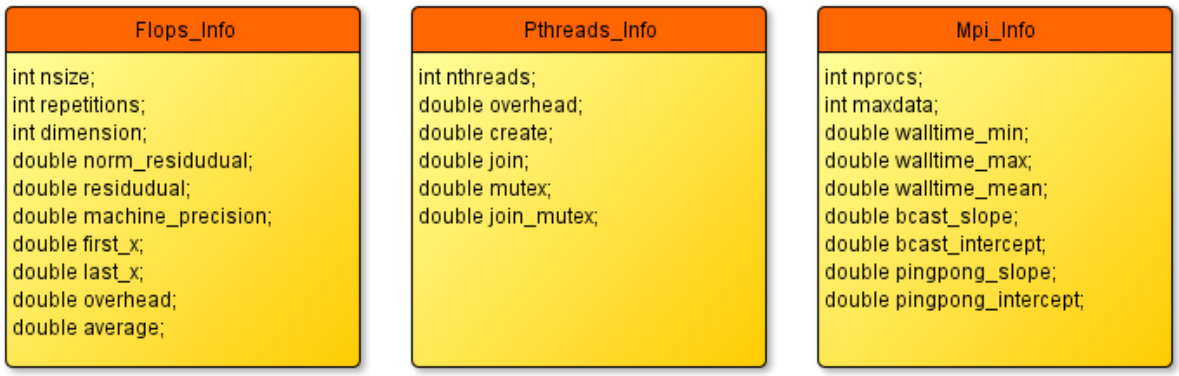


Figura 7.10: Estruturas usadas para medir tempo e desempenho

de soma simples (*create* e *join*, respectivamente), e as mesmas informações para uma função de soma simples protegida por um mutex (*mutex* e *join_mutex*, respectivamente).

Finalmente, a estrutura *Mpi_Info* contém o número de processos utilizados (*nprocs)*, informações valor mínimo, máximo e médio de gasto de tempo da função MPI_Wtime (walltime_min, walltime_max e walltime_mean, respectivamente), valor máximo de dados (*chars*) para ser enviado na comunicação (*maxdata*), e valores de inclinação e intercepto obtido após uma regressão linear dos tempos obtidos para comunicação de diferentes tamanhos de dados em broadcast e send-receive (bcast_slope, bcast_intercept, pingpong_slope e pingpong_intercept, respectivamente).

Outras Estruturas

Algumas estruturas secundárias das bibliotecas foram implementadas através de estruturas mais elementares ou tipos próprios da linguagem C. Apresentaremos agora as mais relevantes implementadas por Colla:

- *Elimination Order:* Vetor de inteiros em que a i-ésima posição corresponde ao id da i-ésima variável a ser eliminada.
- *Moral Graph:* Grafo moral utilizado no processo de eliminação de variáveis, instanciado como um *Graph* direcionado.
- *Elimination Tree:* Árvore de Eliminação com a relação de dependência entre os vértices do grafo, utilizada para montar a árvore de *threads*, instanciada como um *Graph* direcionado.
- *Findings:* Incorpora as evidências, as informações a priori sobre as variáveis, no processo. É instanciado como um *Potential* contendo uma única variável, valendo 1 na posição do estado observado, ou nos estados que passam a ser os únicos possíveis dadas as informações a priori, e 0 em todas as outras.

7.2 Estimação dos Tempos para Avaliar a necessidade de Paralelizar

O módulo de estimação de tempos e avaliação de necessidade de paralelização deve ser rodado de maneira independente do resto do programa. Mais ainda, os resultados da execução deste módulo influenciarão a chamada do programa pois as funções paralelas são separadas das funções seriais por uma questão de organização e desempenho, especialmente para o processamento distribuído como é o caso da inferência em redes Credais com MPI em que a chamada do programa deve ser feita de maneira diferente.

A Figura 7.11 ilustra os procedimentos envolvidos nesta etapa.

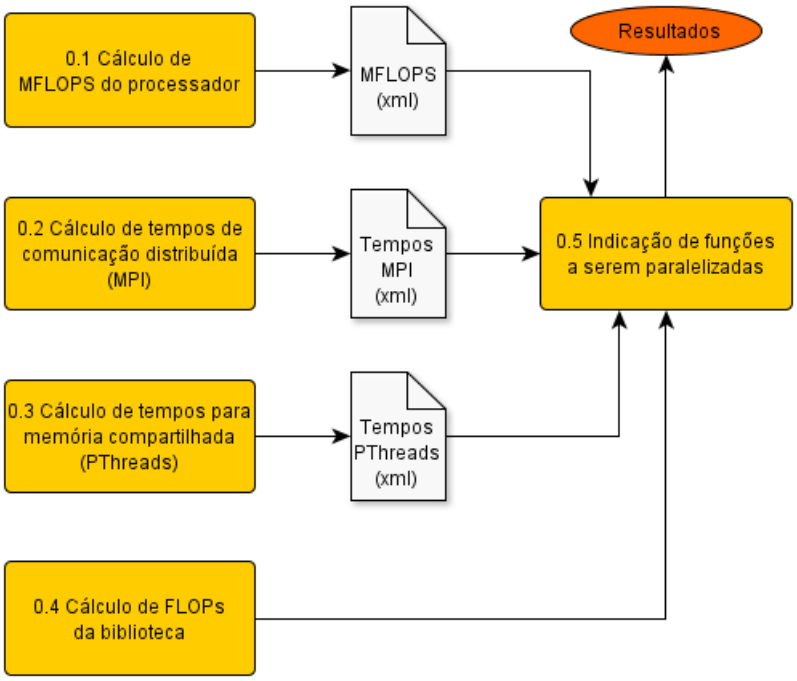


Figura 7.11: Processos e estruturas de dados relacionados à medição de tempo

O formato de arquivo para gravar as informações de saída é XML *(Extensible Markup Language)*. O XML é um formato portável, independente de *hardware* ou *software*, para a criação de documentos com dados organizados de forma hierárquica. Este formato é reconhecido e bastante utilizado pela comunidade científica, inclusive foi o formato adotado por Colla em outras bibliotecas deste conjunto.

Os arquivos XML deste módulo são formados basicamente por *tags*, ou seja, uma chave compreendida entre "<" e ">", que podem ser de início (<TAG>) ou de fim (</TAG>) e dos elementos que são as informações compreendidas entre as *tags*. Uma especificação completa do formato XML pode ser vista em http://www.w3.org/XML/

7.2.1 FLOPS

Conforme foi dito na Seção 5.4.1 para estimar a capacidade de operação de um processador, mais precisamente a quantidade de mega-FLOPS (MFLOPS) nos baseamos fortemente no *benchmark* do linpack (Dongarra, 1987) que faz a resolução de um sistema linear $Ax = b$. O arquivo *timeFlops.c* contém todas as funções utilizadas para a estimação, incluindo as funções de álgebra linear do linpack utilizadas para a resolução do sistema linear. Mantivemos essas funções praticamente inalteradas e dentro deste mesmo arquivo para ficar bastante próximo do *benchmark* original, mas, como veremos mais adiante, temos também no pacote *tools* nossa própria implementação de matriz e funções de álgebra linear que são de uso mais intuitivo e adequadas para outras funções da biblioteca.

A estimação é feita da seguinte forma, primeiro a matriz $A_{n\times n}$ e o vetor $b_{n\times 1}$ são gerados. A na verdade é um vetor que guarda a matriz A linearizada. Um elemento a_{ij} da matriz A é acessado no vetor a da seguinte forma: $A_{ij} = a[lda \cdot j + i]$, sendo que lda é a dimensão principal de A que está relacionada ao espaço utilizado para armazenar a matriz, que é necessariamente igual ou maior ao número de linhas de A. Utilizamos na biblioteca os mesmos valores utilizados originalmente no linpack: $n = 100$ e $lda = 200$.

O vetor b é gerado de forma que a solução do sistema seja o vetor unitário $x = 1$. Dessa forma, após a execução do programa é possível acompanhar se os resultados estão satisfatórios. O programa executa a resolução do sistema através da fatoração da matriz A pelo do método de eliminação de

Gauss com pivoteamento visto na Seção 4.2.2 que usa $\frac{2}{3}n^3$ operações e a resolução do sistema linear com a matriz já fatorada, que usa $2n^2$ operações. Sabendo o total de operações e o tempo que levou para resolver o sistema temos a quantidade de MFLOPS do processador.

O processo é feito 5 vezes e os resultados gravados são a média dos resultados obtidos nessas execuções. O XML de saída possui a seguinte estrutura:

```
<FLOPS_INFO>
  <NSIZE>tamanho da matriz</NSIZE>
  <NORM_RESIDUAL>resíduo normalizado</NORM_RESIDUAL>
  <RESIDUAL>resíduo</RESIDUAL>
  <MACHINE_PRECISION>precisão da máquina</MACHINE_PRECISION>
  <FIRST_X-1>valor do primeiro elemento da solução - 1</FIRST_X-1>
  <LAST_X-1>valor do último elemento da solução - 1</LAST_X-1>
  <OVERHEAD>tempo gasto para gerar a matriz A e o vetor b</OVERHEAD>
  <REPETITIONS>número de repetições</REPETITIONS>
  <DIMENSION>dimensão de armazenamento de A e b</DIMENSION>
  <AVERAGE>valor médio de MFLOPS obtido</AVERAGE>
</FLOP_SINFO>
```

O resíduo normalizado é uma medida da precisão dos resultados e deve ser $O(1)$. Caso seja superior a $O(100)$ é possível que os resultados estejam errados. O número de repetições é a quantidade de vezes que, dentro de cada execução, o sistema foi gerado e resolvido. Esse valor é calculado tomando como base a quantidade de vezes que é possível gerar os dados e fatorar a matriz em 5 segundos.

7.2.2 PThreads

Para medir o *overhead* do uso de *PThreads* estamos considerando o custo de criação, alocação, junção (*join*) e o uso de *mutexes*. O usuário escolhe quantas *threads* serão criadas através do argumento da função *pTime* junto com o nome do arquivo de saída e as informações que serão impressas na tela. Esta função, bem como as outras utilizadas para medir o tempo computacional das *PThreads* estão implementadas no arquivo *timePthreads.c*. Apesar do número de *threads* ser livre, recomendamos utilizar o mesmo número de processadores disponíveis para comunicação com memória compartilhada.

O tempo é medido através da função nativa do Linux *gettimeofday* que trabalha com uma estrutura *timeval* que contém o tempo em segundos e microssegundos (10^{-6} segundos) para armazenar os resultados e por conta disso implementamos uma função que faz a diferença entre dois valores medidos em *timeval* e armazenamos em uma nova estrutura chamada TIME_DIFF que também contém valores em segundos e microssegundos.

A função utilizada para medir o tempo é uma adição simples, com tempo computacional desprezível. Uma versão efetua diretamente a adição e a outra protege essa soma com um *mutex*. É medido o tempo de execução de cada uma delas (para o número de threads previamente estabelecido) e também da sincronização das mesmas (*joins*). Também é medido o tempo de alocação do vetor de threads, o que chamamos de *overhead* nesta função. Os resultados são gravados em um arquivo XML com a seguinte estrutura:

```
<PTHREADS_INFO>
  <NTHREADS>número de threads</NTHREADS>
  <OVERHEAD>tempo de alocação das threads</OVERHEAD>
  <CREATE>execução das threads simples</CREATE>
  <JOIN>sincronização das threads simples</JOIN>
  <MUTEX>execução das threads com mutex</MUTEX>
  <JOIN_MUTEX>sincronização das threads com mutex</JOIN_MUTEX>
</PTHREADS_INFO>
```

7.2.3 MPI

Para medir o tempo das operações de comunicação em MPI implementamos as funções disponíveis no arquivo *timeMpi.c*. Conforme dito na Seção 5.4.1 a própria função do MPI $MPI_Wtime()$ foi utilizada para medir o tempo. Para melhor precisão, inicialmente é calculado o gasto de tempo dessa função para que o mesmo possa ser descontado das operações de comunicação.

O próprio programa detecta a quantidade de processadores disponível, o que reforça que o teste apenas faz sentido se estiver sendo rodado no mesmo ambiente que o programa será executado. Através do *rank* um processo sabe se ele é *Master* (*rank* 0) ou *Slave* (*rank* diferente de 0). A função *Master* é responsável por chamar as operações do MPI, gerar estatísticas e gravar o arquivo de saída. Já as funções *Slave* apenas chamam as operações MPI.

Como o gasto de comunicação é fortemente influenciado pela quantidade de dados transmitidos, optamos por calcular o tempo de comunicação para diferentes tamanhos de massa de dados, com um limite estabelecido pelo usuário, e em posse dos resultados efetuar um ajuste dos valores a uma reta, permitindo variar posteriormente o tamanho do dado para o cálculo de *overhead*, desde que os processadores sejam os mesmos.

As funções utilizadas para fazer o ajuste estão implementadas no arquivo *Matrix.c*. Esta biblioteca contém funções para ajuste de dados a uma reta através do método dos mínimos quadrados e utiliza o método de eliminação de Gauss da Seção 4.2.2. As operações utilizam uma estrutura de dados própria *Matrix* que contém além dos dados (*chars*) as dimensões da matriz e funções de criação, destruição e operações matriciais como multiplicação e transposição que facilitam sua manipulação.

Como saída é gerado um arquivo XML com a seguinte estrutura:

```
<MPI_INFO>
  <NPROCS>número de processadores</NPROCS>
  <MAX_DATA>quantidade máxima de doubles enviados</MAX_DATA>
  <WALLTIME>
    <MIN>menor tempo obtido para MPI_Wtime()</MIN>
    <MAX>maior tempo obtido para MPI_Wtime()</MAX>
    <MEAN>tempo médio obtido para MPI_Wtime()</MEAN>
  </WALLTIME>
  <BROADCAST>
    <SLOPE>inclinação da reta de regressão para o broadcast</SLOPE>
    <INTERCEPT>intercepto da reta de regressão para o broadcast</INTERCEPT>
  </BROADCAST>
  <PINGPONG>
    <SLOPE>inclinação da reta de regressão para o pingpong</SLOPE>
    <INTERCEPT>intercepto da reta de regressão para o pingpong</INTERCEPT>
  </PIGPONG>
</MPI_INFO>
```

7.2.4 Comparação de desempenhos

As funções no arquivo *timeCompare.h* são responsáveis por fazer a leitura dos resultados obtidos de execução de MFLOPS, tempos de *overhead* de MPI e *Pthreads*, informar a quantidade de FLOPS das funções paralelizáveis do programa, atualmente:

- Fatoração numérica com *PThreads*
- Fatoração simbólica blocada com *PThreads*
- Inferência em redes Credais com MPI

e, através de uma função simples chamada *doParallel* implementar a fórmula 5.2 que recebe a quantidade de processadores disponíveis, a quantidade de FLOPS de *overhead* da função e a quantidade de FLOPS da parte paralelizavel da função e retorna 1 se a função for mais vantajosa rodando de forma paralela e 0 caso contrário. Como o *overhead* de MPI e *Pthreads* são medidos em segundos, basta multiplicar o tempo obtido nestas funções pela quantidade de MFLOPS estimada.

Por uma questão de simplificação, para computar o desempenho consideramos que operações de acesso à memória são equivalentes às FLOPs. As equações apresentadas a seguir foram obtidas diretamente do código fonte e consideram o caso médio em algumas situações e pior caso em outras. Lembramos aqui que o objetivo com esta avaliação de desempenho é dar uma direção sobre a vantagem de paralelizar determinado problema, sem ter que executá-lo previamente, e que os valores utilizados não são exatos.

Fatoração numérica com *PThreads*

Para a fatoração numérica com *PThreads* adicionamos ao *overhead* a construção da *ThreadTree*. Para uma árvore com nv variáveis são necessárias cerca de $12nv^2+57nv$ operações, correspondentes à criação dos vértices, dos arcos e do índice de vértices da *threadtree* e à destruição da árvore. Em muitos momentos, como na ordenação para criação do índice do grafo, a quantidade de operações varia com a situação do problema, e não é possível prever antecipadamente, então utilizamos o pior caso. Não contabilizamos a criação e população das *Eliminations* pois elas são necessárias tanto para executar de forma serial quanto paralela. Para o *overhead* específico do uso de *PThreads* temos a criação, alocação e sincronização de *nv-threads*.

As operações que podem ser executadas em paralelo são as operações com potenciais, criação, multiplicação, normalização e destruição. O número de processadores é limitado ao número de folhas da árvore de eliminação e o número mínimo de operações realizadas é igual ao número de níveis da arvore de eliminação. É possível executar a fatoração simbólica e obter essas informações com precisão, ou então escolher os valores que serão utilizados. A eliminação de um potencial consome cerca de nv^4+2nv^3+6nv operações. Consideramos que cada potencial tem $\frac{1}{4}$ do total de variáveis, (o que pode ser um pouco sobrestimado) e que cada variável possui dois estados (o que pode ser subestimado). Esperamos que estas estimativas resultem em um bom guia do total de operações realizadas.

Fatoração blocada com *PThreads*

Para a fatoração blocada paralela temos apenas o *overhead* de criação, alocação, sincronização e destruição de (blocos + 1) threads. O processamento paralelo para a fatoração blocada é sempre vantajoso dado que o *overhead* será pequeno em relação ao processamento dos blocos. Vamos então fazer a comparação relação à fatoração original. A fatoração simbólica serial de um grafo com nv vértices gasta cerca de $12nv^3+10nv^2+236nv-100$ operações correspondentes à criação de vértices, arcos e links, inclusão e remoção de *ids*, consulta de vértices por *id* e filhos de vértices no grafo.

Para executar a fatoração blocada temos o *overhead* da montagem dos subgrafos com os blocos e combinação dos subgrafos fatorados, que corresponde à cerca de $nres^3+5nres^2+66nres+5nv+13nv^2+102nblocs+4nvar\cdot nblocs$ operações, sendo que $nres$ é a quantidade de variáveis em colunas residuais, $nblocks$ é o número de blocos, nv é o número de variáveis no grafo completo. A fatoração paralela é feita chamando a fatoração serial para cada um dos subgrafos, inclusive o subgrafo com as variáveis de colunas residuais, portanto o número máximo de processos possível é (blocos + 1) *threads*.

Lembramos que nem todas as matrizes podem ser colocadas na forma blocada eficientemente gerando muitas colunas residuais ou simplesmente não formando blocos. Nestes casos a fatoração blocada não deve ser utilizada pois o resultado final pode ser comprometido.

Inferência em redes Credais com MPI

O *overhead* da inferência com MPI está na troca de mensagens. Da forma que está implementado, todas as mensagens são strings com o conteúdo dos arquivos xml correspondentes à rede Bayesiana, ordem de eliminação, árvore de eliminação, *query* e evidências, sendo que apenas esta última é diferente para cada processo, mas o tamanho do arquivo é essencialmente o mesmo. Assumindo que um id de variável tem em média dois caracteres, o nome quatro e que as probabilidades são definidas com duas casas decimais, são transmitidos por processo cerca de $740 + 227nv + 56na + 67nt + 25nq$ caracteres, onde nv é o número de variáveis da rede, na o número de ligações, nt a quantidade de variáveis definida com intervalos somada ao número de evidências e nq a quantidade de variáveis consultadas. Contabilizamos, além do envio propriamente, o gasto de gravar e ler os arquivos xml com as informações e a construção do grafo moral e da *ThreadTree* em cada processo, o que totaliza aproximadamente $100na + 130nt + 50nq + 1500 + 130nv + 2nv^2 + nv^3$ operações por processo. O número de processadores está limitado ao número de variáveis transparentes multiplicado pela quantidade de estados que cada uma delas possui (atualmente sempre 2, o extremo inferior e o extremo superior do intervalo).

A parte paralelizável é toda a fatoração numérica, sendo possível fazer a inferência de cada um dos conjuntos originados pela transformação CCM em um processador. É possível realizar a fatoração numérica de uma rede tanto de maneira serial quanto paralela. Deixamos isso como um parâmetro da função para que o usuário possa escolher como será feita para calcular se vale a pena utilizar também o MPI. A fatoração paralela utiliza $12nv^2 + 87nv + (nv^4 + 2nv^3 + 6nv) \cdot nl$ operações, sendo que nl é o número de níveis da árvore de eliminação e estamos considerando que estão disponíveis tantos processadores quanto o número de folhas da árvore. A fatoração serial utiliza $nv^5 + 2nv^4 + 6nv2 + 30nv$ operações aproximadamente.

Nem sempre é possível escolher quais processadores serão alocados pelo MPI, dessa forma, pode ser vantajoso optar diretamente pela fatoração numérica sequencial, pois dependendo da alocação dos processo algumas redes podem acabar disputando um mesmo processador (com memória compartilhada) para a fatoração numérica, o que ocasionaria perda de desempenho.

7.3 Entrada dos dados para Inferência

Para iniciar a inferência a primeira tarefa necessária é a entrada de informações para o programa. Estas informações são passadas através de arquivos com formatos pré-definidos que permitem a chamada automática dos métodos implementados, sem a necessidade de interação com o usuário em tempo de execução, são elas:

- Rede Credal ou Rede Bayesiana;
- Evidência observada;
- Consulta desejada;

A Figura 7.12 mostra os processos e estruturas de dados envolvidos nesta etapa, tanto para inferência em uma rede Credal (entrada com dois arquivos), quanto para inferência em uma rede Bayesiana (um arquivo). Note que nesta etapa a inferência em uma rede Credal contém um processo a mais, mas depois segue como na rede Bayesiana.

A maneira utilizada para carregar uma rede Credal nas bibliotecas é através da utilização de dois arquivos de redes Bayesianas, um com os limites inferiores e outro com os limites superiores (que podem ser iguais) de cada potencial da tabela. O resto dos arquivos, como as variáveis e relação de dependência entre elas deve ser igual nos dois arquivos, caso contrário a função retornará um erro. A especificação da rede Bayesiana, assim como nas bibliotecas originais de Colla (2007) é feita através do formato XMLBIF (XML-based BayesNets Interchange Format). Esse formato foi inicialmente desenvolvido por Fabio Cozzman e consiste em um arquivo xml, com *tags*, desenvolvido para representar grafos direcionados acíclicos com relações probabilísticas. Muitas aplicações para

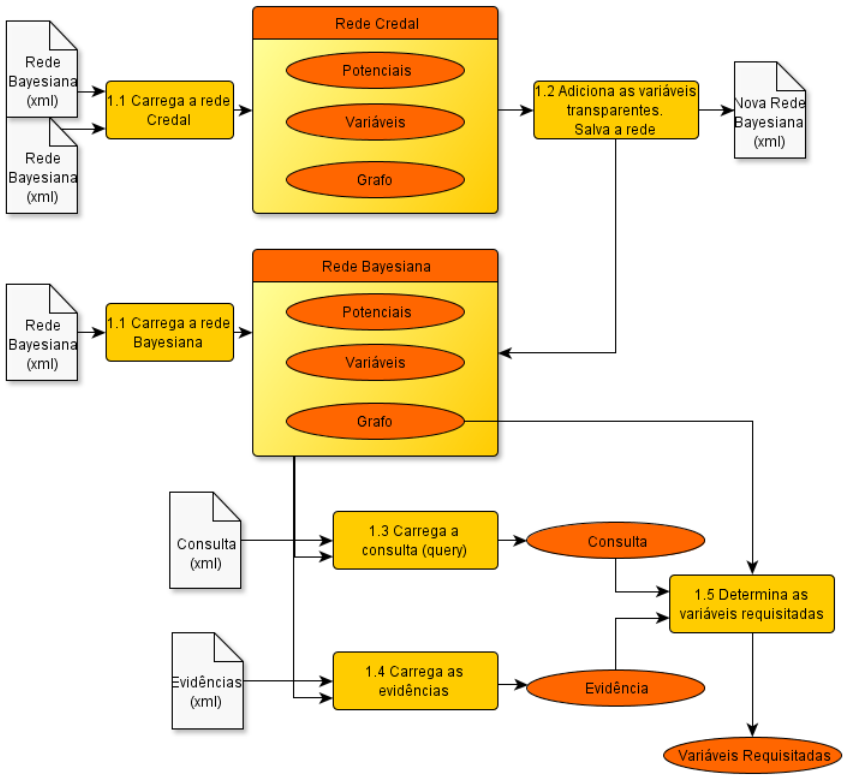


Figura 7.12: Processos e estruturas de dados relacionados à entrada de dados

redes Bayesianas, incluindo o JavaBayes (Cozman, 2001) suportam este formato. Uma descrição completa e documentação deste formato pode ser encontrada em Cozman (1998).

Adicionamos à leitura da rede uma verificação de unicidade de *ids* das variáveis, através de um conjunto de funções para criar *ids* únicos (sequenciais) e validar a inserção de novos *ids*. Em muitos lugares as bibliotecas assumem unicidade de ids e por isso optamos por garantir a consistência dos dados tirando essa obrigação do usuário. Estas funções podem ser vistas no arquivo *IdManager.c*.

Para os demais itens que devem ser especificados optamos pelo formato XML que já vem sendo adotado pelas bibliotecas. As *tags* para especificação da consulta e das evidências são as mesmas utilizadas por Colla. Uma consulta é definida com as *tags query* e *variable*. Caso queiramos conhecer a distribuição de probabilidade conjunta das variáveis A e B devemos criar um arquivo com o seguinte conteúdo:

```
<query>
  <variable>A</variable>
  <variable>B</variable>
</query>
```

a *tag query* deve aparecer somente uma vez como abertura no início do arquivo e outra como fechamento no fim do arquivo, e para cada variável da consulta abrimos e fechamos uma *tag variable* nova. O nome da variável dentro da *tag* deve estar idêntico ao contido no arquivo XMLBIF com a especificação da rede Bayesiana.

Uma evidência é definida com as *tags evidences*, *variable*, *name*, *value*. Para especificar uma evidência com a variável C assumindo o estado $C1$ e a variável D assumindo os estados $D1$ ou $D2$ devemos criar um arquivo com o seguinte conteúdo:

```
<evidences>
  <variable>
    <name>C</name>
    <value>C1</value>
```

```
  </variable>
  <variable>
    <name>D</name>
    <value>D1</value>
    <value>D2</value>
  </variable>
</evidences>
```

a *tag evidences* deve aparecer somente uma vez como abertura no início do arquivo e outra como fechamento no fim do arquivo, para cada variável da evidência abrimos uma *tag variable* nova contendo a *tag name* com o nome e uma *tag value* para cada estado possível ou observado. O nome da variável e os valores dentro das *tags* devem estar idênticos aos contidos no arquivo XMLBIF com a especificação da rede Bayesiana.

Caso estejamos trabalhando com uma rede Credal o próximo passo é acrescentar as variáveis transparentes ao grafo da rede, o que é feito através da função *credal2Bayesian* no arquivo *CNUtils.c* que recebe uma rede Credal e devolve uma rede Bayesiana com o grafo atualizado, com as variáveis transparentes. Como vimos, a estrutura da rede Credal contém os limites inferiores e superiores dos potenciais, portanto, para saber se uma variável transparente deve ser acrescentada basta verificar para cada variável se estes valores são diferentes. Como estamos trabalhando com aritmética de ponto flutuante estabelecemos um valor EPSILON para ser usado na comparação; dois valores são considerados diferentes se o módulo da diferença entre eles for maior que EPSILON. Criamos uma função que exporta uma rede Bayesiana para o formato XMFBIF disponível na biblioteca BNExport, esta função deve ser chamada para exportar a nova rede com as variáveis transparentes que será utilizada na inferência distribuída para rede Credal com MPI.

Com o grafo da rede Bayesiana resultante podemos seguir para o último passo desta etapa que é a determinação das variáveis requisitadas, ou seja, das variáveis probabilisticamete relevantes para a inferência, dada a consulta e as evidências. Trabalhar com as variáveis requisitadas pode reduzir bastante a dimensão do problema, o que leva a uma eficiência computacional, especialmente no caso de inferência para redes Credais uma vez que este processo realiza a inferência em várias redes Bayesianas. O algoritmo utilizado para esta determinação foi o Bayes-Ball descrito na Seção 2.1.4, um algoritmo simples que indica as relações de independência de variáveis de uma rede Bayesiana analisando somente na topologia do grafo e que não utiliza processamento numérico.

Como na biblioteca original, a implementação do Bayes-Ball está no arquivo *bayesball.c* e a funcionalidade de selecionar as variáveis requisitadas está encapsulada pela interface *BNUtils.c*.

7.4 Fatoração Simbólica da Rede Bayesiana

A etapa de fatoração simbólica tem como principal característica a execução dos procedimentos da inferência que não requerem valores numéricos. Após a execução desta etapa são obtidos o grafo preenchido e a árvore e ordem de eliminação. Adicionalmente, inserimos uma rotina que grava os resultados relevantes desta etapa em um arquivo, permitindo que o processamento distribuído possa ser feito diversas vezes sem necessidade de executar esta etapa novamente. Isto será útil tanto para alterar as evidências quanto para realizar a inferência em uma rede Credal. A Figura 7.13 ilustra os principais processos e estruturas de dados envolvidos nesta etapa.

A construção do grafo moral é feita considerando o subconjunto das variáveis requisitadas e o grafo da rede Bayesiana (com as variáveis transparentes se for o caso da inferência ter vindo de uma rede Credal). Nenhuma informação de distribuição de probabilidade das variáveis ou estado das variáveis observadas da rede Bayesiana é solicitada, o que reforça que não há processamento numérico nesta etapa. O algoritmo utilizado para a construção do grafo moral varre o grafo e adiciona arcos entre os pais que possuem um filho em comum.

Após a construção do Grafo Moral para encontrar a ordem de eliminação é possível percorrer dois caminhos, caso o usuário queira fazer o processamento paralelo, o próximo passo é fazer a blocagem da matriz de adjacência obtida através do grafo moral. A blocagem é feita através do

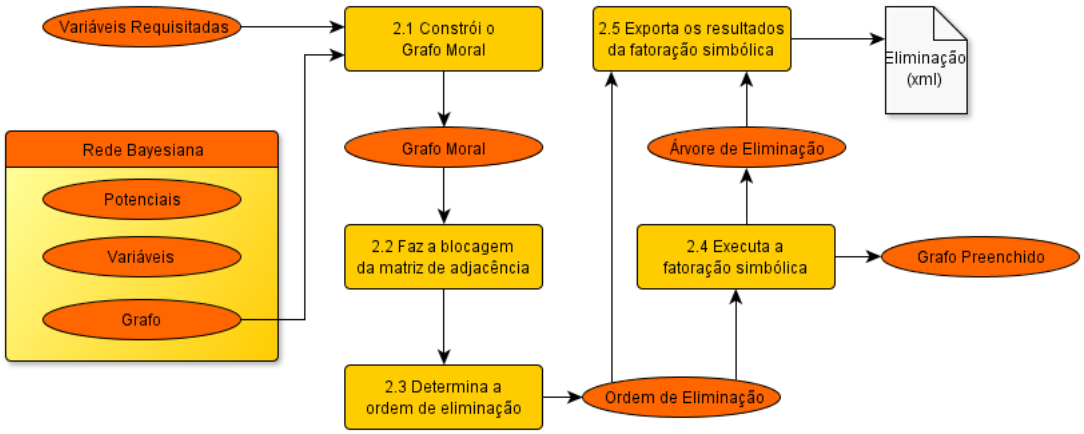


Figura 7.13: Processos e estruturas de dados relacionados à fatoração simbólica

algoritmo *Simulated Annealing* descrito na Seção 4.4.2 que busca, em uma função objetivo encontrar a configuração ótima blocada da matriz propondo mudanças de estado, que no caso são permutações dos elementos da matriz. Após encontrar a forma ótima da matriz repassamos as permutações ao grafo moral, gerando assim a ordem de eliminação. Os processos e estrutura de dados envolvidos neste momento estão representados na Figura 7.14. A Fatoração Simbólica é feita de forma paralela, idealmente com o número de processadores igual ao número de blocos da matriz mais um, utilizando o algoritmo de fatoração simbólica de Cholesky blocada descrito na Seção 4.4.1. É importante mencionar aqui que para que este processo ocorra de forma correta e eficiente é necessário que de fato a rede possa ser colocada na forma angular blocada.

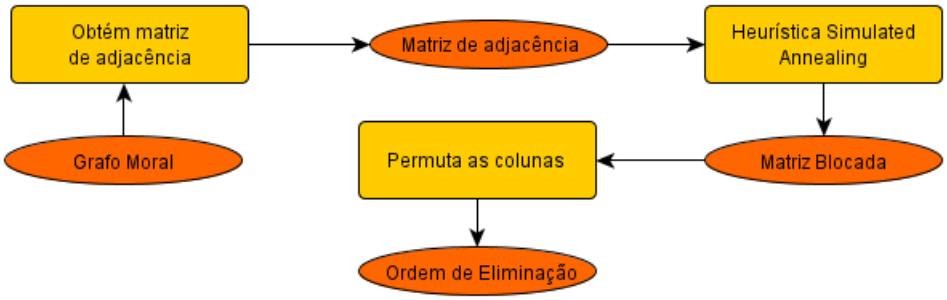


Figura 7.14: Processos e estruturas de dados relacionados à ordem de eliminação paralela

Caso o usuário opte pelo processamento serial (a matriz pode ter uma estrutura pouco blocavel, o custo de *overhead* pode ser alto, não existe mais de um processador disponível) é possível também encontrar a ordem de eliminação utilizando as heurísticas de Gibbs e Busca em Largura descritas na Seção 4.3.1 como feito por Colla (2007). Este procedimento e implementação estão detalhados em seu trabalho e por isso apenas apresentaremos graficamente a sequência lógica deste processo na Figura 7.15. Em seguida, a Árvore de Eliminação é construída através do algoritmo de Fatoração Simbólica detalhado na Seção 6.1.1. Nada impede que o usuário utilize o *Simulated Annealing* para determinar a ordem de eliminação e prossiga com este algoritmo para construir a Árvore de Eliminação.

Finalmente, unificando novamente a continuidade desta etapa, temos a criação de um arquivo de saída, no formato XML, contendo a ordem de eliminação e a Árvore de Eliminação. Reforçamos que a idéia de criar um arquivo além de trabalhar com os ponteiros para estas estruturas em memória é permitir a reutilização dos resultados obtidos em outros momentos, para que possam ser feitas alterações nas evidências, por exemplo. O arquivo contém a *tag* ORDER e entre sua abertura e

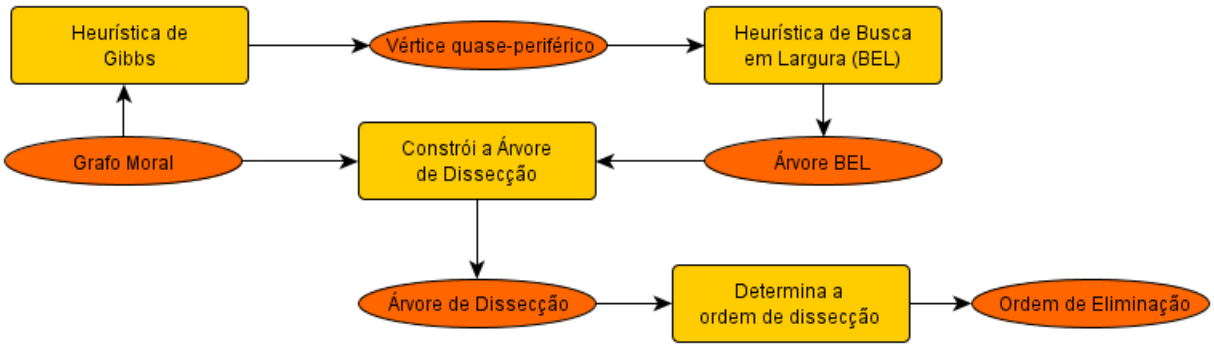


Figura 7.15: Processos e estruturas de dados relacionados à ordem de eliminação serial

fechamento os valores dos *ids* dos vértices na mesma ordem que aparecem no vetor com a ordem de eliminação, separados com espaço, e a *tag* ARC para definir cada arco da árvore de eliminação, cada par de ids que compõe o arco deve vir dentro da abertura e fechamento dessa *tag*. Segue um exemplo de arquivo com os vértices de ids 0, 1 e 2:

```
<ORDER>1 2 0</ORDER>
<ARC>0 1</ARC>
<ARC>0 2</ARC>
```

Grande parte das funções descritas aqui pode ser encontrada na biblioteca *graphtools.c*.

7.5 Criação das Evidências da Rede Bayesiana a partir da Rede Credal

Conforme vimos na Seção 3.1.1 a inferência em redes Credais será feita utilizando a transformação CCM (Cano *et al.*, 1994) que consiste resumidamente em adicionar variáveis, ditas transparentes, como pais das variáveis cujas probabilidades estão especificadas como um conjunto credal, atualizando a tabela de probabilidades da variável filha com os valores da cada extremo do conjunto credal como um estado da variável transparente e em seguida criar várias evidências, contemplando os estados possíveis das variáveis adicionadas. Do modo que a rede está implementada cada variável transparente terá sempre dois estados. Após criadas as evidências elas são gravadas em um arquivo para serem utilizadas na inferência com MPI.

A Figura 7.16 mostra os processos envolvidos nesta etapa. Nela podemos observar que a nova rede Bayesiana com as variáveis transparentes contém toda informação necessária para criar as evidências e seguir com o processamento.

Uma vez conhecido o grafo das redes Bayesianas com as variáveis transparentes, o grafo moral e ordem de eliminação de variáveis são os mesmos para todas as redes. Portanto, para aumentar os desempenho computacional e evitar cálculos repetidos, é conveniente primeiro fazer a fatoração simbólica para e depois fazer a enumeração dos *findings* e realizar a fatoração numérica para cada conjunto de evidências.

A enumeração varre o conjunto de variáveis transparentes e cria uma nova evidência para cada uma das combinações de estados possíveis das variáveis transparentes em um processo que consiste basicamente no produto cartesiano dos estados possíveis de cada uma das variáveis transparentes. A implementação da função que realiza este produto é uma adaptação da função *MB-next_ n_ tuple* da biblioteca de Martin Broadhurst para análise combinatória, que pode ser encontrada em http://www.martinbroadhurst.com/combinatorial-algorithms.html

Para permitir uma inferência em um momento posterior sem precisar gerar novamente a rede Bayesiana a partir da rede Credal é possível salvar a nova rede em formato XMLBIF e também as

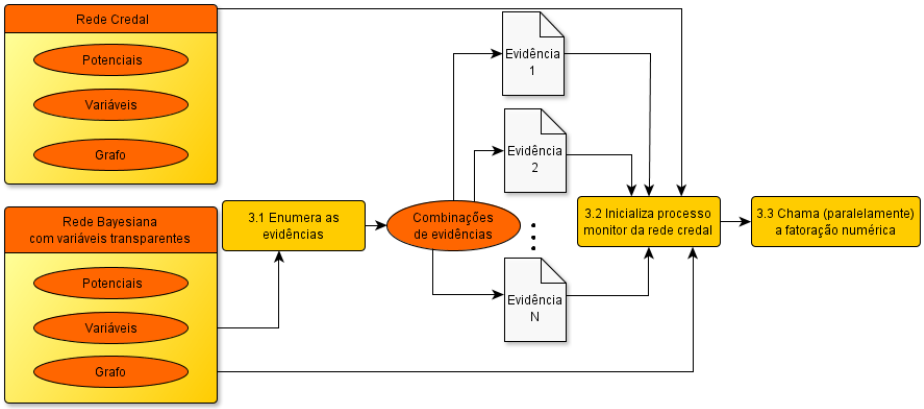


Figura 7.16: Processos e estruturas de dados relacionados à criação de evidências a partir da Rede Credal

novas evidências respeitando a estrutura XML adotada previamente. As funções responsáveis pela enumeração e pela criação da nova rede estão na biblioteca *CNUtils.c*. As funções para salvar a rede Bayesiana e as evidências foram implementadas na biblioteca *BNExport.c*.

7.6 Fatoração Numérica da Rede Bayesiana

Na etapa de Fatoração Numérica temos os cálculos relacionados à inferência envolvendo estados de variáveis observadas e também as distribuições de probabilidade das variáveis. Nesta etapa são feitas operações com produto, marginalizações e normalizações envolvendo potenciais. As estruturas obtidas nas etapas anteriores como a árvore de eliminação são utilizadas para criar um ambiente que possa efetuar os cálculos de forma paralela e eficiente.

Acrescentamos nesta etapa a leitura do arquivo XML resultante da fatoração simbólica, o que permite fazer as duas fatorações em momentos diferentes, e o envio da distribuição de probabilidade resultante para o processo pai, o que ocorre na inferência em redes Credais que explicaremos com mais detalhes adiante, no mais praticamente a totalidade das funções e bibliotecas envolvidas com processamento numérico nesta etapa foi reaproveitada do trabalho de Colla. Para facilitar o entendimento apresentaremos aqui as principais funcionalidades e funcionamento dos métodos, mas uma abordagem mais profunda pode ser vista em Colla (2007). A Figura 7.17 apresenta graficamente os processos estruturas de dados envolvidos na fatoração numérica.

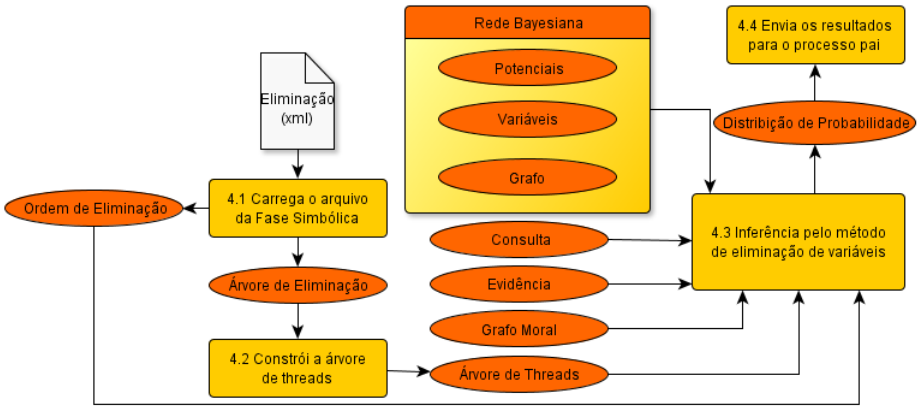


Figura 7.17: Processos e estruturas de dados relacionados à fatoração numérica

A solução adotada por Colla apresenta vantagens como permitir conhecer de antemão o número de operações de eliminações, ordem, e quais podem ser realizadas em paralelo o que possibilita alocar os recursos computacionais antes do processamento numérico, o que torna o código mais eficiente.

Além disso, torna vantajoso o uso de arquiteturas computacionais híbridas (memória compartilhada e distribuída), comuns nos computadores de alto desempenho atuais, para a inferência em redes Credais caso seja possível reservar cada nó para uma rede Bayesiana gerada e, dentro do nó, os processadores com memória compartilhada para realizar a fatoração numérica dessa rede.

Antes de iniciar o processamento é possível carregar os dados obtidos da fatoração simbólica através do arquivo XML. Caso as estruturas resultantes da fatoração simbólica ainda estejam alocadas na memória não é necessário utilizar o arquivo, é possível iniciar diretamente da criação da Árvore de Threads para paralelizar o processamento das eliminações ou pular também este passo caso a escolha seja por uma eliminação puramente sequencial.

A árvore de threads (*ThreadTree* vista na Seção 7.1.2) replica a topologia da árvore de eliminação. O número de operações de eliminação é igual ao número de variáveis requisitadas, portanto com esta medida é possível alocar o vetor de eliminações (estrutura *elimination* vista na Seção 7.1.2) em uma área de dados genérica da *ThreadTree* para que as *threads* (nós da árvore) busquem informações no momento de sua execução.

Como até o momento não foram utilizados dados numéricos, uma vez alocadas estas estruturas, é possível variar as distribuições de probabilidade e estados das variáveis observadas, desde que as variáveis em X_q e E sejam mantidas as mesmas, e aproveitar as estruturas já em memória. A definição das redes (bayesiana ou credal) permanece inalterada ao longo do processamento, dessa forma é possível alterar a evidência E sem ter que recarregar a rede.

Passos para o processamento numérico

Uma vez lido o arquivo resultante da fatoração simbólica, criada a Árvore de Threads e alocado o vetor que armazena as operações de eliminação o processamento numérico segue os seguintes passos (Colla, 2007):

1. Inicializar cada operação de eliminação do vetor de operações associando à *i-ésima* operação a *i-ésima* variável da ordem de eliminação q e alocando uma lista vazia para armazenar os potenciais que serão utilizados na operação.

2. Para cada variável pertencente à Evidência E marginalizar dos potenciais associados a elas as variáveis dos mesmos que não pertençam a X_R. Em seguida, executar o produto do potencial de cada variável pelo respectivo *finding*. O novo potencial gerado após estes dois procedimentos deverá ser associado à variável e será utilizado nas operações de eliminação.

3. Percorrer X_R acrescentando os respectivos potenciais às operações de eliminação de acordo com a ordem de eliminação. Considere o potencial ϕ_j associado a $X_j \in X_R$ com domínio $dom(\phi_j)$, seja ainda $X_k^i \in dom(\phi_j)$ a *i-ésima* variável em q e a primeira do $dom(\phi_j)$ a ser eliminada. O potencial ϕ_j deve ser acrescentado à lista de potenciais da *i-ésima* operação de eliminação.

4. Disparar a execução da Árvore de Threads. Os primeiros vértices a serem executados são os vértices que não possuem descendentes, ou seja, estão na extremidade da árvore. Os demais seguem a seguinte sequência de operações:

   (a) Cada vértice da Árvore de Threads aguarda o momento em que as execuções de todos seus descendentes tenham sido concluídas para executar a respectiva operação de eliminação.

   (b) Cada vértice da Árvore de Threads ao executar a respectiva operação de eliminação deverá:

      i. Calcular o produto dos potenciais da sua lista de potenciais.

      ii. Se a variável associada à operação de eliminação não pertencer a X_q então marginalizar o resultado do produto dos potenciais.

iii. Se o vértice executado não for a raiz da Árvore de Threads
Atribuir o resultado do seu processamento à lista de potenciais de seu pai.
Senão
Concluir a execução da Árvore de Threads.

5. Aguardar a conclusão da execução da Árvore de Threads.

6. Normalizar o resultado o última operação de eliminação, executada pela raiz da Árvore de Threads. O potencial assim obtido é a distribuição de probabilidades resultante da inferência.

Para garantir que não houvesse problemas de concorrência da inserção do potencial resultante dos filhos na lista ligada do pai durante o processamento existem funções específicas que sincronizam o acesso aos elementos da lista e garantem singularidade na manipulação dos mesmos em operações críticas. Além disso as trocas de contexto de execução e gerenciamento de concorrência estão minimizados evitando situações de *race condition*.

Caso a inferência seja em uma rede Credal, será necessário encontrar o valor máximo e mínimo de distribuição de probabilidades entre todas as redes Bayesianas derivadas desta rede. Portanto o último processo desta etapa é o envio do resultado encontrado na fatoração numérica para um processo pai (eventualmente em um outro processador caso a inferência esteja sendo feita com memória distribuída) que reunirá todos os resultados e concluirá a inferência.

7.7 Inferência na Rede Credal

A última etapa da inferência na rede Credal consiste em obter as probabilidades posteriores superior e inferior conforme mostra a Figura 7.18.

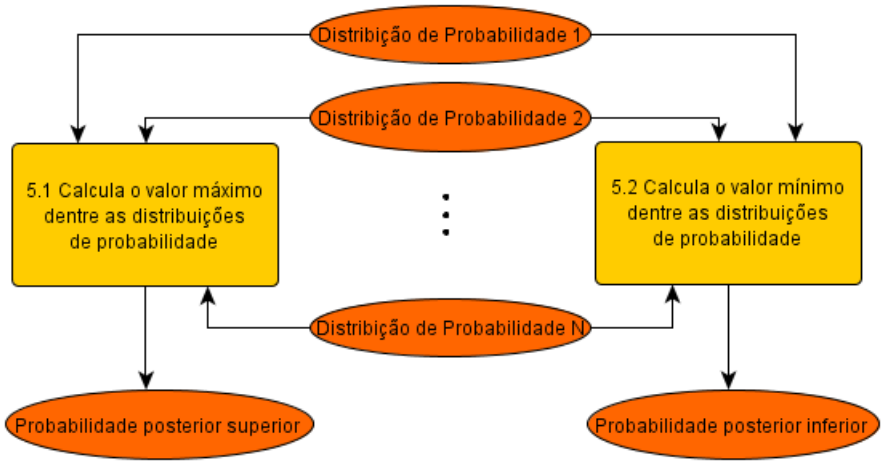


Figura 7.18: Processos e estruturas de dados relacionados à inferência na rede Credal

O processo monitor da rede Credal instanciado na etapa 3, vai receber neste momento os resultados da inferência de cada uma das evidências associadas a uma combinação de estados das variáveis transparentes. De posse desses valores encontrará o valor máximo e mínimo correspondente à probabilidade posterior superior e inferior, respectivamente. Caso o programa esteja rodando paralelamente com MPI é este processo o responsável por disparar as threads de inferência e colocar uma barreira para acumular os valores de distribuição de probabilidade nela obtidos.

A troca de mensagens do MPI é feita através de arquivos que são lidos como uma string que é enviada para os processos. Um dos parâmetros recebidos pelo MPI é o número de processos que deve ser conhecido antes da chamada do programa paralelo. Como só é possível conhecer o número ideal de processos após enumerar as combinações de estados das variáveis transparentes, foi incluído na

biblioteca um método responsável por chamar o MPI com o número correto de processos dentro do programa utilizando a função `MPI_Comm_Spawn` que é chamada após a execução da fase simbólica.

Convém reforçar que é possível executar a inferência em uma única rede Bayesiana simplesmente eliminando a etapa 3 juntamente com esta da sequência de execução.

Capítulo 8

Testes e Resultados

Neste Capítulo serão apresentados os resultados obtidos através dos testes da biblioteca. Testaremos a precisão dos resultados da implementação dos algoritmos, para avaliar se os resultados obtidos estão corretos, e performance dos mesmos para redes de diversos tamanhos para verificar se houve ganho de desempenho através das técnicas de paralelização propostas.

8.1 Testes de Precisão dos Resultados

Para avaliar a precisão dos resultados fizemos testes com as 4 redes utilizadas por Colla e mais 3 redes adicionais, uma elaborada especificamente para fatoração Blocada e duas com mais variáveis e arcos que as outras redes. Para testar redes Credais utilizamos as mesmas redes modificando apenas as tabelas de probabilidades de algumas das variáveis. Apresentaremos a seguir uma breve descrição de cada rede. Os diagramas das redes utilizadas estão no Apêndice D.

- *Exemplo01 (Colla, 2007):* Rede com 10 variáveis com 2 ou 3 estados possíveis e 11 arcos. Esta é a rede utilizada no exemplo de inferência paralela em redes Credais e foi desenvolvida com o objetivo de simular diversas situações que devem ser consideradas quando se trabalha com inferência em Redes Bayesianas.

- *DogProblem (Charniak, 1991):* Rede com 5 variáveis binárias e 4 arcos utilizada em cursos introdutórios de Redes Bayesianas. A rede é usada para identificar se uma família está em casa ou não através de vários fatores, incluindo o latido do cachorro.

- *Asia (Lauritzen e Spiegelhalter, 1990):* Rede binária com 8 variáveis e 8 arcos. Esta é uma rede fictícia utilizada para simular o diagnóstico de Tuberculose ou Câncer de Pulmão de um paciente que acabou de chegar de uma viagem da Ásia.

- *Hailfinder25 (Abramson et al., 1996):* Rede com 56 variáveis com múltiplos estados e 66 arcos. Esta rede é utilizada para previsões meteorológicas do Nordeste do Colorado.

- *Exemplo02:* Rede binária com 13 variáveis e 12 arcos utilizada no exemplo de fatoração simbólica paralela. Esta rede foi criada com o propósito específico de gerar uma matriz de adjacência blocada, com quatro blocos de 3 variáveis e uma variável residual.

- *Munin1 (Andreassen et al., 1989):* Rede com 186 variáveis com múltiplos estados e 273 arcos. Esta rede é utilizada como assistente de um sistema de eletromiografia e é uma subrede da rede Munin.

- *Pigs (Criada por Claus S. Jensen):* Rede com 441 variáveis com múltiplos estados e 592 arcos. Esta rede é composta de informações de suínos reprodutores, utilizada para realizar diagnósticos de doenças e é a maior rede que testaremos em nossas simulações.

As redes DogProblem, Asia, Hailfinder25, Pigs e Munin1 estão disponíveis na internet em repositórios de redes Bayesianas. A utilização de redes pequenas (poucos arcos e poucas variáveis) foi motivada pela possibilidade de fazer os cálculos manualmente e confrontar os resultados em diferentes etapas do processo. Já as redes maiores foram especialmente úteis para os testes de processamento paralelo pois são necessárias mais operações para a inferência, o que gera uma quantidade maior de processamento.

Os testes elaborados para avaliar a precisão foram feitos utilizando as redes descritas anteriormente e totalizam um conjunto de 80 casos buscando variar a topologia da rede, consulta e evidência. Inferências sem evidências e consultas com mais de uma variável na *query* também foram testadas. Sempre que possível os resultados foram confrontados com os obtidos manualmente e, para as redes maiores, foi utilizado o software JavaBayes (Cozman, 2001) como *benchmark*. O JavaBayes foi escolhido por ser uma aplicação de referência para inferência em redes Bayesianas ou redes Credais definidas com pontos extremos.

Para todos os testes os resultados foram os esperados, tanto para os casos simulados manualmente quanto para os confrontados pelo JavaBayes. Para os resultados com *benchmark* manual etapas intermediárias, como a árvore de eliminação (da matriz blocada ou não) também tiveram o resultado esperado. Isso tudo sugere que a implementação está correta e que é possível confiar nos resultados. Entretanto, por ainda estar em uma versão inicial, assim como Colla, sugerimos a elaboração de mais testes, que incluam centenas ou milhares de variáveis mesmo após a identificação das variáveis necessárias para a consulta. Para que estes testes sejam executados com sucesso será necessário um *cluster* com vários nós e uma quantidade suficiente de memória RAM disponível.

Uma limitação da primeira versão da biblioteca que se mantém nesta versão é a multiplicação de potenciais que aloca em memória, como um vetor, toda a tabela de valores de resultados. Por exemplo, para fazer uma multiplicação de potenciais cujo potencial resultante contém 19 variáveis cada uma com 3 estados possíveis precisaríamos alocar $3^{19} = 1162261467$ valores de ponto flutuante, o que equivale a quase dez Gigabytes em memória RAM. Como trabalho futuro para continuidade deste sugerimos a pesquisa de uma maneira mais eficiente de multiplicar grandes potenciais, que utilize menos memória RAM para armazenar o resultado, gravando parte em um arquivo, por exemplo. Uma outra solução seria que a biblioteca detectasse o mais cedo possível que a inferência não poderá ser realizada por conta da memória.

8.2 Testes de Performance

Uma vez considerados corretos os resultados gerados pela biblioteca, queremos avaliar quando há ganho de desempenho utilizando as paralelizações propostas, ou seja, a fatoração simbólica blocada paralela e a inferência em redes Credais com memória distribuída. Também aproveitamos para incrementar os testes iniciais de Colla e acrescentar uma rede maior para testar a paralelização da fase numérica com memória compartilhada proposta pelo mesmo. Por fim, com os resultados dos desempenhos obtidos utilizamos nossa biblioteca de tempos para indicar se cada caso deveria ser realizado em paralelo ou não e confrontamos os resultados.

Em todos os testes fizemos as comparações utilizando métodos das bibliotecas. Optamos por não fazer a comparação entre outros softwares pois o objetivo neste momento é avaliar o ganho de desempenho de cada algoritmo proposto, o que pode ser dificultado se houver diferença de linguagem de implementação ou entre os algoritmos presentes nas outras etapas.

Cada teste foi executado 100 vezes e o tempo apresentado nas tabelas com os resultados é o tempo médio da execuções. Omitiremos dos resultados os testes em que os tempos de execução comparados forem ambos inferiores a 10^{-4} segundos.

8.2.1 Fatoração simbólica blocada

Comparamos o desempenho da fatoração simbólica blocada paralela com a fatoração simbólica sequencial já presente na primeira versão da biblioteca. Testes iniciais demonstraram que a fatora-

ção numérica fica ineficiente caso não haja uma boa ordem de eliminação das variáveis dentro de cada bloco por isso optamos por utilizar as heurísticas de Gibbs e Busca em Largura já implementadas para o caso sequencial dentro de cada bloco, ainda que isto pudesse gerar um maior custo computacional nesta fase.

Apresentaremos o tempo total de execução e também o tempo da inferência executada na fase numérica. Como há a possibilidade de armazenar os resultados produzidos na fase simbólica é importante que haja ganho computacional na fase numérica que potencialmente será executada repetidas vezes.

Os testes foram realizados em ambiente Linux com 2 processadores em memória compartilhada com as seguintes configurações:

Processadores:	2x Intel(R) Core(TM)2 Duo 2.26GHz
Memória RAM:	8.0 Gb
Sistema Operacional:	Ubuntu 12.10 (quantal) 64-bit
Kernel:	3.5.0-27-generic
Compilador:	gcc versão 4.7.2

Os resultados estão apresentados na tabela 8.1. As colunas contém as seguintes informações:

- **Rede**: Rede Bayesiana utilizada para o teste.
- **Teste**: Caso do teste.
- **Tipo**: Algoritmo utilizado (Fatoração Paralela Blocada ou Serial).
- **nvar**: Número de variáveis requisitadas após a aplicação do Bayes-Ball.
- **larg**: Número de folhas da árvore de *threads*.
- **prof**: Maior profundidade da árvore de *threads*.
- **tamanho**: Tamanho de cada bloco.
- **quant**: Quantidade de blocos na rede.
- **tempo total**: Tempo total em segundos decorrido durante o teste.
- **tempo num**: Tempo específico para a fatoração numérica em segundos.

Rede	Teste	Tipo	nvar	larg	prof	tamanho	quant	tempo total	tempo num
Blocada01	01	Blocada	13	4	4	3	4	0,0133	0,0000
Blocada01	01	Serial	13	5	6	-	-	0,0000	0,0000
Blocada01	02	Blocada	7	2	4	3	2	0,0300	0,0000
Blocada01	02	Serial	7	2	4	-	-	0,0000	0,0000
Blocada02	01	Blocada	65	12	17	16	4	0,0900	0,0100
Blocada02	01	Serial	65	13	12	-	-	0,0067	0,0067
Blocada02	02	Blocada	33	6	17	16	2	0,0200	0,0000
Blocada02	02	Serial	33	7	12	-	-	0,0001	0,0001
Blocada03	01	Blocada	197	20	27	49	4	5,0533	4,8567
Blocada03	01	Serial	197	25	28	-	-	4,0900	4,0567
Blocada03	02	Blocada	99	5	26	49	2	1,5967	1,3367
Blocada03	02	Serial	99	12	28	-	-	1,8967	1,8967

Tabela 8.1: Resultados dos testes para fatoração simbólica blocada

Podemos ver que em alguns casos há ganho de desempenho por usar a fatoração blocada paralela. São casos com menos blocos, que conseguem explorar melhor o paralelismo nos processadores disponíveis. Apesar da pequena quantidade de casos de testes utilizada é possível afirmar neste

primeiro momento que há vantagens em utilizar a fatoração blocada paralela quando o grafo da rede possui uma estrutura condizente com o método. Há também um caso que apesar do tempo total com a fatoração blocada ter sido maior, a fase numérica foi mais rápida. Para o caso de várias inferências em uma mesma rede é interessante obter bom desempenho nesta etapa.

Testes com mais processadores e redes maiores poderiam reforçar a qualidade do método. Também seria interessante efetuar o teste em redes reais, ainda que isto implicasse em uma modificação na biblioteca para permitir blocos de tamanhos diferentes.

As figuras 8.1 a 8.4 apresentam as árvores de *threads* produzidas para os dois casos de teste da rede Blocada02. O número dentro de cada nó representa sua posição na ordem de eliminação. Vemos através delas que uma outra vantagem da fatoração simbólica blocada é que ela produz árvores de eliminação mais balanceadas e de menor profundidade. Assumindo que o tamanho dos potenciais envolvidos nas operações não difere muito entre os blocos, uma árvore balanceada induz a um maior balanceamento de carga no processamento paralelo

Com algumas modificações no código é possível dentro de cada bloco obter uma sequência de eliminação sequencial (utilizando a própria ordem de eliminação obtida pela fatoração simbólica). Tal procedimento pode minimizar as trocas de contexto para ambientes com menos processadores realizando uma maior quantidade de operações de uma vez em cada processador, o que, considerando o balanceamento adequado da árvore, pode levar a um desempenho computacional maior. Temos ainda que utilizando a árvore de *threads* com todas as variáveis, é possível realizar consultas descartando os ramos que não contenham as variáveis requisitadas sem precisar realizar uma nova fatoração simbólica, o que torna o processo de inferência mais eficiente.

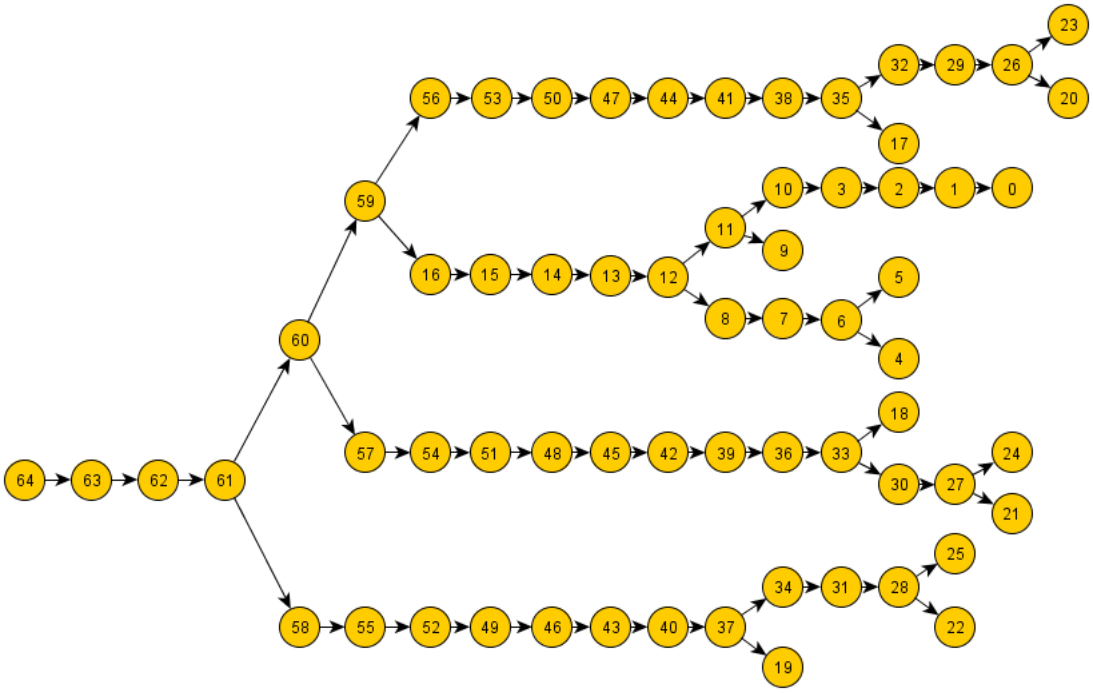


Figura 8.1: Árvore de *Threads* produzida para o teste serial 01 da rede Blocada02

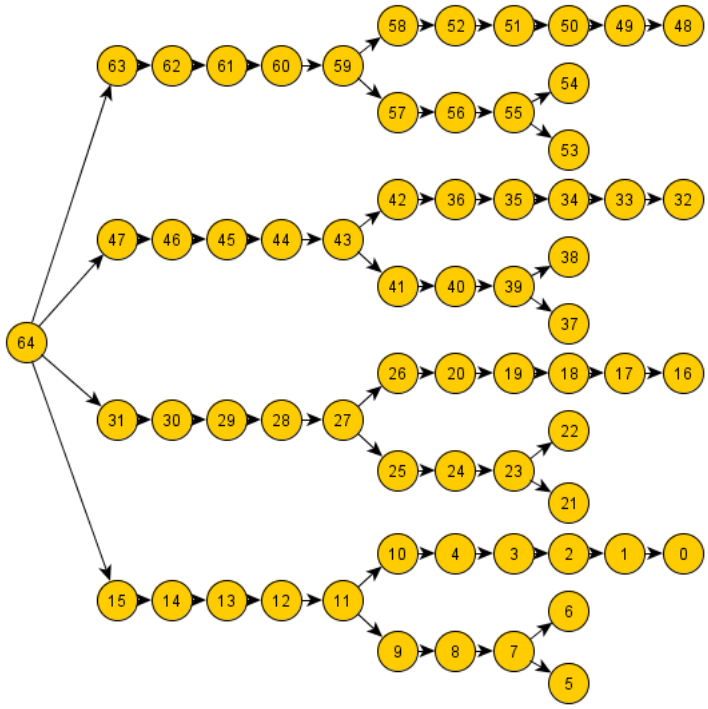


Figura 8.2: Árvore de *Threads* produzida para o teste blocado 01 da rede Blocada02

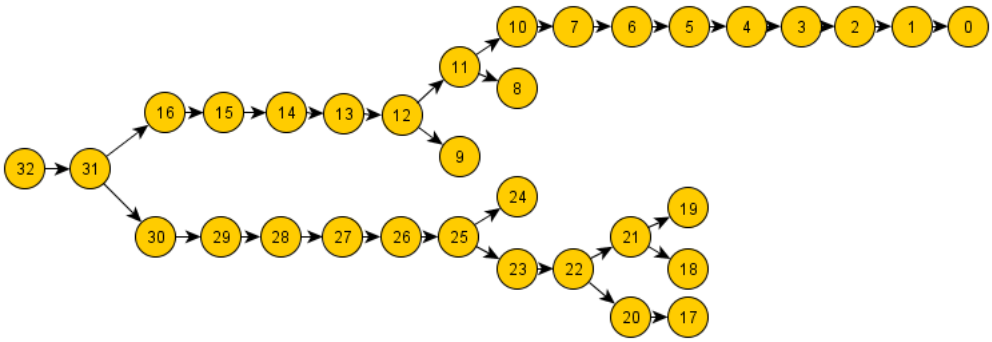


Figura 8.3: Árvore de *Threads* produzida para o teste serial 02 da rede Blocada02

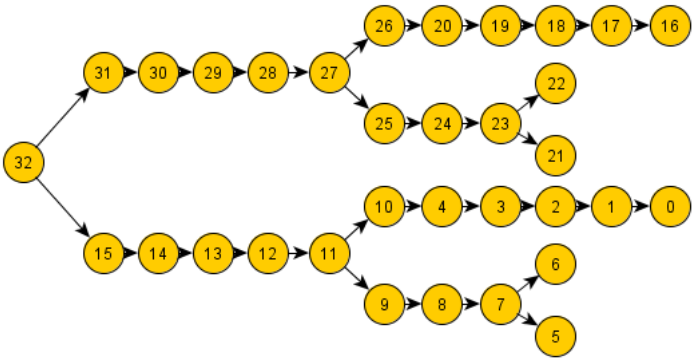


Figura 8.4: Árvore de *Threads* produzida para o teste blocado 02 da rede Blocada02

8.2.2 Inferência paralela em redes Credais

Para a inferência paralela em redes Credais utilizamos a implementação com MPI descrita em 7.7. A versão sequencial do algoritmo é bastante semelhante à paralela, com a diferença que as inferências nas redes Bayesianas geradas através das combinações de estados possíveis das variáveis transparentes são realizadas sequencialmente e que não há a necessidade de buferizar com string os conteúdos dos arquivos com as rede Bayesiana, consulta e ordem de eliminação pois essas informações já estão em memória.

Em ambos os testes é possível obter vantagem de uma fase simbólica pré processada (inclusive podendo ser feita com fatoração blocada paralela quando for conveniente de acordo com a topologia da rede). Adicionamos ainda um terceiro tipo de teste que chama a inferência paralela com MPI mas faz a fatoração numérica linearmente (sequencial). O objetivo deste teste adicional é avaliar o balanceamento de carga, uma vez que não há um controle explícito do mesmo nas chamadas MPI.

Os testes foram realizados em ambiente Linux com duas e três máquinas de quatro processadores em memória compartilhada cada. Seguem as configurações:

Processadores:	4x Intel(R) Core(TM)2 Quad 2.33GHz
Memória RAM:	16.0 Gb
Sistema Operacional:	Slackware 13.37.0 64-bit
Kernel:	2.6.38.7
Compilador:	mpicc versão 4.5.3

Os resultados estão apresentados na tabela 8.2. As colunas contém as seguintes informações:

- **Rede**: Rede Credal utilizada para o teste.
- **Teste**: Caso do teste.
- **Tipo**: Algoritmo utilizado (Inferência Serial, Paralela com Fatoração numérica paralela ou Paralela com Fatoração numérica linear).
- **nvar**: Número de variáveis requisitadas após a aplicação do Bayes-Ball.
- **nte**: Número total de variáveis que compões a evidencia, incluindo as variáveis transparentes.
- **links**: Número de links do grafo moral da rede.
- **prof**: Profundidade da árvore de eliminação.
- **2 comp**: Tempo total em segundos utilizando 2 computadores.
- **3 comp**: Tempo total em segundos utilizando 3 computadores.

Podemos ver que de forma geral para redes maiores, com mais variáveis requisitadas, o custo de envio de mensagens é menor que o gasto com a fatoração numérica, o que torna o uso do MPI eficiente. Apesar do procedimento sequencial apresentar diferença no tempo, paralelizar a fatoração numérica ou processar de forma linear não influenciou muito no resultado final por conta do número de processadores que era reduzido. Na verdade este resultado indica que a maior parte do paralelismo ficou mesmo por conta do processamento das redes geradas com as variáveis transparentes através do MPI o que sugere que o uso de mais processadores pode ser ainda mais vantajoso.

Podemos observar ainda que há ganho de desempenho com a inclusão de mais uma máquina, especialmente para as redes maiores, mais uma vez fazendo com que o custo de comunicação seja menor que o de processamento, o que reforça a vantagem de utilizar o processamento distribuído.

Rede	Teste	Tipo	nvar	nte	links	prof	2 comp	3 comp
Pigs	01	Serial	12	3	18	5	0,06030	0,06330
Pigs	01	Paralela	12	3	18	5	1,07750	0,50680
Pigs	01	Paralela Linear	12	3	18	-	1,07800	0,50320
Pigs	02	Serial	22	3	36	12	0,06260	0,07260
Pigs	02	Paralela	22	3	36	12	0,88050	0,86530
Pigs	02	Paralela Linear	22	3	36	-	1,28120	0,71140
Pigs	03	Serial	17	2	27	10	0,06130	0,06520
Pigs	03	Paralela	17	2	27	10	1,37850	0,50710
Pigs	03	Paralela Linear	17	2	27	-	1,27760	0,51830
Pigs	04	Serial	9	2	13	6	0,06050	0,06220
Pigs	04	Paralela	9	2	13	6	0,97750	0,40600
Pigs	04	Paralela Linear	9	2	13	-	0,98070	0,80540
Pigs	05	Serial	13	2	19	6	0,06040	0,06340
Pigs	05	Paralela	13	2	19	6	1,47950	0,80490
Pigs	05	Paralela Linear	13	2	19	-	1,27790	0,90490
Pigs	06	Serial	11	3	16	6	0,06070	0,06250
Pigs	06	Paralela	11	3	16	6	1,07910	0,60350
Pigs	06	Paralela Linear	11	3	16	-	1,07960	0,60520
Munin1	01	Serial	22	3	38	10	5,09910	5,13180
Munin1	01	Paralela	22	3	38	10	2,48450	2,30860
Munin1	01	Paralela Linear	22	3	38	-	2,38730	2,44890
Munin1	02	Serial	26	3	41	11	0,13920	0,15060
Munin1	02	Paralela	26	3	41	11	1,66060	1,76580
Munin1	02	Paralela Linear	26	3	41	-	1,65210	1,86700
Hailfinder25	01	Serial	23	3	50	15	8,40650	8,43440
Hailfinder25	01	Paralela	23	3	50	15	2,77780	2,60790
Hailfinder25	01	Paralela Linear	23	3	50	-	2,57720	2,07710
Hailfinder25	02	Serial	47	5	96	17	38,49340	38,61600
Hailfinder25	02	Paralela	47	5	96	17	7,84210	6,64340
Hailfinder25	02	Paralela Linear	47	5	96	-	7,96540	6,61950
Hailfinder25	03	Serial	7	3	8	3	0,01730	0,02530
Hailfinder25	03	Paralela	7	3	8	3	1,28760	1,70400
Hailfinder25	03	Paralela Linear	7	3	8	-	1,38570	1,50410

Tabela 8.2: Resultados dos testes para inferência paralela em redes Credais

8.2.3 Fatoração numérica paralela

Os testes para avaliar o ganho de processamento ao utilizar a fatoração numérica paralela foram previamente realizados em Colla (2007). Assim como ele, utilizamos uma versão linear do algoritmo que coincide com a versão paralela até o momento de obter a ordem de eliminação, não monta a árvore de eliminação nem a árvore de *threads* e que efetua as operações com potenciais de maneira serial, eliminando uma variável por vez.

Os testes foram executados no seguinte ambiente:

Processadores:	2x Intel(R) Core(TM)2 Duo 2.26GHz
Memória RAM:	8.0 Gb
Sistema Operacional:	Ubuntu 12.10 (quantal) 64-bit
Kernel:	3.5.0-27-generic
Compilador:	gcc versão 4.7.2

Assim como nos testes anteriores, dada a importância de uma fatoração numérica eficiente, além do tempo total computamos o tempo gasto exclusivamente para a parte numérica da inferência, entretanto, para os testes realizados, o tempo gasto com as outras operações foi inexpressivo em

relação à fase numérica e será omitido. Os resultados podem ser vistos na tabela 8.3.

As colunas contém as seguintes informações:

- **Rede**: Rede Bayesiana utilizada para o teste.
- **Teste**: Caso do teste.
- **Tipo**: Algoritmo utilizado (Fatoração Numérica Paralela ou Serial).
- **nvar**: Número de variáveis requisitadas após a aplicação do Bayes-Ball.
- **larg**: Número de folhas da árvore de *threads*.
- **prof**: Maior profundidade da árvore de *threads*.
- **tempo num**: Tempo específico para a fatoração numérica em segundos.

Observamos que para redes maiores com mais variáveis requisitadas há um ganho de desempenho ao utilizar a versão paralela do algoritmo, reiterando os resultados obtidos em Colla (2007). Entretanto convém mencionar que apenas a alta quantidade de variáveis não é garantia de uma boa paralelização. Nos testes para a fatoração blocada vimos que a topologia da árvore de eliminação é bastante influente no balanceamento de carga entre os processadores. Além disso, uma fator que também exerce bastante influência e é difícil de ser controlado é a quantidade de variáveis envolvida nas operações com potenciais.

Rede	Teste	Tipo	nvar	larg	prof	tempo num
Hailfinder25	03	Paralela	30	7	14	0,4167
Hailfinder25	03	Serial	30	-	-	0,2433
Hailfinder25	04	Paralela	28	7	15	0,0500
Hailfinder25	04	Serial	28	-	-	0,0400
Hailfinder25	05	Paralela	40	11	14	0,7833
Hailfinder25	05	Serial	40	-	-	1,0533
Hailfinder25	07	Paralela	29	8	15	0,0433
Hailfinder25	07	Serial	29	-	-	0,0400
Hailfinder25	08	Paralela	22	5	14	0,5333
Hailfinder25	08	Serial	22	-	-	0,3567
Hailfinder25	09	Paralela	44	14	15	4,0800
Hailfinder25	09	Serial	44	-	-	4,4500
Hailfinder25	10	Paralela	44	14	15	0,6400
Hailfinder25	10	Serial	44	-	-	0,7100
Hailfinder25	11	Paralela	45	15	15	0,4567
Hailfinder25	11	Serial	45	-	-	0,8733
Hailfinder25	12	Paralela	46	16	15	0,5867
Hailfinder25	12	Serial	46	-	-	0,8800
Hailfinder25	13	Paralela	47	17	15	0,6833
Hailfinder25	13	Serial	47	-	-	0,7033
Hailfinder25	14	Paralela	48	18	15	0,7200
Hailfinder25	14	Serial	48	-	-	0,7000
Munin1	02	Paralela	30	8	11	0,0167
Munin1	02	Serial	30	-	-	0,0333
Munin1	03	Paralela	24	5	10	0,0133
Munin1	03	Serial	24	-	-	0,0133
Munin1	04	Paralela	30	8	11	0,0233
Munin1	04	Serial	30	-	-	0,0300
Munin1	05	Paralela	30	8	11	0,0233
Munin1	05	Serial	30	-	-	0,0267
Munin1	06	Paralela	22	5	11	0,0000
Munin1	06	Serial	22	-	-	0,0000
Munin1	11	Paralela	26	8	10	0,0100
Munin1	11	Serial	26	-	-	0,0100
Munin1	12	Paralela	35	9	17	0,6233
Munin1	12	Serial	35	-	-	0,6233
Munin1	13	Paralela	50	13	14	0,0433
Munin1	13	Serial	50	-	-	0,0533
Pigs	01	Paralela	5	2	3	0,0000
Pigs	01	Serial	5	-	-	0,0033

Tabela 8.3: Resultados dos testes para fatoração numérica paralela

8.2.4 Previsão de desempenho

Utilizando a biblioteca de avaliação de tempos buscamos a indicação de quais casos deveriam ser executados em paralelo entre os apresentados anteriormente. Os resultados podem ser vistos nas tabelas 8.4, 8.5 e 8.6. Nelas apresentamos os tempos efetivamente gastos nas redes testadas e a coluna Previsão que marca com um asterisco qual configuração foi indicada pela biblioteca.

Podemos observar que nem todas as previsões condizem com o desempenho obtido. Isso pode ser justificado por um eventual excesso de simplificações ao se considerar sempre o caso médio ou pior caso. Redes com a mesma quantidade de variáveis podem ter topologias bastante diferentes, o que tem impacto direto nas operações com potenciais envolvidas. Quanto mais variáveis presentes em um mesmo potencial, mais custosa em termos de processamento e memória será a operação de multiplicação.

Para a indicação de uso de inferência distribuída em redes Credais houve acerto na previsão especialmente para os testes mais demorados. Tal fato é importante pois para testes rapidamente executados o tempo de execução não é percebido pelo usuário, enquanto nos testes mais demorados um bom desempenho é imprescindível.

Para o uso de inferência blocada o acerto também foi mais significativo para os tempos maiores. Entretanto, como vimos anteriormente, a fatoração blocada, quando aplicável, produz árvores de eliminação balanceadas e de menor profundidade o que é útil para a fatoração numérica, especialmente quando ela for executada muitas vezes, alterando apenas a evidência.

A escolha pela fatoração numérica paralela ou linear sequencialmente apresentou um índice maior de acerto. Alguns casos, como pudemos ver apresentam o mesmo tempo e a biblioteca em geral optou pela fatoração sequencial para economizar recursos computacionais. Ainda assim, para um maior acerto da previsão seriam necessárias mais informações sobre o problema, podemos ver isso claramente nos testes 9 e 10 da rede Hailfinder25 que apresentam os mesmos argumentos de entrada para a função (número de variáveis requisitadas, número de folhas na árvore de eliminação e número de ligações no grafo moral), mas possuem tempos de execução diferentes por conta dos potenciais envolvidos.

Pudemos ver também, em testes intermediários, que no caso de *PThreads* o *overhead* para o uso das mesmas é insignificante em comparação ao restante das operações e o que deve ser considerado mesmo é a quantidade de operações adicionais para o uso dos métodos paralelos, como a montagem da árvore de eliminação e de *threads* para fatoração numérica paralela, ou obtenção da forma blocada da matriz e criação dos subgrafos para a fatoração blocada.

Finalmente, pontuamos que o objetivo dessa biblioteca de previsão de desempenho é apenas dar uma direção da melhor opção e também da quantidade de processamento envolvido em cada método. Uma análise mais profunda seria capaz de gerar resultados mais aprimorados, porém optamos por fazer simplificações de caso médio e pior caso em vários momentos para evitar ter que buscar muitas informações em tempo de execução, fazendo com que o próprio uso da biblioteca já gerasse um grande consumo de recursos que acabaria com a vantagem de tempo de utilizar o método mais eficiente.

Rede	Teste	Tipo	Tempo	Previsão
Pigs	01	Serial.	0,06330	*
Pigs	01	Paralela.	0,50680	
Pigs	02	Serial.	0,07260	
Pigs	02	Paralela.	0,86530	*
Pigs	03	Serial.	0,06520	*
Pigs	03	Paralela.	0,50710	
Pigs	04	Serial.	0,06220	*
Pigs	04	Paralela.	0,40600	
Pigs	05	Serial.	0,06340	*
Pigs	05	Paralela.	0,80490	
Pigs	06	Serial.	0,06250	*
Pigs	06	Paralela.	0,60350	
Munin1	01	Serial.	5,13180	
Munin1	01	Paralela.	2,30860	*
Munin1	02	Serial.	0,15060	
Munin1	02	Paralela.	1,76580	*
Hailfinder25	01	Serial.	8,43440	
Hailfinder25	01	Paralela.	2,60790	*
Hailfinder25	02	Serial.	38,61600	
Hailfinder25	02	Paralela.	6,64340	*
Hailfinder25	03	Serial.	0,02530	*
Hailfinder25	03	Paralela.	1,70400	

Tabela 8.4: Resultados da previsão de desempenho para inferência em redes Credais

Rede	Teste	Tipo	Tempo	Previsão
Blocada01	01	Blocada	0,0133	
Blocada01	01	Serial	0,0000	*
Blocada01	02	Blocada	0,0300	
Blocada01	02	Serial	0,0000	*
Blocada02	01	Blocada	0,0900	*
Blocada02	01	Serial	0,0067	
Blocada02	02	Blocada	0,0200	*
Blocada02	02	Serial	0,0000	
Blocada03	01	Blocada	5,0533	
Blocada03	01	Serial	4,0900	*
Blocada03	02	Blocada	1,5967	*
Blocada03	02	Serial	1,8967	

Tabela 8.5: Resultados da previsão de desempenho para eliminação blocada

Rede	Teste	Tipo	Tempo	Previsão
Hailfinder25	03	Paralela	0,4167	
Hailfinder25	03	Serial	0,2433	*
Hailfinder25	04	Paralela	0,0500	
Hailfinder25	04	Serial	0,0400	*
Hailfinder25	05	Paralela	0,7833	*
Hailfinder25	05	Serial	1,0533	
Hailfinder25	07	Paralela	0,0433	
Hailfinder25	07	Serial	0,0400	*
Hailfinder25	08	Paralela	0,5333	
Hailfinder25	08	Serial	0,3567	*
Hailfinder25	09	Paralela	4,0800	*
Hailfinder25	09	Serial	4,4500	
Hailfinder25	10	Paralela	0,6400	*
Hailfinder25	10	Serial	0,7100	
Hailfinder25	11	Paralela	0,4567	*
Hailfinder25	11	Serial	0,8733	
Hailfinder25	12	Paralela	0,5867	*
Hailfinder25	12	Serial	0,8800	
Hailfinder25	13	Paralela	0,6833	*
Hailfinder25	13	Serial	0,7033	
Hailfinder25	14	Paralela	0,7200	*
Hailfinder25	14	Serial	0,7000	
Munin1	02	Paralela	0,0167	*
Munin1	02	Serial	0,0333	
Munin1	03	Paralela	0,0133	
Munin1	03	Serial	0,0133	*
Munin1	04	Paralela	0,0233	*
Munin1	04	Serial	0,0300	
Munin1	05	Paralela	0,0233	*
Munin1	05	Serial	0,0267	
Munin1	06	Paralela	0,0000	
Munin1	06	Serial	0,0000	*
Munin1	11	Paralela	0,0100	
Munin1	11	Serial	0,0100	*
Munin1	12	Paralela	0,6233	*
Munin1	12	Serial	0,6233	
Munin1	13	Paralela	0,0433	*
Munin1	13	Serial	0,0533	
Pigs	01	Paralela	0,0000	
Pigs	01	Serial	0,0033	*

Tabela 8.6: Resultados da previsão de desempenho para fatoração numérica paralela

Capítulo 9

Conclusões

O objetivo deste trabalho foi incrementar a biblioteca desenvolvida em Colla (2007) adicionando novos métodos para fatoração simbólica e inferência em redes Credais. Durante o processo foram implementadas também algumas pequenas otimizações e correções no código que eram mais facilmente identificáveis por quem não participou da elaboração da biblioteca original.

Reforçamos a separação entre a fase simbólica e a fase numérica do processo de inferência através da criação de arquivos XML com o resultado da fatoração simbólica e exportação de todas as informações envolvidas na inferência. Esta separação viabiliza uma independência real entre as fases permitindo, por exemplo, que cada uma delas seja feita em um computador diferente, sem a necessidade de compartilhamento de memória.

Com esta separação explícita entre as fases simbólica e numérica foi possível implementar a inferência paralela em redes Credais utilizando o método CCM de transformação de variáveis de maneira eficiente, realizando a fase simbólica uma única vez no processo principal e utilizando o MPI na fase numérica de cada rede gerada, minimizando a troca de mensagens e evitando refazer cálculos desnecessários. Utilizar processamento distribuído para a inferência em redes Credais é apropriado pois, para problemas maiores, conforme pudemos observar nos testes, o processamento da fase numérica supera o custo computacional da troca de mensagens. Além disso, como o método apenas altera o estado das evidências geradas pelas variáveis transparentes, o tamanho e topologia das redes são iguais, bem como os potenciais envolvidos em cada operação, o que gera um bom balanceamento de carga, em particular para processadores com as mesmas configurações.

A biblioteca de análise de desempenho ainda está em sua versão inicial mas já apresenta boas direções sobre a necessidade de paralelizar ou não uma inferência, afinal, como pudemos observar nos resultados dos testes, para os casos mais demorados a incidência de acertos foi mais alta. Como o custo de cada método está diretamente relacionado com o algoritmo, seria necessária uma análise mais detalhada das funções, tornando mais variáveis como parâmetros para obter um resultado ainda mais preciso. Utilizando a previsão de desempenho é possível otimizar o tempo computacional e evitar a alocação desnecessária de recursos.

A fatoração simbólica blocada utiliza uma característica particular da matriz de adjacência correspondente ao grafo da rede, que é ser composta por blocos. Apesar desta versão assumir blocos de tamanhos iguais, com poucas modificações no código é possível torná-lo mais abrangente para blocos de tamanhos distintos, o que ocorre com mais frequência em problemas reais. Sua implementação envolveu conceitos de álgebra linear computacional e também de otimização numérica heurística. A principal vantagem da fatoração simbólica blocada é a possibilidade de paralelizar a fase simbólica da eliminação de variáveis, sendo uma boa opção quando houver disponibilidade de mais de um processador com memória compartilhada. Além disso, ela pode ser utilizada como um método alternativo de obtenção da ordem de eliminação. Finalmente, os testes demonstraram que este método gera árvores de eliminação balanceadas, o que é adequado para o processamento paralelo da fase numérica.

Conforme os testes de precisão e performance mostraram, esta segunda versão da biblioteca implementou métodos paralelos eficientes para inferência em redes Bayesianas e credais.

9.1 Sugestões para Pesquisas Futuras

Para trabalhos futuros, além das pequenas melhorias mencionadas ao longo do texto, sugerimos inicialmente uma refatoração das Bibliotecas. Apesar da versão atual já estar separada em módulos independentes, é possível uma separação ainda maior com algumas modificações nas estruturas de dados implementadas. Uma linguagem com orientação a objetos, por exemplo, permitiria criar classes que não necessitam do índice para relacionar uma variável com sua tabela de potenciais. Outra tarefa para a refatoração é a modificação da árvore de threads para que ela possa receber qualquer tipo de operação com potenciais e que seja possível trocar o mecanismo de operações dos potenciais por outras formas de cálculo credal abstrato aumentando bastante a abrangência das bibliotecas.

Após a refatoração sugerimos o estudo e implementação de mais técnicas de inferência para redes Credais, em particular de técnicas de inferência aproximada que permitam trabalhar com redes maiores e também daquelas que tiram vantagem de características específicas da rede, como por exemplo redes com variáveis binárias. É possível também estender a biblioteca para que ela trabalhe com outros tipos de redes como redes intervalares e qualitativas. Estas redes executam a mesma estrutura de fatoração já implementada, sendo necessárias poucas modificações para o seu processamento uma vez que o mecanismo de operações com potenciais já está suficientemente genérico.

Finalmente, sugerimos o desenvolvimento da aplicação principal, também com código aberto, para que haja uma organização das bibliotecas e seja mais fácil sua difusão, permitindo que elas sejam utilizadas amplamente no meio acadêmico.

Apêndice A

Teoria dos Grafos

Este Apêndice tem o objetivo de apresentar os principais conceitos de Teoria dos Grafos para um embasamento teórico dos assuntos cobertos nesta dissertação. Um tratamento teórico mais abrangente sobre grafos pode ser encontrado em Stern (1994), Bondy e Murty (2007) e também em Bang-Jensen e Gutin (2009).

Um grafo direcionado (ou digrafo) consiste em um par ordenado $G = (V_G, A_G)$ em que V_G é um conjunto finito não-vazio de elementos chamados **vértices** (ou **nós**) e A_G é um conjunto finito de **arestas** (ou **arcos**) juntamente com Γ_G, uma função chamada **função de filiação** que associa cada aresta de G a um par ordenado de vértices (i, j) não necessariamente distintos que são ditos **incidentes** à aresta (e vice-versa). A inversa da função de filiação, Γ_G^{-1} é chamada **função de paternidade**. Vértices conectados por uma aresta são chamados **adjacentes** ou **vizinhos**; O conjunto de vizinhos ou **vizinhança** $\mathcal{N}(v)$ de um vértice v consiste de todos os vértices adjacentes a ele, a sua **vizinhança fechada** $\mathcal{N}[v]$ é sua vizinhança $\mathcal{N}(v)$ mais ele próprio. Graficamente é comum representar os vértices de um grafo como pontos e suas arestas como setas. Uma outra forma de representação de um grafo é através da matriz adjacência que consiste em uma matriz B, $n \times n$, usualmente booleana, onde n é o número de vértices do grafo e é preenchida da seguinte forma: se existir uma aresta do vértice i ao vértice j, então o elemento B_i^j de B é não nulo, em geral 1, caso contrário é zero.

A **ordem** de um grafo G é seu número de vértices, $|V_G|$ e seu **tamanho** é o número de arestas que possui, $|A_G|$. O **grau** de um vértice é igual à soma dos graus de saída (o número de arestas saindo do vértice) e de entrada (o número de arestas entrando no vértice).

Um **laço** ou **loop** é uma aresta que parte e chega no mesmo vértice. Um **passeio** é uma sequência alternante de vértices e arestas, começando e terminando em um vértice, onde cada vértice é incidente à aresta que o precede e o sucede na sequência. Um passeio é **fechado** ou também chamado de **circuito** se o primeiro e último vértices do passeio forem o mesmo. Uma **trilha** é um passeio em que todas as arestas são distintas. Um **caminho** é uma trilha na qual não há circuitos. Um **ciclo** é uma trilha na qual o único circuito é toda a trilha.

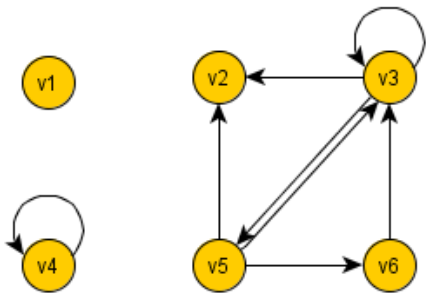


Figura A.1: Exemplo de um grafo

A Figura A.1 apresenta um exemplo ilustrando as definições anteriores: $G = (V_G, A_G)$ é um

grafo em que $V_G = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ e $A_G = \{(3,2),(3,3),(3,5),(4,4),(5,2),(5,3),(5,6),(6,3)\}$. Temos também a função de filiação: $\Gamma(v_1) = \{\}$, $\Gamma(v_2) = \{\}$, $\Gamma(v_3) = \{v_2, v_3, v_5\}$, $\Gamma(v_4) = \{v_4\}$, $\Gamma(v_5) = \{v_2, v_3, v_6\}$, $\Gamma(v_6) = \{v_3\}$ e sua representação pela matriz de adjacência:

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

A ordem de G é 6 e seu tamanho é 8. O grau do vértice 3, v_3, é 5. Podemos ver ainda que (v_4, v_4) é um loop, $\{(v_3, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_5)\}$ é um passeio, $\{(v_3, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_5), (v_5, v_3)\}$ é um circuito, $\{(v_3, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_3), (v_3, v_2)\}$ é uma trilha, $\{(v_5, v_6), (v_6, v_3), (v_3, v_2)\}$ é um caminho e $\{(v_3, v_5), (v_5, v_6), (v_6, v_3)\}$ é um ciclo.

Se existe uma aresta que vai do vértice i ao vértice j em G, dizemos que i faz parte conjunto dos **pais** do vértice j, $i \in pa(j)$, ou, equivalentemente, que j pertence ao conjunto dos **filhos** do vértice i, $j \in ch(i)$. Se os vértices i e j possuem pelo menos um filho em comum, dizemos que j pertence ao conjunto de **esposos** do vértice i, $j \in sp(i)$ (e vice-versa). Se existe um caminho de i até j dizemos que um vértice i é **ancestral** de um vértice j e que j é **descendente** de i. Para representar o conjuntos de descendentes de um vértice definimos a função de descendência $\bar{\Gamma}_G$.

Um grafo direcionado é **acíclico** (GDA) se não contém ciclos. Uma **árvore** $H = (V_H, A_H)$ de **raiz** $v \in V_H$ é um GDA, no qual todos os vértices tem no máximo um pai e apenas a raiz não tem pai. As **folhas** são os vértices sem filhos de uma árvore. Uma **floresta** é grafo composto por um conjunto de árvores.

Um grafo não direcionado (ou não orientado) $G = (V_G, A_G)$ difere-se de um grafo direcionado pelo fato que os vértices associados pelas arestas não possuem ordem, o que faz com que sua matriz de adjacência B seja sempre simétrica. Um grafo não orientado é **conexo** se existe um caminho entre quaisquer dois de seus vértices. Um grafo **completo** é um grafo não direcionado acíclico que para cada vértice existe uma aresta conectando este vértice a cada um dos demais. Um **subgrafo** de um grafo G é um grafo cujos vértices são um subconjunto de V_G e o conjunto de arestas é um subconjunto do conjunto de A_G. Um **clique** em um grafo G é um subgrafo de G que é completo. Um **separador** é um conjunto de vértices que se removidos, separam o grafo em conjuntos conexos distintos.

O **grafo moral** de um GDA G é o grafo não direcionado criado a partir de G, substituindo suas arestas por arestas não direcionadas e adicionando uma aresta (não direcionada, denominada **arco moral**) entre todos os pais de um mesmo vértice (este processo é chamado **casamento**). O **Markov Blanket** de um vértice i em um grafo moral é o conjunto de seus pais, filhos e esposos.

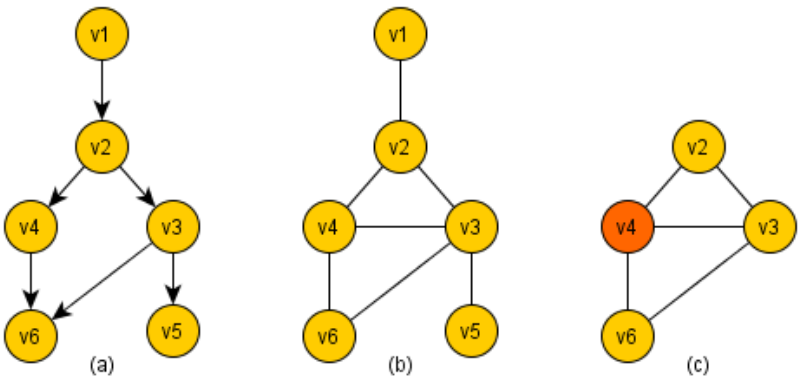


Figura A.2: Exemplo de um GDA, seu grafo Moral e Markov Blanket de um vértice

A Figura A.2 apresenta um exemplo de um GDA (a), seu grafo moral (b) e o *Markov Blanket* do vértice v_4 (c). Algumas das relações que podemos observar em (a) são: v_2 é filho de v_1; v_3 e v_4 são filhos de v_2; v_3 e v_4 são esposos e são pais de v_6; v_3, v_4, v_5, v_6 são descendentes de v_2; v_5 e v_3 não são descendentes de v_4.

Uma ordem em um GDA é uma relação $\prec$, reflexiva e transitiva, e a ordem natural de um grafo G é definida pela função de descendência $\bar{\Gamma}_G$ que implica que se para os vértices i e j vale $i \prec j$ então j é descendente de i ($j \in \bar{\Gamma}_G(i)$). O vetor de índices $N = [1, 2, \ldots, n]$ representa essa ordem natural e o vetor $q = [\sigma(1), \sigma(2), \ldots, \sigma(n)]$ é uma permutação da ordem natural, podendo ser representada simplesmente por $q = [(1), (2), \ldots, (n)]$ quando não houver risco de confusão.

Dada uma **ordem de eliminação** q e um grafo $G = (V, A)$, dizemos que $q(k) = i$ significa que o vértice i foi o k-ésimo vértice de G a ser eliminado e definimos os **grafos de eliminação** $G_k = (V_k, A_k)$ como o grafo obtido removendo de G o vértice $q(k)$ e suas arestas incidentes, e adicionando arestas chamadas **arcos de preenchimento** unindo seus vizinhos transformando-os em um clique. Caso a eliminação dos vértices não implique na adição de nenhum arco de preenchimento dizemos que ela é uma **eliminação perfeita**.

O grafo $P = (V, F)$ obtido através do processo de eliminação onde $F = \cup_{k=1}^{n} A_k$ é chamado **Grafo Preenchido** e seus arcos de preenchimento são a diferença $F - A$. Por questões computacionais definimos também a **eliminação simplificada** que difere do processo de eliminação definido anteriormente quando ao eliminar o vértice $q(k)$ são preenchidos apenas os arcos incidentes ao seu vizinho mais próximo de ser eliminado.

A Figura A.3 mostra em (a) um grafo não direcionado em de (b) a (e) os passos do seu processo de eliminação em (f) seu grafo preenchido, com os arcos adicionados em destaque. Em cada passo os arcos adicionados na etapa aparecem em destaque bem como o próximo vértice a ser eliminado. A ordem de eliminação para o exemplo é $q = [1, 3, 6, 2, 4, 5]$. A Figura A.4 apresenta os passos do processo de eliminação simplificada para o grafo anterior, mantendo a ordem de eliminação. Adicionalmente, o nó a ser eliminado após o próximo nó a ser eliminado também é destacado, em cor mais clara.

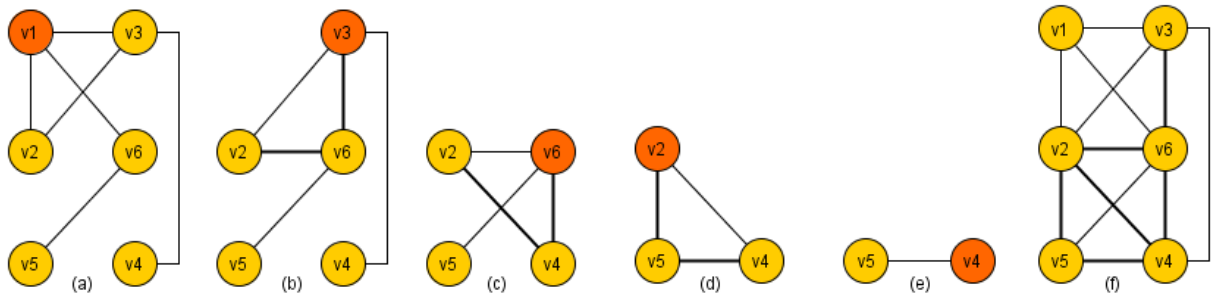


Figura A.3: Processo de eliminação em um grafo não direcionado e grafo preenchido

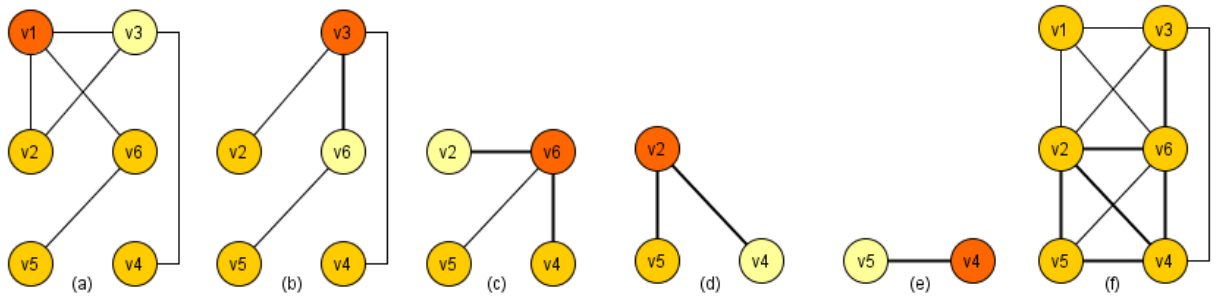


Figura A.4: Processo de eliminação simplificada em um grafo não direcionado e grafo preenchido

Apêndice B

Teoria das Probabilidades

Este apêndice tem o objetivo de apresentar os principais conceitos de Teoria das Probabilidades para continuidade do embasamento teórico dos assuntos cobertos nesta dissertação. Para maiores detalhes sobre o assunto recomendamos a leitura de Ross (2003).

Suponha que temos um experimento cujo resultado é desconhecido, porém sabemos qual é o conjunto Ω de todos os resultados possíveis. Esse conjunto é chamado de **espaço amostral**. Um subconjunto $A \subseteq \Omega$ é chamado de **evento**. Cada um dos elementos $w \in \Omega$ caracteriza um dos estados possíveis de Ω. Associado à esses conjuntos temos $P(A)$, uma função que indica a probabilidade de um evento $A \subseteq \Omega$ ocorrer satisfazendo as seguintes propriedades:

1. $0 \leq P(A) \leq 1$;
2. $P(\Omega) = 1$;
3. Se $A_1, A_2, \ldots, A_n \subseteq \Omega$ são tais que $A_i \cap A_j$, para $i \neq j$ (são mutuamente exclusivos) então $P(A_1 \cup A_2 \cup \cdots A_n) = \sum_{i=1}^{n} P(A_i)$.

Podemos ver ainda algumas consequências dessas propriedades:

- $P(A) \leq P(B) \quad \text{se} \quad A \subseteq B$;
- $P(\emptyset) = 0$;
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- $P(A^c) = 1 - P(A)$

Probabilidade condicional

A probabilidade de A condicionada por B (ou dado B, ou sabendo que B), dado $P(B) > 0$ é definida por:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{B.1}$$

Escrevendo de outra forma, temos a regra fundamental da probabilidade:

$$P(A \mid B)P(B) = P(A \cap B) \tag{B.2}$$

Em ambos os casos podemos ver que o fato de ocorrer B influencia a probabilidade de A. Para calcular $A \cap B$ devemos calcular os eventos que são comuns a B e a A. $P(A \cap B)$ (ou $P(A, B)$) é chamada de **probabilidade conjunta** de A e B. Se $P(A \cap B) = P(A) \cdot P(B)$ dizemos que A e B são **independentes**. Isto significa que $P(A \mid B) = \frac{P(A) \cdot P(B)}{P(B)} = P(A)$, ou seja, que a ocorrência de B não tem efeito sobre a probabilidade de A acontecer.

Regra de Bayes

A regra de Bayes relaciona as probabilidade de A e B com as respectivas probabilidades condicionais. Ela é a base de toda estatística Bayesiana sendo, portanto, muito importante. Ela afirma que:

$$P(B \mid A) = P(A \mid B) \cdot \frac{P(B)}{P(A)} \tag{B.3}$$

$P(B \mid A)$, a probabilidade do evento B ocorrer dado que A ocorreu, é chamada de probabilidade posterior ou posteriori, $P(B)$ é a probabilidade a priori do evento E ocorrer, calculada previamente antes da ocorrência de A e $P(A)$ é a probabilidade marginal, ou seja, a probabilidade de que ocorra A.

Variáveis Aleatórias

Uma **variável aleatória** (ou simplesmente variável) X, é uma função $X : \Omega \to \mathbb{R}$ que associa cada elemento do espaço amostral a um valor real (Bussab e Morettin, 2005).

$P(X = A)$ ou $p(A)$ é a **distribuição de probabilidade** da variável X. Ela é uma função cujo domínio são os valores da variável e cuja imagem são as probabilidades de a variável assumir cada valor do domínio. O conjunto imagem deste tipo de função está sempre restrito ao intervalo entre 0 e 1, valendo sempre 0 no caso de um conjunto vazio (evento impossível) e 1 para todo o espaço amostral (evento certo).

Uma distribuição de probabilidade pode ser discreta (finitos valores pontuais), que estamos particularmente interessados ou contínua (assume valores em toda reta real). Uma **distribuição de probabilidade discreta** descreve o comportamento de uma variável aleatória quantitativa discreta, ou seja, aquela que assume uma quantidade finita ou infinita enumerável de valores e representa a probabilidade da variável aleatória X assumir um certo valor x: $P(X = x)$ sendo que $\sum_u P(X = u) = 1$ e u representa o conjunto de todos os valores possíveis de x.

A **função de distribuição acumulada** (ou simplesmente função de distribuição) de uma variável aleatória discreta representa a probabilidade de uma variável aleatória X assumir valores menores ou iguais a x, ou seja:

$$F(x) = P(X \leq x)$$

Marginalização

Se X e Y são duas variáveis aleatórias discretas, é possível representar sua função de probabilidade conjunta, $p(x, y) = P[X = x, Y = y]$ através de uma tabela onde as linhas indicam todos os x_i estados possíveis de X, as colunas indicam os y_j estados possíveis de Y e, portanto, a entrada correspondente à posição ij representa $P(X = x_i, Y = y_j)$.

A distribuição de probabilidade de X, chamada de **distribuição marginal** de X pode ser obtida de $p(x, y)$ por:

$$p_X(x) = P[X = x] = \sum_{y:p(x,y)>0} p(x, y)$$

Analogamente:

$$p_Y(y) = P[Y = y] = \sum_{x:p(x,y)>0} p(x, y)$$

Este processo de obter a distribuição marginal de uma variável é chamado de **marginalização**. No primeiro caso dizemos que Y foi marginalizada para fora da distribuição conjunta $p(x, y)$ e no segundo caso dizemos que X foi marginalizada para fora.

Exemplo

O Exemplo a seguir foi baseado no exemplo de Ross (2009) e ilustra uma situação de uso de tabelas para distribuições conjuntas e condicionais. Suponha que 15% das famílias de uma certa

comunidade não tenham filhos, 20% tenham 1, 35% tenham 2 e 30% tenham 3 filhos e suponha ainda que em cada família cada criança possa ser menino ou menina com a mesma probabilidade. Se uma família é escolhida aleatoriamente na comunidade, X é o número de filhas e Y é o número de filhos dessa família então temos sua distribuição conjunta dada pela tabela B.1

i j	0	1	2	3	$P_Y(y)$
0	0,15	0,10	0,0875	0,0375	0,3750
1	0,10	0,175	0,1125	0	0,3875
2	0,0875	0,1125	0	0	0,2000
3	0,0375	0	0	0	0,0375
$P_X(y)$	0,3750	0,3875	0,2000	0,0375	1

Tabela B.1: Distribuição conjunta de filhas e filhos de uma família

As probabilidades foram calculadas da seguinte forma:

$$
\begin{aligned}
P[X=0,Y=0] &= P[\text{sem crianças}] = 0,15 \\
P[X=1,Y=0] &= P[\text{1 menina e total de 1 criança}] \\
&= P[\text{1 criança}]P[\text{1 menina} \mid \text{1 criança}] = (0,20)\left(\frac{1}{2}\right) \\
P[X=2,Y=0] &= P[\text{2 meninas e total de 2 crianças}] \\
&= P[\text{2 crianças}]P[\text{2 meninas} \mid \text{2 crianças}] = (0,35)\left(\frac{1}{2}\right)^2
\end{aligned}
$$

Sendo que é fácil estender o raciocínio para as entradas restantes da tabela. A última linha representa a probabilidade marginal de X (número de meninas) e é obtida através da soma de cada coluna. Similarmente, a última coluna representa a probabilidade marginal de Y (número de meninos) e é obtida através da soma de cada linha. Através da tabela com a distribuição conjunta e as distribuições marginais podemos calcular a distribuição condicional $P(X|Y)$ como mostramos na tabela B.2. Podemos observar que neste caso a soma de cada coluna é igual a 1.

i j	0	1	2	3
0	0,40	0,26	0,44	1
1	0,27	0,45	0,56	0
2	0,23	0,29	0	0
3	0,10	0	0	0

Tabela B.2: Distribuição condicionais de filhas dados os filhos de uma família

B.1 Potenciais

Um **potencial** é uma tabela de valores reais sobre um **domínio** finito de variáveis e é utilizado para manipular tabelas de probabilidades. Nesse contexto $dom(\phi)$ representa o domínio do potencial ϕ ((Jensen, 2001)). Dois potenciais podem ser **multiplicados**, com as seguintes propriedades:

1. $dom(\phi_1\phi_2) = dom(\phi_1) \cup dom(\phi_2)$;
2. *Propriedade Comutativa*: $\phi_1\phi_2 = \phi_2\phi_1$;
3. *Propriedade Associativa*: $(\phi_1\phi_2)\phi_3 = \phi_1(\phi_2\phi_3)$;

4. *Existência da Unidade*: O número 1 é um potencial sobre todo conjunto vazio e $1 \cdot \phi = \phi$ para todos potenciais ϕ. O potencial unitário será denotado por $\mathbf{1}$;

   Um potencial pode ser marginalizado. $\sum_A \phi$ é um potencial sobre $dom(\phi) \setminus A$, ou seja, A foi marginalizada para fora. Para potenciais da forma $P(A|V)$, onde V é um conjunto de variáveis temos:

5. *Potencial unitário*: $\sum_A P(A|V) = 1$;

6. *Propriedade comutativa da marginalização*: $\sum_A \sum_B \phi = \sum_B \sum_A \phi$;

7. *Propriedade distributiva da marginalização*: Se $A \notin dom(\phi_1)$, então $\sum_A \phi_1 \phi_2 = \phi_1 \sum_A \phi_2$;

B.1.1 Exemplos

As Tabelas B.3-B.7 apresentam os potenciais $\phi_1(A, B)$ e $\phi_2(B, C)$ e ilustram algumas operações e o processo de marginalização para fora de A e B, chamado também de **projeção** para C: As variáveis A, B e C possuem dois estados mutuamente exclusivos cada, (a_1, a_2), (b_1, b_2), (c_1, c_2), respectivamente e seus valores são apresentados na Tabela B.3. A Tabela B.4 apresenta o produto dos potenciais $\phi(A, B) \cdot \phi_2(B, C)$ sendo que cada entrada da tabela apresenta os estados c_1 e c_2 de C, respectivamente. É possível verificar a propriedade comutativa dos potenciais, uma vez que vale a comutação para produto de números reais. A Tabela B.5 apresenta a marginalização da variável C, através da soma das entradas da tabela com o potencial dos produtos. A Tabela B.6 utiliza o mesmo procedimento para marginalizar C de $\phi_2(B, C)$. Finalmente, a Tabela B.7 é obtida através do produto de $\phi_1(A, B)$ pela marginalização de C de $\phi_2(B, C)$. Podemos verificar a propriedade distributiva da multiplicação de potenciais, uma vez que as Tabelas B.5 e B.7 são equivalentes, dado que vale a propriedade distributiva para números reais.

$B \setminus A$	a_1	a_2
b_1	x_1	x_2
b_2	x_3	x_4

$B \setminus C$	c_1	c_2
b_1	y_1	y_2
b_2	y_3	y_4

Tabela B.3: Potenciais $\phi_1(A, B)$ e $\phi_2(B, C)$

$B \setminus A$	a_1	a_2
b_1	(x_1y_1, x_1y_2)	(x_2y_1, x_2y_2)
b_2	(x_3y_3, x_3y_4)	(x_4y_3, x_4y_4)

Tabela B.4: Produto $\phi_1(A, B) \cdot \phi_2(B, C)$

$B \setminus A$	a_1	a_2
b_1	$x_1y_1 + x_1y_2$	$x_2y_1 + x_2y_2$
b_2	$x_3y_3 + x_3y_4$	$x_4y_3 + x_4y_4$

Tabela B.5: $\sum_C \phi_1(A, B) \cdot \phi_2(B, C)$

B	
b_1	$y_1 + y_2$
b_2	$y_3 + y_4$

Tabela B.6: $\sum_C \phi_2(B, C)$

$B \setminus A$	a_1	a_2
b_1	$x_1(y_1 + y_2)$	$x_2(y_1 y_2)$
b_2	$x_3(y_3 + y_4)$	$x_4(y_3 + y_4)$

Tabela B.7: $\phi_1(A, B) \sum_C \cdot \phi_2(B, C)$

Apêndice C

Redes dos Exemplos

C.1 Rede Credal utilizada para o exemplo de inferência

C.1.1 Rede com limites inferiores

```
<?xml version="1.0" encoding="US-ASCII"?>

<!-- DTD for the XMLBIF 0.3 format -->
<!DOCTYPE BIF [
   <!ELEMENT BIF ( NETWORK )*>
      <!ATTLIST BIF VERSION CDATA #REQUIRED>
   <!ELEMENT NETWORK ( NAME, ( PROPERTY | VARIABLE | PROBABILITY )* )>
   <!ELEMENT NAME (#PCDATA)>
   <!ELEMENT VARIABLE ( NAME, ( VALUE |  PROPERTY )* ) >
      <!ATTLIST VARIABLE TYPE (nature|decision|utility) "nature">
   <!ELEMENT VALUE (#PCDATA)>
   <!ELEMENT PROBABILITY ( FOR | GIVEN | TABLE | PROPERTY )* >
   <!ELEMENT FOR (#PCDATA)>
   <!ELEMENT GIVEN (#PCDATA)>
   <!ELEMENT TABLE (#PCDATA)>
   <!ELEMENT PROPERTY (#PCDATA)>
]>


<BIF>
<NETWORK>
<NAME>Credal</NAME>

<!-- Variables -->
<VARIABLE>
   <NAME>A</NAME>
   <TYPE>DISCRETE</TYPE>
   <VALUE>A1</VALUE>
   <VALUE>A2</VALUE>
   <VALUE>A3</VALUE>
</VARIABLE>

<VARIABLE>
   <TYPE>DISCRETE</TYPE>
   <NAME>B</NAME>
```

```
    <VALUE>B1</VALUE>
    <VALUE>B2</VALUE>
    <VALUE>B3</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>C</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>C1</VALUE>
    <VALUE>C2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>D</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>D1</VALUE>
    <VALUE>D2</VALUE>
    <VALUE>D3</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>E</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>E1</VALUE>
    <VALUE>E2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>F</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>F1</VALUE>
    <VALUE>F2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>G</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>G1</VALUE>
    <VALUE>G2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>H</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>H1</VALUE>
    <VALUE>H2</VALUE>
    <VALUE>H3</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>I</NAME>
    <TYPE>DISCRETE</TYPE>
```

```
  <VALUE>I1</VALUE>
  <VALUE>I2</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>J</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>J1</VALUE>
  <VALUE>J2</VALUE>
  <VALUE>J3</VALUE>
</VARIABLE>

<!-- Probability distributions -->
<PROBABILITY>
  <FOR>A</FOR>
  <TABLE>0.58 0.08 0.27 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>B</FOR>
  <GIVEN>A</GIVEN>
  <TABLE>0.34 0.48 0.12 0.03 0.38 0.53 0.47 0.46 0.01 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>C</FOR>
  <GIVEN>B</GIVEN>
  <TABLE>0.08 0.92 0.51 0.49 0.86 0.14 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>D</FOR>
  <GIVEN>A</GIVEN>
  <GIVEN>C</GIVEN>
  <TABLE>0.01 0.47 0.52 0.23 0.05 0.72 0.51 0.23 0.26 0.26 0.32 0.42
     0.4 0.46 0.14 0.27 0.38 0.35 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>E</FOR>
  <GIVEN>D</GIVEN>
  <TABLE>0.38 0.62 0.92 0.08 0.12 0.88 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>F</FOR>
  <GIVEN>B</GIVEN>
  <TABLE>0.79 0.21 0.43 0.57 0.45 0.55 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>G</FOR>
```

```
  <GIVEN>F</GIVEN>
  <GIVEN>H</GIVEN>
  <TABLE>0.03 0.97 0.39 0.61 0.48 0.52 0.96 0.04 0.51 0.49 0.6 0.4 </
      TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>H</FOR>
  <TABLE>0.5 0.03 0.47 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>I</FOR>
  <GIVEN>E</GIVEN>
  <TABLE>0.42 0.58 0.79 0.21 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>J</FOR>
  <GIVEN>D</GIVEN>
  <GIVEN>G</GIVEN>
  <TABLE>0.43 0.32 0.25 0.21 0.53 0.26 0.07 0.39 0.54 0.14 0.68 0.18
      0.33 0.4 0.27 0.35 0.56 0.09 </TABLE>
</PROBABILITY>

</NETWORK>
</BIF>
```

C.1.2 Rede com limites superiores

```
<?xml version="1.0" encoding="US-ASCII"?>

<!-- DTD for the XMLBIF 0.3 format -->
<!DOCTYPE BIF [
  <!ELEMENT BIF ( NETWORK )*>
    <!ATTLIST BIF VERSION CDATA #REQUIRED>
  <!ELEMENT NETWORK ( NAME, ( PROPERTY | VARIABLE | PROBABILITY )* )>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT VARIABLE ( NAME, ( VALUE |  PROPERTY )* ) >
    <!ATTLIST VARIABLE TYPE (nature|decision|utility) "nature">
  <!ELEMENT VALUE (#PCDATA)>
  <!ELEMENT PROBABILITY ( FOR | GIVEN | TABLE | PROPERTY )* >
  <!ELEMENT FOR (#PCDATA)>
  <!ELEMENT GIVEN (#PCDATA)>
  <!ELEMENT TABLE (#PCDATA)>
  <!ELEMENT PROPERTY (#PCDATA)>
]>


<BIF>
<NETWORK>
<NAME>Credal</NAME>
```

```
<!-- Variables -->
<VARIABLE>
  <NAME>A</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>A1</VALUE>
  <VALUE>A2</VALUE>
  <VALUE>A3</VALUE>
</VARIABLE>

<VARIABLE>
  <TYPE>DISCRETE</TYPE>
  <NAME>B</NAME>
  <VALUE>B1</VALUE>
  <VALUE>B2</VALUE>
  <VALUE>B3</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>C</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>C1</VALUE>
  <VALUE>C2</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>D</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>D1</VALUE>
  <VALUE>D2</VALUE>
  <VALUE>D3</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>E</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>E1</VALUE>
  <VALUE>E2</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>F</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>F1</VALUE>
  <VALUE>F2</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>G</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>G1</VALUE>
  <VALUE>G2</VALUE>
</VARIABLE>
```

```
<VARIABLE>
  <NAME>H</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>H1</VALUE>
  <VALUE>H2</VALUE>
  <VALUE>H3</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>I</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>I1</VALUE>
  <VALUE>I2</VALUE>
</VARIABLE>

<VARIABLE>
  <NAME>J</NAME>
  <TYPE>DISCRETE</TYPE>
  <VALUE>J1</VALUE>
  <VALUE>J2</VALUE>
  <VALUE>J3</VALUE>
</VARIABLE>

<!-- Probability distributions -->
<PROBABILITY>
  <FOR>A</FOR>
  <TABLE>0.62 0.1 0.35 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>B</FOR>
  <GIVEN>A</GIVEN>
  <TABLE>0.36 0.52 0.17 0.05 0.42 0.59 0.51 0.50 0.07 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>C</FOR>
  <GIVEN>B</GIVEN>
  <TABLE>0.08 0.92 0.51 0.49 0.86 0.14 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>D</FOR>
  <GIVEN>A</GIVEN>
  <GIVEN>C</GIVEN>
  <TABLE>0.01 0.47 0.52 0.23 0.05 0.72 0.51 0.23 0.26 0.26 0.32 0.42
      0.4 0.46 0.14 0.27 0.38 0.35 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>E</FOR>
```

```
  <GIVEN>D</GIVEN>
  <TABLE>0.38 0.62 0.92 0.08 0.12 0.88 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>F</FOR>
  <GIVEN>B</GIVEN>
  <TABLE>0.79 0.21 0.43 0.57 0.45 0.55 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>G</FOR>
  <GIVEN>F</GIVEN>
  <GIVEN>H</GIVEN>
  <TABLE>0.03 0.97 0.39 0.61 0.48 0.52 0.96 0.04 0.51 0.49 0.6 0.4 </
      TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>H</FOR>
  <TABLE>0.5 0.03 0.47 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>I</FOR>
  <GIVEN>E</GIVEN>
  <TABLE>0.42 0.58 0.79 0.21 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>J</FOR>
  <GIVEN>D</GIVEN>
  <GIVEN>G</GIVEN>
  <TABLE>0.43 0.32 0.25 0.21 0.53 0.26 0.07 0.39 0.54 0.14 0.68 0.18
      0.33 0.4 0.27 0.35 0.56 0.09 </TABLE>
</PROBABILITY>

</NETWORK>
</BIF>
```

C.1.3 Consulta

```
<query>

        <variable>I</variable>

</query>
```

C.1.4 Evidência

```
<evidences>
```

```
        <variable>
                <name>H</name>
                <value>H2</value>
        </variable>


        <variable>
                <name>J</name>
                <value>J3</value>
        </variable>


</evidences>
```

C.2 Rede Bayesiana utilizada para o exemplo de fatoração blocada

C.2.1 Rede

```
<?xml version="1.0" encoding="US-ASCII"?>

<!-- DTD for the XMLBIF 0.3 format -->
<!DOCTYPE BIF [
  <!ELEMENT BIF ( NETWORK )*>
    <!ATTLIST BIF VERSION CDATA #REQUIRED>
  <!ELEMENT NETWORK ( NAME, ( PROPERTY | VARIABLE | PROBABILITY )* )>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT VARIABLE ( NAME, ( VALUE |  PROPERTY )* ) >
    <!ATTLIST VARIABLE TYPE (nature|decision|utility) "nature">
  <!ELEMENT VALUE (#PCDATA)>
  <!ELEMENT PROBABILITY ( FOR | GIVEN | TABLE | PROPERTY )* >
  <!ELEMENT FOR (#PCDATA)>
  <!ELEMENT GIVEN (#PCDATA)>
  <!ELEMENT TABLE (#PCDATA)>
  <!ELEMENT PROPERTY (#PCDATA)>
]>


<BIF>
<NETWORK>
<NAME>Blocada</NAME>

<!-- Variables -->
<VARIABLE>
  <NAME>A</NAME>
  <TYPE>DISCRETE</TYPE>
```

```
    <VALUE>A1</VALUE>
    <VALUE>A2</VALUE>
</VARIABLE>

<VARIABLE>
    <TYPE>DISCRETE</TYPE>
    <NAME>B</NAME>
    <VALUE>B1</VALUE>
    <VALUE>B2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>C</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>C1</VALUE>
    <VALUE>C2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>D</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>D1</VALUE>
    <VALUE>D2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>E</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>E1</VALUE>
    <VALUE>E2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>F</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>F1</VALUE>
    <VALUE>F2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>G</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>G1</VALUE>
    <VALUE>G2</VALUE>
</VARIABLE>

<VARIABLE>
    <NAME>H</NAME>
    <TYPE>DISCRETE</TYPE>
    <VALUE>H1</VALUE>
    <VALUE>H2</VALUE>
</VARIABLE>
```

```
<VARIABLE>
   <NAME>I</NAME>
   <TYPE>DISCRETE</TYPE>
   <VALUE>I1</VALUE>
   <VALUE>I2</VALUE>
</VARIABLE>

<VARIABLE>
   <NAME>J</NAME>
   <TYPE>DISCRETE</TYPE>
   <VALUE>J1</VALUE>
   <VALUE>J2</VALUE>
</VARIABLE>


<VARIABLE>
   <NAME>K</NAME>
   <TYPE>DISCRETE</TYPE>
   <VALUE>K1</VALUE>
   <VALUE>K2</VALUE>
</VARIABLE>

<VARIABLE>
   <NAME>L</NAME>
   <TYPE>DISCRETE</TYPE>
   <VALUE>L1</VALUE>
   <VALUE>L2</VALUE>
</VARIABLE>

<VARIABLE>
   <NAME>M</NAME>
   <TYPE>DISCRETE</TYPE>
   <VALUE>M1</VALUE>
   <VALUE>M2</VALUE>
</VARIABLE>

<!-- Probability distributions -->
<PROBABILITY>
   <FOR>F</FOR>
   <GIVEN>D</GIVEN>
   <GIVEN>I</GIVEN>
   <TABLE>0.48 0.52 0.28 0.72 0.51 0.49 0.38 0.62 </TABLE>
</PROBABILITY>

<PROBABILITY>
   <FOR>D</FOR>
   <TABLE>0.08 0.92 </TABLE>
</PROBABILITY>

<PROBABILITY>
   <FOR>I</FOR>
```

```
  <GIVEN>A</GIVEN>
  <TABLE>0.08 0.92 0.86 0.14 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>J</FOR>
  <GIVEN>M</GIVEN>
  <GIVEN>E</GIVEN>
  <TABLE>0.48 0.52 0.28 0.72 0.51 0.49 0.38 0.62 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>E</FOR>
  <TABLE>0.51 0.49 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>M</FOR>
  <GIVEN>A</GIVEN>
  <TABLE>0.08 0.92 0.86 0.14 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>L</FOR>
  <GIVEN>B</GIVEN>
  <GIVEN>G</GIVEN>
  <TABLE>0.48 0.52 0.28 0.72 0.51 0.49 0.38 0.62 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>B</FOR>
  <TABLE>0.51 0.49 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>G</FOR>
  <GIVEN>A</GIVEN>
  <TABLE>0.08 0.92 0.86 0.14 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>H</FOR>
  <GIVEN>C</GIVEN>
  <GIVEN>K</GIVEN>
  <TABLE>0.48 0.52 0.28 0.72 0.51 0.49 0.38 0.62 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>C</FOR>
  <TABLE>0.50 0.50 </TABLE>
</PROBABILITY>
```

```
<PROBABILITY>
  <FOR>K</FOR>
  <GIVEN>A</GIVEN>
  <TABLE>0.08 0.92 0.86 0.14 </TABLE>
</PROBABILITY>

<PROBABILITY>
  <FOR>A</FOR>
  <TABLE>0.50 0.50 </TABLE>
</PROBABILITY>

</NETWORK>
</BIF>
```

C.2.2 Consulta

```
<query>

        <variable>J</variable>

        <variable>F</variable>

        <variable>L</variable>

        <variable>H</variable>

</query>
```

C.2.3 Evidência

```
<evidences>



        <variable>

                <name>A</name>

                <value>A2</value>

        </variable>



</evidences>
```

C.2.4 Parâmetros do Simulated Annealing

```
<SIMULATED_ANNEALING>
<ALPHA>1.0</ALPHA>
<BETA>1.0</BETA>
<GAMMA>1.0</GAMMA>
<THETA>5.0</THETA>
<STEPS>30000</STEPS>
```

```
<STOP>3000</STOP>
<BLOCKS>4</BLOCKS>
<SEED>10</SEED>
</SIMULATED_ANNEALING>
```

Apêndice D

Diagrama das Redes dos Testes

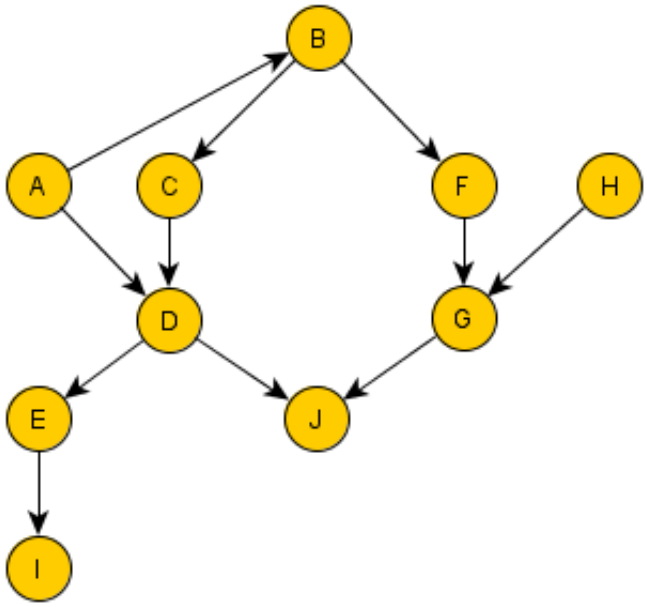


Figura D.1: Diagrama da Rede: Exemplo01

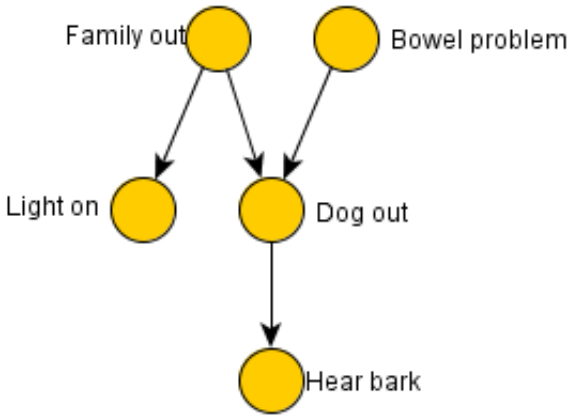


Figura D.2: Diagrama da Rede: DogProblem

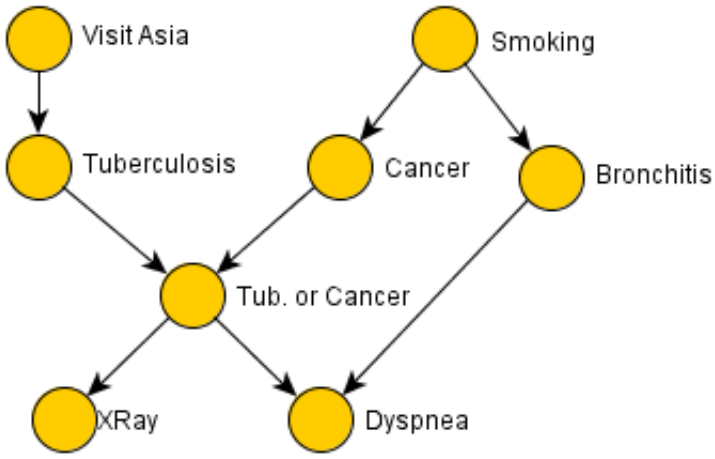


Figura D.3: Diagrama da Rede: Asia

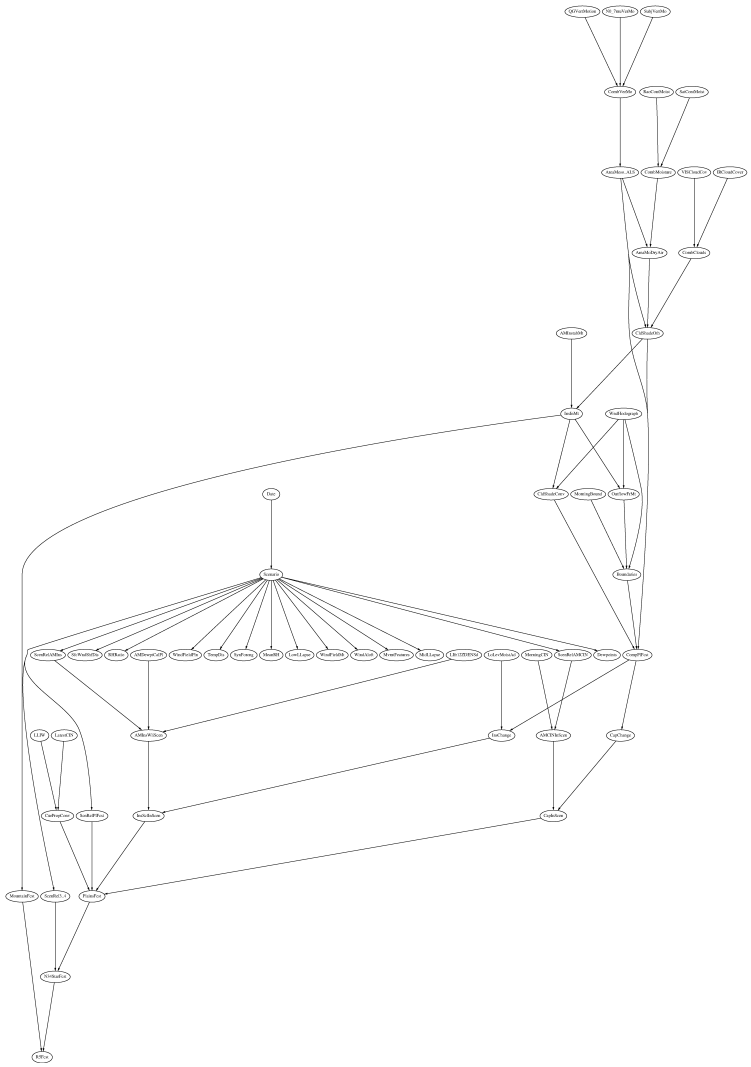


Figura D.4: Diagrama da Rede: Hailfinder25

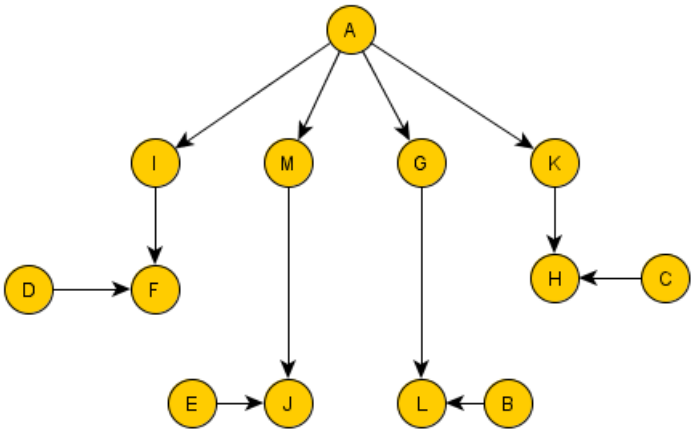


Figura D.5: Diagrama da Rede: Exemplo02

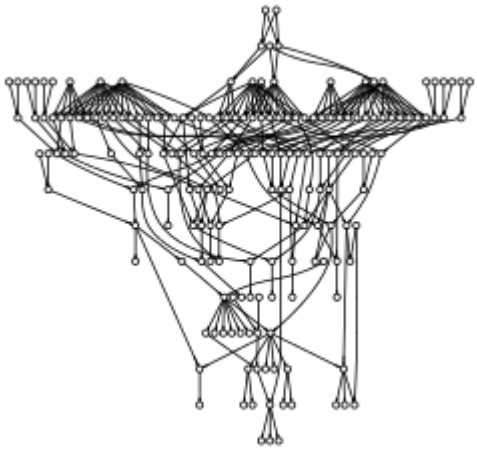


Figura D.6: Diagrama da Rede: Munin1

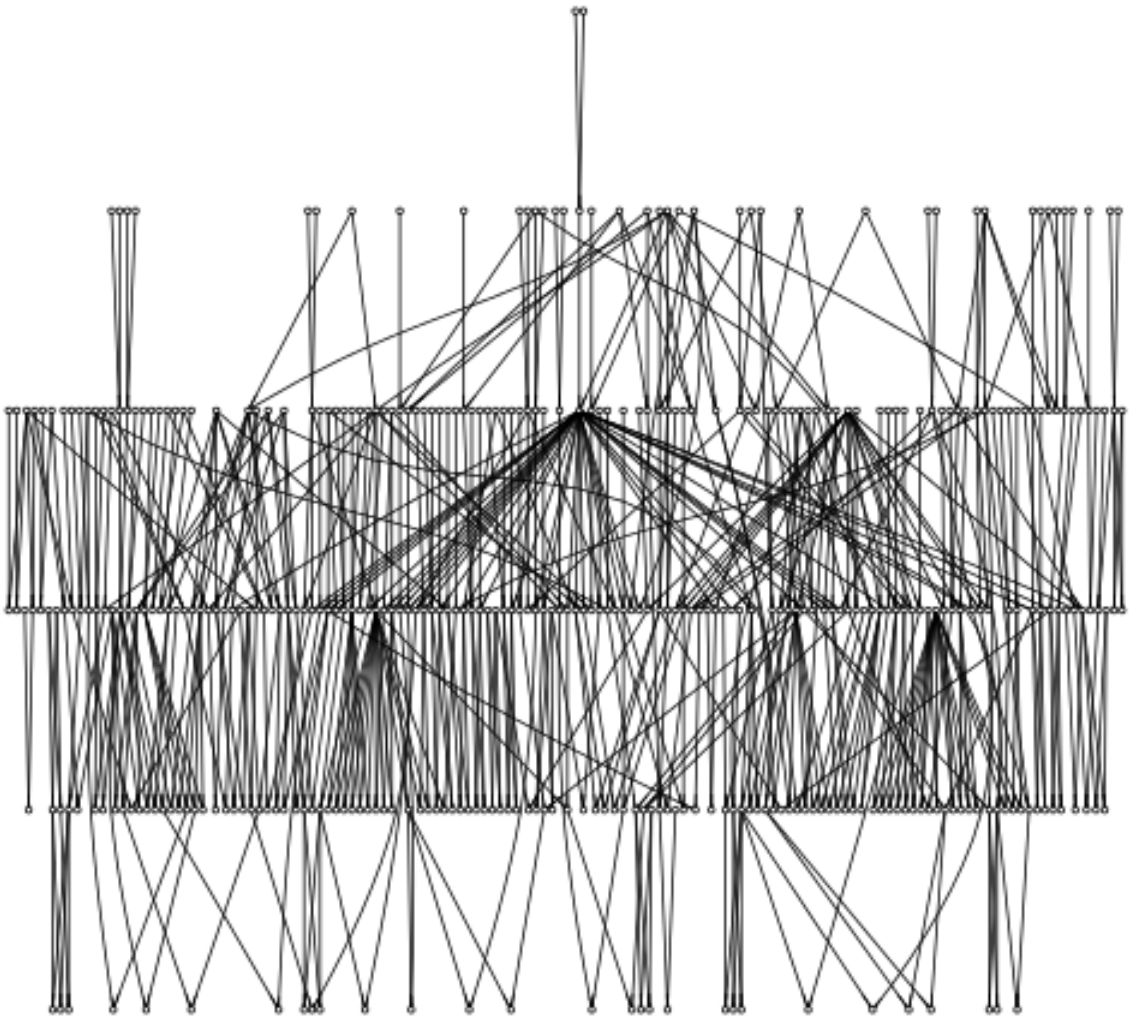


Figura D.7: Diagrama da Rede: Pigs

Referências Bibliográficas

Abramson *et al.***(1996)** Bruce Abramson, John Brown, Ward Edwards, Allan Murphy e Robert L. Winkler. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57–71. Citado na pág. 85

Andreassen *et al.***(1989)** S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck e F. Jensen. MUNIN — an expert EMG assistant. Em John E. Desmedt, editor, *Computer-Aided Electromyography and Expert Systems*, chapter 21. Elsevier Science Publishers, Amsterdam. Citado na pág. 85

Aoyama e Nakano(1999) Yukiya Aoyama e Jun Nakano. IBM RedBook; Practical MPI Programming. Relatório técnico, International Technical Support Organization. URL http://www.redbooks.ibm.com/redbooks/pdfs/sg245380.pdf. Citado na pág. 35

Bang-Jensen e Gutin(2009) J. Bang-Jensen e G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edição. Citado na pág. 99

Barone Jr(1988) Mário Barone Jr. *Álgebra Linear*. USP, 3 edição. Citado na pág. 20

Bondy e Murty(2007) Adrian Bondy e U. S. R. Murty. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 3rd corrected printing. edição. Citado na pág. 99

Bussab e Morettin(2005) Wilton O. Bussab e Pedro A. Morettin. *Estatística Básica*. Saraiva, 5 edição. Citado na pág. 104

Cano *et al.***(1994)** Andrés Cano, José E. Cano e Serafín Moral. Convex sets of probabilities propagation by simulated annealing. Em *In Proceedings of the Fith International Conference IPMU'94*, páginas 4–8. Citado na pág. 16, 80

Cano *et al.***(1993)** José E. Cano, Miguel Delgado e Serafín Moral. An axiomatic framework for propagating uncertainty in directed acyclic networks. *Int. J. Approx. Reasoning*, 8(4):253–280. Citado na pág. 17

Casella e George(1992) George Casella e Edward I. George. Explaining the Gibbs Sampler. *The American Statistician*, 46(3):167–174. Citado na pág. 10

Castillo *et al.***(1997)** E. Castillo, J. M. Gutierrez e A. S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer-Verlag. Citado na pág. 10

Chandra(2001) Rohit et. al Chandra. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers, 1 edição. Citado na pág. 33

Charniak(1991) Eugene Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63. Citado na pág. 17, 85

Colla(2007) Ernesto Coutinho Colla. Aplicação de técnicas de fatoração de matrizes esparsas para inferência em redes bayesianas. Dissertação de Mestrado, Instituto de Matemática e Estatística, Universidade de São Paulo, Brasil. Citado na pág. iii, v, 1, 7, 10, 19, 24, 31, 39, 42, 43, 44, 48, 49, 59, 62, 65, 76, 79, 81, 82, 85, 91, 92, 97

Cowell(1999) R. Cowell. Introduction to inference for Bayesian networks. *Learning in graphical models*, páginas 9–26. Citado na pág. 10

Cozman(2001) Fabio G. Cozman. JavaBayes - Bayesian Networks in Java, 2001. URL http://www.cs.cmu.edu/~{}javabayes. Citado na pág. 17, 77, 86

Cozman(1998) Fabio G. Cozman. The Interchange Format for Bayesian Networks, 1998. URL http://www.cs.cmu.edu/~fgcozman/Research/InterchangeFormat/. Citado na pág. 77

Cozman(2000a) Fabio Gagliardi Cozman. Generalizing variable elimination in bayesian networks. Em *IBERAMIA/SBIA '00: Workshop on Probabilistic Reasoning in Artificial Intelligence*, páginas 27–32. Tec Art. Citado na pág. 4, 7, 10

Cozman(2000b) Fabio Gagliardi Cozman. Credal networks. *Artificial Intelligence*, 120:199–233. Citado na pág. 15, 16

da Rocha e Cozman(2003) José Carlos Ferreira da Rocha e Fabio Gagliardi Cozman. A/r+: Um algoritmo para inferência em redes credais. Em *Encontro Nacional de Inteligência Artificial*. Citado na pág. 15

da Rocha e Cozman(2005) José Carlos Ferreira da Rocha e Fabio Gagliardi Cozman. Inference in credal networks: branch-and-bound methods and the a/r+ algorithm. *Int. J. Approx. Reasoning*, 39(2-3):279–296. Citado na pág. 17

de Campos(2006) Cassio Polpo de Campos. *Redes Credais e Qualitativas: Complexidade e Algoritmos*. Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, Brasil. Citado na pág. 17

de Campos e Cozman(2004) Cassio Polpo de Campos e Fabio Gagliardi Cozman. Inference in credal networs using multilinear programming. Em *Second Starting Ai Researchers' Symposium (STAIRS)*, Frontiers in Artificial Intelligence and Applications. IOS Press. Citado na pág. 17

de Supinski e May(1999) Bronis R. de Supinski e John May. Benchmarking pthreads performance. Em Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 1999, June 28 - Junlly 1, 1999, Las Vegas, Nevada, USA*. CSREA Press. Citado na pág. 37

Dechter e Dechter(1996) Rina Dechter e R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. Em *UAI '96: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence*, páginas 211–219. Citado na pág. 10

Ding(2010) Jianguo Ding. Probabilistic inferences in bayesian networks. *CoRR*, abs/1011.0935. Citado na pág. 4

Dongarra(1987) Jack Dongarra. The linpack benchmark: An explanation. Em Elias N. Houstis, Theodore S. Papatheodorou e Constantine D. Polychronopoulos, editors, *Supercomputing, 1st International Conference, Athens, Greece, June 8-12, 1987, Proceedings*, volume 297 of *Lecture Notes in Computer Science*. Springer. Citado na pág. 37, 72

Fagiuoli e Zaffalon(1998) Enrico Fagiuoli e Marco Zaffalon. 2u: an exact interval propagation algorithm for polytrees with binary variables. *Artificial Intelligence*, 106(1):77–107. Citado na pág. 17

Fung e Del Favero(1994) Robert Fung e Brendan Del Favero. Backward simulation in bayesian networks. Em *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, UAI'94, páginas 227–234, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. Citado na pág. 10

Fung e Chang(1990) Robert M. Fung e Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in bayesian networks. Em *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, páginas 209–220, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co. Citado na pág. 10

Geiger *et al.***(1990)** Dan Geiger, Thomas Verma e Judea Pearl. Identifying independence in Bayesian Networks. *Networks*, 20:507–534. Citado na pág. 4, 7

Grondelle(1999) Jeroen Van Grondelle. Symbolic sparse cholesky factorisation using elimination trees. Dissertação de Mestrado. Citado na pág. 27

Gubitoso(2008) Marcos Dimas Gubitoso. Mac431 - introdução ao processamento paralelo e distribuído. Notas de Aula, 2008. Citado na pág. 32

Haeser e Gomes-Ruggiero(2008) G. Haeser e M. Gomes-Ruggiero. Theoretical aspects of simulated annealing and a two-phase algorithm in global optimization. *TEMA, Tend. Mat. Apl. Comput.*, 9(3):395–404. Citado na pág. 28

Henrion(1986) Max Henrion. Propagating Uncertainty by Logic Sampling in Bayes' Networks. Em *Second Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, PA. Citado na pág. 10

Ide e Cozman(2002) Jaime S. Ide e Fabio G. Cozman. Testing mcmc algorithms with randomly generated bayesian networks, 2002. Citado na pág. 10

Ignácio e Ferreira Filho(2002) Aníbal Alberto Vilcapona Ignácio e Virgílio José Martins Ferreira Filho. MPI: uma ferramenta para implementação paralela. *Pesquisa Operacional*, 22:105 – 116. Citado na pág. 34

Jensen(2001) Finn V. Jensen. *Bayesian Networks and Decision Graphs.* Springer-Verlag New York, Inc., Secaucus, NJ, USA. Citado na pág. 5, 12, 105

Jensen(1996) F.V. Jensen. *An Introduction to Bayesian Networks.* Springer. Citado na pág. 10

Kim e Pearl(1983) J. H. Kim e J. Pearl. A computational model for combined causal and diagnostic reasoning in inference systems. Em *Proceedings of the IJCAI-83*, páginas 190–193, Karlsruhe, Germany. Citado na pág. 10

Lauritzen e Spiegelhalter(1990) S. L. Lauritzen e D. J. Spiegelhalter. Readings in uncertain reasoning. chapter Local computations with probabilities on graphical structures and their application to expert systems, páginas 415–448. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. Citado na pág. 85

Liu(1990) Joseph W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172. Citado na pág. 40

Luecke *et al.***(2009)** Glenn R. Luecke, Jim Coyle, Jim Hoekstra, Marina Kraeva e Ying Xu. Measuring mpi latency and communication rates for small messages, 2009. Citado na pág. 37

Neapolitan(2003) R. E. Neapolitan. *Learning Bayesian Networks.* Prentice Hall. Citado na pág. 5

Nichols *et al.***(1996)** Bradford Nichols, Dick Buttlar e Jacqueline Proulx Farrell. *Pthreads programming - a POSIX standard for better multiprocessing.* O'Reilly. Citado na pág. 33

OpenMP Architecture Review Board(2011) OpenMP Architecture Review Board. *OpenMP Application Program Interface*, 3.1 edição, Julho 2011. URL http://www.openmp.org. Citado na pág. 33

Pearl(1988) J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann. Citado na pág. 4

Pearl(2009) Judea Pearl. *Causality: Models, Reasoning and Inference.* Cambridge University Press, New York, NY, USA. Citado na pág. 3

Ross(2009) Sheldon M. Ross. *First Course in Probability.* Prentice Hall, 8 edição. Citado na pág. 104

Ross(2003) Sheldon M. Ross. *Introduction to Probability Models, Eighth Edition.* Academic Press, 8 edição. Citado na pág. 103

Sanjai(2008) H. A. Sanjai. *Performance Modeling Based Scheduling and Rescheduling of Parallel Applications on Computational Grids.* Tese de Doutorado, Indian Institute of Science. Citado na pág. 36

Shachter(1998) Ross D. Shachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams. Em *In Uncertainty in Artificial Intelligence*, páginas 480–487. Morgan Kaufmann. Citado na pág. 7, 9

Shachter e Peot(1990) Ross D. Shachter e Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. Em *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, páginas 221–234, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co. Citado na pág. 10

Stern e Vavasis(1993) J.M. Stern e S.A. Vavasis. Active set methods for problems in column block angular form. *Computational and Applied Mathematics*, 12(3):199–226. Citado na pág. 26

Stern(1992) Julio M. Stern. Simulated annealing with a temperature dependent penalty function. *INFORMS Journal on Computing*, 4(3):311–319. Citado na pág. 28

Stern(1994) Julio Michael Stern. Esparsidade, estrutura, estabilidade e escalonamento em Álgebra linear computacional. Em Silvio Lemos Meira, editor, *IX Escola de Computação*, páginas 23–35, Recife. UFPE. Citado na pág. 19, 21, 22, 23, 24, 25, 99

Stern(2008) Julio Michael Stern. Decoupling, sparsity, randomization, and objective bayesian inference. *Cybernetics & Human Knowing*, 15(2):49–68. Citado na pág. 19

Taroni *et al.***(2006)** F. Taroni, C. Aitken, P. Garbolino e A. Biedermann. *Bayesian Networks and Probabilistic Inference in Forensic Science.* Statistics in Practice. John Wiley & Sons. Citado na pág. 4

Torti e Wuillemin(2010) Lionel Torti e Pierre-Henri Wuillemin. Structured value elimination with d-separation analysis. Em *Florida Artificial Intelligence Research Society Conference*, páginas 122–127. Citado na pág. 7

Índice Remissivo

bayes-ball, 7
Busca em largura, 25

Cholesky, Fatoração de, 22
Classificação de Matrizes, 19
Computação Paralela
    Eficiência, 35
    Escalabilidade, 35
    Overhead, 36
    Performance, 35
    Speedup, 35
condição de Markov, 4
conjunto convexo de medidas de probabilidade, 15
contexto, 32

d-separação, 4
Decomposição de Matrizes, 20
dependência, 32
distribuição de probabilidade, 104
distribuição marginal, 104

Eliminação de Gauss, 21
    pivoteamento, 21
    preenchimento, 22
escalonamento, 32
eventos independentes, 103
evidência, 5

função de distribuição acumulada, 104

Gibbs, heurística, 25
grafo, 99
    árvore, 100
    adjacentes, vértices, 99
    ancestral, 100
    arcos de preenchimento, 101
    aresta, 99
    caminho, 99
    ciclo, 99
    circuito, 99
    completo, 100
    conexo, 100
    descendente, 100
    direcionado, 99
    eliminação perfeita, 101
    eliminação simplificada, 101
    esposo, 100
    filho, 100
    floresta, 100
    folha, 100
    função de filiação, 99
    grafo moral, 100
    grafo preenchido, 101
    grafos de eliminação, 101
    grau, 99
    incidentes, vértices, 99
    laço, 99
    markov blanket, 100
    não direcionado, 100
    ordem, 99
    ordem de eliminação, 101
    pai, 100
    passeio, 99
    raiz, 100
    separador, 100
    subgrafo, 100
    tamanho, 99
    trilha, 99
    vértices, 99
    vizinhança, 99
    vizinhança fechada, 99
Grafo de Domínio, 13
grafos
    árvore de dissecção, 24
    árvore de eliminação, 23
    ordem de dissecção, 24
    pós-ordem, 24
    teorema da fatoração simbólica, 23

hash, 64, 67

LU, Fatoração, 21

marginalização, 104
matriz
    de permutação, 20
    diagonal, 19

elementar, 21
identidade, 19
inversa, 20
operações elementares, 21
ortogonal, 20
permutação simétrica, 20
positiva definida, 20
Simétrica, 20
transposta, 20
triangular, 19
matriz esparsa, 19
matrizes
quadrada, 19
memória compartilhada, 32
memória distribuída, 33
MPI, 34
mutex, 66, 69, 71, 73

OpenMP, 33
operação atômica, 32

POSIX Threads, 32
potencial, 105
domínio, 105
multiplicação, 105
preempção, 32
probabilidade
espaço amostral, 103
evento, 103
probabilidade adjacente, 4
probabilidade condicional, 103
probabilidade conjunta, 103
processo, 32
programa concorrente, 32
programa sequencial, 32
PThreads, 32, 73–75, 94

Rede Credal
localmente especificada, 16
Redes Bayesianas, 3
Redes Credais
extensão, 16
extensão forte, 16
Redes credais, 15
Regra de Bayes, 104
regra fundamental da probabilidade, 103

Seção crítica, 32
simetria, 34
sincronização, 32

teoria das probabilidades, 103
thread-safe, 66
threads, 32
variáveis questionadas, 6
variável aleatória, 104
variável requisitada, 7

XMLBIF, 76