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

Estratégias de Reestruturação de Descritores Markovianos para a Solução Numérica de Sistemas

Ricardo M. Czekster

Orientador: Paulo Fernandes

Proposta de Tese

Porto Alegre - Março , 2009

Resumo

Uma série de formalismos foram definidos ao longo dos anos com o objetivo de aumentar o nível de abstração e oferecer uma alternativa de modelagem mais sofisticada do que a proporcionada pelas tradicionais Cadeias de Markov. Um exemplo de formalismo bastante utilizado para a obtenção de índices de desempenho de sistemas são as Redes de Autômatos Estocásticos (*Stochastic Automata Networks* - SAN) que utilizam premissas básicas de modelagem e interação de subcomponentes para permitir a análise de sistemas complexos, especialmente direcionado para realidades paralelas e distribuídas. O seu mecanismo atual de solução é extremamente eficiente em termos de memória e utiliza propriedades conhecidas da Álgebra Tensorial (constante e generalizada) para multiplicar a representação utilizada em SAN, ou seja, um conjunto de termos tensoriais denominado *descritor markoviano*, por um vetor de probabilidade. Esta operação fundamental é chamada de *Multiplicação Vetor-Descritor* (MVD) e existem três maneiras de ser realizada: de forma esparsa (ineficiente em memória, eficiente em tempo), utilizando o Algoritmo *Shuffle* (eficiente em memória, ineficiente em tempo, dependendo do modelo) e através do Algoritmo *Split*, que é uma combinação das duas primeiras abordagens. A principal contribuição do *Split* foi a proposta de um método híbrido onde é permitido um aumento razoável de memória para cálcular os índices de desempenho mais rapidamente. O presente trabalho está baseado em três eixos: a) na discussão das principais estratégias para reestruturação de termos tensoriais em descritores, b) na análise teórica e prática destas considerações e c) na proposição de um algoritmo de determinação da forma mais otimizada de tratar o descritor para usar a abordagem *Split*. Para os casos observados foi demonstrado numericamente que o melhor ganho, balanceando-se tempo e memória, é quando reordenam-se as matrizes dos termos tensoriais tratando as do tipo identidade na parte estruturada e os avaliam-se os elementos funcionais apenas uma vez na parte esparsa. Esta operação é equivalente à conversão de descritores generalizados para constantes em tempo de execução e esta tradução resulta em ganhos de tempo consideráveis para determinadas classes de modelos. Observou-se também que as atividades de sincronização entre os autômatos realizam um papel relevante no desempenho obtido, assim como o total de dependências funcionais existentes. Este trabalho, por fim, identifica estas classes de modelos que são mais adequadas para a utilização do Algoritmo *Split* e propõe um algoritmo de reestruturação de descritores markovianos que privilegia as características fundamentais quanto à esparsidade e estruturação dos modelos para balancear os gastos com memória e observar as melhoras no tempo de execução.

Abstract

A lot of formalisms have been defined throughout the years with the objective of enhancing the abstraction level and offer a more sophisticated modeling alternative than traditional Markov Chains. One formalism example that is used very often to calculate performance indexes are the Stochastic Automata Networks (SAN) which defines basic modeling primitives and sub component interactions to allow the analysis of complex systems, specially oriented to parallel and distributed realities. Its solution mechanism is highly memory efficient and uses Tensor Algebra properties (both constant and generalized) to multiply the tensor representation, i.e., a set of tensorial terms known as markovian descriptor by a final probability vector. This fundamental operation is called Vector-Descriptor Multiplication (VDM) defining three ways to be performed: sparsely (memory inefficient, time efficient), using the Shuffle Algorithm (memory efficient, time inefficient, depending on the model) or through the Split Algorithm, a combination between the two previous approaches. The main contribution of Split was to propose a hybrid method where it is possible to use reasonable amounts of memory to obtain the performance indexes more quickly. The present work is based upon three axis: a) discussing the main tensorial term restructuring strategies, b) in the theoretical and practical analysis of this considerations and c) in the proposition of an algorithm to determine the most optimized way of dealing with the descriptor to use the Split approach. For the observed cases it was possible to numerically show that the best gain balancing time and memory spent is when the tensorial terms are reordered considering the identity matrices in the structured part and the functional elements are evaluated just once. This operation is equivalent to converting a generalized descriptor to a constant one in execution time and this translation is valid to a large class of models. It was also observed that automata synchronizations plays a relevant role in the obtained performance, as well as the total of existing functional dependencies. This work, at last, identifies the most suitable models for the usage of the Split Algorithm and proposes an algorithm to restructure markovian descriptors observing the sparsity and model structuring to optimize memory and execution time.

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

CM	*Cadeia de Markov*	9
MIT	*Massachussets Institute of Technology*	10
CTMC	*Continuous Time Markov Chain*	10
DTMC	*Discrete Time Markov Chain*	10
PSS	*Product State Space*	11
RSS	*Reacheable State Space*	12
DDM	*Diagramas de Decisão Multi-valorados*	12
RP	*Redes de Petri*	12
RPE	*Redes de Petri Estocásticas*	12
RPC	*Redes de Petri Coloridas*	13
SAN	*Stochastic Automata Networks*	14
MVD	*Multiplicação Vetor-Descritor*	15
PEPS	*Performance Evaluation of Parallel Systems*	15
MPA	*Markovian Process Algebra*	15
PEPA	*Performance Evaluation Process Algebra*	15
EMPA	*Extended Markovian Process Algebra*	15
GD	*Grafo de Derivação*	16
MT	*Matriz de Transição*	16
pwb	*PEPA Workbench*	18
ATC	*Álgebra Tensorial Clássica*	21
ATG	*Álgebra Tensorial Generalizada*	24
GDC	*Grau de Dependência Constante*	45
GDG	*Grau de Dependência Generalizado*	45
FAS	*First Available Server*	62
ASP	*Alternate Service Patterns*	67
SGSPN	*Superposed Generalized Stochastic Petri Nets*	73

List of Algorithms

Sumário

1 Introdução

A complexidade relacionada à análise de desempenho de sistemas é determinada principalmente pela dificuldade existente na representação dos modelos e não somente pela captura e abstração dos elementos mais relevantes de uma realidade. Esta tarefa pode ser realizada sobretudo com o uso de formalismos como, por exemplo, Cadeias de Markov (CM) onde existem basicamente duas entidades elementares: estados do sistema e transições com taxas ou probabilidades associadas que ditam a frequência média de saída de um estado para chegar em um outro.

Apesar da sua simplicidade para modelar realidades e consequentemente extrair probabilidades de permanência em cada estado, este formalismo possui algumas desvantagens, sendo a principal delas a explosão do espaço de estados ainda na fase de modelagem. Para amenizar este problema, pesquisas foram iniciadas para representações dos sistemas através dos chamados formalismos estruturados, cuja principal característica é ter uma representação markoviana equivalente. Exemplos de tais formalismos são as Redes de Autômatos Estocásticos (*Stochastic Automata Networks* - SAN)[65, 66], Redes de Petri[4] e PEPA (*Performance Evaluation Process Algebra*)[44].

Uma vez que um sistema é modelado utilizando-se de algum formalismo (estruturado ou não), a próxima etapa é determinar as probabilidades estacionárias (quando um equilíbrio foi atingido) ou transientes (quando interrompe-se o processo antes da convergência) de cada estado possível. Através desta solução os índices de desempenho para o sistema também podem ser calculados. Este processo é feito pelo mapeamento das taxas de ocorrência contidas nas transições (considerando escala de tempo contínua) ou das probabilidades de ocorrência (considerando escala de tempo discreta) em uma matriz que pode ser vista como um sistema linear a ser resolvido. A solução buscada, no contexto desta tese, é a análise do sistema na estacionariedade, ou seja, em um ponto onde este não mais evolui, atingindo o equilíbrio e retornando as probabilidades de permanência em cada estado.

Note-se que a solução numérica de modelos endereça uma parte importante da análise de desempenho de realidades complexas, responsável pelo cálculo de índices que atestam de forma numérica e quantitativa o desempenho de inúmeros sistemas. Um exemplo de medidas ou índices que podem ser extraídos através de modelagens matemáticas são os clássicos problemas de alocação, disponibilidade e utilização de recursos. Outro exemplo recorrente de aplicação são modelagens de sistemas distribuídos para detectar a incidência de gargalos ou a validação de possíveis otimizações.

A pesquisa em técnicas de avaliação de desempenho está atualmente direcionada à solução otimizada destes sistemas lineares, observando a quantidade de memória a ser potencialmente gasta tanto para armazenar as matrizes que descrevem os modelos matematicamente quanto para prover soluções em regime estacionário (ou mesmo transiente). A problemática da explosão do espaço de estados explícita em Cadeias de Markov, impulsionou a pesquisa em direção a diferentes formas compactas de modelagem e adaptação das soluções numéricas a estruturas de dados cada vez mais complexas.

As técnicas tradicionais de armazenamento e solução em SAN, por exemplo, são consideradas um grande avanço pois esta representação nunca armazena a matriz de transição de forma plena como as CM. Este formalismo utiliza-se de uma representação tensorial para os modelos, composta por um conjunto finito de matrizes que compõem o que denomina-se *descritor* markoviano. Este formalismo é, portanto, extremamente eficiente em termos de memória necessária pois prioriza o armazenamento de pequenas matrizes esparsas. Entretanto, pouco se sabe sobre os efeitos de se reestruturar um descritor tensorial que descreve uma rede de autômatos.

A solução numérica de descritores markovianos é igualmente foco das pesquisas atuais e os algoritmos especializados para tal são chamados de algoritmos para a *Multiplicação Vetor-Descritor* (MVD). As operações complexas de multiplicação que devem ser efetuadas no nível do descritor markoviano podem ser otimizadas de diferentes maneiras. Os descritores são formados de matrizes com elementos constantes ou funcionais, que são tipos mais complexos para o tratamento computacional apesar de serem simples primitivas de modelagem do formalismo citado. Logo, dada a formação das matrizes que compõem o descritor, este pode ser classificado respectivamente como constante ou generalizado.

Esta proposta de tese tem como objetivo estudar as formas atuais de cálculo eficiente da MVD e propor uma forma de tratamento otimizado do descritor markoviano no nível de composição dos seus termos tensoriais, observando as características das matrizes e do descritor de SAN como um todo. Serão analisados especificamente os algoritmos *Esparso* [69], *Shuffle* [33] e *Split* [24], verificando as suas características, vantagens e desvantagens quando aplicados à solução de modelos. Pesquisas recentes mostram que o Algoritmo *Split*, que propõe a divisão dos termos tensoriais para aplicar uma solução híbrida, é vantajoso para muitos modelos mas pode também ser oneroso computacionalmente se alguns de seus parâmetros não forem devidamente definidos. Desconhece-se também a sua aplicação para modelos compostos por descritores generalizados, ou seja, descritores que possuem elementos funcionais, sendo um desafio a adaptação do mesmo para resolver sistemas com primitivas mais complexas como é o caso de transições funcionais.

Sabe-se que ao utilizar uma representação tensorial de um sistema é possível manipular as matrizes independentemente e efetuar operações tais como permutações (ou reordenamentos) no descritor. Essas operações implicam em alterar o posicionamento original das matrizes no descritor (ou de forma geral dos autômatos em si). Para tanto, faz-se necessário uma maior compreensão dos fatores a serem considerados quando reordenam-se e reestruturam-se os termos tensoriais de um descritor, através da adoção de diferentes estratégias. Ao dividir um termo tensorial para uma solução híbrida do mesmo, pode-se por exemplo também permutar suas matrizes de diversas formas, observando características tais como: o número de avaliações de elementos funcionais necessárias, o número de elementos não-nulos compondo as matrizes e o número de matrizes do tipo identidade existentes.

Em resumo, não conhecem-se os efeitos de se tratar numericamente os descritores generalizados de forma híbrida e as implicações da existência de matrizes do tipo identidade, bem como o impacto das avaliações de elementos funcionais no método. Também ainda não existem estudos para o Algoritmo *Split* efetuar reordenamentos nos termos tensoriais de forma a otimizar sua solução e consequentemente uma aplicação eficiente para uma grande classe de modelos.

1.1 Motivação

O Algoritmo *Split* é recente e inovou na maneira pela qual se realiza a MVD. Essa nova abordagem torna possível balancear os custos em memória, armazenando uma parte do descritor. No restante do termo tensorial é aplicada uma abordagem estruturada que não precisa guardar a matriz plena que corresponde ao gerador infinitesimal da CM desta parte. Sabe-se que o Algoritmo *Shuffle* é otimizado para tratar eficientemente as dependências funcionais onde também realiza permutações na ordem lexicográfica[1] das matrizes para otimizar as avaliações, entre outros mecanismos para ganho de desempenho pesquisados em outros trabalhos [33].

A motivação para este trabalho reside no fato de que conhece-se tanto os algoritmos esparso quanto o *Shuffle* em profundidade; o que não se sabe é como melhor combinar as características destas duas abordagens para tanto descritores baseados em álgebra tensorial clássica quanto generalizada. Estes desafios incentivam a busca de um maior entendimento sobre as propriedades fundamentais existentes na abordagem híbrida do Algoritmo *Split* e quando é mais vantajoso utilizá-las. Enfim, sente-se a falta de análises precisas da viabilidade computacional destas características em relação à memória que será gasta e ao tempo de execução, bem como suas implicações numéricas e algorítmicas.

[1] Na ordem natural, neste caso, à ordem original das matrizes descritas pelos autômatos.

Trata-se de um problema não trivial onde o número de escolhas possíveis para as divisões dos termos tensoriais é grande. Pode-se optar por gastar menos memória e ser penalizado no tempo gasto para as multiplicações de ponto flutuante necessárias. Pode-se escolher avaliar menos funções a cada iteração do método, pois as funções dos modelos podem ter definições simples ou sem muitos parâmetros. Deseja-se gastar mais memória e tentar obter ganhos avaliando as funções apenas uma vez, no início da MVD e utilizar esses valores até o final. Enfim, são muitas as possibilidades existentes de pesquisa sobre esse assunto.

Um outro ponto importante a ser discutido diz respeito à inexistência de um conhecimento sólido sobre quando é melhor modelar um sistema com apenas taxas constantes ao invés de utilizar taxas funcionais e vice-versa [9]. Também deseja-se investigar o relacionamento existente no gasto de memória e as implicações no tempo de execução do método. Faz-se necessário também conhecer as formas de permutar as matrizes do termo tensorial para refletir positivamente no tempo gasto. Ao permutar as matrizes do termo tensorial, pode-se escolher reordenar de acordo com as suas dimensões, seus elementos não-nulos, seu total de identidades, seu total de avaliações de funções ou uma combinação de algumas ou todas estas propriedades. A possibilidade de reordenar o termo tensorial traz outros questionamentos importantes para serem investigados tais como onde e quando avaliam-se as funções, se na parte esparsa do termo tensorial ou na parte estruturada, por exemplo. Uma outra questão diz respeito ao custo envolvido para permutar os elementos e se vale a pena realizar estas operações para ganhar em tempo.

1.2 Objetivo

O principal objetivo deste trabalho é propor um algoritmo de reestruturação de termos tensoriais que, dadas as características de termos constantes ou generalizados que formam os descritores markovianos, forneça uma solução mais eficiente que as existentes atualmente mas que não torne a memória a ser gasta em um impedimento para o método de MVD utilizado. Para a proposição deste algoritmo é necessário definir as características dos termos tensoriais que mais influenciam na MVD, tais como número de matrizes identidade, o número de avaliações de funções e as dimensões de cada matriz, para citar algumas. Assim, será possível determinar a classe de modelos que são melhores adaptadas para a utilização do Algoritmo *Split*, descobrindo as melhores estratégias de divisão.

1.3 Metodologia

Este trabalho exige duas partes igualmente importantes: uma parte teórica para o pleno entendimento dos conceitos e uma parte prática, para validação das hipóteses de pesquisa que serão enunciadas. A parte teórica envolverá estudos variados sobre formalismos estruturados e métodos de solução existentes enquanto que a parte prática será composta por implementações e testes com diversos modelos, verificando a sustentação das ideias e a sua aplicabilidade em diferentes realidades.

Para tanto, enumeram-se a seguir os principais passos a realizar para o cumprimento do objetivo especificado na seção anterior. Cabe ressaltar que os passos descritos objetivam a antecipação de problemas tanto práticos quanto teóricos para validação da proposta das estratégias de reestruturação de descritores markovianos:

- Estudo dos principais conceitos e algoritmos envolvidos em MVD;
- Estudo das propriedades da álgebra tensorial clássica e generalizada bem como sua relação com descritores markovianos;
- Entender a necessidade da avaliação de funções, seu processo interno e utilidade em modelagem estocástica de sistemas;
- Estudo do mecanismo de reordenação dos termos tensoriais, motivação da existência de permutações e aplicabilidade para o Algoritmo *Split*;
- Estudo das operações algorítmicas que são feitas quando utiliza-se o Algoritmo *Split*;

- Listar as principais formas de execução do Algoritmo *Split* e as possibilidades de divisão dos termos tensoriais;
- Realização de um estudo teórico das principais reestruturações passíveis de serem aplicadas, baseado nas características dos descritores markovianos e dos seus termos tensoriais;
- Levantamento de hipóteses de pesquisa, previsão teórica dos casos onde o Algoritmo *Split* pode desempenhar-se melhor e os casos onde a sua execução será impraticável ou menos otimizada e porquê;
- Antecipação das principais formas de reestruturação dos termos tensoriais tais que amplifiquem os ganhos do Algoritmo *Split* ao ser comparado com as abordagens existentes atualmente;
- Otimizações do Algoritmo *Split* e verificação da possibilidade de se precisar armazenar estruturas auxiliares para verificar se o tempo dispendido é menor; estudo para realização de permutações nos termos e verificação da necessidade de se reordenar as matrizes dos termos tensoriais;
- Comparação da execução do Algoritmo *Split* com o método preexistente (Algoritmo *Shuffle*) para validação e comparação dos resultados para o mesmo modelo; teste para verificar a divisão de acordo com todas as possibilidades de divisão do termo tensorial;
- Escolha de modelos relevantes (com descritores variados) com taxas funcionais e constantes definidas para utilização em casos de teste;
- Estudo de representação tensorial em outros formalismos com o objetivo de capturar as principais propriedades existentes para a realização de uma tradução para o formato tensorial e posterior solução MVD com o Algoritmo *Split*;
- Planificação dos testes e experimentos a serem conduzidos escolha dos testes estatísticos (relativos aos intervalos de confiança, principalmente) e número de execuções necessárias para se obter valores confiáveis de tempo e memória;
- Interpretação dos resultados e verificação se o que foi pretendido pela parte teórica foi ou não confirmado pela parte prática;
- Descobrir se o Algoritmo *Split* tem melhor desempenho para uma classe de modelos e investigar a sua relação direta com a estratégia utilizada e a memória gasta para a realização das tarefas necessárias;
- Analisar o custo numérico e prático da realização das avaliações de funções nos descritores generalizados;
- Escrita dos resultados obtidos e discussão;
- Previsão de futuras otimizações para a implementação realizada;
- Discussão de outros aspectos relevantes, por exemplo, sobre a paralelização do método ou novos desenvolvimentos importantes.

1.4 Contribuição

Estas tarefas, quando completas, produzirão resultados importantes no contexto da solução numérica de sistemas baseados em representação tensorial. Em um primeiro momento será possível compreender porque o Algoritmo *Split* obteve os melhores resultados para cada modelo. Essas explicações serão responsáveis pela proposição de um algoritmo que, dado um termo tensorial, escolha automaticamente a melhor divisão para efetuar a parte esparsa e a parte estruturada para a solução. Esse algoritmo observará as características dos termos, ou seja, o número de identidades, o número de avaliações de funções a serem realizadas e a ordem para tratamento das matrizes para decidir onde melhor separá-lo, sem atingir os limites de memória disponível.

Em um segundo momento, ao se conhecer a formação de um descritor markoviano e fornecer meios para realização eficiente de MVD, pode-se pensar em generalizar os modelos que podem ser utilizados. Por exemplo, pode-se definir um conjunto de regras de tradução para que diferentes formalismos estruturados que possuam uma representação tensorial sejam beneficiados com uma solução MVD otimizada. Isso ampliará as maneiras de se analisar o desempenho de sistemas complexos uma vez que será possível modelá-lo utilizando-se todas as

primitivas existentes em um determinado formalismo e resolvê-lo com métodos que encontram-se no estado-da-arte de MVD com representação tensorial, dado que uma conversão para este formato exista e seja válida.

Vale ressaltar que os modelos podem ter alta complexidade no que tange as suas transições pois os mecanismos aqui desenvolvidos funcionarão tanto para o uso de taxas constantes quanto funcionais. Verifica-se que a existência de transições funcionais aumentam a gama de modelagens existentes de realidades complexas e estes descritores generalizados são aqui tratados de igual forma em comparação com descritores constantes. Isso possibilitará que tanto os modelos sejam resolvidos em menos tempo quanto que diferentes formas de representações poderão ser usadas para extração de índices de desempenho confiáveis de realidades diversas. As reestruturações aqui propostas servirão para estender as aplicações do Algoritmo *Split* para diferentes classes de modelos onde ele possa fornecer um desempenho superior em termos de custo computacional ao comparado com as abordagens atuais de MVD.

1.5 Organização

Este trabalho está organizado da seguinte forma: o Capítulo 2 aborda diferentes formalismos presentes na literatura, seguido pelo Capítulo 3 que discorre sobre as formas de MVD com exemplos relevantes. Este trabalho estuda as estratégias para divisão de termos tensoriais constantes e generalizados no Capítulo 4, considerando-se as implicações teóricas e as características mais importantes a serem observadas. O Capítulo 5 lista os principais resultados para alguns experimentos escolhidos e o Capítulo 6 desenvolve a proposta de tese com um algoritmo de determinação do ponto de corte de termos tensoriais e as atividades futuras.

2 Descrição de sistemas com formalismos

Uma das maneiras usuais para descrição de realidades complexas é através do uso de formalismos que capturam suas propriedades e características e, de acordo com um conjunto de regras, definem precisamente as entidades relacionadas e como estas interagem à medida que o sistema evolui. Um importante exemplo é a descoberta de como os objetos transitam de condição em condição, trocando ou não de estado, de acordo com uma taxa ou frequência com a qual acontece, permitindo que sejam extraídos índices computacionais de desempenho. Este capítulo discutirá maneiras de definir modelos abstratos de sistemas através da aplicação direta de diferentes formalismos. No contexto deste trabalho assume-se que o objetivo da descrição com formalismos é descobrir uma solução analítica para o sistema, ao invés de recorrer a outras práticas igualmente importantes porém algumas vezes menos precisas tais como simulação.

Um formalismo normalmente contém um conjunto de regras e uma gramática para descrever um sistema de maneira não ambígua. Para o caso dos formalismos atuais de descrição com vistas à análise de desempenho e soluções analíticas, é comum verificar que a maioria define três entidades principais: estados, transições e eventos. É evidente que, dado que existem diversos formalismos, eles se distinguem de alguma forma. Entretanto, todos recaem sobre uma mesma constante, sendo baseados, fundamentalmente, no formalismo das Cadeias de Markov (melhor explicado na Seção 2.2).

Este capítulo inicia com algumas considerações preliminares na Seção 2.1, seguida da definição de Cadeia de Markov na Seção 2.2, Redes de Petri e Redes de Petri Coloridas na Seção 2.3. Também abordará as Redes de Autômatos Estocásticos na Seção 2.4 e Álgebras de Processo na Seção 2.5, finalizando com uma discussão com uma análise comparativa que evidencia as vantagens e desvantagens dos diversos formalismos na Seção 2.6.

2.1 Modelagem de sistemas

À medida que sistemas complexos tornaram a sua análise difícil e demorada, a necessidade de métodos formais para avaliação de desempenho precisou ser melhor investigada com o objetivo de prover respostas e ajudar na descoberta de eventuais problemas existentes. O objetivo inicial da análise de sistemas foi o de adicionar determinismo e enumerar meios de descrever matematicamente realidades complexas. A idéia era aplicar conceitos formais já presentes e construir modelos que representassem realidades com o maior número de detalhes possível. O próximo passo seria tentar descobrir propriedades de estacionariedade, ou seja, quando o equilíbrio fosse atingido. Igualmente importante era conseguir realizar estas tarefas de maneira precisa e utilizando os recursos disponíveis da melhor forma possível, retornando resultados em uma quantidade de tempo razoável.

As observações iniciais indicaram que simples modelos poderiam ser analiticamente resolvidos em uma quantidade finita e razoável de tempo e mesmo assim ainda capturando as características primordiais do sistema, ou seja, simples modelos eram amplamente escaláveis. Apesar desta ser uma maneira não-trivial à primeira vista de atacar o problema da modelagem de sistemas, foi um resultado marcante porque provou que era suficientemente simples apenas adicionar estados e transições em um dado modelo e ele mesmo assim retornaria respostas consistentes [70]. Os resultados demonstrariam se uma dada realidade estaria degradada e até mesmo quando começaria a desempenhar suas atividades de forma insatisfatória, *antes* da realidade física, por exemplo, isso pode ser feito

sem precisar comprar máquinas ou simular realidades complexas com a adição de componentes físicos em um sistema. De fato, esta foi a maior contribuição da análise de sistemas através da definição de modelos somente com as informações mais relevantes dos sistemas, abstraindo detalhes desnecessários e sendo escaláveis (uma vez que possuem cada vez mais componentes e sua interação é crescentemente complexa).

Os passos para inspecionar sistemas devem envolver apenas as suas operações fundamentais, ou seja, tentar capturar a essência das tarefas realizadas pelo sistema. Quando apropriadamente definidos, estes passos aumentarão as chances de produzirem resultados válidos. Antes de modelar uma realidade complexa é extremamente importante definir sua operação principal em um modelo o mais simples possível, definindo quais estados ele possui e como ele transita de estado para estado. O próximo passo é realizar uma análise minuciosa descartando estados que não são importantes ou até mesmo desnecessários para a operação do sistema como um todo. Esta fase é conhecida por *fase de refinamento*, onde o modelador procura verificar o sistema de acordo com a realidade que foi descrita e descobrir se ela corresponde ao que o sistema realiza de fato.

Após verificar os cenários mapeáveis, aproxima-se o momento de escolher qual formalismo ofertará a melhor maneira de descrever a realidade, dado as suas características e as formas pelas quais ele evolui e interage. Igualmente importante é escolher um formalismo que apresente o melhor conjunto de ferramentas computacionais para solução e verificação pois não basta ser forte em descrição e ineficiente ao calcular os índices de desempenho, dado que este é um dos objetivos mais importantes de se realizar estes estudos. Procura-se então o equilíbrio, onde o sistema é resolvido, ou seja, encontram-se (ou não) soluções para as suas variáveis.

Calcula-se o vetor de probabilidade correspondente à estacionariedade do sistema, *i.e.*, supondo que o sistema foi 'simulado' até um ponto onde suas mudanças não mais afetam o seu estado inicial, ele atingiu um momento onde, dadas as suas taxas iniciais, este não mais evolui. Dessa forma, é possível extrair índices de desempenho comparando com diferentes formas de modelar o problema e adicionando ou excluindo componentes e verificando se ocorreram mudanças nos índices. No caso de degradação dos índices, o mais recomendável é alterar as transições e suas taxas, os estados e os eventos com o intuito de inspecionar qual a melhor forma que o sistema deverá ser composto para refletir a realidade estudada propondo mudanças que vão impactar diretamente na maneira com a qual existe uma melhora de desempenho.

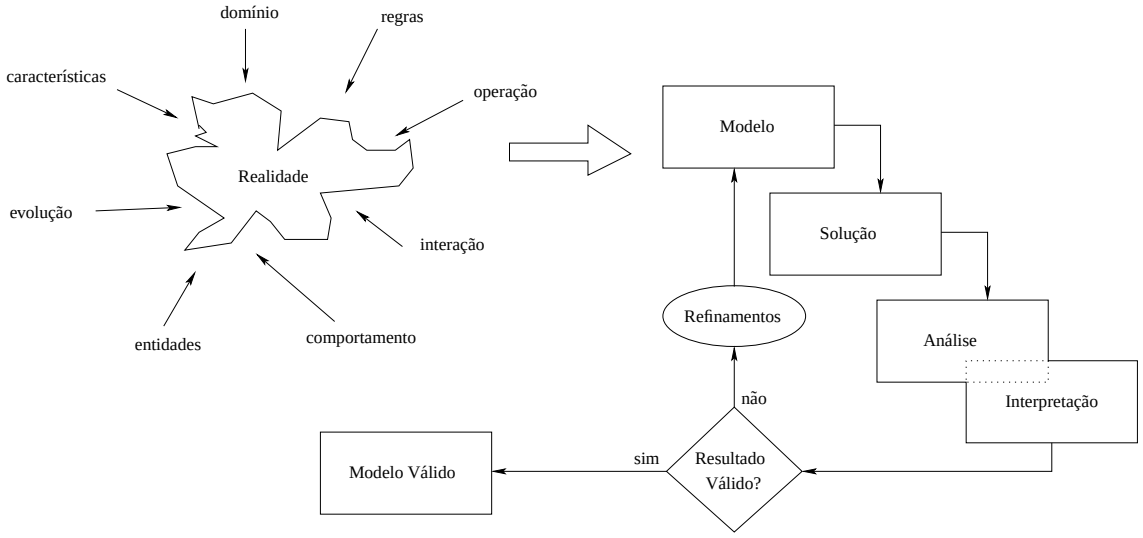


Figura 2.1: Processo de modelagem, abstração da realidade e captura de informações relevantes de um sistema.

A descrição destes relacionamentos está mostrada na Figura 2.1. Uma dada realidade pode ser pensada como um conjunto de entidades dentro de um certo domínio que interage através de um conjunto de regras. Estas entidades possuem estados e transitam entre estes com uma certa frequência com o disparo de eventos, que pode

Formalismo	Data dos primeiros trabalhos	Trabalhos relevantes
Cadeias de Markov	1906	[55][11][62][22]
Redes de Petri	1962	[64][2][1][18]
Redes de Petri Coloridas	1981	[50][52][51]
Redes de Autômatos Estocásticos	1985	[65][66]
Redes de Atividades Estocásticas	1985	[56]
Redes Bem-Formadas	1990	[19][20][21]
Álgebras de Processos	1970?	[10][43]
PEPA	1994	[44][45]
PEPA nets	2001	[40][41][38]
...	...	...

Tabela 2.1: Formalismos e seus principais trabalhos dentro da área de avaliação de desempenho.

ser consideradas operações sobre um estado do sistema que resultam em um novo estado. Após todos os aspectos considerados serem tomados em conta, inicia-se a fase de *formalização* propriamente dita através da definição de um modelo que compreende o processo que resultará na captura dos índices de desempenho. É importante abstrair detalhes e capturar apenas aspectos relevantes do sistema sob consideração. Esta abstração será determinante e ditará como a fase de solução irá executar. Após a solução, os índices devem ser devidamente inspecionados e interpretados na fase de análise, seguida por refinamentos do modelo que utiliza todo o *feedback* obtido na fase anterior. Com essas informações, altera-se o modelo ou atesta-se que constitui em uma modelagem válida que resultou em dados relevantes. Vale lembrar que esse processo é iterativo, ou seja, apenas para quando o modelador está confiante que os resultados obtidos mapearam a realidade de forma satisfatória.

A análise de modelos ajuda a prevenir inconsistências e comportamentos indesejados, tais como aqueles em que o sistema entra em algum estado inválido ou de erro e não consegue sair ou permanece em *loop*. Este é o caso das pesquisas realizadas sobre mecanismos de *verificação* de modelos (*Model Checking*) [3], *i.e.*, sistemas focados na determinação das sequências de passos (também chamados de caminhos) que um sistema segue com probabilidade p (nesse caso, esta seria apenas uma aplicação da área de verificação de sistemas, aqui explicada em um amplo sentido) e até mesmo quanto tempo ele permanece realizando alguma atividade. Trata-se de uma área ativa de pesquisa dentro da comunidade de avaliação de desempenho e seus resultados são extremamente importantes para validar sistemas complexos e inferir outros tipos de análises, verificando exaustivamente a execução do sistema através do modelo.

A Tabela 2.1 lista de forma ampla e geral os principais formalismos existentes na literatura. A tabela também mostra a data dos primeiros trabalhos onde foram aplicados para a extração de índices de desempenho bem como algumas extensões de cada formalismo que foram definidas com o passar do tempo.

2.2 Cadeias de Markov

Uma Cadeia de Markov (CM) é um processo estocástico[1] em tempo discreto ou contínuo com entidades bastante simples: estados e transições associadas a cada um destes. O formalismo foi proposto com um interesse bastante particular, baseado em textos literários (o que demonstra a versatilidade do formalismo, podendo ser utilizado de diversas maneiras em diferentes domínios do conhecimento). O objetivo da aplicação de CM foi inferir a probabilidade de, dado um texto e uma determinada posição desse mesmo texto, a próxima letra a ser analisada

[1]Um processo estocástico ou processo randômico é o oposto de um processo determinístico. Ao invés de se lidar com apenas uma realidade possível de como um processo evolui no tempo, em um processo estocástico existe uma qualidade indeterminada da sua evolução e é descrita através de distribuições de probabilidades.

ser uma vogal dado que o caractere atual é uma consoante. Trata-se de uma propriedade primordial de CMs, a chamada *memoryless property*. Esta propriedade estabelece que toda a informação relevante está contida no estado atual, não interessando por onde se passou anteriormente. Este formalismo ainda é muito utilizado, devido principalmente à sua simplicidade e também ao fato de retornar respostas para realidades variadas através de simples primitivas.

Este formalismo permaneceu desconhecido até ser utilizado no contexto de avaliação de desempenho de sistemas no MIT (*Massachussets Institute of Technology*), sendo aplicado com sucesso em sistemas de compartilhamento de tempo (*time sharing systems*) [70]. A principal observação realizada nesse trabalho foi a de que, mesmo o modelo sendo extremamente simples, conseguiu-se capturar a essência do sistema e providenciar respostas mesmo quando este era adaptado para refletir outros comportamentos.

Hoje em dia, CMs são utilizadas em inúmeros contextos, desde linguística até análise de riscos no mercado de ações. Um dos exemplos da utilidade deste formalismo são os sistemas de indexação e procura da Internet, onde o conjunto de informações existentes é claramente gigantesco e desorganizado. A dificuldade é encontrar páginas com conteúdo que seja relevante para o que se esteja procurando. O principal conceito aqui pode ser modelado como cada página na Internet ser um estado e a abstração é considerar uma entidade que apenas visita estas páginas (para efeitos de descrição, convenciona-se aqui chamá-la de *surfista*). A visitação é realizada de forma aleatória onde, a partir de uma página inicial, deseja visitar as páginas que estão conectadas à essa (em terminologia da Internet, cada página possui diversos *links*, ou seja, diversas transições para outras páginas).

Ao resolver esse sistema de equações massivo onde as variáveis são as páginas, o resultado prático é a probabilidade que surfistas aleatórios tenham visitado as páginas e com qual relevância. Ao descobrir e ordenar estas probabilidades de ocorrência em ordem decrescente (as com maiores probabilidades primeiro), são retornadas as páginas mais prováveis para o usuário. Esse conceito é utilizado pelo sistema de buscas da empresa de computação *Google* (algoritmo aqui descrito de uma forma muito simples) com resultados importantes [53][2].

No contexto de CM, é possível representar um sistema usando tempo contínuo (também chamadas de Cadeias de Markov em escala de tempo contínua – *Continuous Time Markov Chains*, ou CTMC) ou tempo discreto (Cadeias de Markov em escala de tempo discreta – *Discrete Time Markov Chain*, ou DTMC). A Figura 2.2 mostra um exemplo de Cadeia de Markov em escala de tempo contínua com 15 estados e uma taxa para cada transição, definidas como λ, μ, ν ou ξ, dependendo qual estado esteja-se analisando.

Todos os índices de desempenho são computados da distribuição estacionária ou transiente da Cadeia de Markov, resolvendo-se as equações de balanço global do sistema. Todas estas variáveis de entrada são extraídas da matriz de transição P da CM, onde a probabilidade p_{ij} significa a probabilidade condicional para sair do estado i e ir para o estado j, onde $i \neq j$. Para o caso de solução estacionária, a equação pode ser escrita para DTMCs como:

$$\pi P = \pi,$$

ou também, para o caso de CTMCs como:

$$\pi \tilde{\mathcal{Q}} = 0,$$

onde $\tilde{\mathcal{Q}} = \Delta T + I$ e $\Delta T \leq (max_i(|p_{ij}|))^{-1}$. Esta operação irá converter a representação em tempo contínuo representado pelas *taxas de transição* da matriz $\tilde{\mathcal{Q}}$ para uma representação baseada em tempo discreto com as *probabilidades de transição* da matriz P. Neste sistema discreto, as transições acontecem em intervalos de tempo ΔT. Este parâmetro é normalmente escolhido para que a probabilidade existente *entre* duas transições seja negligenciável [69, 70].

Logo, uma CM pode ser vista como um sistema de transição de estados onde o modelador define quais são acessíveis a partir de outros estados e configura uma taxa ou probabilidade para que um dado evento ocorra dentro

[2]Esta descrição encontra-se bastante simplificada neste trabalho, pois são utilizados algoritmos complexos de mapeamento, indexação e associação de *ranks* a páginas, normalmente através de implementações proprietárias inacessíveis à comunidade acadêmica por questões mercadológicas.

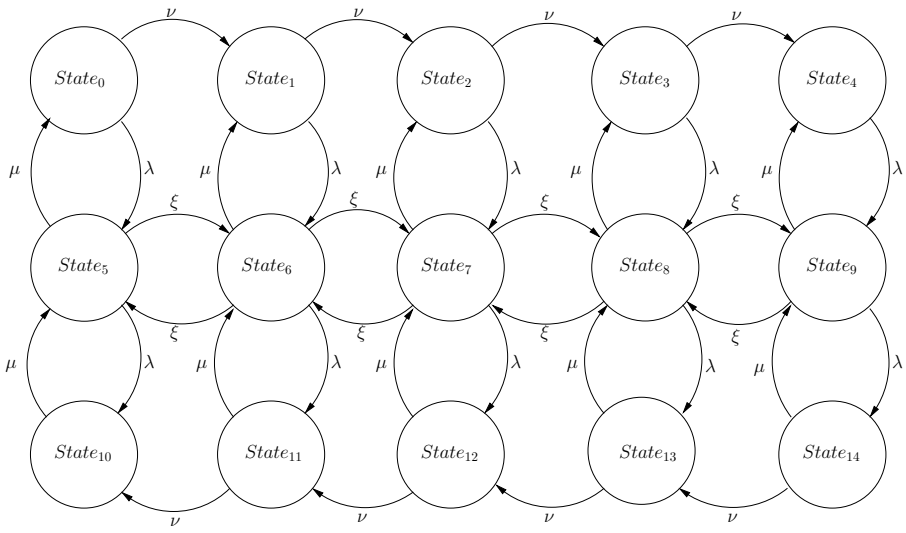


Figura 2.2: Um exemplo de Cadeia de Markov.

de um sistema. Métodos de solução direta desse sistema de equações tais como eliminação Gaussiana (entre outros) não são aplicáveis, pois requerem vastas quantidades de memória. Já técnicas iterativas de solução são melhores utilizadas, principalmente as que armazenam a matriz $\tilde{\mathcal{Q}}$ de forma esparsa, mas mesmo assim ainda existem limites quanto aos modelos que podem ser resolvidos desta forma. O resultado da multiplicação por um vetor v traduz a probabilidade de estar em um estado em um certo tempo, depois que o sistema está operacional. Existem duas distinções principais sobre a duração destas multiplicações em um determinado tempo t sobre o tempo total T. São chamadas de probabilidades *estacionárias* quando atingiram o equilíbrio em T e *transientes* quando para-se o processo em t e analisam-se os índices prévios de desempenho [69].

O presente trabalho não discutirá todos os detalhes pertencentes a Cadeias de Markov. Para maiores informações, consulte outros trabalhos da literatura de CMs [69, 70, 15, 68, 11, 62]. Um dos problemas centrais de CM é fato de haver a chamada *explosão do espaço de estados*, que acontece quando existe um número muito grande de estados, normalmente não oferecendo uma solução do seu sistema linear por falta de recursos computacionais tais como memória ou seu término em um tempo razoável.

2.2.1 Emergência dos formalismos estruturados

Apesar de amplamente utilizada, as CMs possuem problemas fundamentais, sendo o principal a explosão do espaço de estados tornando a solução do sistema impraticável em um tempo razoável. Este problema acompanha as CMs desde o seu início, onde verificou-se que mesmo simples descrições geravam muitos estados e isso implicava em problemas tanto na armazenagem da cadeia quanto nos mecanismos de solução envolvidos.

Para reduzir os efeitos catastróficos da explosão do espaço de estados foram consideradas outras formas de representar os sistemas, mas ainda assim permanecendo com a visão das Cadeias de Markov como alternativa de modelagem válida para sistemas. A solução foi a definição de formalismos estruturados e estes são usados para representar um sistema em um nível superior de abstração, mas possuindo uma Cadeia de Markov equivalente de forma implícita ou subjacente. Ao efetuar o produto cartesiano do espaço de estados (uma combinação de todos os estados passíveis de serem compostos) entre as entidades de um dado formalismo estruturado, por exemplo, autômatos em Redes de Autômatos Estocásticos ou processos em PEPA, produz-se, na verdade, uma Cadeia de Markov equivalente. Este produto cartesiano também é chamado de Espaço de Estados Produto $\mathcal{X}$ (*Product State Space*, ou PSS) e dita a quantidade de estados existentes no sistema.

Cabe ressaltar que os formalismos estruturados reduzem significativamente o problema mencionado acima da explosão do espaço de estados, mas não eliminando-o completamente. Inclusive, formalismos estruturados inserem uma outra problemática para ser pesquisada, relativa à existência de estados inatingíveis. Normalmente, o Espaço de Estados Inatingíveis $\mathcal{X}^R$ (*Reacheable State Space*, ou RSS) é um subconjunto do PSS, *i.e.*, $\mathcal{X}^R \subseteq \mathcal{X}$ [17, 59], entretanto, ainda permanece um problema em aberto a solução eficiente de modelos complexos onde todos, ou quase todos os estados são atingíveis. Estes novos problemas podem ser mitigados utilizando-se estruturas de dados sofisticadas com manipulação simbólica, tais como Diagramas de Decisão Multi-valorados (DDM) [59] ou através do uso de *vetores reduzidos* [8].

Na prática, representações estruturadas da matriz $\tilde{\mathcal{Q}}$ podem ser obtidas através de diferentes formalismos de modelagem: Redes de Petri Estocásticas [2] ou Redes de Autômatos Estocásticos [65, 66] ou até mesmo descrições bastante modulares e composicionais como PEPA [44] ou Gramática de Grafos [31]. O princípio geral de utilizar tensores para representação implícita desta matriz existe desde os primórdios da definição das Redes de Autômatos Estocásticos, mas recentemente observou-se a aplicação com êxito em outros formalismos estocásticos [29, 46].

As vantagens e desvantagens ao se adotar um formalismo variam de caso para caso e é necessário saber previamente as funcionalidades presentes em cada definição, maximizando-se, assim, as análises que são permitidas. O objetivo do resto deste capítulo é descrever outras importantes descrições para modelagem de sistemas sem escapar das técnicas e definições iniciais estabelecidas pelas CMs. O próximo formalismo explicado é o referente as Redes de Petri.

2.3 Redes de Petri

Redes de Petri, (RP) [4] são representações gráficas de modelagem para descrição de sistemas através do uso de anotações. Estruturas comuns de Redes de Petri são nós que correspondem a lugares (*place nodes*), nós que correspondem a transições (*transition nodes*) e arcos dirigidos que representam conexões entre lugares e transições. Dentro de cada lugar, existem marcas (*tokens*) e a distribuição de tais marcas dentro de uma rede é conhecida como uma marcação (*marking*). A maior vantagem da adoção de uma RP está no fato de possibilitar uma visão clara de causalidade e conflito nas regiões que compõem a rede, devido à maneira utilizada para representar o sistema. Uma extensão de RP são as chamadas Redes de Petri Estocásticas, (RPE) [36] , definidas pela introdução de tempo de disparo não-determinístico entre transições.

A Figura 2.3 mostra um exemplo simples de uma RP. A rede descrita pela figura contém quatro lugares e duas transições. A figura mostra como as marcas podem se movimentar de um lugar ao outro dentro da rede.

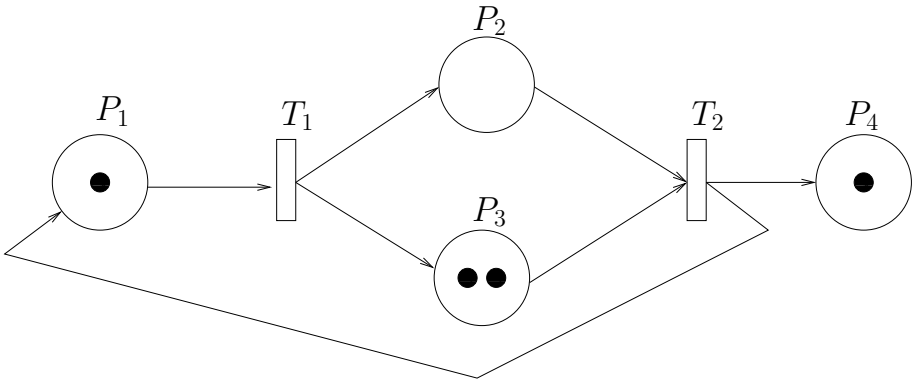


Figura 2.3: Uma Rede de Petri mostrando os lugares (P_1, P_2, P_3, P_4), transições (T_1, T_2) e as marcas (pequenos círculos em preto).

Uma importante vantagem de uma RP é sua característica ao descrever como funciona o fluxo de informações na rede e como operam as estruturas de controle e restrição de movimentação entre os lugares. A rede possui uma estrutura e uma sistema de marcação. A estrutura representa a parte estática do sistema. Os dois tipos de nós são representados por componentes gráficos, círculos para os lugares e retângulos para as transições. Os lugares correspondem aos estados do sistema e as transições a ações que forçam um estado a trocar para outro estado. Os arcos conectam lugares a transições e transições a lugares, sendo chamados, respectivamente, de *arcos de entrada* e *arcos de saída*.

A *marcação* de uma Rede de Petri é a associação de *marcas* a *lugares*. Para uma definição formal de uma RP é necessário especificar a estrutura da rede e a marcação inicial, *i.e.*, onde as marcas serão inicialmente colocadas dentro da rede. A dinâmica do sistema segue um conjunto básico de regras: a) uma transição ocorre quando o lugar de entrada cumpre as condições expressas pelas inscrições dos arcos [4]. O objetivo deste trabalho não é especificar formalmente uma RP, apenas oferecer um apanhado geral sobre este tipo de formalismo. Para maiores informações sobre RP e suas extensões, indica-se [4, 60, 61, 63, 67, 2], que mostram o funcionamento e explicam RPs com um maior nível de detalhe.

2.3.1 Redes de Petri Coloridas

RPs possuem extensões para capturar outros tipos de comportamento de sistemas. Uma extensão bastante conhecida são as Redes de Petri Coloridas (RPC. A principal distinção entre uma Rede de Petri tradicional e uma Rede de Petri Colorida reside nas informações contidas nas *marcas*. Em RPs elas são indistinguíveis (possuem uma mesma cor), enquanto que em RPCs, cada marca pode possuir associada a ela uma característica distinta (por exemplo, uma cor).

Uma RPC é uma linguagem de modelagem usada em realidades onde sincronizações e comunicações são necessárias, combinando as vantagens de Redes de Petri com conceitos de linguagens de programação de alto nível. RPCs utilizam as ideias que descrevem como as entidades cooperam e provê mecanismos para manipulação de estruturas de dados e associação a variáveis que são conceitos oriundos de linguagens de programação. Tais redes, como RP tradicionais, possuem lugares, transições e arcos. Adicionalmente, possuem um outro componente chamado *páginas*, que podem potencialmente conter lugares. O conjunto destes elementos formam uma rede que, quando usados em conjunto e mostrados graficamente, permitem a manipulação de estruturas complexas fornecendo auxílios visuais para reconhecer como as entidades estão conectadas entre si e como interagem.

RPCs podem ser usadas para verificação formal, pois oferece um conjunto de ferramentas desenvolvidas para este propósito específico. Existem dois tipos de métodos de verificação existentes para RPCs: análise do espaço de estados e análise de invariantes. O objetivo da verificação formal é provar matematicamente que um sistema possui um conjunto de propriedades comportamentais. À medida que sistemas atingem uma escala industrial, também tornou difícil sua análise e provas de corretude (*correctedness*). RPCs podem ser usadas para estes propósitos, aliado à outros mecanismos de validação utilizando-se simulação. Cabe ressaltar que a verificação formal de sistemas é normalmente direcionada a partes críticas dos sistemas, logo, é crucial fornecer uma interface razoável entre um dado sistema e seu modelo correspondente.

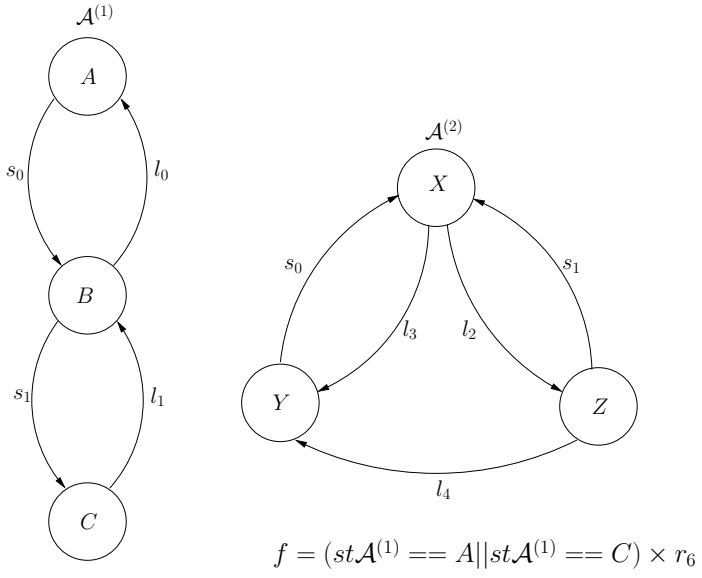
Dentre as vantagens de se adotar RPCs para modelar sistemas, destacam-se: simples primitivas para estabelecer e realizar inferências sobre a interação de componentes dentro de um sistema, modelagem de sub-componentes (também conhecida por modelagem hierárquica), descrição gráfica de sistemas e ferramentas para solução do sistema em questão (através de um software chamado *CPN Tools* [72]). Para outras informações sobre RPCs, sugere-se os seguintes trabalhos [50, 51] e mais recentemente [52].

2.4 Redes de Autômatos Estocásticos

Redes de Autômatos Estocásticos (*Stochastic Automata Networks*, ou SAN) [65, 66] é um formalismo amplamente utilizado para o cálculo de índices de desempenho na área da avaliação de sistemas. Este formalismo baseia-se em entidades semi-autônomas chamadas autômatos e como estes componentes se relacionam para sincronizar atividades e trocar de estados internamente. O conjunto destes autômatos forma uma rede, permitindo que sejam analisadas realidades complexas e foi construído para visar o correto mapeamento de atividades de paralelismo e distribuição. Em SAN, descrevem-se os sistemas de maneira estruturada, observando a CM equivalente e resolvendo este sistema implicitamente. Essa estruturação implica no armazenamento eficiente das estruturas de dados, permitindo que sistemas massivos sejam representados e tenham seus índices de desempenho devidamente interpretados e analisados.

Mais especificamente, uma SAN é um formalismo estruturado que descreve interações entre entidades conhecidas como *autômatos*. Cada autômato desempenha transições específicas na sua estrutura interna, *i.e.*, pode alterar seu comportamento local através de eventos locais e interagir com outros autômatos através de eventos sincronizantes. Um evento sincronizante deve envolver no mínimo dois autômatos e este evento representa que uma mudança de estado acontece concomitantemente nestes dois (ou mais) autômatos. Um evento é disparado de acordo com a definição de uma taxa de ocorrência. Essa taxa de ocorrência de um determinado evento pode ser de dois tipos, constante ou funcional. Uma taxa constante é uma frequência média com a qual a transição ocorre. Uma taxa funcional tem seu valor médio determinado segundo o estado de um ou mais autômatos, de acordo com fórmulas pré-estabelecidas pelos usuários quando modelam o sistema e as interações que são necessárias entre os diversos componentes.

A Figura 2.4 mostra um exemplo de uma SAN simples com eventos locais (*loc*) e sincronizantes (*syn*), contendo taxas constantes e funcionais. A figura mostra uma rede composta por dois autômatos, cooperando sobre um PSS de tamanho $|\mathcal{X}| = 3 \times 3 = 9$. O exemplo possui cinco eventos locais (chamados de l_0, l_1, l_2, l_3, l_4) e dois eventos sincronizantes (s_0, s_1). Uma das taxas é funcional (f) sendo as demais constantes. A taxa funcional será avaliada para r_6 quando o estado do autômato $\mathcal{A}^{(1)}$ for igual a A ou igual a C e para $zero$ (*i.e.*, não ocorrendo), caso contrário (nesse caso, igual a B).



Tipo	Evento	Taxa
syn	s_0	r_0
syn	s_1	r_1
loc	l_0	r_2
loc	l_1	r_3
loc	l_2	r_4
loc	l_3	r_5
loc	l_4	f

Figura 2.4: Um exemplo de SAN com eventos locais e sincronizantes, com taxas constantes e funcionais.

Um modelo SAN é transformado para um *descritor markoviano* (uma representação tensorial para o gerador infinitesimal $\tilde{\mathcal{Q}}$, também chamado de *descritor Kronecker*) [33, 65, 69], *i.e.*, um conjunto de matrizes responsáveis por definir a ocorrência dos eventos locais e sincronizantes de forma tensorial (a Seção 3.3 explica melhor os princípios referentes aos descritores). Após a definição do descritor, é utilizado um método para multiplicá-lo por um vetor de probabilidade, ou seja, efetuar a MVD (maiores informações na Seção 3.4). Cabe ressaltar, no entanto, que o uso de descritores tensoriais por SAN nem sempre gera a solução mais otimizada em termos de tempo gasto. Apesar de reduzir a memória necessária para resolver modelos, dependendo da realidade descrita com SAN, pode-se, algumas vezes, aumentar-se também o tempo necessário para a convergência de resultados válidos, em virtude do formato tensorial [14, 58].

Apesar das suas diferenças algorítmicas, as abordagens de MVD podem ser resumidas em encontrar uma forma eficiente de multiplicar um vetor (usualmente grande) por uma estrutura não trivial. Algumas soluções para RPE [2] traduziam a representação dos modelos em uma única matriz esparsa. Naturalmente, esta proposta esparsa é difícil de ser utilizada para modelos com muitos estados (*e.g.*mais de 500 mil estados), pois normalmente necessita salvar esta matriz contendo muitas vezes muitos elementos não-nulos (*e.g.*mais de 4 milhões). Isto é possível apenas com soluções sofisticadas para armazenar a matriz usando práticas baseadas em disco (*disk-based approaches*) [26], ou técnicas de geração *on-the-fly* [27] aliadas à computação paralela e distribuída [6, 28, 71].

Mais genericamente, estas matrizes são tratadas através da ferramenta PEPS (*Performance Evaluation of Parallel Systems*) [13]. Novos desenvolvimentos estão direcionados a três eixos de pesquisa: utilização de representação simbólica do RSS (usando-se estruturas mais complexas como DDM [23]), aceleração da MVD com o Algoritmo *Split* [24] e técnicas de Simulação Perfeita [35]. Maiores informações sobre métodos de solução de MVD encontram-se no Capítulo 3.

2.5 Álgebras de Processos

Os anos recentes mostraram um crescente aumento na adoção de Álgebras de Processo Markovianas (*Markovian Process Algebras* - MPA) voltadas para a construção de modelos que representem sistemas complexos. Exemplos incluem PEPA [44] (*Performance Evaluation Process Algebra*) e EMPA (*Extended Markovian Process Algebra*) [10], ou seja, formalismos desenvolvidos para capturar o comportamento de sistemas e permitir análise funcional através da solução da CTMC associada. Uma das principais atrações para o uso de álgebras de processo é a habilidade de se construir um modelo composicional baseado em uma definição em alto-nível. Tais álgebras permitem que sejam usadas em diversas aplicações, onde são facilmente modeladas através de simples primitivas que mostram como o sistema evolui de acordo com seu comportamento interno e de acordo com as interações existentes entre os componentes modelados.

2.5.1 PEPA – Performance Evaluation Process Algebra

Como em todas as álgebras de processo, sistemas são representados em PEPA como a composição de *componentes* que realizam *ações*. Em PEPA, as ações ocorrem conforme uma duração, logo, a expressão $(\alpha, r).P$ denota um componente o qual realizou uma ação α com taxa r e evoluiu para um componente P. Aqui $\alpha \in \mathcal{A}$ onde $\mathcal{A}$ é o conjunto de tipos de ações e $P \in \mathcal{C}$ onde $\mathcal{C}$ é o conjunto de tipos de componentes.

PEPA possui um conjunto restrito de combinadores, e as descrições dos sistemas são construídas a partir da execução concorrente e interação simples de componentes sequenciais. A sintaxe destes combinadores está informalmente definida abaixo, maiores detalhes são encontrados em [44].

- *Prefix*: O combinador *prefix* marca uma parada completa, dando ao componente a sua primeira ação designada. Como explicado anteriormente, $(\alpha, r).P$ executa uma ação α com taxa r, e subsequentemente começa a se comportar como P.

- *Choice*: O componente $P + Q$ representa um sistema que pode se comportar tanto como P ou como Q. As atividades de ambos P e Q estão autorizadas (*enabled*). A primeira atividade a acabar distingue-se (ou seja, é autorizada), a outra é descartada.

- *Constant*: É conveniente associar nomes a padrões comportamentais de componentes e constantes realizam essa tarefa definindo uma equação, *i.e.*, $X \stackrel{def}{=} E$. O nome X está no escopo da expressão do lado direito, significando que, por exemplo, $X \stackrel{def}{=} (\alpha, r).X$ desempenha α com taxa r para sempre.

- *Hiding*: A possibilidade de abstrair alguns aspectos do comportamento de componentes é realizado pelo operador *hiding*, denotado por P/L. O conjunto L identifica aquelas atividades as quais são consideradas internas ou privadas ao componente e quais irão aparecer como o tipo desconhecido τ.

- *Cooperation*: Escreve-se $P \underset{L}{\bowtie} Q$ para descrever cooperação entre P e Q sobre L. O conjunto L é o *conjunto de cooperação* e determina as atividades com as quais os componentes devem sincronizar atividades. Para tipos de ação fora de L, os componentes procedem independentemente e concorrentemente com suas atividades autorizadas. Escreve-se também $P \parallel Q$ como uma abreviação para $P \underset{L}{\bowtie} Q$ quando o conjunto L está vazio.

Para atividades do conjunto de cooperação, estas podem apenas completar a atividade quando ambas estiverem autorizadas. Os dois componentes então procedem juntos para completar esta atividade *compartilhada*. A taxa da atividade compartilhada pode ser alterada para refletir o trabalho realizado por ambos os componentes com a finalidade de acabar a atividade. A capacidade total do componente C para realizar tarefas do tipo α é definida pela *taxa aparente* de α em P, denotada por $r_\alpha(P)$. PEPA assume *bounded capacity*, significando que a taxa de uma atividade compartilhada é o mínimo das taxas das atividades dos componentes que estão cooperando.

Em alguns casos a taxa de alguma atividade é deixada sem especificação (denotada por $\top$) e é determinada quando existe cooperação, pela taxa da atividade do outro componente. Estas ações passivas devem ser sincronizadas no modelo final (forçando a ocorrência de uma taxa para a cooperação).

A sintaxe de PEPA pode ser formalmente introduzida a partir da seguinte gramática:

$$\begin{aligned} S &::= (\alpha, r).S \mid S + S \mid C_S \\ P &::= P \underset{L}{\bowtie} P \mid P/L \mid C \end{aligned}$$

onde S denota um *componente sequencial* e P denota um *componente de modelo* o qual é executado em paralelo. C é uma constante que denota tanto um componente sequencial quando um componente de modelo. C_S são constantes que denotam componentes sequenciais. O efeito desta separação sintática entre estes tipos de constantes é para restringir componentes PEPA para serem cooperações de processos sequenciais, uma condição necessária para existir processos Markovianos bem definidos[3].

O significado de uma expressão PEPA é dado pela semântica operacional estrutural que define como um modelo evolui, podendo ser aplicado exaustivamente para formar um sistema de rótulos denominado *Grafo de Derivação* (GD) representando em última análise o produto do espaço de estados do modelo. Trata-se de um grafo o qual cada nodo é uma forma sintática distinta, definida como derivativo (*derivative*) e cada arco representa uma atividade possível causando uma mudança de estado. O estado muda em conjunção com a *Matriz de Transição* (MT), que guarda o nome do evento e a sua taxa de ocorrência para cada dois estados. É importante notar que em PEPA a representação de estados é de fato um sistema de rótulos com *multi-transições* pois guarda a multiplicidade de arcos, particularmente quando componentes repetidos estão envolvidos. O GD pode ser considerado como um diagrama de transição de estados de uma CTMC; $q(P, Q)$ então denota a taxa de transição entre o derivativo

[3]Ou seja, a Cadeia de Markov deve ser *ergódica*, maiores informações em [69].

P e o derivativo Q. Análises de desempenho são feitas em termos da procura da distribuição estacionária de probabilidades. O GD também deve ser fortemente conectado para gerar uma CTMC válida e solúvel.

A seguir, um simples exemplo de um único componente o qual possui três derivativos, $P1$, $P2$ e $P3$, cada um com uma ação distinta com a mesma taxa de ocorrência r.

$$\begin{aligned} P_1 &\stackrel{def}{=} (start, r).P_2 \\ P_2 &\stackrel{def}{=} (run, r).P_3 \\ P_3 &\stackrel{def}{=} (stop, r).P_1 \\ & \quad (P_1 \parallel P_1) \end{aligned}$$

Apesar do fato de que em álgebras de processo possuir diversas vantagens, não existe um atrativo visual de modelagem, como verificado em outros formalismos, como Redes de Petri Coloridas, por exemplo. Esse fato motivou a definição do formalismo das PEPA nets, discutido a seguir.

2.5.2 PEPA nets

Com o objetivo de se criar um formalismo mais estruturado e com representações gráficas, foram definidas as PEPA nets em 2001 [40]. PEPA nets [41] é uma combinação sinergética entre Redes de Petri Coloridas [51] e PEPA. Os modelos são construídos como redes possuindo lugares que contém tanto componentes PEPA estáticos quanto móveis. Esse novo formalismo combina a composicionalidade de PEPA com a notação gráfica das Redes de Petri. PEPA nets oferecem mais expressividade para os modeladores, mas deve ser vista como complementar à PEPA pois alguns comportamentos são melhores definidos em um formalismo ou em outro [41].

PEPA nets foi desenhada para capturar sistemas que possuam dois tipos distintos de mudança de estados, em particular sistemas computacionais onde exista mobilidade. Em tais sistemas, é importante separar a progressão lógica da computação envolvida com a mobilidade de eventos que reconfiguram o sistema. Esse tipo de modelagem é a principal motivação para definição de PEPA nets. Para atingir esse objetivo foi descrita uma combinação de formalismos com componentes PEPA representando as transições locais e os disparos das Redes de Petri capturando as reconfigurações globais do sistema (a mobilidade). PEPA nets são Redes de Petri Coloridas onde as cores das marcas são componentes de PEPA [41]. Cada lugar da RP é um contexto de PEPA o qual pode conter componentes PEPA estáticos e conjuntos de cooperação restringindo a movimentação das marcas dentro dos lugares. As marcas podem se mover para um determinado lugar apenas se existe espaço para um componente do seu próprio tipo. Como PEPA nets baseiam-se em componentes PEPA, é demonstrado em [39] que um mapeamento existe entre estes dois formalismos para o caso de uma modelagem de um desempenho de ferramentas de engenharia de *sofware*.

Existem dois tipos de mudança de estados em uma PEPA net, que pode ser vista como mudanças microscópicas e mudanças macroscópicas. As mudanças microscópicas são locais, afetando componentes dentro dos seus contextos e são representadas usando-se *transições* governadas pela semântica usual de PEPA. Em contraste, as mudanças macroscópicas são globais e capturam o *disparo* de eventos no nível da rede. Estes disparos causam que as marcas sejam transferidas de um lugar para outro [41]. Cada lugar, ou contexto de PEPA, contém *células* especificamente definidas, que só podem ser ocupadas pelo tipo apropriado. Esta definição é fortemente tipada, ou seja, uma célula não pode ter uma marca que não seja a marca definida para esta célula.

Uma PEPA net distingue dois tipos de componentes: *marcas* e *componentes estáticos*. As marcas podem se movimentar entre os lugares (assim como ocorre em Redes de Petri), enquanto que os componentes estáticos não permitem movimentação na rede com alterações globais de estado, representando o ambiente dentro daquele lugar. Estes componentes estáticos podem ser de dois tipos: *stateless* ou *stateful*. Um componente *stateless* não possui derivações e simplesmente oferece uma ou mais ações para interação. Os componentes *stateful* em contraste, possuem um conjunto mais rico de comportamentos incluindo a possibilidade de definir derivativos ou estados.

Um exemplo simples de PEPA net seria uma entidade, denominada um *agente*, que visita diferentes lugares de uma rede. A Figura 2.5 mostra uma visualização gráfica deste sistema. Existem três lugares na rede chamados P_1, P_2 e P_3 e agentes pode se movimentar entre estes lugares de acordo com uma taxa. As transições locais acontecem dentro de cada lugar e os movimentos nas transições da rede causam mudanças globais ao sistema, na figura representadas pelos eventos *go* e *return*. A marcação inicial da rede é um agente em P_2. As definições formais da rede e o que representam os agentes dentro desse exemplo não fazem parte do escopo deste trabalho, maiores informações podem ser obtidas em [39].

PEPA nets podem ser naturalmente usadas no caso de existirem componentes que precisam *circular* em uma rede ou alterar sua posição entre lugares, comunicar com outros eventuais componentes que existam no lugar e retornar para seu lugar de origem. Este tipo de comportamento é mais facilmente perceptível devido à maneira gráfica que as PEPA nets utilizam para mostrar as interações envolvidas entre as diferentes entidades de um sistema e como estas cooperam para realizar atividades. Um outro exemplo válido são processos em uma rede de computadores, ou *grid*, que podem ser enviados para outros lugares de execução onde alteram seus estados locais, mas também ocorre uma movimentação global na rede. É evidente que existem muitas outras situações onde PEPA nets podem modelar realidades complexas e inferir resultados analíticos representativos.

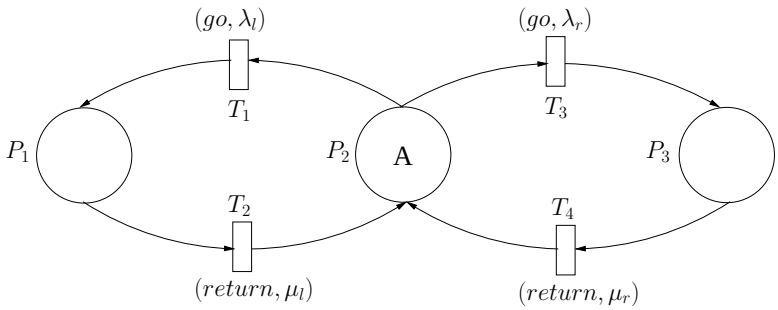


Figura 2.5: Um exemplo de uma PEPA net que descreve agentes móveis.

Tanto modelos PEPA quanto modelos PEPA nets são analisados através da ferramenta computacional *PEPA Workbench* (pwb). Este ambiente completo de análise oferece ferramentas para lidar com a compilação de modelos que geram tanto o GD quanto a MT incluindo opções de agregação de estados (reduzindo o PSS). Este ambiente também gera arquivos para entrada em outras ferramentas de uso matemático, como *Maple* ou *Mathematica* com vistas à solução dos modelos criados.

PEPA nets já foi utilizada para modelar mobilidade [48], modelagem de conceitos de Engenharia de Software [39], *Role-Playing Games* [42] e um ambiente para aplicações móveis usando *UML - Unified Modeling Language* [37], apenas para citar algumas aplicações. A variedade de tipos para modelagem evidenciam a versatilidade do formalismo e como este pode ser adaptado para descrever sistemas complexos.

Para uma completa definição de PEPA nets sugere-se a leitura de [39, 41], onde descrições detalhadas sobre o formalismo são melhor definidas e conceitualizadas. A seguir é apresentada uma breve discussão entre os formalismos expostos neste trabalho e uma comparação sobre as vantagens e desvantagens dos seu usos.

2.6 Discussão

As Cadeias de Markov foram o primeiro formalismo diretamente aplicado à avaliação de desempenho de sistemas computacionais, entretanto, seus conceitos principais podem ser aplicados a outras áreas tais como economia ou onde exista a necessidade de se descobrir probabilidades associadas a estados da realidade mesmo inválidos ou estados de erro. Desde a sua adoção, o problema da explosão do espaço de estados emergiu como um grave impedimento ao seu uso, fato mitigado pelas definições dos formalismos estruturados para análise de sistemas

complexos. As CMs demonstram uma utilidade e aplicabilidade sem restrições, devido principalmente à facilidade do seu uso, mas causam a explosão rápida do espaço de estados, a ponto de não ser possível mais tratá-la. Os formalismos estruturados também possuem o mesmo problema, mas são mais escaláveis e lidam com alternativas viáveis tanto quanto ao armazenamento das estruturas como outras pesquisas, com o objetivo de produzir resultados válidos em um tempo razoável.

Todos os formalismos possuem em comum comportamentos locais que importam a apenas algumas entidades e sincronizantes onde existe a necessidade de se trabalhar cooperativamente. Estes tipos de interação são facilmente adaptáveis para os problemas existentes em computação no que diz respeito à modelagem de comportamentos concorrentes em sistemas distribuídos (entre outros exemplos). Para escolher um formalismo para avaliar um sistema implica em decidir as vantagens e desvantagens existentes entre composicionalidade, localidade, sincronismo, evolução, interação, abstração e percepção (por exemplo, sistemas com representações gráficas).

Pesquisas atuais centram seus esforços nas descoberta do aumento do poder de representação dos sistemas para lidar apropriadamente com a abstração oferecida pelos diferentes formalismos estruturados. Outro tópico importante é melhorar ainda mais os mecanismos de análise e simulação que são oferecidos aos usuários, bem como inferências estatísticas que são geradas nos modelos. Estas análises permitem que sejam descobertos os fatores mais importantes que melhoram ou degradam o desempenho de sistemas, permitindo que os modeladores decidam as melhores condições para a sua operação otimizada bem como através do uso racional dos recursos.

Mais especificamente, ao comparar dois formalismos bastante utilizados como SAN e PEPA nets, constata-se que estes compartilham similaridades, por exemplo, ambos lidam com representações gráficas de sistemas. O mesmo acontece para o caso das Redes de Petri e Redes de Petri Coloridas. Esses recursos visuais auxiliam e simplificam a análise e inspeção de modelos. Uma clara vantagem de PEPA e PEPA nets sobre SAN diz respeito ao produto do espaço de estados (a combinação de todos os estados). Em SAN, pode-se ter que trabalhar com um espaço de estados atingível muito menor que o espaço produto necessário para a solução do modelo. Em PEPA, apenas os estados atingíveis são considerados. Entretanto, em termos de solução, SAN por trabalhar com descritores markovianos, emprega melhor os recursos, principalmente os de memória, para calcular o vetor de probabilidades enquanto que em PEPA, precisa-se incorporar a Matriz de Transição com as taxas dos eventos em uma ferramenta computacional de matemática, como por exemplo, o *Maple*.

Dadas as inúmeras opções existentes de formalismos markovianos, pode-se constatar que na verdade existe um excesso de diferentes formas de modelar sistemas. Entretanto, cada formalismo captura diferentes aspectos de um sistema incorporando variadas formas de análise. Alguns formalismos oferecem opções sofisticadas para representar a interação entre os diversos componentes e comportamentos enquanto que outros se preocupam com a abstração das realidades. Os pesquisadores de formalismos comumente adaptam ideias de diferentes formalismos em suas próprias definições e visões, com o intuito de aumentar o poder de representação e solução. Muitas vezes, são oferecidas maneiras de traduzir modelos escritos de um dado formalismo para outro. Tratam-se de regras que mapeiam cada tipo de ocorrência e como esta deve ser vista no sistema como um todo. O presente trabalho usará estas regras para oferecer uma solução otimizada baseada em descritores tensoriais para qualquer formalismo onde exista esse mapeamento.

O próximo capítulo apresenta tanto a concepção dos descritores markovianos como as propriedades da álgebra tensorial e os principais algoritmos de MVD.

3 Solução de descritores tensoriais

Este capítulo abordará a multiplicação de um vetor de probabilidade por uma estrutura mais complexa, ou seja, por um descritor markoviano. A formação deste descritor corresponde ao *gerador infinitesimal* de um sistema. Quando esta estrutura é multiplicada por um vetor inicial de probabilidades inúmeras vezes, pode ou não convergir para um vetor que encontra-se na estacionariedade, ou seja, um vetor que atingiu o estado de equilíbrio (para o caso de solução estacionária ao invés de solução transiente). Em todos os formalismos estruturados com representação tensorial existem operações que são efetuadas para gerar implicitamente este gerador infinitesimal. A seguir serão definidos os principais conceitos de álgebra tensorial clássica e generalizada nas Seções 3.1 e 3.2, bem como suas propriedades. O capítulo continua com a definição de descritores tensoriais na Seção 3.3 e finaliza com os principais algoritmos existentes de MVD na Seção 3.4.

3.1 Álgebra Tensorial Clássica - ATC

Para resolver estruturas tensoriais é importante definir as principais operações existentes na álgebra tensorial. No contexto deste trabalho, as operações mais importantes são o produto tensorial e a soma tensorial, melhor explicadas a seguir.

3.1.1 Produto tensorial

Por exemplo, sejam as matrizes A e B definidas como:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$$

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

$$C = \begin{pmatrix} a_{1,1}B & a_{1,2}B \\ a_{2,1}B & a_{2,2}B \end{pmatrix} = \left(\begin{array}{ccc|ccc} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,1}b_{1,3} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} & a_{1,2}b_{1,3} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,1}b_{2,3} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} & a_{1,2}b_{2,3} \\ a_{1,1}b_{3,1} & a_{1,1}b_{3,2} & a_{1,1}b_{3,3} & a_{1,2}b_{3,1} & a_{1,2}b_{3,2} & a_{1,2}b_{3,3} \\ \hline a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,1}b_{1,3} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} & a_{2,2}b_{1,3} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,1}b_{2,3} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} & a_{2,2}b_{2,3} \\ a_{2,1}b_{3,1} & a_{2,1}b_{3,2} & a_{2,1}b_{3,3} & a_{2,2}b_{3,1} & a_{2,2}b_{3,2} & a_{2,2}b_{3,3} \end{array}\right)$$

O produto tensorial $C = A \otimes B$ é definido algebricamente pela atribuição do valor $a_{i,j}b_{k,l}$ ao elemento dentro da posição (k, l) do bloco (i, j), *i.e.*:

$$c_{[i,k],[j,l]} = a_{i,j}b_{k,l} \quad \text{onde } i \in [1..\alpha_1],\ j \in [1..\alpha_2],\ k \in [1..\beta_1] \text{ e } l \in [1..\beta_2] \tag{3.1}$$

Esta representação de elementos de uma matriz corresponde ao produto tensorial realizado sobre os elementos $c_{[i,k],[j,l]}$ que estão em ordem lexicográfica de acordo com os seus índices (ou seja, na ordem que foram originalmente definidos).

3.1.2 Soma tensorial

A soma tensorial utiliza um produto tensorial para ser calculada. Sejam duas matrizes quadradas A e B, sua soma tensorial é definida como a soma dos fatores normais de cada matriz, de acordo com a seguinte fórmula:

$$A \oplus B = (A \otimes I_{n_B}) + (I_{n_A} \otimes B) \tag{3.2}$$

onde I_{n_A} e I_{n_B} correspondem a matrizes identidade de dimensões A e B respectivamente.

Por exemplo, sejam A e B matrizes definidas como:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$$

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

$$C = \left(\begin{array}{ccc|ccc} a_{1,1} & 0 & 0 & a_{1,2} & 0 & 0 \\ 0 & a_{1,1} & 0 & 0 & a_{1,2} & 0 \\ 0 & 0 & a_{1,1} & 0 & 0 & a_{1,2} \\ \hline a_{2,1} & 0 & 0 & a_{2,2} & 0 & 0 \\ 0 & a_{2,1} & 0 & 0 & a_{2,2} & 0 \\ 0 & 0 & a_{2,1} & 0 & 0 & a_{2,2} \end{array}\right) + \left(\begin{array}{ccc|ccc} b_{1,1} & b_{1,2} & b_{1,3} & 0 & 0 & 0 \\ b_{2,1} & b_{2,2} & b_{2,3} & 0 & 0 & 0 \\ b_{3,1} & b_{3,2} & b_{3,3} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & 0 & 0 & b_{2,1} & b_{2,2} & b_{2,3} \\ 0 & 0 & 0 & b_{3,1} & b_{3,2} & b_{3,3} \end{array}\right)$$

$$C = \left(\begin{array}{ccc|ccc} a_{1,1}+b_{1,1} & 0 & 0 & a_{1,2} & 0 & 0 \\ b_{2,1} & a_{1,1}+b_{2,2} & 0 & 0 & a_{1,2} & 0 \\ b_{3,1} & 0 & a_{1,1}+b_{3,3} & 0 & 0 & a_{1,2} \\ \hline a_{2,1} & 0 & 0 & a_{2,2}+b_{1,1} & 0 & 0 \\ 0 & a_{2,1} & 0 & 0 & a_{2,2}+b_{2,2} & 0 \\ 0 & 0 & a_{2,1} & 0 & 0 & a_{2,2}+b_{3,3} \end{array}\right)$$

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

$$c_{[i,k],[j,l]} = a_{i,j}\delta_{k,l} + \delta_{i,j}b_{k,l} \quad \text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \tag{3.3}$$

O operador produto tensorial ($\otimes$) tem precedência sobre o operador da soma tensorial ($\oplus$) e estes dois operadores tensoriais, por sua vez, também têm maior precedência sobre os operadores tradicionais de multiplicação e soma ($\times$ e $+$).

3.1.3 Propriedades

As propriedades da ATC de interesse para as SAN são listadas a seguir. Suas demonstrações podem ser vistas em [33].

- Associatividade da soma e do produto tensorial:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{3.4}$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \tag{3.5}$$

- Distributividade com relação à soma convencional:

$$(A + B) \otimes (C + D) = (A \otimes C) + (B \otimes C) + (A \otimes D) + (B \otimes D) \tag{3.6}$$

- Compatibilidade com a multiplicação convencional:

$$(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D) \tag{3.7}$$

- Compatibilidade com a transposição de matrizes:

$$(A \otimes B)^T = A^T \otimes B^T \tag{3.8}$$

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

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{3.9}$$

- Decomposição em fatores normais:

$$A \otimes B = (A \otimes I_{n_B}) \times (I_{n_A} \otimes B) \tag{3.10}$$

- Distributividade com relação à multiplicação pela matriz identidade:

$$(A \times B) \otimes I_n = (A \otimes I_n) \times (B \otimes I_n) \tag{3.11}$$

$$I_n \otimes (A \times B) = (I_n \otimes A) \times (I_n \otimes B) \tag{3.12}$$

- Comutatividade dos fatores normais:

$$(A \otimes I_{n_B}) \times (I_{n_A} \otimes B) = (I_{n_A} \otimes B) \times (A \otimes I_{n_B}) \tag{3.13}$$

3.2 Álgebra Tensorial Generalizada - ATG

A álgebra tensorial generalizada é uma extensão da álgebra tensorial clássica, e tem como principal objetivo permitir a utilização de objetos que são funções discretas sobre linhas de uma matriz. Trabalha-se nas matrizes com elementos passíveis de avaliações diferentes, ou seja, tem-se uma matriz que pode ter diferentes instâncias de acordo com uma avaliação.

A diferença fundamental da ATG com relação à ATC é a introdução do conceito de *elementos funcionais*. Entretanto, 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, *i.e.*, o domínio dessa função é $\mathbb{R}^n$ e seu contra–domínio é $\mathbb{R}$.

Um elemento funcional $b(\mathcal{A})$ é dito *dependente* da matriz $\mathcal{A}$ se algum índice de linha da matriz $\mathcal{A}$ pertencer ao conjunto de parâmetros desse elemento funcional. Uma matriz que contém ao menos um elemento funcional dependente da matriz $\mathcal{A}$ é dita *dependente* da matriz $\mathcal{A}$.

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

- produto tensorial generalizado $\underset{g}{\otimes}$;
- soma tensorial generalizada $\underset{g}{\oplus}$.

A notação definida na Seção 3.1 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 é considerada dependente da matriz C somente);

$a_{ij}(b_k,\mathcal{C})$ 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, ela é 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 $B(\mathcal{A})$.

3.2.1 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:

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

Os elementos da matriz A variam em função dos elementos da matriz B por isso a denominação $A(\mathcal{B})$, ocorrendo o mesmo para a matriz B, onde seus elementos variam em função da matriz A, ou seja, $B(\mathcal{A})$. 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.*:

$$\begin{aligned} c_{[ik][jl]} &= a_{ij}(b_k)b_{kl}(a_i) \\ &\text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \end{aligned} \tag{3.14}$$

3.2.2 Soma Tensorial Generalizada

A soma tensorial generalizada é definida utilizando-se o conceito de matriz identidade com o produto tensorial generalizado da Equação 3.15:

$$A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A}) = (A(\mathcal{B}) \underset{g}{\otimes} I_{n_B}) + (I_{n_A} \underset{g}{\otimes} B(\mathcal{A})) \tag{3.15}$$

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.*:

$$c_{[ik][jl]} = a_{ij}(b_k)\delta_{kl} \; + \; b_{kl}(a_i)\delta_{ij} \qquad \text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \tag{3.16}$$

3.2.3 Propriedades

Em ATG, são definidas as seguintes propriedades fundamentais:

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

$$\begin{aligned}[A(\mathcal{C},\mathcal{D}) + B(\mathcal{C},\mathcal{D})] \underset{g}{\otimes} [C(\mathcal{A},\mathcal{B}) + D(\mathcal{A},\mathcal{B})] = \\ A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \; + \; A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B}) + \\ B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \; + \; B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B})\end{aligned} \tag{3.17}$$

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

$$[\, A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} B(\mathcal{A},\mathcal{C}) \,] \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} [\, B(\mathcal{A},\mathcal{C}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \,] \tag{3.18}$$

$$[\, A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} B(\mathcal{A},\mathcal{C}) \,] \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} [\, B(\mathcal{A},\mathcal{C}) \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) \,] \tag{3.19}$$

- Distributividade com relação à multiplicação pela matriz identidade:

$$[\, A(\mathcal{C}) \times B(\mathcal{C}) \,] \underset{g}{\otimes} I_{n_C} = A(\mathcal{C}) \underset{g}{\otimes} I_{n_C} \; \times \; B(\mathcal{C}) \underset{g}{\otimes} I_{n_C} \tag{3.20}$$

$$[\, I_{n_C} \underset{g}{\otimes} A(\mathcal{C}) \,] \times B(\mathcal{C}) = I_{n_C} \underset{g}{\otimes} A(\mathcal{C}) \; \times \; I_{n_C} \underset{g}{\otimes} B(\mathcal{C}) \tag{3.21}$$

- Decomposição em fatores normais I:

$$A \underset{g}{\otimes} B(\mathcal{A}) = I_{n_A} \underset{g}{\otimes} B(\mathcal{A}) \; \times \; A \underset{g}{\otimes} I_{n_B} \tag{3.22}$$

- Decomposição em fatores normais II:

$$A(\mathcal{B}) \underset{g}{\otimes} B = A(\mathcal{B}) \underset{g}{\otimes} I_{n_B} \; \times \; I_{n_A} \underset{g}{\otimes} B \tag{3.23}$$

- Decomposição em produto tensorial clássico:

$$A \underset{g}{\otimes} B(\mathcal{A}) = \sum_{k=1}^{n_A} \ell_k(A) \underset{g}{\otimes} B(a_k) \tag{3.24}$$

Uma vez descritas as operações existentes em álgebra tensorial (tanto clássica quanto generalizada), é possível dar seguimento ao capítulo através da discussão dos descritores markovianos.

3.3 Descritores markovianos

Sistemas representados por redes de autômatos estocásticos ou outros formalismos estruturados permitem uma visão modular onde as interações entre os diferentes subsistemas são modelados por taxas funcionais ou por taxas sincronizantes. O comportamento independente dos autômatos pode ser modelado por taxas locais, ou seja, taxas que não são alteradas pelos estados de outros autômatos. O descritor markoviano é o gerador infinitesimal $\tilde{\mathcal{Q}}$ da Cadeia de Markov quando as diferentes matrizes são expressas no formato tensorial, de acordo com a Equação 3.25. As diferentes operações que podem ser efetuadas (por exemplo, soma tensorial ou produto tensorial) reproduzem as interações e igualmente como os diferentes autômatos sincronizam atividades.

$$\tilde{\mathcal{Q}} = \bigoplus_{i=1}^{N} \mathcal{Q}_l^{(i)} + \sum_{e \in E} \left(\bigotimes_{i=1}^{N} \mathcal{Q}_{e^+}^{(i)} + \bigotimes_{i=1}^{N} \mathcal{Q}_{e^-}^{(i)} \right) \tag{3.25}$$

onde
- $\mathcal{Q}_l$ são matrizes que representam a ocorrência e o ajuste da diagonal de eventos locais;
- $\mathcal{Q}_{e^+}$ são matrizes que representam a ocorrência de eventos sincronizantes;
- $\mathcal{Q}_{e^-}$ são matrizes que representam o ajuste diagonal dos eventos sincronizantes.

Somas tensoriais são na verdade produtos tensoriais efetuados em matrizes do tipo *identidade* (onde substitui-se uma dada matriz por sua matriz identidade equivalente em termos de dimensão). Com isso, pode-se simplificar a Equação 3.25 em termos de notação do descritor markoviano para:

$$\tilde{\mathcal{Q}} = \sum_{j}^{|\mathcal{L}|} \bigotimes_{i=1}^{N} \mathcal{Q}_j^{(i)} \tag{3.26}$$

onde $\mathcal{L}$ é um conjunto de termos tensoriais do tipo $\{l, e^+, e^-\}$ com cardinalidade $|\mathcal{L}| = N + 2E$.

3.3.1 O efeito dos eventos no descritor markoviano

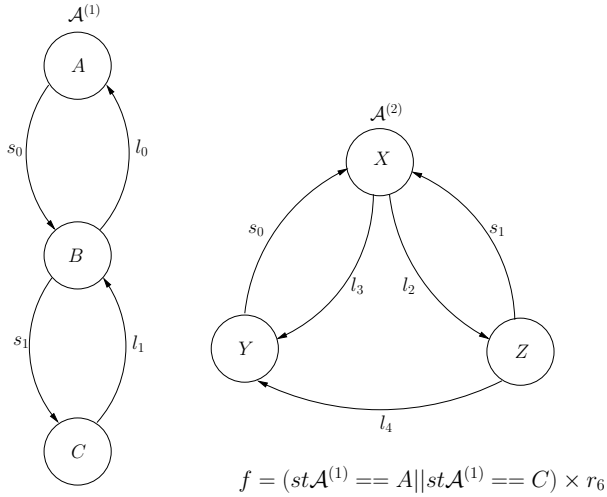
Esta seção discutirá o efeito de cada tipo de evento em um descritor markoviano, ou seja, levantará as implicações em se definir em um dado modelo se um evento é local ou sincronizante ou se a sua taxa é constante ou funcional. Esta análise iniciará pelos eventos locais (inclusive com taxas funcionais), prosseguindo para os eventos sincronizantes. O modelo de estudo possui cinco eventos locais, l_0, l_1, l_2, l_3 e l_4. No descritor markoviano eles correspondem a duas matrizes (devido à existência de dois autômatos) de dimensão três (pois ambos autômatos possuem três estados).

Antes de começar a compor o descritor markoviano com as taxas relacionadas aos autômatos em questão é necessário outras definições no que tange a Cadeia de Markov almejada nesse exemplo. Será utilizada uma SAN de exemplo mostrada na Figura 3.1. Esta rede é composta por dois autômatos de três estados cada e possui eventos locais e sincronizantes com taxas constantes e funcionais.

Para este caso, como existem dois autômatos de três estados cada, o seu PSS (seu Produto do Espaço de Estados, de acordo com a Seção 2.2.1) corresponde a $|\mathcal{X}| = 3 \times 3 = 9$, sendo este o produto cartesiano dos estados do autômato $\mathcal{A}^{(1)} = \{A, B, C\}$ pelos estados do autômato $\mathcal{A}^{(2)} = \{X, Y, Z\}$, ou seja, o conjunto $\mathcal{X} = \{AX, AY, AZ, BX, BY, BZ, CX, CY, CZ\}$.

Logo, a Figura 3.2 mostra a matriz resultante que corresponde ao gerador infinitesimal $\tilde{Q}$ da CM. Cabe ressaltar que esta matriz encontra-se vazia para efeitos de explicação do método. A forma correta de representá-la é preenchendo-se as células com as taxas que mostram a frequência com a qual troca-se de estado na CM.

A seguir, inicia-se a análise pelos eventos locais desta rede de autômatos estocásticos.



Tipo	Evento	Taxa
syn	s_0	r_0
syn	s_1	r_1
loc	l_0	r_2
loc	l_1	r_3
loc	l_2	r_4
loc	l_3	r_5
loc	l_4	f

Figura 3.1: Um exemplo de SAN para conversão na Cadeia de Markov equivalente.

		A			B			C		
		X	Y	Z	X	Y	Z	X	Y	Z
A	X									
	Y		...							
	Z									
B	X									
	Y					...				
	Z									
C	X									
	Y								...	
	Z									

Figura 3.2: Matriz correspondente ao produto cartesiano dos estados dos Autômatos $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(2)}$.

Parte local correspondente aos autômatos $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(2)}$:

$$\mathcal{Q}_l = \mathcal{Q}_l^{\mathcal{A}^{(1)}} \oplus \mathcal{Q}_l^{\mathcal{A}^{(2)}} = \begin{pmatrix} 0 & 0 & 0 \\ r_2 & -r_2 & 0 \\ 0 & r_3 & -r_3 \end{pmatrix} \oplus \begin{pmatrix} -(r_5+r_4) & r_5 & r_4 \\ 0 & 0 & 0 \\ 0 & r_6 & -r_6 \end{pmatrix}$$

Seja $A = \mathcal{Q}_l^{\mathcal{A}^{(1)}}$ e $B = \mathcal{Q}_l^{\mathcal{A}^{(2)}}$. Segundo a Equação 3.2, $A \oplus B = (A \otimes I_{n_B}) + (I_{n_A} \otimes B)$. É necessário calcular cada parte individualmente, começando por $A \otimes I_{n_B}$:

$$A \otimes I_{n_B} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 \end{array}\right)$$

Uma vez calculado $A \otimes I_{n_B}$, é necessário calcular $I_{n_A} \underset{g}{\otimes} B$ utilizando as propriedades da ATG para esse evento em particular, pois deve-se avaliar a função f:

$$I_{n_A} \underset{g}{\otimes} B = \left(\begin{array}{ccc|ccc|ccc} -(r_5+r_4) & r_5 & r_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_6 & -r_6 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & -(r_5+r_4) & r_5 & r_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & -(r_5+r_4) & r_5 & r_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_6 & -r_6 \end{array}\right)$$

Cabe ressaltar que valores nulo marcados em negrito correspondem a avaliações de elementos funcionais que resultam no valor zero, pois essas linhas equivalem ao autômato $\mathcal{A}^{(1)}$ estar no estado B. Para o caso da função f definida no modelo, é retornada uma avaliação para *falso* que é então convertida para o valor zero, e consequentemente, não definindo a taxa r_6 para essa linha (como seria a ocorrência normal).

Somando-se os dois termos que foram previamente calculados, tem-se que:

$$A \oplus B = \left(\begin{array}{ccc|ccc|ccc} -(r_5+r_4) & r_5 & r_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_3 & -r_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline r_2 & 0 & 0 & -(r_5+r_4+r_2) & r_5 & r_4 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & r_3 & 0 & 0 & -(r_5+r_4+r_3) & r_5 & r_4 \\ 0 & 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_3 & 0 & r_6 & -(r_3+r_6) \end{array}\right)$$

A seguir, calculam-se os termos correspondentes aos eventos sincronizantes (parte positiva) e ao ajuste da diagonal (parte negativa):

Parte sincronizante positiva da ocorrência do evento $\mathbf{s_0}$:

$$\mathcal{Q}_{s_{0+}} = \mathcal{Q}_{s_{0+}}^{\mathcal{A}^{(1)}} \otimes \mathcal{Q}_{s_{0+}}^{\mathcal{A}^{(2)}} = \begin{pmatrix} 0 & s_0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & s_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{s_0}$:

$$\mathcal{Q}_{s_{0-}} = \mathcal{Q}_{s_{0-}}^{\mathcal{A}^{(1)}} \otimes \mathcal{Q}_{s_{0-}}^{\mathcal{A}^{(2)}} = \begin{pmatrix} -s_0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -s_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{s_1}$:

$$\mathcal{Q}_{s_{1+}} = \mathcal{Q}_{s_{1+}}^{\mathcal{A}^{(1)}} \otimes \mathcal{Q}_{s_{1+}}^{\mathcal{A}^{(2)}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & s_1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & s_1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{s_1}$:

$$\mathcal{Q}_{s_{1-}} = \mathcal{Q}_{s_{1-}}^{\mathcal{A}^{(1)}} \otimes \mathcal{Q}_{s_{1-}}^{\mathcal{A}^{(2)}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -s_1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

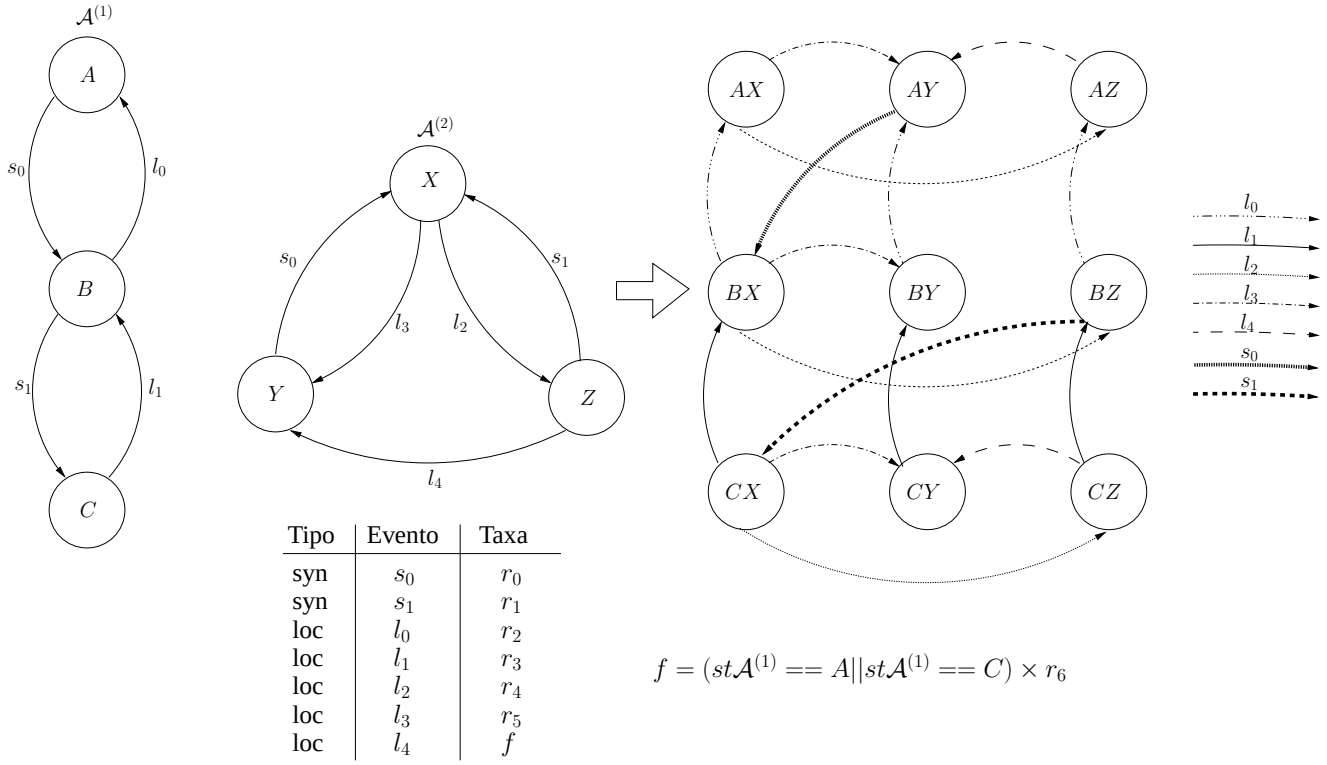
O gerador infinitesimal é dado por:

$$\tilde{\mathcal{Q}} = \mathcal{Q}_l + \left(\mathcal{Q}_{s_{0+}} + \mathcal{Q}_{s_{0-}}\right) + \left(\mathcal{Q}_{s_{1+}} + \mathcal{Q}_{s_{1-}}\right)$$

A Figura 3.3 mostra o gerador infinitesimal produzido a partir deste exemplo de conversão de uma SAN em uma Cadeia de Markov. Nota-se que esta matriz final já conta com os ajustes diagonais necessários (devido aos eventos sincronizantes negativos dos termos que foram criados) e a soma de cada linha resulta em zero.

		A			B			C		
		X	Y	Z	X	Y	Z	X	Y	Z
A	X	$-(r_5+r_4)$	r_5	r_4	0	0	0	0	0	0
	Y	0	$-s_0$	0	s_0	0	0	0	0	0
	Z	0	r_6	$-r_6$	0	0	0	0	0	0
B	X	r_2	0	0	$-(r_5+r_4+r_2)$	r_5	r_4	0	0	0
	Y	0	r_2	0	0	$-r_2$	0	0	0	0
	Z	0	0	r_2	0	0	$-(r_2+s_1)$	s_1	0	0
C	X	0	0	0	r_3	0	0	$-(r_5+r_4+r_3)$	r_5	r_4
	Y	0	0	0	0	r_3	0	0	$-r_3$	0
	Z	0	0	0	0	0	r_3	0	r_6	$-(r_3+r_6)$

Figura 3.3: Gerador infinitesimal ($\tilde{\mathcal{Q}}$) da matriz resultante da conversão de uma SAN para uma Cadeia de Markov.



Tipo	Evento	Taxa
syn	s_0	r_0
syn	s_1	r_1
loc	l_0	r_2
loc	l_1	r_3
loc	l_2	r_4
loc	l_3	r_5
loc	l_4	f

$$f = (st\mathcal{A}^{(1)} == A || st\mathcal{A}^{(1)} == C) \times r_6$$

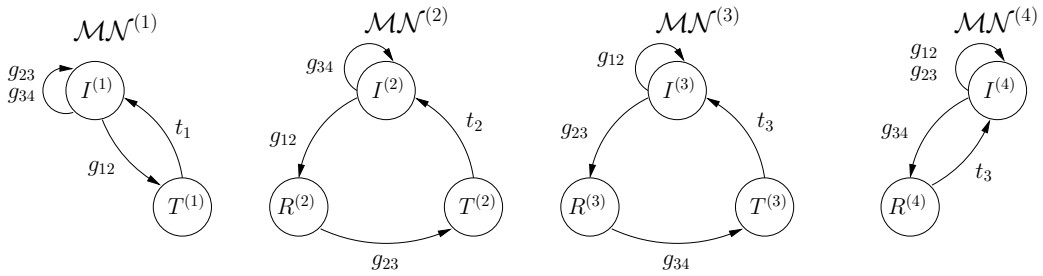
Figura 3.4: Conversão de uma SAN para uma Cadeia de Markov equivalente.

A Figura 3.4 mostra que para converter uma SAN para uma Cadeia de Markov equivalente é necessário iterar sobre o produto dos estados, produzindo todas as combinações possíveis. Isso evidencia o fato da modelagem por SAN ser mais compacta e compartimentalizada, como mencionado anteriormente. Entretanto, ao realizar essa combinação, pode potencialmente gerar estados que nunca serão atingidos (esse problema não ocorre neste modelo em particular com um PSS de nove estados, somente para casos com PSS mais elevados). Esta figura mostra, na parte direita, diferentes tipos de linha (algumas pontilhadas, outras maiores, etc) para cada evento correspondente à SAN. Ressalta-se que em CM não existe distinções quanto as transições, *i.e.*, todas são indistinguíveis. Decidiu-se mostrar assim para que fosse entendido o mapeamento de cada tipo de evento da SAN e sua transição correspondente na CM.

Nota-se que, para a solução de SAN, é suficiente guardar em memória apenas as pequenas matrizes que compõem o descritor, sendo desnecessário salvar o gerador infinitesimal $\tilde{\mathcal{Q}}$ correspondente à Cadeia de Markov. Essa é a principal vantagem de SAN sobre outros formalismos estruturados, pois baseia-se no uso da álgebra tensorial para calcular o vetor de probabilidade estacionário ou transiente do qual são extraídos posteriormente os índices de desempenho.

3.3.2 Um descritor markoviano de um exemplo real

Este descritor está baseado no modelo *Wireless Sensor Networks* para quatro nodos (mostrado na Figura 3.5 e descrito na Seção 5.2.2). Este modelo possui quatro autômatos, sendo dois autômatos com dois estados e outros dois autômatos com três estados. O PSS deste modelo é igual a $|\mathcal{X}| = 2 \times 3 \times 3 \times 2 = 36$ estados com seis eventos onde $l = \{t_1, t_2\}$ corresponde aos eventos locais e o conjunto $e = \{t_3, g_{12}, g_{23}, g_{34}\}$ enumera os eventos sincronizantes.



Tipo	Evento	Taxa	Tipo	Evento	Taxa
loc	t_1	μ_1	syn	g_{12}	λ_{12}
loc	t_2	μ_2	syn	g_{23}	λ_{23}
syn	t_3	μ_3	syn	g_{34}	λ_{34}

Figura 3.5: Modelo *Wireless Sensor Networks* com descritor markoviano constante para quatro nodos.

Logo, considerando-se os eventos locais, os eventos sincronizantes positivos e os ajustes diagonais dos eventos sincronizantes negativos, é necessário um termo com somas tensoriais (correspondendo aos eventos locais) e oito termos com produtos tensoriais (por sua vez equivalendo-se aos eventos sincronizantes). Estes termos estão descritos a seguir:

Parte local[1] correspondente aos autômatos $\mathcal{MN}^{(1)}$ e $\mathcal{MN}^{(2)}$:

$$\mathcal{Q}_l = \mathcal{Q}_l^{\mathcal{MN}^{(1)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(2)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(3)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 0 & 0 \\ \mu_1 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_2 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{t_3}$:

$$\mathcal{Q}_{t_{3+}} = \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_3 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{t_3}$:

$$\mathcal{Q}_{t_{3-}} = \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mu_3 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{g_{12}}$:

$$\mathcal{Q}_{g_{12+}} = \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 0 & \lambda_{12} \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{g_{12}}$:

$$\mathcal{Q}_{g_{12-}} = \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} -\lambda_{12} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{g_{23}}$:

$$\mathcal{Q}_{g_{23+}} = \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} \lambda_{23} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

[1] Cabe ressaltar que os autômatos $\mathcal{MN}^{(3)}$ e $\mathcal{MN}^{(4)}$ não possuem eventos locais, logo estes não tem relevância no cálculo do gerador infinitesimal final. Por este motivo as matrizes correspondentes a estes autômatos encontram-se zeradas em $\mathcal{Q}_{\downarrow}$.

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{g_{23}}$:

$$\mathcal{Q}_{g_{23-}} = \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} -\lambda_{23} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{g_{34}}$:

$$\mathcal{Q}_{g_{34+}} = \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} \lambda_{34} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{g_{34}}$:

$$\mathcal{Q}_{g_{34-}} = \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} -\lambda_{34} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

O gerador infinitesimal é dado por:

$$\tilde{\mathcal{Q}} = \mathcal{Q}_l + \left(\mathcal{Q}_{t_{3+}} + \mathcal{Q}_{t_{3-}}\right) + \left(\mathcal{Q}_{g_{12+}} + \mathcal{Q}_{g_{12-}}\right) + \left(\mathcal{Q}_{g_{23+}} + \mathcal{Q}_{g_{23-}}\right) + \left(\mathcal{Q}_{g_{34+}} + \mathcal{Q}_{g_{34-}}\right)$$

O gerador infinitesimal para esse problema corresponde a uma matriz quadrada de dimensão 36 e está fora do escopo desta seção mostrá-la por completo. Observando-se este descritor markoviano, notam-se algumas características deste modelo em especial. As suas matrizes são extremamente esparsas e as identidades marcam que um determinado evento *não* ocorre em um autômato. A seguir, descreve-se como calcular o vetor de probabilidade estacionária utilizando-se mecanismos sofisticados para multiplicação com descritores markovianos, como descrito na próxima seção.

3.4 Multiplicação Vetor-Descritor

Esta seção tratará das formas de multiplicar um vetor de probabilidade por um descritor. Este método é comumente denominado de Multiplicação Vetor-Descritor (MVD). Um dos ramos de pesquisa em SAN foca-se em formas de melhorar o desempenho deste tipo de multiplicação. Existem atualmente três algoritmos distintos para operar com o produto de um vetor por um descritor: Algoritmo *Esparso*, Algoritmo *Shuffle* [33] e um algoritmo que combina estas duas abordagens denominado *Split* [24]. Estas três maneiras serão mais detalhadas a seguir.

Definindo-se mais formalmente, a MVD pode ser vista como a multiplicação de um vetor de probabilidade v por $|\mathcal{L}| = N + 2E$ termos tensoriais compostos por N matrizes, onde $\mathcal{L}$ é um conjunto de termos tensoriais do tipo $\{l, e^+, e^-\}$, e dada por:

$$\sum_{j=1}^{|\mathcal{L}|} \left(v \times \left[\bigotimes_{i=1}^{N} \mathcal{Q}_j^{(i)} \right] \right) \tag{3.27}$$

Antes de explicar os algoritmos de MVD algumas notações a serem utilizadas são apresentadas.

Sejam:

n_k dimensão da matriz $\mathcal{Q}^{(k)}$;

nz_k total de elementos não-nulos da matriz $\mathcal{Q}^{(k)}$;

$\theta_{(1\ldots N)}$ o conjunto de todas as possíveis combinações de elementos não-nulos das matrizes de $\mathcal{Q}^{(1)}$ até $\mathcal{Q}^{(N)}$;

$\prod_{i=1}^{N} nz_i$ a cardinalidade de $\theta_{(1\ldots N)}$, e consequentemente o número de elementos não-nulos em $\mathcal{Q}$;

$nright_k$ tamanho do espaço de estados correspondente a todas as matrizes *depois* da matriz k-ésima do termo tensorial numericamente definida como $\prod_{i=k+1}^{N} n_i$, com o caso especial $nright_N = 1$;

$nleft_k$ tamanho do espaço de estados correspondente a todas as matrizes *antes* da matriz k-ésima do termo tensorial numericamente definida como $\prod_{i=1}^{k-1} n_i$, com o caso especial $nleft_1 = 1$;

$njump_k$ produto do espaço de estados depois da k-ésima matriz do termo tensorial pela dimensão n_k desta mesma matriz;

Υ, υ vetores de probabilidades;

3.4.1 Algoritmo Esparso

O Algoritmo *Esparso* é o mais intuitivo dos métodos de MVD [69]. A idéia principal do algoritmo é considerar o termo tensorial como uma única e potencialmente grande matriz esparsa a ser multiplicada pelo vetor de probabilidade. Este algoritmo assemelha-se ao método utilizado para a multiplicação de um vetor pelo gerador infinitesimal, a única diferença é que se considera apenas cada termo tensorial como uma grande matriz.

Para um termo tensorial composto por N matrizes $\mathcal{Q}^{(i)}$, cada uma de dimensão n_i com nz_i elementos não-nulos, o Algoritmo *Esparso* gerará todos os elementos em uma matriz $\mathcal{Q}$ resultando em $\mathcal{Q} = \bigotimes_{g\ i=1}^{N} \mathcal{Q}^{(i)}$, com dimensão $\prod_{i=1}^{N} n_i$.

Algorithm 3.1 Algoritmo Esparso - $\Upsilon = \upsilon \times \otimes_{g\ i=1}^{N} \mathcal{Q}^{(i)}$

1: $\Upsilon = 0$
2: **for all** $i_1, \ldots, i_N, j_1, \ldots, j_N \in \theta(1 \ldots N)$ **do**
3: $s = 1$
4: $base_{in} = base_{out} = 0$
5: **for all** $k = 1, 2, \ldots, N$ **do**
6: **evaluate** $Q^{(k)}$
7: $s = s \times q^{(k)}_{(i_k, j_k)}$ {cálculo de um escalar}
8: $base_{in} = base_{in} + ((i_k - 1) \times nright_k)$
9: $base_{out} = base_{out} + ((j_k - 1) \times nright_k)$
10: **end for**
11: $\Upsilon[base_{out}] = \Upsilon[base_{out}] + \upsilon[base_{in}] \times s$
12: **end for**

Cabe ressaltar que a linha 6 mostra que antes de realizar as multiplicações, dependendo do evento e das funções associadas no modelo, serão necessárias avaliações de função através da chamada **evaluate**. Uma possível

implementação do Algoritmo *Esparso* é apresentada no Algoritmo 3.1. O número de multiplicações em ponto flutuante para essa implementação é dada pela Equação 3.28:

$$N \times \prod_{i=1}^{N} nz_i \tag{3.28}$$

Entretanto, nesta versão, todos os elementos não-nulos de $\mathcal{Q}$ são, descrevendo-se o problema computacionalmente, gerados em tempo de execução do algoritmo. Tal geração representa $(N-1)\times\prod_{i=1}^{N} nz_i$ multiplicações que poderiam ser evitadas se uma matriz esparsa (usualmente grande) fosse criada para guardar estes $\prod_{i=1}^{N} nz_i$ elementos não-nulos. Esse procedimento eliminaria as linhas 3 e 6 do algoritmo e reduziria o número de multiplicações em ponto flutuante como definido pela Equação 3.29:

$$\prod_{i=1}^{N} nz_i \tag{3.29}$$

Esta opção, ao utilizar mais memória, reduz o número de cálculo de índices (que são normalmente efetuadas em outros algoritmos, por exemplo, no *Shuffle*, visto a seguir) e permite que o Algoritmo *Esparso* seja bastante eficiente em termos de tempo gasto de execução. Entretanto, a memória gasta para salvar essa estrutura de dados torna alguns modelos não tratáveis, logo, essa opção é normalmente descartada para modelos que contém um espaço produto de estados alto, como é o caso para uma grande variedade de casos reais. O total de chamadas à diretiva de avaliação de funções varia de modelo para modelo e não entra na equação de complexidade.

3.4.2 Algoritmo Shuffle

A seguir, explica-se o Algoritmo *Shuffle*[25], que é extremamente eficiente em termos de memória gasta, e nunca gera explicitamente a matriz $\tilde{\mathcal{Q}}$ ou mesmo uma parte dela, apenas as informações contidas nas matrizes de cada termo tensorial. A seguir, aplica eficazmente operações de álgebra tensorial para o cálculo das informações necessárias. O princípio básico deste algoritmo é a aplicação da decomposição de um termo tensorial na propriedade do produto ordinário de fatores normais [33]:

$$\begin{aligned}
Q^{(1)} \underset{g}{\otimes} Q^{(2)} \underset{g}{\otimes} \ldots \underset{g}{\otimes} Q^{(N-1)} \underset{g}{\otimes} Q^{(N)} = \quad & (\; Q^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \ldots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} I_{n_N} \;) \times \\
& (\; I_{n_1} \underset{g}{\otimes} Q^{(2)} \underset{g}{\otimes} \ldots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} I_{n_N} \;) \times \\
& \qquad \ldots \\
& (\; I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \ldots \underset{g}{\otimes} Q^{(N-1)} \underset{g}{\otimes} I_{n_N} \;) \times \\
& (\; I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \ldots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} Q^{(N)} \;)
\end{aligned}$$

O Algoritmo *Shuffle* consiste em multiplicar sucessivamente o vetor de probabilidade por cada fator normal. Mais precisamente, o vetor v é multiplicado pelo primeiro fator normal e o vetor resultante é multiplicado pelo próximo e assim por diante, até o último.

Internamente, para cada fator normal, as multiplicações são feitas usando-se pequenos vetores chamados z_{in} e z_{out} como descrito no Algoritmo 3.2. Estes vetores pequenos em termos de tamanho guardam os valores de v para multiplicação pela $i^{\text{ésima}}$ matriz do fator normal z_{in} e guardar o resultado em z_{out}.

A linha 5 do algoritmo corresponde a uma avaliação de função (caso seja necessária para o modelo e de acordo com a definição da função) através da diretiva (**evaluate**). A multiplicação pelo vetor v pelo $i^{\text{ésima}}$ fator normal

Algorithm 3.2 Algoritmo Shuffle - $\Upsilon = \upsilon \times \otimes_{g\,i=1}^{N} Q^{(i)}$

for all $i = 1, 2, \ldots, N$ **do**
  $base = 0$
  for all $m = 0, 1, 2, \ldots, nleft_i - 1$ **do**
    for all $j = 0, 1, 2, \ldots, nright_i - 1$ **do**
      evaluate $Q^{(i)}(a^{(1)}_{m_1}, \ldots, a^{(i-1)}_{m_{i-1}})$
      $index = base + j$
      for all $l = 0, 1, 2, \ldots, n_i - 1$ **do**
        $z_{in}[l] = v[index]$
        $index = index + nright_i$
      end for
      $multiply\ \ z_{out} = z_{in} \times Q^{(i)}$
      $index = base + j$
      for all $l = 0, 1, 2, \ldots, n_i - 1$ **do**
        $v[index] = z_{out}[l]$
        $index = index + nright_i$
      end for
    end for
    $base = base + (\, nright_i \times n_i)$
  end for
end for
$\Upsilon = \upsilon$

consiste, generalizadamente, em *embaralhar*[2] os elementos de υ para montar $nleft_i \times nright_i$ vetores de tamanho n_i e multiplicá-los pela matriz $\mathcal{Q}^{(i)}$. Assumindo-se que a matriz $\mathcal{Q}^{(i)}$ é guardada de forma esparsa, o número de operações necessárias para multiplicar um vetor pelo $i^{\text{ésimo}}$ fator normal é: $nleft_i \times nright_i \times nz_i$, onde nz_i corresponde ao número de elementos não-nulos da $i^{\text{ésima}}$ matriz do termo tensorial $\mathcal{Q}^{(i)}$.

Considerando-se o número de multiplicações por todos os fatores normais de um termo tensorial, o custo computacional do Algoritmo *Shuffle* para efetuar a operação de multiplicação de um vetor por um descritor é dada pela Equação 3.30 [33]:

$$\sum_{i=1}^{N} nleft_i \times nright_i \times nz_i = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \tag{3.30}$$

Uma outra vantagem de uso do Algoritmo *Shuffle* consiste no fato de possuir otimizações para modelos com taxas funcionais, ou seja, modelos que utilizam propriedades da Álgebra Tensorial Generalizada e também no que diz respeito ao uso de reordenamentos da estrutura tensorial (também chamadas de permutações). Como no Algoritmo *Esparso* as chamadas para avaliações de funções variam em cada modelo e não estão sendo contabilizadas no cálculo da complexidade final.

3.4.3 Algoritmo Split

Eventos locais em SAN são normalmente esparsos, indicando um comportamento que independe de outros autômatos de um sistema. O mesmo não ocorre com eventos sincronizantes, pois podem potencialmente afetar todos os autômatos de um modelo. Entretanto, dependendo do modelo que está sendo analisado, essa condição

[2] Do inglês, *shuffling*

Split σ	Termo Tensorial $\mathcal{T}$
0	$\overset{\sigma}{\downarrow}$ $\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{shuffle}$
1	$\underbrace{\overset{\sigma}{\downarrow} Q^{(1)}}_{sparse} \otimes \underbrace{Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{shuffle}$
2	$\underbrace{Q^{(1)} \otimes \overset{\sigma}{\downarrow} Q^{(2)}}_{sparse} \otimes \underbrace{\ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{shuffle}$
$\vdots$	$\vdots$
N-2	$\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes \overset{\sigma}{\downarrow} Q^{(N-2)}}_{sparse} \otimes \underbrace{Q^{(N-1)} \otimes Q^{(N)}}_{shuffle}$
N-1	$\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes \overset{\sigma}{\downarrow} Q^{(N-1)}}_{sparse} \otimes \underbrace{Q^{(N)}}_{shuffle}$
N	$\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes \overset{\sigma}{\downarrow} Q^{(N)}}_{sparse}$

Tabela 3.1: O Algoritmo Split apresentado como uma generalização de algoritmos tradicionais.

é inexistente, *i.e.*, um número substancial de ocorrências dizem respeito a entre pelo menos dois autômatos e um número que dificilmente beira o total de autômatos. Esse é um dos motivos pelos quais iniciou-se a pesquisa de algoritmos que resolvessem a MVD de forma híbrida e que tirassem vantagem da propriedade da *Decomposição Aditiva* combinada à decomposição de produtos tensoriais clássicos em fatores normais.

A propriedade da decomposição aditiva dita que qualquer produto tensorial pode ser decomposto em uma soma ordinária de matrizes compostas por um único elemento não-nulo. Seja $\hat{q}_{(i_1,\ldots,i_{N-1},j_1,\ldots,j_N)}$ a matriz de dimensão $\prod_{i=1}^{N} n_i$ composta por apenas um elemento não-nulo o qual encontra-se na posição $i_1, \ldots, i_N, j_1, \ldots, j_N$ e igual a $\prod_{k=1}^{N} q_{i_k,j_k}^{(k)}$, a propriedade da decomposição aditiva pode ser definida como:

$$Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-1)} \otimes Q^{(N)} = \begin{array}{l} \sum_{i_1=1}^{n_1} \cdots \sum_{i_N=1}^{n_N} \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \\ \left(\hat{q}_{(i_1,j_1)}^{(1)} \otimes \ldots \otimes \hat{q}_{(i_N,j_N)}^{(N)} \right) \end{array} \tag{3.31}$$

onde $\hat{q}_{(i,j)}^{(k)}$ é uma matriz de dimensão n_k a qual o elemento na linha i e coluna j é $q_{i,j}^{(k)}$.

O Algoritmo *Split* propõe uma solução combinada usando fundamentalmente a propriedade da decomposição aditiva para um dado conjunto de matrizes em um produto tensorial seguido da aplicação da operação de *embaralhamento* para as matrizes restantes do termo. Cada fator é chamado de Fator Normal Aditivo Unitário (*Additive Unitary Normal Factor - AUNF*). Esse conceito é central para o Algoritmo *Split* e internamente trata-se de uma estrutura de dados que armazena os índices dos vetores de entrada ($base_{in}$) e saída ($base_{out}$), um escalar s e um vetor de tamanho σ contendo os índices de linha e coluna de cada matriz que foram usados para calcular s (esse vetor será usado para avaliação de funções com parâmetros que estejam na parte esparsa posteriormente).

Agora tem-se então uma divisão do termo tensorial em dois grupos distintos: o primeiro contendo matrizes que serão unidas sob a forma de uma lista de fatores normais aditivos unitários e o segundo grupo contendo o restante das matrizes. Em termos de solução, o primeiro grupo lidará com uma solução esparsa, multiplicando tensorialmente cada AUNF pelo segundo grupo usando a solução *Shuffle*. A idéia principal é dividir o termo

tensorial em dois conjuntos e trata-los de forma distinta em termos de multiplicação vetor-descritor.

A Tabela 3.1 apresenta estes conceitos do Algoritmo *Split* de forma gráfica. O índice da matriz escolhida para delimitar o fim do tratamento com a parte esparsa (e subsequentemente, o início da parte *Shuffle* é denominado *parâmetro de corte* σ (ou simplesmente *corte* σ). Casos especiais do Algoritmo *Split* são os casos onde o corte possui valores extremos, *i.e.*, quando $\sigma = N$ significa que a abordagem Esparsa pura está sendo utilizada e quando $\sigma = 0$ não existem matrizes no lado esparso do termo tensorial e apenas o Algoritmo *Shuffle* será executado. Estes são casos particulares do Algoritmo *Split*.

Uma questão pertinente é onde realizar o corte de maneira que o tempo de solução do modelo alcance bom desempenho. Esse é o principal objetivo deste trabalho, juntamente com o desafio de como saber esse valor de corte ao mesmo tempo que é permitido alterar a ordem lexicográfica dos autômatos dentro de um termo tensorial (operando-se com permutações ou reordenamentos) para facilitar, e por vezes otimizar, a MVD. Esses problemas também relacionam-se com o uso ou não de Álgebra Tensorial Generalizada, ou seja, qual o melhor ponto de corte dado que algumas matrizes do termo tensorial possuem taxas funcionais e suas implicações. Essas questões serão melhores debatidas nas próximas seções.

Parte-se a seguir para a definição formal da operação do método em si, de acordo com o Algoritmo 3.3. A ideia principal do método consiste em computar o elemento escalar s de cada *AUNF* em $\theta(1 \dots \sigma)$ multiplicando cada elemento não-nulo s de cada matriz do primeiro conjunto de matrizes (parte esparsa) de $\mathcal{Q}^{(1)}$ até $\mathcal{Q}^{(\sigma)}$ (linhas 5 a 9). De acordo com os elementos de linha que foram utilizados para gerar s, uma fatia contígua do vetor de entrada υ, chamado υ_{in}, e será utilizado em uma estrutura de dados. O vetor υ_{in} de tamanho $nright_{\sigma}$ (correspondente ao produto das dimensões das matrizes após o *parâmetro de corte* σ do termo tensorial) é multiplicado pelo escalar s. Nas linhas 10 e 11 são realizadas as multiplicações de e para cada posição em υ, finalizando a parte esparsa do algoritmo. O vetor resultante υ_{in} também é utilizado como vetor de entrada para a parte Shuffle (linhas 13 a 32) pelo produto tensorial das matrizes do segundo conjunto (de $\mathcal{Q}^{(\sigma+1)}$ até $\mathcal{Q}^{(N)}$). Ao término da parte *Shuffle* o vetor υ obtido é acumulado no vetor final Υ (linhas 33 a 35).

O algoritmo possui três blocos distintos, um bloco onde para cada AUNF calculado (de zero a σ), um vetor auxiliar de tamanho $nright_{\sigma}$ (*i.e.*, $nright$ de zero até o valor do corte) é multiplicado em posições chave originadas durante o cálculo do AUNF em questão. Um segundo bloco, onde o método *Shuffle* é chamado para um sub-espaço à esquerda que desconta as matrizes da parte esparsa (linha 15) e finalmente, uma seção onde este vetor auxiliar que foi modificado no bloco anterior é acumulado no vetor final Υ.

A linha 17 do algoritmo mostra uma diretiva chamada **evaluate** que é utilizada para a avaliação de elementos funcionais. Cabe ressaltar que essa é a primeira vez que o Algoritmo *Split* trata descritores generalizados, ou seja, modelos com taxas funcionais. Essa é basicamente a diferença fundamental entre as alternativas existentes até o momento de MVD que utilizem soluções híbridas de armazenamento e os impactos desta diretiva serão melhores analisados nos Capítulos 4 e 5 referentes as estratégias de modificação dos descritores e aos resultados obtidos respectivamente.

O custo computacional em termos de multiplicações (de acordo com a Equação 3.32) do Algoritmo *Split* é realizado levando-se em conta o número de vezes necessárias para gerar cada elemento não-nulo que resultou em um AUNF $(\sigma - 1)$, mais o número de multiplicações de cada valor escalar por cada posição do vetor υ_{in}. Finalmente, ainda adiciona-se o custo de multiplicar os valores do vetor de entrada υ_{in} pelo produto tensorial das matrizes da parte *Shuffle*.

$$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1)+\left(\prod_{i=\sigma+1}^{N} n_i\right)+\left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{i=\sigma+1}^{N} \frac{nz_i}{n_i}\right)\right] \tag{3.32}$$

Existem algumas otimizações que podem ser implementadas em algoritmos para MVD. Estas modificações em termos de implementação alteram significativamente o custo computacional teórico apresentado na Equação 3.32. Para o caso do Algoritmo *Shuffle*, pode-se otimizar a maneira pela qual o método lida com matrizes do tipo identi-

Algorithm 3.3 Algoritmo Split - $\Upsilon = \upsilon \times \otimes_{g\,i=1}^{N} \mathcal{Q}^{(i)}$

1: $\Upsilon = 0$
2: **for all** $i_1, \ldots, i_\sigma, j_1, \ldots, j_\sigma \in \theta(1 \ldots \sigma)$ **do**
3: $s = 1$
4: $base_{in} = base_{out} = 0$
5: **for all** $k = 1, 2, \ldots, \sigma$ **do**
6: $s = s \times q^{(k)}_{(i_k, j_k)}$ {cálculo do escalar}
7: $base_{in} = base_{in} + ((i_k - 1) \times nright_k)$
8: $base_{out} = base_{out} + ((j_k - 1) \times nright_k)$
9: **end for**
10: **for all** $l = 0, 1, 2, \ldots, nright_\sigma - 1$ **do**
11: $\upsilon_{in}[l] = \upsilon[base_{in} + l] \times s$
12: **end for**
13: **for all** $i = \sigma + 1, \ldots, N$ **do**
14: $base = 0$
15: **for all** $m = 0, 1, 2, \ldots, \frac{nleft_i}{nleft_\sigma} - 1$ **do**
16: **for all** $j = 0, 1, 2, \ldots, nright_i$ **do**
17: **evaluate** $Q^{(i)}(a^{(1)}_{m_1}, \ldots, a^{(i-1)}_{m_{i-1}})$
18: $index = base + j$
19: **for all** $l = 0, 1, 2, \ldots, n_i - 1$ **do**
20: $z_{in}[l] = \upsilon_{in}[index]$
21: $index = index + nright_i$
22: **end for**
23: $multiply \;\; z_{out} = z_{in} \times \mathcal{Q}^{(i)}$
24: $index = base + j$
25: **for all** $l = 0, 1, 2, \ldots, n_i - 1$ **do**
26: $\upsilon_{in}[index] = z_{out}[l]$
27: $index = index + nright_i$
28: **end for**
29: **end for**
30: $base = base + (nright_i \times n_i)$
31: **end for**
32: **end for**
33: **for all** $l = 0, 1, 2, \ldots, nright_\sigma - 1$ **do**
34: $\Upsilon[base_{out} + l] = \Upsilon[base_{out} + l] + \upsilon_{in}[l]$
35: **end for**
36: **end for**

dade. Estas matrizes fazem com que seja desnecessária gerar fatores normais para o cálculo do vetor solução, pois, sendo identidades, as restantes também são identidades, logo todo o termo tensorial é uma identidade. É evidente que, em se tratando deste caso particular, nenhum fator normal seja criado, reduzindo o custo computacional de execução. Esse custo corresponde a transformar a Equação 3.30 na Equação 3.33:

$$\prod_{i=1}^{N} n_i \times \sum_{\substack{i=1 \\ \text{iff} Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i} \tag{3.33}$$

O ideal é gerar sempre fatores normais com identidades (preferencialmente com o uso de permutações do termo tensorial) e utilizar o método esparso para calcular apenas os AUNFs dos termos tensoriais que são pertinentes. Entretanto, dependendo do evento, o número de AUNFs gerados pode onerar a memória a ser gasta para armazenamento dos escalares necessários. Nesse caso, uma análise minuciosa de cada termo tensorial deve ser realizada anteriormente onde detectará casos onde exista um grande subespaço composto por identidades e onde o método esparso obterá um ganho significativo de desempenho frente as demais modalidades de MVD existentes.

Esta melhoria sugere que o mesmo pode ser aplicado na parte do método *Shuffle* no Algorimo 3.3 referente ao Algoritmo *Split* nas matrizes $\mathcal{Q}^{(\sigma+1)}$ a $\mathcal{Q}^{(N)}$. Caso a matriz indexada por i no algoritmo ($\mathcal{Q}^{(i)}$) não seja uma matriz identidade, o custo de $\frac{nz_i}{n_i}$ multiplicações é adicionado.

Analogamente ao Algoritmo *Shuffle*, a Equação 3.32 pode ser reescrita alterando-se o custo para o cálculo a partir de σ. O número de multiplicações resultantes para o Algoritmo *Split* seguirá a Equação 3.34:

$$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1)+\left(\prod_{i=\sigma+1}^{N} n_i\right)+\left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{\substack{i=\sigma+1 \\ iff Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}\right)\right] \tag{3.34}$$

Usualmente, em SAN, os termos tensoriais são esparsos, uma vez que indicam a ocorrência dos eventos em cada autômato (exceto em casos onde existam diversos eventos ocorrendo em diversos autômatos, indicando matrizes quase plenas no termo).

Uma outra otimização que pode ser realizada é quanto ao pré-cálculo dos elementos não-nulos, salvando-os em estruturas de dados que serão acessadas em todas as iterações, mas computados apenas uma vez. Estas otimizações foram amplamente estudadas por [57] e causam uma redução significativa no custo computacional do Algoritmo *Split*, similar ao apresentado na Seção 3.4.1 (Equação 3.28) no que diz respeito ao cálculo dos elementos não-nulos. Logo, o valor final da definição dos número de multiplicações em ponto flutuante para o algoritmo em questão não é mais o definido pela Equação 3.34, mas sim pela Equação 3.35:

$$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[\left(\prod_{i=\sigma+1}^{N} n_i\right)+\left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{\substack{i=\sigma+1 \\ iff Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}\right)\right] \tag{3.35}$$

3.5 Permutações dos termos tensoriais

Uma das operações mais custosas efetuadas na MVD é a avaliação de funções. No contexto deste trabalho, estas avaliações estão presentes nos algoritmos que foram explicados nas Seções 3.4.1, 3.4.2 e 3.4.3 referentes respectivamente aos Algoritmos *Esparso*, *Shuffle* e *Split*. Estas chamadas de avaliações estão presentes dentro dos laços $nright$ e $nleft$, ou seja, são executadas diversas vezes ao longo do método, dependendo do tamanho destes subespaços.

Para diminuir os efeitos destas execuções, são apresentadas técnicas de reordenamento ou permutação dos termos tensoriais. Estas técnicas servem para reorganizar as matrizes dos termos tensoriais detectando uma forma para avaliar menos funções, colocando a diretiva de avaliação dentro de apenas um laço e não em dois laços como é feito originalmente no Algoritmo *Shuffle* [7]. O Algoritmo *Split* em sua versão original não realiza permutações. A inclusão de solução para tensores em ATG é uma contribuição que foi introduzida neste trabalho bem como o estudo e implementação adicional para contemplar tal primitiva do formalismo. Como será melhor explicado nos próximos capítulos, o uso de permutações no Algoritmo *Split* auxiliará na solução otimizada de modelos e para tanto deve ser melhor compreendida e explicada.

Ao observar os casos particulares de produtos tensoriais generalizados, pode-se compreender melhor o interesse de realizar estas otimizações [7]. Por exemplo, o produto tensorial generalizado dado por:

$$\bigotimes_{i=1}^{N}{}_{g} \mathcal{Q}^{(i)}(\mathcal{A}^{(i+1)}, \ldots, \mathcal{A}^{(N)})$$

Os fatores normais da decomposição devem ser tratados de acordo com a seguinte ordem:

$$\begin{array}{cccccc} & I_{nleft_1} & \underset{g}{\otimes} & \mathcal{Q}^{(1)}(\mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(N)}) & \underset{g}{\otimes} & I_{nright_1} \\ \times & I_{nleft_2} & \underset{g}{\otimes} & \mathcal{Q}^{(2)}(\mathcal{A}^{(3)}, \ldots, \mathcal{A}^{(N)}) & \underset{g}{\otimes} & I_{nright_2} \\ \times & \ldots & & \ldots & & \ldots \\ \times & I_{nleft_N} & \underset{g}{\otimes} & \mathcal{Q}^{(N)} & \underset{g}{\otimes} & I_{nright_N} \end{array}$$

Observa-se que cada matriz $\mathcal{Q}^{(i)}(\ldots)$ só depende dos autômatos à sua direita. De maneira análoga, para o seguinte produto tensorial generalizado:

$$\bigotimes_{i=1}^{N}{}_{g} \mathcal{Q}^{(i)}(\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(i-1)})$$

Os fatores normais da decomposição devem ser tratados de acordo com a seguinte ordem:

$$\begin{array}{cccccc} & I_{nleft_N} & \underset{g}{\otimes} & \mathcal{Q}^{(N)}(\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N-1)}) & \underset{g}{\otimes} & I_{nright_N} \\ \times & I_{nleft_N-1} & \underset{g}{\otimes} & \mathcal{Q}^{(N-1)}(\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N-2)}) & \underset{g}{\otimes} & I_{nright_{N-1}} \\ \times & \ldots & & \ldots & & \ldots \\ \times & I_{nleft_1} & \underset{g}{\otimes} & \mathcal{Q}^{(1)} & \underset{g}{\otimes} & I_{nright_1} \end{array}$$

Para esse caso, cada matriz $\mathcal{Q}^{(i)}(\ldots)$ só depende dos autômatos à sua esquerda. Devido a esse fato, como se têm certeza do posicionamento dos autômatos, é possível modificar os algoritmos de MVD para realizarem menos chamadas à diretivas de avaliação de funções. No caso do Algoritmo 3.2, a linha 5 seria movida para dentro do laço de $nright$ (ou seja, logo abaixo da linha 3) e no caso do Algoritmo 3.3, a linha 17 seria colocada logo abaixo da linha 15, pelo mesmo motivo.

Ainda encontra-se em aberto como realizar esse reordenamento de autômatos para se encontrar os casos onde esse procedimento é passível de ser realizado para tentar-se reduzir os números de avaliações de elementos funcionais. Para isso, informalmente, são poucos os casos onde as funções que são definidas dependem do estado de *todos* os outros autômatos. Normalmente, uma função depende de apenas alguns outros autômatos. Este seria o pior caso para a permutação, inclusive seria completamente desnecessário tentar realizar reordenamentos, pois não eliminariam o número de avaliações possíveis.

$\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}$				$\mathcal{A}^{(3)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)}$			
rank	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	*rank*	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$
0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0
2	0	1	0	2	0	2	0
3	0	1	1	3	1	0	0
4	0	2	0	4	1	1	0
5	0	2	1	5	1	2	0
6	1	0	0	6	0	0	1
7	1	0	1	7	0	1	1
8	1	1	0	8	0	2	1
9	1	1	1	9	1	0	1
10	1	2	0	10	1	1	1
11	1	2	1	11	1	2	1

Tabela 3.2: Reordenamentos da estrutura original mostrando os índices de cada autômato e seus respectivos *ranks*.

Não faz parte do escopo deste trabalho explicar detalhadamente as variadas técnicas de reordenamento de termos tensoriais, pois estas já foram exaustivamente feitas em [32, 33, 7]. Este trabalho parte do princípio que a realização de reordenamentos pode ser extremamente eficaz tanto para reduzir-se o número de avaliações como utilizar essas reordens para reduzir o tempo necessário para executar métodos híbridos de solução, neste caso, especificamente para o caso do Algoritmo *Split*.

A seguir, serão apresentados os princípios da permutação em termos tensoriais, seus objetivos e algumas definições de base. Dado que os autômatos podem trocar de posição no termo tensorial, os elementos do vetor de probabilidade final devem trocar de lugar pois a ordem lexicográfica original foi alterada. A Tabela 3.2 mostra os estados de um vetor pelos autômatos ordenados de duas maneiras distintas. Os elementos do vetor são organizados de acordo com uma ordem lexicográfica expressa pela lista de autômatos, igualmente respeitando uma ordem.

À esquerda da tabela é utilizada a ordem $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}$. Os autômatos deste exemplo possuem tamanho $n_1 = 2$, $n_2 = 3$ et $n_3 = 2$. À direita da tabela é realizada a seguinte reordem dos autômatos: $\mathcal{A}^{(3)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)}$. A coluna *rank* equivale ao índice no vetor de probabilidade. Como para esse exemplo o $|\mathcal{X}| = 2 \times 3 \times 2 = 12$, esta coluna varia de 0 a 11.

A ideia aqui é descobrir o próximo estado dado um estado corrente. Para simplificar as operações envolvidas, a melhor forma de se fazer esta descoberta é pensar em um algoritmo que retorne o próximo estado, dada uma ordem. Logo, observa-se os autômatos do último até o primeiro e tenta-se incrementar seu estado local. Um incremento é inválido quando o estado local do autômato é o último estado (não existem mais estados). Quando o incremento é inválido , o estado local do autômato é zerado (*reset* de estado) e se considera o próximo estado (de acordo com uma determinada ordem, podendo não ser a original e sim qualquer ordem). Este procedimento acaba quando um incremento válido foi realizado.

Para exemplificar esse processo, considere-se a ordem $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}$ onde deseja-se saber o próximo estado (um incremento de estado) dado que o estado atual é o $(0, 1, 1)$. O autômato $\mathcal{A}^{(3)}$ é o primeiro a ser considerado. Tenta-se incrementá-lo, mas, por estar já no último estado possível (1), coloca-se nessa posição, o valor 0. A seguir, tenta-se incrementar $\mathcal{A}^{(2)}$, e o incremento é valido, para 2. Logo, o próximo estado global do sistema seguido do estado $(0, 1, 1)$ é o $(0, 2, 0)$ (observe essa ocorrência na Tabela 3.2, entre os *ranks* 3 e 4 do lado esquerdo).

Entretanto, para o caso de uma permutação das posições no termo tensorial, os incrementos não são mais tão simples e devem seguir um cálculo mais elaborado do *rank* final para descoberta do próximo estado. É necessário

se calcular o novo *rank* a partir do *rank* corrente do vetor permutado. Seguindo-se o mesmo exemplo, o *rank* do estado $(0, 1, 1)$ pela nova ordem lexicográfica dada por $\mathcal{A}^{(3)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)}$ é 7. O *rank* após o incremento que corresponde ao estado $(0, 2, 0)$ deve ser 2.

Para calcular esse novo índice, realizam-se incrementos e decrementos, zerando alguns estados locais. Os valores destes incrementos e decrementos variam de acordo com a nova ordem lexicográfica. Um incremento válido para o próximo estado de um estado local do autômato $\mathcal{A}^{(i)}$ corresponde à adição de $nright_i$ no *rank*. No exemplo em questão foi feito um *reset* no estado do autômato $\mathcal{A}^{(3)}$, e sob a nova ordem lexicográfica, $nright_3 = 2 \times 3 = 6$. Com isso é obtido o estado de *rank* $7 - 6 = 1$. A seguir, como ocorrido no exemplo anterior, incrementa-se o estado do autômato $\mathcal{A}^{(2)}$, e este incremento é válido, ou seja, obtem-se o valor 2 para o *rank* final, como seria normalmente retornado mas, neste caso, com o uso de permutações.

Com isso, observa-se que são feitos os mesmos procedimentos para ambos os exemplos, apenas no caso onde houve a permutação das ordens originais é que ao invés de se incrementar foi descontado um valor que correspondia ao $nright_i$, equivalendo ao salto que deve ser feito na estrutura permutada. Trata-se de uma maneira transparente de se efetuar as permutações, incorrendo-se apenas no recálculo de índices para saltar na estrutura tensorial.

As permutações foram inicialmente utilizadas no Algoritmo *Shuffle* com o intuito principal de eliminar chamadas desnecessárias de avaliações de elementos funcionais. Com isso, para alguns modelos, observou-se um ganho de desempenho pois menos operações são efetuadas se uma permutação satisfatória for encontrada. Naturalmente, não é fácil descobrir como melhor permutar cada termo tensorial para obter o máximo de desempenho e esta é uma questão aberta de pesquisa.

No caso do Algoritmo *Split*, as permutações desempenharão um papel ainda mais importante que a verificada para o Algoritmo *Shuffle* e usada com êxito para alguns casos. No caso do *Split*, os reordenamentos serão cruciais para entender qual a melhor forma de dividir os termos tensoriais enviando matrizes ou para o lado estruturado ou enviando as avaliações para a parte esparsa e vice-versa. Somente com as permutações é que é possível verificar onde o método é melhor executado pois torna possível separar cada característica dos termos tensoriais e estudar como afetam a solução dos modelos.

Estas análises finalizam o capítulo que tratou sobre a solução de descritores markovianos. Foi revelado que o Algoritmo *Split* pois possui algumas restrições ao delimitar onde melhor dividir os termos tensoriais, seja em função das identidades existentes ou das dependências funcionais. Atualmente o algoritmo é executado algumas iterações para a coleta de tempos de execução dos termos para então decidir quais cortes aplicará para cada termo tensorial nas iterações seguintes. Esta escolha não segue um procedimento determinístico, baseado nas características das matrizes, trata-se de uma heurística baseada nas amostras das execuções. O próximo capítulo tratará das estratégias de reordenação do descritor para otimizar a solução de modelos complexos.

4 Estratégias de reestruturação

O objetivo deste capítulo é tratar sobre as estratégias para escolha de um ponto de corte σ de descritores markovianos com o objetivo de otimizar o tempo de solução do Algoritmo *Split*. As estratégias que serão estudadas servirão de base para a discussão dos resultados obtidos no Capítulo 5. A Seção 4.1 apresenta os estudos e definições preliminares necessárias para uma maior compreensão do problema da determinação de σ e as escolhas que podem ser feitas para tratar os descritores. A Seção 4.2 mostra uma análise teórica seguida das considerações finais na Seção 4.3 onde uma previsão dos custos computacionais em termos de multiplicações de ponto-flutuante é apresentada.

4.1 Estudos e definições preliminares

Como explicado anteriormente, o Algoritmo *Split* trata os termos tensoriais dos descritores markovianos de forma híbrida, basicamente de duas formas distintas. De um lado do termo acontece a aplicação da solução esparsa e referente à outra parte, a multiplicação que contém os fatores normais restantes (à direita do ponto de corte σ) e aplica o Algoritmo *Shuffle*. Neste trabalhos esta separação dos termos será definida como *parte esparsa* e *parte estruturada*, respectivamente.

Devido as características dos termos tensoriais, o problema está na determinação do ponto de corte ou divisão dos termos tensoriais σ tal que o tempo para execução e a memória a ser gasta seja eficiente e balanceada. Dentre as principais características dos termos existentes, para o escopo deste trabalhos, o interesse está voltado para as matrizes identidades que compõem os termos, as avaliações de elementos funcionais existentes e na quantidade de elementos não nulos de cada matriz. Intrinsicamente relacionado à quantidade de elementos não-nulos estão as sincronizações existentes pois, como exemplificado nas Seções 3.3.1 e 3.3.2, estas cooperações indicam diretamente a quantidade de identidades existentes e a memória a ser potencialmente gasta.

O ponto de corte σ escolhido ditará a memória que será gasta na parte esparsa e deseja-se evitar que um dado termo utilize uma quantidade massiva a ponto de inviabilizar a execução do Algoritmo *Split*. Deve-se identificar cada termo tensorial e verificar a melhor forma de tratá-lo. Cada termo tensorial será dividido de uma forma única (diferentes valores de σ para diferentes termos), aplicando o melhor custo benefício em termos de memória e tempo para executar o método de MVD.

Esta seção utiliza os conceitos explicados anteriormente no Capítulo 3, especificamente as definições referentes à Álgebra Tensorial Clássica e Generalizada descritas na Seções 3.1 e 3.2, descrição dos métodos de MVD da Seção 3.4 e o conceito de AUNF que é descrito na Seção 3.4.3.

Como cada termo tensorial corresponde à ocorrência de apenas um evento, é necessário estudar quantas matrizes este afeta. Precisa-se descobrir o *grau de dependência* existente em cada termo para facilitar a análise das estratégias. Define-se *Grau de Dependência Constante* (GDC) e *Grau de Dependência Generalizado* (GDG) os graus relacionados, respectivamente, aos termos constantes e generalizados. O grau de dependência de um termo tensorial informa basicamente o total de autômatos que são parâmetros para um determinado elemento funcional, ou seja, informa o número de elementos necessários para a aplicação da função. Mais especificamente, tem-se a seguinte classificação para os termos tensoriais de uma SAN:

- Termo constante: um termo constante é um termo que não possui funções, apenas matrizes constantes que indicam onde cada evento ocorre em cada autômato.
  - *GDC Alto*: um termo deste tipo envolve um valor superior à metade (seja t o total de matrizes do termo, a sua metade m corresponde a $m = \frac{t}{2}$) das matrizes que o compõe[1]. Ao envolver metade dos autômatos, implica na existência da outra metade ser composta por matrizes do tipo identidade;
  - *GDC Parcial*: este evento sincroniza suas atividades com menos metade das matrizes existentes no termo;

- Termo generalizado: um termo generalizado possui matrizes com elementos funcionais que devem ser avaliados. Termos deste tipo são classificados quanto ao grau de dependência generalizado, podendo ser:
  - *GDG Alto*: envolvem todas as matrizes do termo tensorial. Trata-se basicamente de uma função que consulta os estados do resto dos autômatos para tomar sua decisão;
  - *GDG Parcial*: envolvem apenas algumas matrizes do termo tensorial. Como só existem essas duas classificações, se um termo não é classificado como possuidor de um GDG Alto, ele é automaticamente definido como GDG Parcial;

Termos tensoriais são compostos por matrizes de diferentes características. As matrizes de um termo podem ser classificadas da seguinte forma (seja n_z o número de elementos não-nulos e n a dimensão de uma matriz):

- Identidade: a matriz é do tipo identidade, possuindo o valor 1 para a diagonal principal e o valor 0 para o restantes das posições.
- Constante: trata-se de uma matriz com mais de um elemento não-nulo, com esparsidade variável (entre $\frac{2}{n}$ até $\frac{n_z}{n}$).
- Elemento: trata-se de um caso especial de matriz constante, ou seja, é uma matriz esparsa com apenas um elemento não-nulo (esparsidade $\frac{1}{n}$).
- Funcional: indica que a matriz possui uma função definida e necessita de informações de outras matrizes do termo (para o escopo deste trabalho, de estados de outros autômatos) para correta avaliação.

Diferentes estratégias de corte devem ser tomadas para os diferentes tipos de termos tensoriais. Para os termos constantes, apenas as dimensões das matrizes, a esparsidade e a ocorrência de matrizes identidade é que importa. Já para os termos generalizados, além destas características, importam também as matrizes que a função necessita para ser avaliada além de onde avaliar essa função, ou na *parte esparsa* ou na *parte estruturada*.

Existem alguns fatores que contribuem para uma degradação do desempenho da MVD tais como as chamadas a avaliações de funções. Os custos computacionais envolvidos variam de definição para definição, mas impactam no aumento do número de operações que são efetuadas para retornar avaliações válidas. A intuição é que como o método iterativo será chamado diversas vezes, para o mesmo conjunto de parâmetros de função, talvez fosse mais interessante apenas avaliá-las uma vez e salvar as avaliações obtidas em uma estrutura de dados auxiliar. Entretanto, essa operação não é clara e deve ser estudada com uma maior profundidade. É claro, no entanto, que diretivas funcionais aumentaram as maneiras pelas quais modelam-se sistemas complexos onde, muitas vezes, as formas de transição são mais complexas do que basear-se na utilização pura e simples de eventos locais e sincronizantes para o caso de SAN.

As avaliações na parte esparsa converterm um descritor markoviano generalizado em constante. Intuitivamente, avaliar as funções na parte esparsa remove a sobrecarga de se ter que avaliá-la inúmeras vezes em potencialmente inúmeras iterações. Entretanto, uma análise teórica faz-se necessária para substanciar essas evidências.

[1]Definiu-se dividir pela metade o termo tensorial apenas por questões de classificação, correspondendo a um valor razoável, ou seja, uma heurística, dada a lista de matrizes dos termos tensoriais.

Para começar esta análise teórica mais aprofundada, é necessário definir precisamente as opções existentes ao dividir termos tensoriais. A seguir é feita uma análise sobre como proceder através do estudo de opções para os casos constante e generalizado. Seja M o total de matrizes de um termo tensorial e seja V o número de matrizes envolvidas na sincronização dado que, por definição, $V = 1$ não acontece, pois não é permitida a definição de um evento sincronizante envolvendo apenas um autômato:

- Para termos constantes, reordenar o termo e reconfigurar o termo privilegiando o fato de que as identidades sejam movidas para o lado estruturado:
  - pior caso: $V = M$, ou seja, não existem identidades, *i.e.*, o evento sincronizante envolve todas as matrizes do termo tensorial.
  - caso médio: um valor V tal que $2 < V < M$. Esse caso significa que existem matrizes do tipo identidade no termo.
  - melhor caso: $V = 2$, ou seja, somente dois autômatos estão sincronizando atividades. Também indica que os AUNFs a serem criados são iguais ao produtório dos elementos não-nulos contabilizados até a primeira matriz identidade, o qual deve ser o valor escolhido para σ.

- Para termos generalizados, escolhe-se basicamente *onde* se avaliar as funções:
  - avaliar na parte esparsa: implica em converter implicitamente o termo tensorial para constante, uma vez que esse procedimento na realidade troca os elementos funcionais por elementos constantes;
  - avaliar na parte estruturada: aplicar as avaliações como são feitas no Algoritmo *Shuffle*, na parte estruturada, um número de vezes que depende dos valores dos deslocamentos $nright$ e $nleft$ da parte correspondente.

O objetivo é minimizar ao máximo o tempo gasto para realizar a MVD em cada termo. A escolha de qual matriz será posicionada em cada parte deve estar de acordo com a classificação do termo tensorial e as características envolvidas. Dessa forma, o uso de reordenamentos do termo tensorial é mandatório, uma vez que, para os termos constantes, não vale a pena posicionar matrizes identidades na parte esparsa (pois criam *AUNFs* desnecessários), devendo estas matrizes serem deslocadas para a *parte estruturada*. Para os termos generalizados, precisa-se da informação contida nas matrizes que fazem parte das dependências da função a ser avaliada.

Se um dado termo possuir uma função e escolhe-se realizar as avaliações na parte esparsa, todas as matrizes que a função depende, inclusive a matriz que a função está definida, devem estar nesta parte. O resto das matrizes que não dependem da função podem ser enviadas para qualquer um dos lados, sendo apenas necessário estudar qual lado oferecerá o melhor desempenho, dependendo da otimização a ser feita, em termos de memória ou de tempo. Caso seja uma matriz identidade que não é parâmetro para a função, esta é melhor tratada na parte estruturada.

Para o caso onde deseja-se aplicar as avaliações na parte estruturada, é necessário apenas que a função permaneça nessa parte, sendo que as outras podem ser posicionadas na parte esparsa. Novamente, neste caso é interessante reter as matrizes identidades na parte estruturada, pois serão descartadas na aplicação do Algoritmo *Shuffle*.

4.1.1 Exemplo de descritor markoviano e solução com o Algoritmo *Split*

Para o seguinte termo tensorial generalizado, composto por cinco autômatos (no caso, cinco matrizes), com diferentes valores para n_z ($\{1, 3, 1, 1, 4\}$), dimensões ($\{2, 3, 2, 3, 4\}$), tipos ($\{$*constante*, *identidade*, *funcional*, *elemento*, *identidade*$\}$) tem-se a função f definida no autômato $\mathcal{A}^{(3)}$ por:

$$f = \Big(st\ \mathcal{A}^{(1)} == 1\ \&\&\ st\ \mathcal{A}^{(5)} == 0 \Big) \times r$$

onde $r \in \mathbb{R}_+^*$ e st uma função *booleana* que recebe dois parâmetros, um autômato e um estado retornando uma avaliação baseado no estado corrente.

Seja σ o valor de corte do termo tensorial (com PSS $|\mathcal{X}| = 2 \times 3 \times 2 \times 3 \times 4 = 144$ estados):

$$\sigma_0 \quad \overset{\mathcal{A}^{(1)}}{\begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}} \quad \sigma_1 \quad \overset{\mathcal{A}^{(2)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \quad \sigma_2 \quad \overset{\mathcal{A}^{(3)}}{\begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix}} \quad \sigma_3 \quad \overset{\mathcal{A}^{(4)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{pmatrix}} \quad \sigma_4 \quad \overset{\mathcal{A}^{(5)}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \quad \sigma_5$$

Os valores possíveis para dividir o termo tensorial variam os pontos de corte de $\sigma_{0..5}$. Note-se que aplicando o Algoritmo *Split* para σ_0, a abordagem escolhida será o equivalente ao Algoritmo *Shuffle* e para σ_5 será o equivalente ao Algoritmo *Esparso*.

Pode-se escolher avaliar a função f na parte estruturada ou na parte esparsa. Caso fosse escolhido realizar as avaliações na parte *estruturada*, teria-se que reordenar o termo tensorial da seguinte maneira:

$$\sigma_0 \quad \overset{\mathcal{A}^{(2)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \quad \sigma_1 \quad \overset{\mathcal{A}^{(4)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{pmatrix}} \quad \underset{\sigma_2}{\downarrow} \quad \overset{\mathcal{A}^{(1)}}{\begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}} \quad \sigma_3 \quad \overset{\mathcal{A}^{(5)}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \quad \sigma_4 \quad \overset{\mathcal{A}^{(3)}}{\begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix}} \quad \sigma_5$$

parte esparsa — parte estruturada

Nesse caso, o conjunto das matrizes que fazem parte da dependência funcional de f é dado por $Dep(f) = \{\mathcal{A}^{(1)}, \mathcal{A}^{(5)}\}$. O corte escolhido para esse caso seria σ_2, com uma matriz do tipo identidade e uma constante para a parte estruturada e a outra matriz identidade, uma constante e a funcional para a parte esparsa, mantendo a consistência para o termo (conservando as dependências necessárias para que o método *Shuffle* consiga avaliar apropriadamente f). Para este caso, teriam-se três AUNFs[2] ($3 \times 1 = 3$).

No caso das avaliações serem feitas na parte *esparsa*, a matriz onde a função está definida entra no conjunto de dependências funcionais, obrigatoriamente, sendo este igual a $Dep(f) = \{\mathcal{A}^{(1)}, \mathcal{A}^{(5)}, \mathcal{A}^{(\mathbf{3})}\}$. Para esse caso, o termo tensorial seria reorganizado da seguinte forma:

$$\sigma_0 \quad \overset{\mathcal{A}^{(1)}}{\begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}} \quad \sigma_1 \quad \overset{\mathcal{A}^{(5)}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}} \quad \sigma_2 \quad \overset{\mathcal{A}^{(3)}}{\begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix}} \quad \underset{\sigma_3}{\downarrow} \quad \overset{\mathcal{A}^{(2)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} \quad \sigma_4 \quad \overset{\mathcal{A}^{(4)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{pmatrix}} \quad \sigma_5$$

parte esparsa — parte estruturada

Para este caso, o ponto de corte σ_3 foi escolhido e tem-se oito AUNFs ($2 \times 4 \times 1 = 8$). Para essa reconfiguração, pode-se escolher enviar a matriz *elemento* da parte estruturada para a parte esparsa, sem o aumento do número de AUNFs permanecendo apenas identidades na parte estruturada. Contudo, não seria possível enviar a identidade de $\mathcal{A}^{(5)}$ para a parte estruturada, uma vez que a função não conseguiria ser avaliada pois uma das matrizes que ela depende está em um local não acessível.

Uma alternativa válida, entretanto, com a preocupação de não aumentar o número de AUNFs que são gerados, é enviar a matriz do tipo elemento definida em $\mathcal{A}^{(4)}$ para a parte esparsa, deixando apenas identidades na parte

[2]Um AUNF é calculado como o produtório dos elementos não-nulos de uma matriz, de 0 a σ. Maiores informações na Seção 3.4.3.

estruturada. Para esse caso, o ponto de corte é igual a σ_4 e o termo tensorial seria assim reorganizado:

$$\underset{\sigma_0}{\mathcal{A}^{(1)}} \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix} \underset{\sigma_1}{\mathcal{A}^{(5)}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \underset{\sigma_2}{\mathcal{A}^{(3)}} \begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix} \underset{\sigma_3}{\mathcal{A}^{(4)}} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{pmatrix} \underset{\sigma_4}{\downarrow} \underset{\sigma_5}{\mathcal{A}^{(2)}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

parte esparsa — parte estruturada

Neste caso, a parte estruturada não terá custo computacional, pois as matrizes que a compõe são todas do tipo identidade. Uma outra importante constatação é que matrizes do tipo elemento, sempre podem ser enviadas para o lado esparso sem um ônus relativo ao aumento do número de AUNFs necessários. Caso essa mesma matriz permanecesse na parte estruturada, ainda assim contribuiria para aumentar os cálculos de deslocamentos à esquerda e à direita, pois o que importa para o Algoritmo *Shuffle* são em primeiro lugar as dimensões das matrizes e em segundo lugar o total de elementos não-nulos.

Ao eliminar a restrição de permitir que funções na parte estruturada possam acessar elementos da parte esparsa, propõe-se mais uma alternativa de reordenamento onde deixa-se apenas a matriz que contem o elemento funcional f mais as identidades na parte estruturada e as demais matrizes na parte esparsa. Esse caso é possível, apesar dos parâmetros das dependências funcionais estarem em partes diferentes, pois ao se calcular os AUNFs guardam-se os índices de linha que o originaram. Cabe ressaltar que o contrário seria impossível (*i.e.*, a função f estar na parte esparsa e uma ou as demais na parte estruturada) pois f precisa destes parâmetros que encontram-se na parte estruturada que não estão disponíveis.

Este caso possui a vantagem de não precisar guardar os AUNFs das matrizes identidades e é teoricamente uma das melhores maneiras de reordenar os termos tensoriais. Como as identidades serão desconsideradas pela parte estruturada e é possível avaliar as funções com sucesso, esta reordem oferece um baixo número de AUNFs necessários, pois a parte esparsa possui uma matriz do tipo elemento.

$$\underset{\sigma_0}{\mathcal{A}^{(1)}} \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix} \underset{\sigma_1}{\mathcal{A}^{(4)}} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{pmatrix} \underset{\sigma_2}{\downarrow} \underset{\sigma_3}{\mathcal{A}^{(2)}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \underset{\sigma_4}{\mathcal{A}^{(5)}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \underset{\sigma_5}{\mathcal{A}^{(3)}} \begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix}$$

parte esparsa — parte estruturada

É importante salientar que não é necessário adotar uma solução puramente esparsa para esse termo tensorial (corte em σ_5), pois aumentaria o número de AUNFs e seria oneroso, pois o método *Shuffle*, para esse caso, não será chamado e, consequentemente, menos operações para cálculos de deslocamentos serão feitos (estas análises teóricas serão feitas nas próximas seções). Igualmente desnecessário é se gastar espaço de memória armazenando-se AUNFs formados a partir da combinação de elementos de matrizes do tipo identidade. Também verifica-se que o aumento da memória não necessariamente implica em um ganho de desempenho, ainda mais para esse caso, que armazenaria os AUNFs com mesmos valores de escalares e nunca aproveitaria os saltos na estrutura tensorial para melhorar a execução.

Este simples exemplo mostra que é crucial alterar a ordem natural das matrizes do termo tensorial para obter uma melhor organização e otimização, uma vez que essa tarefa interfere diretamente na complexidade de execução do método. O custo computacional de se usar reordenamentos é baixo e realizado de forma transparente para o usuário, pois trata-se fundamentalmente de cálculos dos deslocamentos[3], ou seja, novos valores para $nright, nleft$

[3]Esses cálculos são melhores explicados na Seção 3.5.

e $njump$, equivalendo respectivamente aos saltos do subespaço à direita, à esquerda e os saltos na estrutura tensorial de acordo com a dimensão da matriz não-identidade no fator normal, conforme a notação descrita na Seção 3.4 que são feitos apenas na primeira iteração do método e reaproveitados subsequentemente até o final. Este exemplo demonstrou que devem ser adotadas diferentes estratégias para diferentes termos tensoriais, uma vez que é possível definir pontos de corte personalizados, tendo-se em vista as características dos termos.

Uma vez que o problema foi informalmente estudado com as possibilidades de divisão do termo tensorial, nota-se que um conjunto finito de características afetam o desempenho da solução da MVD. Este conjunto é dado por: a) as dimensões das matrizes equivalente aos estados dos autômatos, b) o total de elementos não-nulos, c) o total de avaliações de funções que são feitas, e d) total de matrizes identidades nos termos tensoriais. Estas características afetam diretamente a complexidade dos métodos escolhidos para multiplicação.

A idéia principal deste capítulo é definir as principais estratégias para dividir um termo tensorial de uma maneira que não onere a memória gasta e que otimize o tempo de solução. Como explicado anteriormente, esta divisão implicou na obrigatoriedade de reorganização do termo tensorial. Essa reconfiguração não é custosa em termos computacionais, pois trata-se de cálculos de deslocamentos que são computados na primeira vez e depois apenas consultados na execução dos métodos. O objetivo então é aplicar a melhor transformação ao termo tensorial e dividi-lo em partes que otimizem o tempo de resposta de modelos. Para tanto, é necessário estudar os efeitos de cada propriedade dos termos tensoriais para determinar a melhor maneira de se fazer essa operação. A seguir é realizado um estudo teórico sobre estas propriedades, com o objetivo de deixar claras as escolhas a serem feitas para a solução otimizada de descritores ATC e ATG. Este estudo permitirá antecipar os desempenhos esperados de cada experiência a ser realizada no Capítulo 5, referente aos resultados obtidos.

4.2 Análise teórica

Uma vez definidos os compromissos de desempenho do Algoritmo *Split* e as formas de divisão dos termos tensoriais para otimizar o tempo gasto na solução de modelos baseados em descritores markovianos, a próxima etapa é estudar as implicações teóricas existentes. Um termo tensorial é composto por uma lista de matrizes de diferentes dimensões, tipos e elementos não-nulos. Cada matriz contém informação relevante sobre como os autômatos sincronizam atividades, ou seja, como estes *interferem* uns nos outros. Essa interferência dita como serão formadas as matrizes, ou seja, apontarão a existência de identidades onde um evento não ocorrer em um autômato e matrizes do tipo elemento, indicando a ocorrência de um evento de acordo com uma determinada taxa de ocorrência. Cabe ressaltar que a terminologia usada aqui será usada sem perda de generalidade para *envolvimento* (ou envolver), *interferência* (ou interferir), *afetação* (ou afetar) quando disserem respeito aos autômatos.

A seguir, um estudo das implicações de cada tipo de característica existente em termos tensoriais e o que correspondem ao nível do algoritmo que será executado:

1. dimensão de cada matriz: influenciam os subespaços à direita e à esquerda dos fatores normais, ou seja, implicam no número de vezes que o algoritmo é executado por iteração;
2. tipo da matriz: uma matriz, como discutido anteriormente, pode ser constante, identidade, elemento ou funcional. Cada tipo pode otimizar um diferente aspecto, por exemplo, uma matriz funcional pode otimizar o número de avaliações que são feitas, enquanto que matrizes elemento e identidades dizem respeito à memória que será gasta no método para solução e o número de operações de multiplicação;
3. permutação do termo: um termo tensorial pode sofrer uma transformação na sua ordem original e ter a sequência das suas matrizes reordenadas para potencializar o método de execução da MVD;
4. ponto de corte σ: o ponto de corte determina o quanto de memória precisará ser armazenado para efetuar o método. Este valor deve ser escolhido de forma a maximizar também o tempo de solução.

A seguir é feita uma análise mais aprofundada sobre estas características e seus efeitos na complexidade computacional envolvida, começando-se pelo estudo do efeito das identidades nos termos tensoriais.

4.2.1 Efeito das identidades

O efeito das matrizes do tipo identidade no termo tensorial pode ser comprovado de duas maneiras, em termos constantes e em termos generalizados. Em termos constantes, as identidades podem fazer parte da seção esparsa do termo, executando o Algoritmo *Esparso*, ou da seção estruturada, que executará o Algoritmo *Shuffle*. Por um lado, como mencionado anteriormente, matrizes do tipo identidade na parte esparsa aumentam o número de AUNFs necessários sem guardar informação relevante. Se estas matrizes fossem deslocadas para a parte estruturada, levariam à execução de um menor número de multiplicações em ponto flutuante para o cálculo dos deslocamentos que são necessários ao Algoritmo *Shuffle*. Isso ocorre porque, ao multiplicar uma matriz por seus fatores normais, se esta for uma identidade, o método *não* executará nenhuma operação para esse termo, resultando em um ganho considerável em termos de tempo.

Este efeito benéfico é ampliado se a parte estruturada *apenas* conter matrizes do tipo identidade. Nesse caso, somente é necessário o tempo inicial gasto para se descobrir os AUNFs de cada termo tensorial e depois só é necessário multiplicar estes em índices pré-calculados de linha e coluna para os vetores de entrada e saída do método para obter o vetor solução intermediário onde é verificada a convergência. Esse caso, para os termos tensoriais generalizados, implica que os elementos funcionais foram avaliados na parte esparsa, *i.e.*, este termo não precisa mais ser considerado um termo generalizado, sendo a partir desse momento um termo constante, sem haver a necessidade de outras avaliações de função.

Ao não precisar mais avaliar funções na parte estruturada, descarta-se toda a complexidade envolvida para realizar esta operação. Com isso, o termo generalizado é convertido para um termo constante, em tempo de execução, pois os as estruturas de dados auxiliares para armazenamento dos AUNFs contém os valores das funções já avaliadas. Caso o número de AUNFs existentes não inutilizem a execução do Algoritmo *Split* (passando-se dos limites disponíveis de memória, por exemplo), esta é teoricamente a melhor solução disponível em termos de número de operações em ponto flutuante de multiplicações.

O real ganho do efeito das identidades nos termos tensoriais pode ser avaliado de duas formas, com permutação das matrizes do termo ou sem permutar e utilizar algum critério pré-definido de escolha do ponto de corte σ. Este último caso retira da complexidade as operações de permutação envolvidas. Entretanto, como estas permutações são na verdade pré-cálculos de deslocamentos na estrutura tensorial, realizados apenas uma vez no início do método, são negligenciáveis do ponto de vista da aplicação. Logo, esta maneira resulta em uma otimização possível das reordens das matrizes identidades tanto para termos constantes quanto termos generalizados.

A existência de matrizes do tipo identidade está fortemente relacionada ao termo tensorial informando os autômatos que um dado evento interfere. Dado um evento sincronizante e, se este está sincronizando uma atividade com todos os autômatos em uma determinada transição, este termo não será composto por nenhuma identidade e diversas matrizes do tipo elemento. A abordagem flexível proposta pelo Algoritmo *Split* auxilia na escolha de opções interessantes para o ponto de corte σ, baseadas na memória disponível. Ao verificar um modelo observando os seus eventos sincronizantes e funções é possível antecipar, de maneira teórica, uma boa forma de tratar com os termos tensoriais. A seguir, analisa-se a influência das funções nos termos tensoriais.

4.2.2 Influência do custo de avaliações dos elementos funcionais

A avaliação de funções é uma operação que depende de muitos fatores em um modelo SAN. Este custo depende basicamente de três fatores [34]:

1. Os argumentos da função. Estes valores são calculados e utilizam como entrada um índice global do espaço de estados do sistema;

2. A avaliação da função propriamente dita. Uma vez que se conhece os argumentos, a função pode ser avaliada, retornando o valor da avaliação;

3. O número de avaliações que são feitas. Esse valor é dependente do bloco correspondente ao subespaço da esquerda do método. O método para avaliação de funções é chamada inúmeras vezes, de acordo com o Algoritmo 3.3. O método trata as matrizes de acordo com uma ordem de permutação e converte os seus elementos funcionais em elementos constantes.

Ao dividir o termo tensorial de forma que as funções sejam avaliadas na parte estruturada, aumenta-se o número de operações que são feitas, uma vez que as avaliações necessárias são feitas em cada iteração e sempre resultam no mesmo conjunto de valores. O oposto desta escolha é se fazer dividir o termo tensorial passando todas as matrizes dependentes e a matriz que contém o elemento funcional para a parte esparsa e chamar o método de avaliação de função uma vez no início da solução, guardando os AUNFs já convertidos para escalares não-nulos constantes. Ao se fazer isso, os termos tensoriais generalizados são convertidos para termos constantes e não existem mais avaliações de função como existia anteriormente e o método deve rodar mais rapidamente.

4.2.3 Custo das permutações nos termos tensoriais

Existem inúmeras formas de se permutar um termo tensorial τ composto por matrizes de diversos tipos e esparsidades, sem replicação. O ideal é se descobrir o melhor jeito de reorganizar o termo tensorial para que ele utilize o melhor método em ambos os lados da divisão minimizando o tempo que será gasto para resolver o modelo.

Para permutar um termo tensorial e obter uma ordem de posicionamento ideal existe um custo computacional, para calcular os índices referentes aos deslocamentos necessários que serão utilizados posteriormente no método para que ele salte na estrutura tensorial. Esse reposicionamento faz com que o método seja executado como anteriormente, mas acessando os elementos a estrutura de outra forma, no caso, permutada.

Esses custos de recálculos de índices são negligenciáveis, pois são calculados apenas uma vez e utilizados sempre em cada iteração. Trata-se de saltos e novos valores para tanto os subespaços à esquerda quanto aos da direita do método. O problema da reordenação é realmente descobrir um novo posicionamento que seja ideal e, quando combinado ao ponto de corte, auxiliará para diminuir o tempo requerido para solucionar um modelo e retornar o vetor de probabilidade estacionário final.

Cabe lembrar que cada termo tensorial pode ser reorganizado de uma maneira diferente, devido principalmente aos elementos que o compõem. Este reposicionamento pode levar em consideração onde será executado o Algoritmo *Split*, o quanto existe de memória disponível no sistema, a esparsidade do termo tensorial, o número de identidades e o número de matrizes que serão vistas como os argumentos da função no caso de termos generalizados.

Dependendo do que maximizar ou minimizar, uma nova ordem deve ser produzida. Se as avaliações devem ser reduzidas a um extremo, o termo deverá ser recomposto para que estas sejam enviadas para a parte estruturada ou esparsa. Pode também ser constatado que os argumentos dos elementos funcionais são melhores executados na parte estruturada, que possuem otimizações para esse propósito específico.

4.3 Considerações

A Tabela 4.1 mostra um comparativo das complexidades de cada escolha de divisão do termo tensorial para o Algoritmo *Shuffle* e para o Algoritmo *Split*. No caso do *Shuffle*, tem-se a complexidade do método e a otimização das identidades, que somente caso a matriz não seja uma identidade, o bloco é executado. Para a complexidade do *Split*, tem-se outras possibilidades, tais como execução do método avaliando-se funções na parte esparsa e na parte estruturada e seu efeito direto no número de multiplicações em ponto flutuante que são requeridas.

Nota-se claramente que o melhor custo computacional teórico foi obtido quando, independente do termo ser constante ou generalizado, as avaliações são feitas apenas uma vez, permutando-se as matrizes identidades para

<table>
<tr><td colspan="2">Complexidade Shuffle</td><td>$$\sum_{i=1}^{N} nleft_i \times nright_i \times nz_i = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}$$</td></tr>
<tr><td colspan="2"></td><td>¶ Complexidade do método completo, sem otimizações.</td></tr>
<tr><td colspan="2">Complexidade Shuffle c/otimização</td><td>$$\prod_{i=1}^{N} n_i \times \sum_{\substack{i=1 \\ \text{iff } Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}$$</td></tr>
<tr><td colspan="2"></td><td>¶ Só entra no bloco se matriz não é uma identidade.</td></tr>
<tr><td rowspan="8">Complexidade Split</td><td rowspan="2">método normal</td><td>$$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1)+\left(\prod_{i=\sigma+1}^{N} n_i\right)+\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right)\right]$$</td></tr>
<tr><td>¶ Custo para calcular um AUNF e depois executar Algoritmo Shuffle.</td></tr>
<tr><td rowspan="2">identidades na parte estruturada</td><td>$$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1)+\left(\prod_{i=\sigma+1}^{N} n_i\right)\right]$$</td></tr>
<tr><td>¶ Trata-se do custo para se calcular e multiplicar os AUNFs.</td></tr>
<tr><td>avaliações na parte esparsa</td><td>¶ As avaliações de funções dependem da quantidade de parâmetros necessários, i.e., o número de autômatos que são necessários para o cálculo da complexidade. Custo de avaliação negligenciável, mais custo do Algoritmo Split.</td></tr>
<tr><td>avaliações na parte estruturada</td><td>¶ Avaliam-se as funções na parte estruturada, mais custo do cálculo dos AUNFs</td></tr>
<tr><td>avaliações na parte esparsa

identidades na parte estruturada</td><td>¶ Neste caso, todas as avaliações são feitas na parte esparsa e o Algoritmo Shuffle não é executado, pois só existem identidades na parte estruturada.</td></tr>
</table>

Tabela 4.1: Complexidades existentes para os Algoritmos *Shuffle* e *Split*.

a parte estruturada, caso a memória disponível permita. Nesse caso, as multiplicações necessárias são apenas as que dizem respeito ao cálculo dos AUNFs, deixando toda a parte estruturada sem ser executada, pois as matrizes restantes são identidades. Sendo identidades, os fatores normais gerados serão todos identidades, logo, descartáveis.

Observa-se ainda que o grau de dependência, ou seja, o GDC e o GDG (definidos na Seção 4.1) constituem-se em uma importante métrica para descobrir as características dos termos tensoriais observando-se apenas os eventos sincronizantes e as funções existentes na rede de autômatos. Estes graus afetam diretamente no tempo de execução, pois estão relacionados a questões númericas de eficiência, devido ao número de multiplicações exigido e aos saltos que são dados quando as matrizes são do tipo identidade.

O que realmente ocorre quando avalia-se as funções apenas uma vez, reordena-se o termo tensorial e deixa-se todas as identidades para a parte estruturada é que foi encontrada uma nova forma de perceber o termo tensorial e considerá-lo esparso, ou seja, de execução extremamente rápida, sem precisar chamar as operações da parte estruturada e gastando uma memória razoável. Para tanto, deve-se considerar o papel que os eventos possuem nos descritores markovianos. São os eventos dos termos tensoriais que ditam como o Algoritmo *Split* irá se comportar na sua execução. Eventos que dependem de muitos autômatos para fazer a transição ditarão o tamanho dessa nova matriz esparsa que é implicitamente construída.

Uma vez que a análise teórica mostrou que são executadas menos operações de multiplicações em ponto-flutuante para alguns casos, a próxima etapa é validá-la através da definição e execução de casos de teste e experimentações. Estas experimentações auxiliarão a determinar os melhores casos da MVD, como descrito no próximo capítulo.

5 Resultados numéricos

Este capítulo abordará os resultados numéricos obtidos para um conjunto fixo de experimentos, fundamentados sobre as proposições abordadas pela análise teórica feitas no capítulo anterior. Na Seção 5.1 são definidos os experimentos e as considerações iniciais. Cada modelo é apresentado e explicado seguido dos seus resultados, comparando-se os experimentos e verificando-se os ganhos obtidos a cada comparação na Seção 5.2. O capítulo é finalizado com questionamentos e discussões sobre os resultados obtidos na Seção 5.3.

5.1 Experimentos

O objetivo da definição dos experimentos conduzidos neste capítulo é testar os efeitos das diversas características dos termos tensoriais e verificar, em um ambiente real de teste e desenvolvimento, o desempenho comparando-se com as abordagens MVD vigentes. Deseja-se descobrir quais experimentos obtiveram os melhores tempos para uma iteração com o método de MVD, comparando-os com as abordagens para a mesma classe de modelos com descritores constantes (ou generalizados) e com (ou sem) permutações das ordens originais.

Foram escolhidos *nove* experimentos para execução do conjunto de modelos disponível de acordo com a Tabela 5.1, que mostra as principais diferenças entre cada um. A base de comparação é o método atual de solução baseado no Algoritmo *Shuffle* (executado nos Experimentos $2, 4$ e 9). As experiências foram executadas variando-se as possibilidades julgadas mais interessantes, como avaliação de funções e alteração das posições das matrizes identidades dentro de cada termo tensorial.

Para avaliar-se o custo envolvido na permutação dos elementos dentro do termo tensorial, foram criados os Experimentos $2, 4$ e $1, 3$ para respectivamente o Algoritmo *Shuffle* e o Algoritmo *Split*. Para o caso do Experimento 3, o ponto de corte escolhido é na última matriz constante do termo tensorial. O Experimento 8 calcula a implicação do custo das avaliações de funções, pois deixa apenas a matriz que contém o elemento funcional e as identidades na parte estruturada, utilizando somente as matrizes constantes na parte esparsa.

Os experimentos foram escolhidos devido à análise teórica conduzida na Seção 4.2. Os estudos efetuados indicaram a necessidade de se testar os diferentes métodos de MVD permutando-se o termo tensorial e dividindo-o de acordo com as suas dependências funcionais e outras características, tais como tipos de matrizes (identidades, entre outras) e esparsidade. Os exemplos aqui descritos possuem eventos sincronizantes com taxas funcionais para o caso generalizado onde também foram devidamente convertidos para o seu modelo equivalente constante. O objetivo desta conversão é o de verificar em quais casos é satisfatório traduzir descritores baseados em ATG para ATC, apesar do aumento do número de eventos no termo tensorial [14].

A execução dos experimentos foi realizada em uma arquitetura *Pentium* 3.2 GHz com 4 Gb de memória RAM. A ferramenta PEPS [13] foi utilizada como base onde implementou-se o Algoritmo *Split* para descritores constantes e generalizados. Também foi desenvolvida a parte para armazenamento e geração dos AUNFs e outras estruturas de dados auxiliares. O compilador C/C++ (g++ versão $4.0.4$) foi usado com opções para otimização ($-O3$) e *linking* dinâmico para as funções dos descritores generalizados. Estão sendo comparados o Algoritmo *Split* com o Algoritmo *Shuffle* com permutações para os Experimentos $1, 2, 5, 6, 7, 8$ e 9 e sem reordenamentos para os casos 3 e 4. Foram computados os tempos com intervalos de confiança de 95% para 50 execuções sequenciais com 25

Experimento	Método	Tipo de Descritor	Uso de Permutação	Descrição
1	*Split*	constante	$\surd$	identidades para parte estruturada
2	*Shuffle*	constante	$\surd$	método usual para descritores constantes
3	*Split*	constante	$\times$	sem permutações, cortar na última matriz constante
4	*Shuffle*	constante	$\times$	sem permutações, avaliação do custo da permutação
5	*Split*	generalizado	$\surd$	avaliação de funções na parte estruturada
6	*Split*	generalizado	$\surd$	avaliação de funções na parte esparsa
7	*Split*	generalizado	$\surd$	avaliação de funções na parte esparsa somente identidades para parte estruturada
8	*Split*	generalizado	$\surd$	avaliação de funções na parte estruturada somente identidades para parte estruturada
9	*Shuffle*	generalizado	$\surd$	método usual para descritores generalizados

Tabela 5.1: Detalhamento dos experimentos escolhidos para execução.

iterações cada (para o valor do tempo coletado foi calculado o tempo médio de execução do método considerando-se este número de iterações).

Serão apresentadas tabelas contendo os resultados obtidos para os diversos modelos. Estas tabelas de resultados mostram o nome do modelo em questão (campo *Modelo*), o seu *PSS* e uma divisão entre as informações relativas aos AUNFs, informações relativas ao tempo de execução (em *Tempos*) e ganhos comparando-se sempre dois experimentos (no caso, o ganho será calculado da seguinte maneira, a última coluna dividida pela anterior). Na parte relativa aos AUNFs, tem-se o seu total necessário para o experimento em questão (sempre será rodado um experimento para o Algoritmo *Split* e outro experimento para o Algoritmo *Shuffle*), a sua memória (no campo Mem), descrita em *Kilobytes* (Kb) e o tempo necessário para computar a tabela com os AUNFs, realizado apenas uma vez em todo o método de solução, na coluna *Cálculo*. O tempo dos experimentos são mostrados nas colunas correspondentes, medidos em *segundos* (s), mostrados na tabela com o seu intervalo de confiança de 95% correspondendo à média de uma iteração do método da potência (*Power Method*) fornecido pelo PEPS no arquivo de saída de dados ($.tim$).

Cabe ressaltar que os modelos variam o número de autômatos que os compõem, de forma incremental. Em casos específicos, como os modelos de *Redes de Sensores* e *Mestre-Escravo*, a definição original do descritor generalizado foi convertido explicitamente para o seu equivalente constante em termos de solução numérica.

5.2 Modelos

Todos os testes foram executados para uma lista de modelos (alguns tradicionais, como *Compartilhamento de Recursos* (*Resource/Sharing*) e outros oriundos de pesquisas atuais com arquitetura *Mestre-Escravo* (*Master-Slave*) ou *Redes de Sensores* (*Wireless Sensors*). A existência de *benchmarks* para testes ainda é insuficiente

em análise numérica de sistemas de avaliação de desempenho [16]. As tabelas em cada subseção comparam as diferentes estratégias de divisão dos termos tensoriais. Os demais modelos que compreendem essa seção definem o problema dos *Filósofos*, *First Available Server*, *Alternate Service Patterns* e duas realidades convertidas do formalismo PEPA: *Workcell* e *WebServer* (maiores informações na Seção 2.5). A seguir, uma breve explicação de cada modelo seguido dos resultados obtidos para todos os experimentos acima definidos.

5.2.1 Modelo *Mestre-escravo*

O modelo SAN definido na Figura 5.1 refere-se à implementação paralela do Algoritmo *Propagation* com comunicação assíncrona [5]. Este modelo é responsável por indicar aos desenvolvedores de aplicações paralelas os gargalos e problemas de configurações existentes antes do início da fase de implementação. Este modelo contem três estruturas particulares: um autômato denominado *Mestre*, um autômato que representa um *Buffer* (uma região temporária de armazenamento) e S autômatos *Escravos*, aqui denominados por $Slave^{(i)}$, onde $i = 1 \ldots S$.

O número total de autômatos para este modelo é dado por $(S+2)$ (onde S é o número de Escravos no modelo), o qual possui S eventos locais e $(3S-3)$ eventos sincronizantes para o caso de descritores clássicos e o mesmo número de autômatos mas $(2S+3)$ eventos sincronizantes para o caso dos descritores generalizados. Isso faz com que descritores baseados em ATC possuam $(7S-8)$ termos tensoriais e descritores ATG $(4S+6)$ termos.

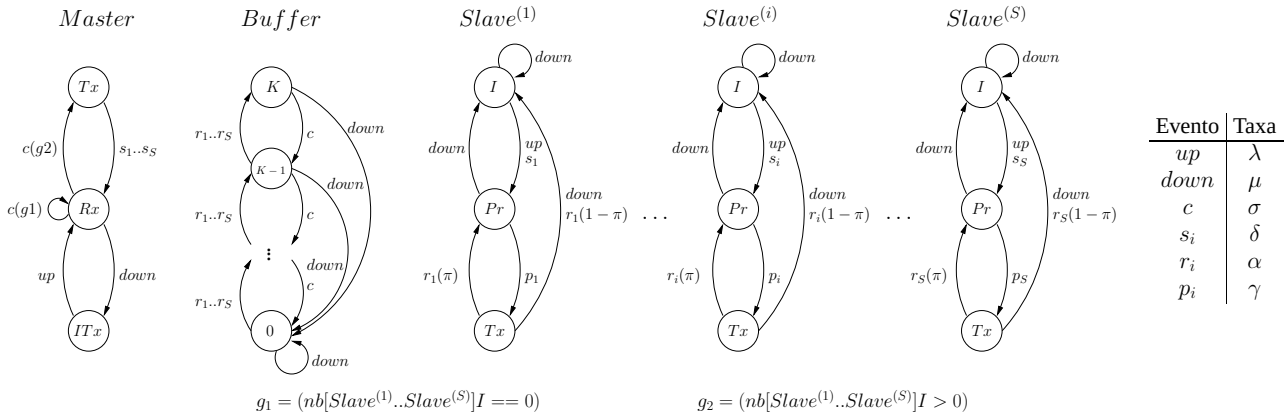


Figura 5.1: Modelo SAN paralelo para Mestre-Escravo.

A Tabela 5.2 mostra os resultados para os Experimentos $1, 2, 3$ e 4. Cabe ressaltar que as informações relativas ao cabeçalho das tabelas estão explicadas na Seção 5.1. Observa-se que dentre estes experimentos a melhor execução foi obtida quando as identidades encontram-se na parte estruturada. O uso de memória constatado foi pequeno (para 10 escravos de aproximadamente 76 Mb), tendo-se em vista o ganho obtido, da ordem de $quatro$ vezes em comparação com o Experimento 2). Os Experimentos 3 e 4 mostram os tempos quando não se permutam os termos, onde infere-se que o Algoritmo *Shuffle* executa mais rápido inclusive que sua versão com permutação, mostrando que o uso de reordenamentos está atrelado ao modelo e as suas características internas.

A Tabela 5.3 mostra os resultados para os Experimentos $5, 6, 7, 8$ e 9. Esta classe de modelos mostra a escalabilidade dos algoritmos, pois a cada incremento de *dois* escravos, tem-se um aumento da ordem de 10 vezes em relação ao tempo dispendido e à memória gasta. O melhor tempo foi verificado para o Experimento 6, que fornece uma boa relação custo/benefício em comparação com o Experimento 7, que além de gastar mais memória ainda leva mais tempo para realizar uma permutação. Para este caso específico, onde o PSS explode facilmente, verifica-se que com 79 Mb de memória tem-se um ganho de aproximadamente *três* vezes em relação à base de comparação do Experimento 9. Pode-se dizer que esta memória gasta é negligenciável pois a plataforma de execução trabalha com 4 Gb de memória RAM. Já no caso do Experimento 8, onde as avaliações são feitas na

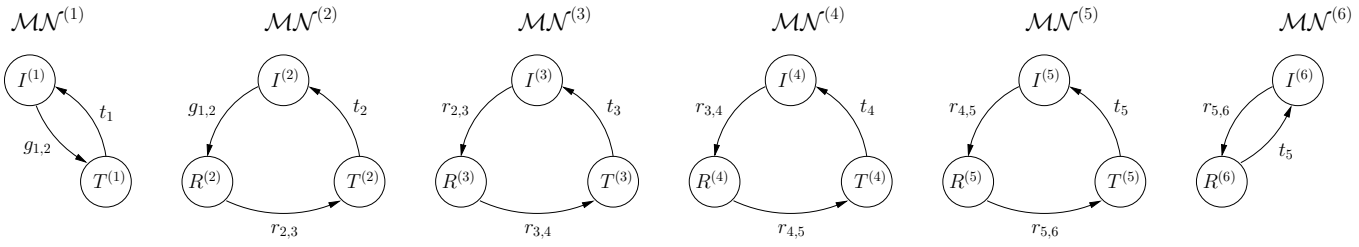
Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
slaves05c	29.889	11.849	370	0.0300 ± 0.0001	**0.0094** $\pm$ **0.0001**	0.0335 ± 0.0001	3.5638
slaves06c	89.667	33.176	1.036	0.0918 ± 0.0011	**0.0331** $\pm$ **0.0001**	0.1206 ± 0.0001	3.6435
slaves07c	269.001	95.635	2.988	0.3388 ± 0.0013	**0.1176** $\pm$ **0.0001**	0.4311 ± 0.0005	3.6658
slaves08c	807.003	280.210	8.756	0.9426 ± 0.0029	**0.3915** $\pm$ **0.0004**	1.4742 ± 0.0012	3.7655
slaves10c	7.263.027	2.463.180	76.974	10.2764 ± 0.0200	**4.1841** $\pm$ **0.0033**	16.1104 ± 0.0069	3.8504
					Experimento 3	Experimento 4	
slaves05c	29.889	50.125	1.566	0.1634 ± 0.0026	0.0179 ± 0.0001	0.0199 ± 0.0001	1.1117
slaves06c	89.667	149.334	4.6666	0.5014 ± 0.0063	0.0604 ± 0.0001	0.0756 ± 0.0001	1.2517
slaves07c	269.001	445.681	13.927	1.5754 ± 0.0156	0.1977 ± 0.0003	0.2970 ± 0.0004	1.5023
slaves08c	807.003	1.332.162	41.630	4.7496 ± 0.0359	0.6333 ± 0.0009	1.0292 ± 0.0007	1.6251
slaves10c	7.263.027	11.939.214	373.100	45.1638 ± 0.2744	6.6515 ± 0.0062	11.3418 ± 0.0385	1.7051

Tabela 5.2: Resultados para o modelo *Mestre-Escravo*, baseados em ATC e Experimentos $\{1, 2$ e $3, 4\}$.

parte estruturada e não existem matrizes constantes, para esse modelo em particular, devido à natureza das funções existentes, observou-se que os cortes foram os mesmos que os praticados pelo Experimento 5. Os tempos e o total de AUNFs obtidos entre esses dois casos atestam que correspondem ao mesmo caso.

Uma outra comparação factível de ser feita diz respeito à conversão implícita do descritor baseado em ATG para descritor ATC, uma vez que as funções estão sendo avaliadas na parte esparsa, para o experimento que executou mais rapidamente (no caso, o Experimento 6). Para esse caso, converter ou não converter não implica necessariamente em um ganho substancial, pois, para o caso de 10 escravos, o tempo para o modelo constante é de 4.1841 segundos e para o modelo generalizado é de 4.2506 segundos, ou seja, um pequeno ganho em termos de tempo. O Experimento 8 mostra que essa conversão já é mais interessante, pois, para o mesmo caso, tem-se 16.1104 segundos para o caso constante e 13.3783 segundos para o baseado em ATG. Cabe ressaltar que uma diferença de três segundos por iteração pode resultar em uma diferença considerável se supor-se que são necessárias 10.000 iterações para a solução com o *Power Method*, ou seja, aproximadamente 8 horas.

5.2.2 Modelo Redes de Sensores



Tipo	Evento	Taxa	Tipo	Evento	Taxa
loc	$t_1, \ldots, t_4$	μ	syn	$r_{2,3}$	$f_{2,3}$
syn	t_5	μ	syn	$r_{3,4}$	$f_{3,4}$
syn	$g_{1,2}$	$f_{1,2}$	syn	$r_{4,5}$	$f_{4,5}$
			syn	$r_{5,6}$	$f_{5,6}$

$f_{1,2} = [(\quad (st\ \mathcal{MN}^{(3)} != T^{(3)}) \ \&\& \ (st\ \mathcal{MN}^{(4)} != T^{(4)}) \ )] * \lambda$

$f_{2,3} = [(\quad (st\ \mathcal{MN}^{(1)} != T^{(1)}) \ \&\& \ (st\ \mathcal{MN}^{(4)} != T^{(4)}) \ \&\& \ (st\ \mathcal{MN}^{(5)} != T^{(5)}) \ )] * \lambda$

$f_{3,4} = [(\ (st\ \mathcal{MN}^{(1)} != T^{(1)}) \ \&\& \ (st\ \mathcal{MN}^{(2)} != T^{(2)}) \ \&\& \ (st\ \mathcal{MN}^{(5)} != T^{(5)}) \ \&\& \ (st\ \mathcal{MN}^{(6)} != T^{(6)}) \ )] * \lambda$

$f_{4,5} = [(\ (st\ \mathcal{MN}^{(2)} != T^{(2)}) \ \&\& \ (st\ \mathcal{MN}^{(3)} != T^{(3)}) \ \&\& \ (st\ \mathcal{MN}^{(6)} != T^{(6)}) \quad)] * \lambda$

$f_{5,6} = [(\ (st\ \mathcal{MN}^{(3)} != T^{(3)}) \ \&\& \ (st\ \mathcal{MN}^{(4)} != T^{(4)}) \quad)] * \lambda$

Figura 5.2: Modelo de Redes de Sensores *Ad Hoc* com 6 nós definido em ATG.

O modelo SAN descrito na Figura 5.2 representa uma cadeia de nós móveis em uma Rede de Sensores (de

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	Experimento 5	Experimento 9	
slaves05f	29.889	10.409	325	0.0200 ± 0.0001	0.0138 ± 0.0001	0.0274 ± 0.0001	1.9855
slaves06f	89.667	30.416	950	0.0802 ± 0.0004	0.0480 ± 0.0001	0.0985 ± 0.0004	2.0521
slaves07f	269.001	90.275	2.821	0.2872 ± 0.0015	0.1670 ± 0.0002	0.3476 ± 0.0003	2.0814
slaves08f	807.003	269.690	8.427	0.9188 ± 0.0019	0.5539 ± 0.0004	1.2023 ± 0.0014	2.1706
slaves10f	7.263.027	2.421.860	75.683	9.1362 ± 0.0168	5.9836 ± 0.0052	13.3783 ± 0.0049	2.2358

					Experimento 6	Experimento 9	
slaves05f	29.889	10.855	339	0.0218 ± 0.0011	**0.0096 ± 0.0001**	0.0274 ± 0.0001	2.8542
slaves06f	89.667	31.834	994	0.0842 ± 0.0014	**0.0346 ± 0.0001**	0.0985 ± 0.0004	2.8468
slaves07f	269.001	94.609	2.956	0.2974 ± 0.0018	**0.1213 ± 0.0001**	0.3476 ± 0.0003	2.8656
slaves08f	807.003	282.772	8.836	0.9608 ± 0.0034	**0.3985 ± 0.0003**	1.2023 ± 0.0014	3.0171
slaves10f	7.263.027	2.539.918	79.372	9.4901 ± 0.0099	**4.2506 ± 0.0028**	13.3783 ± 0.0049	3.1474

					Experimento 7	Experimento 9	
slaves05f	29.889	29.809	931	0.0676 ± 0.0012	0.0134 ± 0.0001	0.0274 ± 0.0001	2.0448
slaves06f	89.667	88.696	2.771	0.2134 ± 0.0014	0.0474 ± 0.0001	0.0985 ± 0.0004	2.0781
slaves07f	269.001	265.195	8.287	0.7150 ± 0.0026	0.1621 ± 0.0002	0.3476 ± 0.0003	2.1444
slaves08f	807.003	794.530	24.829	2.4072 ± 0.0053	0.5215 ± 0.0004	1.2023 ± 0.0014	2.3055
slaves10f	7.263.027	7.145.740	223.304	23.2650 ± 0.0356	5.4150 ± 0.0052	13.3783 ± 0.0049	2.4706

					Experimento 8	Experimento 9	
slaves05f	29.889	10.409	325	0.0202 ± 0.0004	0.0138 ± 0.0001	0.0274 ± 0.0001	1.9855
slaves06f	89.667	30.416	950	0.0808 ± 0.0008	0.0480 ± 0.0001	0.0985 ± 0.0004	2.0521
slaves07f	269.001	90.275	2.821	0.2850 ± 0.0018	0.1664 ± 0.0001	0.3476 ± 0.0003	2.0889
slaves08f	807.003	269.690	8.427	0.9184 ± 0.0035	0.5476 ± 0.0009	1.2023 ± 0.0014	2.1956
slaves10f	7.263.027	2.421.860	75.683	9.1464 ± 0.0598	5.9853 ± 0.0039	13.3783 ± 0.0049	2.2352

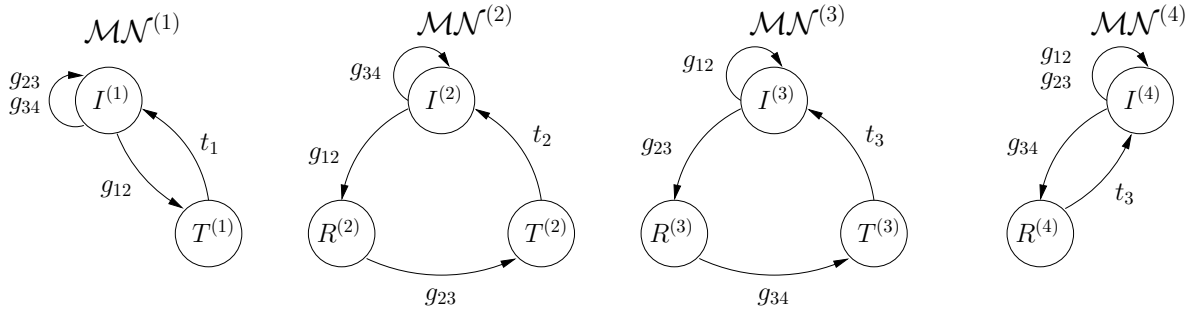
Tabela 5.3: Caso *Mestre-Escravo,* baseados em ATG e Experimentos $\{5, 6, 7, 8, 9\}$.

forma Ad Hoc) definida usando-se ATG, executada sobre o padrão 802.11. Este modelo [30] é uma adaptação do experimento de uma rede ad hoc presente em [54]. A cadeia tem N nós que se movimentam, onde o primeiro é chamado $\mathcal{MN}^{(1)}$ (autômato *Source*) que gera pacotes de acordo com as definições de um padrão de comunicação. Os pacotes são enviados através desta cadeia por autômatos do tipo *Relay* chamados de $\mathcal{MN}^{(i)}$, onde i varia entre 2 e $(N-1)$, até chegar ao último nó que foi chamado de $\mathcal{MN}^{(N)}$ (ou autômato *Sink*).

Os modelos SAN aqui descritos possuem genericamente $(N-2)$ eventos locais e N eventos sincronizantes para descritores contantes onde N é o número de nós móveis. Nesse caso, existem $[(N-2)+2N]$ termos tensoriais. Para o caso de descritores generalizados, tem-se $(N-2)$ eventos locais e igualmente N eventos sincronizantes com N termos. Este modelo foi aumentado para refletir um maior número de nós em uma rede com $10, 12, 14$ e 16 nós.

A Figura 5.3 mostra um modelo com um descritor markoviano baseado em ATC de uma Rede de Sensores com quatro nós que foi traduzida da equivalente rede original que utilizava definições de descritores baseados em ATG. Observa-se que para essa classe de modelos existem mais eventos sincronizantes, devido à conversão dos elementos funcionais em sincronizações [14].

A Tabela 5.4 informa os resultados obtidos para os Experimentos $1, 2, 3$ e 4. Observa-se que o experimento que melhor executou foi o Experimento 1, com um ganho de aproximadamente *sete* vezes em relação ao Experimento 2. A tabela também mostra que as permutações impactam de forma dramática na memória utilizada e, para este caso, no tempo. Nota-se que os termos foram reordenados de forma a otimizar por completo a geração de AUNFs



Tipo	Evento	Taxa	Tipo	Evento	Taxa
loc	t_1	μ	syn	g_{12}	λ_{12}
loc	t_2	μ	syn	g_{23}	λ_{23}
syn	t_3	μ	syn	g_{34}	λ_{34}

Figura 5.3: Modelo de Redes de Sensores *Ad Hoc* com quatro nós usando-se ATC.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
ad10c	26.244	10	≈0	$\approx 0.0000 \pm 0.0000$	**0.0028 ± 0.0001**	0.0221 ± 0.0001	7.8929
ad12c	236.196	12	≈0	$\approx 0.0000 \pm 0.0000$	**0.0382 ± 0.0001**	0.2583 ± 0.0039	6.7618
ad14c	2.125.764	14	≈0	$\approx 0.0000 \pm 0.0000$	**0.3940 ± 0.0010**	2.7428 ± 0.0012	6.9614
ad16c	19.131.876	16	≈0	$\approx 0.0000 \pm 0.0000$	**4.0235 ± 0.0174**	28.1046 ± 0.0729	6.9851

					Experimento 3	Experimento 4	
ad10c	26.244	5.105	159	0.0086 ± 0.0010	0.0038 ± 0.0001	0.0157 ± 0.0001	4.1316
ad12c	236.196	45.929	1.435	0.1196 ± 0.0008	0.0508 ± 0.0008	0.2031 ± 0.0004	3.9980
ad14c	2.125.764	413.345	12.917	1.1962 ± 0.0041	0.5024 ± 0.0041	2.1787 ± 0.0020	4.3366
ad16c	19.131.876	3.720.089	116.252	11.3510 ± 0.0593	5.0693 ± 0.0593	22.6656 ± 0.0008	4.4711

Tabela 5.4: Caso *Redes de Sensores*, modelos constantes, comparando Experimentos $\{1, 2, 3, 4\}$.

(apenas 16 para o maior caso, com 16 nós), refletindo no melhor tempo obtido. Também evidenciou que não necessariamente o uso de mais memória implicará em uma melhor execução; no caso do Experimento 3, foram gastos 116 Mb e ainda assim o tempo foi superior ao constatado no Experimento 1.

Já a Tabela 5.5 exemplifica os resultados para os Experimentos $5, 6, 7, 8$ e 9 para o caso dos modelos com descritores generalizados. Estes casos merecem um comentário específico para cada experimento. No Experimento 5, tem-se a avaliação das funções na parte estruturada. Nota-se que em relação ao Experimento 9 existe um ganho de aproximadamente 2.6 vezes, entretanto, foi utilizado por volta de 40 Mb de memória para o caso dos 16 nós, um valor muito superior ao requerido pelos demais experimentos.

No entanto, apesar de ambos os Experimentos 6 e 7 utilizarem a mesma memória, executam em temos completamente diferentes (um em 11.5802 segundos e o outro em 6.6018 segundos, respectivamente, para o caso dos 16 nós na rede de sensores). Esse fato pode ser explicado pela existência de matrizes do tipo elemento no modelo, que, como mencionado anteriormente, não aumentam o número de AUNFs necessários e mesmo assim, por deixar apenas identidades na parte estruturada, executam mais otimizadamente. O ganho observado foi o maior constatado em todos os modelos executados, da ordem de 12 vezes mais rápido, para um PSS de tamanho elevado para este caso em questão, ou seja, 19 milhões aproximadamente.

O Experimento 8 faz as avaliações de funções na parte estruturada e troca as matrizes identidade que existam na

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	Experimento 5	Experimento 9	
ad10f	26.244	1.513	47	$\approx 0.0000 \pm 0.0000$	0.0177 ± 0.0001	0.0438 ± 0.0002	2.4746
ad12f	236.196	14.257	445	0.0196 ± 0.005	0.2262 ± 0.0004	0.5679 ± 0.0009	2.5106
ad14f	2.125.764	134.137	4.191	0.2432 ± 0.013	2.6409 ± 0.0029	6.8353 ± 0.0076	2.5882
ad16f	19.131.876	1.259.713	39.366	2.5310 ± 0.042	29.8566 ± 0.0445	79.3119 ± 0.2193	2.6564
					Experimento 6	Experimento 9	
ad10f	26.244	406	12	$\approx 0.0000 \pm 0.0000$	0.0067 ± 0.0001	0.0438 ± 0.0002	6.5373
ad12f	236.196	568	17	$\approx 0.0000 \pm 0.0000$	0.0937 ± 0.0001	0.5679 ± 0.0009	6.0608
ad14f	2.125.764	730	22	$\approx 0.0000 \pm 0.0000$	1.0963 ± 0.0020	6.8353 ± 0.0076	6.2349
ad16f	19.131.876	892	27	$\approx 0.0000 \pm 0.0000$	11.5802 ± 0.0101	79.3119 ± 0.2193	6.8489
					Experimento 7	Experimento 9	
ad10f	26.244	406	12	$\approx 0.0000 \pm 0.0000$	$\mathbf{0.0040 \pm 0.0001}$	0.0438 ± 0.0002	10.9500
ad12f	236.196	568	17	$\approx 0.0000 \pm 0.0000$	$\mathbf{0.0568 \pm 0.0001}$	0.5679 ± 0.0009	9.9982
ad14f	2.125.764	730	22	$\approx 0.0000 \pm 0.0000$	$\mathbf{0.6434 \pm 0.0011}$	6.8353 ± 0.0076	10.6221
ad16f	19.131.876	892	27	$\approx 0.0000 \pm 0.0000$	$\mathbf{6.6018 \pm 0.0078}$	79.3119 ± 0.2193	12.0137
					Experimento 8	Experimento 9	
ad10f	26.244	10	≈ 0	0.0000 ± 0.0000	0.0165 ± 0.0001	0.0438 ± 0.0002	2.6545
ad12f	236.196	12	≈ 0	0.0000 ± 0.0000	0.2137 ± 0.0013	0.5679 ± 0.0009	2.6575
ad14f	2.125.764	14	≈ 0	0.0000 ± 0.0000	2.5059 ± 0.0027	6.8353 ± 0.0076	2.7277
ad16f	19.131.876	16	≈ 0	0.0000 ± 0.0000	28.3623 ± 0.0241	79.3119 ± 0.2193	2.7964

Tabela 5.5: Caso *Redes de Sensores*, modelos generalizados, comparando Experimentos $\{5, 6, 7, 8, 9\}$.

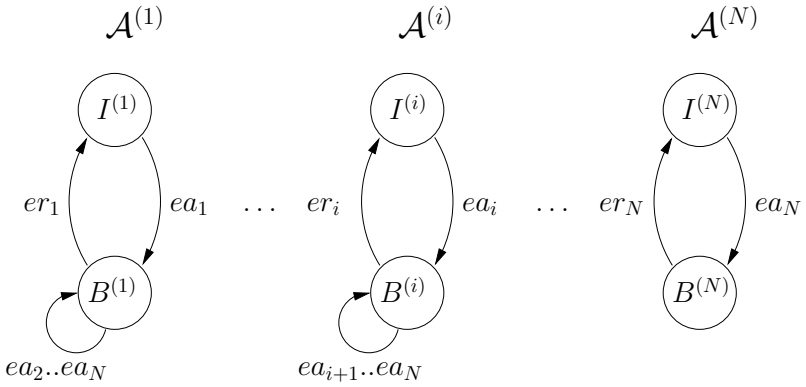
parte esparsa com matrizes constantes. Na parte estruturada só existem matrizes funcionais ou identidades. Esse experimento mostra que foram necessários um número muito baixo em termos de AUNFs, mas mesmo assim, os tempos se comportaram equivalentemente aos obtidos no Experimento 5. Esse fato corrobora que a avaliação de funções onera o desempenho do método da MVD, pois observa-se que, ao trocar esses elementos funcionais por constantes e enviar as identidades para a parte estruturada, os ganhos serão superiores, como visto no Experimento 7. Este caso, para 16 nós, precisou de 6.6018 segundos para uma iteração, valores ainda superiores aos 4.0235 segundos necessários no modelo constante. Para esses casos, observa-se que é melhor converter os modelos de ATG para ATC. No entanto, o Experimento 8 é eficiente em memória, pois precisa no maior caso de apenas 16 AUNFs.

Constata-se que este caso específico das *Redes de Sensores* também possuem uma outra característica marcante, todos os eventos possuem muitas identidades e todas as matrizes restantes possuem apenas um elemento não-nulo, fazendo com que cada termo produza *um* AUNF apenas. Esse é o melhor caso para o Algoritmo *Split*, pois a memória gasta é desprezível e o tempo de solução o mais otimizado possível. Nota-se que os modelos que seguem essas características resultam em um tempo mais otimizado para realizar a MVD. Observa-se que o Algoritmo *Shuffle*, apesar de realizar importantes otimizações para as matrizes identidades, ainda é necessário que ele trate das matrizes esparsas, calculando saltos na estrutura tensorial a cada iteração. No caso da aplicação do Algoritmo *Split*, o melhor a se fazer seria calcular um AUNF, multiplicar nas posições corretas dos vetores de entrada, salvar no vetor de saída e descartar a parte estruturada.

5.2.3 Modelo *First Available Server*, ou FAS

Esta seção apresenta o modelo SAN construído para analisar a disponibilidade do primeiro servidor desocupado (*First Available Server*), para N servidores. Cada servidor $A^{(i)}$ pode ser descrito como possuidor de dois estados distintos: *Idle* (em espera) ($I^{(i)}$) e *Busy* (ocupado) ($B^{(i)}$). Neste exemplo, pacotes que chegam no *terminal de servidores bloqueados* (ou *servers switch block*, partem através da primeira porta de saída (ou servidor) que não está ocupada, a única restrição é a de que pelo menos um servidor não está bloqueado.

O modelo está definido na Figura 5.4 e pode ser visto como uma ferramenta de análise de diferentes sistemas de filas (por exemplo, ocupação de linhas de *call centers*). Cada pacote que chega na fila pode avançar tão logo seja possível que um determinado servidor esteja livre, de acordo com uma prioridade pre-estabelecida. O PSS é formado por 2^N estados.



Tipo	Evento	Taxa
loc	ea_1	λ
loc	er_1	μ
syn	ea_2	λ
loc	er_2	μ
⋮	⋮	⋮
syn	ea_N	λ
loc	er_N	μ

Figura 5.4: Modelo *First Available Server*.

A Tabela 5.6 mostra os resultados para os Experimentos $1, 2, 3$ e 4. A partir da tabela observa-se que para esse modelo em particular, não importa reorganizar os termos (Experimento 1) ou cortar na última matriz constante (Experimento 3), o método realiza a iteração com uma pequena diferença. Outra característica deste modelo é precisar de um número extremamente reduzido de AUNFs para executar o método: por volta de 20. Isso faz com que a memória requerida além do que já é utilizado para guardar a diagonal da estrutura tensorial e os vetores de probabilidade do tamanho do PSS, ser negligenciável. O ganho verificado em frente ao Algoritmo *Shuffle* foi da ordem de nove vezes, para a maioria dos casos e sete vezes para os casos onde os termos foram reordenados. Este modelo está definido apenas para descritores constantes.

Os próximos modelos são oriundos do formalismo definido por PEPA, onde foi modelado dois tipos de modelos distintos: células industriais de produção e um servidor para a Internet. Estes modelos foram traduzidos para SAN, com o objetivo de estudar as características envolvidas na tradução entre estes dois formalismos e as propriedades a serem consideradas neste mapeamento.

5.2.4 Modelos do formalismo PEPA: Workcell e WebServer

Para demonstrar a execução de modelos definidos em outros formalismos, foram escolhidos dois modelos em particular, gerados a partir de PEPA. O primeiro, chamado *Workcell*, modela células industriais de produção e o segundo define um servidor para Internet de alta disponibilidade, chamado *WebServer*. Aqui serão feitas breves explicações sobre o funcionamento geral dos sistemas modelados, maiores informações podem ser encontradas em [49, 12] para, respectivamente, *Workcell* e *WebServer*. Estes modelos foram traduzidos para SAN de forma

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
fas18c	262.144	17	≈0	$\approx 0.0000 \pm 0.0000$	**0.0795 ± 0.0001**	0.7029 ± 0.0043	8.8415
fas19c	524.288	18	≈0	$\approx 0.0000 \pm 0.0000$	**0.1726 ± 0.0003**	1.5550 ± 0.0012	9.0093
fas20c	1.048.576	19	≈0	$\approx 0.0000 \pm 0.0000$	**0.3660 ± 0.0026**	3.4416 ± 0.0087	9.4033
					Experimento 3	Experimento 4	
fas18c	262.144	17	≈0	$\approx 0.0000 \pm 0.0000$	**0.0796 ± 0.0001**	0.5538 ± 0.0008	6.9573
fas19c	524.288	18	≈0	$\approx 0.0000 \pm 0.0000$	**0.1725 ± 0.0001**	1.2254 ± 0.0014	7.1038
fas20c	1.048.576	19	≈0	$\approx 0.0000 \pm 0.0000$	**0.3616 ± 0.0004**	2.8037 ± 0.0181	7.7536

Tabela 5.6: Caso *FAS*, modelos constantes, comparando Experimentos $\{1, 2, 3, 4\}$.

direta, gerada a partir da composicionalidade de PEPA observando-se as movimentações nos estados da CTMC. Esta tradução não é a melhor tradução deste tipo de modelagem, mas mesmo assim, produz os resultados de saída equivalentes à solução PEPA.

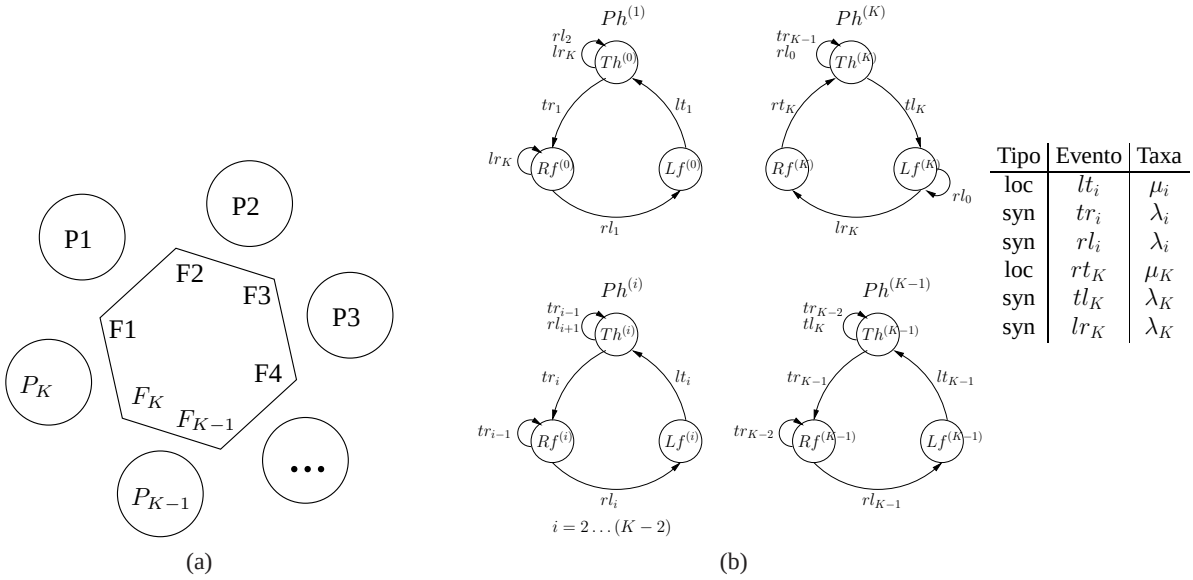
O modelo *Workcell* modela um sistema complexo de uma célula industrial de produção. O objetivo desta célula é construir placas de metais em uma prensa, consistindo dos seguintes componentes principais: cintos de alimentação, tabelas rotatórias, robôs, prensas, cintos de depósitos e artefatos de movimentação, para citar alguns componentes do sistema. O principal objetivo desta modelagem é inferir índices que demonstrem como estes componentes trabalham em conjunto, com vistas à avaliação de desempenho destes componentes.

Já o modelo *WebServer* modela um servidor para Internet de alta disponibilidade, composta principalmente por elementos de notícias. Este modelo prevê qualidade de serviço em disponibilidade e tempo de resposta. O objetivo principal deste tipo de modelagem é experimentar com a engenharia de desempenho de aplicações de alta demanda com vistas à verificação de eficiência de pico, entre outras análises. O modelo executado aqui, possui quatro parâmetros: S compreende o número de servidores no sistema, B, denota a capacidade do *buffer* de memória, R diz respeito ao número de leitores e W ao número de escritores existentes no sistema (como trata-se de um sistema de notícias, as atividades compreendem a leitura e a escrita de materiais). Para o caso deste exemplo em particular, foi escolhido trabalhar com um modelo com um PSS de valor razoável, com os seguintes parâmetros: $S = 4$, $B = 3$, $R = 3$ e $W = 3$.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
pepa_workcell	320.112	16	≈0	$\approx 0.0000 \pm 0.0000$	**0.0357 ± 0.0001**	0.2410 ± 0.0001	6.7507
pepa_webserver	349.920	63	≈1	$\approx 0.0000 \pm 0.0000$	**0.1087 ± 0.0002**	1.0010 ± 0.0012	9.2088
					Experimento 3	Experimento 4	
pepa_workcell	320.112	11.662	364	0.0162 ± 0.0014	0.0385 ± 0.0001	0.1707 ± 0.0001	4.4338
pepa_webserver	349.920	574.453	17.951	1.4496 ± 0.0046	0.2442 ± 0.0006	0.7678 ± 0.0008	3.1441

Tabela 5.7: Casos *Workcell* e *WebServer*, modelos constantes, comparando Experimentos $\{1, 2, 3, 4\}$.

Estes modelos baseados em PEPA existem apenas sob a forma de um descritor clássico e os seus resultados estão sendo mostrados na Tabela 5.7. Esta tabela informa o tempo de execução para os Experimentos $1, 2, 3$ e 4. Para o modelo *workcell*, o tempo verificado é bastante similar nos Experimentos 1 e 3 e, quando comparados ao Experimento 2 e 4 atestam um ganho de seis e quatro vezes. A diferença entre as execuções está na memória gasta em cada experimento. Enquanto que no primeiro nada foi praticamente gasto (por volta de $zero$ bytes), no



Tipo	Evento	Taxa
loc	lt_i	μ_i
syn	tr_i	λ_i
syn	rl_i	λ_i
loc	rt_K	μ_K
syn	tl_K	λ_K
syn	lr_K	λ_K

Figura 5.5: Uma configuração possível de filósofos em uma mesa em (a) e Modelo SAN em (b) para K filósofos.

Experimento 3 foi necessário 364 Kb. Para este modelo, a memória gasta não impacta no tempo de execução.

Para o caso do *WebServer*, a melhor execução foi a alcançada pelo Experimento 1, sendo nove vezes mais rápido quando comparado ao Algoritmo *Shuffle* realizando-se permutações. No caso de se escolher não fazer permutações (Experimentos 3 e 4), o ganho ainda assim é considerável (da ordem de três vezes), mas são necessários 18 Mb para salvar os AUNFs. Visto que é possível reordenar os termos e não gastar em memória e ainda assim ganhar em tempo, esta é claramente a melhor alternativa para a solução eficiente deste modelo. Cabe ressaltar ainda que a estrutura deste modelo em termos de eventos sincronizantes é similar à verificada pelo modelo das *Redes de Sensores*, onde os eventos sincronizantes envolvem apenas dois autômatos, na maioria dos termos tensoriais.

5.2.5 Modelos dos Filósofos

Esta seção apresenta um modelo clássico para avaliação de desempenho para análise de exclusão mútua em compartilhamento de recursos. O modelo em questão é uma abstração chamada de *problema dos filósofos* e, resumidamente, pode ser descrita da seguinte forma: K filósofos estão sentados em uma mesa e a eles somente são permitidas duas ações, comer ou pensar. Os filósofos estão sentados em uma mesa circular com uma tijela de comida no centro desta (de acordo com a Figura 5.5a). Um garfo F_k está posicionado entre cada filósofo P_k, assim, cada um possui um garfo à sua esquerda e um garfo à sua direita. Para comer, um filósofo necessita de dois garfos nas suas mãos, simultaneamente (o problema é que não existem recursos suficientes para todos ao mesmo tempo).

O modelo SAN mostrado na Figura 5.5b possui K autômatos do tipo $P^{(k)}$ representando os filósofos, cada um com três estados: $Th^{(k)}$ (pensando), $Lf^{(k)}$ (pegando garfo da esquerda), $Rf^{(k)}$ (pegando o garfo situado à direita). O filósofo pode reservar o garfo à sua esquerda ou direita para comer utilizando dois garfos disponíveis. Para evitar *deadlock* fica estabelecida uma ordem para pegar os garfos na mesa, para cada filósofo presente no modelo. O PSS deste modelo é dado por 3^K estados.

Os resultados para o modelos dos *Filósofos* está mostrada na Tabela 5.8. Os modelos definidos só existem para descritores clássicos, mostrando os resultados para a faixa de 12 a 16 filósofos. Observa-se que o PSS tem um crescimento exponencial, sendo que para 16 filósofos, ele é da ordem de 43 milhões de estados. Não foi possível computar para este modelo os tempos dos Experimentos 3 e 4 uma vez que foi preciso por volta de 1.2

Gb de memória apenas para armazenar a tabela referente aos AUNFs. Como este modelo é possuidor de muitas identidades, o Experimento 1 mostra claramente o efeito de se passar estes elementos para a parte estruturada: apenas 48 AUNFs são de fato necessários para o caso com mais filósofos, ou seja, o com 16. E esse fato reflete no tempo gasto para executar uma iteração, onde os ganhos médios calculados ficaram na ordem de quatro vezes, entre os Experimentos 1 e 2, que executaram permutações nos termos tensoriais.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
phil12c	531.441	36	≈1	$\approx 0.0000 \pm 0.0000$	**0.1995 ± 0.0007**	0.7319 ± 0.0019	3.6687
phil13c	1.594.323	39	≈1	$\approx 0.0000 \pm 0.0000$	**0.6270 ± 0.0008**	2.3617 ± 0.0009	3.7667
phil14c	4.782.969	42	≈1	$\approx 0.0000 \pm 0.0000$	**1.9989 ± 0.0016**	7.6590 ± 0.0018	3.8316
phil15c	14.348.907	45	≈1	$\approx 0.0000 \pm 0.0000$	**6.2543 ± 0.0318**	24.2281 ± 0.0607	3.8738
phil16c	43.046.721	48	≈1	$\approx 0.0000 \pm 0.0000$	**19.7111 ± 0.0119**	77.4296 ± 0.0279	3.9282

					Experimento 3	Experimento 4	
phil12c	531.441	501.914	15.684	1.2862 ± 0.0032	0.3146 ± 0.0006	0.4960 ± 0.0005	1.5766
phil13c	1.594.323	1.505.747	47.054	4.0062 ± 0.0119	0.9798 ± 0.0020	1.6074 ± 0.0016	1.6405
phil14c	4.782.969	4.517.246	141.163	12.6720 ± 0.0311	3.0583 ± 0.0037	5.1728 ± 0.0047	1.6914
phil15c	14.348.907	13.551.743	423.491	39.7944 ± 0.0946	9.8197 ± 0.0102	16.6351 ± 0.0203	1.6941
phil16c	43.046.721	40.655.234	1.270.476	≈ 136.1200	–	–	–

Tabela 5.8: Caso *Filósofos*, modelos constantes, comparando Experimentos $\{1, 2, 3, 4\}$.

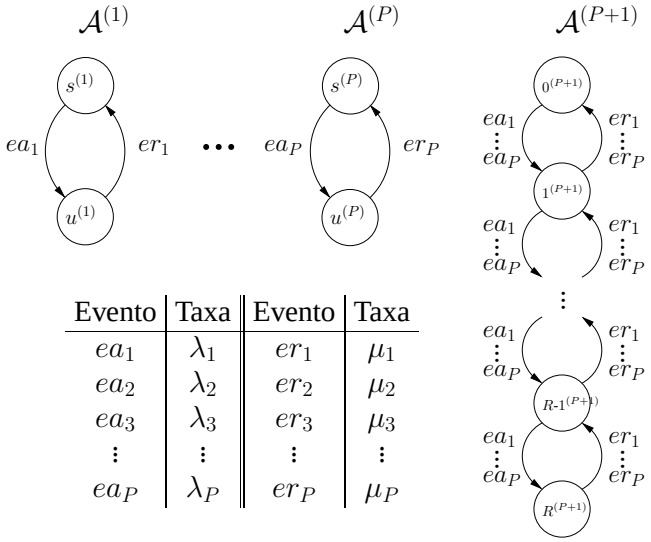
A Tabela 5.8 mostra a influência dos aspectos semânticos na geração dos termos tensoriais, pois, ao aumentar o número de filósofos em um determinado modelo, implica no aumento proporcional de sincronizações que ocorrem (e, consequentemente, de termos). As variações do número de filósofos do modelo foram chamadas de *phil*K, onde K representa o número de filósofos existentes. Este modelo evidenciou o fato que é totalmente desvantajoso salvar AUNFs que correspondem a identidades, ou seja, valores que não serão de forma alguma alterados, pois possuem o valor 1 e implicam em um aumento substancial de elementos que é totalmente desnecessário que seja calculado. Para o caso de 15 filósofos, por exemplo, foi necessário 423 Mb de memória e o método executou em 9.8 segundos, enquanto que gastando-se por volta de 1 Kb, foi executado em 6.2 segundos.

O modelo dos filósofos, assim como o modelo das *Redes de Sensores* mostram quando a abordagem utilizada pelo Algoritmo *Split* realmente ganha em termos de tempo de execução. O motivo deste acontecimento é relacionado ao fato de que este modelo em particular possui sincronizações entre poucos autômatos e diversas matrizes identidade. O corte realizado é extremamente baixo e isso reflete o número de AUNFs necessários para o cálculo. Também mostrou que é completamente ineficiente usar matrizes identidade para compor a listagem dos AUNFs, visto que elas não influenciam no valor que está sendo salvo e poderiam servir melhor na parte estruturada, que salta fatores normais identidade, ou seja, o Algoritmo *Shuffle* não é chamado e logo, nunca calcula índices de salto na estrutura tensorial.

5.2.6 Modelo Compartilhamento de Recursos

O modelo dos Compartilhamentos de Recursos é outro modelo clássico de avaliação de desempenho, tratando sobre como N processos compartilham R recursos. A Figura 5.6 representa um sistema de compartilhamento de recursos onde cada processo é representado por um autômato $A^{(i)}$ composto por dois estados: $S^{(i)}$ (dormindo, ou *sleeping*) e $U^{(i)}$ (usando ou *using*). Um conjunto de recursos é representado pelo autômato $A^{(N+1)}$ o qual possui $R + 1$ estados indicando o número de recursos que estão sendo utilizados.

Este modelo apresenta um conjunto de autômatos no qual os eventos sincronizantes estão relacionados a apenas dois autômatos cada um. Isto significa que os termos tensoriais serão compostos por muitas matrizes de baixa



Evento	Taxa	Evento	Taxa
ea_1	λ_1	er_1	μ_1
ea_2	λ_2	er_2	μ_2
ea_3	λ_3	er_3	μ_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ea_P	λ_P	er_P	μ_P

Figura 5.6: Modelo clássico de Compartilhamento de Recursos através de SAN.

esparsidade e igualmente muitas identidades.

A característica principal deste modelo é não possuir eventos locais, logo, a parte local do descritor não gerará fatores normais, ou seja, termos onde todas as matrizes menos uma são do tipo identidade. O restante dos termos disponíveis (sincronizantes) são particularmente interessantes para o Algoritmo *Split*, já que a abordagem utilizada pelo *Shuffle* já otimiza de forma suficientemente considerável as somas tensoriais.

Os principais resultados para o modelo do *Compartilhamento de Recursos* está mostrado na Tabela 5.9. A tabela mostra as diferentes configurações de redes de autômatos, denominadas *rsP_R* que indicam no nome do modelo o número de processos existentes (P) para R recursos compartilhados. Verifica-se as comparações entre os tempos obtidos de *uma* iteração entre os Experimentos $1, 2$ e $3, 4$. Inicialmente, verifica-se que para o caso onde existem 20 processos para 25 recursos (o nome do modelo é $rs20\_25c$), não foi possível computar os tempos devido ao alto custo em memória para salvar os AUNFs, da ordem de 1.6 Gb. Esta classe de modelos também possui muitas identidades, e o experimento que obteve os maiores ganhos foi o Experimento 1, sendo melhor em média três vezes que o Experimento 2, sendo necessário 31 Kb de memória para guardar os AUNFs.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
rs14_10c	180.224	280	8	$\approx 0.0000 \pm 0.0000$	**0.1067 ± 0.0001**	0.3127 ± 0.0005	2.9306
rs14_11c	196.608	308	9	$\approx 0.0000 \pm 0.0000$	**0.1173 ± 0.0001**	0.3409 ± 0.0008	2.9062
rs15_15c	524.288	450	14	$\approx 0.0000 \pm 0.0000$	**0.3448 ± 0.0007**	1.0169 ± 0.0012	2.9492
rs20_25c	27.262.976	1000	31	$\approx 0.0000 \pm 0.0000$	**24.7120 ± 0.0106**	88.0908 ± 0.0383	3.5647

					Experimento 3	Experimento 4	
rs14_10c	180.224	327.660	10.239	0.9914 ± 0.0026	0.2019 ± 0.0003	0.2093 ± 0.0002	1.0367
rs14_11c	196.608	360.426	11.263	1.0720 ± 0.0028	0.2226 ± 0.0004	0.2306 ± 0.0003	1.0359
rs15_15c	524.288	983.010	30.719	2.9890 ± 0.0263	0.6448 ± 0.0140	0.6812 ± 0.0008	1.0565
rs20_25c	27.262.976	52.428.750	1.638.398	≈ 234.9400	–	–	–

Tabela 5.9: Caso *Compartilhamento de Recursos*, modelos constantes, comparando Experimentos $\{1, 2, 3, 4\}$.

Esta classe de modelos, como no caso dos filósofos, mostra claramente que a realização de permutações é amplamente necessária, e este fato está evidente na memória gasta em todos os modelos. Nos casos onde permutaram-se os termos tensoriais, a memória gasta é muito inferior aos experimentos onde não foram usados reordenamentos (para o caso específico do Algoritmo *Split*, uma vez que o Algoritmo *Shuffle* é eficiente em gastos de memória).

5.2.7 Modelo *Alternate Service Patterns*, ou ASP

Este exemplo descreve um sistema de rede aberta composto por quatro filas (F_1, F_2, F_3 e F_4), representadas pelos autômatos $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}, \mathcal{A}^{(4)}$, com capacidades finitas (K_1, K_2, K_3 e K_4) respectivamente. O padrão de roteamento de consumidores para esse caso, chegam em F_1 e F_2 com taxa λ_1 e λ_2, podendo sair de Q_1 para Q_3 se existir espaço (comportamento bloqueante) e também sair de F_2 para F_3 se existir espaço ou sair do modelo em caso contrário (comportamento de perda). Os consumidores também podem sair de F_3 para F_4 com comportamento bloqueante.

Enquanto F_1, F_2 e F_4 possuem comportamento de serviço padrão (no caso, *single*), *i.e.*, uma mesma taxa média de serviço para todos os consumidores (μ_1, μ_2 e μ_4 respectivamente), a fila F_3 possui um comportamento alternado de padrão de serviço (*Alternate Service Pattern*, ou ASP). A taxa de serviço para esta fila varia de acordo com P diferentes taxas de padrão de serviço (μ_{31}, ..., μ_{3P}). A fila F_3 pode trocar seu padrão simultaneamente de acordo com o final do serviço de um consumidor. Isso faz com que quando um consumidor é servido pelo padrão P_i, F_3 pode permanecer servindo o próximo consumidor no mesmo padrão com probabilidade π_{ii}, ou pode alternar para um padrão de serviço diferente P_j com probabilidade π_{ij} (para todos os padrões de serviço $P_i : \sum_{j=1}^{P} \pi_{ij} = 1$).

O modelo SAN descrito na Figura 5.7 é composto por um autômato usado para cada fila com serviço simples ($\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$ e $\mathcal{A}^{(4)}$) e outros dois autômatos para a fila de padrão de serviço alternado ($\mathcal{A}^{(3)}$ e $\mathcal{A}^{(5)}$).

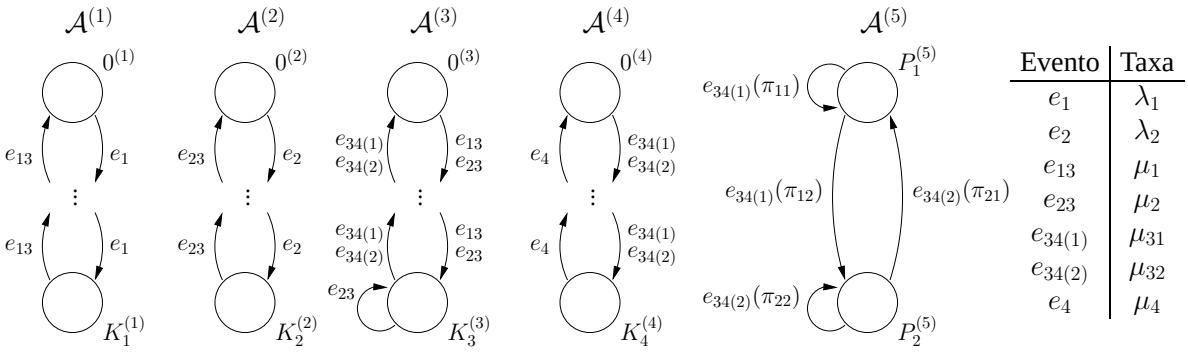


Figura 5.7: Modelo *Alternate Service Pattern* com descritor constante.

O modelo ASP mostra que o melhor tempo encontrado foi correspondente ao Experimento 4, conforme a Tabela 5.10. Este experimento obteve os melhores índices para uma iteração, quando não são executadas permutações nos termos tensoriais. A tabela mostra os casos com descritores constantes onde os tempos, apesar de serem parecidos entre os Experimentos 1 e 4, mostram um ganho relativo implicando que a melhor abordagem a ser escolhida nestes casos, é provavelmente a adotada pelo Algoritmo *Shuffle*. Cabe ressaltar que a evolução dos termos tensoriais ocorre de forma escalar, ou seja, para N servidores sempre existem cinco autômatos, $(N + 2)$ eventos sincronizantes e 3 locais, totalizando $(2(N + 2) + 3)$ eventos. Por volta da metade de cada termo tensorial é composto por matrizes do tipo identidade.

Este modelo encerra a seção dos resultados obtidos. A seguir é feita uma discussão sobre os experimentos e os resultados obtidos.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem (Kb)	Cálculo (s)	Experimento 1	Experimento 2	
asp10c	252.890	10.298	321	0.0142 ± 0.0014	0.1506 ± 0.0005	0.1911 ± 0.0007	1.2689
asp12c	342.732	21.036	657	0.0398 ± 0.0004	0.2478 ± 0.0001	0.2991 ± 0.0006	1.2070
asp14c	399.854	28.524	891	0.0518 ± 0.0011	0.3309 ± 0.0032	0.4013 ± 0.0013	1.2128

					Experimento 3	**Experimento 4**	
asp10c	252.890	2.094.078	65.439	5.1660 ± 0.0670	0.5146 ± 0.0015	**0.1490 ± 0.0006**	0.2895
asp12c	342.732	3.508.284	109.633	9.0288 ± 0.0488	0.8481 ± 0.0029	**0.2331 ± 0.0008**	0.2748
asp14c	399.854	4.473.756	149.179	12.6500 ± 0.0829	1.1451 ± 0.0049	**0.3130 ± 0.0013**	0.2333

Tabela 5.10: Caso *ASP*, modelos constantes, comparando Experimentos $\{1, 2, 3, 4\}$.

5.3 Discussão

Os resultados obtidos mostraram claramente quando o Algoritmo *Split* é melhor utilizado, ou seja, quando existem termos tensoriais esparsos, no melhor caso, com apenas um ou poucos elementos não-nulos, termos com alta incidência de matrizes identidade (o evento não ocorre em diversos autômatos), efetua-se permutações na ordem original e, para descritores generalizados, as avaliações de elementos funcionais são efetuadas quando os AUNFs são calculados, apenas uma vez no início do método. O melhor caso pode ser comprovado através do caso das *Redes de Sensores*, onde foi preciso salvar N escalares na tabela de AUNFs, correspondendo ao número de termos existentes, ou seja, um AUNF por termo tensorial. Os ganhos para o caso constante foram da ordem de 12 vezes em relação à abordagem do Algoritmo *Shuffle*.

É inegável que a utilização de funções está intrinsicamente relacionada à modelagem de interações complexas em sistemas, mas como métodos iterativos de solução são utilizados nesse escopo, não justifica a avaliação frequente de elementos funcionais a cada execução do método de MVD. Estes elementos funcionais sempre retornarão os mesmos valores, logo, seria mais eficiente utilizar estruturas de dados auxiliares para guardar apenas a primeira avaliação das funções, ou seja, que convertam os descritores generalizados em constantes em tempo de execução. Isso faria com que os usuários continuassem a modelar sistemas de forma a capturar certos detalhes impossíveis de serem mapeados com eventos locais e sincronizantes, ao mesmo tempo que iria otimizar a convergência da solução no método iterativo. Este é o mesmo argumento pelo qual não se guardam os AUNFs que são gerados a cada execução para cada termo tensorial, pois serão sempre os mesmos, sem mudança alguma. Não é otimizado ter que se recalcular essa tabela a cada iteração uma vez que os ganhos em termos de desempenho são consideráveis para esses casos.

Mas para entender o processo interno do algoritmo na sua essência, é necessário observar os detalhes do descritor markoviano em si. Uma vez que os eventos locais são somas simples de matrizes, são os eventos sincronizantes, com taxas constantes ou funcionais, que determinam o comportamento do Algoritmo *Split* em termos de custos de memória e tempo de solução. Como definido anteriormente, quando um dado evento sincronizante não ocorre em um dado autômato, no nível do descritor, a sua matriz correspondente é uma matriz do tipo identidade. Sendo uma matriz identidade, no Algoritmo *Shuffle*, ela é negligenciada, mas ainda assim conta para o cálculo de $nleft$ e de $nright$ do método (em termos de índices) para todas as matrizes que não forem deste tipo. A alternativa proposta pelo Algoritmo *Split* é permutar os termos tensoriais, utilizar o Algoritmo Esparso para escolher elementos das matrizes e calcular os índices onde devem ser multiplicadas nos vetores de entrada e de saída e determinar um ponto de corte onde nada mais é necessário para executar o método de solução.

Cabe ressaltar ainda que o Algoritmo *Split*, quando posiciona todas as identidades para a parte estruturada e trabalha apenas com os AUNFs que correspondem as taxas dos eventos sincronizantes, não são gerados fatores normais e, ocorrendo isso, não é necessário que os vetores auxiliares sejam passados para o resto dos fatores

normais, como ocorre na abordagem *Shuffle*. Este fato é uma limitação quando deseja-se resolver modelos com paralelização do método, o que não ocorre no *Split*, que precisa apenas dos AUNFs que são gerados para montar seus vetores auxiliares.

O custo gasto para reordenar o termo tensorial, calcular alguns índices auxiliares e utilizar esses índices ao longo do método de MVD é negligenciável ao utilizar o Algoritmo *Split*, como constatado em todos os experimentos da Seção 5.2. Verificou-se que a maioria dos modelos considerados executaram mais rápido que o Algoritmo *Shuffle* com ou sem permutação, onde a tabela com os AUNFs ficou dentro do limite aceitável de tolerância.

Com o uso dos reordenamentos, constatou-se que a memória requerida sempre ficou dentro de um patamar aceitável, sendo o único efeito limitador o tamanho do vetor estacionário de probabilidade. Entretanto, o mesmo ocorre com o Algoritmo *Shuffle*. Dada a relativa memória utilizada para salvar os AUNFs requerida inclusive quando o PSS é elevado, pode-se afirmar que o método somente não executará para os mesmos limites que são impostos ao *Shuffle*, estimado atualmente em 65 milhões de estados. Se a memória for realmente uma preocupação, basta executar o Algoritmo *Split* cortando o termo tensorial antes da primeira matrix, ou seja, nesse caso, adotar a prática do Algoritmo *Shuffle*.

Um aspecto que não foi estudado até o presente momento é o efeito das dimensões das matrizes (principalmente quando não correspondem à identidades) na solução do modelo. Esta variável pode ser melhor estudada no modelo *Mestre-Escravo*, que possui um autômato modelado como um *Buffer*, ou seja, uma região temporária de memória em uma determinada problemática (este tipo de modelagem é bastante utilizada em sistemas paralelos e distribuídos, sendo útil para verificar quando estas estruturas chegam aos seus limites ou quando estão subutilizadas). Esses estudos podem alterar significativamente o que se sabe sobre o Algoritmo *Split* pois, na parte estruturada, na parte da geração dos fatores normais, um grande bloco será descartado quando a matriz em questão possuir uma grande dimensão.

Maiores estudos nesse sentido devem ser conduzidos para um maior entendimento sobre o comportamento do método quando existem matrizes com elevadas dimensões. Por exemplo, o modelo *Mestre-Escravo* pode redefinir o tamanho do *buffer* para possuir uma faixa de valores que testem o efeito das dimensões no método, com o cuidado de não atingir os limites de memória, visto que este exemplo, para 10 Mestres, o PSS é de 43 milhões.

6 Proposta

Como mencionado ao longo deste trabalho o Algoritmo *Split* introduziu o conceito de que é possível gastar um pouco de memória, fugindo da visão de eficiência em memória do Algoritmo *Shuffle*, com vistas à obtenção de resultados numéricos em um menor tempo de execução. A forma escolhida de otimização fundamentou-se na pesquisa de melhorias específicas ao método de MVD. Isso se deu pois, ao melhorar o tempo de execução de uma iteração é evidente que ao usar-se o método iterativo de solução de sistemas esse ganho seria propagado e o desempenho final seria considerável.

Entretanto, os aspectos teóricos que foram desenvolvidos no Algoritmo *Split* não demonstraram claramente as melhores formas de divisão de tanto descritores markovianos constantes como generalizados. Trata-se de um tópico aberto de pesquisa especificamente em SAN, onde faz-se necessário conhecer as diferentes características dos termos tensoriais para compor a melhor forma de dividir as matrizes dos termos e otimizar a forma atual de MVD. O objetivo deste capítulo é descrever um algoritmo inicial para determinação desta divisão e listar as atividades que faltam para o término da tese.

6.1 Determinação do ponto de corte σ para o Algoritmo *Split*

Os resultados obtidos forneceram uma ideia razoável de quando é melhor usar as diferentes abordagens propostas. Os exemplos utilizados mostraram as vantagens e as características dos termos tensoriais que devem ser observadas para determinar onde melhor dividí-los e reposicionar as matrizes de forma a obter um tempo otimizado. A seguir, é discutida uma proposta para a escolha deste ponto de corte, observando a memória que será potencialmente gasta, refletindo no tempo de solução.

As equações teóricas do custo computacional para solução de termos tensoriais, aliado aos resultados obtidos na Seção 5.1 permitem que seja proposto um algoritmo para divisão de descritores clássicos e generalizados. Este algoritmo leva em consideração o efeito do custo em termos de operações de ponto flutuante requeridas pelo Algoritmo *Split* bem como a memória na construção dos AUNFs. Um número de AUNFs igual a zero indica que o corte escolhido é o referente ao Algoritmo *Shuffle* com mesmo custo computacional.

Antes da introdução do algoritmo, alguns fatos conhecidos sobre a solução de termos tensoriais que foram verificados com os resultados produzidos e a análise teórica:

1. avaliar funções apenas uma vez é melhor do que avalia-las muitas vezes por muitas iterações;
2. matrizes constantes e matrizes elemento sempre podem ser deslocadas para a parte esparsa;
3. uma esparsidade baixa do termo tensorial (o número de elementos não-nulos existentes) implica em um número de AUNFs igualmente baixo;
4. matrizes identidade podem sempre ser deslocadas para a parte estruturada, pois na parte esparsa, geram um número de AUNFs maior que antes desta reorganização e não alteram o valor, pois são iguais a um;
5. termos altamentes dependentes são melhores tratados pelo Algoritmo *Shuffle*;
6. para o Algoritmo *Split* o melhor caso é quando os modelos possuem diversos autômatos que sincronizam suas atividades com poucos eventos (isso faz com que existam muitas identidades no termo tensorial);

O Algoritmo 6.1 recebe como entrada um termo tensorial e decide o ponto de corte a ser utilizado a partir das características das matrizes (dimensões, identidades, elementos não-nulos) que o compõe e o tipo de termo. Este algoritmo é uma delineação do que será utilizado para a otimização da MVD nos descritores constantes e generalizados. O funcionamento do algoritmo é o seguinte: a partir da descoberta do tipo do termo tensorial τ (que pode ser uma de três possibilidades, constante, parcialmente dependente ou totalmente dependente), é realizado o processo de descoberta do ponto de corte σ. Caso este tipo seja constante, ocorre uma transformação no termo tensorial enviando as matrizes identidade para o final e armazenando-se esse índice na variável i. Caso haja memória suficiente para essa operação, a transformação é validada e σ recebe o valor do índice i. Já no caso do termo ser parcialmente dependente, tem-se a verificação da ocorrência de matrizes elemento (ou constantes) e o envio de matrizes para a parte esparsa do termo tensorial, exceto identidades. O valor de σ para esse caso é o índice da última matriz dependente depois desta transformação. Por fim, se o tipo for altamente dependente, a melhor possibilidade para lidar com esse termo é adotar a abordagem totalmente estruturada, pois inclusive pode-se ter dependências cíclicas e funções especificamente que serão tratadas de forma adequada pelo Algoritmo *Shuffle*.

Algorithm 6.1 Algoritmo para determinação do ponto de corte σ do termo tensorial τ

1: $M \leftarrow$ {memória disponível}
2: $A \leftarrow$ {memória da tabela de AUNFs, assume-se que já foram calculados previamente}
3: $tipo \leftarrow T(\tau)$ {descobre o tipo do termo tensorial τ}
4: **if** $tipo =$ constante **then**
5: $i \leftarrow transforma(\tau)$ {envia as identidades para o final, guarda índice da última matriz constante}
6: $A \leftarrow \mathcal{E}(i, \tau)$ {a função $\mathcal{E}$ descobre o total de AUNFs para i e τ}
7: **if** $A < M$ **then** {se existe memória suficiente para a operação}
8: $\sigma \leftarrow i$ {o ponto de corte sigma recebe o índice i}
9: **else**
10: {tenta encontrar um outro ponto de corte utilizando menos memória (mais estruturação)}
11: **end if**
12: **else if** $tipo =$parcialmente dependente **then**
13: $c \leftarrow constante(\tau)$ {verifica a existência de matrizes constantes}
14: $i \leftarrow transforma(\tau, c)$ {envia as identidades para o final, a partir do índice da última matriz constante}
15: $A \leftarrow \mathcal{E}(i, \tau)$ {a função $\mathcal{E}$ descobre o total de AUNFs para i e τ}
16: **if** $A < M$ **then**
17: $j \leftarrow depend(\tau)$ {descobre o índice da última matriz dependente}
18: $\sigma \leftarrow c + j$ determina o ponto de corte a partir da última matriz constante + última dependente
19: **else**
20: {tenta encontrar um outro ponto de corte utilizando menos memória (mais estruturação)}
21: **end if**
22: **else if** $tipo =$ totalmente dependente **then**
23: $u \leftarrow ultima(\tau)$ {a função *ultima* descobre o índice da última matriz do termo}
24: $A \leftarrow \mathcal{E}(u, \tau)$ {a função $\mathcal{E}$ descobre o total de AUNFs para i e τ}
25: **if** $A < M$ **then** {considerar a chamada de método esparso puro}
26: $\sigma \leftarrow u$
27: **else** {utiliza a abordagem puramente estruturada}
28: $\sigma \leftarrow 0$
29: **end if**
30: **end if**
31: $retorna\ \sigma$ {retorna o valor de σ para τ calculado}

Ainda falta considerar no algoritmo o uma verificação nas funções que possuem poucos parâmetros ou são melhores tratadas na parte estruturada. Isso fará com que menos memória seja gasta, uma vez que quanto mais estruturada é a escolha da divisão, menos memória é gasta para efetuar-se a MVD. Para esse caso ainda são necessários maiores estudos quanto à como devem ser as funções dos termos tensoriais generalizados para que exista um ganho real na multiplicações que são efetuadas.

Termos com matrizes densas farão com que sejam calculados um número de AUNFs que talvez inviabilize o método e a melhor abordagem a seguir é dividir o termo em um ponto onde a memória a ser gasta é suficientemente calculada para abrigar um número considerável de elementos. Em casos altamente dependentes em termos de funções, a melhor alternativa é cortar em σ_0 e resolver o termo de forma estruturada. Em casos onde existem funções que dependem de um número restrito de outros autômatos, o Algoritmo *Split* possui um bom desempenho, pois ocorrem diversas identidades no termo e esparsidade dita que serão criados um número de AUNFs que é menor que a memória disponível.

A experiência em modelagem SAN dita que são poucos os casos onde existem matrizes plenas ou altamente densas em um termo tensorial. Os termos tensoriais possuem um padrão bem definido e o Algoritmo *Split* utiliza esse padrão de definição dos descritores para resolver os modelos em menos tempo, visto que o método é executado mais rápido, por iteração, que as abordagens pré-existentes no que tange a MVD especificamente para SAN.

6.2 Mapeamento de formalismos estruturados para formato tensorial

Um dos objetivos de se realizar as otimizações em MVD é permitir que outros formalismos sejam diretamente beneficiados com as pesquisas que existem. Uma maneira de se realizar essa tarefa não trivial é definir formalmente as regras de tradução e mapeamento a serem aplicadas nos modelos para que estes possuam um formato tensorial que possa ser utilizado pelos métodos de MVD existentes.

Para realizar esta tarefa, maiores estudos são necessários no que diz respeito a esse mapeamento, especificando as principais regras para transformar os modelos em uma representação estruturada que possibilite a definição em um descritor markoviano, com vistas à extração de índices de desempenho de sistemas. Sabe-se que existem uma série de detalhes que implicam nessa tradução e estes mecanismos devem ser melhor explorados para a conclusão desta tese.

A seguir são mostradas as ideias iniciais quanto as principais questões envolvidas nessa transformação:

- trabalhos relacionados: sabe-se que já existem trabalhos onde esse mapeamento foi executado com sucesso, por exemplo, descritores tensoriais para Redes de Petri [28, 29], Redes de Petri Generalizadas Superpostas (*Superposed Generalized Stochastic Petri Nets* - SGSPNs) [29, 58], uma proposta de mapeamento de PEPA para SAN [47] e uma representação *Kronecker* para esses modelos [46]. Cabe ressaltar que estes trabalhos relacionados servirão como base para a descrição das regras já existentes;

- realizar um exemplo de tradução válido, mostrando-se os ganhos da realização destas tarefas. Estes trabalhos já foram iniciados na problemática de PEPA nets;

- estudos sobre a estruturação de modelos e como modelar as realidades e descobrir os mapeamentos existentes entre os diferentes formalismos;

O objetivo deste mapeamento não é reduzir o uso dos diferentes formalismos e propor uma generalização em torno de um único formato. Cada formalismo possui uma forma de representação única e serve a objetivos distintos. A contribuição está relacionada à, dada a existência de um formato tensorial em um descritor markoviano, aplicar os métodos otimizados de MVD presentes e discutidos nesse trabalho para a extração de índices de desempenho de sistemas. Estes estudos e aplicações serão utilizados na elaboração de um capítulo que versará sobre as formas de se compor os descritores tensoriais para que estes sejam utilizados na MVD de forma otimizada. A seguir, mostra-se o cronograma de atividades.

6.3 Cronograma de Atividades

Para dar seguimento à essa proposta de tese, é necessário relacionar as atividades já concluídas e das atividades faltantes, bem como um cronograma a ser seguido. A seguir, uma descrição completa destas tarefas, seguidas de comentários, discussões e avaliações.

6.3.1 Atividades Concluídas

As atividades concluídas até o momento mostram que a parte de implementação está praticamente concluída bem como a análise dos resultados obtidos a partir da execução dos métodos propostos em uma série de experimentos. A seguir a lista das tarefas que foram concluídas com sucesso até o presente momento:

1. Implementação do Algoritmo *Split* baseado em descritores ATC e ATG na ferramenta PEPS;
2. Planificação dos testes e experimentos a serem realizados e escolha de modelos relevantes para análise;
3. Execução de testes e validação estatística a partir da análise dos intervalos de confiança dos resultados obtidos;
4. Início da escrita do volume final da tese;
5. Implementação de um protótipo para validação da tradução utilizando PEPA nets.

6.3.2 Atividades Faltantes

Entretanto, ainda faltam algumas tarefas o término deste trabalho. A Tabela 6.1 mostra um resumo das atividades que faltam serem concluídas até a defesa da tese de doutorado.

	Mar	Apr	Mai	Jun	Jul	Aug	Sep	Oct
1.	√							
2.		√						
3.		√	√					
4.	√	√	√	√				
5.		√	√	√	√	√	√	
6.							√	
7.								√

Tabela 6.1: Atividades até a defesa da tese de doutorado.

A seguir, um maior detalhamento das atividades a serem concluídas:

1. Aplicação das considerações realizadas na defesa da proposta de tese; revisões no código-fonte, refinamentos e testes para otimizações;
2. Documentação das implementações feitas na ferramenta PEPS; implementação de opções para os usuários na ferramenta, para utilização dos conceitos desenvolvidos no Algoritmo *Split*;
3. Mapeamento de formalismos estruturados para um formato tensorial; especificação dos detalhes envolvidos nessa transformação e implicações de cada tipo de mapeamento na estrutura tensorial; escrita do capítulo de tradução e mapeamento no volume da tese com aplicação a um dos formalismos, e.g., PEPA nets;
4. Escrita de artigos científicos para publicação em conferências ou revistas nacionais ou internacionais;
5. Preparação do volume final da tese de doutorado, com a definição do algoritmo de otimização da divisão dos termos tensoriais dos descritores markovianos;
6. Preparação para apresentação da defesa, concepção dos *slides*, fluxo de idéias, discussões, questões a serem abordadas;
7. Defesa da tese de doutorado.

6.3.3 Publicações

Nesta seção são apresentadas as publicações do trabalho realizado até o momento.

- *"Split: a flexible and efficient algorithm to vector-descriptor product"*

  Aceito no SMCTools'07 (International Workshop on Tools for solving Structured Markov Chains)

  Categoria: paper completo.

  Autores: Czekster R. M., Fernandes P., Vincent J. M., Webber T.

  Local: Nantes, France, 2007.

O seguinte trabalho foi submetido:

- *"Hybrid vector-descriptor product solution exploiting time-memory trade-offs"*

  Submetido para *Performance Evaluation Review* (revista internacional)

  Categoria: paper completo.

  Autores: Czekster R. M., Fernandes P., Webber T.

Será iniciada a preparação de um trabalho com o objetivo de ser submetido à uma revista internacional qualificada de avaliação de desempenho versando sobre os estudos realizados nesse trabalho referentes às reestruturações dos termos tensoriais dos descritores markovianos tanto para o caso constante como para o caso generalizado. Este trabalho discutirá as formas mais otimizadas para dividir os termos tensoriais mostrando os resultados que foram obtidos para a série de modelos da literatura que mapeiam diferentes realidades. O título do trabalho é:

- *"Time-memory tradeoffs for the optimized solution of generalized Kronecker descriptors."*

6.4 Considerações finais e perspectivas futuras

A avaliação de desempenho e modelagem de sistemas auxilia no entendimento de sistemas complexos e possibilita uma análise que produz resultados numéricos relevantes. Estes mecanismos são utilizados na descoberta de problemas e gargalos (para citar apenas alguns) em sistemas antes que estes sejam colocados fisicamente em produção (*i.e.*, compra de máquinas para um *cluster*, aumento do número de servidores em uma empresa, etc). Esta abordagem analítica permite que sejam inferidos índices de desempenho que auxiliam os modeladores a apropriadamente dimensionar seus sistemas de forma a otimizar alguma determinada característica ou métrica. Estes índices variam de sistema para sistema e também as modelagens existentes das diferentes realidades, dependendo única e exclusivamente da criatividade e capacidade de abstração dos usuários dos formalismos de avaliação de desempenho existentes atualmente.

Este trabalho direcionou-se ao estudo de formalismos mais importantes existentes atualmente, em particular no tangente a Redes de Autômatos Estocásticos e sua solução numérica. Foi constatado que, ao dividir-se os termos tensoriais que compõem um descritor, um novo problema surge, relacionado à onde exatamente dividir o termo de forma a maximizar o tempo gasto para a solução de modelos. Este trabalho estudou diversas estratégias para a solução de descritores constantes e generalizados ao mesmo tempo que comparou estas diferentes possibilidades para o Algoritmo *Split*. Com estes resultados, foi possível descrever um algoritmo que calcula o melhor ponto de corte para um termo tensorial, observando a memória gasta e o tempo ganho de solução, tanto para descritores constantes quanto para descritores generalizados.

As abordagens anteriores a esta possuíam características extremas, ou seja, ou efetuavam a MVD de forma extremamente rápida ocupando vastas quantidade de memória ou extremamente eficiente em termos de gastos em

memória mas levavam muito tempo para atingir a solução. Este trabalho combinou estes dois fatores primordiais da solução de descritores markovianos e propôs uma técnica híbrida para descritores constantes e generalizados que utiliza um total de memória razoável e realiza a MVD em menos tempo. O Algoritmo *Split* apresentado foi refinado para aumentar a variedade de modelos que resolve, no caso, descritores baseados em ATG. Pesquisas anteriores a esse trabalho desconheciam-se as implicações do algoritmo ao lidar com descritores generalizados. Foram elencados as principais problemáticas ao se trabalhar com termos tensoriais generalizados, enumerando as formas de se dividi-los com o intuito de respeitar as limitações das funções (ou seja, não permitindo indefinições de qualquer sorte ao avaliar elementos funcionais devido aos parâmetros que são requeridos).

Este trabalho proporcionou um estudo teórico que listou as classes de modelos onde o Algoritmo *Split* ofereceu ganhos consideráveis em termos de tempo para efetuar a MVD e uma validação prática que discutiu os principais experimentos que deveriam ser executados para que fosse possível comparar as diferentes formas suficientemente. O algoritmo da determinação do ponto de corte proposto neste trabalho auxiliará na solução mais otimizada de modelos complexos. As classes que obtiveram os tempos mais otimizados auxiliaram na descoberta das propriedades fundamentais existentes sobre os descritores markovianos. Foi possível inferir que a interação entre os autômatos (uma relação direta com as sincronizações que são realizadas ao nível das entidades participantes destes eventos) é que ditará o desempenho do método pois está diretamente relacionado à quantidade de AUNFs que serão necessários e no número de matrizes identidades existentes em cada termo tensorial.

Ao comparar modelos constantes e generalizados, observou-se que, para alguns casos, não vale a pena realizar esta tarefa. Por este motivo, o uso de taxas funcionais realmente auxilia na modelagem de sistemas complexos, como discutido em outros trabalhos [14]. O método de MVD a ser executado deve se preocupar com a melhor forma de realizar as multiplicações envolvidas. O algoritmo de reestruturação dos termos tensoriais aqui proposto preserva o poder de modelagem oferecido por SAN através da definição de taxas funcionais ao mesmo tempo que torna transparente para os usuários a conversão de descritores baseados em ATG para ATC (quando isso se faz necessário, ao avaliar-se as funções na parte esparsa), atividade feita neste trabalho de forma implícita.

6.4.1 Resumo

Este trabalho enumerou as principais estratégias de reestruturação de descritores markovianos constantes e generalizados e propôs uma nova maneira de se determinar um ponto de corte que utilize os recursos de forma razoável e produza uma resposta em menos tempo em comparação com as alternativas disponíveis anteriormente. Os experimentos conduzidos no Capítulo 5 foram cruciais para determinar as condições necessárias para que os modelos melhor executem o Algoritmo *Split*.

A série de modelos aqui apresentados correspondem a tanto modelagens clássicas da literatura como compartilhamento de recursos quanto pesquisas recentes que foram modeladas em SAN como arquiteturas Mestre-Escravo e Redes de Sensores. Este modelos possuem características peculiares quanto ao seu descritor markoviano, com matrizes com diferentes ordens, total de elementos não-nulos, identidades e elementos funcionais. O Algoritmo *Split* ofereceu uma nova maneira de se efetuar a MVD utilizando-se um incremento razoável de memória quando comparado ao Algoritmo *Shuffle*. O *Split* também é mais flexível que o Algoritmo *Sparso* que é custoso quanto ao armazenamento dos elementos e oferece um tempo mais otimizado de solução dos modelos em questão. Estes resultados obtidos inspiraram o algoritmo de determinação do corte dos termos tensoriais (delineado na Seção 6.1) que leva em consideração a memória a ser potencialmente gasta para cada caso.

Foram utilizados descritores tensoriais a partir da modelagem existente em SAN. Entretanto, cabe ressaltar que os resultados obtidos até o momento servirão para qualquer descritor markoviano a ser criado, com taxas constantes ou funcionais. O novo capítulo que versará sobre a tradução de diferentes realidades para descritores será importante para a contribuição final da tese pois permitirá uma solução otimizada para muitas modelagens. A ideia é que os descritores possuem propriedades fundamentais quanto à maneira pela qual são definidos, logo a aplicabilidade deste trabalho é válida para diferentes modelagens.

Bibliografia

[1] M. Ajmone-Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli, and G. Franceschinis. An Introduction to Generalized Stochastic Petri Nets. *Microelectronics and Reliability*, 31(4):699–725, 1991.

[2] M. Ajmone-Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[4] G. Balbo. *Introduction to Generalized Stochastic Petri Nets*, volume 4486, chapter Performance Modelling and Markov Chains, pages 83–131. Springer-Verlag, 2007.

[5] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Models for Master/Slave Parallel Programs. *Electronic Notes In Theoretical Computer Science*, 128(4):101–121, April 2005.

[6] L. Baldo, L. G. Fernandes, P. Roisenberg, P. Velho, and T. Webber. Parallel PEPS Tool Performance Analysis using Stochastic Automata Networks. In M. Donelutto, D. Laforenza, and M. Vanneschi, editors, *Euro-Par 2004 International Conference on Parallel Processing*, volume 3149 of *Lecture Notes in Computer Science*, pages 214–219, Pisa, Italy, August/September 2004. Springer-Verlag Heidelberg.

[7] A. Benoit. *Méthodes et algorithmes pour l'évaluation des performances des systèmes informatiques à grand espace d'états*. PhD thesis, Institut National Polytechnique de Grenoble, France, 2003.

[8] A. Benoit, B. Plateau, and W. Stewart. Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems. *Future Generation Computer Systems*, 22(7):838–847, 2006.

[9] A. Benoit, B. Plateau, and W. J. Stewart. Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems. In *Proceedings of International Parallel and Distributed Processing Symposium*, pages 275–282, Nice, France, April 2003.

[10] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1–2):1–54, 1998.

[11] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 1998.

[12] J. Bradley, N. Dingle, S. Gilmore, and W. Knottenbelt. Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In G. Kotsis, editor, *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 344–351, University of Central Florida, Oct. 2003. IEEE Computer Society Press.

[13] L. Brenner, P. Fernandes, B. Plateau, and I. Sbeity. PEPS2007 – Stochastic Automata Networks Software Tool. In *QEST 2007*, pages 163–164. IEEE Press, 2007.

[14] L. Brenner, P. Fernandes, and A. Sales. The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations. *Int. Journal of Simulation: Systems, Science & Technology*, 6(3-4):52–60, February 2005.

[15] P. Buchholz. A new approach combining simulation and randomization for the analysis of large continuous time Markov Chains. *ACM Trans. Model. Comput. Simul.*, 8(2):194–222, 1998.

[16] P. Buchholz. Structured analysis techniques for large markov chains. In *SMCtools '06: Proceeding from the 2006 workshop on Tools for solving structured Markov chains*, volume 2, New York, NY, USA, 2006. ACM.

[17] P. Buchholz and P. Kemper. Hierarchical reachability graph generation for Petri nets. *Formal Methods in Systems Design*, 21(3):281–315, 2002.

[18] G. Chiola, M. Ajmone-Marsan, G. Balbo, and G. Conte. Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, 1993.

[19] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. *Proceedings of the 11th International Conference on Application and Theory of Petri Nets'90*, pages 387–410, 1990.

[20] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transaction on Computers*, 42(11):1343–1360, 1993.

[21] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured petri nets. *Theoretical Computer Science*, 176(1-2):39–65, 1997.

[22] M.-Y. Chung, G. Ciardo, S. Donatelli, N. He, B. Plateau, W. J. Stewart, E. Sulaiman, and J. Yu. A Comparison of Structural Formalisms for Modeling Large Markov Models. In *IPDPS Next Generation Software Program - NSFNGS - PI Workshop*, Santa Fe, New Mexico, USA, April 2004.

[23] G. Ciardo and A. S. Miner. Storage Alternatives for Large Structured State Spaces. In *9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 1245 of *LNCS*, pages 44–57, St. Malo, France, 1997. Springer-Verlag Heidelberg.

[24] R. M. Czekster, P. Fernandes, J. M. Vincent, and T. Webber. Split: a flexible and efficient algorithm to vector-descriptor product. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[25] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers*, C-30(2):116–125, 1981.

[26] D. D. Deavours and W. H. Sanders. An efficient disk-based tool for solving very large markov models. In *Proceedings of the 9th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, pages 58–71, St. Malo, France, 1997. Lecture Notes in Computer Science, no. 1245.

[27] D. D. Deavours and W. H. Sanders. "on-the-fly" solution techniques for stochastic petri nets and extensions. In *PNPM '97: Proceedings of the 6th International Workshop on Petri Nets and Performance Models*, page 132, Washington, DC, USA, 1997. IEEE Computer Society.

[28] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[29] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In R. Valette, editor, *Proceedings of the* 15^{th} *International Conference on Applications and Theory of Petri Nets*, pages 258–277. Springer-Verlag Heidelberg, 1994.

[30] F. L. Dotti, P. Fernandes, A. Sales, and O. M. Santos. Modular Analytical Performance Models for Ad Hoc Wireless Networks. In 3^{rd} *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 164–173, Trentino, Italy, April 2005. IEEE Press.

[31] F. L. Dotti and L. Ribeiro. Specification of Mobile Code Systems using Graph Grammars. In *Formal Methods for Open Object-Based Distributed Systems IV*, pages 45–63, Stanford, USA, 2000. Kluwer Academic Publishers.

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

[33] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor - Vector multiplication in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[34] P. Fernandes, B. Plateau, and W. J. Stewart. Optimizing tensor product computations in stochastic automata networks. *RAIRO. Recherche opérationnelle*, 32(3):325–351, 1998.

[35] P. Fernandes, J. M. Vincent, and T. Webber. Perfect Simulation of Stochastic Automata Networks. In *Proceedings of 15th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'08)*, volume 5055 of *LNCS*, pages 249–263. Springer-Verlag Heidelberg, June 2008.

[36] G. Florin and S. Natkin. Les reseaux de Petri stochastiques. *Tecniques et Sciences Informatiques*, 4(1):143–160, 1985.

[37] S. Gilmore, V. Haenel, J. Hillston, and J. Tenzer. A design environment for mobile applications. *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10, 2006.

[38] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: a structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, 2003.

[39] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. Software performance modelling using PEPA nets. In *Proceedings of the Fourth International Workshop on Software and Performance*, pages 13–24, Redwood Shores, California, USA, Jan. 2004. ACM Press.

[40] S. Gilmore, J. Hillston, and M. Ribaudo. PEPA-coloured stochastic Petri nets. In *Proceedings of the Seventeenth UK Performance Engineering Workshop*, pages 155–166, 2001.

[41] S. Gilmore, J. Hillston, M. Ribaudo, and L. Kloul. PEPA nets: A structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, Oct. 2003.

[42] S. Gilmore, L. Kloul, and D. Piazza. Modelling Role-Playing Games Using PEPA nets. *Proceedings of the 19th International Symposium on Computer and Information Sciences (ISCIS 2004)*, 3280:523–532, 2004.

[43] N. Gotz, H. Hermanns, U. Herzog, V. Mertsiotakis, and M. Rettelbach. Stochastic process algebras. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, pages 3–17. Basic Research Series, Springer, 1995.
[44] J. Hillston. *A compositional approach to performance modelling*. Cambridge University Press, New York, USA, 1996.
[45] J. Hillston. Process algebras for quantitative analysis. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 239–248, Chicago, June 2005. IEEE Computer Society Press.
[46] J. Hillston and L. Kloul. An Efficient Kronecker Representation for PEPA models. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop)*, pages 120–135, Aachen, Germany, September 2001. Springer-Verlag Heidelberg.
[47] J. Hillston and L. Kloul. Formal techniques for performance analysis: blending SAN and PEPA. *Formal Aspects of Computing*, 19(1):3–33, 2007.
[48] J. Hillston and M. Ribaudo. Modelling mobility with PEPA nets. *Proceedings of the 19th International Conference on Computer and Information Sciences*, 3280:513–522, 2004.
[49] D. Holton. A PEPA specification of an industrial production cell. pages 542–551.
[50] K. Jensen. Coloured Petri Nets and the Invariant Method. *Theoretical Computer Science*, 14:317–336, 1981.
[51] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Springer-Verlag London, UK, 1996.
[52] K. Jensen, L. M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9:213–254, 2007.
[53] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.
[54] J. Li, C. Blake, D. S. J. D. Couto, H. I. Lee, and R. Morris. Capacity of Ad Hoc Wireless Networks. In 7^{th} *Annual International Conference on Mobile Computing and Networking*, pages 61–69, Rome, Italy, July 2001. ACM Press.
[55] A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. In *The notes of the Imperial Academy of Sciences of St. Petersburg VIII Series*, volume XXII, No. 9, Physio-Mathematical College, December 5 1907.
[56] J. Meyer, A. Movaghar, and W. Sanders. Stochastic Activity Networks: Structure, Behavior, and Application. *International Workshop on Timed Petri Nets table of contents*, pages 106–115, 1985.
[57] A. S. Miner. *Data Structures for the Analysis of Large Structured Markov Models*. PhD thesis, The College of William and Mary, Williamsburg, VA, 2000.
[58] A. S. Miner and G. Ciardo. A data structure for the efficient Kronecker solution of GSPNs. In *Proceedings of the* 8^{th} *International Workshop on Petri Nets and Performance Models*, pages 22–31, Zaragoza, Spain, September 1999.
[59] A. S. Miner and G. Ciardo. Efficient Reachability Set Generation and Storage Using Decision Diagrams. In *Proceedings of the* 20^{th} *International Conference on Applications and Theory of Petri Nets*, volume 1639 of *LNCS*, pages 6–25, Williamsburg, VA, USA, June 1999. Springer-Verlag Heidelberg.
[60] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, C-31(9):913–917, 1982.
[61] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
[62] J. R. Norris. *Markov Chains*. Cambridge University Press, New York, USA, 1998.
[63] J. L. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3):223–252, 1977.
[64] C. A. Petri. Communicaton with automata. *DTIC Research Report AD0630125*, 1966.
[65] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, USA, 1985. ACM Press.
[66] B. Plateau and K. Atif. Stochastic Automata Networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.
[67] W. Reisig. *Petri nets: an introduction*. Springer-Verlag Heidelberg, 1985.
[68] O. Stenflo. Ergodic Theorems for Markov chains represented by Iterated Function Systems. *Bull. Polish Acad. Sci. Math.*, 49(1):27–43, 2001.
[69] W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.
[70] W. J. Stewart. *Performance Modelling and Markov Chains*, volume 4486, chapter Performance Modelling and Markov Chains, pages 1–34. Springer-Verlag, 2007.
[71] C. Tadonki and B. Philippe. Parallel Multiplication of a Vector by a Kronecker Tensor Product of matrices. *Parallel and Distributed Computing Practices*, 2(4):53–67, 1999.
[72] L. Wells. Performance analysis using CPN tools. In *valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodologies and tools*, page 59, New York, NY, USA, 2006. ACM.

A Modelos e experimentos

Este apêndice mostrará as definições dos principais modelos utilizados ao longo do trabalho. Apenas alguns modelos serão mostrados neste apêndice, devido ao volume de dados existente (foi escolhido apenas um caso de cada modelo disponível). Serão apresentados os dados dos seguintes casos (apresenta-se também a seção onde cada resultado encontra-se melhor explicado):

Experimentos para descritores constantes (1 e 3):

- *slaves_10c* – Seção 5.2.1;
- *ad14c* – Seção 5.2.2;
- *fas20c* – Seção 5.2.3;
- *phil14c* – Seção 5.2.5;
- *rs15_15c* – Seção 5.2.6;

Experimentos para descritores generalizados (5, 6, 7 e 8):

- *slaves_10f* – Seção 5.2.1;
- *ad14f* – Seção 5.2.2;

Juntamente com a definição da SAN na Seção A.1 (o arquivo de definição dos autômatos), serão mostradas as esparsidades dos termos tensoriais positivos, o ponto de corte escolhido em cada experimento, a memória e o número dos AUNFs e a ordem escolhida para as permutações. Serão mostrados também trechos do arquivo de resultados do PEPS (.tim).

A.1 Definições SAN

A.1.1 Caso slaves_10c

```
identifiers
  P = [0..9];
  NP = 10;
  K = [0..40];
  t_l = 1.6; // tempo de processamento de um esccravo
  t_s = 16; // tempo de envio do mestre para um dos escravos
  t_r = 10.5; // tempo de envio do buffer
  t_c = 6.3;
  t_up = 11;
  pi1 = 0.75;
  pi2 = 0.25;
  t_d = 0.1;

events
  syn d    (t_d);
  loc l[P] (t_l);
  syn s[P] (t_s);
  syn r[P] (t_r);
  syn c    (t_c);
```

```
  syn c0   (t_c);
  syn c1   (t_c);
  syn c2   (t_c);
  syn c3   (t_c);
  syn c4   (t_c);
  syn c5   (t_c);
  syn c6   (t_c);
  syn c7   (t_c);
  syn c8   (t_c);
  syn c9   (t_c);
  syn up   (t_up);

reachability = ((st Master != ITx) || ((st Buffer == Buf[0]) && (nb [Slave0..Slave9] I == NP))) &&
               !((st Master == MTx) && (st Buffer == Buf[40]) && (nb [Slave0..Slave9] Tx == NP)) &&
               !((st Master == MRx) && (st Buffer == Buf[0]) && (nb [Slave0..Slave9] I == NP));

network propagation (continuous)
  aut Master
    stt MTx to(MRx) s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9]
    stt MRx to(ITx) d
            to(==)  c
            to(MTx) c0 c1 c2 c3 c4 c5 c6 c7 c8 c9
    stt ITx to(MRx) up

  aut Buffer
    stt Buf[K] to(++)     r[0] r[1] r[2] r[3] r[4] r[5] r[6] r[7] r[8] r[9]
               to(--)     c c0 c1 c2 c3 c4 c5 c6 c7 c8 c9
               to(Buf[0]) d

  aut Slave0
    stt I   to(Pr) up s[0]
            to(I)  d c0
    stt Pr  to(Tx) l[0]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[0](pi1)
            to(I)  r[0](pi2) d
            to(Tx) c

  aut Slave1
    stt I   to(Pr) up s[1]
            to(I)  d c1
    stt Pr  to(Tx) l[1]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[1](pi1)
            to(I)  r[1](pi2) d
            to(Tx) c

  aut Slave2
    stt I   to(Pr) up s[2]
            to(I)  d c2
    stt Pr  to(Tx) l[2]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[2](pi1)
            to(I)  r[2](pi2) d
            to(Tx) c

  aut Slave3
    stt I   to(Pr) up s[3]
            to(I)  d c3
    stt Pr  to(Tx) l[3]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[3](pi1)
            to(I)  r[3](pi2) d
            to(Tx) c

  aut Slave4
    stt I   to(Pr) up s[4]
            to(I)  d c4
    stt Pr  to(Tx) l[4]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[4](pi1)
            to(I)  r[4](pi2) d
            to(Tx) c

  aut Slave5
    stt I   to(Pr) up s[5]
            to(I)  d c5
    stt Pr  to(Tx) l[5]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[5](pi1)
            to(I)  r[5](pi2) d
            to(Tx) c

  aut Slave6
    stt I   to(Pr) up s[6]
```

```
            to(I)  d c6
    stt Pr  to(Tx) l[6]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[6](pi1)
            to(I)  r[6](pi2) d
            to(Tx) c

  aut Slave7
    stt I   to(Pr) up s[7]
            to(I)  d c7
    stt Pr  to(Tx) l[7]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[7](pi1)
            to(I)  r[7](pi2) d
            to(Tx) c

  aut Slave8
    stt I   to(Pr) up s[8]
            to(I)  d c8
    stt Pr  to(Tx) l[8]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[8](pi1)
            to(I)  r[8](pi2) d
            to(Tx) c

  aut Slave9
    stt I   to(Pr) up s[9]
            to(I)  d c9
    stt Pr  to(Tx) l[9]
            to(I)  d
            to(Pr) c
    stt Tx  to(Pr) r[9](pi1)
            to(I)  r[9](pi2) d
            to(Tx) c

results
  master_tx = (st Master == MTx);
  master_rx = (st Master == MRx);
  master_itx = (st Master == ITx);
  slave1_i = (st Slave1 == I);
  slave1_pr = (st Slave1 == Pr);
  slave1_tx = (st Slave1 == Tx);
  slaves_pr = (nb [Slave0..Slave9] Pr == NP);
  slaves_i =  (nb [Slave0..Slave9] I == NP);
  slaves_tx = (nb [Slave0..Slave9] Tx == NP);
```

A.1.2 Caso slaves_10f

```
identifiers
  P = [0..9];
  NP = 10;
  K = [0..40];
  t_l = 1.6; // tempo de processamento de um esccravo
  t_s = 16;  // tempo de envio do mestre para um dos escravos
  t_r = 10.5; // tempo de envio do buffer
  t_c = 6.3;
  t_up = 11;
  pi1 = 0.75;
  pi2 = 0.25;
  g1 =  nb [Slave0..Slave9] I == 0;
  g2 =  nb [Slave0..Slave9] I > 0;
  t_d = 0.1;

events
  syn d    (t_d);
  loc l[P] (t_l);
  syn s[P] (t_s);
  syn r[P] (t_r);
  syn c    (t_c);
  syn up   (t_up);

reachability = ((st Master != ITx) || ((st Buffer == Buf[0]) && (nb [Slave0..Slave9] I == NP))) &&
               !((st Master == MTx) && (st Buffer == Buf[40]) && (nb [Slave0..Slave9] Tx == NP)) &&
               !((st Master == MRx) && (st Buffer == Buf[0]) && (nb [Slave0..Slave9] I == NP));

network propagation (continuous)
  aut Master
    stt MTx to(MRx) s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9]
    stt MRx to(ITx) d
            to(==)  c(g1)
            to(MTx) c(g2)
    stt ITx to(MRx) up

  aut Buffer
    stt Buf[K] to(++)     r[0] r[1] r[2] r[3] r[4] r[5] r[6] r[7] r[8] r[9]
```

```
            to(--)     c
            to(Buf[0]) d

  aut Slave0
    stt I   to(Pr) up s[0]
            to(I)  d
    stt Pr  to(Tx) l[0]
            to(I)  d
    stt Tx  to(Pr) r[0](pi1)
            to(I)  r[0](pi2) d

  aut Slave1
    stt I   to(Pr) up s[1]
            to(I)  d
    stt Pr  to(Tx) l[1]
            to(I)  d
    stt Tx  to(Pr) r[1](pi1)
            to(I)  r[1](pi2) d


  aut Slave2
    stt I   to(Pr) up s[2]
            to(I)  d
    stt Pr  to(Tx) l[2]
            to(I)  d
    stt Tx  to(Pr) r[2](pi1)
            to(I)  r[2](pi2) d

  aut Slave3
    stt I   to(Pr) up s[3]
            to(I)  d
    stt Pr  to(Tx) l[3]
            to(I)  d
    stt Tx  to(Pr) r[3](pi1)
            to(I)  r[3](pi2) d


  aut Slave4
    stt I   to(Pr) up s[4]
            to(I)  d
    stt Pr  to(Tx) l[4]
            to(I)  d
    stt Tx  to(Pr) r[4](pi1)
            to(I)  r[4](pi2) d

  aut Slave5
    stt I   to(Pr) up s[5]
            to(I)  d
    stt Pr  to(Tx) l[5]
            to(I)  d
    stt Tx  to(Pr) r[5](pi1)
            to(I)  r[5](pi2) d

  aut Slave6
    stt I   to(Pr) up s[6]
            to(I)  d
    stt Pr  to(Tx) l[6]
            to(I)  d
    stt Tx  to(Pr) r[6](pi1)
            to(I)  r[6](pi2) d

  aut Slave7
    stt I   to(Pr) up s[7]
            to(I)  d
    stt Pr  to(Tx) l[7]
            to(I)  d
    stt Tx  to(Pr) r[7](pi1)
            to(I)  r[7](pi2) d

  aut Slave8
    stt I   to(Pr) up s[8]
            to(I)  d
    stt Pr  to(Tx) l[8]
            to(I)  d
    stt Tx  to(Pr) r[8](pi1)
            to(I)  r[8](pi2) d

  aut Slave9
    stt I   to(Pr) up s[9]
            to(I)  d
    stt Pr  to(Tx) l[9]
            to(I)  d
    stt Tx  to(Pr) r[9](pi1)
            to(I)  r[9](pi2) d

results
  master_tx = (st Master == MTx);
  master_rx = (st Master == MRx);
  master_itx = (st Master == ITx);
  slave1_i = (st Slave1 == I);
  slave1_pr = (st Slave1 == Pr);
```

```
  slave1_tx = (st Slave1 == Tx);
  slaves_pr = (nb [Slave0..Slave9] Pr == NP);
  slaves_i =  (nb [Slave0..Slave9] I == NP);
  slaves_tx = (nb [Slave0..Slave9] Tx == NP);
```

A.1.3 Caso ad14c

```
identifiers
  bandwidth_b = 2000000;                  // bandwidth in bits/seconds
  bandwidth_B = bandwidth_b/8;            // bandwidth in bytes/seconds
  bandwidth_Mbps = bandwidth_b/1000000;   // bandwidth in bytes/seconds
  size_pack = 1500;                         // package size in bytes
  RTS = 40;                               // Request To Send header in bytes
  CTS_ACK = 39;                           // Clear To Send and Acknowledge headers in bytes
  MAC = 47;                               // Medium Access Control header in bytes
  IFT = 50+(3*10)+280;                    // InterFrame Time (DIFS + 3*SIFS + Average Backoff Time) in microseconds (approximated)
  IFS = ((bandwidth_B/1000000)*IFT);      // InterFrame Size cost in bytes (approximated)
  overhead = RTS+CTS_ACK+MAC+IFS;         // headers
  mu = (bandwidth_B)/(size_pack+overhead); // maximum throughput rate (theoretically as many package as the medium can handle)
  lambda = 50000;                         // package generation rate (one package/DIFS)

  r12  = lambda;
  r23  = lambda;
  r34  = lambda;
  r45  = lambda;
  r56  = lambda;
  r67  = lambda;
  r78  = lambda;
  r89  = lambda;
  r910 = lambda;
  r1011= lambda;
  r1112= lambda;
  r1213= lambda;
  r1314 = lambda;

events
  loc tx1 mu;                             // transmission of one package
  loc tx2 mu;                             // transmission of one package
  loc tx3 mu;                             // transmission of one package
  loc tx4 mu;                             // transmission of one package
  loc tx5 mu;                             // transmission of one package
  loc tx6 mu;                             // transmission of one package
  loc tx7 mu;                             // transmission of one package
  loc tx8 mu;                             // transmission of one package
  loc tx9 mu;                             // transmission of one package
  loc tx10 mu;                             // transmission of one package
  loc tx11 mu;                             // transmission of one package
  loc tx12 mu;                             // transmission of one package
  syn tx13 mu;                             // transmission of one package
  syn g12  r12;                           // package generated from 1 to 2
  syn g23  r23;                           // package routed from 2 to 3
  syn g34  r34;                           // package routed from 3 to 4
  syn g45  r45;                           // package routed from 4 to 5
  syn g56  r56;                           // package routed from 5 to 6
  syn g67  r67;                           // package routed from 6 to 7
  syn g78  r78;                           // package routed from 7 to 8
  syn g89  r89;                           // package routed from 8 to 9
  syn g910 r910;                          // package routed from 9 to 10
  syn g1011 r1011;                        // package routed from 10 to 11
  syn g1112 r1112;                        // package routed from 11 to 12
  syn g1213 r1213;                        // package routed from 12 to 13
  syn g1314 r1314;                        // package routed from 13 to 14

partial reachability = (nb I == 14);       // initial state

network Ad (continuous)
  aut MN_1
    stt I to (T) g12
        to (I) g23 g34
    stt T to (I) tx1

  aut MN_2
    stt I to (R) g12
        to (I) g34 g45
    stt R to (T) g23
    stt T to (I) tx2

  aut MN_3
    stt I to (R) g23
        to (I) g12 g45 g56
    stt R to (T) g34
    stt T to (I) tx3

  aut MN_4
    stt I to (R) g34
        to (I) g12 g23 g56 g67
    stt R to (T) g45
    stt T to (I) tx4
```

```
  aut MN_5
    stt I to (R) g45
        to (I) g23 g34 g67 g78
    stt R to (T) g56
    stt T to (I) tx5

  aut MN_6
    stt I to (R) g56
        to (I) g34 g45 g78 g89
    stt R to (T) g67
    stt T to (I) tx6

  aut MN_7
    stt I to (R) g67
        to (I) g45 g56 g89 g910
    stt R to (T) g78
    stt T to (I) tx7

  aut MN_8
    stt I to (R) g78
        to (I) g56 g67 g910 g1011
    stt R to (T) g89
    stt T to (I) tx8

  aut MN_9
    stt I to (R) g89
        to (I) g67 g78 g1011 g1112
    stt R to (T) g910
    stt T to (I) tx9

  aut MN_10
    stt I to (R) g910
        to (I) g78 g89 g1112 g1213
    stt R to (T) g1011
    stt T to (I) tx10

  aut MN_11
    stt I to (R) g1011
        to (I) g89 g910 g1213 g1314
    stt R to (T) g1112
    stt T to (I) tx11

  aut MN_12
    stt I to (R) g1112
        to (I) g910 g1011 g1314
    stt R to (T) g1213
    stt T to (I) tx12

  aut MN_13
    stt I to (R) g1213
        to (I) g1011 g1112
    stt R to (T) g1314
    stt T to (I) tx13

  aut MN_14
    stt I to (R) g1314
        to (I) g1112 g1213
    stt R to (T) tx13


results
  t_MN_1_I = st MN_1 == I;
  t_MN_1_T = (st MN_1 == T) * bandwidth_Mbps * (size_pack/(size_pack+overhead));
```

A.1.4 Caso ad14f

```
identifiers
  bandwidth_b = 2000000;                   // bandwidth in bits/seconds
  bandwidth_B = bandwidth_b/8;             // bandwidth in bytes/seconds
  bandwidth_Mbps = bandwidth_b/1000000;    // bandwidth in bytes/seconds
  size_pack = 1500;                          // package size in bytes
  RTS = 40;                                // Request To Send header in bytes
  CTS_ACK = 39;                            // Clear To Send and Acknowledge headers in bytes
  MAC = 47;                                // Medium Access Control header in bytes
  IFT = 50+(3*10)+280;                     // InterFrame Time (DIFS + 3*SIFS + Average Backoff Time) in microseconds (approximated)
  IFS = ((bandwidth_B/1000000)*IFT);       // InterFrame Size cost in bytes (approximated)
  overhead = RTS+CTS_ACK+MAC+IFS;          // headers
  mu = (bandwidth_B)/(size_pack+overhead); // maximum throughput rate (theoretically as many package as the medium can handle)
  lambda = 50000;                          // package generation rate (one package/DIFS)

  r12  = lambda  * (                                  (st MN_3 == I) && (st MN_4 == I));
  r23  = lambda  * (                 (st MN_1 == I) && (st MN_4 == I) && (st MN_5 == I));
  r34  = lambda  * ((st MN_1 == I) && (st MN_2 == I) && (st MN_5 == I) && (st MN_6 == I));
  r45  = lambda  * ((st MN_2 == I) && (st MN_3 == I) && (st MN_6 == I) && (st MN_7 == I));
  r56  = lambda  * ((st MN_3 == I) && (st MN_4 == I) && (st MN_7 == I) && (st MN_8 == I));
  r67  = lambda  * ((st MN_4 == I) && (st MN_5 == I) && (st MN_8 == I) && (st MN_9 == I));
  r78  = lambda  * ((st MN_5 == I) && (st MN_6 == I) && (st MN_9 == I) && (st MN_10 == I));
```

```
  r89  = lambda  * ((st MN_6 == I) && (st MN_7 == I) && (st MN_10 == I)&& (st MN_11 == I));
  r910 = lambda  * ((st MN_7 == I) && (st MN_8 == I) && (st MN_11 == I)&& (st MN_12 == I));
  r1011 = lambda  * ((st MN_8 == I) && (st MN_9 == I) && (st MN_12 == I)&& (st MN_13 == I));
  r1112 = lambda  * ((st MN_9 == I) && (st MN_10 == I) && (st MN_13 == I)&& (st MN_14 == I));
  r1213 = lambda * ((st MN_10 == I) && (st MN_11 == I)&& (st MN_14 == I));
  r1314 = lambda * ((st MN_11 == I) && (st MN_12 == I));

events
  loc tx1 mu;                              // transmission of one package
  loc tx2 mu;                              // transmission of one package
  loc tx3 mu;                              // transmission of one package
  loc tx4 mu;                              // transmission of one package
  loc tx5 mu;                              // transmission of one package
  loc tx6 mu;                              // transmission of one package
  loc tx7 mu;                              // transmission of one package
  loc tx8 mu;                              // transmission of one package
  loc tx9 mu;                              // transmission of one package
  loc tx10 mu;                              // transmission of one package
  loc tx11 mu;                              // transmission of one package
  loc tx12 mu;                              // transmission of one package
  syn tx13 mu;                              // transmission of one package
  syn g12  r12;                            // package generated from 1 to 2
  syn g23  r23;                            // package routed from 2 to 3
  syn g34  r34;                            // package routed from 3 to 4
  syn g45  r45;                            // package routed from 4 to 5
  syn g56  r56;                            // package routed from 5 to 6
  syn g67  r67;                            // package routed from 6 to 7
  syn g78  r78;                            // package routed from 7 to 8
  syn g89  r89;                            // package routed from 8 to 9
  syn g910 r910;                           // package routed from 9 to 10
  syn g1011 r1011;                         // package routed from 10 to 11
  syn g1112 r1112;                         // package routed from 11 to 12
  syn g1213 r1213;                         // package routed from 12 to 13
  syn g1314 r1314;                         // package routed from 13 to 14

partial reachability = (nb I == 14);        // initial state

network Ad (continuous)
  aut MN_1
    stt I to (T) g12
    stt T to (I) tx1

  aut MN_2
    stt I to (R) g12
    stt R to (T) g23
    stt T to (I) tx2

  aut MN_3
    stt I to (R) g23
    stt R to (T) g34
    stt T to (I) tx3

  aut MN_4
    stt I to (R) g34
    stt R to (T) g45
    stt T to (I) tx4

  aut MN_5
    stt I to (R) g45
    stt R to (T) g56
    stt T to (I) tx5

  aut MN_6
    stt I to (R) g56
    stt R to (T) g67
    stt T to (I) tx6

  aut MN_7
    stt I to (R) g67
    stt R to (T) g78
    stt T to (I) tx7

  aut MN_8
    stt I to (R) g78
    stt R to (T) g89
    stt T to (I) tx8

  aut MN_9
    stt I to (R) g89
    stt R to (T) g910
    stt T to (I) tx9

  aut MN_10
    stt I to (R) g910
    stt R to (T) g1011
    stt T to (I) tx10

  aut MN_11
    stt I to (R) g1011
    stt R to (T) g1112
    stt T to (I) tx11
```

```
  aut MN_12
    stt I to (R) g1112
    stt R to (T) g1213
    stt T to (I) tx12

  aut MN_13
    stt I to (R) g1213
    stt R to (T) g1314
    stt T to (I) tx13

  aut MN_14
    stt I to (R) g1314
    stt R to (I) tx13

results
  t_MN_1_I = st MN_1 == I;
  t_MN_1_T = (st MN_1 == T) * bandwidth_Mbps * (size_pack/(size_pack+overhead));
```

A.1.5 Caso fas20c

```
//=== FAS - First Available Server (N=20) ===
identifiers
  // Acquisition rate
  lambda = 2.000000;
  // Release rate
  mu = 1.000000;

events
  loc t1 (lambda);   syn t2 (lambda);
  syn t3 (lambda);   syn t4 (lambda);
  syn t5 (lambda);   syn t6 (lambda);
  syn t7 (lambda);   syn t8 (lambda);
  syn t9 (lambda);   syn t10 (lambda);
  syn t11 (lambda);  syn t12 (lambda);
  syn t13 (lambda);  syn t14 (lambda);
  syn t15 (lambda);  syn t16 (lambda);
  syn t17 (lambda);  syn t18 (lambda);
  syn t19 (lambda);  syn t20 (lambda);
  loc r1 (mu);       loc r2 (mu);
  loc r3 (mu);       loc r4 (mu);
  loc r5 (mu);       loc r6 (mu);
  loc r7 (mu);       loc r8 (mu);
  loc r9 (mu);       loc r10 (mu);
  loc r11 (mu);      loc r12 (mu);
  loc r13 (mu);      loc r14 (mu);
  loc r15 (mu);      loc r16 (mu);
  loc r17 (mu);      loc r18 (mu);
  loc r19 (mu);      loc r20 (mu);

reachability = 1;

network FAS20c (continuous)

  aut Server1 stt idle to (busy) t1
              stt busy to (idle) r1
                       to (busy) t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server2 stt idle to (busy) t2
              stt busy to (idle) r2
                       to (busy) t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server3 stt idle to (busy) t3
              stt busy to (idle) r3
                       to (busy) t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server4 stt idle to (busy) t4
              stt busy to (idle) r4
                       to (busy) t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server5 stt idle to (busy) t5
              stt busy to (idle) r5
                       to (busy) t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server6 stt idle to (busy) t6
              stt busy to (idle) r6
                       to (busy) t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server7 stt idle to (busy) t7
              stt busy to (idle) r7
                       to (busy) t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server8 stt idle to (busy) t8
              stt busy to (idle) r8
                       to (busy) t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server9 stt idle to (busy) t9
              stt busy to (idle) r9
```

```
                    to (busy) t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server10 stt idle to (busy) t10
              stt busy to (idle) r10
                       to (busy) t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server11 stt idle to (busy) t11
              stt busy to (idle) r11
                       to (busy) t12 t13 t14 t15 t16 t17 t18 t19 t20

  aut Server12 stt idle to (busy) t12
              stt busy to (idle) r12
                       to (busy) t13 t14 t15 t16 t17 t18 t19 t20

  aut Server13 stt idle to (busy) t13
              stt busy to (idle) r13
                       to (busy) t14 t15 t16 t17 t18 t19 t20

  aut Server14 stt idle to (busy) t14
              stt busy to (idle) r14
                       to (busy) t15 t16 t17 t18 t19 t20

  aut Server15 stt idle to (busy) t15
              stt busy to (idle) r15
                       to (busy) t16 t17 t18 t19 t20

  aut Server16 stt idle to (busy) t16
              stt busy to (idle) r16
                       to (busy) t17 t18 t19 t20

  aut Server17 stt idle to (busy) t17
              stt busy to (idle) r17
                       to (busy) t18 t19 t20

  aut Server18 stt idle to (busy) t18
              stt busy to (idle) r18
                       to (busy) t19 t20

  aut Server19 stt idle to (busy) t19
              stt busy to (idle) r19
                       to (busy) t20

  aut Server20 stt idle to (busy) t20
              stt busy to (idle) r20

results
used1 = (st Server1 == busy);
used2 = (st Server2 == busy);
```

A.1.6 Caso phil14c

```
//=== Dining philosophers problem (P=14) ===
identifiers
  // Number of philosophers
  P = 14;
  // Acquisition rate
  lambda = 1.000000;
  // Release rate
  mu = 2.000000;

events

  syn t_r_0 (lambda);
  syn r_l_0 (lambda);
  loc l_t_0 (mu);

  syn t_r_1 (lambda);
  syn r_l_1 (lambda);
  loc l_t_1 (mu);

  syn t_r_2 (lambda);
  syn r_l_2 (lambda);
  loc l_t_2 (mu);

  syn t_r_3 (lambda);
  syn r_l_3 (lambda);
  loc l_t_3 (mu);

  syn t_r_4 (lambda);
  syn r_l_4 (lambda);
  loc l_t_4 (mu);

  syn t_r_5 (lambda);
  syn r_l_5 (lambda);
  loc l_t_5 (mu);

  syn t_r_6 (lambda);
  syn r_l_6 (lambda);
```

```
  loc l_t_6 (mu);

  syn t_r_7 (lambda);
  syn r_l_7 (lambda);
  loc l_t_7 (mu);

  syn t_r_8 (lambda);
  syn r_l_8 (lambda);
  loc l_t_8 (mu);

  syn t_r_9 (lambda);
  syn r_l_9 (lambda);
  loc l_t_9 (mu);

  syn t_r_10 (lambda);
  syn r_l_10 (lambda);
  loc l_t_10 (mu);

  syn t_r_11 (lambda);
  syn r_l_11 (lambda);
  loc l_t_11 (mu);

  syn t_r_12 (lambda);
  syn r_l_12 (lambda);
  loc l_t_12 (mu);

  syn t_l_13 (lambda);
  syn l_r_13 (lambda);
  loc r_t_13 (mu);

reachability = ((nb Thinking) == P);

network PHILOSOPHERS (continuous)

  aut P13
    stt Thinking to (Left)     t_l_13
                to (Thinking) r_l_0 t_r_12
    stt Left     to (Right)    l_r_13
                to (Left)     r_l_0
    stt Right    to (Thinking) r_t_13

  aut P12
    stt Thinking to (Right)    t_r_12
                to (Thinking) t_r_11 t_l_13
    stt Right    to (Left)     r_l_12
                to (Right)    t_r_11
    stt Left     to (Thinking) l_t_12

  aut P11
    stt Thinking to (Right)    t_r_11
                to (Thinking) t_r_10 r_l_12
    stt Right    to (Left)     r_l_11
                to (Right)    t_r_10
    stt Left     to (Thinking) l_t_11

  aut P10
    stt Thinking to (Right)    t_r_10
                to (Thinking) t_r_9 r_l_11
    stt Right    to (Left)     r_l_10
                to (Right)    t_r_9
    stt Left     to (Thinking) l_t_10

  aut P9
    stt Thinking to (Right)    t_r_9
                to (Thinking) t_r_8 r_l_10
    stt Right    to (Left)     r_l_9
                to (Right)    t_r_8
    stt Left     to (Thinking) l_t_9

  aut P8
    stt Thinking to (Right)    t_r_8
                to (Thinking) t_r_7 r_l_9
    stt Right    to (Left)     r_l_8
                to (Right)    t_r_7
    stt Left     to (Thinking) l_t_8

  aut P7
    stt Thinking to (Right)    t_r_7
                to (Thinking) t_r_6 r_l_8
    stt Right    to (Left)     r_l_7
                to (Right)    t_r_6
    stt Left     to (Thinking) l_t_7

  aut P6
    stt Thinking to (Right)    t_r_6
                to (Thinking) t_r_5 r_l_7
    stt Right    to (Left)     r_l_6
                to (Right)    t_r_5
    stt Left     to (Thinking) l_t_6

  aut P5
```

```
    stt Thinking to (Right)    t_r_5
                to (Thinking) t_r_4 r_l_6
    stt Right    to (Left)     r_l_5
                to (Right)    t_r_4
    stt Left     to (Thinking) l_t_5

  aut P4
    stt Thinking to (Right)    t_r_4
                to (Thinking) t_r_3 r_l_5
    stt Right    to (Left)     r_l_4
                to (Right)    t_r_3
    stt Left     to (Thinking) l_t_4

  aut P3
    stt Thinking to (Right)    t_r_3
                to (Thinking) t_r_2 r_l_4
    stt Right    to (Left)     r_l_3
                to (Right)    t_r_2
    stt Left     to (Thinking) l_t_3

  aut P2
    stt Thinking to (Right)    t_r_2
                to (Thinking) t_r_1 r_l_3
    stt Right    to (Left)     r_l_2
                to (Right)    t_r_1
    stt Left     to (Thinking) l_t_2

  aut P1
    stt Thinking to (Right)    t_r_1
                to (Thinking) t_r_0 r_l_2
    stt Right    to (Left)     r_l_1
                to (Right)    t_r_0
    stt Left     to (Thinking) l_t_1

  aut P0
    stt Thinking to (Right)    t_r_0
                to (Thinking) r_l_1 l_r_13
    stt Right    to (Left)     r_l_0
                to (Right)    l_r_13
    stt Left     to (Thinking) l_t_0

results

PhilThinking0 = (st P0 == Thinking);
PhilRight0    = (st P0 == Right);
PhilLeft0     = (st P0 == Left);
```

A.1.7 Caso rs15_15c

```
//=== Resource Sharing (N=15 P=15) ===
identifiers
  // Number of Resources
  R = 15;
 _R = [0..15];
  // Acquisition rate
  lambda = 1.000000;
  // Release rate
  mu = 2.000000;

events

  syn t0 (lambda);   syn t1 (lambda);   syn t2 (lambda);   syn t3 (lambda);
  syn t4 (lambda);   syn t5 (lambda);   syn t6 (lambda);   syn t7 (lambda);
  syn t8 (lambda);   syn t9 (lambda);   syn t10 (lambda);  syn t11 (lambda);
  syn t12 (lambda);  syn t13 (lambda);  syn t14 (lambda);  syn r0 (mu);
  syn r1 (mu);   syn r2 (mu);   syn r3 (mu);   syn r4 (mu);
  syn r5 (mu);   syn r6 (mu);   syn r7 (mu);   syn r8 (mu);
  syn r9 (mu);   syn r10 (mu);  syn r11 (mu);  syn r12 (mu);
  syn r13 (mu);  syn r14 (mu);

reachability = ((nb [a14..a0] on <= R) && (nb [a14..a0] on == st Res));

network RS15_15 (continuous)

  aut Res
    stt using[_R]
      to (++)
        t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14
      to (--)
        r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14

  aut a14
    stt off to (on)  t14
    stt on  to (off) r14

  aut a13
    stt off to (on)  t13
    stt on  to (off) r13
```

```
  aut a12
    stt off to (on)  t12
    stt on  to (off) r12

  aut a11
    stt off to (on)  t11
    stt on  to (off) r11

  aut a10
    stt off to (on)  t10
    stt on  to (off) r10

  aut a9
    stt off to (on)  t9
    stt on  to (off) r9

  aut a8
    stt off to (on)  t8
    stt on  to (off) r8

  aut a7
    stt off to (on)  t7
    stt on  to (off) r7

  aut a6
    stt off to (on)  t6
    stt on  to (off) r6

  aut a5
    stt off to (on)  t5
    stt on  to (off) r5

  aut a4
    stt off to (on)  t4
    stt on  to (off) r4

  aut a3
    stt off to (on)  t3
    stt on  to (off) r3

  aut a2
    stt off to (on)  t2
    stt on  to (off) r2

  aut a1
    stt off to (on)  t1
    stt on  to (off) r1

  aut a0
    stt off to (on)  t0
    stt on  to (off) r0

results
  used = (st Res);
```

A.2 Experimento 1

A.2.1 Caso slaves_10c - Experimento 1

```
================================================================
File slaves10c.tim
================================================================
slaves10c.san -- A model with 12 automata and 33 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Automata sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 3553 Kbytes
================================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
================================================================
Number of AUNFs: 2463180 (76974 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 12 3 3 3 3 3 3 3 3 3 3 3 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
    0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
```

```
   4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
   5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
   6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
   7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
   8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
   9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
  10. [  0  1  2  3  4  5  6  7  8  9 10 11|]    [   1  41   3   3   3   3   3   3   3   3   3|  ]
  11. [  0  1  2  3  4  5  6  7  8  9 10 11|]    [   1  40   2   2   2   2   2   2   2   2   2|  ]
  12. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  13. [  0  1  3| 2  4  5  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  14. [  0  1  4| 2  3  5  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  15. [  0  1  5| 2  3  4  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  16. [  0  1  6| 2  3  4  5  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  17. [  0  1  7| 2  3  4  5  6  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  18. [  0  1  8| 2  3  4  5  6  7  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  19. [  0  1  9| 2  3  4  5  6  7  8 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  20. [  0  1 10| 2  3  4  5  6  7  8  9 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  21. [  0  1 11| 2  3  4  5  6  7  8  9 10 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
  22. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]    [   1   1   1   1   1   1   1   1   1   1|  I   ]
  23. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  24. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  25. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  26. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  27. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  28. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  29. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  30. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  31. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
  32. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
```

A.2.2 Caso ad14c - Experimento 1

```
============================================================
File ad14c.tim
============================================================
ad14c.san -- A model with 14 automata and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:         1 states
Automata sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 14 (0 Kb)
Chosen cuts: [ 2 4 5 6 6 6 6 6 6 6 6 6 5 4 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1| I I I I I I I I I I I I ]
  1. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1| I I I I I I I I I I ]
  2. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1| I I I I I I I I I ]
  3. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  4. [  1  2  3  4  5  6| 0  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  5. [  2  3  4  5  6  7| 0  1  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  6. [  3  4  5  6  7  8| 0  1  2  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  7. [  4  5  6  7  8  9| 0  1  2  3 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  8. [  5  6  7  8  9 10| 0  1  2  3  4 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  9. [  6  7  8  9 10 11| 0  1  2  3  4  5 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
 10. [  7  8  9 10 11 12| 0  1  2  3  4  5  6 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
 11. [  8  9 10 11 12 13| 0  1  2  3  4  5  6  7 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
 12. [  9 10 11 12 13| 0  1  2  3  4  5  6  7  8 ]  [ 1 1 1 1 1| I I I I I I I I I ]
 13. [ 10 11 12 13| 0  1  2  3  4  5  6  7  8  9 ]  [ 1 1 1 1| I I I I I I I I I ]
```

A.2.3 Caso fas20c - Experimento 1

```
============================================================
File fas20c.tim
============================================================
fas20c.san -- A model with 20 automata and 19 events
User name: 'FAS20c'
------------
Problem Size
------------
Product state space:           1048576 states
Reachable state space:         1048576 states
Automata sizes:                   [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 522 Kbytes
============================================================
```

```
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 19 (0 Kb)
Chosen cuts: [ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1|I I I I I I I I I I I I I I I I I I I ]
  1. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1|I I I I I I I I I I I I I I I I I ]
  2. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1|I I I I I I I I I I I I I I I ]
  3. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1|I I I I I I I I I I I I I I ]
  4. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1|I I I I I I I I I I I I I ]
  5. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1|I I I I I I I I I I I I ]
  6. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1|I I I I I I I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1|I I I I I I I I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1|I I I I I I I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I ]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I ]
 12. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I ]
 13. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I ]
 14. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I ]
 15. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16|17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I ]
 16. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17|18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I ]
 17. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18|19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I ]
 18. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19|] [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|]
```

A.2.4 Caso phil14c - Experimento 1

```
============================================================
File phil14c.tim
============================================================
phil14c.san -- A model with 14 automata and 28 events
User name: 'PHILOSOPHERS'
------------
Problem Size
------------
Product state space:           4782969 states
Reachable state space:         1 states
Automata sizes:                   [ 3 3 3 3 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 2342 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 42 (1 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 2 1| I I I I I I I I I I I I ]
  1. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 ]  [ 2 1| I I I I I I I I I I I I ]
  2. [ 11 12| 0  1  2  3  4  5  6  7  8  9 10 13 ]  [ 2 1| I I I I I I I I I I I I ]
  3. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1| I I I I I I I I I I I I ]
  4. [ 10 11| 0  1  2  3  4  5  6  7  8  9 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
  5. [ 11 12| 0  1  2  3  4  5  6  7  8  9 10 13 ]  [ 1 1| I I I I I I I I I I I I ]
  6. [  9 10| 0  1  2  3  4  5  6  7  8 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
  7. [ 10 11| 0  1  2  3  4  5  6  7  8  9 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
  8. [  8  9| 0  1  2  3  4  5  6  7 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
  9. [  9 10| 0  1  2  3  4  5  6  7  8 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 10. [  7  8| 0  1  2  3  4  5  6  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 11. [  8  9| 0  1  2  3  4  5  6  7 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 12. [  6  7| 0  1  2  3  4  5  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 13. [  7  8| 0  1  2  3  4  5  6  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 14. [  5  6| 0  1  2  3  4  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 15. [  6  7| 0  1  2  3  4  5  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 16. [  4  5| 0  1  2  3  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 17. [  5  6| 0  1  2  3  4  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 18. [  3  4| 0  1  2  5  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 19. [  4  5| 0  1  2  3  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 20. [  2  3| 0  1  4  5  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 21. [  3  4| 0  1  2  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 22. [  1  2| 0  3  4  5  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 23. [  2  3| 0  1  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 24. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 25. [  1  2| 0  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 26. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 27. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 ]  [ 1 2| I I I I I I I I I I I I ]
```

A.2.5 Caso rs15_15c - Experimento 1

```
============================================================
File rs15_15c.tim
============================================================
rs15_15c.san -- A model with 16 automata and 30 events
```

```
User name: 'RS15_15'
------------
Problem Size
------------
Product state space:           524288 states
Reachable state space:         32768 states
Automata sizes:                   [ 16 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 266 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 450 (14 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0 15| 1  2  3  4  5  6  7  8  9 10 11 12 13 14 ]  [  15 1| I I I I I I I I I I I I I I ]
  1. [  0 14| 1  2  3  4  5  6  7  8  9 10 11 12 13 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  2. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  3. [  0 12| 1  2  3  4  5  6  7  8  9 10 11 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  4. [  0 11| 1  2  3  4  5  6  7  8  9 10 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  5. [  0 10| 1  2  3  4  5  6  7  8  9 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  6. [  0  9| 1  2  3  4  5  6  7  8 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  7. [  0  8| 1  2  3  4  5  6  7  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  8. [  0  7| 1  2  3  4  5  6  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
  9. [  0  6| 1  2  3  4  5  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 10. [  0  5| 1  2  3  4  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 11. [  0  4| 1  2  3  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 12. [  0  3| 1  2  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 13. [  0  2| 1  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 14. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 15. [  0 15| 1  2  3  4  5  6  7  8  9 10 11 12 13 14 ]  [  15 1| I I I I I I I I I I I I I I ]
 16. [  0 14| 1  2  3  4  5  6  7  8  9 10 11 12 13 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 17. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 18. [  0 12| 1  2  3  4  5  6  7  8  9 10 11 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 19. [  0 11| 1  2  3  4  5  6  7  8  9 10 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 20. [  0 10| 1  2  3  4  5  6  7  8  9 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 21. [  0  9| 1  2  3  4  5  6  7  8 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 22. [  0  8| 1  2  3  4  5  6  7  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 23. [  0  7| 1  2  3  4  5  6  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 24. [  0  6| 1  2  3  4  5  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 25. [  0  5| 1  2  3  4  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 26. [  0  4| 1  2  3  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 27. [  0  3| 1  2  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 28. [  0  2| 1  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 29. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
```

A.3 Experimento 3

A.3.1 Caso slaves_10c - Experimento 3

```
============================================================
File slaves10c.tim
============================================================
slaves10c.san -- A model with 12 automata and 33 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Automata sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 3553 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 11939214 (373100 Kb)
Chosen cuts: [ 3 4 5 6 7 8 9 10 11 12 12 12 3 4 5 6 7 8 9 10 11 12 12 3 4 5 6 7 8 9 10 11 12 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]  [ 1  I 1|I I I I I I I I I ]
  1. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]  [ 1  I I 1|I I I I I I I I ]
  2. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]  [ 1  I I I 1|I I I I I I I ]
  3. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]  [ 1  I I I I 1|I I I I I I ]
  4. [  0  1  2  3  4  5  6| 7  8  9 10 11 ]  [ 1  I I I I I 1|I I I I I ]
  5. [  0  1  2  3  4  5  6  7| 8  9 10 11 ]  [ 1  I I I I I I 1|I I I I ]
  6. [  0  1  2  3  4  5  6  7  8| 9 10 11 ]  [ 1  I I I I I I I 1|I I I ]
  7. [  0  1  2  3  4  5  6  7  8  9|10 11 ]  [ 1  I I I I I I I I 1|I I ]
  8. [  0  1  2  3  4  5  6  7  8  9 10|11 ]  [ 1  I I I I I I I I I 1|I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1  I I I I I I I I I I 1|]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3|]
```

```
 11. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 40 2 2 2 2 2 2 2 2 2 2|]
 12. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]  [ 1 40 1|I I I I I I I I I I ]
 13. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]  [ 1 40 I 1|I I I I I I I I I ]
 14. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]  [ 1 40 I I 1|I I I I I I I I ]
 15. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]  [ 1 40 I I I 1|I I I I I I I ]
 16. [  0  1  2  3  4  5  6| 7  8  9 10 11 ]  [ 1 40 I I I I 1|I I I I I I ]
 17. [  0  1  2  3  4  5  6  7| 8  9 10 11 ]  [ 1 40 I I I I I 1|I I I I I ]
 18. [  0  1  2  3  4  5  6  7  8| 9 10 11 ]  [ 1 40 I I I I I I 1|I I I I ]
 19. [  0  1  2  3  4  5  6  7  8  9|10 11 ]  [ 1 40 I I I I I I I 1|I I I ]
 20. [  0  1  2  3  4  5  6  7  8  9 10|11 ]  [ 1 40 I I I I I I I I 1|I I ]
 21. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 40 I I I I I I I I I 1|I ]
 22. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1  I 1 1 1 1 1 1 1 1 1 1|I ]
 23. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]  [ I 40 2|I I I I I I I I I I ]
 24. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]  [ I 40 I 2|I I I I I I I I I ]
 25. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]  [ I 40 I I 2|I I I I I I I I ]
 26. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]  [ I 40 I I I 2|I I I I I I I ]
 27. [  0  1  2  3  4  5  6| 7  8  9 10 11 ]  [ I 40 I I I I 2|I I I I I I ]
 28. [  0  1  2  3  4  5  6  7| 8  9 10 11 ]  [ I 40 I I I I I 2|I I I I I ]
 29. [  0  1  2  3  4  5  6  7  8| 9 10 11 ]  [ I 40 I I I I I I 2|I I I I ]
 30. [  0  1  2  3  4  5  6  7  8  9|10 11 ]  [ I 40 I I I I I I I 2|I I I ]
 31. [  0  1  2  3  4  5  6  7  8  9 10|11 ]  [ I 40 I I I I I I I I 2|I I ]
 32. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ I 40 I I I I I I I I I 2|I ]
```

A.3.2 Caso ad14c - Experimento 3

```
============================================================
File ad14c.tim
============================================================
ad14c.san -- A model with 14 automata and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:         1 states
Automata sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 413345 (12917 Kb)
Chosen cuts: [ 14 4 5 6 7 8 9 10 11 12 13 14 14 14 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I I 1 1|  ]
  1. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1|I I I I I I I I I I ]
  2. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1|I I I I I I I I I ]
  3. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1|I I I I I I I I ]
  4. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 ]  [ I 1 1 1 1 1 1|I I I I I I I ]
  5. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 ]  [ I I 1 1 1 1 1 1|I I I I I I ]
  6. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 ]  [ I I I 1 1 1 1 1 1|I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 ]  [ I I I I 1 1 1 1 1 1|I I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 ]  [ I I I I I 1 1 1 1 1 1|I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 ]  [ I I I I I I 1 1 1 1 1 1|I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 ]  [ I I I I I I I 1 1 1 1 1 1|I ]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I 1 1 1 1 1 1|]
 12. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I 1 1 1 1 1|]
 13. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I 1 1 1 1|]
```

A.3.3 Caso fas20c - Experimento 3

```
============================================================
File fas20c.tim
============================================================
fas20c.san -- A model with 20 automata and 19 events
User name: 'FAS20c'
------------
Problem Size
------------
Product state space:           1048576 states
Reachable state space:         1048576 states
Automata sizes:                   [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 522 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 19 (0 Kb)
Chosen cuts: [ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
```

```
  0. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1|I I I I I I I I I I I I I I I I I I ]
  1. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1|I I I I I I I I I I I I I I I I ]
  2. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1|I I I I I I I I I I I I I I ]
  3. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1|I I I I I I I I I I I I I ]
  4. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1|I I I I I I I I I I I I ]
  5. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1|I I I I I I I I I I I ]
  6. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1|I I I I I I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1|I I I I I I I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1|I I I I I I I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I ]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I ]
 12. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I ]
 13. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I ]
 14. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I ]
 15. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16|17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I ]
 16. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17|18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I ]
 17. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18|19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I ]
 18. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19|]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|]
```

A.3.4 Caso phil14c - Experimento 3

```
============================================================
File phil14c.tim
============================================================
phil14c.san -- A model with 14 automata and 28 events
User name: 'PHILOSOPHERS'
------------
Problem Size
------------
Product state space:           4782969 states
Reachable state space:         1 states
Automata sizes:                  [ 3 3 3 3 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 2342 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 4517246 (141163 Kb)
Chosen cuts: [ 14 14 13 14 12 13 11 12 10 11 9 10 8 9 7 8 6 7 5 6 4 5 3 4 2 3 2 14 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I I I 2 1|]
  1. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ 2 I I I I I I I I I I I I 1|]
  2. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 ]  [ I I I I I I I I I I I 2 1|I ]
  3. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I I 1 1|]
  4. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 ]  [ I I I I I I I I I I 2 1|I I ]
  5. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 ]  [ I I I I I I I I I I 1 1|I ]
  6. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 ]  [ I I I I I I I I I 2 1|I I I ]
  7. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 ]  [ I I I I I I I I I 1 1|I I ]
  8. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 ]  [ I I I I I I I I 2 1|I I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 ]  [ I I I I I I I I 1 1|I I I ]
 10. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 ]  [ I I I I I I I 2 1|I I I I ]
 11. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 ]  [ I I I I I I I 1 1|I I I I ]
 12. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 ]  [ I I I I I I 2 1|I I I I I ]
 13. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 ]  [ I I I I I I 1 1|I I I I I ]
 14. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 ]  [ I I I I I 2 1|I I I I I I ]
 15. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 ]  [ I I I I I 1 1|I I I I I I ]
 16. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ I I I I 2 1|I I I I I I I ]
 17. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
 18. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ I I I 2 1|I I I I I I I I ]
 19. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ I I I 1 1|I I I I I I I I ]
 20. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ I I 2 1|I I I I I I I I I ]
 21. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ I I I 1 1|I I I I I I I I ]
 22. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 ]  [ I 2 1|I I I I I I I I I ]
 23. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ I I 1 1|I I I I I I I I ]
 24. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1|I I I I I I I I I I ]
 25. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 ]  [ I 1 1|I I I I I I I I I ]
 26. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1|I I I I I I I I I I ]
 27. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ 1 I I I I I I I I I I 2|]
```

A.3.5 Caso rs15_15c - Experimento 3

```
============================================================
File rs15_15c.tim
============================================================
rs15_15c.san -- A model with 16 automata and 30 events
User name: 'RS15_15'
------------
Problem Size
------------
Product state space:           524288 states
Reachable state space:         32768 states
Automata sizes:                  [ 16 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
```

```
Automata sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 266 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 983010 (30719 Kb)
Chosen cuts: [ 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|]  [  15 I I I I I I I I I I I I I 1|]
  1. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 ]  [  15 I I I I I I I I I I I I 1|I ]
  2. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 ]  [  15 I I I I I I I I I I I 1|I I ]
  3. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 ]  [  15 I I I I I I I I I I 1|I I I ]
  4. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 ]  [  15 I I I I I I I I I 1|I I I I ]
  5. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 ]  [  15 I I I I I I I I 1|I I I I I ]
  6. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 ]  [  15 I I I I I I I 1|I I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 ]  [  15 I I I I I I 1|I I I I I I I ]
  8. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 ]  [  15 I I I I I 1|I I I I I I I I ]
  9. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 ]  [  15 I I I I 1|I I I I I I I I I ]
 10. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 ]  [  15 I I I 1|I I I I I I I I I ]
 11. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I I 1|I I I I I I I I I I ]
 12. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I 1|I I I I I I I I I I I ]
 13. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1|I I I I I I I I I I I I ]
 14. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1|I I I I I I I I I I I I I ]
 15. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|]  [  15 I I I I I I I I I I I I 1|]
 16. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 ]  [  15 I I I I I I I I I I I 1|I ]
 17. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 ]  [  15 I I I I I I I I I I 1|I I ]
 18. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 ]  [  15 I I I I I I I I I 1|I I I ]
 19. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 ]  [  15 I I I I I I I I 1|I I I I ]
 20. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 ]  [  15 I I I I I I I 1|I I I I I ]
 21. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 ]  [  15 I I I I I I 1|I I I I I I ]
 22. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 ]  [  15 I I I I I 1|I I I I I I I ]
 23. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 ]  [  15 I I I I 1|I I I I I I I I ]
 24. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 ]  [  15 I I I 1|I I I I I I I I I ]
 25. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 ]  [  15 I I 1|I I I I I I I I I I ]
 26. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I 1|I I I I I I I I I I I ]
 27. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1|I I I I I I I I I I I I ]
 28. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I 1|I I I I I I I I I I I I ]
 29. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1|I I I I I I I I I I I I I ]
```

A.4 Experimento 5

A.4.1 Caso slaves_10f - Experimento 5

```
============================================================
File slaves10f.tim
============================================================
slaves10f.san -- A model with 12 automata and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Automata sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2421860 (75683 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 1 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3|]
 11. [  1| 2  3  4  5  6  7  8  9 10 11  0 ]  [ 40|I I I I I I I I I 2 ]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [  1 1 1 1 1 1 1 1 1 1|I ]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
```

```
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I I ]
```

A.4.2 Caso ad14f - Experimento 5

```
============================================================
File ad14f.tim
============================================================
ad14f.san -- A model with 14 automata and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:         1 states
Automata sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 134137 (4191 Kb)
Chosen cuts: [ 2 11 10 9 9 9 9 9 9 9 9 9 10 11 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1|I I I I I I I I I I I I ]
  1. [  1  4  5  6  7  8  9 10 11 12 13| 2  3  0 ]  [ 1 I I I I I I I I I|I I 1 ]
  2. [  2  5  6  7  8  9 10 11 12 13| 0  3  4  1 ]  [ 1 I I I I I I I I I|I I I 1 ]
  3. [  3  6  7  8  9 10 11 12 13| 0  1  4  5  2 ]  [ 1 I I I I I I I I I|I I I I 1 ]
  4. [  0  4  7  8  9 10 11 12 13| 1  2  5  6  3 ]  [ I 1 I I I I I I I|I I I I 1 ]
  5. [  0  1  5  8  9 10 11 12 13| 2  3  6  7  4 ]  [ I I 1 I I I I I I|I I I I 1 ]
  6. [  0  1  2  6  9 10 11 12 13| 3  4  7  8  5 ]  [ I I I 1 I I I I I|I I I I 1 ]
  7. [  0  1  2  3  7 10 11 12 13| 4  5  8  9  6 ]  [ I I I I 1 I I I I|I I I I 1 ]
  8. [  0  1  2  3  4  8 11 12 13| 5  6  9 10  7 ]  [ I I I I I 1 I I I|I I I I 1 ]
  9. [  0  1  2  3  4  5  9 12 13| 6  7 10 11  8 ]  [ I I I I I I 1 I I|I I I I 1 ]
 10. [  0  1  2  3  4  5  6 10 13| 7  8 11 12  9 ]  [ I I I I I I I 1 I|I I I I 1 ]
 11. [  0  1  2  3  4  5  6  7 11|13  8  9 12 10 ]  [ I I I I I I I I 1|I I I I 1 ]
 12. [  0  1  2  3  4  5  6  7  8 12|13  9 10 11 ]  [ I I I I I I I I I 1|I I I 1 ]
 13. [  0  1  2  3  4  5  6  7  8  9 13|10 11 12 ]  [ I I I I I I I I I I 1|I I 1 ]
```

A.5 Experimento 6

A.5.1 Caso slaves_10f - Experimento 6

```
============================================================
File slaves10f.tim
============================================================
slaves10f.san -- A model with 12 automata and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Automata sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2539918 (79372 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 11 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3 3|]
 11. [  2  3  4  5  6  7  8  9 10 11  0| 1 ]  [ I  I I I I I I I I I 2|40]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [ 1  1 1 1 1 1 1 1 1 1 1| I]
```

```
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I I ]
```

A.5.2 Caso ad14f - Experimento 6

```
============================================================
File ad14f.tim
============================================================
ad14f.san -- A model with 14 automata and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:         1 states
Automata sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 730 (22 Kb)
Chosen cuts: [ 2 3 4 5 5 5 5 5 5 5 5 5 4 3 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1|I I I I I I I I I I I I ]
  1. [  2  3  0| 1  4  5  6  7  8  9 10 11 12 13 ]  [ I I 1|1 I I I I I I I I I ]
  2. [  0  3  4  1| 2  5  6  7  8  9 10 11 12 13 ]  [ I I I 1|1 I I I I I I I I ]
  3. [  0  1  4  5  2| 3  6  7  8  9 10 11 12 13 ]  [ I I I I 1|1 I I I I I I I ]
  4. [  1  2  5  6  3| 0  4  7  8  9 10 11 12 13 ]  [ I I I I 1|I 1 I I I I I I ]
  5. [  2  3  6  7  4| 0  1  5  8  9 10 11 12 13 ]  [ I I I I 1|I I 1 I I I I I ]
  6. [  3  4  7  8  5| 0  1  2  6  9 10 11 12 13 ]  [ I I I I 1|I I I 1 I I I I ]
  7. [  4  5  8  9  6| 0  1  2  3  7 10 11 12 13 ]  [ I I I I 1|I I I I 1 I I I ]
  8. [  5  6  9 10  7| 0  1  2  3  4  8 11 12 13 ]  [ I I I I 1|I I I I I 1 I I ]
  9. [  6  7 10 11  8| 0  1  2  3  4  5  9 12 13 ]  [ I I I I 1|I I I I I I 1 I ]
 10. [  7  8 11 12  9| 0  1  2  3  4  5  6 10 13 ]  [ I I I I 1|I I I I I I I 1 ]
 11. [ 13  8  9 12 10| 0  1  2  3  4  5  6  7 11 ]  [ I I I I 1|I I I I I I I 1 ]
 12. [ 13  9 10 11| 0  1  2  3  4  5  6  7  8 12 ]  [ I I I 1|I I I I I I I I 1 ]
 13. [ 10 11 12| 0  1  2  3  4  5  6  7  8  9 13 ]  [ I I 1|I I I I I I I I I 1 ]
```

A.6 Experimento 7

A.6.1 Caso slaves_10f - Experimento 7

```
============================================================
File slaves10f.tim
============================================================
slaves10f.san -- A model with 12 automata and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Automata sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 7145740 (223304 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 12 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I  I ]
```

```
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I  I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I  I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3  3|]
 11. [  2  3  4  5  6  7  8  9 10 11  0  1|]  [ I  I I I I I I I I I 2 40|]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [ 1  1 1 1 1 1 1 1 1 1| I ]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I  I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I  I ]
```

A.6.2 Caso ad14f - Experimento 7

```
============================================================
File ad14f.tim
============================================================
ad14f.san -- A model with 14 automata and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:         1 states
Automata sizes:                  [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 730 (22 Kb)
Chosen cuts: [ 2 4 5 6 6 6 6 6 6 6 6 6 6 5 4 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1|I I I I I I I I I I I ]
  1. [  2  3  0  1| 4  5  6  7  8  9 10 11 12 13 ]  [ I I 1 1|I I I I I I I I I ]
  2. [  0  3  4  1  2| 5  6  7  8  9 10 11 12 13 ]  [ I I I 1 1|I I I I I I I I ]
  3. [  0  1  4  5  2  3| 6  7  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
  4. [  1  2  5  6  3  4| 0  7  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
  5. [  2  3  6  7  4  5| 0  1  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
  6. [  3  4  7  8  5  6| 0  1  2  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
  7. [  4  5  8  9  6  7| 0  1  2  3 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
  8. [  5  6  9 10  7  8| 0  1  2  3  4 11 12 13 ]  [ I I I I 1 1|I I I I I I I ]
  9. [  6  7 10 11  8  9| 0  1  2  3  4  5 12 13 ]  [ I I I I 1 1|I I I I I I I ]
 10. [  7  8 11 12  9 10| 0  1  2  3  4  5  6 13 ]  [ I I I I 1 1|I I I I I I I ]
 11. [ 13  8  9 12 10 11| 0  1  2  3  4  5  6  7 ]  [ I I I I 1 1|I I I I I I I ]
 12. [ 13  9 10 11 12| 0  1  2  3  4  5  6  7  8 ]  [ I I I 1 1|I I I I I I I ]
 13. [ 10 11 12 13| 0  1  2  3  4  5  6  7  8  9 ]  [ I I 1 1|I I I I I I I ]
```

A.7 Experimento 8

A.7.1 Caso slaves_10f - Experimento 8

```
============================================================
File slaves10f.tim
============================================================
slaves10f.san -- A model with 12 automata and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Automata sizes:                  [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2421860 (75683 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 1 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) informa
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
```

```
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3 3|]
 11. [  1| 2  3  4  5  6  7  8  9 10 11  0 ]  [ 40|I I I I I I I I I I 2 ]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [  1 1 1 1 1 1 1 1 1 1|I ]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I I ]
```

A.7.2 Caso ad14f - Experimento 8

```
============================================================
File ad14f.tim
============================================================
ad14f.san -- A model with 14 automata and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:         1 states
Automata sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 2 ]
Automata sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 14 (0 Kb)
Chosen cuts: [ 2 1 1 1 1 1 1 1 1 1 1 1 1 ]
Synchr. Term Information: evt. [ automata permutation orders and cut (|) information ] [ identities list]
    0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]    [ 1 1|I I I I I I I I I I I I ]
    1. [  1| 4  5  6  7  8  9 10 11 12 13  2  3  0 ]    [ 1|I I I I I I I I I I I I 1 ]
    2. [  2| 5  6  7  8  9 10 11 12 13  0  3  4  1 ]    [ 1|I I I I I I I I I I I I 1 ]
    3. [  3| 6  7  8  9 10 11 12 13  0  1  4  5  2 ]    [ 1|I I I I I I I I I I I I 1 ]
    4. [  4| 0  7  8  9 10 11 12 13  1  2  5  6  3 ]    [ 1|I I I I I I I I I I I I 1 ]
    5. [  5| 0  1  8  9 10 11 12 13  2  3  6  7  4 ]    [ 1|I I I I I I I I I I I I 1 ]
    6. [  6| 0  1  2  9 10 11 12 13  3  4  7  8  5 ]    [ 1|I I I I I I I I I I I I 1 ]
    7. [  7| 0  1  2  3 10 11 12 13  4  5  8  9  6 ]    [ 1|I I I I I I I I I I I I 1 ]
    8. [  8| 0  1  2  3  4 11 12 13  5  6  9 10  7 ]    [ 1|I I I I I I I I I I I I 1 ]
    9. [  9| 0  1  2  3  4  5 12 13  6  7 10 11  8 ]    [ 1|I I I I I I I I I I I I 1 ]
   10. [ 10| 0  1  2  3  4  5  6 13  7  8 11 12  9 ]    [ 1|I I I I I I I I I I I I 1 ]
   11. [ 11| 0  1  2  3  4  5  6  7 13  8  9 12 10 ]    [ 1|I I I I I I I I I I I I 1 ]
   12. [ 12| 0  1  2  3  4  5  6  7  8 13  9 10 11 ]    [ 1|I I I I I I I I I I I I 1 ]
   13. [ 13| 0  1  2  3  4  5  6  7  8  9 10 11 12 ]    [ 1|I I I I I I I I I I I I 1 ]
```

Estes dados encerram os resultados obtidos juntamente com os trechos dos arquivos de resultado do PEPS (.tim).

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo