Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Otimizações para a multiplicação vetor-descritor através do algoritmo *Slice*

Ricardo De Gasperi Presotto

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação.

Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

Porto Alegre, outubro de 2007.

Dados Internacionais de Catalogação na Publicação (CIP)

P934o Presotto, Ricardo De Gasperi
Otimizações para a multiplicação vetor-descritor através do algoritmo Slice / Ricardo De Gasperi Presotto. – Porto Alegre, 2007.
115 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

1. Informática. 2. Avaliação de Desempenho (Informática). 3. Algoritmos. 4. Redes de Autômatos Estocásticos. I. Título.

CDD 004.2

Ficha Catalográfica elaborada pelo
Setor de Processamento Técnico da BC-PUCRS

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Reconhecido pelo Parecer No. 930/98.C.N.E. Homologação Publicada no D.O.U. de 30/12/98.

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Otimizações Para a Multiplicação Vetor-Descritor Através do Algoritmo Slice***", apresentada por Ricardo De Gasperi Presotto, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 23/02/2006 pela Comissão Examinadora:

Prof. Dr. Paulo Henrique Lemelle Fernandes – PPGCC/PUCRS
Orientador

Prof. Dr. César Augusto Fonticielha De Rose – PPGCC/PUCRS

Prof. Dr. Dalcidio Moraes Claudio – PPGCC/PUCRS

Prof. Dr. Nicolas Maillard – UFRGS

Homologada em 26./.11./..07.., conforme Ata No. 24..... pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

AV. IPIRANGA, 6681 - PRÉDIO 16/SALA 106
90619-900 - PORTO ALEGRE -RS -BRASIL
TELEFONE: 55 51 33203611 FAX 55 51 3320.3621
http: //www.pucrs.br/inf/pos e-mail: ppgcc@inf.pucrs.br

Resumo

Os estudos na área de Avaliação de Desempenho de Sistemas têm evoluído nos últimos anos, em especial com a definição do formalismo de Redes de Autômatos Estocásticos (SAN) e com a implementação do algoritmo *Shuffle*, o qual implementa um método eficiente para a execução da multiplicação vetor-descritor, necessária para a resolução de modelos SAN. Mais recentemente, foi proposto um novo método para a multiplicação vetor-descritor, o método *Slice*, que introduzindo novos conceitos, prometia na teoria ser mais eficiente que o tradicional método *Shuffle*. Pois neste estudo, este recém definido e até então pouco explorado método, foi estudado em detalhes e realizada uma implementação incluindo algumas otimizações no seu algoritmo original. Ainda, durante este estudo, foram realizadas algumas modificações no algoritmo do método *Slice* com o intuito de resolver modelos SAN funcionais, uma vez que as versões anteriores não eram capaz de tratar funções. Para demonstrar a eficiência do método *Slice* e das otimizações propostas, diversos experimentos foram conduzidos utilizando dois modelos SAN. Os resultados de tempo e custo computacional foram analisados e discutidos durante este estudo, comparando-os inclusive com resultados do tradicional método *Shuffle*. Desta forma, verificou-se o quanto o método *Slice* pode ser útil na resolução de sistemas, uma vez que os resultados práticos mostram que o método *Slice* é mais eficiente que a solução tradicional (*Shuffle*) na maioria dos casos.

Abstract

Studies on performance evaluation have progressed in the last years, specially with Stochastic Automata Networks (SAN) and with the implementation of the Shuffle algorithm, which implements an efficient method for vector-descriptor multiplication, needed to solve SAN models. Recently, a new vector-descriptor method was proposed, the Slice method, presenting new concepts to be more efficient than the tradicional Shuffle method. In this work the Slice method was studied in detail and a new implementation was developed including some new optimizations. In addition, the Slice algorithm was modified in order to solve functional SAN models, since the previous version was unable to deal with functions. To demonstrate the performance of the Slice method and its optimizations experiments were made using two different SAN models. The results were analyzed and compared with Shuffle results. Thus, it was possible to verify that the Slice method is more efficient than the tradicional Shuffle method in most cases.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Símbolos e Abreviaturas

SAN	Stochastic Automata Networks	19
PEPS	Performance Evaluation of Parallel Systems	27
AUNF	*Additive Unitary Normal Factor*	46
ATC	Álgebra Tensorial Clássica	85
ATG	Álgebra Tensorial Generalizada	85

Capítulo 1

Introdução

A avaliação de desempenho de sistemas tem se tornado cada vez mais difícil, pois os sistemas em geral envolvem a cada dia que passa mais detalhes, tornando-se cada vez mais complexos. O alto nível de detalhes dos sistemas encontrados no mundo real geram diversos problemas para a avaliação de desempenho. Estes problemas vão desde a dificuldade na modelagem do sistema em um certo formalismo, até problemas relacionados ao custo computacional necessário para o cálculo da solução do sistema.

Um dos principais problemas encontrados na avaliação de desempenho de sistemas é a explosão do número de estados do sistema [17]. Muitos estudos vem sendo realizados nesta área com o intuito de encontrar técnicas para solucionar, ou ao menos minimizar este problema. Algumas abordagens, como o formalismo de Redes de Autômatos Estocásticos (SAN) [8, 15], apresentam uma modelagem que visa reduzir o impacto da explosão do espaço de estados. Porém, ainda assim, este problema persiste e tem se tornado uma grande barreira para a evolução da área de Avaliação de Desempenho de Sistemas.

Atualmente, existem diversas técnicas para avaliação de desempenho de sistemas, porém nenhuma delas apresenta resultados satisfatórios para todo e qualquer sistema. O que ocorre, é que algumas técnicas são melhores para alguns sistemas específicos, enquanto outras se mostram mais adequadas para os demais tipos de sistemas. Portanto, estudos nesta área ainda são realizados com grandes expectativas de bons resultados, não pensando em encontrar uma única técnica para cobrir todos os sistemas possíveis, mas principalmente para buscar melhores resultados em termos de desempenho.

Com o surgimento do formalismo de Redes de Autômatos Estocásticos (SAN - *Stochastic Automata Networks*), muitos estudos voltaram-se para o cálculo da resolução de sistemas modelados por este formalismo, uma vez que em termos de armazenamento, SAN apresenta uma forma bastante compacta e eficiente. Sendo assim, o que se espera dos estudos nesta área é uma forma eficiente de resolver sistemas descritos por este formalismo, de maneira a exigir o menor custo computacional possível. Com este objetivo, duas técnicas podem ser citadas, a técnica *Shuffle*, tradicional e já bastante explorada e divulgada mundialmente [8], e a técnica *Slice*, mais recente e ainda pouco divulgada, uma vez que seus estudos até então eram apenas teóricos [9].

1.1 Objetivo do Trabalho

A forma mais adequada de resolver modelos SAN é utilizando-se métodos iterativos, os quais possuem como operação fundamental a multiplicação de um vetor de probabilidades π pelo gerador infinitesimal Q (denominada *Multiplicação Vetor-Descritor*). O objetivo principal deste trabalho é estudar em detalhes a técnica *Slice* de multiplicação vetor-descritor e provar na prática o seu bom desempenho na resolução de sistemas. Além disso, a medida que cada parte do algoritmo desta técnica seja completamente entendida, são descobertas e propostas algumas otimizações, demonstrando tanto na teoria como na prática o ganho de desempenho que cada otimização resultaria.

Como um objetivo secundário, mas também bastante importante para a evolução dos estudos nesta área, pretende-se extender a técnica *Slice* com o intuito de resolver não apenas modelos com taxas exclusivamente constantes, mas também sistemas que apresentem modelagens contendo taxas funcionais. De fato, isto seria fundamental para que a técnica *Slice* seja reconhecida globalmente como uma boa alternativa para resolução de modelos SAN, uma vez que a maioria dos sistemas requerem taxas funcionais em suas modelagens.

Enfim, ao final deste estudo, será possível conhecer melhor a técnica *Slice* e saber o quanto esta técnica realmente contribuirá para a evolução da resolução de sistemas, não apenas na teoria, mas também na prática através de exemplos de modelos SAN.

1.2 Estrutura do Volume

Este trabalho é formado por cinco capítulos incluindo esta introdução e as considerações finais. O estudo é dividido basicamente em três partes principais, os conceitos básicos, a técnica *Slice* na teoria e a análise do desempenho desta técnica.

No Capítulo 2 são apresentados todos os conceitos básicos necessários para o completo entendimento do estudo que segue. Inicialmente é descrito o formalismo de Redes de Autômatos Estocásticos, seguido pela explicação de como é realizada a resolução de modelos SAN e também definição do descritor Markoviano das SAN. Ao final, o capítulo 2 ainda apresenta a tradicional técnica de multiplicação vetor-descritor, conhecida como *Shuffle*, a qual é atualmente o parâmetro de comparação para qualquer nova técnica desta área.

O Capítulo 3 apresenta em detalhes a técnica *Slice*, a qual é o tema principal deste estudo. Nas páginas deste capítulo é possível entender como esta técnica realiza a multiplicação vetor-descritor, passando por cada etapa deste processo, conhecendo assim o conceito de Fator Normal Unitário Aditivo, que é a principal inovação da técnica *Slice*.

No capítulo seguinte, o Capítulo 4, é apresentada uma análise numérica do desempenho da técnica *Slice*, comparando-a eventualmente com o desempenho da técnica *Shuffle*. Esta análise é dividida em duas partes, uma parte teórica, em que é estimado o custo computacional para ambas as técnicas através das fórmulas de custo computacional, e uma parte prática, a qual apresenta resultados práticos da implementação da técnica *Slice*, incluindo memória e tempo necessários para a execução de alguns exemplos de modelos SAN.

Finalmente, nas considerações finais são apresentadas as conclusões em relação ao desempenho da técnica *Slice* para resolução de modelos SAN, demonstrando assim a sua real contribuição para a área de avaliação de desempenho de sistemas, bem como os possíveis trabalhos futuros relacionados a este estudo.

Capítulo 2

Conceitos Básicos

Neste capítulo são apresentados alguns conceitos básicos importantes para o entendimento do trabalho desenvolvido. Primeiramente, o formalismo Redes de Autômatos Estocásticos é descrito de forma a esclarecer os seus principais conceitos, o seu propósito e como é realizada a solução de sistemas neste formalismo. Em seguida, é apresentado o método *Shuffle*, uma técnica tradicional utilizada para resolução de sistemas modelados através do formalismo SAN. Estes conceitos são importantes para facilitar a compreensão da técnica *Slice*, a qual é o foco deste trabalho.

2.1 Redes de Autômatos Estocásticos

Redes de Autômatos Estocásticos (SAN) é um formalismo, baseado em Cadeias de Markov [12, 17], para modelagem de sistemas. Seu surgimento ocorreu nos anos 80 com Plateau [13, 14]. A idéia central deste formalismo é modelar um sistema através de diversos subsistemas, chamados de autômatos, que podem ou não interagir entre si. Através do uso de SAN a modelagem se apresenta mais compacta e modular, tornando-se muito interessante em sistemas com grandes espaços de estados.

A grande diferença e principal vantagem de Redes de Autômatos Estocásticos (SAN) em relação a Cadeias de Markov é exatamente a modularização, pois esta permite uma forma mais eficiente de armazenamento dos dados do sistema, evitando em muitos casos a explosão do espaço de estados [15], algo muito comum em Cadeias de Markov. Desta forma, é possível encontrar soluções para sistemas maiores, com grande quantidade de dados.

É interessante ressaltar que toda SAN pode ser representada por um único autômato estocástico, o qual contém todos os estados possíveis do sistema. Esse único autômato corresponde à Cadeia de Markov equivalente ao modelo SAN. Logo, este formalismo é capaz de manter o poder de modelagem que se tinha com a utilização de Cadeias de Markov.

A seguir, são apresentados cada um dos conceitos relacionados às SAN e ainda uma breve introdução dos métodos numéricos mais utilizados nas soluções estacionárias e transiente das SAN.

2.1.1 Autômatos Estocásticos

Imagine um sistema com um conjunto finito de estados e finitas transições entre eles, onde o sistema passa de um estado para outro por meio de eventos. Esta é uma definição simplificada de um autômato estocástico.

O sistema pode encontrar-se em qualquer um de seus estados, os quais sintetizam as informações relativas às entradas anteriores e informam ainda os possíveis comportamentos do sistema diante das entradas seguintes [10].

A denominação de estocásticos atribuída a esses autômatos dá-se pela razão do tempo ser tratado como uma variável aleatória, a qual obedece uma distribuição exponencial na escala de tempo contínua, ou geométrica no caso de escala de tempo discreta. Neste estudo utiliza-se a escala de tempo contínua.

2.1.2 Estados Locais e Globais

Existem duas formas de observar estados em um modelo SAN. Uma forma é observar o estado individual de cada autômato pertencente ao modelo, chamado este de estado local. Outra, é considerar a combinação dos estados de todos os autômatos envolvidos no modelo, o qual denomina-se de estado global do sistema. Portanto, a mudança do estado de qualquer autômato do modelo, necessariamente acarretará na mudança do estado global do sistema.

As mudanças de estados locais nos autômatos ocorrem através de transições, que por sua vez são disparadas pela ocorrência de eventos ao longo do tempo. Cada transição deve ter ao menos um evento associado a ela, podendo este ser um evento local ou sincronizante.

2.1.3 Eventos Locais e Sincronizantes

Pode-se definir um evento como uma ação relevante que modifica o estado global do sistema. Os eventos estão diretamente associados às transições do modelo, isto é, sempre que um evento ocorre, uma ou mais transições são realizadas.

Em SAN, os eventos podem ser de dois tipos, locais ou sincronizantes. Eventos locais são aqueles que modificam o estado local de um único autômato sem interferir no estado local dos demais autômatos do modelo. Estes eventos são normalmente utilizados quando se quer modelar comportamentos independentes entre autômatos.

Por outro lado, é possível modelar sincronismo entre dois ou mais autômatos através de eventos sincronizantes. Estes eventos disparam transições em mais de um autômato, modificando assim os estados locais dos autômatos envolvidos.

Todo modelo SAN para estar completo deve apresentar uma tabela de eventos. Esta tabela contém os identificadores dos eventos, as taxas de ocorrências e ainda o tipo de cada evento pertencente ao modelo. Deve-se consultar essa tabela para distinguir os eventos locais dos sincronizantes, uma vez que a representação gráfica não apresenta tal informação.

A Figura 2.1 apresenta um modelo SAN formado por dois autômatos com três eventos locais e um evento sincronizante, como pode ser observado na Tabela 2.1.

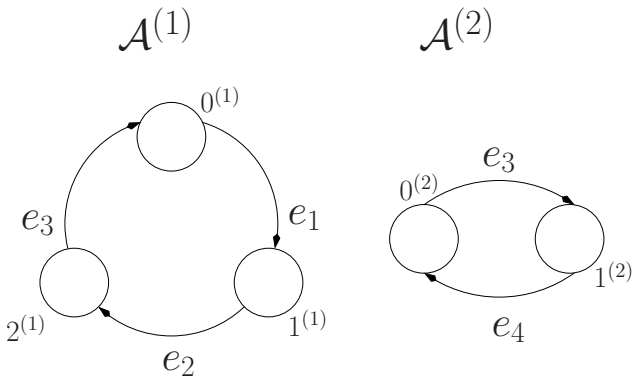


Figura 2.1: Modelo SAN com 2 autômatos com evento sincronizante

Evento	Taxa de Ocorrência	Tipo
e_1	τ_1	Local
e_2	τ_2	Local
e_3	τ_3	Sincronizante
e_4	τ_4	Local

Tabela 2.1: Taxa de ocorrência e tipo dos eventos do modelo SAN apresentado na Figura 2.1.

Neste exemplo, o autômato $\mathcal{A}^{(1)}$ possui três estados $0^{(1)}$, $1^{(1)}$ e $2^{(1)}$, enquanto que o autômato $\mathcal{A}^{(2)}$ do modelo possui apenas dois estados $0^{(2)}$ e $1^{(2)}$. O modelo apresenta ainda quatro eventos, sendo três eventos locais (e_1, e_2 e e_4) e um sincronizante (e_3). Quando o evento sincronizante (e_3) ocorre, o autômato $\mathcal{A}^{(1)}$ passa do estado local $2^{(1)}$ para $0^{(1)}$ ao mesmo passo que o autômato $\mathcal{A}^{(2)}$ sai do estado $0^{(2)}$ e vai para o estado $1^{(2)}$.

2.1.4 Taxas e Probabilidades Funcionais

Todo evento em um modelo SAN deve ter associado a si uma taxa de ocorrência e uma probabilidade de rotação. Tanto a taxa de ocorrência como a probabilidade de rotação podem ser definidas como valores constantes ou valores funcionais. Quando as taxas e/ou probabilidades são definidas como valores funcionais, estas são então ditas taxas e/ou probabilidades funcionais. Neste caso, os valores assumidos por estas taxas e probabilidades dependem dos estados locais dos demais autômatos do modelo.

A utilização de taxas e probabilidades funcionais proporciona uma segunda possibilidade de interação entre autômatos nos modelos SAN, a outra possibilidade é a utilização de eventos sincronizantes[1] como visto anteriormente. As funções associadas com as taxas e/ou probabilida-

[1] A utilização de taxas e probabilidades funcionais não está limitada aos eventos locais e podem ser empregadas nos eventos sincronizantes exatamente como nos eventos locais.

des permitem atribuir a um mesmo evento diferentes valores conforme o estado global do sistema.

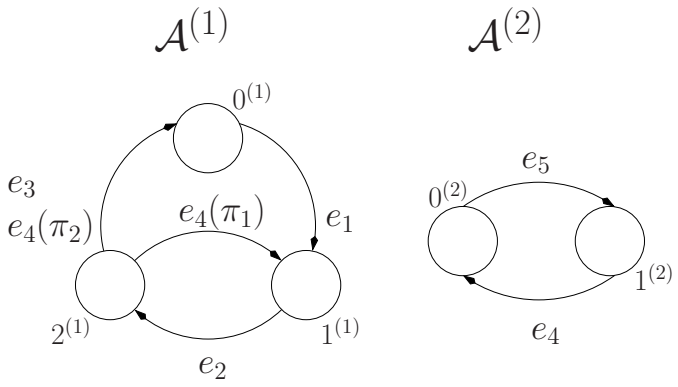


Figura 2.2: Modelo SAN com dois autômatos, um evento sincronizante e uma taxa funcional

Evento	Taxa de Ocorrência	Tipo
e_1	τ_1	Local
e_2	τ_2	Local
e_3	τ_3	Local
e_4	τ_4	Sincronizante
e_5	f_1	Local

Tabela 2.2: Taxa de ocorrência dos eventos do modelo SAN apresentado na Figura 2.2.

A Figura 2.2 apresenta um exemplo de modelo SAN com 2 autômatos, o primeiro com 3 estados e o segundo com 2 estados. Neste exemplo, pode-se observar no autômato $\mathcal{A}^{(1)}$ a utilização de probabilidades para as diferentes transições do evento sincronizante e_4. Além disso, analisando a Tabela 2.2, nota-se que o evento e_5 do autômato $\mathcal{A}^{(2)}$ possui uma função f_1 associada a sua taxa de ocorrência. Dessa forma, a taxa de ocorrência deste evento será determinada pela função f_1.

A função f_1 por sua vez é definida por:

$$f_1 = \begin{cases} \lambda_1 & \text{se autômato } \mathcal{A}^{(1)} \text{ está no estado } 0^{(1)}; \\ 0 & \text{se autômato } \mathcal{A}^{(1)} \text{ está no estado } 1^{(1)}; \\ \lambda_2 & \text{se autômato } \mathcal{A}^{(1)} \text{ está no estado } 2^{(1)}; \end{cases}$$

Isto significa que a transição do estado $0^{(2)}$ para o estado $1^{(2)}$ ocorre com uma taxa de ocorrência λ_1, caso o autômato $\mathcal{A}^{(1)}$ esteja no estado $0^{(1)}$, ocorre com uma taxa λ_2, caso o autômato $\mathcal{A}^{(1)}$ esteja no estado $2^{(1)}$, e não ocorre caso o autômato $\mathcal{A}^{(1)}$ esteja no estado $1^{(1)}$.

2.1.5 Função de Atingibilidade

Função de atingibilidade nada mais é do que uma função booleana que determina os estados atingíveis em um modelo SAN. Esta função deve ser utilizada sempre que for identificado que um

ou mais estados do sistema nunca serão alcançados. Por outro lado, a função de atingibilidade deve assumir o valor 1 caso todos os estados do modelo SAN sejam atingíveis.

Considerando a Figura 2.1 podemos supor, por exemplo, que o autômato $A^{(1)}$ não pode se encontrar no estado $2^{(1)}$ se o autômato $A^{(2)}$ estiver no estado $1^{(2)}$, e vice-versa.

Para isto deve-se definir a seguinte função de atingibilidade[2]:

$$reachability = !\,((st\ A^{(1)} ==\ 2^{(1)})\ \&\&\ (st\ A^{(2)} ==\ 1^{(2)}));$$

Dessa forma, os possíveis estados globais desta SAN seriam: $0^{(1)}0^{(2)}$, $0^{(1)}1^{(2)}$, $1^{(1)}0^{(2)}$, $1^{(1)}1^{(2)}$ e $2^{(1)}0^{(2)}$.

2.1.6 Funções de Integração

As funções de integração podem ser definidas em qualquer modelo SAN. A proposta destas funções é obter a probabilidade do modelo encontrar-se em um determinado estado.

As funções de integração[3] podem ser definidas de forma a avaliarem a probabilidade do modelo estar em um conjunto de estados, podendo obter assim índices de desempenho e confiabilidade do modelo. A avaliação dessas funções é realizada sobre o *vetor de probabilidades* que contém a probabilidade do modelo se encontrar em cada um de seus possíveis estados. Por exemplo, considerando a SAN apresentada na Figura 2.2, a função de integração f, definida abaixo, avaliaria a probabilidade do autômato $A^{(1)}$ estar no estado $2^{(1)}$:

$$f = (st\ A^{(1)} ==\ 2^{(1)})$$

2.2 Solução Estacionária das SAN

2.2.1 Métodos Numéricos Iterativos

Existem alguns métodos iterativos que podem ser aplicados à resolução das SAN, entre estes, destaca-se: o método de Arnoldi, GMRES e o método da Potência implementados na ferramenta PEPS [1]. Nestes métodos, a operação básica é a multiplicação de um vetor de probabilidades por uma matriz, ou seja, a cada iteração é gerada uma seqüência $\pi^{(k)}$ de valores aproximados do vetor de probabilidades estacionárias que devem convergir para a solução π. Logo o número de iterações torna-se um fator relevante na verificação do custo total de aplicação destes métodos.

O Método da Potência em especial será o foco do estudo comparativo deste trabalho, uma vez que serão estudados algoritmos de resolução de SAN que se utilizam deste método. Todos os conceitos vistos nesta seção podem ser encontrados com maiores detalhes em [6, 11, 17].

[2] A função de atingibilidade utiliza a mesma notação da linguagem de programação C.

[3] Da mesma forma que as funções de atingibilidade, as funções de integração fazem uso da sintaxe da linguagem de programação C.

O Método da Potência Tradicional

O princípio básico deste método é, através da multiplicação de um vetor de probabilidades $\pi^{(0)}$ por uma matriz de probabilidades P, obter o vetor solução ou auto-vetor π, isto se considerarmos que a matriz P estará elevada no ∞, ou seja, $\pi = \pi^{(0)}\ P^{\infty}$. Normalmente, as soluções dos métodos iterativos, como é o caso do método da Potência, são obtidas através de aproximações constantes, onde uma solução é considerada satisfatória somente se obedece a algum critério de tolerância estabelecido. Este critério pode ser observado através da diferença entre as iterações anteriores e a atual iteração. Desta forma, a solução esperada é $\pi^{(n)}$, e para sabermos seu valor é necessário conhecer o valor de $\pi^{(n-1)}$, e assim sucessivamente, como a seguir:

$$\begin{aligned}
\pi^{(1)} &= \pi^{(0)}\ P \\
\pi^{(2)} &= \pi^{(1)}\ P \\
\pi^{(3)} &= \pi^{(2)}\ P \\
&\ldots \\
\pi^{(n)} &= \pi^{(n-1)}\ P
\end{aligned}$$

Portanto, para o cálculo do vetor de probabilidades estacionárias $\pi^{(n)}$, considera-se o esquema iterativo descrito genericamente por $\pi^{(n)} = \pi^{(n-1)}\ P$. No âmbito da multiplicação vetor-descritor necessária para solução de modelos SAN, considera-se P uma dada matriz de transição Q, que deve estar normalizada (aqui representada como $\frac{Q}{\Delta}$), somada a uma matriz identidade I. Além disto, $\pi^{(0)}$ pode ser um vetor de probabilidades inicialmente equiprovável, ou um vetor inicializado com probabilidades pré-definidas segundo algum critério. A solução que buscamos neste caso é um vetor de probabilidades normalizado, onde a soma de seus elementos é igual a 1. Logo, o sistema linear a resolver é:

$$\pi \left(\frac{Q}{\Delta} + I \right) = 0$$

Considerando que um descritor SAN é um gerador infinitesimal dado por $(\frac{Q}{\Delta} + I)$, a solução estacionária de uma SAN é simplesmente a resolução deste sistema linear. Neste caso, com matrizes de transição, realiza-se o seguinte esquema de iterações:

$$\pi^{(n)} = \pi^{(n-1)} \left(\frac{Q}{\Delta} + I \right) \tag{2.1}$$

Neste esquema de iterações demonstrado na Equação 2.1, tem-se o termo $(\frac{Q}{\Delta} + I)$ que representa uma matriz de probabilidade (denominada matriz P anteriormente). Logo, o esquema iterativo pode ser representado por:

$$\begin{aligned}
\pi^{(1)} &= \pi^{(0)} \left(\frac{Q}{\Delta} + I\right) \\
\pi^{(2)} &= \pi^{(1)} \left(\frac{Q}{\Delta} + I\right) \\
\pi^{(3)} &= \pi^{(2)} \left(\frac{Q}{\Delta} + I\right) \\
&\cdots \\
\pi^{(n)} &= \pi^{(n-1)} \left(\frac{Q}{\Delta} + I\right)
\end{aligned}$$

Note que o esquema iterativo pode também ser descrito como:

$$\pi^{(n)} = \pi^{(n-1)}\, I + \pi^{(n-1)}\, \frac{Q}{\Delta} \tag{2.2}$$

2.3 O Descritor Markoviano das SAN

Para entender o Descritor Markoviano das SAN, primeiramente é necessário compreender o conceito de matriz de transição de um autômato estocástico.

A matriz de transição de um autômato estocástico nada mais é que uma representação matricial das possíveis trocas de estados do autômato, levando-se em conta também as taxas das transições.

Para entender como pode-se obter a matriz de transição correspondente a um determinado autômato estocástico, consideremos a cadeia de Markov descrita na Figura 2.3. A matriz de transição Q desta cadeia de Markov é uma matriz quadrada de ordem n_Q igual ao número de estados pertencentes a esta cadeia, neste caso, $n_Q = 4$ (estados A, B, C e D respectivamente).

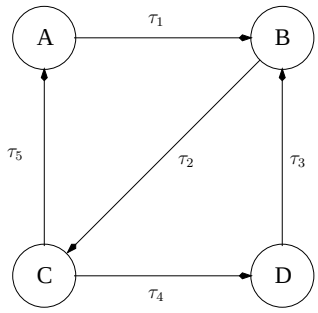


Figura 2.3: Exemplo de cadeia de Markov

Cada linha e cada coluna da matriz de transição Q é associada a um estado da cadeia de Markov segundo a ordem lexicográfica dos mesmos. Logo, considerando o exemplo dado, a primeira linha e a primeira coluna de Q correspondem ao estado A, a segunda linha e a segunda coluna correspondem ao estado B, assim como a terceira linha e a terceira coluna correspondem ao estado C, e a quarta linha e quarta coluna ao estado D. Sendo assim, a matriz de transição Q correspondente à cadeia de Markov representada pela Figura 2.3 é:

$$Q = \begin{pmatrix} -\tau_1 & \tau_1 & 0 & 0 \\ 0 & -\tau_2 & \tau_2 & 0 \\ \tau_5 & 0 & -(\tau_4 + \tau_5) & \tau_4 \\ 0 & \tau_3 & 0 & -(\tau_3) \end{pmatrix}$$

Os elementos q_{ij} de uma matriz Q correspondem as taxas de disparo das transições do autômato estocástico que se está sendo representando, considerando o estado associado à linha i como o estado de partida da transição e o estado associado à coluna j o estado de chegada desta mesma transição. Dessa forma, obtém-se os elementos não-diagonais da matriz Q, os elementos da diagonal principal desta matriz são definidos de forma a ser nula a soma dos elementos em cada uma das linhas da matriz. Assim, a diagonal expressará o ajuste necessário para que a soma de todos os elementos de cada linha seja igual a zero. Portanto, os elementos da diagonal principal serão necessariamente negativos ou nulos.

O formalismo SAN fornece uma descrição compacta da matriz de transição (gerador infinitesimal) correspondente à Cadeia de Markov associada ao modelo completo, a qual é chamada de *Descritor Markoviano*. Essa é mais uma vantagem do formalismo SAN em relação aos demais formalismos.

O descritor Markoviano, considerando as matrizes de transição de cada autômato do modelo, descreve de forma algébrica o gerador infinitesimal da cadeia de Markov associada à SAN [6, 15].

Cada autômato estocástico $A^{(i)}$ de uma modelo SAN possui uma matriz de transição local $Q_l^{(i)}$, que agrupa todas as taxas correspondentes as transições locais, e $2E$ matrizes de transição sincronizantes, as quais agrupam todas as taxas de transições dos eventos sincronizantes e do conjunto ε, chamadas $Q_{e^+}^{(i)}$ e $Q_{e^-}^{(i)}$, onde E é o total de eventos sincronizantes do modelo SAN e ε é o conjunto de identificadores destes eventos. Estas matrizes de transição, locais e sincronizantes, são utilizadas para expressar o descritor Markoviano de uma rede de autômatos estocásticos, como apresentado nas seções subseqüentes.

2.3.1 Definição do Descritor Markoviano

O descritor Markoviano de uma rede de autômatos estocásticos é descrito através de operações tensoriais entre as matrizes de transição do modelo. Este descritor é expresso em duas partes: a parte local, que corresponde às transições locais de cada autômato do modelo, e a parte sincronizante, que corresponde às transições disparadas pelos eventos sincronizantes pertencentes ao modelo SAN.

Parte Local do Descritor Markoviano

A parte local do descritor Markoviano é definida por uma soma tensorial das matrizes locais de cada autômato do modelo, como segue:

$$Q_l = \bigoplus_{i=1}^{n}{}_g \, Q_l^{(i)}$$

Considerando o exemplo de modelo SAN apresentado pela Figura 2.1, vejamos como seriam descritos os seus eventos locais. Para cada autômato $A^{(i)}$ do modelo, existe uma matriz de transição local $Q_l^{(i)}$ associada. Logo, para o exemplo da Figura 2.1 tem-se duas matrizes locais, $Q_l^{(1)}$ e $Q_l^{(2)}$. A matriz de transição local Q_l do autômato equivalente à esta rede de autômatos estocásticos corresponde a soma tensorial das matrizes de transições locais, como é mostrado a seguir:

$$Q_l = Q_l^{(1)} \oplus Q_l^{(2)} = \begin{pmatrix} -\tau_1 & \tau_1 & 0 \\ 0 & -\tau_2 & \tau_2 \\ 0 & 0 & 0 \end{pmatrix} \bigoplus \begin{pmatrix} 0 & 0 \\ \tau_4 & -\tau_4 \end{pmatrix}$$

$$Q_l = \left(\begin{array}{cc|cc|cc} -(\tau_1) & 0 & \tau_1 & 0 & 0 & 0 \\ \tau_4 & -(\tau_1+\tau_4) & 0 & \tau_1 & 0 & 0 \\ \hline 0 & 0 & -(\tau_2) & 0 & \tau_2 & 0 \\ 0 & 0 & \tau_4 & -(\tau_2+\tau_4) & 0 & \tau_2 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \tau_4 & -(\tau_4) \end{array}\right)$$

Portanto, uma matriz de transição local agrupa todas as taxas dos eventos locais do autômato. Se o modelo SAN da Figura 2.1 apresentasse somente eventos locais, o gerador infinitesimal Q do autômato equivalente a este modelo seria simplesmente à soma tensorial[4] das matrizes de transição locais do modelo.

Parte Sincronizante do Descritor Markoviano

A parte sincronizante do descritor Markoviano é composta pelas matrizes de transição dos eventos sincronizantes do modelo SAN. Para cada evento sincronizante e para cada autômato do modelo são descritas duas matrizes: uma descrevendo as ocorrências do evento sincronizante (matriz positiva), e a outra (matriz negativa) descrevendo o ajuste diagonal correspondente a cada taxa descrita na matriz de ocorrência.

Para cada evento sincronizante do modelo SAN é definido um autômato mestre e um ou mais autômatos escravos, dentre os autômatos que sofrem influência destes eventos. Podem ainda existir autômatos que não são influenciados por determinados eventos sincronizantes. Neste caso, as matrizes positivas e negativas destes eventos para estes autômatos serão matrizes identidade, uma vez que não ocorrerá nenhuma mudança de estado nestes autômatos em decorrência de tal evento.

A matriz positiva correspondente ao autômato mestre contém a taxa de disparo de um dado evento sincronizante e, enquanto que as matrizes positivas dos autômatos escravos contêm uma taxa de disparo igual a um, ou igual a probabilidade da transição relacionada ao evento ocorrer. Vejamos a parte sincronizante positiva do modelo SAN da Figura 2.1 considerando o autômato $A^{(1)}$ mestre para o evento sincronizante e_3:

$$Q_{e_3^+} = Q_{e_3^+}^{(1)} \otimes Q_{e_3^+}^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \tau_3 & 0 & 0 \end{pmatrix} \bigotimes \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

[4] A definição de soma e produto tensorial pode ser vista no apêndice A.

$$Q_{e_3^+} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \tau_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

As matrizes de ajuste diagonal, ou matrizes negativas, possuem apenas elementos na diagonal principal, os quais correspondem aos ajustes correspondentes as taxas apresentadas nas matrizes positivas. Na matriz do autômato mestre, a taxa é igual a da matriz positiva, porém negativa, ao passo que nas matrizes dos autômatos escravos, as taxas são sempre iguais a um. Sendo assim, para o exemplo da Figura 2.1 teríamos as seguintes matrizes negativas, considerando o autômato $A^{(1)}$ como mestre:

$$Q_{e_3^-} = Q^{(1)}_{e_3^-} \otimes Q^{(2)}_{e_3^-} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\tau_3 \end{pmatrix} \bigotimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$Q_{e_3^-} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & -\tau_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Portanto, para cada evento sincronizante do modelo SAN são descritos dois produtos tensoriais, um produto das matrizes positivas e outro das matrizes negativas, sendo definido genericamente pela seguinte equação:

$$Q = \sum_{e \in \varepsilon} \left(\bigotimes_{i=1}^{n}{}_g \, Q^{(i)}_{e^+} + \bigotimes_{i=1}^{n}{}_g \, Q^{(i)}_{e^-} \right)$$

O Descritor Markoviano Completo

Como dito anteriormente, o descritor Markoviano é composto de duas partes, a parte correspondente as matrizes de transição locais e a parte contendo as matrizes sincronizantes (positivas e negativas). Sendo assim, o descritor Markoviano completo para o exemplo da Figura 2.1 pode ser descrito da seguinte forma:

$$\begin{aligned} Q &= Q_l^{(1)} \bigoplus_g Q_l^{(2)} \\ &+ Q_{e_3^+}^{(1)} \bigotimes_g Q_{e_3^+}^{(2)} \\ &+ Q_{e_3^-}^{(1)} \bigotimes_g Q_{e_3^-}^{(2)} \end{aligned}$$

Resultando para este exemplo no seguinte descritor Markoviano:

$$Q = \left(\begin{array}{cc|cc|cc} -(\tau_1) & 0 & \tau_1 & 0 & 0 & 0 \\ \tau_4 & -(\tau_1+\tau_4) & 0 & \tau_1 & 0 & 0 \\ \hline 0 & 0 & -(\tau_2) & 0 & \tau_2 & 0 \\ 0 & 0 & \tau_4 & -(\tau_2+\tau_4) & 0 & \tau_2 \\ \hline 0 & \tau_3 & 0 & 0 & -(\tau_3) & 0 \\ 0 & 0 & 0 & 0 & \tau_4 & -(\tau_4) \end{array}\right)$$

Portanto, considerando que o descritor Markoviano completo é a soma da parte local com a parte sincronizante de cada autômato do modelo SAN, pode-se descrevê-lo genericamente através da seguinte equação:

$$Q = \bigoplus_{i=1}^{n}{}_g\, Q_l^{(i)} + \sum_{e\in\varepsilon} \left(\bigotimes_{i=1}^{n}{}_g\, Q_{s^+}^{(i)} + \bigotimes_{i=1}^{n}{}_g\, Q_{s^-}^{(i)} \right)$$

Considerando ainda que uma soma tensorial entre matrizes é decomposta na soma convencional de produtos tensoriais destas matrizes [5], o descritor Markoviano pode ser descrito de outra forma como mostra a Tabela 2.3, a qual apresenta todas as matrizes de transição necessárias para descrever uma rede de autômatos estocásticos.

2.4 Técnica *Shuffle*

Também conhecida como *Técnica de Embaralhamento*, a técnica *Shuffle*[6] [2, 6, 8] implementa uma técnica tradicional utilizada em uma das operações fundamentais realizadas pelos métodos iterativos, a multiplicação de um vetor de probabilidades v, pelo gerador infinitesimal Q. Esta multiplicação é dada pela seguinte expressão, sendo N o número de autômatos na rede, e E o número de eventos sincronizantes:

$$vQ = v \sum_{j=1}^{(N+2E)} \left[\bigotimes_{i=1}^{N}{}_g\, Q_j^{(i)} \right] = \sum_{j=1}^{(N+2E)} \left[v \bigotimes_{i=1}^{N}{}_g\, Q_j^{(i)} \right]$$

[5] A definição de soma tensorial pode ser vista em detalhes na seção A.1.2 do apêndice A.

[6] A técnica ou algoritmo *Shuffle* foi desenvolvido por Fernandes, Plateau e Stewart [8]. Apesar de não ter recebido inicialmente esta denominação, o mesmo vem sendo recentemente denominado como tal [3].

$\sum$	N		$\begin{array}{ccccccccc} Q_l^{(1)} & \underset{g}{\otimes} & I_{n_2} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & I_{n_{N-1}} & \underset{g}{\otimes} & I_{n_N} \\ I_{n_1} & \underset{g}{\otimes} & Q_l^{(2)} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & I_{n_{N-1}} & \underset{g}{\otimes} & I_{n_N} \\ & & & & \vdots & & & & \\ I_{n_1} & \underset{g}{\otimes} & I_{n_2} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & Q_l^{(N-1)} & \underset{g}{\otimes} & I_{n_N} \\ I_{n_1} & \underset{g}{\otimes} & I_{n_2} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & I_{n_{N-1}} & \underset{g}{\otimes} & Q_l^{(N)} \end{array}$
	$2E$	e^+	$\begin{array}{ccccccccc} Q_{1^+}^{(1)} & \underset{g}{\otimes} & Q_{1^+}^{(2)} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & Q_{1^+}^{(N-1)} & \underset{g}{\otimes} & Q_{1^+}^{(N)} \\ & & & & \vdots & & & & \\ Q_{E^+}^{(1)} & \underset{g}{\otimes} & Q_{E^+}^{(2)} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & Q_{E^+}^{(N-1)} & \underset{g}{\otimes} & Q_{E^+}^{(N)} \end{array}$
		e^-	$\begin{array}{ccccccccc} Q_{1^-}^{(1)} & \underset{g}{\otimes} & Q_{1^-}^{(2)} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & Q_{1^-}^{(N-1)} & \underset{g}{\otimes} & Q_{1^-}^{(N)} \\ & & & & \vdots & & & & \\ Q_{E^-}^{(1)} & \underset{g}{\otimes} & Q_{E^-}^{(2)} & \underset{g}{\otimes} & \cdots & \underset{g}{\otimes} & Q_{E^-}^{(N-1)} & \underset{g}{\otimes} & Q_{E^-}^{(N)} \end{array}$

Tabela 2.3: Descritor Markoviano

Em especial, são tratadas as particularidades envolvidas na seguinte operação de multiplicação:

$$v \bigotimes_{i=1}^{N}{}_g \, Q_j^{(i)}$$

A seguir, serão estabelecidas algumas definições sobre seqüências finitas de matrizes, importantes para a apresentação do algoritmo utilizado no método *Shuffle*.

Sejam

n_i dimensão da i-ésima matriz de uma seqüência;

$nleft_i$ produto das dimensões de todas as matrizes à esquerda da i-ésima matriz de uma seqüência, *i.e.*, $\prod_{k=1}^{i-1} n_k$ (caso particular: $nleft_1 = 1$);

$nright_i$ produto das dimensões de todas as matrizes à direita da i-ésima matriz de uma seqüência, *i.e.*, $\prod_{k=i+1}^{N} n_k$ (caso particular: $nright_N = 1$);

$\bar{n}_i$ produto das dimensões de todas as matrizes exceto a i-ésima matriz de uma seqüência, *i.e.*, $\prod_{k=1,k\neq i}^{N} n_k$ ($\bar{n}_i = nleft_i\ nright_i$);

Veremos na próxima seção que através da propriedade de decomposição de produtos tensoriais a multiplicação de um vetor v pelo termo $\bigotimes_{i=1}^{N} Q^{(i)}$ pode ser reescrita de uma forma mais modular utilizando-se fatores normais. Fator normal é um caso especial de produto tensorial entre uma matriz Q e uma matriz identidade I, sendo dois fatores normais possíveis: $Q \underset{g}{\otimes} I$ *e* $I \underset{g}{\otimes} Q$. Assim, a implementação desta operação torna-se mais flexível.

2.4.1 Multiplicação de Fatores Normais

Com o intuito de decompor o termo $\bigotimes_{i=1}^{N} Q^{(i)}$ e assim facilitar a operação de multiplicação do vetor pelo termo, faz-se uso da propriedade de decomposição de produtos tensoriais mostrada abaixo.

$$\bigotimes_{i=1}^{N} A^{(i)} = \prod_{i=1}^{N} \left(I_{nleft_i} \otimes A^{(i)} \otimes I_{nright_i}\right)$$

Dessa forma, o termo $\bigotimes_{i=1}^{N} Q^{(i)}$ pode ser reescrito da seguinte maneira:

$$\begin{aligned} Q^{(1)} \otimes Q^{(2)} \otimes \ \ldots \ \otimes \ Q^{(N-1)} \otimes Q^{(N)} = \ & Q^{(1)} \otimes I_{nright_1} \times \\ & I_{nleft_2} \otimes Q^{(2)} \otimes I_{nright_2} \times \\ & \ldots \\ & I_{nleft_{N-1}} \otimes Q^{(N-1)} \otimes I_{nright_{N-1}} \times \\ & I_{nleft_N} \otimes Q^{(N)} \end{aligned}$$

Essa nova forma de calcular o produto tensorial entre as matrizes da SAN simplifica bastante a multiplicação do vetor v pelo descritor infinitesimal Q. Pois agora, é possível multiplicar o vetor por cada fator normal individualmente, podendo-se criar fases bem distintas. Assim, o cálculo como um todo se torna mais modular e o custo com memória diminui consideravelmente, uma vez que não é mais necessário gerar todo o descritor Markoviano antes de efetuar a multiplicação com o vetor.

Vale salientar que tudo isso é possível graças à propriedade de associatividade da multiplicação (convencional) de matrizes. Além disto, a propriedade da comutatividade entre fatores normais (citada na Equação A.13 da Seção A.1.3) permite a multiplicação de fatores normais em qualquer ordem, se necessário.

Sendo assim, para calcular o produto vetor-descritor, basta multiplicar o vetor v pelo primeiro fator normal, multiplicar então o resultado pelo segundo fator normal, e assim por diante, até o último dos fatores normais.

Por exemplo, considerando as três matrizes $A^{(1)}$, $A^{(2)}$ e $A^{(3)}$, anteriormente teríamos o seguinte produto a resolver:

$$v \times (A^{(1)} \otimes A^{(2)} \otimes A^{(3)})$$

Aplicando a propriedade de decomposição de produtos tensoriais, temos agora as seguintes operações a realizar:

$$v \times (A^{(1)} \otimes I_{n_2} \otimes I_{n_3}) \times (I_{n_1} \otimes A^{(2)} \otimes I_{n_3}) \times (I_{n_1} \otimes I_{n_2} \otimes A^{(3)})$$

Ou simplesmente:

$$v \times (A^{(1)} \otimes I_{nright_1}) \times (I_{nleft_2} \otimes A^{(2)} \otimes I_{nright_3}) \times (I_{nleft_3} \otimes A^{(3)})$$

Portanto, genericamente temos:

$$v \times (A^{(1)} \otimes I_{nright_1}) \times (I_{nleft_2} \otimes A^{(2)} \otimes I_{nright_3}) \times \cdots \times (I_{nleft_N} \otimes A^{(N)})$$

Onde, N é o número total de matrizes da SAN.

Claramente nota-se que agora é possível realizar a multiplicação do vetor por cada matriz individualmente, não sendo mais necessário trabalhar com todas as matrizes em um mesmo momento.

Nas próximas seções veremos como funciona e como foi implementado o algoritmo de Deslocamento para a multiplicação do vetor v por cada tipo de fator normal.

Multiplicação do Último Fator Normal

A multiplicação do vetor v pelo último fator normal é dada por:

$$v \times I_{nleft_N} \otimes Q^{(N)}$$

Sabe-se que o produto tensorial de uma matriz identidade por uma matriz M qualquer sempre resulta em uma matriz de blocos diagonais, onde cada bloco é a própria matriz M. Logo, a matriz resultante do produto tensorial $I_{nleft_N} \otimes Q^{(N)}$ é uma matriz de $nleft_N$ blocos diagonais, onde cada bloco é simplesmente a matriz $Q^{(N)}$. Dessa forma, podemos dividir o vetor v em $nleft_N$ vetores (chamados de z_{in}) com n_N elementos e multiplicar cada um destes com a matriz $Q^{(N)}$ que corresponde ao bloco diagonal da matriz resultante. A Figura 2.4 ilustra como seria este procedimento.

O Algoritmo 2.1 mostra os passos da multiplicação $v \times I_{nleft_N} \otimes Q^{(N)}$. Este algoritmo possui um *loop* principal (linha 2) que realiza cada multiplicação do vetor z_{in} pela matriz $Q^{(N)}$. Internamente, existem ainda dois outros laços (linha 4 e 10) que servem para gerar o vetor z_{in} extraindo

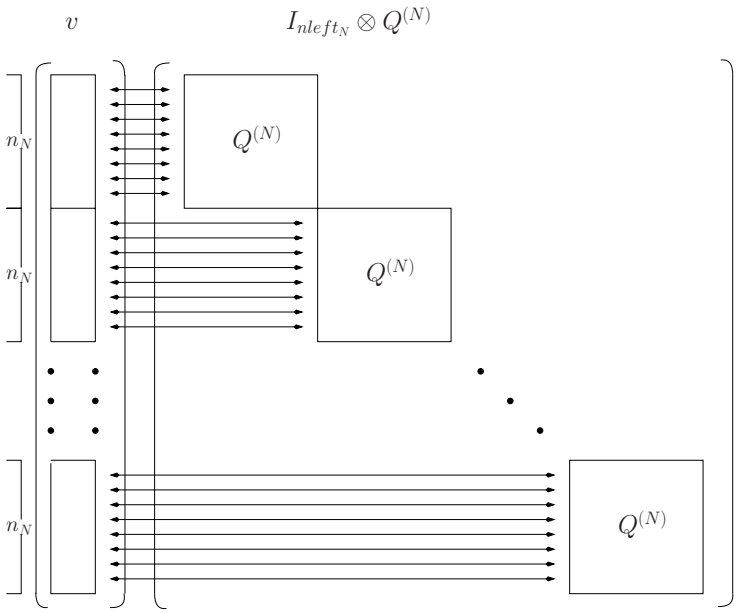


Figura 2.4: Multiplicação $v \times I_{nleft_N} \otimes Q^{(N)}$

valores do vetor v, como mostra a Figura 2.5, e para acumular os resultados da multiplicação no vetor v. Na linha 8 ocorre a multiplicação propriamente dita do vetor z_{in} pela matriz $Q^{(N)}$ e o resultado armazenado em outro vetor denominado z_{out} de mesmo tamanho do vetor z_{in}. Note que todo o procedimento é realizado apenas com a utilização do vetor v e da matriz $Q^{(N)}$, não sendo necessário gerar a matriz $I_{nleft_N} \otimes Q^{(N)}$ em nenhum momento.

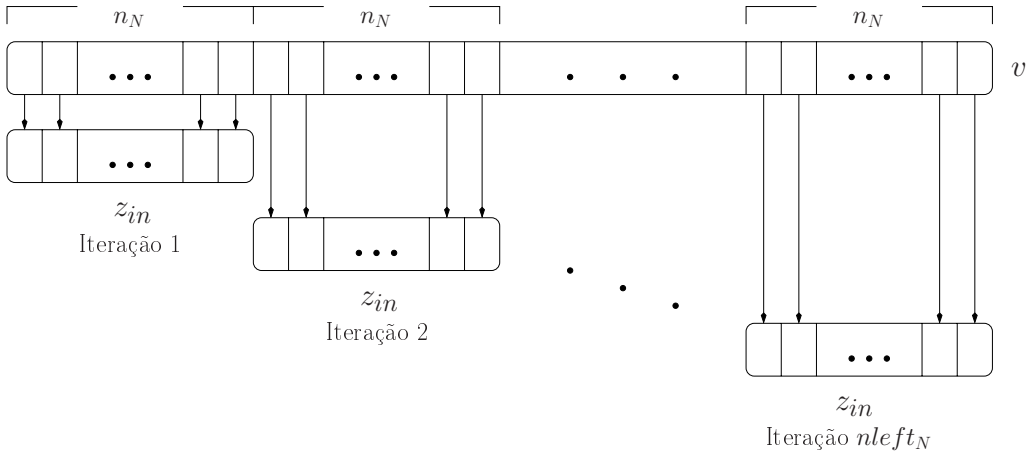


Figura 2.5: Algoritmo de multiplicação do último fator normal

Multiplicação do Primeiro Fator Normal

A multiplicação do vetor v pelo primeiro fator normal é dada por:

Algoritmo 2.1

```
1  base = 0;
2  for k = 1, 2, ..., nleft_N
3  do  index = base + 1;
4      for l = 1, 2, ..., n_N
5      do  z_in[l] = v[index];
6          index = index + 1;
7      end do
8      multiply   z_out = z_in × Q^(N);
9      index = base + 1;
10     for l = 1, 2, ..., n_N
11     do  v[index] = z_out[l];
12         index = index + 1;
13     end do
14     base = base + n_N;
15 end do
```

Algoritmo 2.1: Multiplicação $v \times I_{nleft_N} \otimes Q^{(N)}$

$$v \times Q^{(1)} \otimes I_{nright_1}$$

A multiplicação do vetor v pelo primeiro fator normal é semelhante a do último fator normal, a única diferença está no processo de extração dos valores do vetor v para gerar cada vetor z_{in}. Essa diferença se dá pelo fato da matriz $Q^{(1)} \otimes I_{nright_1}$ possuir uma forma distinta em relação a matriz envolvida na outra multiplicação, veja abaixo:

$$Q^{(1)} \otimes I_{nright_1} = \begin{pmatrix} q^{(1)}_{1,1} I_{nright_1} & q^{(1)}_{1,2} I_{nright_1} & \cdots & q^{(1)}_{1,n_1} I_{nright_1} \\ q^{(1)}_{2,1} I_{nright_1} & q^{(1)}_{2,2} I_{nright_1} & \cdots & q^{(1)}_{2,n_1} I_{nright_1} \\ \vdots & \vdots & \ddots & \vdots \\ q^{(1)}_{n_1,1} I_{nright_1} & q^{(1)}_{n_1 1,2} I_{nright_1} & \cdots & q^{(1)}_{n_1 1,n_1} I_{nright_1} \end{pmatrix}$$

Dessa forma, para gerar o vetor z_{in} não basta extrair n_N elementos consecutivos do vetor v, é preciso extrair os elementos alternadamente, como mostra a Figura 2.6. Isto é, tem-se que buscar um elemento a cada $nright_1$ elementos do vetor v.

O algoritmo 2.2 apresenta a multiplicação do vetor v pela matriz $Q^{(1)} \otimes I_{nright_1}$. Este algoritmo forma $nright_1$ vetores z_{in} buscando os elementos alternadamente no vetor v (linhas 4-7). Realiza a multiplicação do vetor z_{in} pela matriz $Q^{(1)}$ (linha 8) e acumula o resultado no vetor auxiliar z_{out}. Este vetor então é atribuído ao vetor v (linhas 10-13) que ao final conterá o resultado de toda a operação.

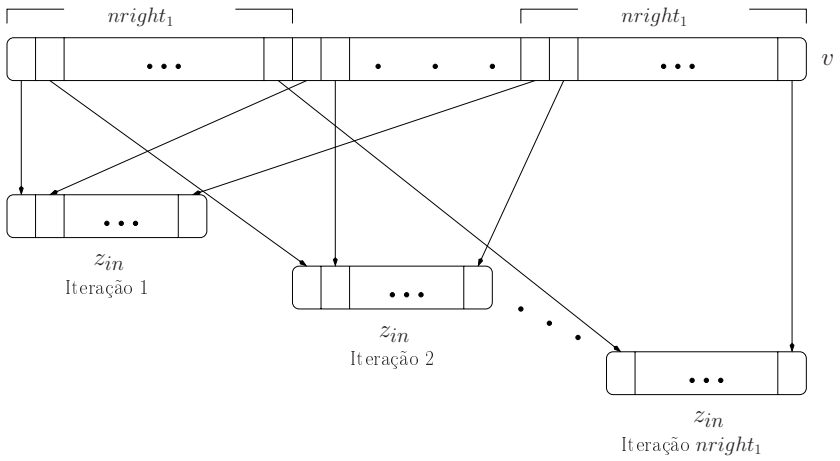


Figura 2.6: Execução da multiplicação do primeiro fator normal

Multiplicação Completa

A multiplicação do vetor v pelos demais fatores normais (do segundo fator normal ao penúltimo fator normal) é realizada com a combinação das duas outras. A técnica básica consiste em aplicar a propriedade da pseudo-comutatividade do fator normal:

$$\begin{aligned} I_{nleft_i} \otimes Q^{(i)} \otimes I_{nright_i} = P_\sigma \times \left(I_{nleft_i} \otimes I_{nright_i} \otimes Q^{(i)} \right) {P_\sigma}^T \\ \text{onde } \sigma \text{ é a permutação que vai de} \{1, \ldots, i-1, i, i+1, \ldots, N\} \\ \text{à } \{1, \ldots, i-1, i+1, \ldots, N, i\}. \end{aligned} \tag{2.3}$$

Isto nos leva a sempre multiplicar os fatores normais, inclusive o primeiro e o último, da seguinte maneira:

$$I_{nleft_i} \otimes I_{nright_i} \otimes Q^{(i)} = I_{\bar{n}_i} \otimes Q^{(i)}$$

As multiplicações são realizadas sempre de acordo com a posição da matriz $Q^{(i)}$. O Algoritmo 2.3 resume os passos implementados para multiplicar um vetor v por um produto tensorial $\otimes_{i=1}^{N} Q^{(i)}$. Neste algoritmo os fatores normais são tratados do primeiro ao último. Entretanto, de acordo com a propriedade da comutatividade dos fatores normais, outra ordem pode também ser aplicada. Note que as multiplicações pelo primeiro e pelo último fator normal também são cobertos por este algoritmo, logo os algoritmos 2.1 e 2.2 não precisam serem implementados, são usados apenas para o entendimento.

O Algoritmo 2.3 forma $nleft_i \times nright_i$ vetores z_{in} buscando os elementos nas posições corretas do vetor v (linhas 6-9), conforme o $nright_i$. Em seguida, realiza a multiplicação do vetor z_{in} pela matriz $Q^{(i)}$ (linha 10) e acumula o resultado no vetor auxiliar z_{out}. Este vetor então é atribuído, de forma análoga a extração, ao vetor v (linhas 12-15) que ao final terá o resultado de toda a multiplicação.

Algoritmo 2.2

```
1   base = 0;
2   for j = 1, 2, ..., nright_1
3   do  index = base + j;
4       for l = 1, 2, ..., n_1
5       do  z_in[l] = v[index];
6           index = index + nright_1;
7       end do
8       multiply    z_out = z_in × Q^(1);
9       index = base + j;
10      for l = 1, 2, ..., n_1
11      do  v[index] = z_out[l];
12          index = index + nright_1;
13      end do
14  end do
```

Algoritmo 2.2: Multiplicação $v \times Q^{(1)} \otimes I_{nright_1}$

2.4.2 Etapas da Multiplicação Vetor-Descritor

Esta seção apresenta através de um exemplo todas as etapas que constituem o procedimento de multiplicação vetor-descritor.

Considere um modelo SAN definido por três autômatos: $A^{(1)}$, $A^{(2)}$ e $A^{(3)}$.

Multiplicação da Parte Local

A multiplicação do vetor v pela parte local do descritor passa pelos seguintes termos, sendo $Q_l^{(i)}$ o descritor correspondente à descrição das transições locais de cada autômato i:

$$Q_l^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3} \qquad I_{n_1} \underset{g}{\otimes} Q_l^{(2)} \underset{g}{\otimes} I_{n_3} \qquad I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_l^{(3)}$$

Cada termo é multiplicado pelo vetor v previamente inicializado. O resultado de cada multiplicação é então acumulado em um vetor auxiliar w. Após realizar as multiplicações com cada um dos três termos, o valor resultante acumulado no vetor w é então atribuído novamente às posições de v. O que ocorre, na realidade, resume-se nas seguintes operações:

$v \times (Q_l^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor w

$v \times (I_{n_1} \underset{g}{\otimes} Q_l^{(2)} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado acumulado no vetor w

$v \times (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_l^{(3)}) \rightarrow$ Resultado acumulado no vetor w

Algoritmo 2.3

```
1  for i = 1, 2, ..., N
2  do base = 0;
3      for k = 1, 2, ..., nleft_i
4      do for j = 1, 2, ..., nright_i
5          do index = base + j;
6              for l = 1, 2, ..., n_i
7              do z_in[l] = v[index];
8                  index = index + nright_i;
9              end do
10             multiply    z_out = z_in × Q^(i);
11             index = base + j;
12             for l = 1, 2, ..., n_i
13             do v[index] = z_out[l];
14                 index = index + nright_i;
15             end do
16         end do
17         base = base + ( nright_i × n_i);
18     end do
19 end do
```

Algoritmo 2.3: Multiplicação $v \times \otimes_{i=1}^{N} Q^{(i)}$

Multiplicação da Parte Sincronizante

Deve-se multiplicar o vetor v pelo descritor de cada evento sincronizante e_i. Supondo que a SAN possua três eventos sincronizantes (e_1, e_2 e e_3), as seguintes operações são necessárias:

$v \times (Q_{e_1}^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor x

$x \times (I_{n_1} \underset{g}{\otimes} Q_{e_1}^{(2)} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor y

$y \times (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_{e_1}^{(3)}) \rightarrow$ Resultado armazenado no vetor z

Resultados em z são então acumulados no vetor w

$v \times (Q_{e_2}^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor x

$x \times (I_{n_1} \underset{g}{\otimes} Q_{e_2}^{(2)} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor y

$y \times (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_{e_2}^{(3)}) \rightarrow$ Resultado armazenado no vetor z

Resultados em z são então acumulados no vetor w

$v \times (Q_{e_3}^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor x

$x \times (I_{n_1} \underset{g}{\otimes} Q_{e_3}^{(2)} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado no vetor y

$y \times (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_{e_3}^{(3)}) \rightarrow$ Resultado acumulado no vetor z

Resultados em z são então acumulados no vetor w

É importante salientar que a multiplicação pela parte sincronizante deve ser realizada tanto para a parte positiva quanto para a parte negativa, portanto o procedimento acima deve ser realizado duas vezes.

Ao final, o vetor auxiliar w contém o resultado da multiplicação do vetor v pelo descritor Markoviano do modelo, mostrado de forma geral na Tabela 2.3, ou seja:

$$w = v\,[(\bigoplus_{i=1}^{n}{}_g\, Q_l^{(i)}) + (\bigotimes_{j=1}^{E}{}_g\, Q_{e_j^+}^{(i)}) + (\bigotimes_{j=1}^{E}{}_g\, Q_{e_j^-}^{(i)})] =$$

$$v\,Q = \sum_{j=1}^{(N+2E)} \bigotimes_{i=1}^{N}{}_g\, Q_j^{(i)}$$

2.4.3 Custo Computacional

O custo computacional do algoritmo *Shuffle*, segundo seus criadores[8], para realizar o produto de um vetor por um termo tensorial é obtido pelo número de multiplicações vetor-matriz executadas (linha 10 do Algoritmo 2.3). Para cada iteração i do algoritmo são executadas $nleft_i \times nright_i$ multiplicações vetor-matriz com matrizes de tamanho n_i. Supondo que as matrizes $Q^{(i)}$ não possuem elementos nulos, o número de multiplicações para cada produto vetor-matriz é igual a $(n_i)^2$. Lembrando que $\bar{n}_i \times n_i = nleft_i \times nright_i \times n_i = \prod_{i=1}^{N} n_i$, o custo computacional do Algoritmo *Shuffle* é definido por:

$$\sum_{i=1}^{N} \left(\bar{n}_i \times (n_i)^2\right) = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} n_i \tag{2.4}$$

Porém, normalmente as matrizes $Q^{(i)}$ possuem um número considerável de elementos nulos, os quais são desconsiderados pelo algoritmo. Logo, o número de multiplicações para cada produto vetor-matriz será, freqüentemente, inferior a $(n_i)^2$. Assim, sendo nz_i o número de elementos não nulos de uma dada matriz $Q^{(i)}$, o custo computacional apresentado neste algoritmo é então:

$$\sum_{i=1}^{N} (\bar{n}_i \times nz_i) = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \tag{2.5}$$

Na prática, considerando que para obter as soluções estacionárias dos modelos SAN é necessário multiplicar um vetor de probabilidades pelo Descritor Markoviano completo do modelo, tem-se um custo computacional dado por:

$$\sum_{k=1}^{1+2E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i^{(k)}}{n_i} \right) \tag{2.6}$$

O capítulo seguinte apresenta uma descrição detalhada da técnica *Slice*, bem como os principais trechos de seu algoritmo.

Capítulo 3

Técnica de Fatiamento ou *Slice*

O principal problema para a resolução de Redes de Autômatos Estocásticos (SAN) vem sendo ao longo do tempo a explosão do espaço de estados. Isso se deve ao fato de que para resolver uma SAN é necessário realizar a multiplicação de um vetor de probabilidades pelo gerador infinitesimal, operação que normalmente envolve uma grande quantidade de dados, exigindo um alto processamento e uso de memória. Com o intuito de minimizar a utilização de recursos computacionais, diferentes técnicas vêm sendo estudadas, entre elas destacam-se a técnica *Shuffle*[1] [2, 6, 8] e a técnica *Slice* [9].

A técnica *Slice* é uma técnica recentemente descoberta que implementa a resolução de Redes de Autômatos Estocásticos (SAN). A operação fundamental realizada nesta técnica, assim como na maioria das outras técnicas semelhantes, é a multiplicação de um vetor de probabilidades v pelo gerador infinitesimal Q. Esta multiplicação é dada genericamente pela seguinte expressão, sendo N o número de autômatos na rede, e E o número de eventos sincronizantes:

$$vQ = v \sum_{j=1}^{(N+2E)} \left[\bigotimes_{i=1}^{N}{}_{g} Q_j^{(i)} \right] = \sum_{j=1}^{(N+2E)} \left[v \bigotimes_{i=1}^{N}{}_{g} Q_j^{(i)} \right]$$

Em especial, serão tratadas as particularidades envolvidas na execução e na implementação da operação de multiplicação apresentada a seguir, visto que esta é a operação fundamental para a resolução de modelos SAN.

$$v \bigotimes_{i=1}^{N}{}_{g} Q_j^{(i)}$$

Todas as definições estabelecidas na seção 2.4 do capítulo 2 sobre seqüências finitas de matrizes serão da mesma forma utilizadas neste capítulo na definição do algoritmo da técnica *Slice*.

[1] O algoritmo *Shuffle* foi desenvolvido por Fernandes, Plateau e Stewart [8]. Apesar de não ter recebido inicialmente esta denominação, o mesmo vem sendo recentemente denominado como tal [3].

Para realizar a multiplicação vetor-descritor, a técnica *Slice*, utilizando-se de propriedades da álgebra tensorial, introduz uma nova forma de decompor esta operação, chamada de *Decomposição Aditiva em Fatores Normais Unitários Aditivos*, a qual é abordada nas seções subseqüentes.

3.1 Fatores Normais Unitários Aditivos

Fatores Normais Unitários Aditivos é uma nova maneira de decompor um produto tensorial qualquer. O objetivo é reduzir as operações necessárias para executar a multiplicação de um dado vetor $\pi^{(i)}$ por um produto tensorial entre matrizes.

Os fatores normais unitários aditivos levam em consideração as primeiras $(N-1)$ matrizes componentes do produto tensorial, onde N é o número total de matrizes envolvidas na operação. Cada fator normal unitário aditivo (AUNF) ρ_i é constituído pelo produto tensorial da matriz resultante do produto tensorial entre as primeiras $(N-1)$ matrizes contendo apenas um elemento não-nulo por vez, pela última matriz N.

Por exemplo, o produto tensorial $A \otimes B \otimes C$, onde as matrizes têm ordem n=2, teria a formação dos seguintes fatores normais unitários aditivos:

1° Fator Normal Unitário Aditivo (ρ_1)

$$\left(\begin{array}{cc|cc} a_{11}\,b_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \otimes \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

2° Fator Normal Unitário Aditivo (ρ_2)

$$\left(\begin{array}{cc|cc} 0 & a_{11}\,b_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \otimes \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

3° Fator Normal Unitário Aditivo (ρ_3)

$$\left(\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ a_{11}\,b_{21} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \otimes \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

...

16° Fator Normal Unitário Aditivo (ρ_{16})

$$\left(\begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{22}\,b_{22} \end{array}\right) \otimes \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Esta decomposição é chamada de *Decomposição Aditiva em Fatores Normais Unitários Aditivos*, uma vez que os termos são somados. Ao multiplicarmos cada AUNF ρ_i pelo vetor de probabilidades π, cada resultado é somado em um vetor acumulador Π, que ao final das multiplicações conterá a solução do modelo. Segundo Webber [18], para diversos modelos tem-se um ganho significativo em termos do número de multiplicações realizadas.

3.2 Etapas da Multiplicação Vetor-Descritor

Essa seção apresenta as etapas que constituem a multiplicação vetor-descritor utilizando a decomposição em fatores normais unitários aditivos.

A multiplicação vetor-descritor é composta pela multiplicação do vetor de probabilidades pela parte local e pela parte sincronizante do descritor Markoviano, como veremos a seguir.

3.2.1 Multiplicação da Parte Local

Considerando o exemplo anterior, a multiplicação do vetor v pela parte local do descritor passa pelos seguintes termos, sendo $Q_l^{(i)}$ o descritor correspondente à descrição das transições locais de cada autômato i:

$$Q_l^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3} \qquad I_{n_1} \underset{g}{\otimes} Q_l^{(2)} \underset{g}{\otimes} I_{n_3} \qquad I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_l^{(3)}$$

Cada termo é multiplicado pelo vetor v previamente inicializado. O resultado de cada multiplicação é então acumulado em um vetor auxiliar w. Após realizar as multiplicações com cada um dos três termos, o valor resultante acumulado no vetor w é então atribuído novamente às posições de v. O que ocorre, na realidade, resume-se nas seguintes operações:

$v \times (Q_l^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado armazenado em w

$v \times (I_{n_1} \underset{g}{\otimes} Q_l^{(2)} \underset{g}{\otimes} I_{n_3}) \rightarrow$ Resultado acumulado em w

$v \times (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} Q_l^{(3)}) \rightarrow$ Resultado acumulado em w

Na prática, o algoritmo da técnica *Slice* realiza a multiplicação da parte local do descritor *fatiando* cada um destes termos, como detalhado a seguir.

A multiplicação do vetor de probabilidades pelo gerador infinitesimal é dividida em duas partes da mesma forma como o descritor Markoviano, conforme exposto na seção 2.3 do capítulo 2. A primeira parte corresponde a multiplicação do vetor de probabilidades pela parte local do modelo SAN e a segunda a multiplicação do mesmo vetor de probabilidades pela parte sincronizante do modelo. Ao final, soma-se os vetores resultantes de cada uma destas operações encontrando-se então o vetor com o resultado final da multiplicação do vetor de probabilidades pelo descritor Markoviano.

A Tabela 3.1 apresenta a parte local do descritor Markoviano, a qual contém apenas as matrizes de transição dos eventos locais do modelo SAN. Observa-se nesta tabela que a parte local do

gerador infinitesimal é formado pelo somatório de N produtos tensoriais de N matrizes. Sendo assim, a multiplicação do vetor de probabilidades pela parte local do gerador infinitesimal pode ser dividida em N partes, onde cada uma corresponde a multiplicação do vetor de probabilidades pelo produto tensorial apresentado em cada linha da Tabela 3.1.

$\sum$	N	$Q_l^{(1)}$	$\underset{g}{\otimes}$	I_{n_2}	$\underset{g}{\otimes}$	$\cdots$	$\underset{g}{\otimes}$	$I_{n_{N-1}}$	$\underset{g}{\otimes}$	I_{n_N}
		I_{n_1}	$\underset{g}{\otimes}$	$Q_l^{(2)}$	$\underset{g}{\otimes}$	$\cdots$	$\underset{g}{\otimes}$	$I_{n_{N-1}}$	$\underset{g}{\otimes}$	I_{n_N}
						$\vdots$				
		I_{n_1}	$\underset{g}{\otimes}$	I_{n_2}	$\underset{g}{\otimes}$	$\cdots$	$\underset{g}{\otimes}$	$Q_l^{(N-1)}$	$\underset{g}{\otimes}$	I_{n_N}
		I_{n_1}	$\underset{g}{\otimes}$	I_{n_2}	$\underset{g}{\otimes}$	$\cdots$	$\underset{g}{\otimes}$	$I_{n_{N-1}}$	$\underset{g}{\otimes}$	$Q_l^{(N)}$

Tabela 3.1: Parte Local do Descritor Markoviano

Percebe-se ainda que cada produto tensorial é composto por *N-1* matrizes identidades e apenas uma matriz $Q_l^{(i)}$, a qual pode conter qualquer valor em seus elementos, onde i é o índice do autômato no modelo SAN. Sabe-se que em um produto tensorial, as matrizes identidades não modificam os elementos das demais matrizes, apenas replicam e reposicionam estes valores na matriz resultante. Dessa forma, a matriz resultante do produto tensorial de cada linha da Tabela 3.1 será sempre uma matriz contendo os mesmos elementos da matriz $Q_l^{(i)}$, porém replicados e reposicionados. Portanto, a operação de multiplicação do vetor de probabilidades por cada um destes produtos tensoriais resume-se em descobrir onde cada um dos elementos não-nulos da matriz $Q_l^{(i)}$ seria posicionado na matriz resultante, pois assim basta multiplicar estes valores pelos elementos correspondentes no vetor de probabilidades.

Na prática, o algoritmo da técnica *Slice* descobre apenas onde apareceria a primeira ocorrência de cada valor não-nulo da matriz $Q_l^{(i)}$ na matriz resultante do produto tensorial. Sabendo-se onde estaria posicionada a primeira ocorrência do elemento e considerando a posição da matriz $Q_l^{(i)}$ no produto tensorial, é possível descobrir quais valores do vetor de probabilidades devem ser multiplicados por este elemento não-nulo. O Algoritmo 3.1 mostra como é realizada a multiplicação de um valor não-nulo da matriz $Q_l^{(i)}$ pelos elementos correspondentes no vetor de probabilidades, onde *bi* e *bj* são inicializados respectivamente com a linha e a coluna da primeira ocorrência de uma elemento não-nulo da matriz $Q_l^{(i)}$ na matriz resultante, *nv* é o tamanho do vetor de probabilidades, *ni* a dimensão da matriz $Q_l^{(i)}$, *nRight* é o produto das dimensões de todas as matrizes à direita da matriz $Q_l^{(i)}$, *vProb* o vetor de probabilidades, *dElem* o elemento não-nulo da matriz $Q_l^{(i)}$ e *vRes* o vetor acumulador resultante. É importante salientar que em momento algum os produtos tensoriais entre as matrizes identidades e as matrizes $Q_l^{(i)}$ são realizados, reduzindo assim consideravelmente o custo computacional de toda operação.

Para realizar a multiplicação do vetor de probabilidades por cada um dos N produtos tensoriais descritos na Tabela 3.1, o algoritmo da técnica *Slice* executa nz_i vezes o código apresentado no Algoritmo 3.1, onde nz_i é o número de elementos não-nulos da matriz $Q_l^{(i)}$. Portanto, para

Algoritmo 3.1

```
1   while   (bi < nv) and (bj < nv)
2   do  i = bi;
3       j = bj;
4       for n = 0, 1, ..., nRight − 1
5       do  vRes[j] = vRes[j] + vProb[i] ∗ dElem;
6           i = i + 1;
7           j = j + 1;
8       end do
9       bi = bi + nRight ∗ ni;
10      bj = bj + nRight ∗ ni;
11  end do
```

Algoritmo 3.1: Multiplicação de um elemento não-nulo de uma matriz $Q_l^{(i)}$ pelo vetor de probabilidades

realizar a multiplicação completa do vetor de probabilidades pela parte local do descritor Markoviano, basta executar repetidas vezes o código apresentado no Algoritmo 3.1.

3.2.2 Multiplicação da Parte Sincronizante

É na multiplicação do vetor de probabilidades pela parte sincronizante do descritor Markoviano que se encontra a principal diferença entre a técnica *Slice* e as demais, vejamos um exemplo.

Considerando três matrizes de ordem $n_i = 2$ e um vetor de probabilidades $\pi^{(0)}$ de tamanho $n = 8$ $(n_1 \times n_2 \times n_3)$, tem-se para este exemplo a definição de $({n_1}^2 \times {n_2}^2)$ fatores normais unitários aditivos, um para cada elemento não-nulo do produto tensorial $A \otimes B$. Vejamos então as etapas da multiplicação vetor-descritor para o termo $A \otimes B \otimes C$ supondo as seguintes matrizes:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \otimes \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \otimes \begin{pmatrix} 9 & 10 \\ 11 & 12 \end{pmatrix}$$

- Multiplicação de $\pi^{(0)}$ pelo 1° Fator Normal Unitário Aditivo (ρ_1)

$$\pi^{(0)} \times \left[\left(\begin{array}{cc|cc} 1\times 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right) \otimes \begin{pmatrix} 9 & 10 \\ 11 & 12 \end{pmatrix} \right]$$

Este AUNF é decomposto em dois fatores normais tradicionais:

$$\pi^{(0)} \times \left[\left(\begin{array}{cc|cc} 1\times 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right) \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right] = \pi^{(1)}$$

O vetor $\pi^{(1)}$ terá dois elementos não-nulos (ordem n_N da última matriz).

$$\pi^{(1)} \times [\left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right) \otimes \left(\begin{array}{cc} 9 & 10 \\ 11 & 12 \end{array}\right)] = \pi^{(2)}$$

O resultado em $\pi^{(2)}$ é então acumulado em um vetor acumulador Π.

- Multiplicação de $\pi^{(0)}$ pelo 2° Fator Normal Unitário Aditivo (ρ_2)

$$\pi^{(0)} \times [\left(\begin{array}{cc|cc} 0 & 1\times 6 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \otimes \left(\begin{array}{cc} 9 & 10 \\ 11 & 12 \end{array}\right)]$$

Novamente o AUNF é decomposto em dois fatores normais tradicionais:

$$\pi^{(0)} \times [\left(\begin{array}{cc|cc} 0 & 1\times 6 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \otimes \left(\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right)] = \pi^{(1)}$$

O vetor $\pi^{(1)}$ terá dois elementos não-nulos (ordem n_N da última matriz).

$$\pi^{(1)} \times [\left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right) \otimes \left(\begin{array}{cc} 9 & 10 \\ 11 & 12 \end{array}\right)] = \pi^{(2)}$$

O resultado em $\pi^{(2)}$ é então acumulado em um vetor acumulador Π.

- E assim por diante, chegando neste exemplo em um total de 16 fatores normais unitários aditivos. Todos são multiplicados pelo vetor $\pi^{(0)}$ seguindo as mesmas etapas demonstradas acima. Ao final de todas essas etapas, o vetor de probabilidades Π conterá a solução do produto tensorial.

Temos então basicamente duas operações que devem ser realizadas para cada AUNF com o intuito de obter o vetor de probabilidades Π. Essas operações são a multiplicação do vetor $\pi^{(0)}$ pelo primeiro termo do AUNF e a multiplicação do vetor $\pi^{(1)}$ pelo segundo termo deste mesmo AUNF.

Para entender como o algoritmo da técnica *Slice* realiza a multiplicação do vetor de probabilidades pela parte sincronizante do descritor Markoviano, vejamos na Tabela 3.2 a parte sincronizante do gerador infinitesimal, que corresponde apenas as matrizes de transição dos eventos sincronizantes do modelo SAN.

$\sum$	$2E$	e^+	$Q_{1^+}^{(1)} \underset{g}{\otimes} Q_{1^+}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{1^+}^{(N-1)} \underset{g}{\otimes} Q_{1^+}^{(N)}$ $\vdots$ $Q_{E^+}^{(1)} \underset{g}{\otimes} Q_{E^+}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{E^+}^{(N-1)} \underset{g}{\otimes} Q_{E^+}^{(N)}$
		e^-	$Q_{1^-}^{(1)} \underset{g}{\otimes} Q_{1^-}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{1^-}^{(N-1)} \underset{g}{\otimes} Q_{1^-}^{(N)}$ $\vdots$ $Q_{E^-}^{(1)} \underset{g}{\otimes} Q_{E^-}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{E^-}^{(N-1)} \underset{g}{\otimes} Q_{E^-}^{(N)}$

Tabela 3.2: Parte Sincronizante do Descritor Markoviano

A parte sincronizante do descritor Markoviano é dividida em duas partes, uma correspondendo as matrizes positivas dos eventos sincronizantes e a outra referente as matrizes negativas destes mesmos eventos. Observando-se a Tabela 3.2, nota-se que a parte das matrizes sincronizantes positiva é exatamente igual a correspondente as matrizes negativas, podendo se diferenciar apenas no conteúdo das matrizes. Por este motivo, são apresentados a seguir apenas os detalhes do algoritmo da técnica *Slice* referentes a multiplicação do vetor de probabilidades pela parte positiva do gerador infinitesimal, uma vez que a operação envolvendo a parte negativa dos eventos sincronizantes é realizada da mesma maneira.

Observando novamente a Tabela 3.2, percebe-se que diferentemente da parte local, a parte sincronizante do descritor Markoviano é composto por produtos tensoriais de matrizes que não são necessariamente identidades, o que torna o cálculo da multiplicação um pouco mais complicado.

Assim como na multiplicação pela parte local do gerador infinitesimal, é possível modularizar o cálculo da parte sincronizante em multiplicações mais simples. Estas correspondem as multiplicações do vetor de probabilidades por cada produto tensorial das N matrizes de cada evento sincronizante do modelo SAN, representados por cada linha da Tabela 3.2. Além disso, como apresentado na seção 3.1, o produto tensorial entre as matrizes sincronizantes é decomposto em diversos Fatores Normais Unitários Aditivos, que por sua vez são decompostos em Fatores Normais Tradicionais, tornando o cálculo ainda mais modularizado. Dessa forma, a multiplicação

do vetor de probabilidades pela parte sincronizante do descritor Markoviano resume-se a um somatório do resultado da multiplicação do vetor de probabilidades pelo primeiro Fator Normal Tradicional do AUNF, multiplicado pelo segundo Fator Normal Tradicional do mesmo AUNF.

O Algoritmo 3.2 apresenta em linhas gerais os passos implementados para realizar a multiplicação do vetor de probabilidades $vProb$ pela parte sincronizante do descritor Markoviano.

Algoritmo 3.2

```
1  for e = 0, 1, ..., nE − 1
2  do while  AUNF = nextAUNF()
3     do  vAux = vProb × AUNF_1;
4         vRes = vRes + vAux × AUNF_2;
5     end do
6  end do
```

Algoritmo 3.2: Multiplicação do vetor de probabilidades pela parte sincronizante do descritor Markoviano

O Algoritmo 3.2 percorre todos os eventos sincronizantes através do laço na linha 1, onde nE é o total de eventos sincronizantes do modelo SAN. Para cada evento, um novo laço (linha 2) percorre todos os Fatores Normais Unitários Aditivos que compõem o produto tensorial entre as matrizes sincronizantes deste evento. Enfim, para cada AUNF multiplica-se o vetor de probabilidades $vProb$ pelo primeiro Fator Normal Tradicional do AUNF, e em seguida, o resultado armazenado em $vAux$ é multiplicado pelo segundo Fator Normal Tradicional do AUNF e o resultado é acumulado no vetor $vRes$, que ao final conterá o vetor resultado de toda a operação.

Para descobrir cada um dos Fatores Normais Unitários Aditivos dos eventos sincronizantes de um modelo SAN, basta na verdade descobrir a linha, a coluna e o valor de cada um dos elementos da matriz resultante do produto tensorial das $N-1$ primeiras matrizes sincronizantes do evento. O Algoritmo 3.3 apresenta o código necessário para descobrir cada um destes valores com suas respectivas linhas e colunas nesta matriz resultante, sem realizar de fato o produto tensorial entre as matrizes.

O Algoritmo 3.3 mostra como foi implementada a rotina que descobre o elemento não-nulo de cada AUNF. Primeiramente, verifica-se se é o primeiro AUNF para calcular (linha 1), se for, é necessário percorrer todas as matrizes para pegar um elemento de cada para calcular o elemento não-nulo do AUNF. A linha 6 apresenta o laço que percorre as matrizes para pegar seus elementos. Para otimizar este processo, as multiplicações entre os elementos das matrizes a medida que estão sendo realizadas são armazenadas (linha 10) em um vetor $vFator$, com o intuito de serem reutilizadas quando necessário, conforme descrito em [4]. Dessa forma, não é necessário percorrer todas as matrizes sempre que o elemento não-nulo de um AUNF for calculado, nem mesmo realizar todas as multiplicações novamente. A função *nextElem* (linha 7) retorna um elemento não-nulo de uma determinada matriz, ou zero caso já tenha retornado todos. Se todos os elementos de uma matriz já foram utilizados, o algoritmo verifica se esta é a primeira matriz (linha 13), caso seja, retorna zero sinalizando que não existe mais AUNF. Caso não seja a primeira matriz, o algoritmo retorna para a matriz anterior (linha 16) para pegar o próximo elemento desta matriz e então prossegue o cálculo normalmente.

Algoritmo 3.3

```
1  if   Reset
2       nCurAut = 0;
3  else
4       nCurAut = nLastAut − 1;
5  end if
6  for n = nCurAut, ..., nLastAut − 1
7  do  vFator[n] = nextElem(nCurAut);
8       if   vFator[n]! = 0
9            if   n > 0
10                vFator[n] = vFator[n] × vFator[n − 1];
11           end if
12      else
13           if   n = 0
14           return0;
15           else
16           n = n − 2;
17           end if
18      end if
19 end do
20 returnvFator[nCurAut];
```

Algoritmo 3.3: Calcula o elemento não-nulo de cada AUNF

Nota-se que o Algoritmo 3.3 não apresenta de que forma a linha e a coluna do elemento são descobertas, isto porque estas são facilmente obtidas através do uso de uma classe que transforma um índice composto (que contém o índice de cada matriz envolvida na operação) no índice da matriz resultante. Por exemplo, considerando três matrizes de ordem 2, o índice "010", que corresponde ao índice "0"na primeira matriz, "1"na segunda e "0"na terceira, equivaleria ao índice "2"na matriz resultante do produto tensorial destas matrizes. Sendo assim, o algoritmo apenas atualiza o índice composto com as linhas e as colunas dos elementos de cada uma das matrizes envolvidas no cálculo.

Uma vez calculado o elemento não-nulo do AUNF, realiza-se a multiplicação do vetor de probabilidades *vProb* pelo primeiro Fator Normal Tradicional do AUNF, como mostra a linha 3 do Algoritmo 3.2. A implementação desta operação é apresentada pelo Algoritmo 3.4.

Conforme apresentado na seção 3.1, o primeiro Fator Normal Tradicional de um AUNF é composto pelo produto tensorial de uma matriz unitária, que contém o elemento não-nulo calculado pelo Algoritmo 3.3, por uma matriz identidade de ordem n_N, onde n_N é a ordem da matriz do último evento sincronizante do modelo. Sendo assim, basta multiplicar n_N elementos consecutivos do vetor de probabilidades pelo elemento não-nulo do AUNF, conforme mostra o Algoritmo 3.4, onde i e j são respectivamente o índice da linha e da coluna do elemento não-nulo da matriz unitária. Ao final desta operação, o vetor resultante $vAux$ conterá n_N elementos não-nulos.

Algoritmo 3.4

1 **for** $n = 0, 1, \ldots, n_N - 1$
2 **do** $vAux[j] = vProb[i] \times dFator$;
3 $i = i + 1$;
4 $j = j + 1$;
5 **end do**

Algoritmo 3.4: Multiplicação do vetor de probabilidades pelo primeiro Fator Normal Tradicional do AUNF

Por fim, a última operação que compõe a multiplicação do vetor de probabilidades pela parte sincronizante do gerador infinitesimal é a multiplicação do vetor resultante $vAux$ da operação anterior pelo segundo Fator Normal Tradicional do AUNF, conforme a linha 4 do Algoritmo 3.2.

O segundo Fator Normal Tradicional de cada AUNF, conforme a seção 3.1, é composto pelo produto tensorial entre uma matriz identidade de ordem $nleft_N$ e a última matriz, onde $nleft_N$ é o produto das dimensões de todas as matrizes à esquerda da última matriz do cálculo. Sabe-se que o produto tensorial de uma matriz identidade por uma matriz M qualquer sempre resulta em uma matriz de blocos diagonais, onde cada bloco é a própria matriz M. Logo, a matriz resultante do produto tensorial $I_{nleft_N} \otimes Q^{(N)}$ é uma matriz de $nleft_N$ blocos diagonais, onde cada bloco é exatamente a matriz $Q^{(N)}$. Considerando que o vetor $vAux$ possui apenas n_N elementos não-nulos e que estes elementos estão posicionados de forma a coincidir exatamente com um bloco diagonal da matriz resultante, basta multiplicarmos estes elementos pela matriz $Q^{(N)}$. O Algoritmo 3.5 apresenta os passos necessários para realizar esta operação.

Algoritmo 3.5

1 $index = j$;
2 **for** $l = 0, 1, \ldots, n_N - 1$
3 **do** $z_{in}[l] = vAux[index]$;
4 $index = index + 1$;
5 **end do**
6 **multiply** $z_{out} = z_{in} \times Q^{(N)}$;
7 $index = j$;
8 **for** $l = 0, 1, \ldots, n_N - 1$
9 **do** $vRes[index] = vRes[index] + z_{out}[l]$;
10 $index = index + 1$;
11 **end do**

Algoritmo 3.5: Multiplicação $v \times I_{nleft_N} \otimes Q^{(N)}$

Primeiramente, o Algoritmo 3.5 extrai os n_N elementos não-nulos do vetor $vAux$ e coloca-os em um vetor menor chamado de z_{in} (linhas 1 a 5), onde $index$ é inicializado pelo índice da coluna j correspondente ao elemento não-nulo da matriz unitária do primeiro Fator Normal Tradicional do AUNF. Em seguida, multiplica-se o vetor z_{in} pela última matriz (linha 6) e por fim acumula os resultados no vetor $vRes$ (linhas 7 a 11), que contém previamente os resultados acumulados de todas as operações anteriores. Note que todo o procedimento é realizado apenas com a utilização

do vetor $vAux$ e da matriz $Q^{(N)}$, não sendo necessário gerar a matriz $I_{nleft_N} \otimes Q^{(N)}$ em nenhum momento.

Após realizar cada uma das operações descritas nesta seção, tem-se então o vetor resultado da multiplicação do vetor de probabilidades pela parte sincronizante do descritor Markoviano. Basta então somar este vetor com o vetor resultado da parte local para enfim encontrar-se o vetor contendo o resultado final da multiplicação vetor-descritor.

3.3 Taxas Funcionais na técnica *Slice*

A utilização de taxas funcionais é muito comum no formalismo de redes de autômatos estocásticos. Muitos modelos SANapresentam taxas funcionais com intuito de facilitar e compactar a descrição do sistema no formalismo, uma vez que algumas peculiaridades do sistema podem ser mais facilmente modelados utilizando-se este importante recurso.

Até então, os poucos estudos realizados sobre o algoritmo da técnica *Slice* não levaram em consideração a utilização de taxas funcionais em modelos SAN. Acreditando ser fundamental que o algoritmo *Slice* funcione também para modelos envolvendo taxas funcionais, foi realizada uma análise do algoritmo da técnica *Slice* com o intuito de descobrir quais partes deveriam ser modificadas ou até mesmo reescritas para suportar o uso de taxas funcionas nos modelos.

Porém, após análise do algoritmo original da técnica *Slice*, chegou-se a conclusão que o uso de taxas funcionais não modificaria a estrutura geral do algoritmo. A única parte do algoritmo que necessitaria de uma pequena modificação é a parte responsável por gerar os elementos não-nulos de cada AUNF. Isto porque as taxas funcionais, como visto na seção 2.1.4 do capítulo 2, assumem valores que dependem do estado local dos demais autômatos do modelo, sendo assim necessário sempre reavaliar todos os elementos.

Com isso, a otimização proposta e implementada para o cálculo dos elementos não-nulos de cada AUNF apresentado no Algoritmo 3.3 não é mais válida. Pois com o uso de taxas funcionais, é necessário sempre percorrer todas as matrizes para avaliar as possíveis funções, uma vez que a cada mudança de um estado local (linha de um elemento de uma matriz de transição), todas as outras taxas funcionais do modelo podem ser afetadas. Portanto, a otimização proposta na seção 3.2.2, a qual armazena resultados parciais de multiplicações entre elementos de matrizes de transições sincronizantes, não pode ser utilizada quando o modelo SAN apresentar taxas funcionais.

O Algoritmo 3.6 apresenta como é realizado o cálculo dos elementos não-nulos de cada AUNF para modelos SAN que incluem taxas funcionais na sua definição.

Na linha 1, a função *nextElemIndex* descobre o próximo índice composto (índice de cada matriz envolvida no cálculo) para o elemento a ser calculado. Em seguida, é atribuída a variável *nFator* o valor do elemento da primeira matriz. Depois, este valor é multiplicado pelo elemento da segunda matriz, da terceira e assim sucessivamente (linhas 3, 4 e 5) até a penúltima matriz do modelo. Enfim, na linha 6, a variável *nFator* conterá o próximo fator da AUNF que é então retornado.

Algoritmo 3.6

```
1  nextElemIndex(nIndex);
2  nFator = getElem(0, nIndex[0]);
3  for n = 1, ..., nLastAut − 1
4  do  nFator = nFator ∗ getElem(n, nIndex[n]);
5  end do
6  returnnFator;
```

Algoritmo 3.6: Calcula o elemento não-nulo de cada AUNF para modelos SAN com taxas funcionais

Novas alternativas de otimização para o algoritmo da técnica *Slice* precisam ser estudados no caso do uso de taxas funcionais nos modelos SAN. Em especial, seria interessante estudar em detalhes o cálculo dos elementos não-nulos dos fatores normais unitários aditivos (AUNF) quando do uso de taxas funcionais, tentando-se encontrar uma forma de otimizar este cálculo, visto que a otimização proposta neste estudo só é válida para modelos com taxas exclusivamente constantes.

3.4 Custo Computacional

O custo computacional da técnica *Slice* para a parte local de um modelo SAN é igual ao custo da técnica *Shuffle* para esta mesma parte, uma vez que o número de multiplicações em ponto-flutuante é o mesmo, o que difere uma da outra é apenas a forma como o cálculo é realizado. Portanto, considerando N o número de autômatos do modelo SAN, n_i a dimensão da i-ésima matriz e nz_i o número de elementos não-nulos da matriz i, o custo computacional da técnica *Slice* para a parte local do modelo é representado pela seguinte expressão:

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \tag{3.1}$$

O custo computacional para a parte sincronizante do modelo depende do custo necessário para multiplicar um vetor de probabilidades π por um fator normal unitário aditivo, o qual é representado pela expressão 3.2, onde n_N e nz_N são respectivamente a dimensão e o número de elementos não-nulos da última matriz do termo.

$$\Big[(N-2) + n_N + nz_N\Big] \tag{3.2}$$

Na expressão acima temos $(N-2)$ multiplicações entre elementos das $N-1$ matrizes envolvidas. Além disto, como para cada AUNF são gerados dois fatores normais tradicionais, para o primeiro fator normal temos n_N multiplicações pelo vetor π, e para o segundo, nz_N multiplicações. Sabendo-se que o número de fatores normais unitários aditivos a tratar é dado por

todas as combinações de elementos não-nulos das $N-1$ matrizes iniciais, tem-se como custo computacional total para a parte sincronizante a expressão a seguir:

$$\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big] \tag{3.3}$$

Portanto, o custo computacional total da técnica *Slice* é a soma dos custos da parte local e sincronizante do modelo SAN, resultando na seguinte expressão:

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right) \tag{3.4}$$

Capítulo 4

Análise Numérica

Para demonstrar o quanto a técnica *Slice* pode contribuir para a resolução de modelos SAN, este capítulo apresenta uma análise do custo computacional e de resultados práticos obtidos de dois exemplos de modelos SAN.

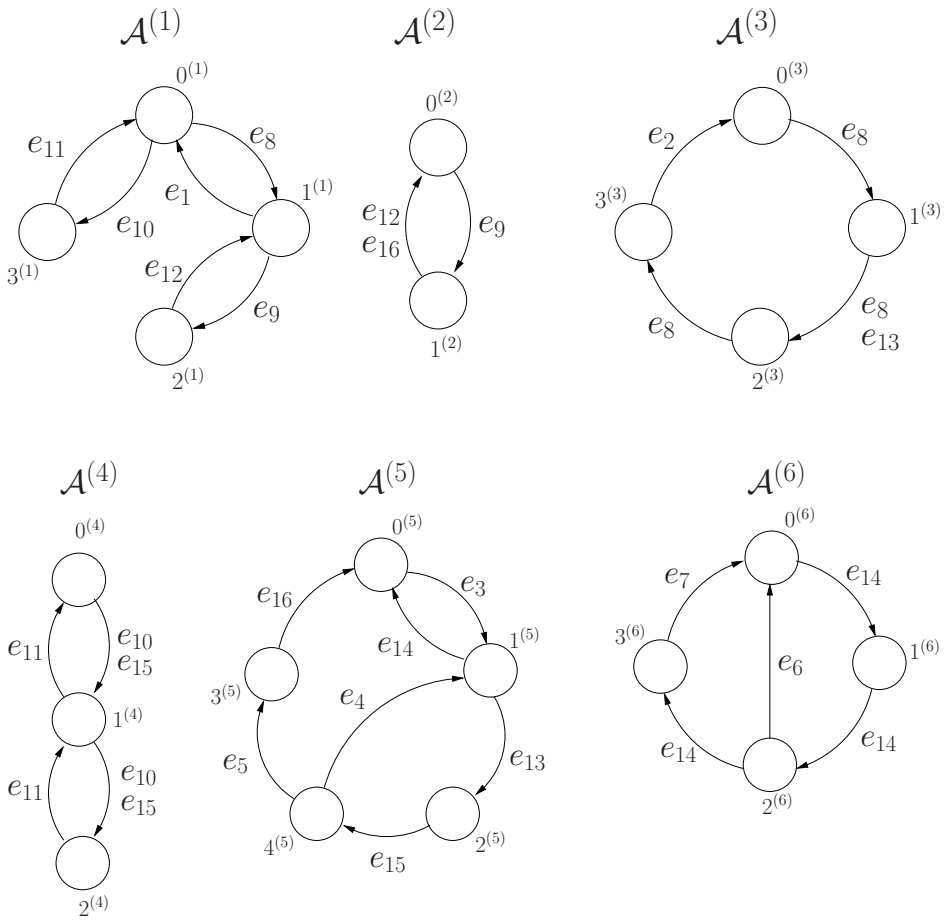


Figura 4.1: Primeiro exemplo de modelo SAN

Evento	Taxa de Ocorrência	Tipo
e_1	τ_1	Local
e_2	τ_2	Local
e_3	τ_3	Local
e_4	τ_4	Local
e_5	τ_5	Local
e_6	τ_6	Local
e_7	τ_7	Local
e_8	τ_8	Sincronizante
e_9	τ_9	Sincronizante
e_{10}	τ_{10}	Sincronizante
e_{11}	τ_{11}	Sincronizante
e_{12}	τ_{12}	Sincronizante
e_{13}	τ_{13}	Sincronizante
e_{14}	τ_{14}	Sincronizante
e_{15}	τ_{15}	Sincronizante
e_{16}	τ_{16}	Sincronizante

Tabela 4.1: Taxa de ocorrência dos eventos do modelo SAN apresentado na Figura 4.1.

Com intuito de fazer uma análise completa do desempenho do algoritmo da técnica *Slice*, dois exemplos de modelos SAN com diferentes características foram criados. Além disso, para os testes práticos, diferentes valores foram atribuídos para as taxas dos eventos locais e sincronizantes dos modelos. Primeiramente, todas as taxas dos eventos dos modelos foram definidas com valores constantes, em seguida, a maioria das taxas receberam valores funcionais, possibilitando assim uma análise do comportamento da técnica *Slice* com e sem taxas funcionais, uma vez que a avaliação das taxas funcionais pode gerar um custo relevante em termos de tempo de execução.

Evento	Taxa de Ocorrência	Tipo
e_1	τ_1	Local
e_2	τ_2	Local
e_3	τ_3	Local
e_4	τ_4	Local
e_5	τ_5	Local
e_6	τ_6	Local
e_7	τ_7	Sincronizante
e_8	τ_8	Sincronizante
e_9	τ_9	Sincronizante
e_{10}	τ_{10}	Sincronizante
e_{11}	τ_{11}	Sincronizante
e_{12}	τ_{12}	Sincronizante
e_{13}	τ_{13}	Sincronizante
e_{14}	τ_{14}	Sincronizante

Tabela 4.2: Taxa de ocorrência dos eventos do modelo SAN apresentado na Figura 4.2.

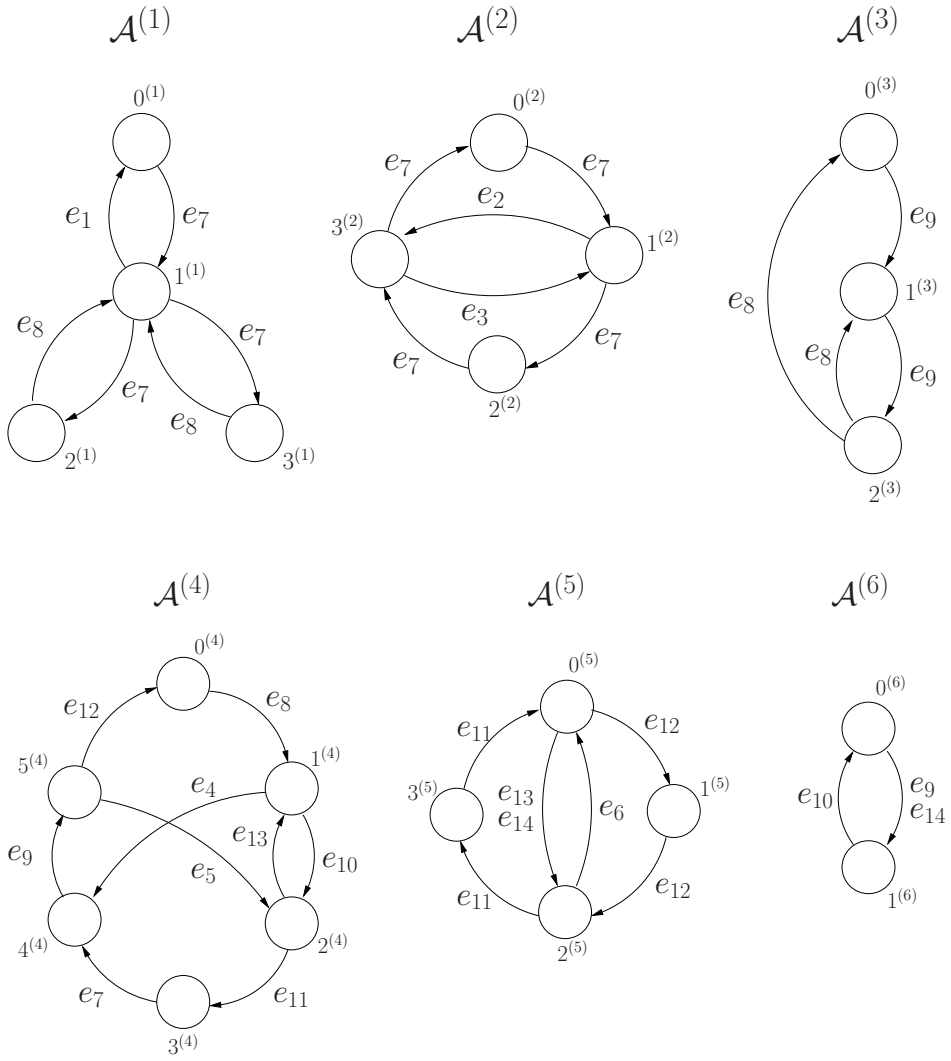


Figura 4.2: Segundo exemplo de modelo SAN

As Figuras 4.1 e 4.2 apresentam os autômatos referentes aos exemplos de modelo SAN utilizados na análise numérica, ao passo que as Tabelas 4.1 e 4.2 descrevem respectivamente os eventos destes modelos. Observa-se nestas tabelas que as taxas são definidas por variáveis, pois como dito anteriormente, estas taxas recebem diferentes valores para cada caso de teste.

Como resultado da análise apresentada neste capítulo pretende-se obter o desempenho da técnica *Slice* sem e com otimizações. Em especial duas otimizações foram testadas, a otimização da diagonal, já estudada detalhadamente em trabalhos anteriores [8], e a otimização na geração dos fatores, a qual é proposta neste trabalho para modelos com taxas exclusivamente constantes. Além disso, é realizada uma comparação entre os resultados da técnica *Slice* e os resultados da tradicional técnica *Shuffle*. Afinal, um dos objetivos da técnica *Slice* é apresentar uma forma mais eficiente de resolver modelos SAN em relação as demais técnicas existentes.

4.1 Análise do Custo Computacional

Antes mesmo de qualquer implementação do algoritmo da técnica *Slice*, é possível verificar na teoria qual seria o seu custo computacional. Utilizando-se da fórmula do custo computacional da técnica *Slice*, facilmente pode-se calcular o número de multiplicações em ponto-flutuante necessárias para realizar uma iteração da resolução de modelos SAN. Essa informação é determinante para avaliar na teoria o desempenho de uma técnica, uma vez que as multiplicações em ponto-flutuante são as principais operações realizadas neste processo. Portanto, sabendo-se o número de multiplicações em ponto-flutuante necessárias para resolver um modelo SAN, tem-se então uma noção do tempo de processamento necessário para a resolução do modelo.

Nesta seção, serão considerados o custo computacional da técnica *Shuffle*, da técnica *Slice*, e também do algoritmo da técnica *Slice* com a otimização na geração dos fatores. Além disso, estes custos serão calculados de duas formas, a primeira sem considerar uma otimização de pré-cálculo da diagonal [8] e a segunda considerando esta otimização.

A otimização do pré-cálculo da diagonal, ou simplesmente otimização da diagonal, consiste em calcular e armazenar previamente todos os elementos da diagonal principal do descritor do modelo SAN, desconsiderando assim os elementos da diagonal principal das matrizes locais e todas as matrizes negativas dos eventos sincronizantes do modelo. Com isso, diminui-se o número de multiplicações em cada iteração da resolução do modelo SAN. Por outro lado, o custo em memória para armazenar a diagonal principal do descritor pode ser bastante considerável dependendo do tamanho do modelo SAN em questão, o qual será abordado na análise dos resultados práticos.

Observando as fórmulas do custo computacional da técnica *Shuffle* e *Slice*, percebe-se que o custo para realizar os cálculos da parte local de um modelo SAN é igual para as duas técnicas. Isto ocorre pois a parte local dos modelos é representada através de uma soma tensorial, a qual é reescrita em um somatório de produtos tensoriais envolvendo inúmeras matrizes identidades. Por conseqüência, a grande quantidade de matrizes identidades envolvidas no cálculo, acaba anulando a vantagem em utilizar-se a técnica *Slice*, uma vez que diversas multiplicações se tornam desnecessárias.

Analisando-se mais detalhadamente o cálculo da parte local, percebe-se que este nada mais é do que a multiplicação de elementos de uma dada matriz do modelo por diversos elementos do vetor de probabilidades, os quais dependem da localização da matriz do modelo no produto tensorial com as matrizes identidades. Ou seja, a multiplicação do vetor de probabilidades por um fator normal. Este processo é exatamente o foco da técnica *Shuffle*, como visto na seção 2.4.1. Por isso, pode-se concluir que na teoria, o cálculo da parte local é realizado de forma mais eficiente pela técnica *Shuffle*, pois ao contrário desta, a técnica *Slice* não se preocupa com o custo necessário para descobrir os elementos do vetor ao qual cada elemento das matrizes do modelo devem ser multiplicados. Por outro lado, a parte local do algoritmo da técnica *Slice*, assim como a parte sincronizante, é mais flexível a implementações paralelas, pois a multiplicação do vetor de probabilidades pelas matrizes do modelo é realizada separadamente para cada elemento das matrizes, enquanto que na técnica *Shuffle* esta operação é realizada utilizando-se de uma vez só todos os elementos de cada matriz do modelo SAN.

Sendo assim, quando se tratar de uma implementação seqüencial, sugere-se utilizar a técnica *Shuffle* para o cálculo da parte local do modelo, enquanto que para implementações paralelas, a

técnica *Slice* mostra-se mais adequada.

4.1.1 Custo Computacional sem otimização da diagonal

Nesta seção serão apresentados o custo computacional para as técnicas *Shuffle* e *Slice* sem considerar a otimização da diagonal. Ou seja, fazem parte do cálculo todos os elementos diagonais das matrizes locais dos modelos, assim como todas as matrizes negativas dos eventos sincronizantes.

Custo Computacional para o primeiro exemplo

Sabendo-se que N é o número total de autômatos do modelo, E é o número de eventos sincronizantes, n_i é o número de estados do autômato i e que nz_i é o número de transições que o autômato i possui para cada evento local e sincronizante do modelo, pode-se facilmente calcular o custo computacional para cada uma das técnicas.

Primeiramente, vejamos para o exemplo apresentado pela Figura 4.1 quais seriam os valores para cada uma das variáveis das fórmulas do custo computacional das técnicas, considerando a parte local e sincronizante do modelo SAN. Observando a Figura 4.1 e a Tabela 4.1 sabe-se que para este exemplo, o número de autômatos N é 6 e o total de eventos sincronizantes E é 9. A Tabela 4.3 apresenta ainda a definição das variáveis n_i e nz_i para a parte local do primeiro exemplo de modelo SAN.

i	n_i	nz_i
1	4	2
2	2	0
3	4	2
4	3	0
5	5	5
6	4	4

Tabela 4.3: Definição das variáveis para a parte local do primeiro modelo SAN

i	n_i	nz_i								
		e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}	e_{15}	e_{16}
1	4	1	1	1	1	1	4	4	4	4
2	2	2	1	2	2	1	2	2	2	1
3	4	3	4	4	4	4	1	4	4	4
4	3	3	3	2	2	3	3	3	2	3
5	5	5	5	5	5	5	1	1	1	1
6	4	4	4	4	4	4	3	4	4	4

Tabela 4.4: Definição das variáveis para a parte sincronizante positiva e negativa do primeiro modelo SAN

Na Tabela 4.4 são apresentadas as definições das variáveis n_i e nz_i para cada evento sincronizante do modelo, referindo-se a parte sincronizante do primeiro exemplo de modelo SAN. Como nenhum autômato deste modelo apresenta duas ou mais transições saindo de um único estado para outros utilizando o mesmo evento sincronizante, pode-se considerar os valores apresentados na Tabela 4.4 para as matrizes positivas e negativas do modelo.

Considerando as Tabelas 4.3 e 4.4, e ainda a fórmula do custo computacional para a técnica *Shuffle*, o primeiro exemplo de modelo SAN apresenta o seguinte custo computacional (cálculo completo no apêndice B):

$$\sum_{k=1}^{1+2E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right) = 171.992$$

Portanto, utilizando-se a técnica *Shuffle* são necessárias 171.992 multiplicações em ponto-flutuante para cada iteração da resolução deste primeiro exemplo de modelo SAN. Desta forma, tem-se uma noção do tempo de processamento necessário para realizar este cálculo.

Sabendo que n_N e nz_N são respectivamente a ordem e o número de elementos não-nulos da matriz do último autômato do modelo SAN, no caso o autômato $A^{(6)}$, o custo computacional da técnica *Slice* para este exemplo é (consulte o apêndice B para detalhes do cálculo):

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N \Big] \right) = 20.160$$

Portanto, se a técnica *Slice* for utilizada para resolver este exemplo de modelo SAN, serão necessárias apenas 20.160 multiplicações em ponto-flutuante para cada iteração, ao passo que para a técnica *Shuffle* seriam necessárias 171.992 destas operações.

Custo Computacional para o segundo exemplo

Analisando a Figura 4.2 e a Tabela 4.2 observa-se que para este segundo exemplo de modelo SAN, o número de autômatos N é 6 e o total de eventos sincronizantes E é 8. As Tabelas 4.5, 4.6 e 4.7 apresentam ainda as definições das variáveis n_i e nz_i para a parte local, sincronizante positiva e sincronizante negativa respectivamente para este modelo SAN.

Considerando os valores apresentados pelas Tabelas 4.5, 4.6 e 4.7, o cálculo do custo computacional da técnica *Shuffle* para este exemplo de modelo SAN seria (cálculo completo no apêndice B):

$$\sum_{k=1}^{1+2E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right) = 187.968$$

i	n_i	nz_i
1	4	2
2	4	4
3	3	0
4	6	4
5	4	2
6	2	0

Tabela 4.5: Definição das variáveis para a parte local do segundo modelo SAN

i	n_i	nz_i							
		e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}
1	4	3	2	4	4	4	4	4	4
2	4	4	4	4	4	4	4	4	4
3	3	3	2	2	3	3	3	3	3
4	6	6	6	6	2	2	2	1	6
5	4	4	4	4	4	2	2	1	1
6	2	2	2	1	1	2	2	2	1

Tabela 4.6: Definição das variáveis para a parte sincronizante positiva do segundo modelo SAN

i	n_i	nz_i							
		e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}
1	4	2	2	4	4	4	4	4	4
2	4	4	4	4	4	4	4	4	4
3	3	3	1	2	3	3	3	3	3
4	6	6	6	6	2	2	2	1	6
5	4	4	4	4	4	2	2	1	1
6	2	2	2	1	1	2	2	2	1

Tabela 4.7: Definição das variáveis para a parte sincronizante negativa do segundo modelo SAN

Portanto, utilizando-se a técnica *Shuffle* para resolver o segundo exemplo de modelo SAN, são necessárias 187.968 multiplicações em ponto-flutuante para cada iteração da resolução deste modelo.

Para este mesmo exemplo, pode-se calcular também o provável custo computacional para a técnica *Slice*, o qual seria (detalhes do cálculo no apêndice B):

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right) \quad = \quad 49.344$$

Portanto, se utilizarmos a técnica *Slice* para resolver este mesmo exemplo de modelo SAN, serão necessárias 49.344 multiplicações em ponto-flutuante para cada iteração da resolução deste modelo.

Custo Computacional para *Slice* Otimizado

Como visto anteriormente, no capítulo 3, para modelos com taxas exclusivamente constantes é possível realizar uma otimização no cálculo da parte sincronizante dos modelos SAN. A idéia é otimizar o processo de descoberta dos fatores normais unitários aditivos, reaproveitando as multiplicações já realizadas entre os elementos das matrizes do termo calculado.

A fórmula do custo computacional original para a parte sincronizante da técnica *Slice* considera que as multiplicações entre os elementos das matrizes são realizas toda vez para cada fator normal unitário aditivo. Analisando novamente esta equação, nota-se que para cada AUNF, representadas pelo termo, são contabilizadas $N-2$ multiplicações.

Custo computacional original para parte sincronizante.

$$\sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right)$$

Implementando-se a otimização proposta, não se fazem necessárias tantas multiplicações para descobrir os fatores normais unitários aditivos. A lógica é simples, se é preciso realizar diversas multiplicações entre três ou mais números, variando-se um número de cada vez, pode-se armazenar uma parte do cálculo para ser reaproveitada, evitando assim repetições de multiplicações já realizadas, vejamos um exemplo pequeno:

Expressões para serem resolvidas:

$$3 \times 4 \times 2 \times 5$$
$$3 \times 4 \times 2 \times 8$$
$$3 \times 4 \times 6 \times 5$$
$$3 \times 4 \times 6 \times 8$$

Sem otimização seriam necessária 12 multiplicações.

$$\begin{aligned}
3 \times 4 \times 2 \times 5 &= 12 \times 2 \times 5 = 24 \times 5 = 120\\
3 \times 4 \times 2 \times 8 &= 12 \times 2 \times 8 = 24 \times 8 = 192\\
3 \times 4 \times 6 \times 5 &= 12 \times 6 \times 5 = 72 \times 5 = 360\\
3 \times 4 \times 6 \times 8 &= 12 \times 6 \times 8 = 72 \times 8 = 576
\end{aligned}$$

Com otimização seriam necessária apenas 7 multiplicações.

$$\begin{array}{rcrcrcr} 3\times 4\times 2\times 5 & = & 12\times 2\times 5 & = & 24\times 5 & = & 120 \\ \times 8 & = & \times 8 & = & 24\times 8 & = & 192 \\ \times 6\times 5 & = & 12\times 6\times 5 & = & 72\times 5 & = & 360 \\ \times 8 & = & \times 8 & = & 72\times 8 & = & 576 \end{array}$$

Neste pequeno exemplo já pode-se ter uma noção da quantidade de multiplicações que seriam evitadas pela otimização do cálculo. Percebe-se que os resultados intermediários 12, 24 e 72 são armazenados e reutilizados para os próximos cálculos, evitando assim uma série de multiplicações desnecessárias.

Logo, considerando essa otimização, pode-se reescrever a fórmula do custo computacional para a parte sincronizante da técnica *Slice* da seguinte forma:

$$\sum_{k=1}^{2E}\left(\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right] + \prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]\right) \tag{4.1}$$

Sendo assim, considerando que os exemplos 1 e 2 de modelo SAN apresentem apenas taxas constantes, o custo computacional para estes utilizando o algoritmo otimizado da técnica *Slice* seria (cálculo completo no apêndice B):

Exemplo 1:

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E}\left(\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right] + \prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]\right) = 17.468$$

Exemplo 2:

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E}\left(\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right] + \prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]\right) = 35.056$$

4.1.2 Custo Computacional com otimização da diagonal

O custo computacional para as técnicas *Shuffle* e *Slice* apresentado nesta seção leva em consideração a otimização da diagonal. Ou seja, são eliminados do cálculo todos os elementos diagonais das matrizes locais dos modelos, assim como todas as matrizes negativas dos eventos sincronizantes. Dessa forma, o número de multiplicações em ponto-flutuante certamente será reduzido. Vejamos a seguir como seriam os custos computacionais para os dois exemplos estudados.

Custo Computacional para o primeiro exemplo

Tendo realizado o cálculo do custo computacional sem a otimização da diagonal, torna-se fácil de descobrir o custo considerando esta otimização. Basta refazer os cálculos da parte local do modelo, uma vez que os valores de nz_i para as matrizes locais sofrem modificações, somar com o resultado da parte sincronizante positiva obtido anteriormente, e por fim desconsiderar o cálculo da parte sincronizante negativa do modelo SAN.

i	n_i	nz_i
1	4	1
2	2	0
3	4	1
4	3	0
5	5	3
6	4	2

Tabela 4.8: Definição das variáveis para a parte local do primeiro modelo SAN

Considerando a Tabela 4.8, que apresenta a definição das variáveis n_i e nz_i para a parte local do primeiro exemplo de modelo SAN com a otimização da diagonal, o custo computacional para as técnicas *Shuffle* e *Slice* seriam (detalhes do cálculo no apêndice B):

Custo computacional da técnica *Shuffle* para este modelo SAN.

$$\sum_{k=1}^{1+E}\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) \;=\; 86.208$$

Custo computacional da técnica *Slice* para este modelo SAN.

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right) \;=\; 10.272$$

Portanto, observa-se que com a otimização da diagonal, o número de multiplicações em ponto-flutuante para cada iteração é reduzido de forma significativa, tanto para a técnica *Shuffle* quanto para a técnica *Slice*.

Custo Computacional para o segundo exemplo

Assim como realizado para o exemplo anterior, para descobrir o custo computacional considerando a otimização da diagonal, deve-se recalcular apenas a parte local do segundo modelo SAN de exemplo.

i	n_i	nz_i
1	4	1
2	4	2
3	3	0
4	6	2
5	4	1
6	2	0

Tabela 4.9: Definição das variáveis para a parte local do segundo modelo SAN

Considerando a Tabela 4.9, que apresenta as definições das variáveis n_i e nz_i para a parte local do modelo com a otimização da diagonal, o custo computacional para as técnicas *Shuffle* e *Slice* seriam (cálculo completo no apêndice B):

Custo computacional da técnica *Shuffle* para este modelo SAN.

$$\sum_{k=1}^{1+E}\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) \;=\; 94.656$$

Custo computacional da técnica *Slice* para este modelo SAN.

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right) \;=\; 26.592$$

Portanto, mais uma vez verifica-se que com a otimização da diagonal, o número de multiplicações em ponto-flutuante para cada iteração diminui significativamente para ambas as técnicas.

Custo Computacional para *Slice* Otimizado

Considerando que a otimização dos fatores proposta para o algoritmo da técnica *Slice* refere-se somente à parte sincronizante do modelo, e que a otimização da diagonal não modifica o cálculo da parte sincronizante positiva, pode-se utilizar estas duas otimizações em conjunto, obtendo um custo computacional de 8.926 e 18.800 para o primeiro e segundo exemplos de modelo SAN respectivamente.

4.1.3 Análise dos Resultados Obtidos

Após calcular o custo computacional das técnicas *Shuffle* e *Slice* para os dois exemplos de modelo SAN apresentados pelas Figuras 4.1 e 4.2, é possível fazer uma análise sobre os resultados obtidos.

Técnica	Sem Diagonal		Com Diagonal	
	Exemplo 1	Exemplo 2	Exemplo 1	Exemplo 2
Shuffle	171.992	187.968	86.208	94.656
Slice	20.160	49.344	10.272	26.592
Slice Otimizado	17.468	35.056	8.926	18.800

Tabela 4.10: Custo Computacional para os modelos SAN de exemplo

A Tabela 4.10 apresenta os resultados do custo computacional para calcular uma iteração da resolução dos exemplos 1 e 2 através das técnicas *Shuffle* e *Slice*. O custo computacional é representado pelo número de multiplicações em ponto-flutuante necessárias para realizar o cálculo.

Analisando os custos computacionais das técnicas *Slice* e *Shuffle* para os dois modelos SAN apresentados nesta seção, percebe-se claramente na teoria que a técnica *Slice* apresenta um desempenho bastante superior a técnica *Shuffle* na resolução destes modelos, seja utilizando-se a otimização da diagonal ou não.

Sem a otimização da diagonal a técnica *Slice* obteve um custo computacional mais de $8,5$ vezes menor para o exemplo 1 e de aproximadamente $3,8$ vezes menor para o exemplo 2. Considerando a otimização da diagonal, a técnica *Slice* mantém uma boa vantagem em relação a *Shuffle*, porém diminui um pouco. Para o exemplo 1, a técnica *Slice* apresenta um custo computacional quase $8,4$ vezes melhor que a *Shuffle*, enquanto que para o exemplo 2 essa relação é de aproximadamente $3,56$. Pode-se atribuir esta pequena diferença da vantagem da técnica *Slice* sobre a *Shuffle*, quando utiliza-se a otimização da diagonal, ao fato de que com esta otimização não são contabilizadas as multiplicações da parte sincronizante negativa do modelo, a qual a técnica *Slice* levaria vantagem em relação a técnica *Shuffle*. Porém, percebe-se que para ambas as técnicas a otimização da diagonal reduz consideravelmente o custo computacional, demonstrando assim que a otimização da diagonal também é válida para a técnica *Slice*.

Além disso, utilizando-se a otimização na geração dos fatores para modelos com taxas exclusivamente constantes, observa-se que o custo computacional da técnica *Slice* tornou-se ainda melhor, apresentando uma redução no número de multiplicações de aproximadamente 13% e 29% para os exemplos 1 e 2 respectivamente. O que aumenta ainda mais a vantagem da técnica *Slice* em relação a *Shuffle*, chegando a apresentar um custo computacional quase 10 vezes menor para o primeiro exemplo e acima de 5 vezes menor para o segundo exemplo. Comprovando-se assim, que a otimização proposta para a técnica *Slice* reduz significativamente o número de multiplicações em ponto-flutuante necessárias para realizar o cálculo de cada iteração da resolução do modelo SAN.

Considerando que para resolver um modelo SAN normalmente são necessárias inúmeras iterações e que os custos computacionais apresentados pela Tabela 4.10 são para o cálculo de apenas uma iteração da resolução dos modelos SAN, pode-se imaginar o quanto seria o ganho total no desempenho para a resolução completa de um modelo SAN.

4.2 Análise de Resultados Práticos

A análise do custo computacional é importante para fundamentar os estudos realizados e também para encorajar os pesquisadores a continuar os estudos caso estes resultados teóricos sejam positivos. Assim, considerando que os resultados da análise do custo computacional da técnica *Slice* foram muito significativos, torna-se válida a realização de uma análise sobre resultados práticos desta técnica. Realizando essa análise será possível verificar se os resultados teóricos se confirmam na prática, provando assim o bom desempenho da técnica *Slice*.

Para realizar uma análise sobre resultados práticos foi preciso implementar o algoritmo da técnica *Slice*, uma vez que nenhuma implementação deste algoritmo havia sido realizada até então. Logo, uma nova versão da ferramenta PEPS foi desenvolvida incluindo o algoritmo da técnica *Slice*. Dessa forma, foi possível realizar testes práticos com as duas técnicas, pois o algoritmo *Shuffle* já havia sido implementado nesta ferramenta.

É importante salientar que a análise realizada neste trabalho é exclusivamente numérica e baseou-se apenas em algoritmos seqüenciais para ambas técnicas. Ou seja, não foram realizadas implementações paralelas e nem otimizações algorítmicas, como a utilização de pacotes de softwares para otimizar determinadas funções.

A implementação da técnica *Slice* foi realizada utilizando-se o algoritmo *Slice* para a parte sincronizante e o algoritmo *Shuffle* para o cálculo da parte local dos modelos SAN, uma vez que na teoria para implementações seqüenciais este apresenta-se mais adequado para esta parte. Porém, em momento algum houve influência nos resultados dos testes realizados, pois a principal diferença entre estas duas técnicas está exatamente na parte sincronizante dos modelos SAN.

Para obter resultados práticos através da ferramenta PEPS , utilizou-se novamente os dois exemplos de modelos SAN apresentados pelas Figuras 4.1 e 4.2. Os cálculos foram realizados sobre o sistema operacional *Linux* em um computador Pentium IV Xeon 2, 2 GHz com 512 MB de memória cache e 4 GB de memória principal.

Além disso, com o intuito de obter resultados de tempo mais precisos, foram executadas dez vezes cada caso do teste, com aproximadamente 1.000 iterações, e considerado como resultado final a média dos tempos obtidos. Logo, todos resultados apresentados nas tabelas e gráficos das próximas seções são referentes ao tempo de execução de 1.000 iterações da resolução do modelo SAN.

Para distinguir os resultados da técnica *Slice* sem e com a otimização da diagonal nas tabelas e gráficos apresentados nesta seção, estabeleceu-se que a legenda *Slice-SD* representa resultados da técnica *Slice* sem a otimização da diagonal, ao passo que a legenda *Slice-CD* representa os resultados com esta otimização.

4.2.1 Resultados sem a otimização da diagonal

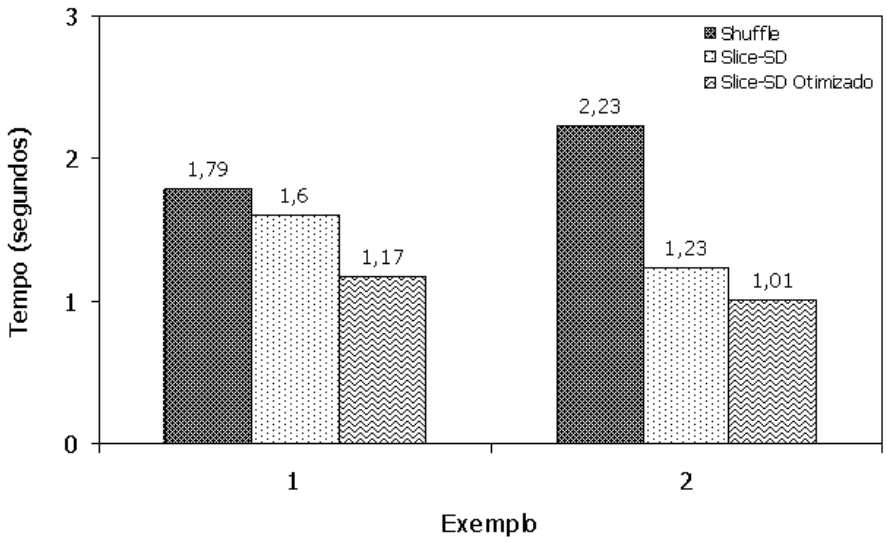
Nesta seção serão apresentados e analisados alguns resultados práticos para a técnica *Slice* sem considerar a otimização da diagonal. Além disso, são apresentados os resultados para a técnica *Shuffle*, porém esta utilizando a otimização da diagonal, uma vez que não existe a im-

plementação desta técnica sem esta otimização. Sendo assim, torna-se necessário apresentar também o total de memória exigida por cada uma das técnicas, para então possibilitar uma análise do custo/benefício de cada técnica.

Resultados para modelos com taxas exclusivamente constantes

Primeiramente foram realizados testes com os dois exemplos de modelos SAN contendo apenas taxas constantes, ou seja, nenhuma taxa dos modelos foi definida através de função. Dessa forma, será possível analisar com maior precisão o desempenho das técnicas, uma vez que o tempo que se gastaria para avaliar funções não é contabilizado.

A Figura 4.3 apresenta o gráfico com os tempos de execução obtidos através da ferramenta PEPS para as técnicas *Shuffle* e *Slice* para os dois exemplos SAN com taxas exclusivamente constantes, além da memória consumida por cada uma das técnicas.



Técnica	Exemplo 1		Exemplo 2	
	Tempo	Memória	Tempo	Memória
Shuffle	1,79 s	92 KB	2,23 s	109 KB
Slice-SD	1,60 s	3 KB	1,23 s	3 KB
Slice-SD Otimizado	1,17 s	3 KB	1,01 s	3 KB

Figura 4.3: Tempo e memória para a execução de modelos com taxas constantes

Observando o gráfico e a tabela da Figura 4.3, percebe-se que a técnica *Slice* mesmo sem utilizar a otimização da diagonal apresenta um melhor desempenho que a técnica *Shuffle*, tanto para o exemplo 1 quanto para o exemplo 2. Além disso, o consumo de memória foi bastante inferior na técnica *Slice*, o que já se previa pelo fato de esta não estar utilizando a otimização da diagonal.

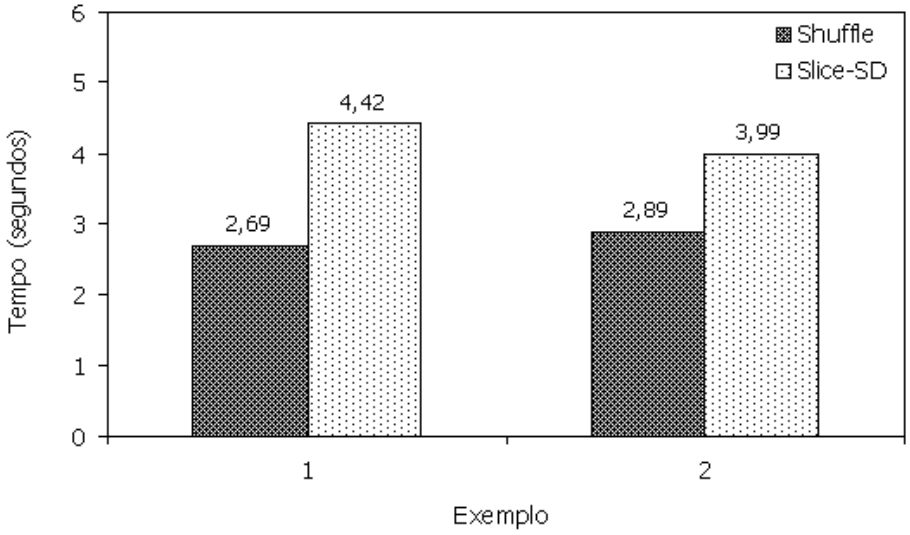
Nota-se também, que a técnica *Slice* obteve um desempenho melhor para o segundo exemplo, o que nos leva a crer que as características deste exemplo sejam mais favoráveis ao algoritmo da técnica *Slice*.

Ainda, é possível verificar o desempenho da técnica *Slice* quando utilizada a otimização na geração dos fatores normais unitários aditivos. Percebe-se que para ambos os exemplos esta otimização se mostra válida, uma vez que obteve-se uma redução no tempo de execução de aproximadamente 27% e 18% para os exemplos 1 e 2 respectivamente.

Resultados para modelos com taxas funcionais

Uma vez analisados os desempenhos das técnicas para modelos SAN com taxas exclusivamente constantes, é importante verificar também o comportamento destas técnicas quando os modelos SAN possuem taxas funcionais. Isto porque a maioria dos sistemas modelados em SAN exige esse tipo de taxa de transição.

A Figura 4.4 apresenta o gráfico com o tempo e a memória exigida para a execução das técnicas *Shuffle* e *Slice* para os exemplos de modelos SAN com diversas taxas funcionais.



Técnica	Exemplo 1		Exemplo 2	
	Tempo	Memória	Tempo	Memória
Shuffle	2,69 s	92 KB	2,89 s	109 KB
Slice-SD	4,42 s	3 KB	3,99 s	3 KB

Figura 4.4: Tempo e memória para a execução de modelos com taxas funcionais

Os resultados apresentados pela Figura 4.4 demonstram que o custo para avaliar taxas funcionais ainda é bastante elevado, principalmente para a técnica *Slice*. Percebe-se claramente isto,

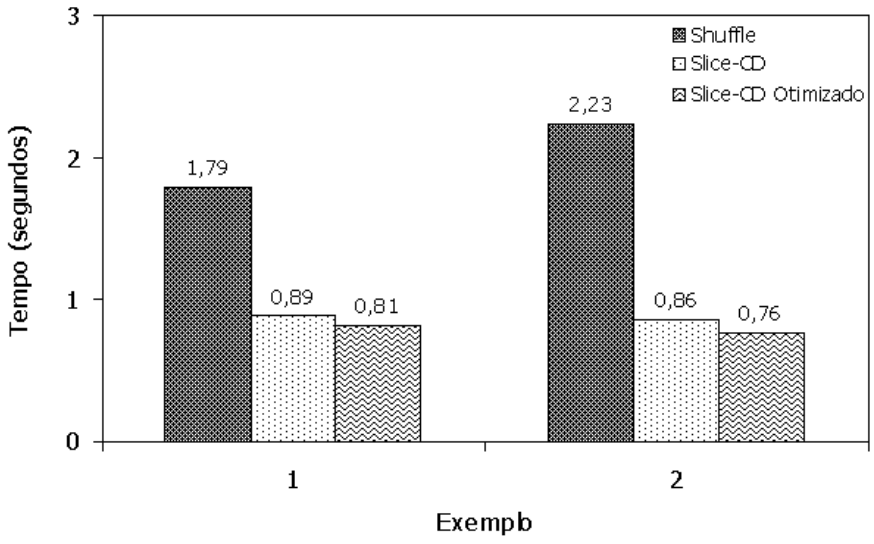
pois para modelos com taxas funcionais o seu desempenho diminuiu significativamente, tornou-se menos eficiente que a técnica *Shuffle*. De qualquer forma, os resultados para a técnica *Slice* são ainda assim bastante interessantes, uma vez que esta não fez uso da otimização da diagonal e também por se tratar da sua primeira implementação. Além disso, o estudo da otimização dos fatores pode ser aprofundado, de forma a abranger também modelos com taxas funcionais, o que certamente iria melhorar consideravelmente o desempenho da técnica nestas situações.

4.2.2 Resultados com a otimização da diagonal

Nesta seção serão apresentados e analisados alguns resultados práticos para as técnicas *Shuffle* e *Slice* para os modelos SAN de exemplo, considerando-se a utilização da otimização da diagonal para ambas as técnicas.

Resultados para modelos com taxas exclusivamente constantes

A Figura 4.5 apresenta o gráfico com os tempos de execução obtidos através da ferramenta PEPS para as técnicas *Shuffle* e *Slice* para os dois exemplos SAN com taxas exclusivamente constantes.



Técnica	Exemplo 1	Exemplo 2
Shuffle	1,79 s	2,23 s
Slice-CD	0,89 s	0,86 s
Slice-CD Otimizado	0,81 s	0,76 s

Figura 4.5: Tempo de execução para modelos com taxas constantes

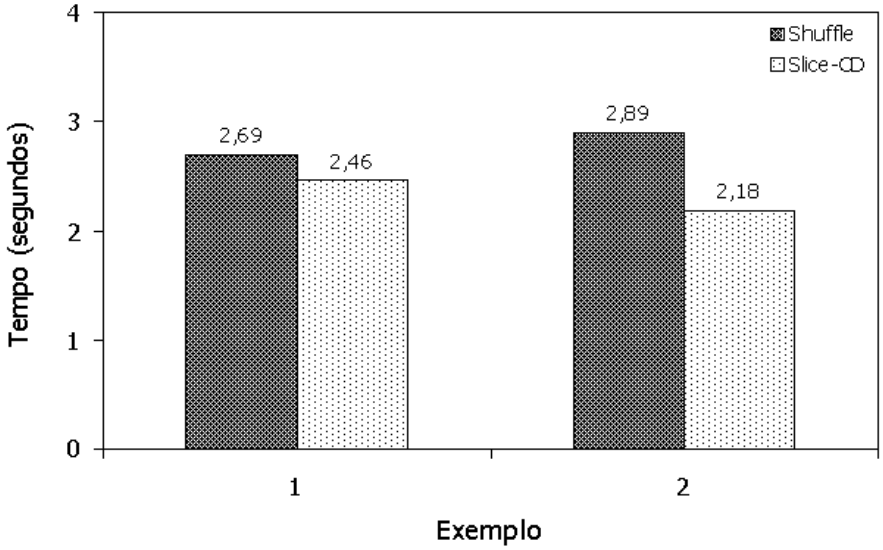
Através da Figura 4.5 é possível analisar o desempenho das técnicas *Shuffle* e *Slice* quando utilizada a otimização da diagonal. Percebe-se que para os dois exemplos a técnica *Slice* apre-

senta um menor tempo de execução, chegando a ser menos da metade do tempo gasto pela técnica *Shuffle* para ambos exemplos. Isso demonstra que para resolver estes exemplos a técnica *Slice* certamente seria a melhor opção.

Além disso, mais uma vez a otimização proposta para o processo de geração dos fatores apresentou resultados significativos.

Resultados para modelos com taxas funcionais

A Figura 4.6 apresenta o gráfico com os tempos de execução das técnicas *Shuffle* e *Slice* para os exemplos de modelos SAN com diversas taxas funcionais.



Técnica	Exemplo 1	Exemplo 2
Shuffle	2,69 s	2,89 s
Slice-CD	2,46 s	2,18 s

Figura 4.6: Tempo de execução para modelos com taxas funcionais

A Figura 4.6 apresenta resultados bastante animadores em relação a resolução de modelos SAN com taxas funcionais através da técnica *Slice*, uma vez que mesmo sendo a primeira implementação desta técnica, os resultados foram consideravelmente melhores do que os obtidos pela tradicional técnica *Shuffle*.

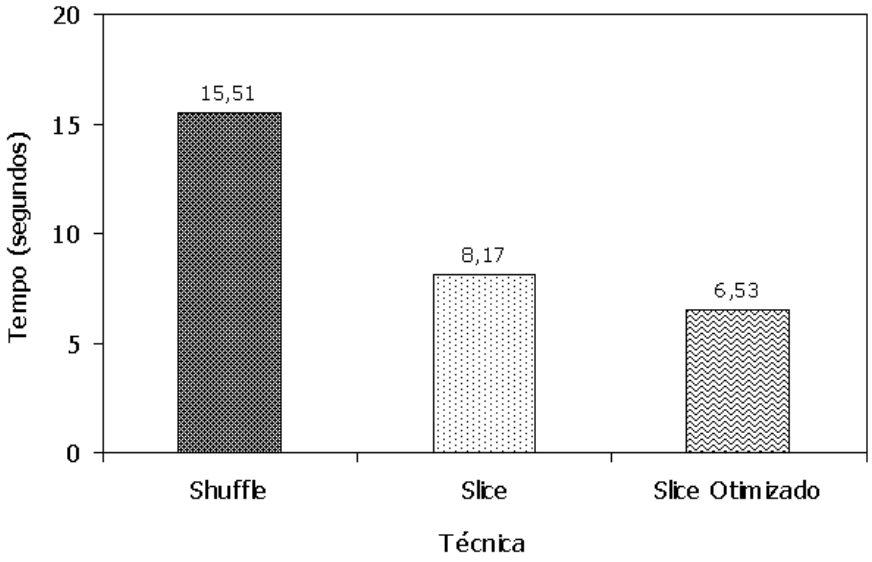
4.2.3 Exemplo com grande espaço de estados

A otimização da diagonal é realmente bastante interessante quando se tem disponível memória suficiente para armazenar a diagonal do modelo sem a necessidade de realizar *swap*. Porém,

se isso não ocorrer, a otimização perde sua validade, podendo até piorar o desempenho da técnica utilizada.

Considerando que a técnica *Slice* apresenta um custo computacional significativamente menor que o custo apresentado pela técnica *Shuffle* e que os resultados práticos comprovam um melhor desempenho para a técnica *Slice*, pode-se não ser interessante utilizar a otimização da diagonal para esta técnica, pois mesmo aumentando um pouco o custo computacional, o ganho na utilização da memória seria compensador. Esta decisão deveria ser tomada caso a caso, dependendo do tamanho do sistema modelado.

Com intuito de demonstrar que para modelos maiores a otimização da diagonal pode não ser interessante, criou-se um exemplo de modelo SAN com 12 autômatos com o número de estados variando entre 2 a 6, totalizando em 10.616.832 o espaço de estados do modelo. Calculou-se então o tempo necessário para executar cada iteração deste modelo utilizando-se a técnica *Shuffle* com a otimização da diagonal e a técnica *Slice* sem esta otimização. Para os cálculos foi utilizado um computador com um processador Pentium IV 2,8 GHz, 256 MB de memória cache e 512 MB de memória principal, sobre o sistema operacional *Linux*. A Figura 4.7 apresenta uma tabela com o tempo de execução para apenas uma iteração da resolução deste modelo, assim como a memória consumida por cada técnica. Além disso, esta figura apresenta também um gráfico comparando os tempos de execução.



Técnica	Tempo	Memória
Shuffle	$15,51s$	324 MB
Slice	8,17 s	30,4 MB
Slice Otimizado	6,53 s	30,4 MB

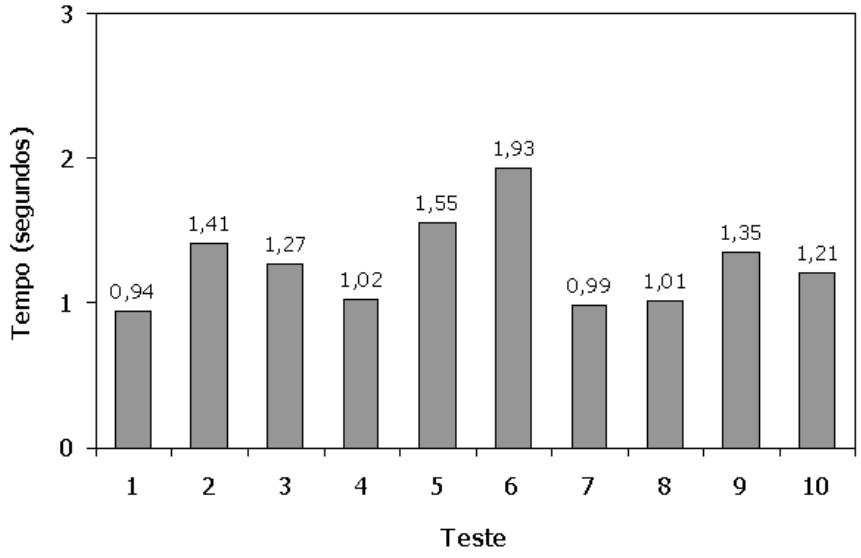
Figura 4.7: Tempo e memória necessários para executar o modelo SAN de exemplo

Observando a Figura 4.7, percebe-se que para este modelo SAN, a implementação da técnica

Shuffle com a otimização da diagonal necessitou de um grande volume de memória, tornando necessário a realização de *swap*, o que acabou influenciando diretamente no seu desempenho. Por este motivo, a técnica *Slice* sem a otimização da diagonal apresentou um desempenho bastante superior. Percebe-se ainda que utilizando a versão otimizada da técnica *Slice*, a qual realiza menos multiplicações para calcular cada fator normal unitário aditivo, o desempenho é ainda melhor, chegando a reduzir em aproximadamente 58% o tempo de execução de cada iteração.

4.2.4 Variações de resultados para a técnica *Slice*

Alguns estudos já demonstravam que a ordem dos autômatos do modelo SAN influencia o desempenho da técnica *Slice*. Na verdade, o que realmente influencia a técnica *Slice*, assim como a própria técnica *Shuffle*, é a ordem das matrizes de cada termo do cálculo da iteração da resolução do modelo SAN.



Teste	Ordem	Tempo
1	$1-2-3-4-5-6$	$0,94$ s
2	$1-2-3-4-6-5$	$1,41$ s
3	$1-2-3-6-5-4$	$1,27$ s
4	$1-2-6-5-4-3$	$1,02$ s
5	$1-6-5-4-3-2$	$1,55$ s
6	$6-5-4-3-2-1$	$1,93$ s
7	$5-4-3-2-1-6$	$0,99$ s
8	$5-4-2-1-6-3$	$1,01$ s
9	$4-2-1-6-3-5$	$1,35$ s
10	$2-1-6-3-5-4$	$1,21$ s

Figura 4.8: Tempo de execução para variações do exemplo 1

A técnica *Shuffle*, por existir a mais tempo e por já ter sido bastante estudada, apresenta uma implementação onde as matrizes são permutadas de forma a realizar o cálculo de cada termo com a melhor disposição possível destas matrizes. O algoritmo desta técnica foi implementado pela primeira vez em 1987 por Plateau [16], e revisado três vezes desde então. Em 1990, Atif [15] realizou a primeira revisão, em 1994 e 1997 foi a vez de Fernandes [7, 8] dar a sua contribuição revisando novamente o algoritmo da técnica *Shuffle*. Isso demonstra o quanto a técnica *Shuffle* já foi melhorada ao longo do tempo.

Por outro lado, a técnica *Slice* está apenas na sua primeira versão, implementada durante a execução deste trabalho. Apesar da implementação do algoritmo desta técnica ter sido realizada de forma cuidadosa, buscando sempre o melhor desempenho, certamente novas versões deste algoritmo serão implementadas no futuro. Até porque, esta primeira versão da técnica *Slice* não inclui permutações entre matrizes o que resultaria em um melhor desempenho da técnica. Isso nem poderia ser realizado, pois o estudo da melhor ordem das matrizes para a técnica *Slice* ainda precisa ser concluído.

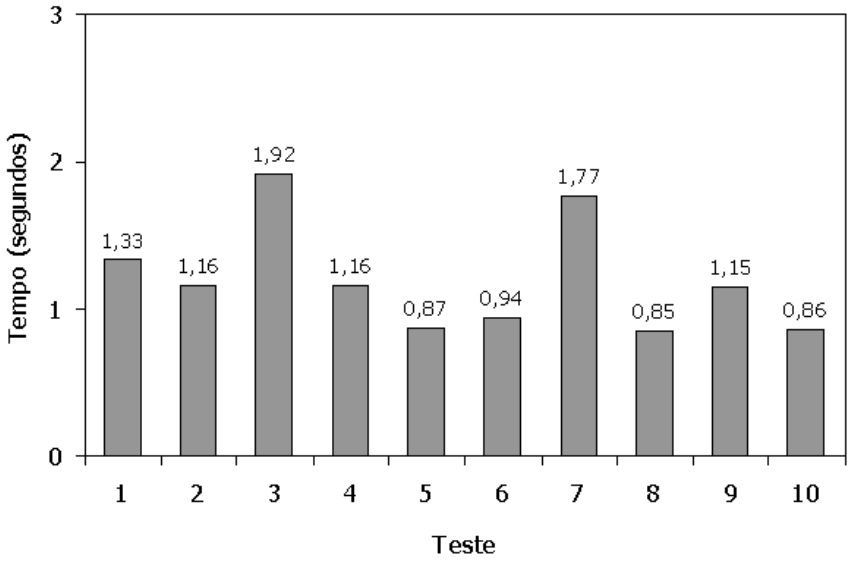
Com o intuito de demonstrar como a ordem dos autômatos ou das matrizes do modelo influenciam diretamente no desempenho da técnica *Slice*, foram testados diversas variações da ordem dos autômatos dos exemplos 1 e 2. Para estes testes considerou-se apenas o algoritmo da técnica *Slice* utilizando-se a otimização da diagonal, mas sem levar em consideração a otimização dos fatores normais unitários aditivos.

As Figuras 4.8 e 4.9 apresentam diversas variações na ordem dos autômatos dos dois modelos SAN de exemplo, assim como o tempo necessário para executar 1.000 iterações para cada nova seqüência.

Analisando as Figuras 4.8 e 4.9 percebe-se que a ordem dos autômatos nos modelos SAN realmente influencia o desempenho da técnica *Slice*. Isso já era esperado, uma vez que o número de fatores normais unitários aditivos depende diretamente da ordem das matrizes nos termos do cálculo. Ou seja, uma determinada seqüência de autômatos pode gerar mais fatores do que outra, e conseqüentemente exigir mais multiplicações em ponto-flutuante, o que certamente consumiria mais tempo para execução.

Observando os gráficos e tabelas das Figuras 4.8 e 4.9 nota-se que os resultados são muito semelhantes para as seqüências que apresentam o mesmo autômato no final. Isso se deve pois a ordem dos $N-1$ primeiros autômatos do modelo não influencia o número de fatores normais unitários aditivos. Por outro lado, a troca do último autômato do modelo modifica as matrizes envolvidas no cálculo dos fatores, o que pode gerar mais ou menos fatores. Dessa forma, parece correto dizer que para a técnica *Slice* o que importa é qual será o último autômato do modelo. Porém, se considerarmos a otimização na geração dos fatores, passa a ser importante também a ordem dos $N-1$ primeiros autômatos, pois dependendo desta ordem a otimização dos fatores pode ser mais efetiva. Além disso, se pensarmos na ordem das matrizes de cada termo individualmente e não simplesmente na ordem dos autômatos do modelo SAN, imagina-se que resultados ainda melhores poderão ser alcançados.

Portanto, torna-se bastante interessante a realização de um estudo detalhado para descobrir a melhor ordem das matrizes para cada termo do cálculo da técnica *Slice*. Assim, como realizado para a técnica *Shuffle*, será possível implementar uma nova versão da técnica *Slice* incorporando



Teste	Ordem	Tempo
1	$1-2-3-4-5-6$	$1,33$ s
2	$1-2-3-4-6-5$	$1,16$ s
3	$1-2-3-6-5-4$	$1,92$ s
4	$1-2-6-5-4-3$	$1,16$ s
5	$1-6-5-4-3-2$	$0,87$ s
6	$6-5-4-3-2-1$	$0,94$ s
7	$6-5-1-3-2-4$	$1,77$ s
8	$6-1-3-4-5-2$	$0,85$ s
9	$3-1-6-4-2-5$	$1,15$ s
10	$2-3-4-6-5-1$	$0,86$ s

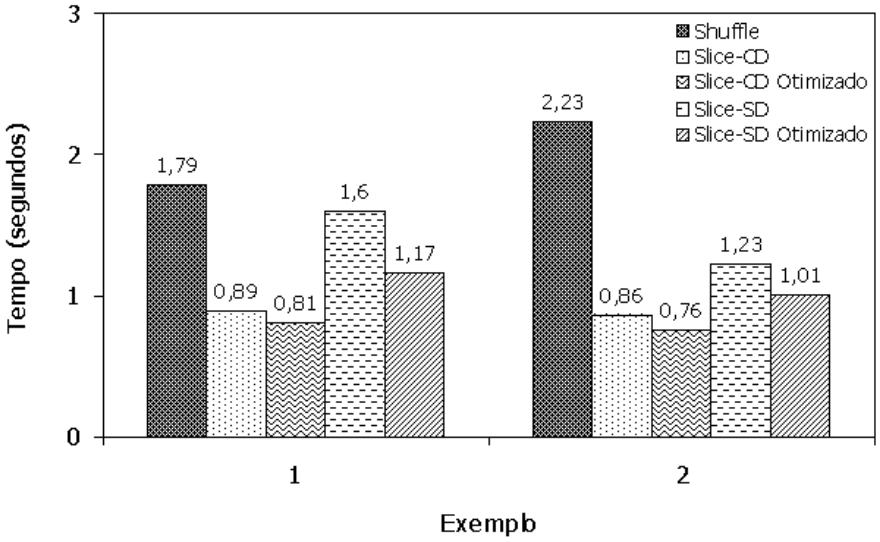
Figura 4.9: Tempo de execução para variações do exemplo 2

mais esta otimização, o que certamente resultará em um desempenho ainda melhor.

4.2.5 Análise dos Resultados Obtidos

A Figura 4.10 reúne os principais resultados apresentados ao longo deste capítulo, facilitando assim uma análise completa do desempenho da técnica *Slice*.

Observando a Figura 4.10 percebe-se que mesmo sem a otimização da diagonal a técnica *Slice* apresenta melhores resultados que a tradicional técnica *Shuffle*. Ou seja, com a técnica *Slice* é possível resolver modelos SAN em menos tempo e consumindo menos memória. Além disso, comparando os resultados obtidos, fica claro que a otimização na geração dos fatores obteve sucesso, diminuindo consideravelmente o tempo necessário para a execução de cada iteração da resolução do modelo SAN.



Técnica	Exemplo 1		Exemplo 2	
	Tempo	Memória	Tempo	Memória
Shuffle	1,79 s	92 KB	2,23 s	109 KB
Slice-CD	0,89 s	92 KB	0,86 s	109 KB
Slice-CD Otimizado	0,81 s	92 KB	0,76 s	109 KB
Slice-SD	1,60 s	3 KB	1,23 s	3 KB
Slice-SD Otimizado	1,17 s	3 KB	1,01 s	3 KB

Figura 4.10: Resultados finais de tempo e memória.

Capítulo 5

Considerações Finais

Infelizmente muitos sistemas em funcionamento hoje em dia nunca tiveram o seu desempenho avaliado de forma consistente, entenda-se por sistemas não só programas de computador, mas toda e qualquer atividade que envolva dados de entrada e de saída. Por exemplo, o funcionamento de um posto de gasolina pode ser considerado um sistema, onde os clientes e o lucro seriam os principais dados deste sistema. Neste caso, diversas avaliações sobre o seu desempenho poderiam ser realizadas, desde a melhor forma de disponibilizar as bombas de gasolina, até o número de funcionários necessário e a capacidade dos tanques de combustível, tudo isso dependendo do número de clientes do posto e visando sempre a satisfação dos clientes e o lucro do proprietário.

A precária forma de avaliação de sistemas ou mesmo a falta desta prática vem prejudicando inúmeras empresas no mercado acirrado dos negócios de hoje em dia. Diversas empresas têm apresentado prejuízos e até mesmo encerrando suas atividades muitas vezes por falta de um planejamento e avaliação adequados do seu negócio. Isso se deve pois até então não existe uma forma consistente e de baixo custo para realizar uma boa avaliação dos sistemas.

Atualmente, para realizar uma completa avaliação de um sistema razoavelmente grande, ao qual envolve diversos fatores e peculiaridades, exigiria-se um enorme custo computacional, muitas vezes tornando-se inviável de ser realizada. Por este motivo, os estudos na área de avaliação de desempenho de sistemas não podem parar, pelo contrário, deve-se investir cada vez mais nesta área, pois tende a ser uma área bastante promissora.

Com a definição do formalismo de Redes de Autômatos Estocásticos (SAN), a modelagem de sistemas tornou-se mais compacta e modular, facilitando assim a modelagem de sistemas maiores e com um maior nível de detalhes. Porém, mesmo com o formalismo SAN, a resolução de alguns sistemas ainda exige um alto custo computacional, uma vez que o espaço de estados destes modelos pode ser muito grande.

Diversas técnicas para resolução de modelos SAN vêm sendo estudadas ao longo do tempo, e muitos resultados positivos já foram encontrados, como a criação e aperfeiçoamento da tradicional técnica *Shuffle*, a qual tornou-se inclusive uma referência mundial no assunto. Porém, novas técnicas têm se mostrado muito interessantes e apresentado ótimos resultados, como é o caso da técnica *Slice* estudada em detalhes neste trabalho.

No decorrer deste estudo foi realizada uma análise profunda da técnica *Slice*, com o intuito de explorar mais os seus pontos fortes e buscar melhorias para os seus pontos que apresentavam

um menor desempenho. Para tanto, implementou-se a primeira versão desta técnica, tirando-a definitivamente do papel, e possibilitando pela primeira vez uma análise de resultados práticos do seu desempenho para resolução de modelos SAN. Além disso, estendeu-se o algoritmo da técnica *Slice* para abranger também modelos com taxas funcionais, o que até então não tinha sido estudado nem mesmo na teoria. Por fim, ainda foi proposta e implementada uma otimização em um dos principais pontos do algoritmo desta técnica, a geração dos fatores normais unitários aditivos.

Após implementar a técnica *Slice*, realizou-se então uma série de testes em diversas situações, visando analisar o desempenho da técnica na resolução de modelos SAN. Com os resultados dos testes realizados foi possível constatar que a técnica *Slice* é realmente uma ótima alternativa para a resolução de sistemas modelados pelo formalismo SAN, uma vez que seus resultados superaram na maior parte das vezes a tradicional técnica *Shuffle*. Além disso, o fato da técnica *Slice* apresentar uma maior flexibilidade para implementações paralelas certamente contribuirá para o seu reconhecimento nesta área de pesquisa.

Ainda como resultados adicionais deste estudo, surgem ao menos cinco possibilidades de trabalhos futuros. Considerando que a análise realizada neste estudo teve como foco principal o número de multiplicações em ponto-flutuante necessárias para os cálculos e não a forma com que o algoritmo foi implementado, pode-se imaginar como um trabalho futuro uma análise algorítmica desta técnica, visando melhorar sua implementação. Seguindo a otimização realizada neste trabalho para a geração dos fatores, poderia-se estudar no futuro uma forma de estender esta otimização para modelos com taxas funcionais, uma vez que neste estudo esta foi limitada a modelos com taxas exclusivamente constantes. Ainda, um estudo profundo para descobrir a melhor ordem das matrizes dos termos do cálculo poderia contribuir muito para o desempenho da técnica *Slice*, como visto na análise de resultados práticos apresentados neste trabalho.

Além disso, um outro trabalho futuro poderia estudar e propor um algoritmo híbrido, o qual utilizaria de forma eficiente as técnicas *Shuffle* e *Slice* para cada termo do cálculo da resolução do modelo SAN, levando-se em consideração as melhores situações para cada técnica. Por fim, ainda seria muito interessante a implementação de uma versão paralela do algoritmo da técnica *Slice*, pois dessa forma certamente seria possível resolver sistemas com maiores espaços de estados.

Enfim, pode-se afirmar que este trabalho contribuiu muito para a evolução da técnica *Slice* e conseqüentemente para a área de avaliação de desempenho de sistemas. Pois, a partir de agora será possível resolver inúmeros sistemas com esta técnica e assim seguir os estudos para quem sabe um dia a técnica *Slice* tornar-se a nova referência mundial para resolução de modelos SAN.

Referências Bibliográficas

[1] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, W. J. Stewart. **The PEPS Software Tool.** P. Kemper, W. H. Sanders, 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation TOOLS 2003, LNCS 2794, pp. 98-115, Urbana, Illinois, EUA, Setembro 2003.

[2] A. Benoit, B. Plateau, W. J. Stewart. **Memory efficient iterative methods for stochastic automata networks**. INRIA, Rapport de Recherche no. 4259, França, 2001. *ftp://ftp.inria.fr/INRIA/Publication/RR/RR-4259.ps.gz*

[3] A. Benoit, B. Plateau, W. J. Stewart. **Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems**. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), pp. 275–282. IEEE Computer Society Press, Nice, França, Abril 2003.

[4] P. Buchholz, G. Ciardo, S. Donatelli, P. Kemper. **Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models**. INFORMS Journal on Computing, vol. 13, no. 3, 2000, pp. 203-222.

[5] M. Davio. **Kronecker products and shuffle algebra**. IEEE Transactions on Computers, vol. C-30, no. 2, 1981, pp. 116-125.

[6] P. Fernandes. **Méthodes numériques pour la solution de systèmes Markoviens à grand espace d'états**. Institut National Polytechnique de Grenoble, França, 1998.

[7] P. Fernandes, B. Plateau. **Stochastic Automata Networks (SAN) - Modelling and Evaluation**. Proceedings of the second Process Algebras and Performance Modelling workshop. Ed. U. Herzog, M. Rettelbach, Univ. Erlangen, vol. 21-22, Julho 1994, pp. 127-136.

[8] P. Fernandes, B. Plateau, W. J. Stewart. **Efficient descriptor-vector multiplication in stochastic automata networks**. Journal of the ACM, vol. 45, no. 3, 1998, pp. 381-414.

[9] P. Fernandes, R. Presotto, A. Sales, T. Webber. **An Alternative Algorithm to Multiply a Vector by a Kronecker Represented Descriptor**. 21^{th} Annual UK Performance Engineering Workshop, Newcastle, Julho 2005.

[10] J. E. Hopcroft, J. D. Ullman. **Introduction to automata theory, languages and computation**. Addison-Wesley, EUA, 1979.

[11] H. Schneider, G. P. Barker. **Matrices and Linear Algebra**. Holt, Rinehart and Winston, EUA, 1973.

[12] J. R. Norris. **Markov Chains**. Cambridge University Press, EUA, 1998.

[13] B. Plateau. **De l'Evaluation du Parallélisme et de la Synchronisation**. Paris-Sud, Orsay, 1984.

[14] B. Plateau. **On the stochastic structure of parallelism and synchronization models for distributed algorithms**. Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems, pp. 147-154. ACM Press - Austin, Texas, EUA, 1985.

[15] B. Plateau, K. Atif. **Stochastic automata networks for modelling parallel systems**. IEEE transactions on software engineering, vol. 17, no. 10, 1991, pp. 1093-1108.

[16] B. Plateau, J.M. Fourneau, K.H. Lee. **PEPS: A package for solving complex Markov model of parallel systems**. Modeling Techniques and Tools for Computer Performance Evaluation, Espanha, 1988.

[17] W. J. Stewart. **Introduction to the numerical solution of Markov chains**. Princeton University Press, 1994.

[18] T. Webber. **Alternativas para o Tratamento Numérico Otimizado da Multiplicação Vetor-Descritor**. PUCRS-FACIN-PPGCC, Porto Alegre, 2003.

Apêndice A

Álgebra Tensorial

São apresentados nas seções a seguir os conceitos de Álgebra Tensorial Clássica [5] e de Álgebra Tensorial Generalizada [6], necessários para o entendimento das SAN. A primeira seção introduz os conceitos de Álgebra Tensorial Clássica (ATC) e cita suas principais propriedades. Na segunda seção introduz-se a Álgebra Tensorial Generalizada (ATG) e cita as propriedades de interesse para o formalismo de SAN[1].

A.1 Álgebra Tensorial Clássica

A Álgebra Tensorial Clássica é definida por dois operadores matriciais:

- produto tensorial (também chamado de *Produto de Kronecker*);
- soma tensorial.

A notação utilizada na definição dos operadores da ATC e de suas propriedades é introduzida à medida em que se tornar necessária.

Sejam:

$\mathbb{N}$	conjunto dos números naturais;
$\mathbb{R}$	conjunto dos números reais;
$[a..b]$	subconjunto de $\mathbb{N}$ que contém todos os valores de a até b (a e b incluídos);
$[a, b]$	subconjunto de $\mathbb{R}$ que contém todos os valores de a até b (a e b incluídos);
$]a, b]$	subconjunto de $\mathbb{R}$ que contém todos os valores de a até b (a excluído, b incluído).

A.1.1 Produto Tensorial

O produto tensorial de duas matrizes A e B, de dimensões $(\alpha_1 \times \alpha_2)$ e $(\beta_1 \times \beta_2)$, respectivamente, é uma matriz de dimensões $(\alpha_1\beta_1 \times \alpha_2\beta_2)$. Essa matriz pode ser vista como uma matriz constituída de $\alpha_1 \times \alpha_2$ blocos, cada um de dimensão $\beta_1 \times \beta_2$. A definição de cada um

[1] Uma discussão mais abrangente sobre ATC e ATG, com demonstração das propriedades não-triviais, pode ser vista em [8].

dos elementos da matriz resultante é feita levando-se em conta a qual bloco o referido elemento pertence e a sua posição interna dentro desse bloco.

Sejam:

A matriz A;

$A \otimes B$ produto tensorial das matrizes A e B;

$A \times B$ produto (convencional) das matrizes A e B;

a_{ij} elemento da i-ésima linha e j-ésima coluna da matriz A;

$a_{[ik][jl]}$ elemento da k-ésima linha do i-ésimo bloco horizontal e da l-ésima coluna do j-ésimo bloco vertical da matriz A.

Sejam, por exemplo, as duas matrizes A e B:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{pmatrix}$$

O produto tensorial definido por $C = A \otimes B$ é igual a:

$$C = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix}$$

$$C = \left(\begin{array}{cccc|cccc} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{11}b_{14} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} & a_{12}b_{14} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{11}b_{24} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} & a_{12}b_{24} \\ a_{11}b_{31} & a_{11}b_{32} & a_{11}b_{33} & a_{11}b_{34} & a_{12}b_{31} & a_{12}b_{32} & a_{12}b_{33} & a_{12}b_{34} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{21}b_{14} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} & a_{22}b_{14} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{21}b_{24} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} & a_{22}b_{24} \\ a_{21}b_{31} & a_{21}b_{32} & a_{21}b_{33} & a_{21}b_{34} & a_{22}b_{31} & a_{22}b_{32} & a_{22}b_{33} & a_{22}b_{34} \end{array}\right)$$

Neste exemplo, o elemento c_{47} ($c_{47} = a_{22}b_{13}$) encontra-se dentro do bloco $(2,2)$ e sua posição interna neste bloco é $(1,3)$. O produto tensorial $C = A \otimes B$ é definido algebricamente pela atribuição do valor $a_{ij}b_{kl}$ ao elemento de posição (k,l) do bloco (i,j), *i.e.*:

$$\begin{aligned} c_{[ik][jl]} &= a_{ij}b_{kl} \\ &\text{onde } i \in [1..\alpha_1],\ j \in [1..\alpha_2],\ k \in [1..\beta_1] \text{ e } l \in [1..\beta_2] \end{aligned} \tag{A.1}$$

Essa representação dos elementos da matriz correspondente ao produto tensorial induz uma relação de ordem sobre os elementos $c_{[ik][jl]}$, que é a ordem lexicográfica das duplas de índice $([ik][jl])$.

Fator Normal

Um caso particular de produto tensorial é o produto tensorial de uma matriz quadrada por uma matriz identidade. Esse produto tensorial é denominado *fator normal*. Com uma matriz quadrada A e uma matriz identidade (de dimensão n) I_n, dois fatores normais são possíveis: $(A \otimes I_n)$ e $(I_n \otimes A)$.

Sejam a matriz A do exemplo anterior e uma matriz identidade de dimensão 3:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Os fatores normais $A \otimes I_3$ e $I_3 \otimes A$ são:

$$A \otimes I_3 = \left(\begin{array}{ccc|ccc} a_{11} & 0 & 0 & a_{12} & 0 & 0 \\ 0 & a_{11} & 0 & 0 & a_{12} & 0 \\ 0 & 0 & a_{11} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22} & 0 & 0 \\ 0 & a_{21} & 0 & 0 & a_{22} & 0 \\ 0 & 0 & a_{21} & 0 & 0 & a_{22} \end{array}\right) \qquad I_3 \otimes A = \left(\begin{array}{cc|cc|cc} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & a_{11} & a_{12} & 0 & 0 \\ 0 & 0 & a_{21} & a_{22} & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & a_{11} & a_{12} \\ 0 & 0 & 0 & 0 & a_{21} & a_{22} \end{array}\right)$$

Um caso ainda mais particular de produto tensorial é o produto tensorial de duas matrizes identidade. O resultado é uma matriz identidade cuja dimensão é igual ao produto das dimensões das duas matrizes, *i.e.*:

$$I_n \otimes I_m = I_m \otimes I_n = I_{nm}$$

A.1.2 Soma Tensorial

Diferentemente do produto tensorial, que é definido para matrizes quaisquer, a soma tensorial é definida somente para matrizes quadradas.

Sejam:

$A \oplus B$ soma tensorial das matrizes quadradas A e B;

$A + B$ soma (convencional) das matrizes A e B;

n_A dimensão (número de linhas e de colunas) da matriz quadrada A;

δ_{ij} elemento da i-ésima linha e da j-ésima coluna de uma matriz identidade ($\delta_{ij} = 1$ se $i = j$ e $\delta_{ij} = 0$ se $i \neq j$).

A soma tensorial de duas matrizes quadradas A e B é definida como a soma (convencional) dos fatores normais das duas matrizes segundo a fórmula:

$$A \oplus B = (A \otimes I_{n_B}) + (I_{n_A} \otimes B) \tag{A.2}$$

Sejam por exemplo as matrizes A e B dadas por:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

A soma tensorial dada por $C = A \oplus B$ é igual a:

$$C = A \oplus B = (A \otimes I_3) + (I_2 \otimes B)$$

$$C = \left(\begin{array}{ccc|ccc} a_{11} & 0 & 0 & a_{12} & 0 & 0 \\ 0 & a_{11} & 0 & 0 & a_{12} & 0 \\ 0 & 0 & a_{11} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22} & 0 & 0 \\ 0 & a_{21} & 0 & 0 & a_{22} & 0 \\ 0 & 0 & a_{21} & 0 & 0 & a_{22} \end{array}\right) + \left(\begin{array}{ccc|ccc} b_{11} & b_{12} & b_{13} & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & 0 & 0 & 0 \\ b_{31} & b_{32} & b_{33} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & b_{11} & b_{12} & b_{13} \\ 0 & 0 & 0 & b_{21} & b_{22} & b_{23} \\ 0 & 0 & 0 & b_{31} & b_{32} & b_{33} \end{array}\right)$$

$$C = \left(\begin{array}{ccc|ccc} a_{11}+b_{11} & b_{12} & b_{13} & a_{12} & 0 & 0 \\ b_{21} & a_{11}+b_{22} & b_{23} & 0 & a_{12} & 0 \\ b_{31} & b_{32} & a_{11}+b_{33} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22}+b_{11} & b_{12} & b_{13} \\ 0 & a_{21} & 0 & b_{21} & a_{22}+b_{22} & b_{23} \\ 0 & 0 & a_{21} & b_{31} & b_{32} & a_{22}+b_{33} \end{array}\right)$$

A soma tensorial $C = A \oplus B$ é definida algebricamente pela atribuição do valor $a_{ij}\delta_{kl} + \delta_{ij}b_{kl}$ ao elemento de posição (k, l) do bloco (i, j), *i.e.*:

$$c_{[ik][jl]} = a_{ij}\delta_{kl} + \delta_{ij}b_{kl} \qquad \text{onde } i, j \in [1..n_A] \text{ e } k, l \in [1..n_B] \tag{A.3}$$

O operador produto tensorial ($\otimes$) tem prioridade sobre o operador soma tensorial ($\oplus$) e os dois operadores tensoriais têm prioridade sobre os operadores tradicionais de multiplicação e adição de matrizes ($\times$ e $+$).

A.1.3 Propriedades da ATC

As propriedades da ATC de interesse para as SAN são listadas a seguir. Suas demonstrações podem ser vistas em [8].

- Associatividade:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{A.4}$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \tag{A.5}$$

- Distributividade com relação à soma:

$$(A + B) \otimes (C + D) = (A \otimes C) + (B \otimes C) + (A \otimes D) + (B \otimes D) \tag{A.6}$$

- Compatibilidade com a multiplicação:

$$(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D) \tag{A.7}$$

- Compatibilidade com a transposição de matrizes:

$$(A \otimes B)^T = A^T \otimes B^T \tag{A.8}$$

- Compatibilidade com a inversão de matrizes (se A e B são matrizes inversíveis):

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{A.9}$$

- Decomposição em fatores normais:

$$A \otimes B = (A \otimes I_{n_B}) \times (I_{n_A} \otimes B) \tag{A.10}$$

- Distributividade com relação à multiplicação pela matriz identidade:

$$(A \times B) \otimes I_n = (A \otimes I_n) \times (B \otimes I_n) \tag{A.11}$$

$$I_n \otimes (A \times B) = (I_n \otimes A) \times (I_n \otimes B) \tag{A.12}$$

- Comutatividade dos fatores normais:

$$(A \otimes I_{n_B}) \times (I_{n_A} \otimes B) = (I_{n_A} \otimes B) \times (A \otimes I_{n_B}) \tag{A.13}$$

A.2 Álgebra Tensorial Generalizada

A álgebra tensorial generalizada é uma extensão da álgebra tensorial clássica. A diferença fundamental da ATG com relação à ATC é a introdução do conceito de *elementos funcionais*. Doravante, uma matriz pode ser composta de elementos constantes (pertencentes a $\mathbb{R}$) ou de elementos funcionais. Um elemento funcional é uma função real dos índices de linha de uma ou mais matrizes[2].

Um elemento funcional b é dito *dependente* da matriz A se algum índice de linha da matriz A pertencer ao conjunto de parâmetros desse elemento funcional. Por abuso de linguagem, denomina-se *parâmetro* de um elemento funcional toda matriz da qual o elemento funcional é dependente. Uma matriz que contém ao menos um elemento funcional dependente da matriz A é dita *dependente* da matriz A. Os *parâmetros* de uma matriz são a união dos parâmetros de todos seus elementos funcionais.

Assim como a ATC, a ATG é definida por dois operadores matriciais:

- produto tensorial generalizado;
- soma tensorial generalizada.

A notação definida na seção A.1 continua sendo válida para as *matrizes constantes* (*i.e.*, matrizes sem elementos funcionais). As matrizes com elementos funcionais, denominadas *matrizes funcionais*, são descritas com o uso da notação seguinte:

Sejam:

a_k índice de linha k da matriz A;

$A(\mathcal{B},\mathcal{C})$ matriz funcional A que possui como parâmetros as matrizes B e C;

$a_{ij}(\mathcal{B},\mathcal{C})$ elemento funcional (i,j) da matriz $A(\mathcal{B},\mathcal{C})$;

$A(b_k,\mathcal{C})$ matriz funcional $A(\mathcal{B},\mathcal{C})$ na qual o índice de linha da matriz B já é conhecido e igual a k (essa matriz pode ser considerada dependente da matriz C somente);

[2] *i.e.*, o domínio dessa função é $\mathbb{R}^n$ e seu contra–domínio é $\mathbb{R}$

$a_{ij}(b_k, \mathcal{C})$ elemento funcional (i, j) da matriz $A(b_k, \mathcal{C})$;

$A(b_k, c_l)$ matriz funcional $A(\mathcal{B}, \mathcal{C})$ na qual os índices de linha das matrizes B e C já são conhecidos e iguais a k e l respectivamente (uma vez que todos os parâmetros da matriz são conhecidos, pode ser considerada uma matriz constante);

$a_{ij}(b_k, c_l)$ elemento constante (elemento funcional de valor determinado) (i, j) da matriz $A(b_k, c_l)$;

$\ell_k(A)$ matriz com todos os elementos iguais a zero, exceto aqueles pertencentes à linha k que é igual à linha k da matriz A ($A = \sum_{k=1}^{n_A} \ell_k(A)$);

$A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A})$ produto tensorial generalizado entre as matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$;

$A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A})$ soma tensorial generalizada entre as matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$.

A.2.1 Produto Tensorial Generalizado

Sejam por exemplo duas matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$ dadas por:

$$A(\mathcal{B}) = \begin{pmatrix} a_{11}[\mathcal{B}] & a_{12}[\mathcal{B}] \\ a_{21}[\mathcal{B}] & a_{22}[\mathcal{B}] \end{pmatrix} \qquad B(\mathcal{A}) = \begin{pmatrix} b_{11}[\mathcal{A}] & b_{12}[\mathcal{A}] & b_{13}[\mathcal{A}] \\ b_{21}[\mathcal{A}] & b_{22}[\mathcal{A}] & b_{23}[\mathcal{A}] \\ b_{31}[\mathcal{A}] & b_{32}[\mathcal{A}] & b_{33}[\mathcal{A}] \end{pmatrix}$$

O produto tensorial definido por $C = A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A})$ é igual a:

$$C = \left(\begin{array}{ccc|ccc} a_{11}(b_1)b_{11}(a_1) & a_{11}(b_1)b_{12}(a_1) & a_{11}(b_1)b_{13}(a_1) & a_{12}(b_1)b_{11}(a_1) & a_{12}(b_1)b_{12}(a_1) & a_{12}(b_1)b_{13}(a_1) \\ a_{11}(b_2)b_{21}(a_1) & a_{11}(b_2)b_{22}(a_1) & a_{11}(b_2)b_{23}(a_1) & a_{12}(b_2)b_{21}(a_1) & a_{12}(b_2)b_{22}(a_1) & a_{12}(b_2)b_{23}(a_1) \\ a_{11}(b_3)b_{31}(a_1) & a_{11}(b_3)b_{32}(a_1) & a_{11}(b_3)b_{33}(a_1) & a_{12}(b_3)b_{31}(a_1) & a_{12}(b_3)b_{32}(a_1) & a_{12}(b_3)b_{33}(a_1) \\ \hline a_{21}(b_1)b_{11}(a_2) & a_{21}(b_1)b_{12}(a_2) & a_{21}(b_1)b_{13}(a_2) & a_{22}(b_1)b_{11}(a_2) & a_{22}(b_1)b_{12}(a_2) & a_{22}(b_1)b_{13}(a_2) \\ a_{21}(b_2)b_{21}(a_2) & a_{21}(b_2)b_{22}(a_2) & a_{21}(b_2)b_{23}(a_2) & a_{22}(b_2)b_{21}(a_2) & a_{22}(b_2)b_{22}(a_2) & a_{22}(b_2)b_{23}(a_2) \\ a_{21}(b_3)b_{31}(a_2) & a_{21}(b_3)b_{32}(a_2) & a_{21}(b_3)b_{33}(a_2) & a_{22}(b_3)b_{31}(a_2) & a_{22}(b_3)b_{32}(a_2) & a_{22}(b_3)b_{33}(a_2) \end{array}\right)$$

O produto tensorial generalizado $C = A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A})$ é definido algebricamente pela atribuição do valor $a_{ij}(b_k)b_{kl}(a_i)$ ao elemento $c_{[ik][jl]}$, *i.e.*:

$$\begin{aligned} c_{[ik][jl]} &= a_{ij}(b_k)b_{kl}(a_i) \\ &\text{onde } i, j \in [1..n_A] \text{ e } k, l \in [1..n_B] \end{aligned} \tag{A.14}$$

A.2.2 Soma Tensorial Generalizada

A soma tensorial generalizada é definida utilizando-se o produto tensorial generalizado na equação A.2:

$$A \underset{g}{\oplus} B = (A \underset{g}{\otimes} I_{n_B}) + (I_{n_A} \underset{g}{\otimes} B) \tag{A.15}$$

Sejam as matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$ utilizadas para descrever o produto tensorial generalizado. A soma tensorial definida por $C = A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A})$ é igual a:

$$C = \left(\begin{array}{ccc|ccc} a_{11}(b_1)+b_{11}(a_1) & b_{12}(a_1) & b_{13}(a_1) & a_{12}(b_1) & 0 & 0 \\ b_{21}(a_1) & a_{11}(b_2)+b_{22}(a_1) & b_{23}(a_1) & 0 & a_{12}(b_2) & 0 \\ b_{31}(a_1) & b_{32}(a_1) & a_{11}(b_3)+b_{33}(a_1) & 0 & 0 & a_{12}(b_3) \\ \hline a_{21}(b_1) & 0 & 0 & a_{22}(b_1)+b_{11}(a_2) & b_{12}(a_2) & b_{13}(a_2) \\ 0 & a_{21}(b_2) & 0 & b_{21}(a_2) & a_{22}(b_2)+b_{22}(a_2) & b_{23}(a_2) \\ 0 & 0 & a_{21}(b_3) & b_{31}(a_2) & b_{32}(a_2) & a_{22}(b_3)+b_{33}(a_2) \end{array}\right)$$

A soma tensorial generalizada $C = A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A})$ é definida algebricamente pela atribuição do valor $a_{ij}(b_k)\delta_{kl} + b_{kl}(a_i)\delta_{ij}$ ao elemento $c_{[ik][jl]}$, *i.e.*:

$$\begin{aligned} c_{[ik][jl]} = a_{ij}(b_k)\delta_{kl} \;+\; b_{kl}(a_i)\delta_{ij} \\ \text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \end{aligned} \tag{A.16}$$

A.2.3 Propriedades

- Distributividade do produto tensorial generalizado com relação à soma convencional de matrizes:

$$\begin{aligned} [A(\mathcal{C},\mathcal{D}) + B(\mathcal{C},\mathcal{D})] \underset{g}{\otimes} [C(\mathcal{A},\mathcal{B}) + D(\mathcal{A},\mathcal{B})] = \\ A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \;+\; A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B}) + \\ B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \;+\; B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B}) \end{aligned} \tag{A.17}$$

- Associatividade do produto tensorial generalizado e da soma tensorial generalizada:

$$[\, A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} B(\mathcal{A},\mathcal{C}) \,] \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} [\, B(\mathcal{A},\mathcal{C}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \,] \tag{A.18}$$

$$[\, A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} B(\mathcal{A},\mathcal{C}) \,] \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} [\, B(\mathcal{A},\mathcal{C}) \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) \,] \tag{A.19}$$

- Distributividade com relação à multiplicação pela matriz identidade:

$$[\, A(\mathcal{C}) \times B(\mathcal{C}) \,] \underset{g}{\otimes} I_{n_C} = A(\mathcal{C}) \underset{g}{\otimes} I_{n_C} \;\times\; B(\mathcal{C}) \underset{g}{\otimes} I_{n_C} \tag{A.20}$$

$$[\, I_{n_C} \underset{g}{\otimes} A(\mathcal{C}) \,] \times B(\mathcal{C}) = I_{n_C} \underset{g}{\otimes} A(\mathcal{C}) \;\times\; I_{n_C} \underset{g}{\otimes} B(\mathcal{C}) \tag{A.21}$$

- Decomposição em fatores normais I:

$$A \underset{g}{\otimes} B(\mathcal{A}) = I_{n_A} \underset{g}{\otimes} B(\mathcal{A}) \;\times\; A \underset{g}{\otimes} I_{n_B} \tag{A.22}$$

- Decomposição em fatores normais II:

$$A(\mathcal{B}) \underset{g}{\otimes} B = A(\mathcal{B}) \underset{g}{\otimes} I_{n_B} \;\times\; I_{n_A} \underset{g}{\otimes} B \tag{A.23}$$

- Decomposição em produto tensorial clássico:

$$A \underset{g}{\otimes} B(\mathcal{A}) = \sum_{k=1}^{n_A} \ell_k(A) \otimes B(a_k) \tag{A.24}$$

Apêndice B

Cálculo do Custo Computacional

Veja a seguir o cálculo completo do Custo Computacional das técnicas *Shuffle* e *Slice*, com e sem as otimizações estudadas, para os dois exemplos de modelos SAN apresentados neste trabalho, representados pelas Figuras 4.1 e 4.2, e Tabelas 4.1 e 4.2 respectivamente.

B.1 Custo Computacional sem otimização da diagonal

Esta seção apresenta os cálculos do Custo Computacional das técnicas *Shuffle* e *Slice* desconsiderando a otimização da diagonal.

B.1.1 Custo Computacional para o primeiro exemplo

Para realizar o cálculo do Custo Computacional para o primeiro exemplo de modelo SAN, foram considerados os valores das Tabelas 4.3 e 4.4, além das fórmulas do custo computacional da técnica *Shuffle* e *Slice*.

Técnica *Shuffle*

O Custo Computacional da técnica *Shuffle* para este modelo SAN é calculado da seguinte forma:

Fórmula do custo computacional da técnica *Shuffle*.

$$\sum_{k=1}^{1+2E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right)$$

Cálculo do termo $\prod_{i=1}^{N} n_i$ (não depende da parte local ou sincronizante do modelo).

$$4 \times 2 \times 4 \times 3 \times 5 \times 4 \quad = \quad 1.920$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para a parte local do modelo.

$$\frac{2}{4}+\frac{0}{2}+\frac{2}{4}+\frac{0}{3}+\frac{5}{5}+\frac{4}{4} \;=\; \frac{10+0+10+0+20+20}{20} \;=\; \frac{60}{20} \;=\; 3$$

Custo computacional total para a parte local do modelo.

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \;=\; 1.920 \times 3 \;=\; 5.760$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_8.

$$\frac{1}{4}+\frac{2}{2}+\frac{3}{4}+\frac{3}{3}+\frac{5}{5}+\frac{4}{4} \;=\; \frac{1}{4}+1+\frac{1}{4}+1+1+1 \;=\; \frac{4}{4}+4 \;=\; 5$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_9.

$$\frac{1}{4}+\frac{1}{2}+\frac{4}{4}+\frac{3}{3}+\frac{5}{5}+\frac{4}{4} \;=\; \frac{1}{4}+\frac{1}{2}+1+1+1+1 \;=\; \frac{3}{4}+4 \;=\; \frac{19}{4}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{10}.

$$\frac{1}{4}+\frac{2}{2}+\frac{4}{4}+\frac{2}{3}+\frac{5}{5}+\frac{4}{4} \;=\; \frac{1}{4}+1+1+\frac{2}{3}+1+1 \;=\; \frac{11}{12}+4 \;=\; \frac{59}{12}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{11}.

$$\frac{1}{4}+\frac{2}{2}+\frac{4}{4}+\frac{2}{3}+\frac{5}{5}+\frac{4}{4} \;=\; \frac{1}{4}+1+1+\frac{2}{3}+1+1 \;=\; \frac{11}{12}+4 \;=\; \frac{59}{12}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{12}.

$$\frac{1}{4}+\frac{1}{2}+\frac{4}{4}+\frac{3}{3}+\frac{5}{5}+\frac{4}{4} \;=\; \frac{1}{4}+\frac{1}{2}+1+1+1+1 \;=\; \frac{3}{4}+4 \;=\; \frac{19}{4}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{13}.

$$\frac{4}{4}+\frac{2}{2}+\frac{1}{4}+\frac{3}{3}+\frac{1}{5}+\frac{3}{4} \;=\; 1+1+\frac{1}{4}+1+\frac{1}{5}+\frac{3}{4} \;=\; \frac{24}{20}+3 \;=\; \frac{84}{20} \;=\; \frac{21}{5}$$

Cálculo do termo $\sum_{i=1}^{N}\frac{nz_i}{n_i}$ para o evento sincronizante e_{14}.

$$\frac{4}{4}+\frac{2}{2}+\frac{4}{4}+\frac{3}{3}+\frac{1}{5}+\frac{4}{4} \;=\; 1+1+1+1+\frac{1}{5}+1 \;=\; \frac{1}{5}+5 \;=\; \frac{26}{5}$$

Cálculo do termo $\sum_{i=1}^{N}\frac{nz_i}{n_i}$ para o evento sincronizante e_{15}.

$$\frac{4}{4}+\frac{2}{2}+\frac{4}{4}+\frac{2}{3}+\frac{1}{5}+\frac{4}{4} \;=\; 1+1+1+\frac{2}{3}+\frac{1}{5}+1 \;=\; \frac{13}{15}+4 \;=\; \frac{73}{15}$$

Cálculo do termo $\sum_{i=1}^{N}\frac{nz_i}{n_i}$ para o evento sincronizante e_{16}.

$$\frac{4}{4}+\frac{1}{2}+\frac{4}{4}+\frac{3}{3}+\frac{1}{5}+\frac{4}{4} \;=\; 1+\frac{1}{2}+1+1+\frac{1}{5}+1 \;=\; \frac{7}{10}+4 \;=\; \frac{47}{10}$$

Custo computacional para a parte sincronizante positiva do modelo.

$$\sum_{k=1}^{E}\left(\prod_{i=1}^{N}n_i\times\sum_{i=1}^{N}\frac{nz_i}{n_i}\right) \;=\; 1.920\times\left(5+\frac{19}{4}+\frac{59}{12}+\frac{59}{12}+\frac{19}{4}+\frac{21}{5}+\frac{26}{5}+\frac{73}{15}+\frac{47}{10}\right) \;=$$

$$1.920\times\left(\frac{300+285+295+295+285+252+312+292+282}{60}\right) \;=$$

$$1.920\times\frac{2.598}{60} \;=\; 32\times 2.598 \;=\; 83.136$$

Sabendo que a parte sincronizante negativa deste modelo tem o mesmo custo da parte positiva,
o custo computacional total para a parte sincronizante deste modelo é:

$$\sum_{k=1}^{2E}\left(\prod_{i=1}^{N}n_i\times\sum_{i=1}^{N}\frac{nz_i}{n_i}\right) \;=\; 83.136+83.136 \;=\; 166.272$$

Logo, o custo computacional total para uma iteração da resolução deste modelo SAN é:

$$\sum_{k=1}^{1+2E}\left(\prod_{i=1}^{N}n_i\times\sum_{i=1}^{N}\frac{nz_i}{n_i}\right) \;=\; 5.760+166.272 \;=\; 171.992$$

Técnica *Slice*

Para a técnica *Slice*, o Custo Computacional para este modelo SAN é calculado da seguinte forma:

Fórmula do custo computacional da técnica *Slice*.

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \left[(N-2) + n_N + nz_N\right]\right)$$

Custo computacional total para a parte local do modelo (igual ao *Shuffle*).

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \quad = \quad 1.920 \times 3 \quad = \quad 5.760$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_8.

$$1 \times 2 \times 3 \times 3 \times 5 \quad = \quad 90$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_8.

$$(6-2) + 4 + 4 \quad = \quad 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_9.

$$1 \times 1 \times 4 \times 3 \times 5 \quad = \quad 60$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_9.

$$(6-2) + 4 + 4 \quad = \quad 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{10}.

$$1 \times 2 \times 4 \times 2 \times 5 \quad = \quad 80$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{10}.

$$(6-2)+4+4 \;=\; 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{11}.

$$1\times 2\times 4\times 2\times 5 \;=\; 80$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{11}.

$$(6-2)+4+4 \;=\; 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{12}.

$$1\times 1\times 4\times 3\times 5 \;=\; 60$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{12}.

$$(6-2)+4+4 \;=\; 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{13}.

$$4\times 2\times 1\times 3\times 1 \;=\; 24$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{13}.

$$(6-2)+4+3 \;=\; 11$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{14}.

$$4\times 2\times 4\times 3\times 1 \;=\; 96$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{14}.

$$(6-2)+4+4 \;=\; 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{15}.

$$4\times 2\times 4\times 2\times 1 \;=\; 64$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{15}.

$$(6-2)+4+4 \;=\; 12$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{16}.

$$4\times 1\times 4\times 3\times 1 \;=\; 48$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_{16}.

$$(6-2)+4+4 \;=\; 12$$

Custo computacional para a parte sincronizante positiva do modelo.

$$\sum_{k=1}^{E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right) \;=$$

$$(90\times 12)+(60\times 12)+(80\times 12)+(80\times 12)+(60\times 12)+(24\times 11)+(96\times 12)+(64\times 12)+(48\times 12) \;=$$

$$1.080+720+960+960+720+264+1.152+768+576 \;=\; 7.200$$

Lembrando que a parte sincronizante positiva e negativa deste modelo possuem o mesmo custo computacional,

o custo computacional total para a parte sincronizante deste modelo é:

$$\sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right) \;=\; 7.200+7.200 \;=\; 14.400$$

Logo, o custo computacional total para uma iteração da resolução deste modelo SAN é:

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right) \;=\; 5.760 + 14.400 \;=\; 20.160$$

B.1.2 Custo Computacional para o segundo exemplo

Para realizar o cálculo do Custo Computacional para o segundo exemplo de modelo SAN , foram considerados os valores das Tabelas 4.5, 4.6 e 4.7.

Técnica *Shuffle*

O Custo Computacional da técnica *Shuffle* para este modelo SAN é calculado da seguinte maneira:

Fórmula do custo computacional da técnica *Shuffle*.

$$\sum_{k=1}^{1+2E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right)$$

Cálculo do termo $\prod_{i=1}^{N} n_i$ (não depende da parte local ou sincronizante do modelo).

$$4 \times 4 \times 3 \times 6 \times 4 \times 2 \;=\; 2.304$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para a parte local do modelo.

$$\frac{2}{4} + \frac{4}{4} + \frac{0}{3} + \frac{4}{6} + \frac{2}{4} + \frac{0}{2} \;=\; \frac{6+12+0+8+6+0}{12} \;=\; \frac{32}{12} \;=\; \frac{8}{3}$$

Custo computacional total para a parte local do modelo.

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \;=\; 2.304 \times \frac{8}{3} \;=\; \frac{18.432}{3} \;=\; 6.144$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_7 (matriz positiva).

$$\frac{3}{4} + \frac{4}{4} + \frac{3}{3} + \frac{6}{6} + \frac{4}{4} + \frac{2}{2} \;=\; \frac{3}{4} + 1 + 1 + 1 + 1 + 1 \;=\; \frac{3}{4} + 5 \;=\; \frac{23}{4}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_8 (matriz positiva).

$$\frac{2}{4}+\frac{4}{4}+\frac{2}{3}+\frac{6}{6}+\frac{4}{4}+\frac{2}{2} = \frac{2}{4}+1+\frac{2}{3}+1+1+1 = \frac{14}{12}+4 = \frac{62}{12} = \frac{31}{6}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_9.

$$\frac{4}{4}+\frac{4}{4}+\frac{2}{3}+\frac{6}{6}+\frac{4}{4}+\frac{1}{2} = 1+1+\frac{2}{3}+1+1+\frac{1}{2} = \frac{7}{6}+4 = \frac{31}{6}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{10}.

$$\frac{4}{4}+\frac{4}{4}+\frac{3}{3}+\frac{2}{6}+\frac{4}{4}+\frac{1}{2} = 1+1+1+\frac{2}{6}+1+\frac{1}{2} = \frac{5}{6}+4 = \frac{29}{6}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{11}.

$$\frac{4}{4}+\frac{4}{4}+\frac{3}{3}+\frac{2}{6}+\frac{2}{4}+\frac{2}{2} = 1+1+1+\frac{2}{6}+\frac{2}{4}+1 = \frac{10}{12}+4 = \frac{58}{12} = \frac{29}{6}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{12}.

$$\frac{4}{4}+\frac{4}{4}+\frac{3}{3}+\frac{2}{6}+\frac{2}{4}+\frac{2}{2} = 1+1+1+\frac{2}{6}+\frac{2}{4}+1 = \frac{10}{12}+4 = \frac{58}{12} = \frac{29}{6}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{13}.

$$\frac{4}{4}+\frac{4}{4}+\frac{3}{3}+\frac{1}{6}+\frac{1}{4}+\frac{2}{2} = 1+1+1+\frac{1}{6}+\frac{1}{4}+1 = \frac{5}{12}+4 = \frac{53}{12}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_{14}.

$$\frac{4}{4}+\frac{4}{4}+\frac{3}{3}+\frac{6}{6}+\frac{1}{4}+\frac{1}{2} = 1+1+1+1+\frac{1}{4}+\frac{1}{2} = \frac{3}{4}+4 = \frac{19}{4}$$

Apenas os eventos e_7 e e_8 possuem valores diferentes para as suas matrizes positivas e negativas, pois nos autômatos $A^{(1)}$ e $A^{(3)}$ existem transições que partem do mesmo estado para

outros, tendo estes eventos associados. Dessa forma, as matrizes negativas destes eventos possuirão menos elementos não-nulos, uma vez que na matriz negativa as taxas destes eventos são somadas e posicionadas na diagonal principal da matriz com sinal negativo, realizando assim o ajuste necessário ao modelo SAN. Por este motivo, é preciso recalcular o termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para as matrizes negativas destes eventos, como segue:

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_7 (matriz negativa).

$$\frac{2}{4}+\frac{4}{4}+\frac{3}{3}+\frac{6}{6}+\frac{4}{4}+\frac{2}{2} \;=\; \frac{2}{4}+1+1+1+1+1 \;=\; \frac{1}{2}+5 \;=\; \frac{11}{2}$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$ para o evento sincronizante e_8 (matriz negativa).

$$\frac{2}{4}+\frac{4}{4}+\frac{1}{3}+\frac{6}{6}+\frac{4}{4}+\frac{2}{2} \;=\; \frac{2}{4}+1+\frac{1}{3}+1+1+1 \;=\; \frac{10}{12}+4 \;=\; \frac{58}{12} \;=\; \frac{29}{6}$$

Custo computacional para a parte sincronizante positiva do modelo.

$$\sum_{k=1}^{E}\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) \;=\; 2.304 \times \left(\frac{23}{4}+\frac{31}{6}+\frac{31}{6}+\frac{29}{6}+\frac{29}{6}+\frac{29}{6}+\frac{53}{12}+\frac{19}{4}\right) \;=$$

$$2.304 \times \left(\frac{69+62+62+58+58+58+53+57}{12}\right) \;=$$

$$2.304 \times \frac{477}{12} \;=\; 192 \times 477 \;=\; 91.584$$

Custo computacional para a parte sincronizante negativa do modelo.

$$\sum_{k=1}^{E}\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) \;=\; 2.304 \times \left(\frac{11}{2}+\frac{29}{6}+\frac{31}{6}+\frac{29}{6}+\frac{29}{6}+\frac{29}{6}+\frac{53}{12}+\frac{19}{4}\right) \;=$$

$$2.304 \times \left(\frac{66+58+62+58+58+58+53+57}{12}\right) \;=$$

$$2.304 \times \frac{470}{12} \;=\; 192 \times 470 \;=\; 90.240$$

Custo computacional total para a parte sincronizante do modelo.

$$\sum_{k=1}^{2E}\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) \;=\; 91.584 + 90.240 \;=\; 181.824$$

Logo, o custo computacional total para uma iteração da resolução deste modelo SAN é:

$$\sum_{k=1}^{1+2E}\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) \;=\; 6.144 + 181.824 \;=\; 187.968$$

Técnica *Slice*

O Custo Computacional da técnica *Slice* para este modelo SAN é calculado da seguinte forma:

Fórmula do custo computacional da técnica *Slice*.

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right)$$

Custo computacional total para a parte local do modelo (igual ao *Shuffle*).

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i} \;=\; 2.304 \times \frac{8}{3} \;=\; \frac{18.432}{3} \;=\; 6.144$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_7 (matriz positiva).

$$3 \times 4 \times 3 \times 6 \times 4 \;=\; 864$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_7 (matriz positiva).

$$(6-2) + 2 + 2 \;=\; 8$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_8 (matriz positiva).

$$2 \times 4 \times 2 \times 6 \times 4 \;=\; 384$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_8 (matriz positiva).

$$(6-2) + 2 + 2 \;=\; 8$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_9.

$$4 \times 4 \times 2 \times 6 \times 4 \quad = \quad 768$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_9.

$$(6-2) + 2 + 1 \quad = \quad 7$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{10}.

$$4 \times 4 \times 3 \times 2 \times 4 \quad = \quad 384$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_{10}.

$$(6-2) + 2 + 1 \quad = \quad 7$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{11}.

$$4 \times 4 \times 3 \times 2 \times 2 \quad = \quad 192$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_{11}.

$$(6-2) + 2 + 2 \quad = \quad 8$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{12}.

$$4 \times 4 \times 3 \times 2 \times 2 \quad = \quad 192$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_{12}.

$$(6-2) + 2 + 2 \quad = \quad 8$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{13}.

$$4 \times 4 \times 3 \times 1 \times 1 \quad = \quad 48$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_{13}.

$$(6-2) + 2 + 2 \quad = \quad 8$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_{14}.

$$4 \times 4 \times 3 \times 6 \times 1 \quad = \quad 288$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_{14}.

$$(6-2) + 2 + 1 \quad = \quad 7$$

Mais uma vez deve-se recalcular os termos $\prod_{i=1}^{N-1} nz_i$ e $(N-2) + n_N + nz_N$ pois os eventos e_7 e e_8 possuem valores diferentes para as suas matrizes positivas e negativas. Sendo assim, o cálculo destes termos para as matrizes negativas dos eventos e_7 e e_8 são realizados da seguinte maneira:

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_7 (matriz negativa).

$$2 \times 4 \times 3 \times 6 \times 4 \quad = \quad 576$$

Cálculo do termo $(N-2) + n_N + nz_N$ para o evento sincronizante e_7 (matriz negativa).

$$(6-2) + 2 + 2 \quad = \quad 8$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i$ para o evento sincronizante e_8 (matriz negativa).

$$2 \times 4 \times 1 \times 6 \times 4 \quad = \quad 192$$

Cálculo do termo $(N-2)+n_N+nz_N$ para o evento sincronizante e_8 (matriz negativa).

$$(6-2)+2+2 \;=\; 8$$

Custo computacional para a parte sincronizante positiva do modelo.

$$\sum_{k=1}^{E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right) \;=$$

$$(864\times 8)+(384\times 8)+(768\times 7)+(384\times 7)+(192\times 8)+(192\times 8)+(48\times 8)+(288\times 7) \;=$$

$$6.912+3.072+5.376+2.688+1.536+1.536+384+2.016 \;=\; 23.520$$

Custo computacional para a parte sincronizante negativa do modelo.

$$\sum_{k=1}^{E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right) \;=$$

$$(576\times 8)+(192\times 8)+(768\times 7)+(384\times 7)+(192\times 8)+(192\times 8)+(48\times 8)+(288\times 7) \;=$$

$$4.608+1.536+5.376+2.688+1.536+1.536+384+2.016 \;=\; 19.680$$

Custo computacional total para a parte sincronizante do modelo.

$$\sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right) \;=\; 23.520+19.680 \;=\; 43.200$$

Logo, o custo computacional total para uma iteração da resolução deste modelo SAN é:

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N}\frac{nz_i}{n_i}\right)+\sum_{k=1}^{2E}\left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2)+n_N+nz_N\Big]\right) \;=\; 6.144+43.200 \;=\; 49.344$$

B.1.3 Custo Computacional para *Slice* com Otimização dos Fatores

Considerando a otimização dos Fatores Normais Unitários Aditivos (AUNF) proposta para a técnica *Slice*, o custo computacional desta técnica para os dois exemplos estudados é calculado conforme apresentado nas seções seguintes.

Primeiro exemplo

Com a otimização dos Fatores na técnica *Slice*, o custo computacional da parte sincronizante do primeiro exemplo de modelo SAN é calculado da seguinte forma:

Custo computacional otimizado da parte sincronizante positiva do modelo.

$$\sum_{k=1}^{E}\left(\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right] + \prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]\right)$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_8.

$$(1\times 2) + (1\times 2\times 3) + (1\times 2\times 3\times 3) + (1\times 2\times 3\times 3\times 5) \;=\; 2+6+18+90 \;=\; 116$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_8.

$$(1\times 2\times 3\times 3\times 5)\times(4+4) \;=\; 90\times 8 \;=\; 720$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_9.

$$(1\times 1) + (1\times 1\times 4) + (1\times 1\times 4\times 3) + (1\times 1\times 4\times 3\times 5) \;=\; 1+4+12+60 \;=\; 77$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_9.

$$(1\times 1\times 4\times 3\times 5)\times(4+4) \;=\; 60\times 8 \;=\; 480$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{10}.

$$(1\times 2) + (1\times 2\times 4) + (1\times 2\times 4\times 2) + (1\times 2\times 4\times 2\times 5) \;=\; 2+8+16+80 \;=\; 106$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{10}.

$$(1\times 2\times 4\times 2\times 5)\times(4+4) \;=\; 80\times 8 \;=\; 640$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{11}.

$$(1\times 2)+(1\times 2\times 4)+(1\times 2\times 4\times 2)+(1\times 2\times 4\times 2\times 5) \;=\; 2+8+16+80 \;=\; 106$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{11}.

$$(1\times 2\times 4\times 2\times 5)\times(4+4) \;=\; 80\times 8 \;=\; 640$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{12}.

$$(1\times 1)+(1\times 1\times 4)+(1\times 1\times 4\times 3)+(1\times 1\times 4\times 3\times 5) \;=\; 1+4+12+60 \;=\; 77$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{12}.

$$(1\times 1\times 4\times 3\times 5)\times(4+4) \;=\; 60\times 8 \;=\; 480$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{13}.

$$(4\times 2)+(4\times 2\times 1)+(4\times 2\times 1\times 3)+(4\times 2\times 1\times 3\times 1) \;=\; 8+8+24+24 \;=\; 64$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{13}.

$$(4\times 2\times 1\times 3\times 1)\times(4+3) \;=\; 24\times 7 \;=\; 168$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{14}.

$$(4\times 2)+(4\times 2\times 4)+(4\times 2\times 4\times 3)+(4\times 2\times 4\times 3\times 1) \;=\; 8+32+96+96 \;=\; 232$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{14}.

$$(4\times 2\times 4\times 3\times 1)\times(4+4) \;=\; 96\times 8 \;=\; 768$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{15}.

$$(4\times 2)+(4\times 2\times 4)+(4\times 2\times 4\times 2)+(4\times 2\times 4\times 2\times 1) \;=\; 8+32+64+64 \;=\; 168$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{15}.

$$(4\times 2\times 4\times 2\times 1)\times(4+4) \;=\; 64\times 8 \;=\; 512$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{16}.

$$(4\times 1)+(4\times 1\times 4)+(4\times 1\times 4\times 3)+(4\times 1\times 4\times 3\times 1) \;=\; 4+16+48+48 \;=\; 116$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{16}.

$$(4\times 1\times 4\times 3\times 1)\times(4+4) \;=\; 48\times 8 \;=\; 384$$

Custo computacional da parte sincronizante positiva do modelo.

$$\begin{gathered}(116+720)+(77+480)+(106+640)+(106+640)+(77+480)+\\ (64+168)+(232+768)+(168+512)+(116+384) \;=\\ 836+557+746+746+557+232+1.000+680+500 \;=\; 5.854\end{gathered}$$

Sabendo-se que o custo da parte sincronizante negativa é igual a positiva para este modelo,
o custo computacional total da parte sincronizante deste modelo é:

$$5.854+5.854 \;=\; 11.708$$

Adicionando o custo da parte local, tem-se então o custo computacional total do *Slice* otimizado para o exemplo 1.

$$5.760+11.708 \;=\; 17.468$$

Segundo exemplo

Considerando a otimização dos Fatores da técnica *Slice*, o custo computacional da parte sincronizante do segundo exemplo de modelo SAN é calculado da seguinte maneira:

Custo computacional otimizado da parte sincronizante positiva do modelo.

$$\sum_{k=1}^{E}\left(\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right] + \prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]\right)$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_7 (matriz positiva).

$$(3\times 4)+(3\times 4\times 3)+(3\times 4\times 3\times 6)+(3\times 4\times 3\times 6\times 4) \;=\; 12+36+216+864 \;=\; 1.128$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_7 (matriz positiva).

$$(3\times 4\times 3\times 6\times 4)\times(2+2) \;=\; 864\times 4 \;=\; 3.456$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_8 (matriz positiva).

$$(2\times 4)+(2\times 4\times 2)+(2\times 4\times 2\times 6)+(2\times 4\times 2\times 6\times 4) \;=\; 8+16+96+384 \;=\; 504$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_8 (matriz positiva).

$$(2\times 4\times 2\times 6\times 4)\times(2+2) \;=\; 384\times 4 \;=\; 1.536$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_9.

$$(4\times 4)+(4\times 4\times 2)+(4\times 4\times 2\times 6)+(4\times 4\times 2\times 6\times 4) \;=\; 16+32+192+768 \;=\; 1.008$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_9.

$$(4\times 4\times 2\times 6\times 4)\times(2+1) \;=\; 768\times 3 \;=\; 2.304$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{10}.

$$(4\times 4)+(4\times 4\times 3)+(4\times 4\times 3\times 2)+(4\times 4\times 3\times 2\times 4) \;=\; 16+48+96+384 \;=\; 544$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{10}.

$$(4\times 4\times 3\times 2\times 4)\times(2+1) \;=\; 384\times 3 \;=\; 1.152$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{11}.

$$(4\times 4)+(4\times 4\times 3)+(4\times 4\times 3\times 2)+(4\times 4\times 3\times 2\times 2) \;=\; 16+48+96+192 \;=\; 352$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{11}.

$$(4\times 4\times 3\times 2\times 2)\times(2+2) \;=\; 192\times 4 \;=\; 768$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{12}.

$$(4\times 4)+(4\times 4\times 3)+(4\times 4\times 3\times 2)+(4\times 4\times 3\times 2\times 2) \;=\; 16+48+96+192 \;=\; 352$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{12}.

$$(4\times 4\times 3\times 2\times 2)\times(2+2) \;=\; 192\times 4 \;=\; 768$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{13}.

$$(4\times 4)+(4\times 4\times 3)+(4\times 4\times 3\times 1)+(4\times 4\times 3\times 1\times 1) \;=\; 16+48+48+48 \;=\; 160$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{13}.

$$(4\times 4\times 3\times 1\times 1)\times(2+2) \;=\; 48\times 4 \;=\; 192$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_{14}.

$$(4\times 4)+(4\times 4\times 3)+(4\times 4\times 3\times 6)+(4\times 4\times 3\times 6\times 1) \;=\; 16+48+288+288 \;=\; 640$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_{14}.

$$(4\times 4\times 3\times 6\times 1)\times(2+1) \;=\; 288\times 3 \;=\; 864$$

Custo computacional da parte sincronizante positiva do modelo.

$$(1.128+3.456)+(504+1.536)+(1.008+2.304)+(544+1.152)+$$
$$(352+768)+(352+768)+(160+192)+(640+864) \;=$$
$$4.584+2.040+3.312+1.696+1.120+1.120+352+1.504 \;=\; 15.728$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_7 (matriz negativa).

$$(2\times 4)+(2\times 4\times 3)+(2\times 4\times 3\times 6)+(2\times 4\times 3\times 6\times 4) \;=\; 8+24+144+576 \;=\; 752$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_7 (matriz negativa).

$$(2\times 4\times 3\times 6\times 4)\times(2+2) \;=\; 576\times 4 \;=\; 2.304$$

Cálculo do termo $\sum_{i=2}^{N-1}\left[\prod_{j=1}^{i} nz_i\right]$ para o evento sincronizante e_8 (matriz negativa).

$$(2\times 4)+(2\times 4\times 1)+(2\times 4\times 1\times 6)+(2\times 4\times 1\times 6\times 4) \;=\; 8+8+48+192 \;=\; 256$$

Cálculo do termo $\prod_{i=1}^{N-1} nz_i \times \left[n_N + nz_N\right]$ para o evento sincronizante e_8 (matriz negativa).

$$(2\times 4\times 1\times 6\times 4)\times(2+2) \;=\; 192\times 4 \;=\; 768$$

Custo computacional da parte sincronizante negativa do modelo.

$$(752 + 2.304) + (256 + 768) + (1.008 + 2.304) + (544 + 1.152) +$$

$$(352 + 768) + (352 + 768) + (160 + 192) + (640 + 864) \quad =$$

$$3.056 + 1.024 + 3.312 + 1.696 + 1.120 + 1.120 + 352 + 1.504 \quad = \quad 13.184$$

Custo computacional total para a parte sincronizante do modelo.

$$15.728 + 13.184 = 28.912$$

Adicionando o custo da parte local, tem-se então o custo computacional total do *Slice* otimizado para o exemplo 2.

$$6.144 + 28.912 = 35.056$$

B.2 Custo Computacional com otimização da diagonal

Esta seção apresenta os cálculos do Custo Computacional das técnicas *Shuffle* e *Slice* considerando a otimização da diagonal. Ou seja, são eliminados do cálculo todos os elementos diagonais das matrizes locais dos modelos SAN, assim como todas as matrizes negativas dos eventos sincronizantes.

Portanto, é necessário refazer apenas os cálculos da parte local dos modelos SAN, pois da parte sincronizante basta descartar a parte negativa.

B.2.1 Custo Computacional para o primeiro exemplo

Para realizar o cálculo do Custo Computacional para o primeiro exemplo de modelo SAN, foram considerados os novos valores para a parte local, apresentados na Tabela 4.8, assim como o cálculo da parte sincronizante positiva que já havia sido realizado.

Técnica *Shuffle*

O Custo Computacional da técnica *Shuffle* para este modelo SAN é calculado da seguinte forma:

Fórmula do custo computacional para a parte local do modelo.

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}$$

Cálculo do termo $\prod_{i=1}^{N} n_i$.

$$4 \times 2 \times 4 \times 3 \times 5 \times 4 \;=\; 1.920$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$.

$$\frac{1}{4} + \frac{0}{2} + \frac{1}{4} + \frac{0}{3} + \frac{3}{5} + \frac{2}{4} \;=\; \frac{5+0+5+0+12+10}{20} \;=\; \frac{32}{20} \;=\; \frac{8}{5}$$

Custo computacional total para a parte local do modelo.

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \;=\; 1.920 \times \frac{8}{5} \;=\; \frac{15.360}{5} \;=\; 3.072$$

Custo computacional da técnica *Shuffle* para este modelo SAN.

$$\sum_{k=1}^{1+E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right) \;=\; 3.072 + 83.136 \;=\; 86.208$$

Técnica *Slice*

Considerando que o cálculo da parte local é igual para ambas as técnicas, o Custo Computacional da técnica *Slice* para este modelo SAN é calculado assim:

Custo computacional da técnica *Slice* para este modelo SAN.

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N \Big] \right) \;=\; 3.072 + 7.200 \;=\; 10.272$$

B.2.2 Custo Computacional para o segundo exemplo

Para realizar o cálculo do Custo Computacional para o segundo exemplo de modelo SAN , foram considerados os valores da Tabela 4.9, bem como o cálculo da parte sincronizante positiva já realizado anteriormente.

Técnica *Shuffle*

O Custo Computacional da técnica *Shuffle* para este modelo SAN é calculado da seguinte maneira:

Fórmula do custo computacional para a parte local do modelo.

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}$$

Cálculo do termo $\prod_{i=1}^{N} n_i$.

$$4 \times 4 \times 3 \times 6 \times 4 \times 2 \quad = \quad 2.304$$

Cálculo do termo $\sum_{i=1}^{N} \frac{nz_i}{n_i}$.

$$\frac{1}{4} + \frac{2}{4} + \frac{0}{3} + \frac{2}{6} + \frac{1}{4} + \frac{0}{2} \quad = \quad \frac{3+6+0+4+3+0}{12} \quad = \quad \frac{16}{12} \quad = \quad \frac{4}{3}$$

Custo computacional total para a parte local do modelo.

$$\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \quad = \quad 2.304 \times \frac{4}{3} \quad = \quad \frac{9.216}{3} \quad = \quad 3.072$$

Custo computacional da técnica *Shuffle* para este modelo SAN.

$$\sum_{k=1}^{1+E} \left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \right) \quad = \quad 3.072 + 91.584 \quad = \quad 94.656$$

Técnica *Slice*

Considerando que o cálculo da parte local é igual para ambas as técnicas, o Custo Computacional da técnica *Slice* para este modelo SAN é calculado da seguinte forma:

Custo computacional da técnica *Slice* para este modelo SAN.

$$\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right) + \sum_{k=1}^{2E} \left(\prod_{i=1}^{N-1} nz_i \times \Big[(N-2) + n_N + nz_N\Big]\right) \;=\; 3.072 + 23.520 \;=\; 26.592$$

B.2.3 Custo Computacional para *Slice* com Otimização dos Fatores

Considerando que a otimização dos fatores para o algoritmo da técnica *Slice* refere-se somente à parte sincronizante do modelo, e que a otimização da diagonal não modifica o cálculo da parte sincronizante positiva, basta somar para cada exemplo o custo da parte local calculado na seção B.2 com o custo otimizado da parte sincronizante positiva calculado na seção B.1.3, como apresentado a seguir.

Primeiro exemplo

Custo computacional do *Slice* otimizado para o primeiro exemplo.

$$3.072 + 5.854 \;=\; 8.926$$

Segundo exemplo

Custo computacional do *Slice* otimizado para o segundo exemplo.

$$3.072 + 15.728 \;=\; 18.800$$