

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Camillo de Lellis Falcão da Silva

**Novos Algoritmos de Simulação Estocástica com
Atraso para Redes Gênicas**

Juiz de Fora

2014

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Camillo de Lellis Falcão da Silva

**Novos Algoritmos de Simulação Estocástica com
Atraso para Redes Gênicas**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Itamar Leite de Oliveira

Juiz de Fora

2014

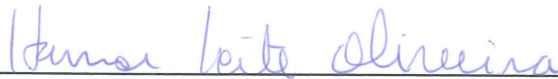
Camillo da Lellis Falcão da Silva

**Novos Algoritmos de Simulação Estocástica com Atraso para Redes
Gênicas**

Dissertação apresentada ao Programa
de Pós-graduação em Ciência da
Computação da Universidade Federal
de Juiz de Fora como requisito parcial
à obtenção do grau de Mestre.

Aprovada em 22 de Maio de 2014.

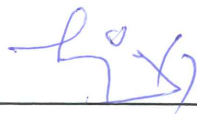
BANCA EXAMINADORA



Prof. Dr. Itamar Leite de Oliveira – Orientador
Universidade Federal de Juiz de Fora



Prof. Dr. Guilherme Albuquerque Pinto
Universidade Federal de Juiz de Fora



Prof. Dr. Luismar Marques Porto
Universidade Federal de Santa Catarina

Ao meu filho Heitor, que tem me proporcionado muitas alegrias nesses seus primeiros dias de vida. À minha esposa Léia, pelo amor, pela companhia e por suportar minhas ausências em um momento tão importante de nossas vidas. À minha mãe Júlia por tudo que fez e faz por mim.

AGRADECIMENTOS

Agradeço à minha esposa Léia por todo o amor e paciência com que me acompanhou durante a elaboração desse trabalho.

Agradeço à minha mãe Júlia por sempre me incentivar em todas as ocasiões.

Agradeço aos professores do PGCC pelos direcionamentos, pela inspiração e dedicação.

Agradeço aos alunos do PGCC e à funcionária Gláucia pela amizade e apoio.

Agradeço ao professor Custódio Motta e ao professor Marco Antônio Araújo por terem me preparado para o desafio do mestrado e por terem me incentivado a seguir a carreira acadêmica.

Por último e não menos importante, agradeço ao meu orientador Itamar Leite de Oliveira pelos conhecimentos passados e por ter acreditado e confiado em mim. O professor Itamar é um orientador presente e amigo, cujo grande conhecimento da área de pesquisa relacionada a esse trabalho me serve de inspiração.

*“I have come to believe that one’s
knowledge of any dynamical
system is deficient unless one
knows a valid way to numerically
simulate that system on a
computer.”*

Daniel T. Gillespie

RESUMO

Atualmente, a eficiência dos algoritmos de simulação estocástica para a simulação de redes de regulação gênica (RRG) tem motivado diversos trabalhos científicos. O interesse por tais algoritmos deve-se ao fato de as novas tecnologias em biologia celular – às vezes chamadas de tecnologias de alto rendimento (*high throughput technology cell biology*) – terem mostrado que a expressão gênica é um processo estocástico. Em RRG com atrasos, os algoritmos para simulação estocástica existentes possuem problemas – como crescimento linear da complexidade assintótica, descarte excessivo de números aleatórios durante a simulação e grande complexidade de codificação em linguagens de programação – que podem resultar em um baixo desempenho em relação ao tempo de processamento de simulação de uma RRG. Este trabalho apresenta um algoritmo para simulação estocástica que foi chamado de método da próxima reação simplificado (SNRM). Esse algoritmo mostrou-se mais eficiente que as outras abordagens existentes para simulações estocásticas realizadas com as RRGs com atrasos. Além do SNRM, um novo grafo de dependências para reações com atrasos também é apresentado. A utilização desse novo grafo, que foi nomeado de *delayed dependency graph* (DDG), aumentou consideravelmente a eficiência de todas as versões dos algoritmos de simulação estocástica com atrasos apresentados nesse trabalho. Finalmente, uma estrutura de dados que recebeu o nome de lista ordenada por *hashing* é utilizada para tratar a lista de produtos em espera em simulações de RRGs com atrasos. Essa estrutura de dados também se mostrou mais eficiente que uma *heap* em todas as simulações testadas. Com todas as melhorias mencionadas, este trabalho apresenta um conjunto de estratégias que contribui de forma efetiva para o desempenho dos algoritmos de simulação estocástica com atrasos de redes de regulação gênica.

Palavras-chave: SSA. DSSA. Simulação Estocástica. SNRM. DSNRM. DDG. Lista ordenada por *hashing*.

ABSTRACT

New Stochastic Simulation Algorithms with Delay for Gene Regulatory Networks

Recently, the time efficiency of stochastic simulation algorithms for gene regulatory networks (GRN) has motivated several scientific works. Interest in such algorithms is because the new technologies in cell biology – called high-throughput technologies cell biology – have shown that gene expression is a stochastic process. In GRN with delays, the existing algorithms for stochastic simulation have some drawbacks – such as linear growth of complexity, excessive discard of random numbers, and the coding in a programming language can be hard – that result in poor performance during the simulation of very large GRN. This work presents an algorithm for stochastic simulation of GRN. We called it simplified next reaction method (SNRM). This algorithm was more efficient than other existing algorithms for stochastically simulation of GRN with delays. Besides SNRM, a new dependency graph for delayed reactions is also presented. The use of this new graph, which we named it delayed dependency graph (DDG), greatly increased the efficiency of all versions of the algorithms for stochastic simulation with delays presented in this work. Finally, a data structure that we named hashing sorted list is used to handle the waiting list of products in simulations of GRN with delays. This data structure was also more efficient than a heap in all tested simulations. With all the improvements mentioned, this work presents a set of strategies that contribute effectively to increasing performance of stochastic simulation algorithms with delays for gene regulatory networks.

Keywords: SSA. DSSA. Stochastic Simulation. SNRM. DSNRM. DDG. Hashing Sorted List.

LISTA DE FIGURAS

2.1	Célula de um organismo eucarioto e de um procarioto	22
2.2	O ciclo celular.	23
2.3	Fita dupla de DNA	25
2.4	Dogma central da biologia molecular estendido, adaptado de (NELSON et al., 2008).	27
2.5	Operon Trp.	28
2.6	Representação de um grafo	42
2.7	Representação de uma matriz	42
2.8	Representações de um grafo	43
2.9	Uma <i>min-heap</i>	45
2.10	Estrutura de uma tabela <i>hash</i>	47
2.11	Fluxograma do algoritmo DM	54
2.12	Seleção do tipo roleta para a próxima reação	54
2.13	Fluxograma do algoritmo FRM	55
2.14	Fluxograma do algoritmo NRM	58
2.15	Fluxograma do algoritmo MNRM	59
2.16	Fluxograma do algoritmo <i>Rejection Method</i>	62
2.17	Fluxograma para o algoritmo DMNRM	65
3.1	Fluxograma para o algoritmo SNRM	67
3.2	Exemplo do Grafo de Dependências - DG	71
3.3	Exemplo do Grafo de Dependências - DDG	72
3.4	Fluxograma para o algoritmo DSNRM	74
3.5	Exemplo de vetor de listas ligadas	79
3.6	Estrutura de dados lista ordenada por <i>hashing</i>	80
4.1	Grafos de Dependências para o Modelo 4.1	82
4.2	Ganho em desempenho do algoritmo RM + DG + DDG em relação ao algo- ritmo RM + DG	83
4.3	Partes do DG para modelo da equação 4.2 (<i>Colloidal Aggregation</i>)	85

4.4	DDG para modelo da equação 4.2 (<i>Colloidal Aggregation</i>) com atraso	86
4.5	Comparação do desempenho dos algoritmos RM, MNRM e SNRM para o modelo 4.2 (<i>Colloidal Aggregation Model</i>)	88
4.6	Grafos de dependências para o modelo 4.3	89
4.7	Comparação do desempenho dos algoritmos RM, MNRM e SNRM para o modelo 4.3 (rede fracamente acoplada)	89
4.8	Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.4 (rede fortemente acoplada com múltiplos atrasos) com acoplamento de grau 2	91
4.9	Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.4 (rede fortemente acoplada com múltiplos atrasos) com acoplamento de grau 6	92
4.10	Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.4 (rede fortemente acoplada com múltiplos atrasos) com acoplamento de grau 10	93
4.11	Comparação do número de atualizações por segundo para o modelo 4.5 (proposto por Zhu et al. (2007)) entre o algoritmo RM + <i>Heap</i> e o algoritmo RM + <i>Hashing</i>	95
4.12	Comparação do número de atualizações por segundo para o modelo 4.2 (<i>Colloidal Aggregation Model</i>) entre o algoritmo RM + <i>Heap</i> e o algoritmo RM + <i>Hashing</i>	95

LISTA DE TABELAS

4.1	Reações para modelo da equação 4.2 sem considerar atrasos	85
4.2	Reações para modelo da equação 4.2 considerando atrasos	86

LISTA DE ALGORITMOS

2.1	Um algoritmo para soma dos n primeiros números com complexidade $O(n)$.	40
2.2	Um algoritmo para soma dos n primeiros números com complexidade $O(1)$.	41
2.3	Uma função <i>hash</i> simples	46
2.4	Pseudocódigo para o algoritmo DM	53
2.5	Pseudocódigo para o algoritmo FRM	56
2.6	Pseudocódigo para o algoritmo NRM	57
2.7	Pseudocódigo para o algoritmo MNRM	60
2.8	Pseudocódigo para o algoritmo RM	63
2.9	Pseudocódigo para o algoritmo DMNRM	64
3.1	Pseudocódigo para o algoritmo SNRM	68
3.2	Pseudocódigo para o algoritmo DSNRM	73
3.3	Remoção de produtos em espera em uma lista ordenada por <i>hashing</i>	78
3.4	Inserção de item na lista ordenada por <i>hashing</i>	78

LISTA DE ABREVIACÕES

DG	<i>Dependency Graph.</i>
DDG	<i>Delayed Dependency Graph.</i>
DM	<i>Direct Method.</i>
DMNRM	<i>Modified Next Reaction Method.</i>
DSSA	<i>Delayed Stochastic Simulation Algorithm.</i>
DSNRM	<i>Delayed Simplified Next Reaction Method.</i>
FRM	<i>First Reaction Method.</i>
NRM	<i>Next Reaction Method.</i>
MNRM	<i>Modified Next Reaction Method.</i>
RRG	Rede de Regulação Gênica.
SNRM	<i>Simplified Next Reaction Method.</i>
SSA	<i>Stochastic Simulation Algorithm.</i>

SUMÁRIO

1	INTRODUÇÃO	15
1.1	DEFINIÇÃO DO PROBLEMA	17
1.2	OBJETIVOS	17
1.2.1	Objetivo Geral	17
1.2.2	Objetivos Específicos	18
1.3	HIPÓTESES	18
1.4	ESTRUTURA DA DISSERTAÇÃO	19
2	FUNDAMENTAÇÃO TEÓRICA	20
2.1	CONCEITOS BIOLÓGICOS	20
2.1.1	A Célula e os Organismos Vivos	20
2.1.2	O Ciclo Celular	22
2.1.3	Os Genes e o Genoma	24
2.1.4	Regulação Gênica	26
2.1.5	Redes Bioquímicas	29
2.1.6	Modelos de Redes Bioquímicas	30
2.2	CONCEITOS MATEMÁTICOS E ESTATÍSTICOS	32
2.2.1	Processos Estocásticos	32
2.2.2	Cadeias de Markov e a Equação de Chapman-Kolmogorov	33
2.2.3	A Equação Mestra	35
2.2.4	Método de Monte Carlo	37
2.3	CONCEITOS COMPUTACIONAIS	39
2.3.1	Análise Assintótica de Algoritmos	39
2.3.2	Grafos	41
2.3.3	<i>Heap</i>	44
2.3.4	Tabelas <i>Hash</i>	45
2.4	SIMULAÇÃO DE REDES DE REGULAÇÃO GÊNICA	48
2.4.1	Simulação Determinística	48
2.4.2	Simulação Estocástica	49

2.5	ALGORITMO DE SIMULAÇÃO ESTOCÁSTICA	50
2.5.1	SSA para Modelos Sem Atraso	52
2.5.2	SSA para Modelos Com Atraso	61
3	CONTRIBUIÇÕES	66
3.1	CONTRIBUIÇÕES PARA O DESEMPENHO DAS SIMULAÇÕES SEM ATRASO	66
3.1.1	Versão Simplificada do Algoritmo NRM.....	66
3.2	CONTRIBUIÇÕES PARA O DESEMPENHO DAS SIMULAÇÕES COM ATRASO	69
3.2.1	Grafo de Dependências para Reações com Atraso	69
3.2.2	Versão Simplificada do algoritmo NRM para Tratar Reações com Atrasos ...	72
3.2.3	Estrutura de Dados Lista Ordenada Por <i>Hashing</i> para Controle da Lista de Produtos em Espera.....	74
4	RESULTADOS.....	81
4.1	DDG	81
4.2	NRM SIMPLIFICADO PARA REAÇÕES COM ATRASO	83
4.2.1	Uma Rede Fortemente Acoplada.....	84
4.2.2	Uma Rede Fracamente Acoplada	88
4.2.3	Modelo Fortemente Acoplado em Redes com Múltiplos Atrasos	90
4.3	LISTA ORDENADA POR <i>HASHING</i>	94
5	CONCLUSÕES	96
	REFERÊNCIAS	97

1 INTRODUÇÃO

O estudo da bioquímica mostra como coleções de moléculas inanimadas – que formam os organismos vivos – interagem para manter e perpetuar a vida, sendo que as mesmas leis da química e da física que regem o universo inanimado são válidas para o animado (NELSON et al., 2008).

O artigo de Watson e Crick (1953) apresentou à comunidade científica um modelo da dupla-hélice da estrutura do DNA. Outros estudos como os relacionados à duplicação de DNA (MESELSON; STAHL, 1958) e o sequenciamento de nucleotídeos de uma molécula de RNA (HOLLEY et al., 1965) fazem parte do início da biologia molecular, ramo da bioquímica que estuda a biologia em nível molecular. Como a biologia molecular é uma disciplina que precisa lidar com grande quantidade de dados, é comum a necessidade de apoio computacional em experimentos dessa disciplina, principalmente nos últimos anos com o advento das tecnologias de alto rendimento (PALSSON, 2006).

O estudo da organização morfológica e funcional das células é um campo de pesquisa da bioquímica conhecido como biologia celular. A computação pode auxiliar a biologia celular com modelos e formalismos para descrever e analisar sistemas complexos como as células. Esse ramo de pesquisa interdisciplinar é conhecido como biologia de sistemas. Em biologia de sistemas, conceitos relacionados à biologia celular e molecular, química, física, matemática e estatística são comumente úteis (CARAVAGNA, 2011).

Supondo que um pesquisador esteja estudando como determinado gene é regulado, ele pode criar uma rede bioquímica que indique as reações e as constantes de velocidades das mesmas. Após isso, esse pesquisador pode desejar saber mais sobre o comportamento dessa rede, simulando-a em computadores.

Existem duas formulações para simulação dinâmica de redes de regulação gênica: a determinística e a estocástica. Na formulação determinística, a variação das espécies no tempo pode ser calculada utilizando-se um conjunto de equações diferenciais ordinárias acopladas. Já na formulação estocástica, essa variação pode ser calculada utilizando-se a equação mestra. A função que satisfaz a equação mestra mede a probabilidade de serem encontradas várias populações moleculares a cada instante de tempo (GILLESPIE, 1976).

A simulação determinística pode não ser a mais adequada para algumas redes de

regulação gênica, pois trabalhos como (ARKIN et al., 1998) e (ELOWITZ et al., 2002) indicam que a expressão gênica é estocástica.

A equação mestra é uma equação de ganho e perda que descreve a probabilidade de ocupação dos estados para um determinado sistema ao longo do tempo. Esse tipo de equação possui solução viável somente em redes muito simples. O algoritmo *Stochastic Simulation Algorithm* (SSA) de Gillespie (1977) possibilitou que as simulações estocásticas fossem processadas em computadores. Esse algoritmo, que utiliza técnicas derivadas do método de Monte Carlo, apresenta somente uma trajetória para a rede, enquanto a equação mestra apresenta simultaneamente a probabilidade de todas as trajetórias.

O método de Monte Carlo recebeu esse nome em alusão ao cassino de Monte Carlo, sendo Metropolis e Ulam (1949) a primeira publicação a utilizar esse nome para o método em questão. O método de Monte Carlo utiliza amostras aleatórias para estudar o comportamento de um determinado sistema. O método de Monte Carlo pode ser utilizado para simular qualquer processo de Markov. Em um processo de Markov, o estado futuro depende somente do estado atual, ou seja, é um processo “sem memória”. Os processos de Markov receberam este nome em homenagem ao matemático russo Andrei Andreyevich Markov, devido às suas contribuições para o estudo desses tipos de processos.

Bratsun et al. (2005) propuseram uma generalização do algoritmo de Gillespie que trabalha com eventos bioquímicos com atraso (*delays*). Esse método recebeu o nome *Delayed Stochastic Simulation Algorithm* (DSSA). Em simulações com atraso, alguns processos bioquímicos de múltiplos passos podem ser substituídos por uma simples reação com atraso. Nessa reação, os eventos iniciais são separados do surgimento dos produtos por uma quantidade de tempo. A introdução de atrasos nas simulações estocásticas de redes de regulação gênica contribuiu para melhorar o desempenho computacional dessas simulações.

Para acelerar o SSA, várias abordagens foram propostas, sendo uma das mais importantes a utilização do grafo de dependência (DG do inglês *Dependency Graph*) proposto por Gibson e Bruck (2000). Para a maioria dos modelos de redes de regulação gênica, a implementação desse grafo reduz significativamente o número de recálculos de propensões requerido pelo SSA.

Os conceitos de reações com atraso, DG, processos de Markov e método de Monte Carlo serão explicados no decorrer deste trabalho.

As simulações estocásticas de redes de regulação gênica envolvem múltiplas disciplinas

e o seu objetivo é a obtenção de informações sobre o comportamento dinâmico dessas redes. O presente trabalho está relacionado ao ramo de pesquisa multidisciplinar conhecido como Biologia de Sistemas. Mais especificamente, este trabalho está relacionado a algoritmos exatos de simulação estocástica que não possuem limitação do número de reagentes, ou seja, não serão abordados os algoritmos de aproximação como *tau-leaping* de Gillespie (2001) e os algoritmos que possuem limite para o número de reagentes de uma reação, como os de propensidades parciais de Ramaswamy et al. (2009) e Ramaswamy e Sbalzarini (2010, 2011).

Os códigos fontes dos algoritmos apresentados neste trabalho podem ser obtidos mediante solicitação aos autores.

1.1 DEFINIÇÃO DO PROBLEMA

Os algoritmos de simulação estocástica de redes de regulação gênica possuem um desempenho que impedem o seu uso para simulações que envolvam milhares de reações, como as necessárias para simular um organismo completo como uma bactéria. Mesmo com diversos trabalhos voltados ao desempenho desses algoritmos, dos quais destacam-se os artigos de Gibson e Bruck (2000) e McCollum et al. (2006), muito trabalho ainda precisa ser feito para que essas simulações possam continuar auxiliando os pesquisadores em novas descobertas sobre os organismos vivos.

1.2 OBJETIVOS

Como visto anteriormente, a simulação estocástica de uma rede bioquímica envolve múltiplas disciplinas. O presente trabalho possui como foco a obtenção de melhorias em algumas tecnologias relacionadas a esse problema. Para ser mais específico, o trabalho apresentado possui o seu foco em desempenho de algoritmos de simulação estocástica de redes de regulação gênica.

1.2.1 OBJETIVO GERAL

Melhorar o desempenho das simulações de redes de regulação gênica com atraso.

1.2.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral proposto, os objetivos específicos são:

1. Desenvolver um grafo de dependências que apresente um melhor desempenho que o DG. Esse novo grafo seria utilizado em simulações de redes de regulação gênica que trabalham com reações com atraso.
2. Criar uma nova estrutura de dados, que apresente um melhor desempenho que a *heap*, para tratar os produtos em espera.
3. Propor uma nova versão do algoritmo *Modified Next Reaction Method* (MNRM) que trabalhe com tempos absolutos, de forma que esse novo algoritmo tenha um crescimento logarítmico da complexidade no pior caso, ao invés de um crescimento linear.

1.3 HIPÓTESES

Cada objetivo específico possui uma hipótese. Todas as hipóteses foram testadas neste trabalho, mas não à exaustão. As simulações de redes de regulação gênica são processos complexos e cada algoritmo que possui essa finalidade é geralmente construído de forma que o seu desempenho seja otimizado para modelos com determinada característica. Logo, dificilmente um único algoritmo é capaz de apresentar-se como o melhor para qualquer tipo de rede. Devido a isso, é comum as publicações apresentarem melhorias com testes aplicados a apenas alguns tipos de modelos e apontarem uma tendência de comportamento para o algoritmo que apresentam.

Nesse trabalho, as hipóteses são:

1. Para o desenvolvimento de um novo grafo de dependências para simulações de redes de regulação gênica com atraso, é provável que um melhor desempenho seja obtido utilizando-se um grafo cujas arestas apontem das espécies para as reações, ao invés de apontarem das reações para as reações.
2. Quanto à lista de produtos em espera, a estrutura de dados mais utilizada para o seu controle é a *heap*. Mesmo sendo uma estrutura de dados que apresenta um bom desempenho para o problema, acredita-se que a *heap* pode ser superada por uma

estrutura que utilize uma estratégia do tipo *hash* para distribuição dos produtos em espera e essa nova estrutura não deve ter um impacto negativo no desempenho do sistema devido a algumas especificidades do problema, relatados na subseção 3.2.3.

3. Relacionado ao algoritmo MNRM, é possível fazer uma versão desse algoritmo que trabalhe com tempos absolutos ao invés de tempos relativos e, com isso, melhorar o desempenho desse algoritmo.

1.4 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado em cinco capítulos, sendo as referências adicionadas ao final do último capítulo. No capítulo 2 são abordados os principais conceitos necessários ao entendimento dos demais capítulos. O capítulo 3 apresenta os métodos propostos e as principais diferenças em relação a outros métodos já publicados. O capítulo 4 apresenta alguns resultados obtidos em simulações de redes de regulação gênica, sendo utilizado um novo algoritmo proposto e outros algoritmos. O capítulo 5 conclui o trabalho e apresenta algumas possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

As simulações de redes de regulação gênica, que fazem parte de uma área multidisciplinar conhecida como biologia de sistemas, envolvem conceitos de diferentes áreas do conhecimento. Este capítulo apresenta os principais conceitos necessários ao entendimento do presente trabalho. A seção 2.1 apresenta alguns conceitos biológicos. A seção 2.2 apresenta conceitos matemáticos relacionados às simulações de redes de regulação gênica. Na seção 2.3, alguns conceitos computacionais utilizados nessa dissertação são explicados. A seção 2.4 apresenta as simulações determinísticas e as estocásticas. O fechamento do capítulo é feito na seção 2.5 que apresenta os principais algoritmos de simulação estocástica de redes de regulação gênica.

2.1 CONCEITOS BIOLÓGICOS

Entre os conceitos biológicos mais importantes para facilitar a compreensão do presente trabalho, destacam-se o entendimento da célula e da formação dos organismos vivos, o conhecimento do ciclo celular, o que são genes e genoma, o que é conhecido por regulação de genes e o que é uma rede bioquímica. As próximas seções apresentam esses conceitos com mais detalhes.

2.1.1 A CÉLULA E OS ORGANISMOS VIVOS

De forma semelhante ao átomo, que é considerado a unidade fundamental das estruturas químicas, a célula é considerada a estrutura fundamental dos organismos vivos. Tanto a matéria viva quanto o mundo inorgânico são compostos pelos mesmos elementos, porém existem diferenças em suas organizações. Os organismos vivos tendem a manter uma ordem nas transformações químicas para que as estruturas e as funções biológicas não sejam alteradas, porém essa não é a sua principal característica. Nos seres vivos existe o fenômeno da hereditariedade, onde o organismo parental transfere as informações específicas das características que seus descendentes devem ter. Esse fenômeno diferencia a vida de outros processos como a queima de uma vela e o desenvolvimento de um cristal, onde também são geradas estruturas ordenadas (ALBERTS et al., 2010; ROBERTIS;

HIB, 2001). Segundo Karp (2010), cada célula possui uma alta complexidade e organização, uma programação genética e um meio de utilizá-la, a capacidade de reproduzir mais de si mesma, a capacidade de adquirir e utilizar energia, a capacidade de executar uma variedade de reações químicas e envolver-se em atividades mecânicas. Além disso, as células respondem a estímulos, possuem auto-regulação e evoluem.

Os organismos vivos são formados por uma ou mais células, sendo que os unicelulares são a maior parte dos organismos vivos conhecidos. Os organismos multicelulares são formados a partir da divisão de uma única célula. Essa única célula possui toda a programação necessária para gerar como resultado um organismo complexo como o humano que, formado por mais de 10^{13} células, possui diversos grupos de células que realizam funções especializadas e estão ligadas por um complexo sistema de comunicação (ALBERTS et al., 2010).

Para classificarmos os organismos vivos, é necessário considerar a existência de algumas armadilhas. A forma mais precisa de realizar essa classificação é através das sequências de DNA desses organismos, trabalho que vem destacando três grandes domínios: bactérias, arqueobactérias e eucariotos. Uma bactéria é um organismo unicelular e procarionte, e, dessa forma, sua célula é desprovida de envoltório nuclear. Os eucariotos são organismos unicelulares ou pluricelulares cujas células possuem envoltório nuclear. O exterior de uma arqueobactéria é muito semelhante ao de uma bactéria, porém as arqueobactérias possuem a maquinaria de manipulação genética semelhante à dos eucariotos e o metabolismo e a conversão de energia semelhante à das bactérias (ALBERTS et al., 2010; NELSON et al., 2008).

Na figura 2.1 são mostradas uma célula eucariótica e uma procariótica, destacando-se suas semelhanças e diferenças.

É conhecido pela comunidade científica que não existem nada nos organismos vivos que desobedeça às leis da química e da física. A química da vida está baseada fundamentalmente em compostos de carbono, sendo que as células possuem 70% de água em sua composição. Dessa forma, a quase totalidade das reações químicas das quais a vida depende acontece em soluções aquosas. Mesmo assim, a mais simples das células possui uma química muito mais complexa do que qualquer outro sistema químico inorgânico conhecido (ALBERTS et al., 2010).

A segunda lei da termodinâmica diz que no universo, ou em qualquer sistema isolado,

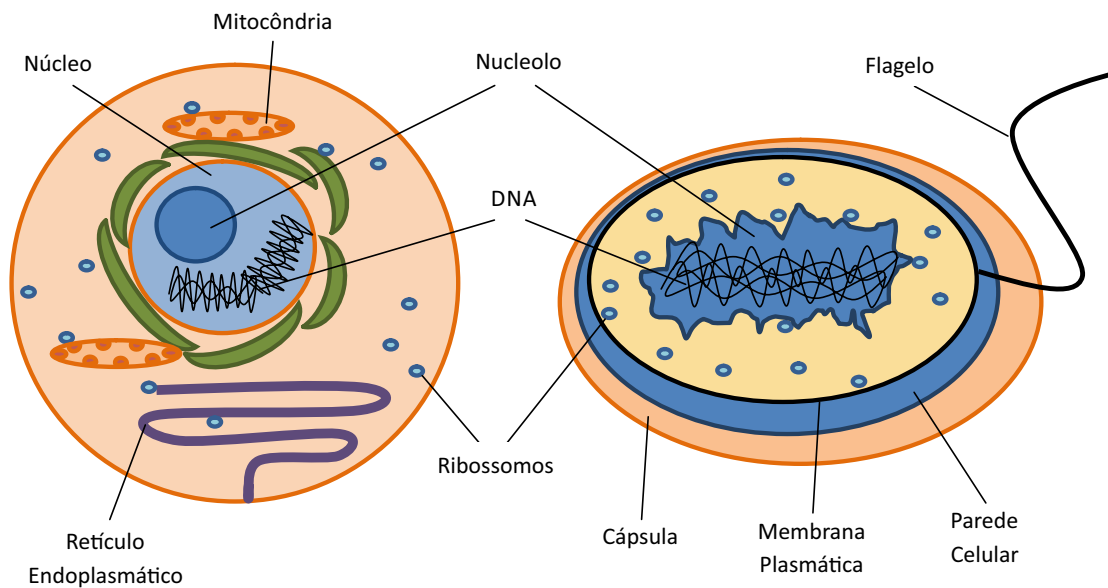


Figura 2.1: Representação de uma célula de um organismo eucarioto (esquerda) e de um organismo procarioto (direita). Note que a célula procariótica não possui núcleo.

o grau de desordem somente pode crescer. De forma inocente, alguém pode pensar que ao sobreviverem, crescerem e formarem organismos complexos, as células estão gerando ordem e, dessa forma, desafiando a segunda lei da termodinâmica. Isso está incorreto, pois a ordem gerada em uma célula é mais que compensada pela desordem gerada exteriormente à mesma pela liberação de energia térmica para o ambiente. Dessa forma, a entropia total aumenta, ou seja, o sistema como um todo fica menos ordenado, exatamente como a segunda lei da termodinâmica determina (ALBERTS et al., 2010).

2.1.2 O CICLO CELULAR

É de conhecimento da comunidade científica que uma nova célula só pode ser formada pela divisão de outra célula. A duplicação de componentes celulares e a divisão celular, também conhecido como ciclo celular, é o mecanismo pelo qual todos os organismos vivos se reproduzem (ALBERTS et al., 2010).

O ciclo celular inicia-se no momento exato onde uma nova célula é criada, passa pelo momento no qual esta célula duplica o seu material genético e termina quando o material celular duplicado é distribuído entre as células filhas. Durante o ciclo celular, a célula passa por dois períodos importantes: a interfase e a divisão celular. É na interfase que ocorrem as funções mais importantes do ciclo celular e isso é válido tanto para o núcleo quanto para o citoplasma. A interfase é o período mais longo pela qual todas as células

passam, mas é importante destacar que as células nervosas, depois do nascimento, não se dividem. Dessa forma, as células nervosas permanecem em interfase durante toda a vida do organismo (ROBERTIS; HIB, 2001).

A interfase pode compreender os períodos G1, S e G2. O ciclo de divisão celular é composto pela interfase e pela fase mitótica (M), que inclui a mitose e a divisão celular. A fase S recebeu essa letra devido à inicial “S” do nome “síntese de DNA”. É na fase S que o DNA celular é duplicado. A fase M é onde ocorre a Mitose. É na fase M que o material celular é distribuído entre as células-filhas. O G das fases G1 e G2 significa gap em inglês, palavra que pode ser traduzida como intervalo. A fase G2 é o tempo que transcorre entre o final da fase S e o início da fase M (ver figura 2.2). Durante a fase G2 a célula possui o dobro do material genético que tinha na fase G1, sendo que após a fase M o material genético é dividido em porções virtualmente iguais entre as células-filhas, que iniciam o seu ciclo celular na fase G1 (ROBERTIS; HIB, 2001).

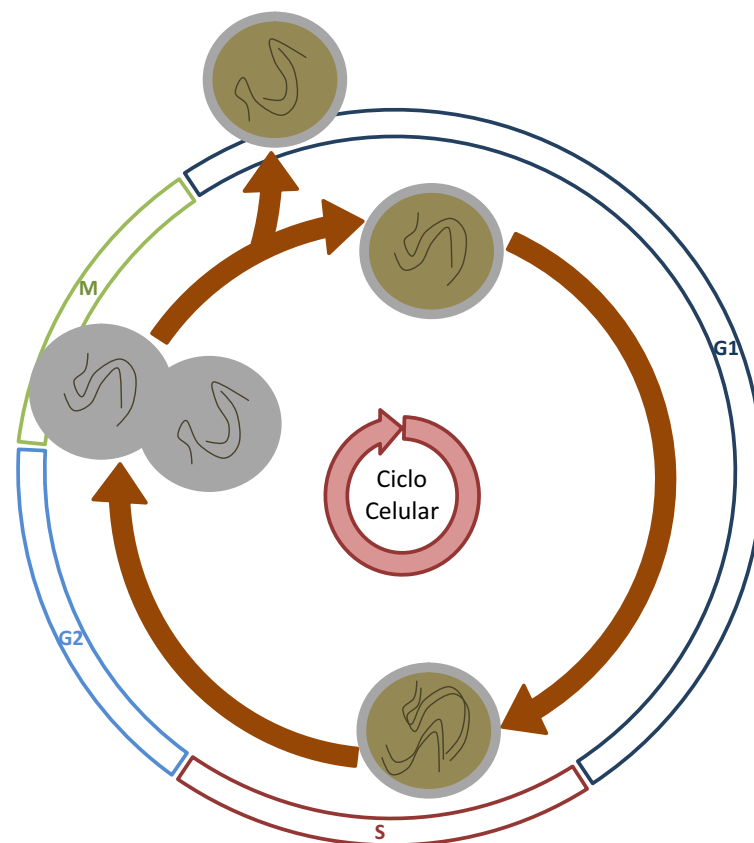


Figura 2.2: O ciclo celular.

Enquanto em organismos unicelulares cada divisão celular dá origem a um novo organismo, em organismos multicelulares são necessárias sequências longas e complexas de divisões celulares para que um novo organismo seja produzido. Em células eucarióticas,

existe uma complexa rede de proteínas reguladoras, conhecida como sistema de controle do ciclo celular. Dentro da célula, o sistema de controle do ciclo celular retarda eventos posteriores até que eventos anteriores sejam completados. Em animais multicelulares, esse sistema responde a sinais de outras células, podendo estimular ou bloquear a produção de novas células (ALBERTS et al., 2010).

O início e o final das fases do ciclo celular são determinados por mecanismos que coordenam os processos de síntese no citoplasma e no núcleo celular. Esses mecanismos são controlados por uma complexa rede de sinais. Graças a essa regulação, as células que desaparecem por envelhecimento ou por morte programada podem ser substituídas, um organismo pode ter o seu crescimento controlado e algumas feridas podem ser recuperadas (ROBERTIS; HIB, 2001).

2.1.3 OS GENES E O GENOMA

Existem dois tipos de ácidos nucleicos: o ácido ribonucleico (RNA) e o ácido desoxirribonucleico (DNA) (KARP, 2010). O DNA é composto por duas cadeias polipeptídicas formadas por quatro subunidades (ver figura 2.3). Essas subunidades são conhecidas como nucleotídeos, que são compostos por um açúcar, a desoxirribose, ligada a um único grupo fosfato e uma base. Essa base pode ser a adenina (A), a citosina (C), a guanina (G) ou a timina (T). A extremidade ligada ao grupo fosfato é conhecida como extremidade 5' enquanto outra extremidade é conhecida como extremidade 3' (ALBERTS et al., 2010).

Como o nucleotídeo A pode parear-se eficientemente somente com o T e C apenas com G, as fitas de DNA são complementares. Dessa forma, antes da divisão celular, essas fitas são separadas e servem como molde para a criação de seu complemento. Antes da mitose, o DNA e todas as outras estruturas celulares internas devem estar duplicadas. O DNA está localizado no núcleo celular dos organismos eucariotos. Já nas bactérias e nas arqueobactérias o DNA encontra-se misturado no citoplasma celular (ALBERTS et al., 2010; ROBERTIS; HIB, 2001).

O gene, que é formado por sequências específicas de ácidos nucleicos, é a unidade fundamental da hereditariedade (KARP, 2010). Um gene possui a informação necessária para sintetizar uma molécula de RNA e, se a molécula sintetizada corresponder a um RNA mensageiro, a partir deste RNA construir uma proteína (ROBERTIS; HIB, 2001).

O RNA possui uma estrutura semelhante à do DNA, exceto por ser formado por uma

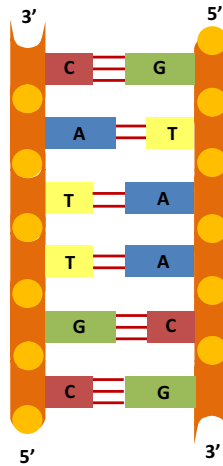


Figura 2.3: Fita dupla de DNA. O DNA é composto por duas fitas unidas por ligações de hidrogênio entre os quatro tipos de bases pareadas. Nesta figura, o DNA está sendo mostrado em forma plana, sendo que na realidade ele é torcido, formando uma dupla-hélice.

só cadeia de nucleotídeos e por possuir o açúcar ribose no lugar do desoxirribose e a base uracila (U) no lugar da timina (T).

Existem três classes principais de RNA que participam da síntese proteica: RNA mensageiro, RNA ribossômico e RNA transportador. O RNA mensageiro leva a informação genética que estabelece a sequência de aminoácidos das proteínas, sendo que essa informação é copiada do DNA. O RNA ribossômico representa 50% da massa do ribossomo, que é a estrutura de sustentação molecular para as reações químicas que ocorrem durante a síntese proteica. O RNA transportador identifica e transporta os aminoácidos até o ribossomo (ROBERTIS; HIB, 2001).

O número de genes de um organismo está relacionado à complexidade desse mesmo organismo. Enquanto existem aproximadamente 25 mil genes humanos, algumas bactérias simples possuem aproximadamente 500 genes (ALBERTS et al., 2010).

Os genes são os elementos que contêm as informações que determinam as características de uma espécie e as de um indivíduo. A informação hereditária é transmitida de uma célula para as suas células-filhas durante a divisão celular e de uma geração de determinado organismo a outra, por meio de células reprodutoras (ROBERTIS; HIB, 2001).

A informação genética consiste principalmente de instruções para a produção de proteínas, sendo que as proteínas são macromoléculas que realizam a maioria das funções celulares. As proteínas são parte das estruturas celulares, regulam a expressão gênica, permitem que uma célula se mova e possibilitam a comunicação entre células (ALBERTS

et al., 2010).

Nas células, a transmissão da informação genética é feita pelos cromossomos, que são compostos por DNA e proteínas. Um único cromossomo é composto por diversos genes. As bactérias e arqueobactérias carregam os seus genes em uma única molécula de DNA (ALBERTS et al., 2010; ROBERTIS; HIB, 2001).

Se fosse possível ligar as fitas de DNA dos cromossomos humanos e esticá-las de forma que cada molécula ficasse uma do lado da outra, o tamanho da sequência de DNA seria de aproximadamente dois metros, porém o tamanho do núcleo de uma célula humana é de aproximadamente $6 \mu\text{m}$. Para que o DNA humano caiba perfeitamente no núcleo de sua célula, esse DNA é compactado em uma série de 46 de cromossomos. A compressão do DNA em cada um desses cromossomos é realizada por proteínas que sucessivamente enrolam e dobram o DNA em níveis de organização cada vez mais altos (ALBERTS et al., 2010).

Genoma é o nome dado à série completa de informações de DNA de um determinado organismo. O livro de Alberts et al. (2010) contém mais de mil e setecentas páginas, porém a sequência completa dos nucleotídeos do genoma humano escrita utilizando-se as letras que representam cada nucleotídeo (A, C, G e T) preencheria as páginas de mais de mil livros deste. O número total de proteínas distintas cujas instruções estão presentes no genoma humano é de cerca de 24 mil, sendo que a cada divisão celular, a célula precisa copiar o seu genoma e passá-lo para as células filhas. É notável a necessidade de ferramentas computacionais de alto desempenho para lidar com essa grande quantidade de dados.

O grande volume de dados obtidos pelo sequenciamento do genoma humano indica um caminho de grande dificuldade e enormes oportunidades para os pesquisadores. Todavia, como existem sequências de DNA que foram conservadas na evolução, o domínio da biologia molecular de uma simples bactéria já permitiria um grande avanço na área e até mesmo no entendimento de organismos mais complexos.

2.1.4 REGULAÇÃO GÊNICA

A síntese de moléculas de RNA utilizando moléculas de DNA como molde é um processo denominado transcrição. O mRNA é utilizado diretamente como molde para a síntese de proteína em um processo denominado tradução. Todas as células expressam a sua

informação genética seguindo o fluxo de DNA para RNA e deste para proteína (ALBERTS et al., 2010). Esse princípio é denominado dogma central da biologia molecular e está representado na figura 2.4.

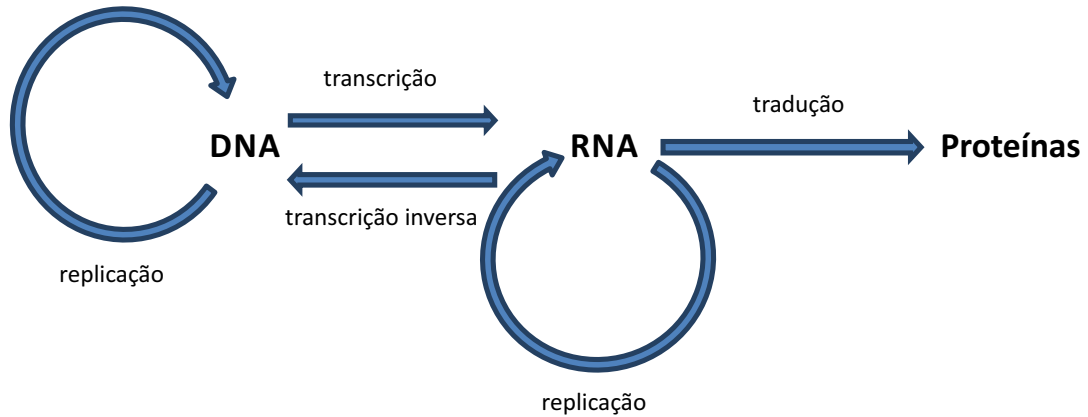


Figura 2.4: Dogma central da biologia molecular estendido, adaptado de (NELSON et al., 2008).

A transcrição é mediada e regulada por interações entre proteínas, DNA e RNA, especialmente aquelas que envolvem proteínas componentes da RNA polimerase. A RNA polimerase se liga ao DNA em uma região conhecida como região promotora e, dessa forma, a transcrição é iniciada. A regulação do início da transcrição frequentemente implica em mudanças na forma que o RNA polimerase interage com um promotor (ROBERTIS; HIB, 2001).

A afinidade de ligação entre o RNA polimerase e o DNA é afetada pela considerável variação das sequências de nucleotídeos dos promotores. Na ausência de proteínas reguladoras, essa variação na sequência de nucleotídeos pode afetar a frequência da iniciação da transcrição em um fator de 1.000 ou mais (NELSON et al., 2008).

Ao menos três tipos de proteínas regulam o início da transcrição mediada e regulada pela RNA polimerase. São elas: fatores de especificidade, repressores e ativadores. As proteínas de fatores de especificidade alteram a especificidade do RNA polimerase de um promotor ou de um conjunto de promotores. As proteínas repressoras impedem o RNA polimerase de acessar o promotor. As proteínas ativadoras melhoram a interação entre o RNA polimerase e o promotor (NELSON et al., 2008).

Em bactérias, os genes podem ser agrupados no cromossomo e são transcritos conjuntamente. Esse é o mecanismo simples que coordena a regulação de genes de uma bactéria. Muitos RNAs mensageiros bacterianos codificam duas ou mais proteínas, ou seja, pos-

suem múltiplos genes em um único transcrito, e um único promotor inicia a transcrição para todos os genes desse agrupamento. Esse agrupamento de genes, o seu promotor e algumas sequências adicionais que trabalham junto na regulação representam uma única unidade de transcrição, conhecida como operon (ALBERTS et al., 2010; NELSON et al., 2008). A figura 2.5 apresenta o Operon Trp.

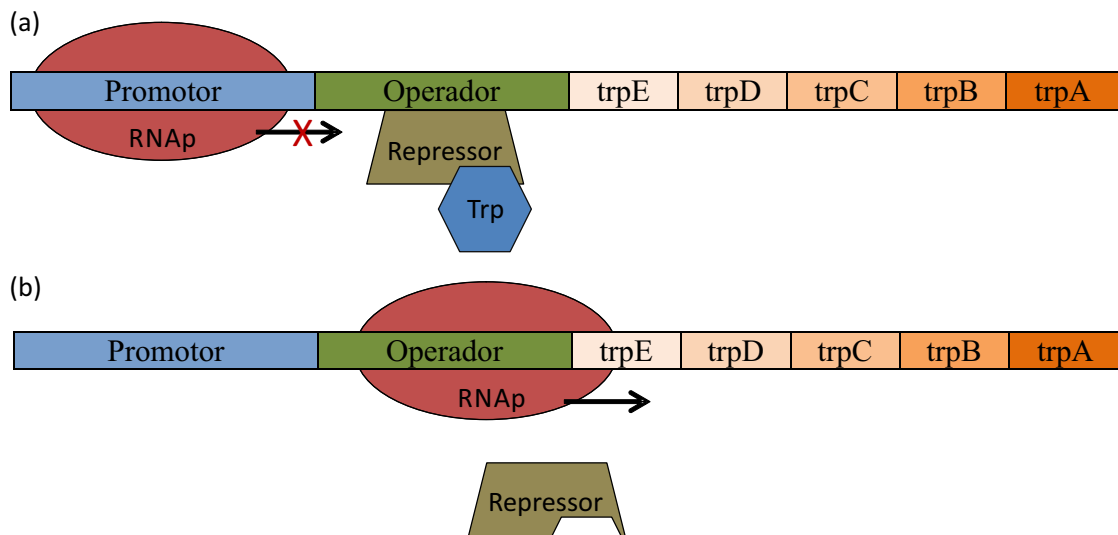


Figura 2.5: Operon Trp. (a) Quando o *triptofano* (*Trp*) está presente, o repressor se liga ao operador e a transcrição é bloqueada. (b) Na ausência de *triptofano*, o repressor se dissocia do operador e a síntese de RNA prossegue.

Já no processo de tradução, como existem somente quatro tipos de nucleotídeos no RNA mensageiro e vinte tipos distintos de aminoácidos em uma proteína, não existe uma correspondência direta entre um nucleotídeo do RNA e um aminoácido da proteína. Dessa forma, a sequência de nucleotídeos de uma molécula de RNA mensageiro é lida em grupos de três nucleotídeos, denominados códon. Um códon pode especificar um aminoácido ou o final da tradução (ALBERTS et al., 2010).

A tradução do RNA mensageiro em proteína ocorre através de moléculas adaptadoras que se ligam ao RNA mensageiro em uma extremidade e a um aminoácido em outra. Essa molécula adaptadora é conhecida como RNA transportador (ROBERTIS; HIB, 2001).

A síntese proteica é realizada no ribossomo, que é formado por mais de 50 proteínas ribossomais diferentes e diversas moléculas de RNA ribossomais. Um ribossomo é composto por uma subunidade grande e uma pequena. As duas subunidades do ribossomo permanecem separadas quando a síntese de proteínas não está ativa. Essas extremidades

se unem sobre uma molécula de RNA mensageiro para iniciar a síntese de uma proteína. Conforme os códons do RNA mensageiro encontram os sítios ativos dos ribossomos, os códons são traduzidos em uma sequência de aminoácidos, usando os RNA transportadores para adicionar cada aminoácido na sequência correta à extremidade da cadeia polipeptídica que está sendo formada. Quando um códon de terminação é encontrado, o ribossomo libera a proteína e suas duas subunidades separam-se novamente (ALBERTS et al., 2010).

2.1.5 REDES BIOQUÍMICAS

O estudo de várias redes de interação entre moléculas é fundamental para o entendimento do comportamento celular. Algumas dessas redes são: redes de interações entre proteínas, redes metabólicas, redes de sinalização e redes reguladoras transcricionais. O comportamento de uma célula é resultado da interconexão entre essas redes, ou seja, de uma rede de redes de interação (BARABASI; OLTVAI, 2004).

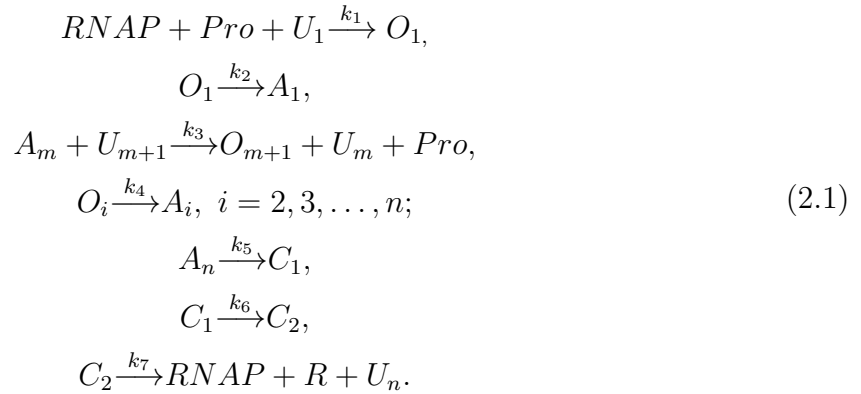
As redes reguladoras transcricionais, foco das simulações do presente trabalho, também são conhecidas como redes de regulação gênica. Os produtos dos genes possibilitam a sobrevivência das células enquanto as redes de regulação gênica controlam os padrões de tempo e quantidade desses produtos. Dessa forma, vários processos celulares são afetados por essas redes (HOVATTA et al., 2005).

Todos os processos da vida, incluindo o ciclo celular e o metabolismo, dependem dos genes e estes são regulados por seus produtos ou por produtos de outros genes, formando assim uma rede de regulação gênica. Dessa forma, entender a dinâmica dessas redes pode auxiliar na compreensão de doenças causadas pela desregulação desses processos celulares; além disso, predições precisas do comportamento dessas redes podem tornar projetos biotecnológicos mais baratos e rápidos (KARLEBACH; SHAMIR, 2008).

Como consequência das inovações nos métodos experimentais, os pesquisadores estão trabalhando com redes extremamente complexas, o que gera a necessidade de analisar e integrar grande quantidade de dados experimentais. É nesse cenário que os métodos computacionais podem auxiliar os pesquisadores nas descobertas de padrões e na predição do comportamento dessas redes sobre certas condições (KARLEBACH; SHAMIR, 2008). Alguns desses métodos serão abordados no presente trabalho.

2.1.6 MODELOS DE REDES BIOQUÍMICAS

Levando em consideração um organismo procarioto, Roussel e Zhu (2006) apresentaram um modelo para uma rede de transcrição:

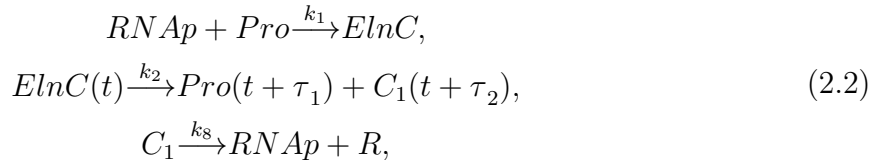


No modelo do conjunto de equações 2.1, RNAP representa a enzima RNA polimerase, Pro representa a região promotora de um gene ao qual o RNAP se liga. Supondo a existência de um molde de DNA com uma região de ligação para o promotor e n regiões de nucleotídeos, considerando a primeira região como a região de ligação do promotor, uma região i é representada por U_i se ela não está ocupada por um RNA polimerase. Uma região é representada por O_i se está ocupada por um RNA polimerase e por A_i se o RNA polimerase que está ligada a essa região foi ligada a um nucleosídeo trifosfato conhecido como NTP e está pronto para realizar a sua ação catalítica. Ainda no modelo supracitado, C_1 representa um complexo de terminação, gerado quando o RNA polimerase identifica a sequência de terminação. O estado C_2 representa um estado de pausa e o RNA transcrito é representado por R.

Os modelos de redes de controle genético no nível de detalhes do conjunto de equações 2.1 tornam-se complexos à medida que mais genes são adicionados. É possível diminuir a complexidade dos modelos reduzindo o processo de múltiplos passos de alongamento para um simples processo com atraso, que concentra somente a entrada e saída do processo. Dessa forma, é possível tratar processos bioquímicos de múltiplos passos, como a transcrição e a tradução – sendo que cada um desses processos envolvem um grande número de reações elementares – como reações com atraso, na qual os eventos iniciais são separados do surgimento de produtos por um intervalo de tempo (ROUSSEL; ZHU, 2006).

A rede bioquímica representada pelo conjunto de equações 2.1 pode ser substituída pela rede que utiliza atrasos, representada no conjunto de equações 2.2. Nessa rede, a

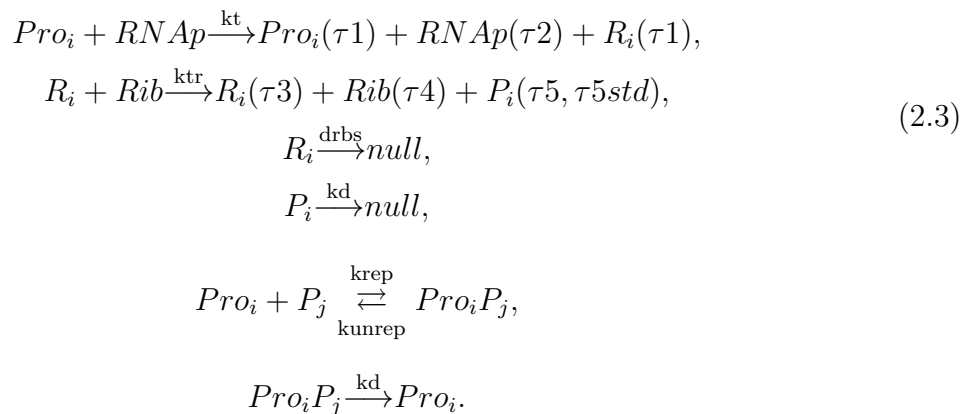
inicialização é reduzida à ligação do RNA polimerase à sequência promotora, o que forma o complexo de alongamento ElnC. Já o processo de alongamento é representado por uma simples reação com atraso.



Nos modelos dos conjuntos de equações 2.1 e 2.2, k_1, k_2, \dots, k_8 são constantes de velocidades estocásticas que possuem a dimensão de acordo com o inverso da molaridade da reação e do tempo.

As redes representadas pelos conjuntos de equações 2.1 e 2.2 não possuem regulação. Um bom exemplo de rede de regulação de genes é conhecida como *Toggle Switch* e foi sintetizada experimentalmente em *E. coli* (GARDNER et al., 2000).

A *Toggle Switch* é uma rede gênica simples composta por dois genes. Nessa rede, cada gene pode ser inibido pelo repressor transcrito pelo outro gene (ZHU et al., 2007). O modelo do *Toggle Switch*, baseado no modelo de Ribeiro (2010) encontra-se representado pelo conjunto de equações 2.3. Para essas reações, considere $i, j = 1, 2$ com $i \neq j$ quando ambos os índices estiverem presentes.



Para o modelo da rede *Toggle Switch* apresentado, considere que Pro_i representa o promotor do gene i , RNAp representa uma RNA polimerase, Rib um ribossomo e R_i é o local de ligação do ribossomo de cada RNA mensageiro. Os tempos de atraso estão representados por $\tau_1, \tau_2, \dots, \tau_5$. Já τ_5std representa uma variação possível para o atraso, que deve obedecer à distribuição normal. A palavra *null* à direita de uma reação significa que os reagentes dessa reação serão degradados e que o seu produto não é relevante para

o modelo.

2.2 CONCEITOS MATEMÁTICOS E ESTATÍSTICOS

A simulação estocástica de redes de regulação gênica interliga múltiplas disciplinas. Dentre essas disciplinas, a conceituação estatístico-matemática do problema é muito importante. Nas próximas subseções serão abordados os processos estocásticos e a subclasse desses processos conhecido como processos de Markov. Será abordado também a equação de Chapman-Kolmogorov, a equação mestra e os métodos de Monte Carlo.

2.2.1 PROCESSOS ESTOCÁSTICOS

A palavra estocástica significa aleatório. Enquanto um modelo determinístico prediz um único resultado a partir de um determinado conjunto de circunstâncias, um modelo estocástico prevê um conjunto de possíveis resultados ponderados por suas probabilidades (TAYLOR; KARLIN, 1998).

Uma variável estocástica, ou número aleatório, é um objeto X definido por um conjunto de valores possíveis, também conhecido como conjunto de estados, e uma distribuição de probabilidades sobre este conjunto. Esse conjunto pode ser discreto, como o número de moléculas de um componente de uma mistura de elementos químicos reagentes, pode ser contínuo, como a diferença potencial entre os pontos finais de uma resistência elétrica, ou pode ser multidimensional, como a coleção de todos os números de moléculas dos diversos componentes de uma mistura reagente (TAYLOR; KARLIN, 1998; KAMPEN, 2007).

No caso de um conjunto de valores possíveis contínuo e unidimensional, a distribuição da probabilidade de uma variável estocástica é dada pela função não negativa $P(x) dx$, normalizada de forma que $\int P(x) dx = 1$, sendo $P(x) dx$ a probabilidade de X ter um valor entre x e $x + dx$ (KAMPEN, 2007).

Um processo estocástico é uma função de duas variáveis, sendo uma variável estocástica X e outra variável representando o tempo t , conforme representado na fórmula: $Y_X(t) = f(X, t)$. Ao inserir para X um de seus possíveis valores x , uma função comum de t , que pode ser chamada de função de amostra, é obtida: $Y_x(t) = f(x, t)$.

2.2.2 CADEIAS DE MARKOV E A EQUAÇÃO DE CHAPMAN-KOLMOGOROV

Processos de Markov é uma subclasse dos processos estocásticos com estados discretos e que possuem a propriedade de Markov. A propriedade de Markov é definida como:

$$f(y_n, t_n | y_1, t_1; \dots; y_{n-1}, t_{n-1}) = f(y_n, t_n | y_{n-1}, t_{n-1}). \quad (2.4)$$

Dessa forma, os processos classificados como processos de Markov são processos sem memória, ou seja, somente o estado presente é relevante para a predição dos estados futuros (TAYLOR; KARLIN, 1998; KAMPEN, 2007).

Os processos de Markov são totalmente determinados por duas funções: $f(y_1, t_1)$ e $f(y_2, t_2 | y_1, t_1)$ (KAMPEN, 2007). Dessa forma, para $t_1 < t_2 < t_3$:

$$f(y_1, t_1; y_2, t_2; y_3, t_3) = f(y_3, t_3 | y_1, t_1; y_2, t_2) \cdot f(y_2, t_2 | y_1, t_1) \cdot f(y_1, t_1). \quad (2.5)$$

Pela propriedade de Markov, $f(y_3, t_3 | y_1, t_1; y_2, t_2) = f(y_3, t_3 | y_2, t_2)$, então temos:

$$f(y_1, t_1; y_2, t_2; y_3, t_3) = f(y_3, t_3 | y_2, t_2) \cdot f(y_2, t_2 | y_1, t_1) \cdot f(y_1, t_1) \quad (2.6)$$

A propriedade supracitada torna tratáveis os processos de Markov e, com ela, já é possível caminhar para o entendimento da equação de Chapman-Kolmogorov. Para isso, integrando a equação 2.6 sobre y_2 obtêm-se:

$$f(y_1, t_1; y_3, t_3) = f(y_1, t_1) \int_{-\infty}^{\infty} f(y_3, t_3 | y_2, t_2) f(y_2, t_2 | y_1, t_1) dy_2 \quad (2.7)$$

Dividindo-se ambos os lados por $f(y_1, t_1)$ ou utilizando

$f(y_1, t_1; y_3, t_3) = f(y_3, t_3 | y_1, t_1) \cdot f(y_1, t_1)$, chega-se à equação de Chapman-Kolmogorov:

$$f(y_3, t_3 | y_1, t_1) = \int_{-\infty}^{\infty} f(y_3, t_3 | y_2, t_2) f(y_2, t_2 | y_1, t_1) dy_2 \quad (2.8)$$

Como a equação 2.8 é uma função de transição de probabilidade, é possível reescrever

a equação de Chapman-Kolmogorov da seguinte forma (TAYLOR; KARLIN, 1998):

$$P(y_3, t_3|y_1, t_1) = \int_{-\infty}^{\infty} P(y_3, t_3|y_2, t_2) P(y_2, t_2|y_1, t_1) dy_2. \quad (2.9)$$

Já em processos de Markov estacionários, a probabilidade de transição depende somente do intervalo de tempo, o que permite escrever a equação de Chapman-Kolmogorov da seguinte forma (KAMPEN, 2007):

$$P(y_3|y_1, \tau + \tau') = \int_{-\infty}^{\infty} P(y_3|y_2, \tau') P(y_2|y_1, \tau) dy_2 \quad (2.10)$$

A solução da equação de Chapman-Kolmogorov pode fornecer uma descrição completa para qualquer processo de Markov, porém nenhuma solução geral para esta equação é conhecida (KAMPEN, 2007).

Uma subclasse dos processos de Markov é conhecida como Cadeias de Markov, tendo as seguintes propriedades (KAMPEN, 2007):

1. O alcance de Y é um conjunto de estados discretos;
2. A variável de tempo é discreta e pode assumir somente valores inteiros;
3. O processo precisa ser estacionário ou precisa ser homogêneo de forma que a probabilidade de transição dependa somente da diferença de tempo.

Para facilitar o entendimento sobre a tratabilidade das cadeias de Markov, apresentaremos um exemplo de cadeia de Markov homogênea, ou seja, uma cadeia cujas probabilidades de transição são constantes no tempo. Para esse exemplo, podemos supor a existência de um pequeno país com fluxos migratórios de pessoas entre três destinos principais: campo, cidades capitais e cidades do interior. Para esse exemplo, consideraremos que $\frac{1}{3}$ da população reside no campo, que $\frac{1}{6}$ da população reside na capital e que $\frac{1}{2}$ da população reside em cidades do interior. Se utilizarmos t para contagem do tempo, no tempo $t = 0$ (P_0) temos o vetor de estados iniciais, representando a proporção de pessoas em cada destino ou estado:

$$P_0 = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{6} \\ \frac{1}{2} \end{bmatrix}. \quad (2.11)$$

Para o exemplo dado, a probabilidade de migração de um estado para outro depende somente do estado atual e é constante no tempo. Logo, trata-se de uma cadeia de Markov. Considerando t_{ij} a probabilidade de alteração do estado j para o estado i em uma unidade de tempo, temos a matriz de transição:

$$T = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \end{bmatrix}. \quad (2.12)$$

Após uma unidade de tempo, a divisão do estado será obtida pelo produto $P_1 = TP_0$. Ao término de uma unidade de tempo, temos:

$$P_1 = TP_0 = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{3} \\ \frac{1}{6} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{18} \\ \frac{7}{18} \\ \frac{1}{3} \end{bmatrix}. \quad (2.13)$$

Como T é constante, após k unidades de tempo, teremos $P_k = TP_{k-1} = T^2P_{k-2} = T^kP_0$. Dessa forma, podemos calcular P_2 conhecendo somente P_0 e T da seguinte forma:

$$P_2 = T^2P_0 = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{3} \\ \frac{1}{6} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{7}{27} \\ \frac{25}{54} \\ \frac{5}{18} \end{bmatrix}. \quad (2.14)$$

2.2.3 A EQUAÇÃO MESTRA

Para muitos processos observáveis, a probabilidade de transição em um intervalo de tempo muito curto pode ser calculada por (KAMPEN, 2007):

$$P(x|z, \tau') = (1 - a_0\tau') \delta(x - z) + \tau'W(x|z) + \sigma(\tau'^2), \quad (2.15)$$

sendo $W(y_2, y_1)$ a probabilidade de transição de x para z por unidade de tempo e $1 - a_0\tau'$ a probabilidade de nenhuma transição ocorrer durante τ' e, conseqüentemente:

$$a_0(z) = \int_{-\infty}^{\infty} W(x|z) dy_2. \quad (2.16)$$

Utilizando a equação 2.15, na equação de Chapman-Kolmogorov 2.10, dividindo por

τ' e fazendo $\tau' \rightarrow 0$, obtém-se a equação mestra:

$$\frac{\partial}{\partial t} P(y_3|y_1) = \int [W(y_3 | y_2) P(y_2 | y_1) - W(y_2 | y_3) P(y_3 | y_1)] dy_2 \quad (2.17)$$

Correra e Frank (2011) deduziram a equação mestra de uma forma que deixa claro o conceito por trás dessa modelagem. Para isso, foi utilizado um sistema com k estados cujo objetivo é calcular em um instante t qualquer e em um estado l qualquer, a quantidade de elementos $N_l(t)$. A variação do número de ocupação de um determinado estado l entre um instante t e um instante $t + \Delta t$ é representada por ΔN_l . Essa variação corresponde ao número de elementos que partem de qualquer estado para o estado determinado descontados do número de elementos que partem do estado determinado para qualquer outro estado, ou seja:

$$\Delta N_l(t) = [\Delta P_{l \leftarrow 1}(t) N_1(t) + \Delta P_{l \leftarrow 2}(t) N_2(t) + \dots + \Delta P_{l \leftarrow K}(t) N_k(t)] - [\Delta P_{1 \leftarrow l}(t) N_l(t) + \Delta P_{2 \leftarrow l}(t) N_l(t) + \dots + \Delta P_{k \leftarrow l}(t) N_l(t)] \quad (2.18)$$

Considerando $\Delta P_l = \Delta N_l/N$ em um intervalo de tempo Δt suficientemente curto e dividindo a equação pelo número total de elementos N , onde $N = \sum_i^k N_i$, chega-se à equação:

$$\Delta P_l(t) = [\Delta P_{l \leftarrow 1}(t) P_1(t) + \Delta P_{l \leftarrow 2}(t) P_2(t) + \dots + \Delta P_{l \leftarrow K}(t) P_k(t)] - [\Delta P_{1 \leftarrow l}(t) P_l(t) + \Delta P_{2 \leftarrow l}(t) P_l(t) + \dots + \Delta P_{k \leftarrow l}(t) P_l(t)] \quad (2.19)$$

O conjunto de l equações ΔP_l deve ser resolvido simultaneamente, pois as probabilidades de ocupação são interdependentes. Esse conjunto de equações pode ser representado através de uma notação matricial:

$$\Delta P = \begin{pmatrix} \Delta P_1 \\ \vdots \\ \Delta P_k \end{pmatrix} = \begin{pmatrix} \Delta P_{1,1}(t) P_1(t) + \dots + \Delta P_{1,k}(t) P_k(t) \\ \vdots \\ \Delta P_{k,1}(t) P_1(t) + \dots + \Delta P_{k,k}(t) P_k(t) \end{pmatrix} - \begin{pmatrix} \Delta P_{1,1}(t) P_1(t) + \dots + \Delta P_{k,1}(t) P_1(t) \\ \vdots \\ \Delta P_{1,k}(t) P_k(t) + \dots + \Delta P_{k,k}(t) P_k(t) \end{pmatrix} \quad (2.20)$$

Ao definir a matriz de probabilidades de transição infinitesimais T como a equação

2.21, é possível representar a equação 2.20 para um determinado estado de interesse l através de uma equação de Markov 2.22.

$$T = \begin{pmatrix} \Delta P_{1,1} & + \cdots + & \Delta P_{1,k} \\ & \vdots & \\ \Delta P_{k,1} & + \cdots + & \Delta P_{k,k} \end{pmatrix} \quad (2.21)$$

$$\Delta P_l = \sum_m (\Delta T_{l,m} P_m - \Delta T_{m,l} P_l) \quad (2.22)$$

Para a equação 2.20, o índice $m = (1, 2, \dots, l, \dots, k)$ e $\Delta T_{l,m}$ representa o termo da m -ésima coluna e l -ésima linha da matriz T .

A equação 2.22 pode ser reescrita na forma diferencial, ou seja, $\Delta T \rightarrow 0$. Dessa forma, obtêm-se a equação mestra 2.23 que descreve a evolução temporal das probabilidades de ocupação de um determinado sistema.

$$\frac{\partial P_l}{\partial t} = \sum_m (W_{l,m} P_m - W_{m,l} P_l) \quad (2.23)$$

Na equação 2.23, a matriz W é chamada matriz de coeficientes de transição. Considerando $R_{i,j} = dP_{i,j}/dt$, a matriz W é dada por:

$$W = \begin{pmatrix} R_{1,1} & + \cdots + & R_{1,k} \\ & \vdots & \\ R_{k,1} & + \cdots + & R_{k,k} \end{pmatrix} \quad (2.24)$$

A equação mestra é uma equação de ganho e perda que descreve a probabilidade de ocupação dos estados para um determinado sistema ao longo do tempo. Para essa equação, o termo de ganho, $W_{l,m} P_m$, descreve todos os termos m que dão origem a termos l enquanto o termo de perda, $-W_{m,l} P_l$, descreve todos os termos l que dão origem a termos m (CORRERA; FRANK, 2011).

2.2.4 MÉTODO DE MONTE CARLO

O método de Monte Carlo é uma abordagem estatística ao estudo de equações integro-diferenciais, sendo que essas equações ocorrem em vários ramos das ciências naturais

(METROPOLIS; ULAM, 1949).

Segundo Sobol (1994), o método de Monte Carlo é um método numérico de solução de problemas matemáticos por amostragem aleatória que só pôde emergir com o aparecimento dos computadores.

A ideia central do método de Monte Carlo é solucionar problemas fazendo amostragem aleatória. Após a amostragem, os dados obtidos podem ser submetidos a uma média, a um cálculo de interesse ou podem até mesmo serem plotados em um gráfico. O método de Monte Carlo é genérico e usualmente explicado com demonstrações de sua utilização em diversos problemas de diferentes áreas do conhecimento.

Em uma simulação de Monte Carlo, o primeiro passo é gerar um grande conjunto de N amostras do processo de Markov em questão. Uma vez que esse conjunto de amostras tenha sido gerado, então o processo pode ser numericamente estimado. A estimativa será exata no limite $N \rightarrow \infty$. Na prática, N será finito, o que indica que existirá algum tipo de incerteza na estimativa. Se N for suficientemente grande, é possível tratar quantitativamente essa incerteza utilizando o teorema do limite central, obtendo-se um intervalo de confiança (GILLESPIE, 1991).

É possível estimar computacionalmente qualquer propriedade dinâmica de um processo de Markov utilizando métodos de Monte Carlo, porém esse método tende ao consumo exagerado de tempo devido à incerteza decrescer lentamente com o crescimento de N .

Para mais informações sobre métodos de Monte Carlo, consulte Gillespie (1991), Sobol (1994) e Metropolis e Ulam (1949).

A seção 2.5 apresenta a utilização do método de Monte Carlo para simular o processo de Markov conhecido como simulação estocástica de redes de regulação gênica. O primeiro algoritmo apresentado recebeu o nome *Stochastic Simulation Algorithm - Direct Method* (SSA-DM ou somente DM). É importante destacar que o DM soluciona esse processo de Markov utilizando números aleatórios para selecionar a próxima reação e o tempo em que a mesma ocorrerá. Para solucionar a próxima reação, um número aleatório é utilizado pelo algoritmo em uma estratégia do tipo roleta, onde a probabilidade de seleção de uma reação é proporcional à propensidade da mesma. A propensidade de uma reação é proporcional ao número de moléculas dos reagentes e a um dado experimental que representa a constante de velocidade da reação. Já o tempo em que ocorrerá a próxima reação é inversamente proporcional à soma das propensidades de todas as reações, ou seja,

propensidades maiores indicam que existe uma maior probabilidade da reação ocorrer em um tempo próximo ao tempo atual. Para gerar os resultados esperados, o DM é executado diversas vezes e cada uma de suas saídas representa uma trajetória da equação mestra do mesmo processo, o que torna esse algoritmo um bom método de aproximação do resultado da equação mestra.

2.3 CONCEITOS COMPUTACIONAIS

Alguns conceitos computacionais são fundamentais para o entendimento do presente trabalho, com destaque para a explicação sobre o que é a análise assintótica de algoritmos e a definição de algumas estruturas de dados, tais como: grafos, *heap*, lista de prioridades e tabelas *hash*. Esses conceitos são apresentados a seguir.

2.3.1 ANÁLISE ASSINTÓTICA DE ALGORITMOS

A análise assintótica de um algoritmo tem o objetivo de descobrir comportamento do tempo de execução do mesmo. As análises mais utilizadas são a do pior caso, a do melhor caso e a do caso médio. Como um limite superior sobre o tempo de execução de um algoritmo para uma entrada qualquer, o tempo de execução para o pior caso apresenta uma garantia de que o algoritmo não irá demorar mais tempo. O tempo de execução de um algoritmo para o melhor caso fornece o melhor desempenho do algoritmo para todas as entradas possíveis. Já o tempo de execução para o caso médio pode ser difícil de ser obtido, pois pode não ser aparente o que constitui uma entrada média para um determinado algoritmo (CORMEN et al., 2001). No presente trabalho, será apresentada a análise para o pior caso.

Para simplificar os cálculos, na análise assintótica, os custos reais de cada instrução são desconsiderados e em seu lugar são utilizadas constantes para identificação de seus custos. Dessa forma, uma operação de soma recebe o mesmo peso de uma operação de divisão quando estamos trabalhando com análise assintótica de algoritmos (SZWARCFITER; MARKENZON, 1994).

Ainda para a simplificação dos cálculos, somente são considerados o termo de maior ordem de magnitude em uma fórmula, como o termo an^2 para a fórmula $an^2 + n + C$. Ao considerar somente o termo de maior magnitude, deve-se desconsiderar a sua constante,

ou seja, considera-se somente n^2 para a fórmula apresentada anteriormente. Isso é possível devido aos termos de ordens de magnitudes mais baixas não impactarem significativamente o tempo de execução de um algoritmo para grandes valores de n , onde n representa o tamanho da entrada. Já a constante do termo de maior magnitude pode ser desconsiderada devido ao seu impacto no tempo de execução do algoritmo ser menos significativo que a taxa de crescimento (CORMEN et al., 2001).

Para exemplificar a utilização na análise assintótica de algoritmos, a implementação de um programa para o cálculo do somatório dos n primeiros números pode ser utilizado. O algoritmo 2.1 apresenta uma primeira abordagem para esse problema.

Algoritmo 2.1: Um algoritmo para soma dos n primeiros números. Considerando o valor 1 como peso da execução de cada instrução desse algoritmo, a complexidade do mesmo pode ser calculada considerando o número de vezes que o algoritmo passará por cada linha multiplicado pelo peso de execução de cada linha. A linha 2 será executada uma vez e possui peso 1. A linha 3 será executada $n + 1$ vezes e considera-se o peso dessa linha como sendo 1. A linha 4 será executada n vezes e o seu peso considerado é 2. A linha 6 é executada uma vez e o seu peso considerado é 1. Logo, a complexidade do algoritmo é $1 + (n+1) * 1 + 2n + 1$, que pode ser simplificado como $3n + 3$. Como as constantes e as constantes multiplicativas são desconsideradas na análise assintótica, a complexidade para o pior caso para esse algoritmo é $O(n)$, ou seja, a complexidade desse algoritmo cresce de forma linear em relação a n .

Entrada: número de elementos (n).

```

início
[02] soma = 0;
[03] para i = 1 até n
[04]     soma = soma + i;
[05] fim-para;
[06] imprima soma;
fim.
```

O algoritmo 2.2 apresenta uma abordagem mais eficiente que a utilizada no algoritmo 2.1.

Algoritmo 2.2: Um algoritmo para soma dos n primeiros números. Considerando o valor 1 como peso da execução de cada instrução desse algoritmo, a complexidade do mesmo pode ser calculada considerando o número de vezes que o algoritmo passará por cada linha multiplicado pelo peso de execução de cada linha. A linha 2 será executada uma vez e possui peso 4. Logo, a complexidade do algoritmo é 4. Como as constantes são desconsideradas na análise assintótica, a complexidade para o pior caso para esse algoritmo é $O(1)$, ou seja, a complexidade desse algoritmo é constante e independe do tamanho de n . É importante destacar que para o exemplo dado, se n for muito grande, o processador do computador pode precisar fazer um número proporcional a $\log n$ passos

Entrada: número de elementos (n).

início

[02] imprima $(1 + n) * n / 2$;

fm.

2.3.2 GRAFOS

As relações entre objetos de um determinado conjunto podem ser representadas por um grafo. Um grafo é composto por vértices e arestas, sendo que os vértices representam os objetos e as arestas representam as relações entre esses objetos.

Um dígrafo é um grafo orientado, ou seja, a existência de uma aresta $A \rightarrow B$ não implica na existência da $B \rightarrow A$. Um exemplo de um dígrafo pode ser visualizado na figura 2.6.

Considerando V o conjunto de vértices de um grafo e E o conjunto de arestas, um grafo $G(V, E)$ é uma estrutura cujo tamanho é descrito por dois parâmetros (V e E). Um grafo é esparso quando E é muito menor que V^2 . Já um grafo é considerado denso quando E é um valor próximo a V^2 (CORMEN et al., 2001).

O número de arestas que chegam em um vértice é conhecido como grau de entrada desse vértice. O número de arestas que saem de um vértice é conhecido como grau de saída desse vértice.

Matrizes são utilizadas para a representação de grafos no presente trabalho. Uma matriz é um vetor bidimensional $v_{i,j}$ onde é comum o índice i representar a linha e o

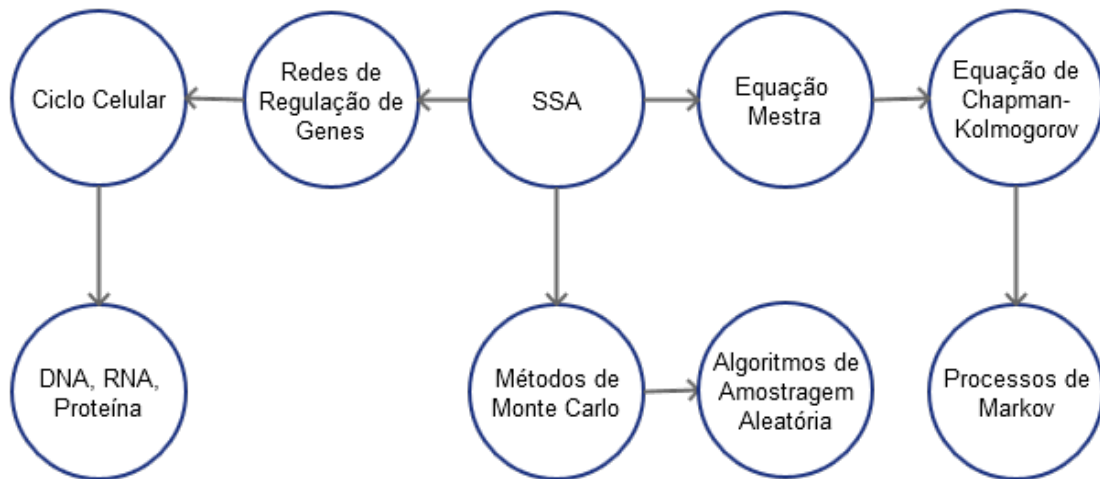


Figura 2.6: Grafo representando as relações entre os conceitos envolvidos no algoritmo *Stochastic Simulation Algorithm* (SSA) que será apresentado com detalhes na seção 2.5. É possível notar, por exemplo, que o SSA está relacionado à Equação Mestra, que por sua vez está relacionada à Equação de Chapman-Kolmogorov, que está relacionada às cadeias de Markov.

índice j representar a coluna da matriz. Dessa forma, a complexidade para acesso a um elemento de uma matriz é constante, ou seja, $O(1)$. Veja a figura 2.7.

(a)

	$j \longrightarrow$			
i	7	3	2	9
1	1	6	5	0
↓	1	4	2	3

(b)

Elemento	Posição em memória
3	118
2	117
4	116
1	115
0	114
5	113
6	112
1	111
9	110
2	109
3	108
7	107

Figura 2.7: Matriz de inteiros com $i = [0, \dots, 3)$ e $j = [0, \dots, 4)$. (a) é a representação da matriz em uma tabela com i linhas e j colunas. (b) é a representação da forma como os elementos da matriz (a) ficam alocados na memória do computador.

Uma matriz M pode ser utilizada para representar a existência de ligações entre dois

elementos de um determinado conjunto. Sendo N o número total de elementos do conjunto, temos os valores de i e j variando de 0 até $N-1$. Dessa forma, tanto i quanto j serão índices que representam os elementos e, caso exista uma ligação entre o elemento $i=2$ e $j=3$ o valor 1 deve ser armazenado na posição $M_{2,3}$ e, caso não exista a ligação, o valor 0 deve ser armazenado. Essa é uma típica representação de um grafo utilizando uma matriz.

Os índices das linhas e das colunas da matriz M supracitada representam os vértices. Os valores da matriz M representam as arestas.

Geralmente os grafos são representados de duas maneiras: como uma coleção de listas de adjacências ou como uma matriz de adjacências. A representação em lista de adjacências é preferível para grafos esparsos enquanto a representação em matrizes de adjacências é preferível para grafos densos ou quando é necessário saber rapidamente se dois vértices estão conectados por uma aresta.

Quando existe a necessidade de conhecer todas as arestas que partem de um determinado elemento, a utilização de uma matriz de adjacências compacta pode apresentar um melhor desempenho. Em uma matriz de adjacências compacta, as linhas representam os elementos enquanto as colunas são preenchidas somente com os índices dos vértices para os quais existem arestas partindo do vértice representado pela linha. Considerando a existência de três vértices (A, B, C) e 4 arestas ($A \rightarrow B, A \rightarrow C, B \rightarrow C, C \rightarrow A$), a figura 2.8 apresenta duas formas de representação para esse grafo.

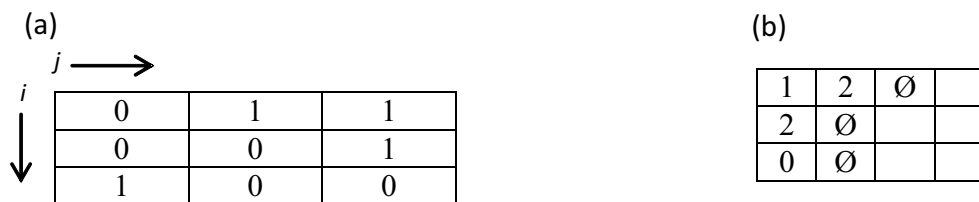


Figura 2.8: (a) Grafo $G(V, E)$ representado por uma matriz de adjacências. (b) grafo $G(V, E)$ representado por uma matriz de adjacências compacta. O símbolo \emptyset deve ser substituído por um valor qualquer que indique que não existem mais vértices ligados por arestas que partem do vértice representado pelo índice da linha. Note que para buscar todas as arestas que partem de $i = 1$, é necessário percorrer três colunas da matriz (a) e somente duas colunas da matriz (b).

2.3.3 HEAP

Uma *heap* é uma estrutura de dados que possui o objetivo de ordenar elementos de forma que o maior ou o menor elemento de um conjunto seja obtido com a complexidade $O(1)$. Uma *heap* construída para retornar o maior elemento é conhecida como *max-heap*, enquanto uma *heap* construída para retornar o menor elemento é conhecida como *min-heap* (BRASS, 2008).

Uma *heap* pode ser representada por uma lista linear, como um vetor. Essa é a forma de representação adotada no presente trabalho.

Em uma *heap*, cada índice ou chave do vetor representa a prioridade do respectivo elemento (SZWARCFITER; MARKENZON, 1994).

Em uma *min-heap* representada por uma lista linear, os n elementos s_0, s_1, \dots, s_{n-1} precisam satisfazer a propriedade representada no conjunto de inequações 2.25 (BRASS, 2008). A figura 2.9 apresenta uma *min-heap*.

$$\begin{aligned} s_i &\leq s_{2i+1}, \quad \text{se } 2i + 1 < n \\ &e \\ s_i &\leq s_{2i+2}, \quad \text{se } 2i + 2 < n. \end{aligned} \tag{2.25}$$

Em uma *max-heap* representada por uma lista linear, os n elementos s_0, s_1, \dots, s_{n-1} precisam satisfazer a propriedade representada no conjunto de inequações 2.26.

$$\begin{aligned} s_i &\geq s_{2i+1}, \quad \text{se } 2i + 1 < n \\ &e \\ s_i &\geq s_{2i+2}, \quad \text{se } 2i + 2 < n. \end{aligned} \tag{2.26}$$

A complexidade de atualização de um elemento em uma *heap* é $O(\log n)$, enquanto a complexidade de obtenção do menor elemento em uma *min-heap* ou do maior elemento em uma *max-heap* é $O(1)$ (CORMEN et al., 2001).

No presente trabalho, as *heaps* são utilizadas para representar listas de prioridades. Essas listas são utilizadas para obter a reação de menor tempo ou o produto de menor tempo agendado. Dessa forma, quando for utilizado o nome *heap* nas demais seções, considere a utilização de uma *min-heap*.

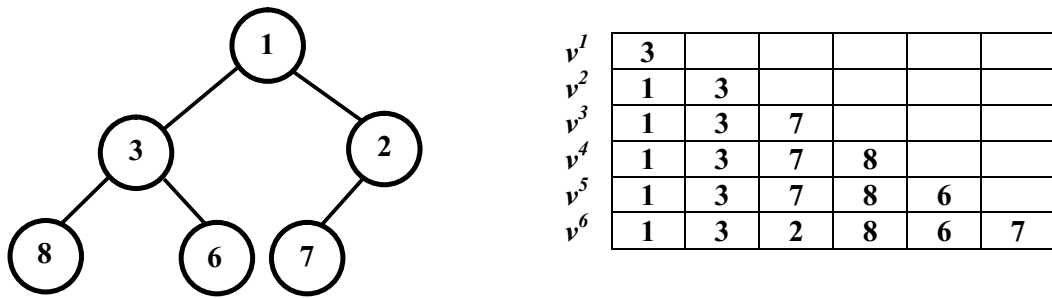


Figura 2.9: *min-heap* construída a partir da seguinte entrada de elementos: $\{3,1,7,8,6,2\}$. (a) representação gráfica da *min-heap*. (b) representação da *min-heap* utilizando um vetor, sendo mostrado uma versão do vetor para cada inserção de elemento (v^1, v^2, \dots, v^6). Note que os elementos são inseridos ao final do vetor e, se forem menores que seu pai ($\lfloor i/2 \rfloor$), o elemento é trocado com o seu pai recursivamente, até assumir a posição correta.

2.3.4 TABELAS *HASH*

As tabelas *hash* são coleções de pares chave/valor onde uma função de mapeamento transforma a chave em um índice de uma tabela, permitindo que esse valor seja recuperado quando informada a sua chave (CELES et al., 2004).

Uma tabela *hash* é composta por um vetor e uma função de mapeamento, conhecida como função *hash*. O papel da função *hash* é mapear cada chave em um índice do vetor. Dessa forma, a tabela *hash* utiliza a função de mapeamento para calcular a posição onde um elemento será inserido no vetor. Essa mesma função de mapeamento é utilizada para recuperar a posição do elemento no vetor quando for solicitada a extração de um valor da tabela *hash* (CORMEN et al., 2001).

Uma das características presentes em uma tabela *hash* é que a função *hash* conhece somente o tamanho do vetor, mas não conhece os elementos contidos nele, o que permite a inclusão de itens dinamicamente em uma tabela *hash*. Se for considerado que a escolha do tamanho do vetor de uma tabela *hash* é um fator importante para o consumo de memória e que a função *hash* não conhece o posicionamento dos itens que já estão alocados no vetor, existe a possibilidade de duas chaves distintas serem mapeadas para um mesmo índice. Uma das soluções para esse problema é utilizar listas encadeadas como itens do vetor da tabela *hash* (SZWARCFITER; MARKENZON, 1994).

Ao tentar recuperar um elemento em uma tabela *hash*, se a função *hash* mapeia uma determinada chave para um índice que possui mais de um elemento, o sistema de busca

percorre a lista encadeada até encontrar o elemento que possui a mesma chave informada como parâmetro de busca.

Considerando-se a utilização de listas encadeadas em cada posição de alocação do vetor da tabela *hash*, pode-se considerar também que, na pior das hipóteses, uma tabela *hash* possuirá um desempenho de busca semelhante ao de uma lista comum. Porém esse cenário só ocorreria se a função *hash* mapeasse todas as chaves para um mesmo índice (CORMEN et al., 2001).

Considerando o exposto no parágrafo anterior, nota-se a importância da qualidade da função *hash* para obtenção de uma distribuição uniforme das chaves. Quanto melhor for a função de distribuição, melhor será o desempenho de uma tabela *hash*. Existem algumas características facilmente identificáveis de uma boa função *hash*:

- a) O índice retornado é totalmente determinado pela chave a ser mapeada;
- b) A função utiliza toda a chave a ser mapeada, ou seja, nenhuma parte da chave a ser mapeada pode ser desconsiderada pela função;
- c) A função retorna valores distribuídos uniformemente dentre os índices existentes no vetor;
- d) A função gera índices muito diferentes para chaves de valores aproximados, ou seja, uma boa função *hash* mapearia a chave “ab” para um índice diferente das chaves “abc” e “ba”.
- e) A função é rápida o suficiente para que seu tempo de processamento não seja significativo se comparado ao tempo de uma busca ou inserção de um item em um vetor.

Para exemplificar, o algoritmo 2.3 apresenta uma função *hash* simples.

Algoritmo 2.3: Uma função *hash* simples. Na linha 03 inicia-se um laço que percorrerá todos os caracteres da chave recebida como parâmetro e somará o valor inteiro que representa o caractere à variável soma, na linha 04. A linha 06 apresenta o retorno da função que é o resto da divisão da variável soma pelo tamanho do vetor.

Entrada: chave que identifica o item (chave), tamanho escolhido para a tabela *hash* (tamanhoVetor).

```

início
[02] sum = 0;
[03] para cada (caractere c pertencente à chave)
[04]     soma = soma + ConverterParaInteiro(c);
[05] fim-para;
[06] retornar retornarRestoDaDivisão(soma, tamanhoVetor);
fim.
```

De acordo com a lista de características de uma boa função *hash*, a função *hash* apresentada no algoritmo 2.3 não é considerada boa, pois não atende à quarta característica (letra d). Isso ocorre devido à função retornar o mesmo valor para as chaves “ab” e “ba”. Outra característica interessante da função apresentada no algoritmo 2.3, é que o seu funcionamento atende melhor à característica c, da lista de características de uma boa função *hash*, quando o tamanho do vetor é um número primo.

Sendo n o número de elementos de um vetor, para localizarmos um elemento nesse, podemos utilizar um algoritmo de busca linear para comparar cada elemento do vetor com o elemento procurado. Dessa forma, a complexidade do algoritmo de busca é expressa por $O(n)$. Sabendo o índice do vetor que contém o elemento, a complexidade da busca é $O(1)$ (CELES et al., 2004).

A figura 2.10 apresenta a estrutura de uma tabela *hash*.

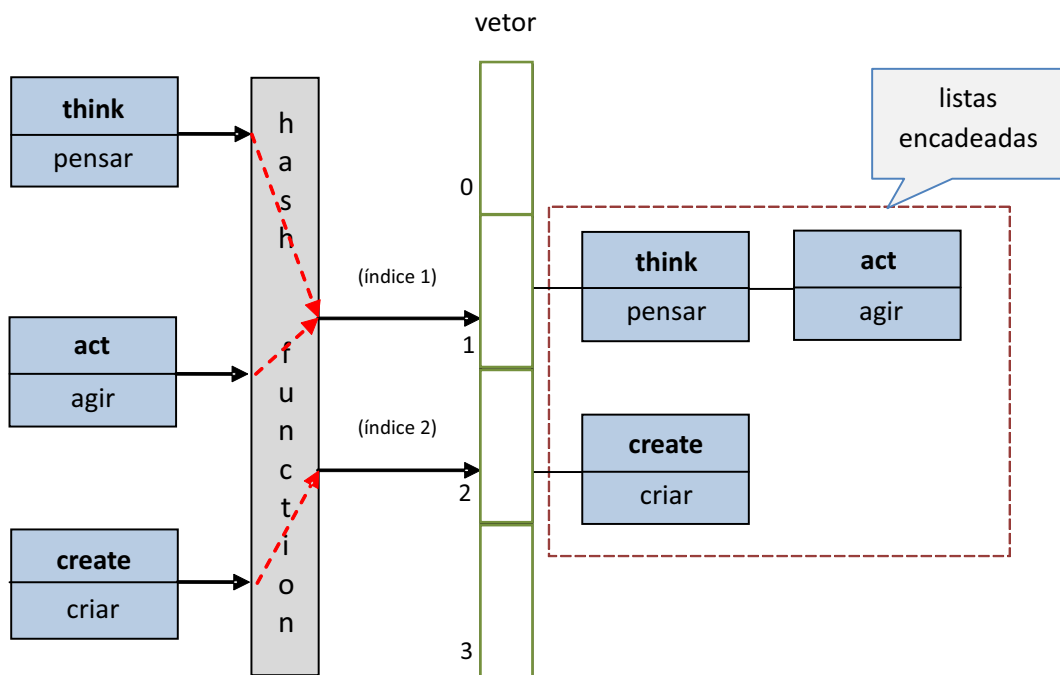


Figura 2.10: Estrutura de uma tabela *hash* cujo objetivo é retornar uma palavra em português e a sua correspondente em inglês. Dessa forma, o par chave-valor será composto pela chave sendo uma palavra em inglês e o valor sendo uma palavra em português. É possível notar no índice 1 que “think” e “act” foram mapeados para um mesmo índice do vetor. Para esse caso, a tabela *hash* pode utilizar listas encadeadas, de forma que a tabela *hash* compara a chave de busca com a chave armazenada e, se forem diferentes, a tabela *hash* verifica o próximo item da lista encadeada.

Com um tamanho adequado para o vetor e a escolha de uma função *hash* eficiente, estatisticamente pode-se considerar a eficiência da busca de um elemento em uma tabela *hash* como sendo $O(1)$ e, em seu pior caso, quando a função *hash* for de baixa qualidade ou os espaços de alocação estiverem mal dimensionados, pode-se chegar à complexidade $O(n)$, que ocorreria se todos os elementos de uma tabela *hash* fossem inseridos em um único índice da tabela (SZWARCFITER; MARKENZON, 1994).

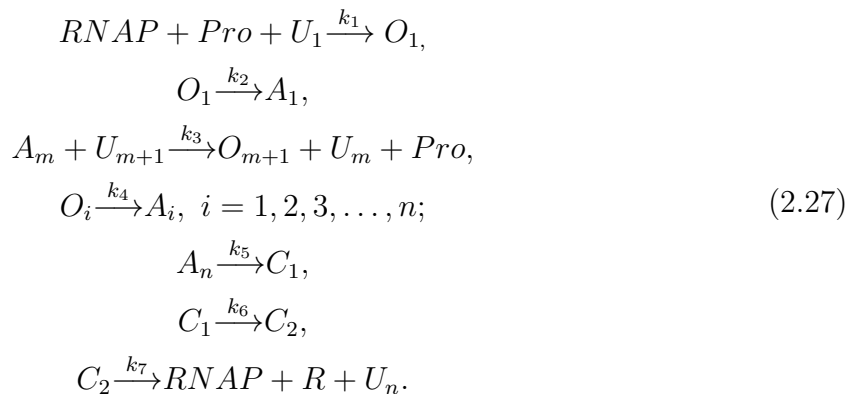
2.4 SIMULAÇÃO DE REDES DE REGULAÇÃO GÊNICA

Existem duas formulações para simulação dinâmica de redes de regulação gênica: a determinística e a estocástica. Na formulação determinística, a variação das espécies no tempo pode ser calculada utilizando-se um conjunto de equações diferenciais ordinárias acopladas. Já na formulação estocástica, essa variação pode ser calculada utilizando-se a equação mestra (equação 2.17). A função que satisfaz a equação mestra mede a probabilidade de serem encontradas várias populações moleculares a cada instante de tempo (GILLESPIE, 1976).

2.4.1 SIMULAÇÃO DETERMINÍSTICA

As simulações determinísticas de redes de regulação gênica utilizam um conjunto de equações diferenciais ordinárias (EDO's) acopladas para descrever processos bioquímicos (OLIVEIRA, 2008). Essas equações descrevem a alteração de cada tipo de molécula em função dos níveis de algumas moléculas relacionadas.

É possível formular uma solução analítica para alguns conjuntos simples de EDO's, mas uma solução numérica pode ser necessária para conjuntos complexos (KLIPP et al., 2005).



Considerando $i = 2, 3, \dots, n$, a dinâmica das reações para a rede apresentada no conjunto de equações 2.27 pode ser representada pelo seguinte conjunto de EDO's (equações 2.28):

$$\begin{aligned}
\frac{dRNAP}{dt} &= -k_1 (RNAP.Pro.U_1) + k_7 C_2, \\
\frac{dPro}{dt} &= -k_1 (RNAP.Pro.U_1) + k_3 (A_m.U_{m+1}), \\
\frac{dU_1}{dt} &= -k_1 (RNAP.Pro.U_1) + k_3 (A_1.U_2) + k_7 C_2, \\
\frac{dO_1}{dt} &= -k_2 O_1 + k_1 (RNAP.Pro.U_1), \\
\frac{dA_1}{dt} &= -k_3 (A_1.U_2) - k_5 A_1 + k_2 O_1, \\
\frac{dC_1}{dt} &= -k_6 C_1 + k_5 A_n, \\
\frac{dC_2}{dt} &= -k_7 (RNAP.R.U_n) + k_6 C_1, \\
\frac{dA_i}{dt} &= -k_3 (A_{i-1}.U_i) + k_3 (A_i.U_{i+1}) + k_7 C_2, \\
\frac{dU_i}{dt} &= -k_3 (A_i.U_{i+1}) - k_5 A_i + k_4 O_i, \\
\frac{dO_i}{dt} &= -k_4 O_i + k_3 (A_{i-1}.U_i).
\end{aligned} \tag{2.28}$$

2.4.2 SIMULAÇÃO ESTOCÁSTICA

Embora a simulação determinística de redes de regulação gênica tenha se mostrado uma abordagem de sucesso, ela possui alguns problemas. Em subsistemas celulares, onde é comum encontrarmos um número pequeno de moléculas de determinada espécie, essa abordagem pode levar a flutuações aleatórias que podem alterar consideravelmente o seu comportamento dinâmico. Além disso, sistemas bi-estáveis ou multi-estáveis não podem ser descritos corretamente pela abordagem determinística. Uma outra desvantagem da simulação determinística é que a própria estocasticidade de um sistema pode ser uma propriedade importante para o estudo do modelo (PAHLE, 2009).

Segundo McAdams e Arkin (1997), como as moléculas de um sistema possuem uma natureza discreta, o uso de EDO's para descrever processos bioquímicos pode levar à obtenção de dados insatisfatórios. Além disso, nas simulações com EDO's, considera-se o processo como determinístico, não levando em consideração as flutuações aleatórias.

Uma solução para as limitações da simulação determinística de redes de regulação gênica é utilizar a simulação estocástica. Essa última calcula explicitamente a variação do número de moléculas das espécies envolvidas em uma reação química ao longo do tempo

(OLIVEIRA, 2008).

Uma desvantagem da simulação estocástica é que ela demanda mais poder computacional que a simulação determinística. Os métodos exatos de simulação estocástica possuem o tempo de processamento proporcional ao número de reações que ocorrerem durante a simulação (PAHLE, 2009).

A simulação estocástica de uma rede de regulação gênica pode ser feita utilizando-se a equação mestra (equação 2.17), porém esse método pode ser computacionalmente inviável devido à complexidade de alguns modelos. Gillespie (1977) propôs uma aproximação sistemática à equação mestra. O método de Gillespie, conhecido como *Stochastic Simulation Algorithm* (SSA), utiliza métodos derivados de Monte Carlo para simular numericamente o processo de Markov que é analiticamente descrito pela equação mestra.

O SSA é um algoritmo exato, uma vez que ele gera resultados equivalentes ao resultado da equação mestra. Com esse algoritmo, a simulação de algumas redes de regulação gênica tornou-se possível devido ao algoritmo apresentar somente uma trajetória para a rede, enquanto a equação mestra apresenta simultaneamente a probabilidade de todas as trajetórias. Para que o SSA possa ser utilizado para se obter as mesmas informações que poderiam ser obtidas utilizando-se a equação mestra, é necessário executá-lo várias vezes e armazenar cada trajetória simples, de forma que o conjunto de trajetórias obtido seja estudado estatisticamente (KLIPP et al., 2005). Os algoritmos SSA's serão descritos com mais detalhes na próxima seção.

2.5 ALGORITMO DE SIMULAÇÃO ESTOCÁSTICA

Como mencionado anteriormente, um passo importante para tornar a simulação estocástica de redes de regulação gênica computacionalmente viável foi o desenvolvimento do algoritmo de simulação estocástica conhecido como *Stochastic Simulation Algorithm* de Gillespie (1976, 1977), que é um algoritmo exato, uma vez que ele gera resultados equivalentes ao resultado da equação mestra. Com esse algoritmo, a simulação de algumas redes de regulação gênica foi viabilizada devido ao algoritmo apresentar somente uma trajetória para a rede, enquanto a equação mestra apresenta simultaneamente a probabilidade de todas as trajetórias (KLIPP et al., 2005).

É importante destacar que o SSA não é uma tentativa de solucionar numericamente a equação mestra para um dado sistema. Ao invés disso, ele é um procedimento sistemático

no qual métodos rigorosamente derivados das técnicas de Monte Carlo são empregados para simular numericamente o processo de Markov descrito analiticamente pela equação mestra (GILLESPIE, 1976). Dessa forma, uma saída do SSA corresponde a uma das soluções apresentadas pela equação mestra e todas as saídas possíveis para o SSA correspondem à solução para a equação mestra.

O algoritmo SSA possui dois passos principais. O primeiro passo é a escolha do tempo em que a próxima reação ocorrerá e o segundo passo é a escolha da próxima reação.

Alguns parâmetros utilizados pelos diversos algoritmos de simulação estocástica são comuns. Dentre esses parâmetros, destacam-se o estado inicial, a estequiometria, e as constantes de velocidades das reações.

O estado inicial em um algoritmo de simulação estocástica é a representação da quantidade de moléculas de cada espécie do modelo no tempo imediatamente anterior à primeira reação executada. Esse parâmetro é comumente representado em um vetor cujos índices indicam as espécies e os valores armazenados indicam as suas respectivas quantidades.

A estequiometria de uma rede bioquímica é a representação das proporções definidas de cada espécie em cada reação. Nos algoritmos que serão apresentados, esse parâmetro foi representado por uma matriz $v_{i,j}$ cujas linhas representam as reações e as colunas representam as espécies. Os valores armazenados de cada espécie em cada reação representam a quantidade a ser alterada daquela espécie após a execução da reação. Dessa forma, valores negativos indicam que a espécie terá a sua população diminuída, enquanto valores positivos indicam que a espécie terá a sua população aumentada. Para os algoritmos que trabalham com reações com atraso, a estequiometria foi representada por duas matrizes, sendo a primeira a matriz de reagentes e a segunda a matriz de produtos das reações, porém o pseudocódigo desses algoritmos não apresenta essas matrizes explicitamente devido à necessidade de simplificação dos mesmos.

A constante de velocidade de uma reação (k) é medida experimentalmente ou estimada matematicamente. As constantes de velocidades das reações são comumente representadas por um vetor onde o índice indica a reação e o valor armazenado indica a constante de velocidade.

Os algoritmos que serão apresentados utilizam um vetor que é gerado e controlado dentro de cada simulação. Esse vetor é chamado vetor de propensidades (a). O índice desse vetor indica a reação enquanto o seu valor indica o quão provável é a sua escolha

como a próxima reação a ser executada. Dessa forma, reações com grande propensidade possuem mais chance de serem escolhidas como a próxima reação. A propensidade de uma reação pode ser calculada multiplicando-se o número de combinações possíveis das quantidades existentes de cada reagente, considerando a estequiometria da reação, pela constante de velocidade da reação.

Os algoritmos que trabalham com reações com atraso recebem esse parâmetro como uma matriz (d). As linhas dessa matriz representam as reações, as colunas representam os produtos e os valores armazenados representam os tempos em segundos que cada produto em espera precisa aguardar até ser retornado.

Os algoritmos apresentados utilizam o grafo de dependências proposto por Gibson e Bruck (2000). Esse grafo é conhecido pela sigla DG, do inglês *Dependency Graph*. Mais informações sobre esse grafo podem ser obtidas na subseção 3.2.1.

2.5.1 SSA PARA MODELOS SEM ATRASO

Originalmente, o SSA foi desenvolvido para reações sem atraso. Existem algumas variações de implementação do SSA, sendo que o *Direct Method* (DM), o *First Reaction Method* (FRM), o *Next Reaction Method* (NRM) e o *Modified Next Reaction Method* (MNRM) são apresentados no presente trabalho.

Proposto por Gillespie (1977), o algoritmo 2.4 apresenta o pseudocódigo para o DM. O fluxograma para esse algoritmo é apresentado na figura 2.11. O DM é o primeiro e provavelmente o mais conhecido algoritmo publicado para simulação estocástica de redes de regulação gênica.

A seleção da próxima reação no DM é do tipo roleta, onde a chance de cada reação ser selecionada é proporcional à sua propensidade. Ver figura 2.12.

Considerando-se:

1. N o número de espécies do modelo;
2. M o número de reações do modelo;
3. k o maior grau de saída do DG ($k < M$);
4. A atualização da propensidade de uma reação como sendo constante.

Algoritmo 2.4: Pseudocódigo para o algoritmo DM. Primeiramente, é realizada a soma da propensidade de cada reação (linha 1). Na linha 5 é obtido o próximo incremento de tempo (θ). Esse incremento de tempo será somado ao tempo atual de simulação (t) para que seja obtido o tempo em que a próxima reação (j) ocorrerá. A seleção da próxima reação (j) é feita na linha 6, percorrendo-se as propensidades de cada reação como em uma seleção do tipo roleta cujo número sorteado é obtido multiplicando-se o número aleatório U_2 à soma das propensidades.

Entrada: Estequiometria (v), constantes de velocidades das reações (k), estado inicial (S), tempo de simulação (T).

Saída: dinâmica dos estados (X).

início

[01] para-cada reação k , calcule a_k ;

[02] $a_0(t) = \sum_{j=1}^m a_j(t)$;

[03] enquanto $t < T$;

[04] gere U_1 e U_2 como variáveis randômicas no intervalo $(0, 1)$;

[05] $\theta = \frac{1}{a_0(t)} \ln(1/U_1)$;

[06] selecione j tal que $\sum_{k=1}^{j-1} a_k(t) < U_2 a_0(t) \leq \sum_{k=1}^j a_k(t)$;

[07] $X(t + \theta) = X(t) + v_j$;

[08] $t = t + \theta$;

[09] para-cada aresta (j, α) no grafo de dependências DG faça:

[10] Calcule a_α ;

[11] Atualize $a_0 = a_0 - a_{\alpha,old} + a_\alpha$;

[12] fim-para-cada;

[13] fim-enquanto;

fim.

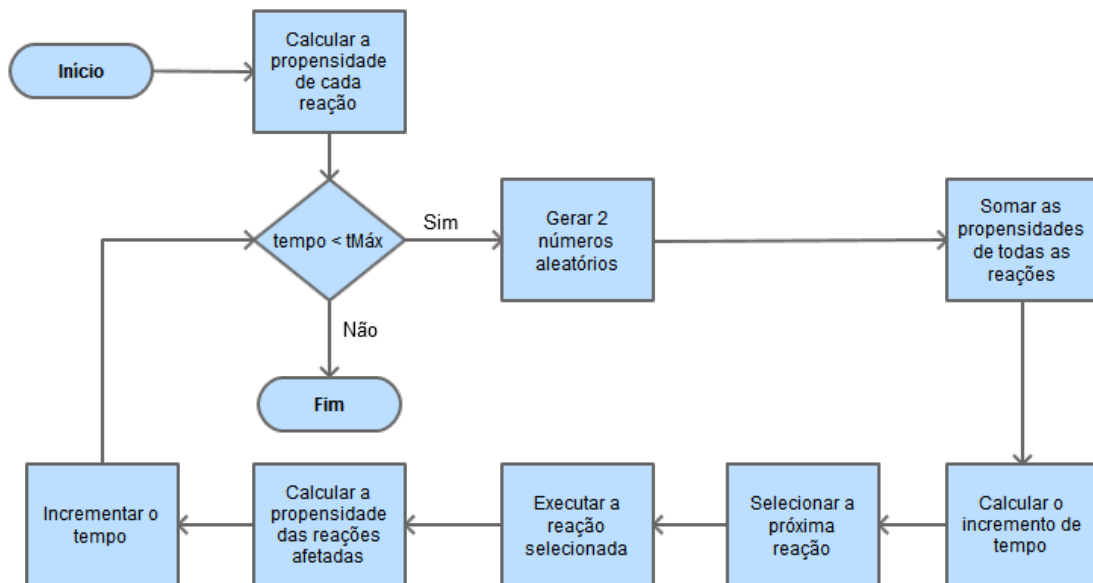


Figura 2.11: Fluxograma do algoritmo DM.

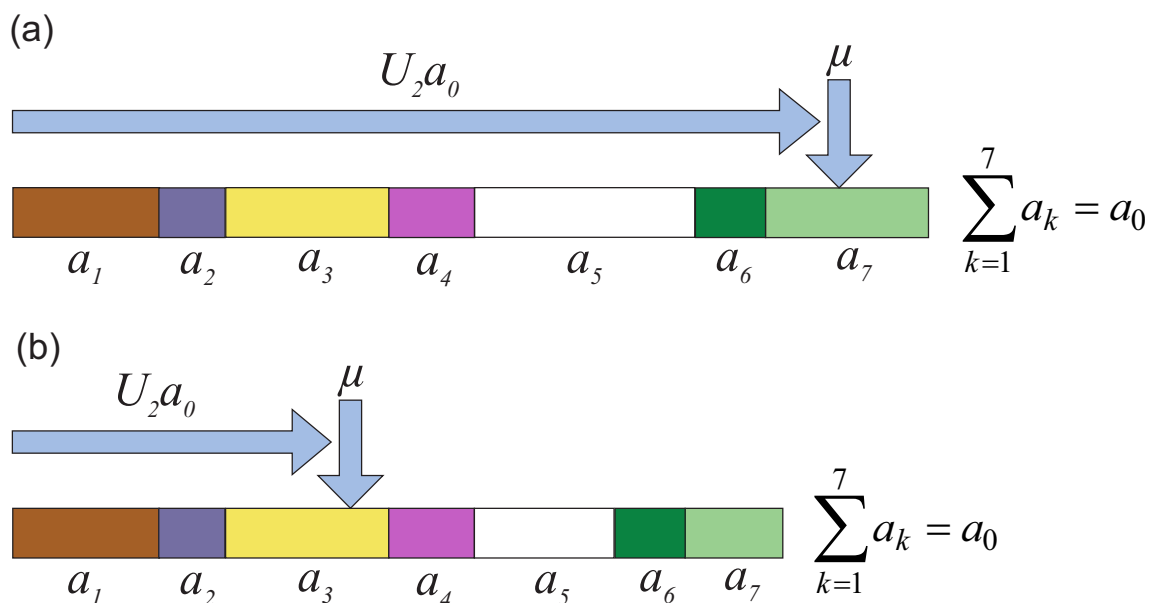


Figura 2.12: Seleção do tipo roleta para a próxima reação. Em uma rede com sete reações (r_1, r_2, \dots, r_7) com suas respectivas propensidades (a_1, a_2, \dots, a_7), a próxima reação é escolhida pelo valor resultante do número aleatório gerado (U_2) multiplicado pelo somatório das propensidades de todas as reações (a_0). Em (a) a reação de índice 7 foi selecionada. Note em (b) que a execução da reação de índice 7 selecionada anteriormente alterou a propensidade das reações de índices 5 e 7. Em (b) a reação de índice 3 foi selecionada.

A complexidade assintótica do DM quando utiliza-se o DG é $O(M)$. Em cada passada desse algoritmo, o único passo linear é a escolha do índice da próxima reação, passo 6 do algoritmo 2.4. Há modelos onde k pode crescer com o número de reações M ou com número de espécies N , portanto a complexidade será $O(M^2)$ ou $O(M + N)$.

O algoritmo FRM também foi proposto por Gillespie (1977) e o seu pseudocódigo é apresentado no algoritmo 2.5. O fluxograma para esse algoritmo é apresentado na figura 2.13. Esse algoritmo não é tão eficiente quanto o DM devido à sua necessidade de gerar novos números aleatórios e calcular novos tempos de disparo para todas as reações a cada reação executada.

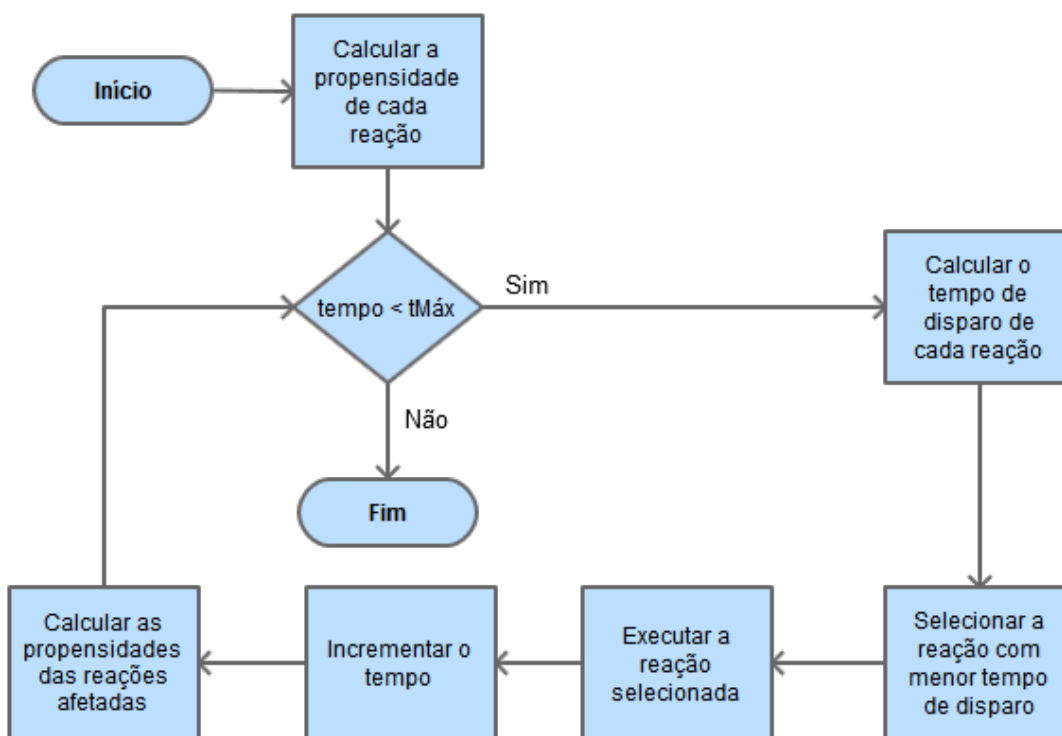


Figura 2.13: Fluxograma do algoritmo FRM.

Considerando-se:

1. N o número de espécies do modelo;
2. M o número de reações do modelo;
3. A atualização da propensidade de uma reação como sendo constante.

A complexidade assintótica de cada passada do FRM é $O(M)$. Como neste algoritmo, a cada reação disparada, é gerado um novo número aleatório para cada reação do modelo, então é necessário calcular o próximo tempo de disparo para cada reação. Com isso, apesar do FRM possuir complexidade assintótica igual à do DM, $O(M)$, ele geralmente apresenta um desempenho inferior ao DM.

Algoritmo 2.5: Pseudocódigo para o algoritmo FRM. Inicialmente, a propensidade de cada reação é calculada na linha 1. Um novo número aleatório é gerado para cada reação (linha 4). O tempo relativo de disparo de cada reação é calculado na linha 6.

Entrada: Estequiometria (v), constantes de velocidades das reações (k), estado inicial (S), tempo de simulação (T).

Saída: dinâmica dos estados (X).

início

[01] para-cada reação k , calcule a_k ;

[02] enquanto $t < T$;

[03] para-cada reação k :

[04] Gere um novo número aleatório r ;

[05] Calcule a_k ;

[06] faça $\tau_k = (-\log(r)/a_k)$;

[07] fim-para-cada;

[08] $t = t + (\tau_\mu = \min\{\tau\})$;

[09] Atualize o número de cada espécie para refletir a execução da reação μ ;

[10] fim-enquanto;

fim.

Para K reações executadas e para M reações em uma rede de regulação gênica, o algoritmo FRM utiliza MK números aleatórios. Já o algoritmo DM utiliza $2K$ números aleatórios. Para que não haja desperdício de números aleatórios na simulação, somente K números aleatórios deveriam ser gerados. O algoritmo NRM, proposto por Gibson e Bruck (2000), utiliza $K + M$ números aleatórios.

O NRM é baseado no FRM, com a principal diferença de trabalhar com tempos absolutos ao invés dos tempos relativos utilizados pelo FRM. Para conseguir utilizar somente $K + M$ números aleatórios, o algoritmo reutiliza números aleatórios de todas as reações do modelo, exceto o da reação executada. Para que não seja necessário recalculá-lo o tempo de disparo de todas as reações a cada reação executada, o grafo de dependências (DG do inglês *Dependency Graph*), elaborado por Gibson e Bruck (2000), é utilizado para que o algoritmo saiba quais reações são afetadas pela reação executada. Tal grafo também é útil para obter as reações que precisam ter as suas propensidades recalculadas. Outras informações sobre o DG podem ser obtidas na subseção 3.2.1.

Ainda como parte da estratégia para descartar no máximo N números aleatórios na simulação, o NRM reaproveita os números utilizados para o cálculo do próximo tempo de disparo das reações que foram identificadas pelo DG para terem a sua propensidade recalculada. Somente a reação executada recebe um novo número aleatório. O algoritmo 2.6 apresenta o pseudocódigo do NRM e a figura 2.14 apresenta o seu fluxograma.

Algoritmo 2.6: Pseudocódigo para o algoritmo NRM. A linha 12 deve ser substituída por $t_\alpha = (a_{\alpha,oldNonZero} / a_\alpha)(t_\alpha - t_1) + t$ quando a_α deixar de ser zero, sendo $a_{\alpha,oldNonZero}$ a última propensidade diferente de zero e t_1 o tempo em que a_α se tornou zero. Para ranquear as reações de forma que seja fácil a obtenção da reação com menor tempo de disparo, uma fila de prioridades indexada P é utilizada (linhas 5 e 14). O reaproveitamento de números aleatórios ocorre na linha 12. Na linha 13 o próximo tempo de disparo da reação selecionada é obtido após obtenção de um novo número aleatório.

Entrada: Estequiometria (v), constantes de velocidades das reações (k), estado inicial (S), tempo de simulação (T).

Saída: dinâmica dos estados (X).

início

[01] para-cada reação k :

[02] Calcule a propensidade a_k ;

[03] Gere um número aleatório r ;

[04] faça $\tau_k = (1/a_k)\ln(1/r)$;

[05] Armazene τ_k em uma fila de prioridades indexada P ;

[06] fim para-cada;

[07] enquanto $t < T$;

[08] $t = \tau_\mu = \min\{P\}$;

[09] Atualize o número de cada espécie para refletir a execução da reação μ ;

[10] para-cada aresta (μ, α) no grafo de dependências DG:

[11] Atualize a_α ;

[12] se $\alpha \neq \mu$, faça $t_\alpha = (a_{\alpha,old}/a_\alpha)(t_\alpha - t) + t$;

[13] se $\alpha = \mu$, para um novo número aleatório r , faça $t_\alpha = (\ln(1/r)/a_\alpha) + t$;

[14] atualize o antigo t_α em P com o novo valor;

[15] fim-para-cada;

[16] fim-enquanto;

fim.

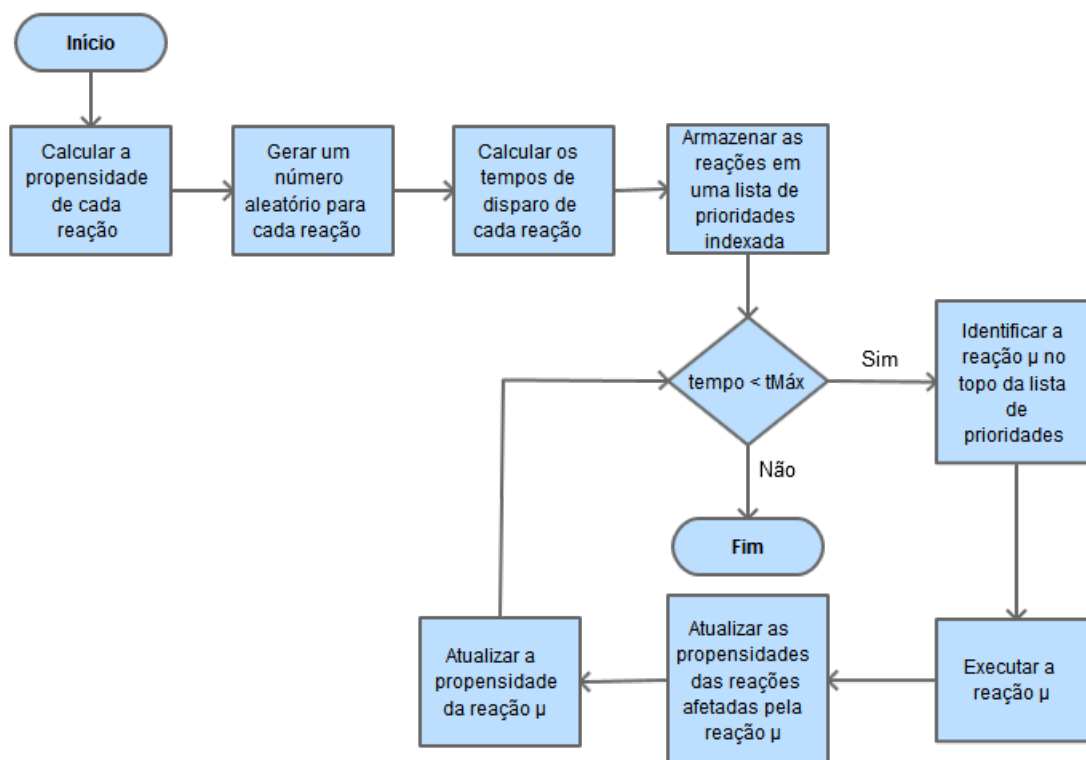


Figura 2.14: Fluxograma do algoritmo NRM.

Considerando-se:

1. k o maior grau de saída do DG ($k < M$);
2. M o número de reações do modelo;
3. A atualização da propensidade de uma reação como sendo constante.

A complexidade assintótica do NRM quando utiliza-se o DG é $O(k \log M)$. Essa complexidade é obtida devido à utilização da lista prioridades indexada. Se for considerado k como sendo constante, a complexidade do NRM torna-se $O(\log M)$.

Uma desvantagem do NRM é a necessidade de armazenar alguns valores anteriores para que o algoritmo possa reaproveitar os números aleatórios de reações que tiverem as suas propensidades zeradas e, posteriormente, voltaram a ter uma propensidade maior que zero. As linhas adicionadas ao algoritmo para tratar essas exceções aumentam a complexidade ciclomática do mesmo, o que pode aumentar as chances de erro de programação.

O próximo algoritmo apresentado é o MNRM, proposto por Anderson (2007). Esse algoritmo representa os tempos de inicialização de reações químicas como tempos de disparo de processos de Poisson com tempos internos fornecidos por funções de propensidades

integradas. Esse algoritmo utiliza $N + k$ números aleatórios, onde k é o número de reações do modelo e N é o número de reações executadas durante a simulação. O algoritmo 2.7 apresenta o pseudocódigo para o MNRM e a figura 2.15 apresenta o seu fluxograma.

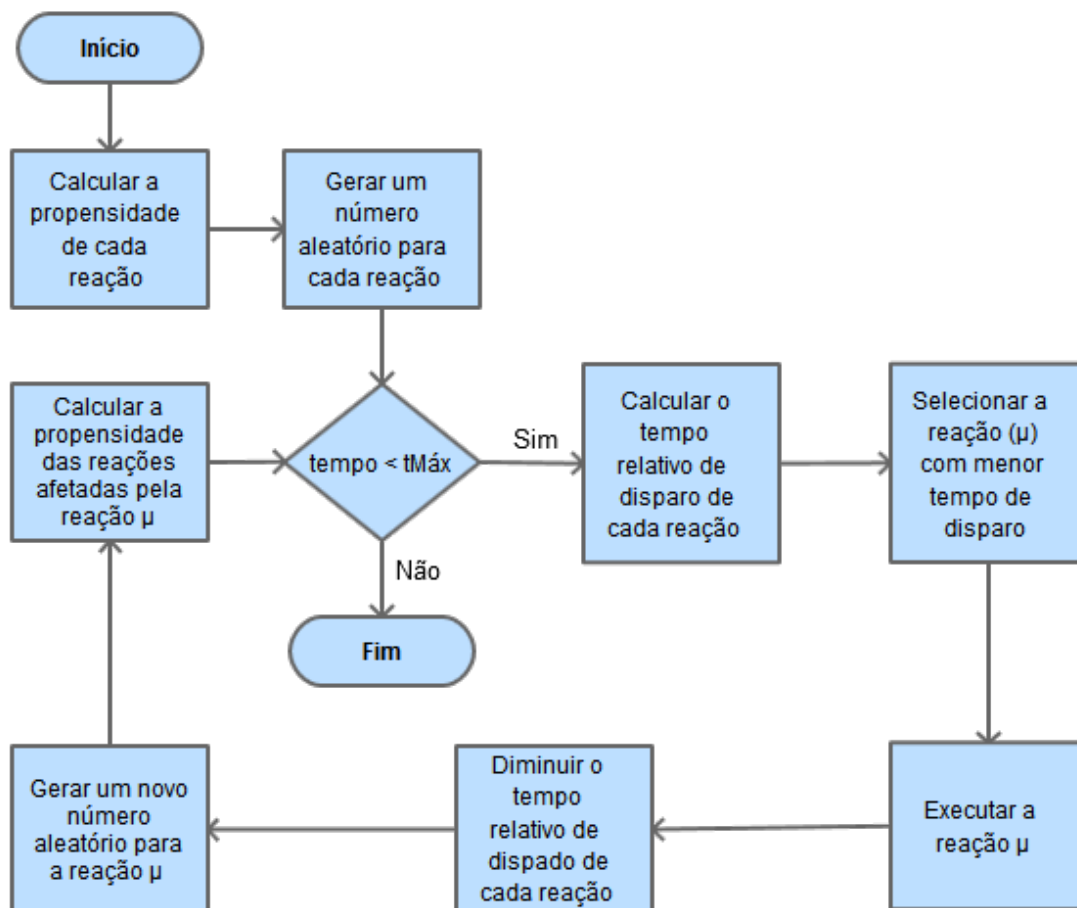


Figura 2.15: Fluxograma do algoritmo MNRM.

Algoritmo 2.7: Pseudocódigo para o algoritmo MNRM. No passo [13] utiliza-se o DG para atualizar apenas as propensidades das reações afetadas pela execução da reação μ . O passo [08], onde é selecionada a reação de menor tempo de disparo, pode ser feito durante o laço do passo [07].

Entrada: Estequiometria, estado inicial (S), tempo de simulação (T).

Saída: dinâmica dos estados (X).

início

[01] $T_k = 0$;

[02] para cada reação k

[03] Calcule a função de propensidade a_k ;

[04] Gere um número aleatório r no intervalo $(0,1)$; $P_k = \ln(1/r)$;

[05] fim-para;

[06] enquanto $t < T$;

[07] para cada reação k , $\Delta t_k = (P_k - T_k) / a_k$;

[08] $\theta = \Delta t_\mu = \min\{\Delta t\}$;

[09] Atualize o número de cada espécie para refletir a execução da reação μ ;

[10] para cada k , $T_k = T_k + a_k\theta$;

[11] Gere um número aleatório r no intervalo $(0,1)$; $P_\mu = P_\mu + \ln(1/r)$;

[12] $t = t + \theta$;

[13] para-cada aresta (μ, α) no grafo de dependências DG, calcule a_α ;

[14] fim-enquanto;

fim.

Considerando-se:

1. M o número de reações do modelo;
2. A atualização da propensidade de uma reação como sendo constante.

A complexidade assintótica do MNRM é $O(M)$, pois esse algoritmo, a cada reação executada, recalcula os tempos de disparo das M reações do modelo (linhas 7 e 10 do algoritmo 2.7).

O *Modified Next Reaction Method* não precisa tratar exceções como o NRM e não precisa lidar com o peso de processamento de uma *heap*, porém esse algoritmo trabalha com tempos relativos, o que torna necessário o recálculo dos tempos de disparo de todas

as reações a cada reação executada (passo [07]). Devido a isso, a complexidade do código executado a cada reação disparada cresce linearmente com o número de reações do modelo.

2.5.2 SSA PARA MODELOS COM ATRASO

Os primeiros trabalhos sobre simulação estocástica da expressão de genes tinham como tendência o foco em eventos de transcrição e tradução instantâneos, enquanto o atraso ocorrido nos processos estocásticos naturais recebia pouca atenção. Bratsun et al. (2005) propuseram uma generalização para o algoritmo de Gillespie que trabalha com eventos bioquímicos com atraso. A introdução do atraso em simulações estocásticas de redes de regulação de genes acabou contribuindo também para melhorar o desempenho computacional dessas simulações, uma vez que o alongamento na transcrição e tradução é modelado para uma única reação com atraso ao invés de várias reações. Ver equações 2.1 e 2.2.

A modelagem de sistemas de controle genético – no nível de detalhes necessário à utilização do SSA – torna-se proibitiva à medida que mais genes são adicionados. Uma ideia intuitiva para melhorar o desempenho do SSA é justamente reduzir o processo de múltiplos passos de alongamento para um simples processo com atraso que concentra somente a entrada e saída do processo (ver equação 2.2). Dessa forma, é possível tratar processos bioquímicos de múltiplos passos, como a transcrição e a tradução – sendo que cada um desses processos envolvem um grande número de reações elementares – como reações com atraso, na qual os eventos iniciais são separados do surgimento de produtos por um intervalo de tempo (ROUSSEL; ZHU, 2006). Um exemplo de um modelo sem atraso foi apresentado anteriormente no conjunto de equações 2.1 e o seu modelo equivalente com a utilização de atrasos foi apresentado no conjunto de equações 2.2.

Pelo fato dos DSSAs terem que manter a lista de produtos em espera, a complexidade de um DSSA pode ser pior que a do SSA em que esse DSSA foi baseado. Usando uma *heap* para manipular a lista de produtos em espera, a complexidade de atualização nessa lista será $O(\log L)$, onde L é número de elementos da lista de produtos em espera. Seja A a complexidade de qualquer SSA visto anteriormente, a complexidade do DSSA baseado neste mesmo algoritmo será $\max(A, O(\log L))$.

Na seção 4.3, a implementação da lista de produtos em espera é baseada em uma *hash* cuja complexidade de inserção e recuperação do menor valor tende a ser constante. Com isso, a complexidade do DSSA será a mesma do SSA em que o DSSA foi baseado.

A primeira versão do *Delayed Stochastic Simulation Algorithm* (DSSA) apresentada no presente trabalho chama-se *Rejection Method* (RM) e foi proposto por Bratsun et al. (2005). Esse método utiliza como base o método direto do SSA, acrescentando uma lista de produtos em espera. Esse método descarta um número aleatório sempre que o tempo da simulação alcança o menor tempo de retorno de um produto em espera. O algoritmo 2.8 apresenta o pseudocódigo do RM e o seu fluxograma é apresentado na figura 2.16.

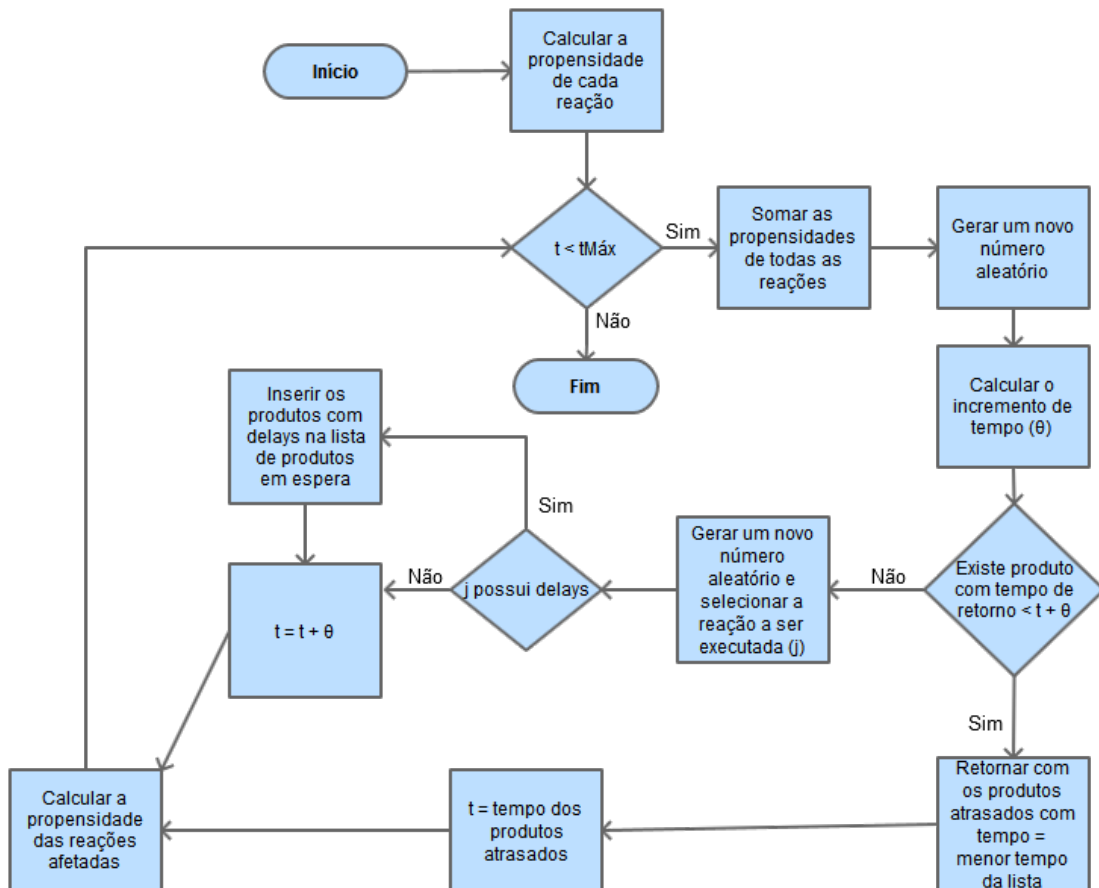


Figura 2.16: Fluxograma do algoritmo *Rejection Method*.

O próximo algoritmo apresentado é o *Modified Next Reaction Method for Delays* (DMNRM), proposto por Anderson (2007). Esse algoritmo utiliza como base o algoritmo MNRM com a diferença de trabalhar com o controle da lista de produtos em espera. O algoritmo 2.9 apresenta o pseudocódigo do DMNRM. O fluxograma deste algoritmo é apresentado na figura 2.17.

O algoritmo DMNRM não descarta números aleatórios a cada produto em espera retornado. Assim como o MNRM, o DMNRM trabalha com tempos relativos, o que torna necessário o recálculo do tempo de todas as reações a cada reação executada (conferir

Algoritmo 2.8: Pseudocódigo para o algoritmo RM. As principais diferenças do DSSA quando comparado ao DM (SSA) podem ser visualizadas nas linhas 7 e 18. Na linha 7, se o tempo em que ocorrerá a próxima reação superar o menor tempo de agendamento de um produto em espera (L_0), o sistema retornará com esse produto ao invés de executar a próxima reação. Na linha 18, o produto em espera p é adicionado à lista L , não alterando a quantidade de moléculas deste produto no tempo de simulação atual.

Entrada: Estequiometria, estado inicial (S), *delays* (d), tempo de simulação (T).

Saída: dinâmica dos estados (X).

início

[01] crie uma lista vazia de produtos em espera L ;

[02] para cada reação k , calcule a_k ;

[03] $a_0(t) = \sum_{j=1}^m a_j(t)$;

[04] enquanto $t < T$;

[05] gere U_1 como variável randômica no intervalo $(0, 1)$;

[06] $\theta = \frac{1}{a_0(t)} \ln(1/U_1)$;

[07] se tempo(L_0) está no intervalo $(t, t + \theta]$

[08] $\tau = \text{tempo}(L_0)$

[09] para-cada produto em espera p com tempo τ faça:

[10] retorne o produto em espera p ;

[11] para cada aresta (p, α) no grafo de dependências DG, calcule a_α e a_0 ;

[12] fim-para-cada;

[13] $t = \tau$;

[14] senão

[15] gere U_2 como variável randômica no intervalo $(0, 1)$;

[16] selecione j tal que $\sum_{k=1}^{j-1} a_k(t) < U_2 a_0(t) \leq \sum_{k=1}^j a_k(t)$;

[17] Atualize o número de cada espécie para refletir a execução da reação j ;

[18] Se j possui produtos com *delays*, insira-os em L ;

[19] $t = t + \theta$;

[20] para-cada aresta (j, α) no grafo de dependências DG, calcule a_α e a_0 ;

[21] fim-se;

[22] fim-enquanto;

fim.

Algoritmo 2.9: Pseudocódigo para o algoritmo DMNRM. As principais diferenças do DMNRM em relação à sua versão sem atraso (MNRM) estão entre as linhas 10 e 15. Na linha 10, o valor do próximo passo de tempo a ser utilizado é o menor valor entre o menor tempo de disparo das reações e o menor valor da lista de produtos em espera (L_0), subtraindo esse valor do tempo t em que a última reação foi disparada. Na linha 11 é verificado se o tempo selecionado é igual ao menor tempo de um produto em espera e, caso afirmativo, o produto em espera será retornado na linha 14, as propensidades das reações afetadas será atualizada na linha 15 e o tempo atual da simulação atualizado na linha 17. Na linha 20, o produto em espera j é adicionado à lista L , não alterando o número de moléculas do sistema nesse ponto.

Entrada: Estequiometria, estado inicial (S), *delays* (d), tempo de simulação (T).

Saída: dinâmica dos estados (X).

início

```

[01] crie uma lista vazia de produtos em espera  $L$ ;
[02]  $P_k = 0$ ;  $T_k = 0$ ;
[03] para cada reação  $k$ 
[04]     Calcule a função de propensidade  $a_k$ ;
[05]     Gere um número aleatório  $r_k$  no intervalo (0,1);  $P_k = \ln(1/r_k)$ ;
[06] fim-para;
[07] enquanto  $t < T$ ;
[08]     para cada reação  $k$ ,  $\Delta t_k = (P_k - T_k) / a_k$ ;
[09]      $\Delta t_\mu = \min\{\Delta t\}$ 
[10]      $\theta = \min\{\Delta t_\mu, L_0 - t\}$ 
[11]     se tempo( $L_0$ ) está no intervalo  $(t, t + \theta)$ 
[12]          $\tau = \text{tempo}(L_0)$ ;
[13]         para-cada produto em espera  $p$  com tempo  $\tau$  faça:
[14]             retorne o produto em espera  $p$ ;
[15]             para cada aresta  $(p, \alpha)$  no grafo de dependências DG, calcule  $a_\alpha$ ;
[16]         fim-para-cada;
[17]          $t = \tau$ ;
[18]     senão
[19]         Atualize o número de cada espécie para refletir a execução da reação  $\mu$ ;
[20]         Se  $\mu$  possui produtos com delays, insira-os em  $L$ ;
[21]         para cada  $k$ ,  $T_k = T_k + a_k \theta$ ;
[22]         Gere um número aleatório  $r$  no intervalo (0,1);  $P_\mu = P_\mu + \ln(1/r)$ ;
[23]          $t = t + \theta$ ;
[24]         para-cada aresta  $(\mu, \alpha)$  no grafo de dependências DG, calcule  $a_\alpha$ ;
[25]     fim-se;
[26] fim-enquanto;
fim.
```

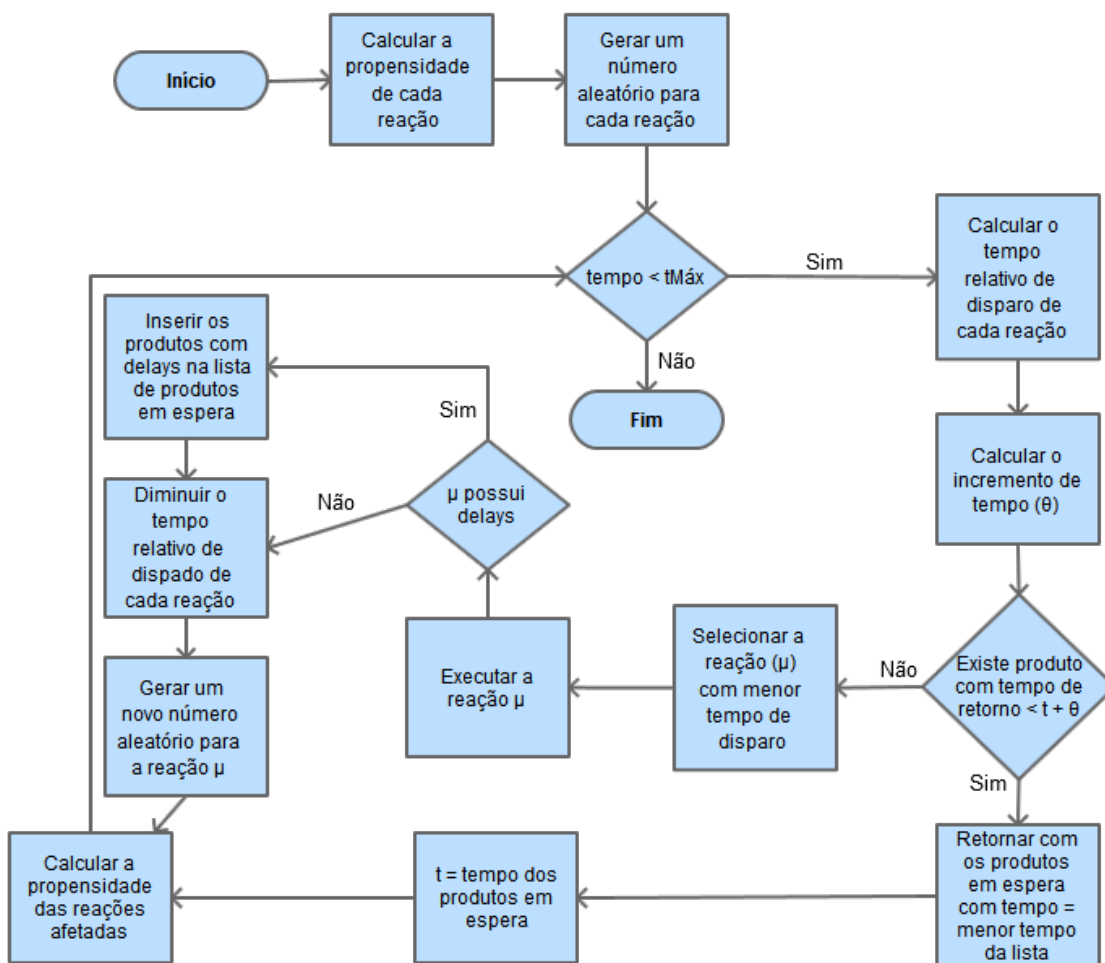


Figura 2.17: Fluxograma para o algoritmo DMNRM.

linhas 8 e 18 do algoritmo 2.9).

3 CONTRIBUIÇÕES

Esse capítulo apresenta algumas desvantagens – em relação ao desempenho de processamento – dos algoritmos apresentados no capítulo anterior. Essas desvantagens são discutidas e algumas melhorias são propostas. As propostas de melhorias para os algoritmos de simulação estocástica de redes de regulação gênica que não trabalham com atrasos são apresentadas na seção 3.1, enquanto as melhorias que trabalham especificamente com algoritmos que tratam atrasos encontram-se na seção 3.2.

3.1 CONTRIBUIÇÕES PARA O DESEMPENHO DAS SIMULAÇÕES SEM ATRASO

Essa seção apresenta um novo algoritmo, chamado *Simplified Next Reaction Method*, que é uma adaptação do *Modified Next Reaction Method*. A cada reação executada, esse novo algoritmo recalcula somente o tempo de disparo das reações afetadas pela reação executada. Com essa contribuição, um passo de complexidade linear do algoritmo MNRM pode ser substituído por um passo de complexidade $O(k \log n)$, onde k é o maior grau de saída dos vértices do grafo de dependências utilizado.

3.1.1 VERSÃO SIMPLIFICADA DO ALGORITMO NRM

Devido à sua implementação simplificada e ao não descarte de números aleatórios, o MNRM apresenta um bom desempenho, mas esse algoritmo atualiza todos os tempos de disparo das reações a cada reação executada (ver linhas 8 e 21 do algoritmo 2.7), o que faz com que, em sistemas com um número elevado de reações, o MNRM possa não manter um desempenho à altura do NRM, que atualiza somente as reações afetadas pela reação executada (via grafo de dependências DG).

O presente trabalho propõe o algoritmo *Simplified Next Reaction Method* (SNRM), baseado no MNRM. Esse algoritmo é simples de implementar, não apresenta exceções para o cálculo dos tempos de disparo e possui tempos de disparo atualizados somente para as reações afetadas pela reação executada.

De forma diferente à do MNRM, o SNRM não trabalha com tempos relativos e, devido

a isso, existe a necessidade de guardar o tempo (U) da última atualização do tempo de disparo de cada reação. No tempo de simulação t , sendo U_k o tempo da última atualização da propensidade da reação k e P_k o primeiro tempo interno após T_k no qual a reação k é executada, podemos escrever a fórmula de atualização de T_k como:

$$T_k = T_k + a_k(t - U_k) \quad (3.1)$$

Com isso, a fórmula de recálculo do tempo de disparo (τ) passa a ser:

$$\tau_k = ((P_k - T_k) / a_k) + t \quad (3.2)$$

O algoritmo 3.1 apresenta o pseudocódigo do SNRM. A figura 3.1 apresenta o fluxograma para o algoritmo SNRM.

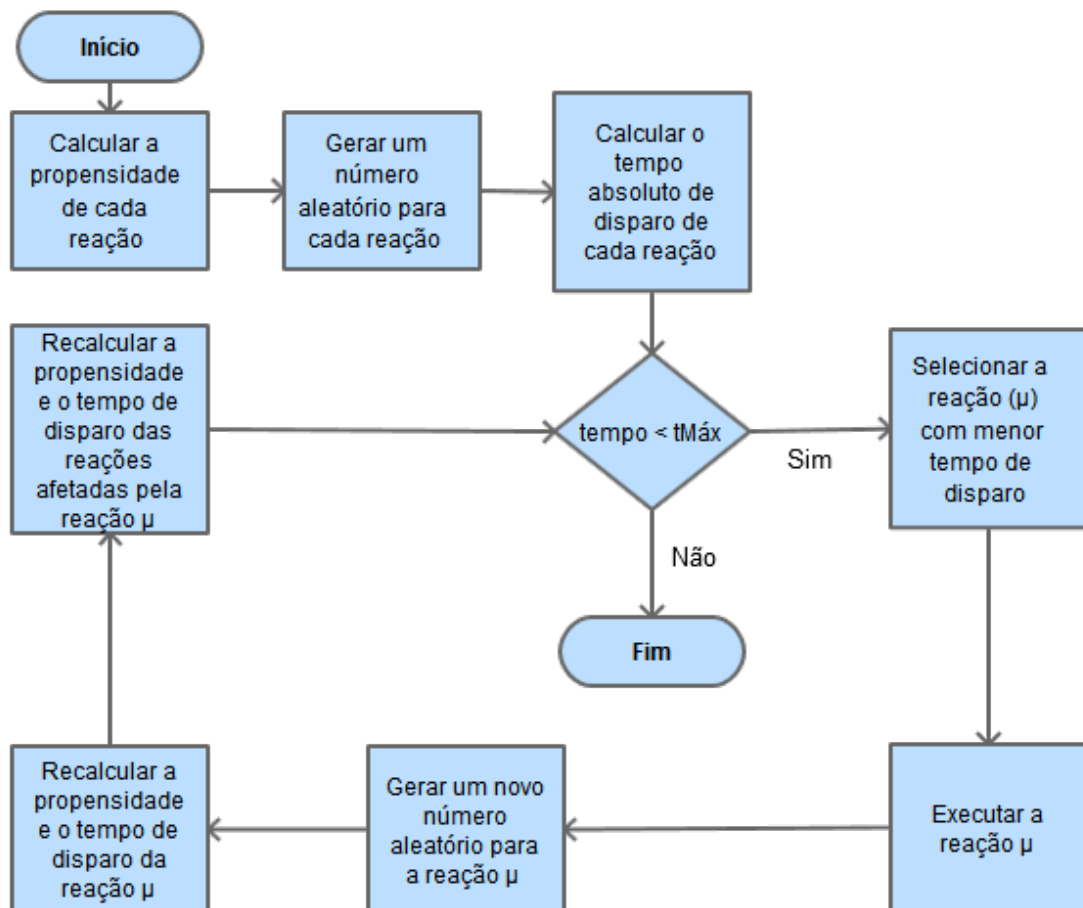


Figura 3.1: Fluxograma para o algoritmo SNRM.

O algoritmo SNRM é equivalente ao MNRM e a principal diferença entre os dois algoritmos é a estratégia de atualização do tempo de disparo de cada reação. Enquanto

Algoritmo 3.1: Pseudocódigo para o algoritmo SNRM.

Entrada: Estequiometria, estado inicial (S), *delays* (d), tempo de simulação (Z).

Saída: dinâmica dos estados (X).

início

[01] para cada reação k

[02] Calcule a função de propensidade a_k ;

[03] Gere um número aleatório r no intervalo $(0,1)$;

[04] $P_k = \ln(1/r)$; $U_k = t$;

[05] $\tau_k = (P_k - T_k) / a_k + t$;

[06] Armazene τ_k em uma fila de prioridades indexada H ;

[07] fim-para;

[08] enquanto $t < Z$;

[09] $t = \tau_\mu = H_0$;

[10] Atualize o número de cada espécie para refletir a execução da reação μ ;

[11] Gere um número aleatório r no intervalo $(0,1)$;

[12] $P_\mu = P_\mu + \ln(1/r)$;

[13] para-cada aresta (μ, α) , no grafo de dependências DG:

[14] $T_\alpha = T_\alpha + a_\alpha(t - U_\alpha)$; $U_\alpha = t$;

[15] Atualize a_α ;

[16] $\tau_\alpha = (P_\alpha - T_\alpha) / a_\alpha + t$;

[17] atualize a posição de τ_α em H ;

[18] fim-para-cada;

[19] fim-enquanto;

fim.

o MNRM atualiza o tempo de disparo restante para a execução de cada reação (linhas 8 e 21 do algoritmo 2.9), o SNRM atualiza somente os tempos de disparos das reações afetadas pelo grafo de dependência (linhas 14 e 16 do algoritmo 3.1). Isso é possível devido ao SNRM trabalhar com tempos absolutos ao invés de tempos parciais. A fórmula de transformação de T_k usada no SNRM utiliza a mesma estratégia do NRM: fazer a transformação calculando o Δ como a diferença entre o tempo atual e o tempo da última atualização.

Considerando-se:

1. k o maior grau de saída do DG ($k < M$);
2. M o número de reações do modelo;
3. A atualização da propensidade de uma reação como sendo constante.

A complexidade assintótica do SNRM quando utiliza-se o DG é $O(k \log M)$. Essa complexidade é obtida devido à utilização da lista prioridades indexada. Se for considerado k como sendo constante, a complexidade do NRM torna-se $O(\log M)$.

Em vários testes realizados, o desempenho dos algoritmos SNRM e NRM foi semelhante. É importante destacar que a implementação do SNRM é mais simples que a do NRM, pois o NRM necessita de diversos controles extras para tratar as exceções à sua fórmula de reaproveitamento de números aleatórios.

3.2 CONTRIBUIÇÕES PARA O DESEMPENHO DAS SIMULAÇÕES COM ATRASO

Essa seção apresenta um novo grafo de dependências para reações com atraso. Também é apresentada uma estrutura de dados que proporciona um melhor desempenho que a *heap* para controle da lista de produtos em espera e um novo algoritmo que é uma adaptação do *Simplified Next Reaction Method* para Reações com Atraso.

3.2.1 GRAFO DE DEPENDÊNCIAS PARA REAÇÕES COM ATRASO

Para aumentar o desempenho do SSA, Gibson e Bruck (2000) propuseram o Dependency-Graph, que é um dígrafo $G(V, E)$ onde V é o conjunto de reações $\{v_1, v_2, \dots, v_N\}$ e

E é o conjunto de arestas direcionadas v_i para v_j , sendo que essas arestas respeitam a restrição $(Reactants(v_i) \cup Products(v_i)) \cap Reactants(v_j) \neq \emptyset$. Em algumas situações, pode ser útil trabalhar com dependências adicionais, sendo que para esse caso a restrição é $Affects(v_i) \cap DependsOn(a_{v_j}) \neq \emptyset$, onde $Affects(v_i)$ é o conjunto de espécies do modelo que têm a sua quantidade alterada quando a reação μ é executada e $DependsOn(a_{v_j})$ é o conjunto de espécies do modelo que afetam o valor da propensidade da reação v_j .

A utilização do DG geralmente melhora o desempenho dos algoritmos de simulação estocástica de redes de regulação gênica. Isso também é válido para os algoritmos que trabalham com atrasos, mas para esse caso é comum o DG ainda obrigar esses algoritmos a realizar mais cálculos que o necessário.

O presente trabalho propõe um novo grafo de dependências para reações com atraso. Esse grafo é chamado *Delayed Dependency Graph* (DDG) e foi baseado no *Dependency Graph* (DG), proposto por Gibson e Bruck (2000), sendo o DDG utilizado somente em reações com atraso.

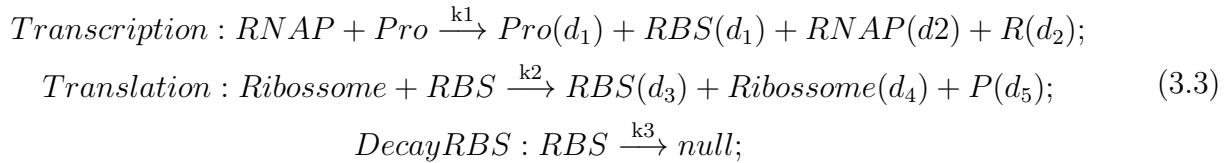
O DDG é um dígrafo $G(V, E)$ onde V é a união entre o conjunto de substâncias $\{s_1, s_2, \dots, s_N\}$ e o conjunto de reações $\{v_1, v_2, \dots, v_N\}$. Para esse dígrafo, E é o conjunto de arestas direcionadas s_i para v_j que respeitem a restrição $s_i \cap Reactants(v_j) \neq \emptyset$. Em algumas situações, pode ser útil trabalhar com dependências adicionais, sendo que para esse caso a restrição é $s_i \cap DependsOn(a_{v_j}) \neq \emptyset$, onde $DependsOn(a_{v_j})$ é o conjunto de substâncias que afetam o valor da propensidade da reação v_j .

Utilizando o DG, a execução de uma reação v_μ é o evento que dispara a atualização das propensidades das reações que são afetadas pela reação v_μ no SSA. Com o DDG, esse evento é a alteração no número de moléculas do produto em espera retornado. Em sistemas com múltiplos atrasos que trabalham com o DG, a atualização das propensidades pode ser feita diversas vezes, mesmo que desnecessário. Essa atualização pode ocorrer no momento em que a reação for executada e em todas as vezes que um produto em espera for alterado, uma vez que a volta de um produto atrasado da reação v_μ utiliza o grafo da mesma forma que o disparo da reação v_μ .

Considere uma espécie química A cujo número de moléculas foi modificado pelo DSSA utilizando o DDG. Dessa forma, o DDG irá possibilitar ao algoritmo calcular somente as propensidades das reações cuja espécie A é reagente.

Para exemplificar o DDG, considere o modelo de transcrição e tradução gênica pro-

posto por Zhu et al. (2007), que encontra-se no conjunto de equações 3.3. A palavra *null* à direita da última reação significa que o reagente dessa reação será degradado e que o seu produto não é relevante para o modelo.



A figura 3.2 apresenta o grafo DG enquanto a figura 3.3 apresenta o grafo DDG, ambos para o modelo 3.3. Considerando-se o retorno do produto atrasado RNAP pode-se exemplificar a utilização dos dois grafos. Utilizando o DG para esse exemplo, a propensidade de todas as reações (Transcription, Translation e Decay RBS) precisam ser atualizadas (figura 3.2). Já com a utilização do DDG, o retorno do RNAP provoca somente a atualização da reação Transcription (figura 3.3).

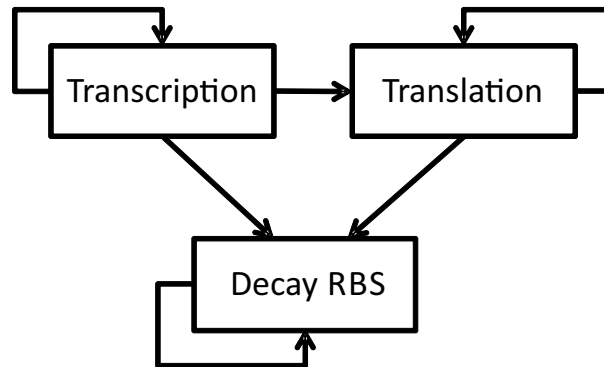


Figura 3.2: Exemplo do Grafo de Dependências - DG para o modelo 3.3.

É importante observar que o DG descreve relações entre as reações, enquanto o DDG descreve relações entre espécies e reações (ver figuras 3.2 e 3.3).

O desenvolvimento do DDG permite um ganho de desempenho do DSSA, mas pode não substituir por completo o DG, devido às reações sem atraso e às alterações imediatas nas quantidades de determinadas espécies pertencentes a reações com atraso. Utilizar o DG em conjunto com o DDG pode ser a melhor alternativa, de forma que somente o retorno dos produtos da lista de produtos em espera deve utilizar o DDG, enquanto o disparo das reações deve utilizar o DG.

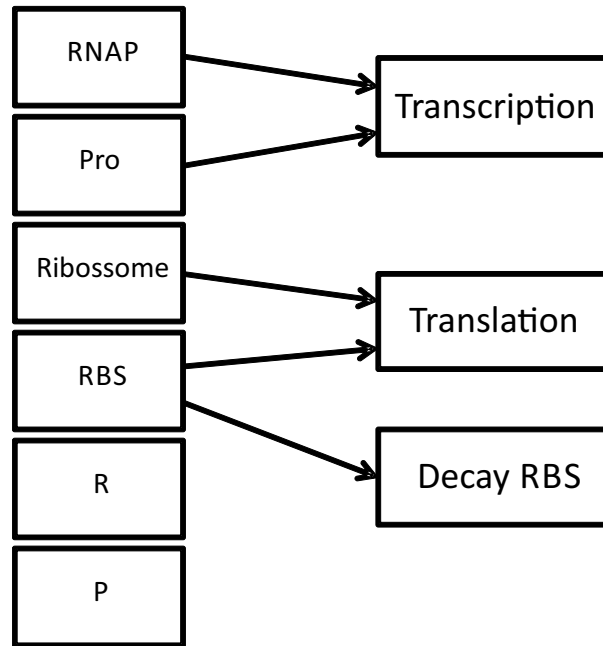


Figura 3.3: Exemplo do Grafo de Dependências - DDG para o modelo 3.3.

3.2.2 VERSÃO SIMPLIFICADA DO ALGORITMO NRM PARA TRATAR REAÇÕES COM ATRASOS

No presente trabalho, o algoritmo SNRM foi adaptado para trabalhar com reações com atrasos, não sendo necessário descartar números aleatórios a cada produto retornado. Com essa abordagem, foi obtido um algoritmo que tende a ter um desempenho superior ao RM e ao MNRM em redes de regulação de genes com muitas reações e múltiplos atrasos.

A alteração que faz com que o SNRM possa trabalhar com reações com atraso é simples. Nessa configuração, o algoritmo possui mais de uma vantagem em relação aos métodos derivados do método direto:

1. O SNRM não descarta números aleatórios;
2. Ele não precisa de uma implementação complexa para que seja possível o não descarte de números aleatórios;
3. Após identificar os tempos iniciais de cada reação, o SNRM utiliza somente um número aleatório para cada reação executada.

Devido à utilização dos grafos de dependências (DG e DDG), o SNRM tende a apresentar melhores resultados em redes fracamente acopladas, porém, devido a duas vantagens (1 e 3), o algoritmo pode apresentar um bom desempenho mesmo em redes fortemente

Algoritmo 3.2: Pseudocódigo para o algoritmo DSNRM. As principais diferenças entre o SNRM e o DSNRM podem ser notadas à partir da linha 10. Nessa linha o valor do próximo passo de tempo a ser utilizado é o menor valor entre o menor tempo da lista de prioridades das reações (H_0) e o menor valor da lista de produtos em espera (L_0), subtraindo esse valor do tempo t em que a última reação foi disparada. Na linha 11, o algoritmo checa se existe produto em espera a ser retornado. Entre as linhas 12 e 21 o algoritmo atualiza o número de moléculas de todas as espécies que possuem o tempo de retorno igual ao tempo corrente e atualiza os tempos de disparo das reações afetadas pelo grafo de dependências DDG.

Entrada: Estequiometria, estado inicial (S), *delays* (d), tempo de simulação (Z).

Saída: dinâmica dos estados (X).

início

```

[01] crie uma lista vazia de produtos em espera  $L$ ;
[02] para cada reação  $k$ 
[03]     Calcule a função de propensidade  $a_k$ ;
[04]     Gere um número aleatório  $r$  no intervalo  $(0,1)$ ;
[05]      $P_k = \ln(1/r)$ ;
[06]      $\tau_k = (P_k - T_k) / a_k$ ;
[07]     Armazene  $\tau_k$  em uma fila de prioridades indexada  $H$ ;
[08] fim-para;
[09] enquanto  $t < Z$ ;
[10]      $\theta = \min\{H_0, L_0\} - t$ ;
[11]     se tempo( $L_0$ ) está no intervalo  $(t, t + \theta]$ 
[12]          $t = \text{tempo}(L_0)$ ;
[13]         para-cada reação  $k$  contendo um produto com delay  $q$  no tempo  $t$ ;
[14]             Atualize o número de moléculas de  $q$ ;
[15]             para-cada aresta  $(k, \alpha)$ , no grafo de dependências DDG:
[16]                  $T_\alpha = T_\alpha + a_\alpha(t - U_\alpha)$ ;  $U_\alpha = t$ ;
[17]                 Atualize  $a_\alpha$ ;
[18]                  $\tau_\alpha = ((P_\alpha - T_\alpha) / a_\alpha) + t$ ;
[19]                 atualize a posição de  $\tau_\alpha$  em  $H$ ;
[20]             fim-para-cada;
[21]         fim-para-cada;
[22] senão
[23]      $t = \tau_\mu = H_0$ ;
[24]     Atualize o número de cada espécie para refletir a execução da reação  $\mu$ ;
[25]     Gere um número aleatório  $r$  no intervalo  $(0,1)$ ;
[26]      $P_\mu = P_\mu + \ln(1/r)$ ;
[27]     para-cada aresta  $(\mu, \alpha)$ , no grafo de dependências DG:
[28]          $T_\alpha = T_\alpha + a_\alpha(t - U_\alpha)$ ;  $U_\alpha = t$ ;
[29]         Atualize  $a_\alpha$ ;
[30]          $\tau_\alpha = ((P_\alpha - T_\alpha) / a_\alpha) + t$ ;
[31]         atualize a posição de  $\tau_\alpha$  em  $H$ ;
[32]     fim-para-cada;
[33] fim-se;
[34] fim-enquanto;
fim.
```

acopladas, se comparado com o algoritmo RM. Já quando comparado ao MNRM com atraso, o SNRM com atraso possui a vantagem de atualizar somente os tempos das reações afetadas pelo grafo de dependências.

A figura 3.4 apresenta o fluxograma para o DSNRM.

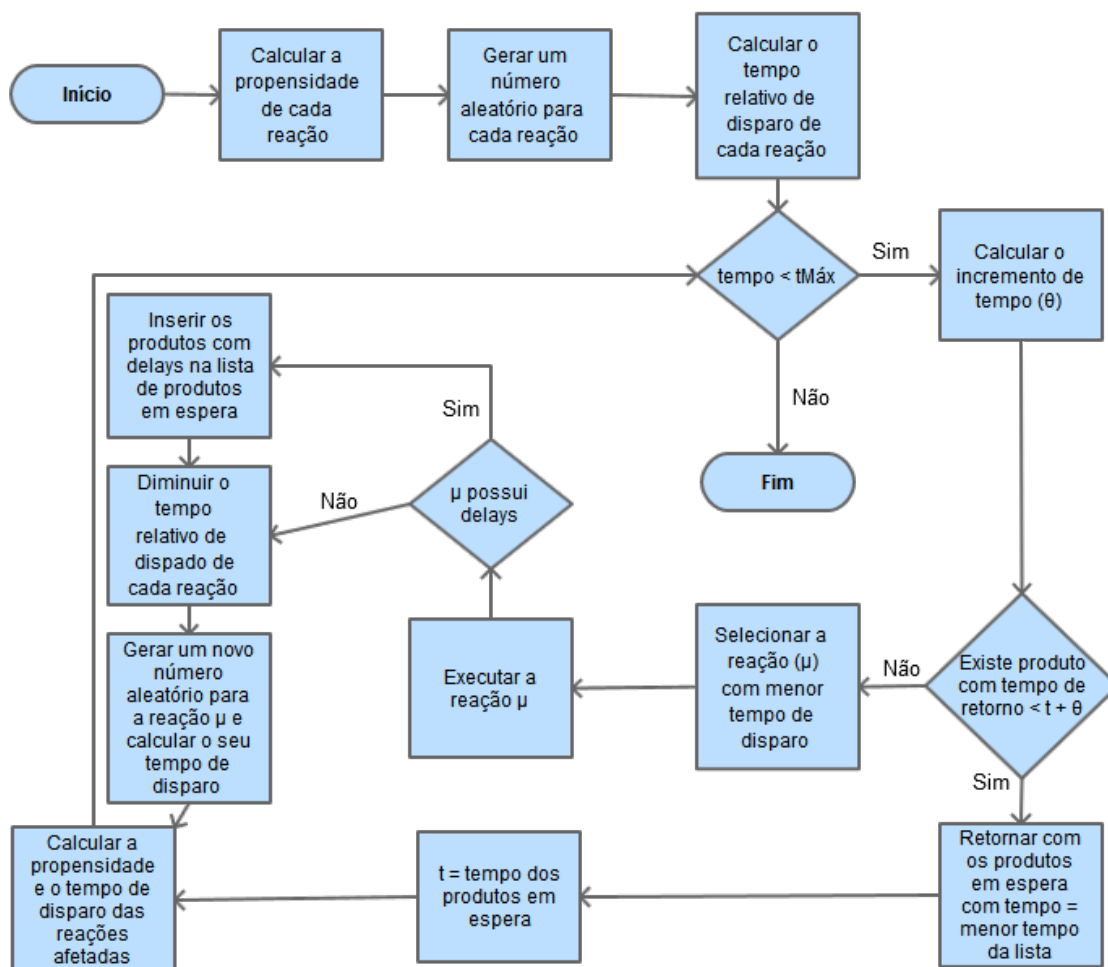


Figura 3.4: Fluxograma para o algoritmo DSNRM.

O algoritmo 3.2 apresenta o pseudocódigo do DSNRM.

3.2.3 ESTRUTURA DE DADOS LISTA ORDENADA POR *HASHING* PARA CONTROLE DA LISTA DE PRODUTOS EM ESPERA

Uma boa escolha para controlar a lista de produtos em espera no DSSA é a estrutura de dados conhecida como *heap*, pois essa estrutura é adequada para retornar o menor valor em um conjunto de valores (ver subseção 2.3.3). Em uma *heap*, o método *heapify*, que atualiza a posição de um item na *heap*, é disparado para cada item inserido ou removido. Apesar

de ser uma boa opção para uma lista de produtos em espera, o custo computacional do método *heapify* pode tornar a *heap* uma estrutura de dados não otimizada para o controle da lista de produtos em espera.

Com algumas alterações em uma tabela *hash* é possível obter uma estrutura de dados para ordenação de itens. Para isso, é necessário definir o vetor da tabela *hash* de forma que ele possa contemplar todos os possíveis valores que serão ordenados, o que pode tornar a ordenação por tabela *hash* dispendiosa. Essa tabela *hash* então passaria a não possuir método de busca de item por chave, pois o seu vetor passaria a ser percorrido do menor índice para o maior, o que possibilitaria que esses mesmos itens fossem retornados de forma ordenada. Ainda para possibilitar o funcionamento dessa tabela *hash*, a sua função *hash* precisa inserir os itens nas posições do vetor de forma que os mesmos fiquem em ordem crescente quando o vetor for percorrido do menor índice para o maior. Para não confundir essa estrutura de dados alterada com a tabela *hash* tradicional, a nova estrutura será chamada de lista ordenada por *hashing*.

É possível identificar alguns padrões na entrada de itens na lista de produtos em espera do DSSA. No presente trabalho esses padrões são explorados para fazer uma simples lista ordenada por *hashing* superar o desempenho da *heap* utilizada no DSSA.

Os padrões identificados são:

1. O tempo agendado, que será inserido em uma *heap*, está sempre em um tempo futuro e nunca pode estar atrás do tempo atual da simulação;
2. Quando o tempo de simulação para um produto em espera é alcançado, esse produto é removido da *heap*.

Considerando esses padrões, é possível estruturar uma função *hash* que é uma função linear da diferença entre o tempo de agendamento do produto em espera e o tempo corrente da simulação. Com essa função, é possível montar uma lista ordenada por *hashing* para o controle da lista de produtos em espera que geralmente é mais eficaz que a *heap*.

Analisando as operações de atualização de itens em uma *heap* de tamanho n , obtêm-se $O(\log n)$ como complexidade assintótica para as operações de inserção e remoção. Fazendo a mesma análise na estrutura de dados de lista ordenada por *hashing*, obtêm-se $O(1)$ como complexidade para a operação de inserção e $O(n)$ como complexidade para a operação de remoção. Apesar da complexidade linear para a operação de remoção da

estrutura com *hashing*, o tempo real gasto nessa operação pode não ser bem representado pela complexidade assintótica da mesma. Isso ocorre devido a não ser necessário percorrer a lista desde o seu início até que se encontre a sua posição preenchida de menor tempo. Somente a diferença entre o tempo de simulação atual e o tempo futuro precisa ser percorrida, sendo que esse controle é natural para o algoritmo DSSA.

Para o correto entendimento da lista ordenada por *hashing*, considere L como sendo uma lista circular, d_{max} como o valor do maior atraso existente em uma determinada rede de regulação gênica e n como o total de nós da lista circular dessa rede e g como o intervalo de tempo que cada nó representa. Assim, temos:

$$\sum_{i=1}^n g = ng > d_{max} + g; \quad (3.4)$$

Dessa forma, o tempo total representado pela lista ordenada por *hashing* deve ser maior que o tempo do maior atraso somado ao tempo do intervalo que cada nó representa. Essa restrição é necessária para que a distribuição dos produtos em espera seja feita na lista ordenada por *hashing* de forma que um produto em espera colocado no nó L_i tenha seu tempo de retorno ao sistema menor que o do produto colocado no nó L_{i+x} , sendo $\{i + x \in Z \mid i + x < n\}$. Para ser mais preciso, a regra é um pouco mais complexa: considerando que k é o índice que controla a casa atual da lista ordenada por *hashing*, então deve ser respeitada a seguinte restrição:

$$\begin{aligned} L_i > L_{i+x} \mid i < k < i + x, \\ L_i < L_{i+x} \mid k \leq i \text{ ou } k \geq i + x. \end{aligned} \quad (3.5)$$

Sendo w o índice até o qual o algoritmo deve caminhar na lista ordenada por *hashing*, t o tempo corrente da simulação, t_u o último tempo em que k foi atualizado e θ o próximo passo de tempo que será dado, temos:

$$w = k + \left\lfloor \frac{t + \theta - t_u}{g} \right\rfloor \quad (3.6)$$

Dessa forma, o algoritmo caminha $w - k$ nós em uma iteração.

Um produto atrasado é representado por um tempo de agendamento e por um índice de uma espécie. A função f_1 retorna o tempo agendado para um determinado produto em espera. Se a casa atual da lista ordenada por *hashing* é L_k , o primeiro produto em espera

P contido nessa casa é representado por $L_{k,0}$ e deve ser retornado quando $f_1(L_{k,0}) \leq t + \theta$, ou seja, esse produto deve ser retornado quando k atinge a casa onde P está e quando o tempo de agendamento de P for menor ou igual ao próximo tempo sorteado.

O algoritmo 3.3 é um pequeno trecho da implementação do retorno de produtos em espera utilizando uma lista ordenada por *hashing* L . Nesse algoritmo, os produtos em espera são retornados em acordo com a evolução do tempo de simulação.

Como o algoritmo 3.3 mostra em detalhes a lista L do algoritmo RM utilizando uma lista ordenada por *hashing*, as variáveis L , t e θ são as mesmas do algoritmo RM. É importante destacar que, para todos os algoritmos apresentados nesse trabalho que trabalham com reações com atrasos, a lista ordenada por *hashing* pode ser utilizada para o controle da lista de produtos em espera.

Quanto à inserção de itens na lista ordenada por *hashing*, se o tempo do atraso do produto P for t_1 , o nó de índice j para o qual esse produto deve ser inserido na lista ordenada por *hashing* é igual a:

$$j = \left\lfloor \frac{(t + t_1) \bmod \sum_{i=1}^n g}{g} \right\rfloor \quad (3.7)$$

Após a identificação da casa j , deve-se percorrer a lista ligada L_j e inserir P na posição correta de seu tempo $(t + t_1)$, conforme o algoritmo 3.4, que detalha a inserção de um item na lista L quando se utiliza uma lista ordenada por *hashing*.

A figura 3.5 apresenta um exemplo de um vetor de listas ligadas de uma lista ordenada por *hashing*.

Devido à evolução do tempo nas simulações estocásticas não ocorrer a passos fixos, nota-se a possibilidade de esses passos randômicos serem utilizados como vantagem para a distribuição dos atrasos em uma lista ordenada por *hashing*. Isso devido ao aumento das propensidades das reações fazerem com que a evolução do tempo seja mais lenta. Logo, em simulações que envolvem muitas moléculas, as espécies que entrarem na lista de produtos em espera tendem a estar mais próximas umas das outras, ou até mesmo a ocuparem o mesmo índice do vetor. Até certo ponto, isso pode aumentar a velocidade dos retornos dos produtos, sendo que ultrapassado esse ponto, a necessidade de ordenação da lista encadeada pode prejudicar o desempenho da simulação.

É importante destacar que a afirmação sobre a proximidade dos produtos em espera

Algoritmo 3.3: Remoção de produtos em espera em uma lista ordenada por *hashing*. A função f_2 retorna o índice da espécie de um produto em espera. A variável k controla qual o nó a ser verificado na lista ordenada por *hashing*, sendo que essa variável é incrementada se nenhum produto for retornado e se a diferença entre o próximo tempo de simulação ($t + \theta$) e o tempo de simulação da última atualização da reação (t_u) ultrapassar o intervalo de tempo g (linha 14). Para identificar se existem produtos em espera a serem retornados, a primeira posição do nó L_k deve ser consultado (linha 5). Na atualização do valor da variável k , considerando-se w como sendo o próximo valor de k , todos os nós entre k e w devem ser verificados ($L_{[k..w]}$), exceto se algum produto for retornado.

Entrada: tempo atual da simulação (t), intervalo de tempo que cada nó da lista ordenada por *hashing* representa (g), valor do próximo salto de tempo (θ), lista de produtos em espera (L) e tempo da última atualização de k (T_u).

início

[01] faça

[02] defina td ;

[03] defina $termine = \text{falso}$;

[04] se $L_{k,0} \neq \text{NULL}$ então

[05] se $f_1(L_{k,0}) < (t + \theta)$ então

[06] $td = f_1(L_{k,0})$;

[07] $t = td$;

[08] enquanto $f_1(L_{k,0}) = td$ faça

[09] retire $L_{k,0}$ e atualize o número de moléculas de $f_2(L_{k,0})$;

[10] fim-enquanto;

[11] fim-se;

[12] $termine = \text{verdadeiro}$;

[13] fim-se;

[14] se $termine = \text{falso}$ e $(t + \theta - t_u) > g$ então

[15] $t_u = t_u + g$;

[16] $k = k + 1$;

[17] fim-se;

[18] repetir enquanto $termine = \text{falso}$ e $(t + \theta - t_u) > g$;

fim.

Algoritmo 3.4: Inserção de item na lista ordenada por *hashing*.

Entrada: tempo atual da simulação (t), tamanho da lista ordenada por *hashing* (n), intervalo de tempo que cada nó da lista ordenada por *hashing* representa (g), lista ordenada por *hashing* para controle dos produtos em espera (L), produto em espera (P) e atraso do produto $P(t_1)$.

início

[01] $j = \left\lfloor \frac{(t+t_1) \bmod \sum_{i=1}^n g}{g} \right\rfloor$

[02] Insira P em L_j , em posição que mantenha os produtos em espera de L_j ordenados pelos seus respectivos tempos de retorno;

fim.

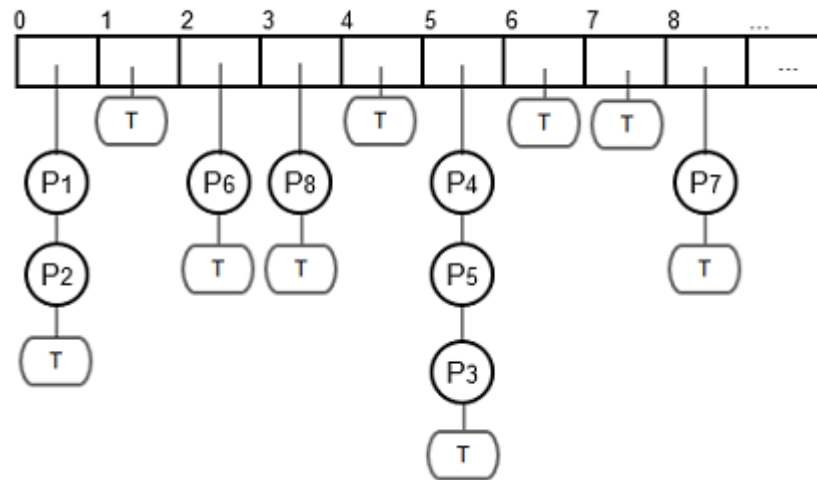


Figura 3.5: Exemplo de vetor de listas ligadas com 8 produtos atrasados inseridos: $\{P_1, P_2, \dots, P_8\}$. Cada item do vetor possui uma estrutura dinâmica chamada lista ligada. O último item de uma lista ligada chama-se terminador e identifica que não existem mais produtos atrasados naquela posição do vetor.

em simulações com reações com propensidades elevadas considera que o número de valores distintos de atrasos da rede é muito menor que o número médio de produtos em espera em uma simulação.

Existe a necessidade de escolher corretamente o tamanho da lista ordenada por *hashing*, pois tanto uma lista muito grande quanto uma lista muito pequena podem impactar negativamente no desempenho da simulação.

A figura 3.6 apresenta a estrutura lista ordenada por *hashing*.

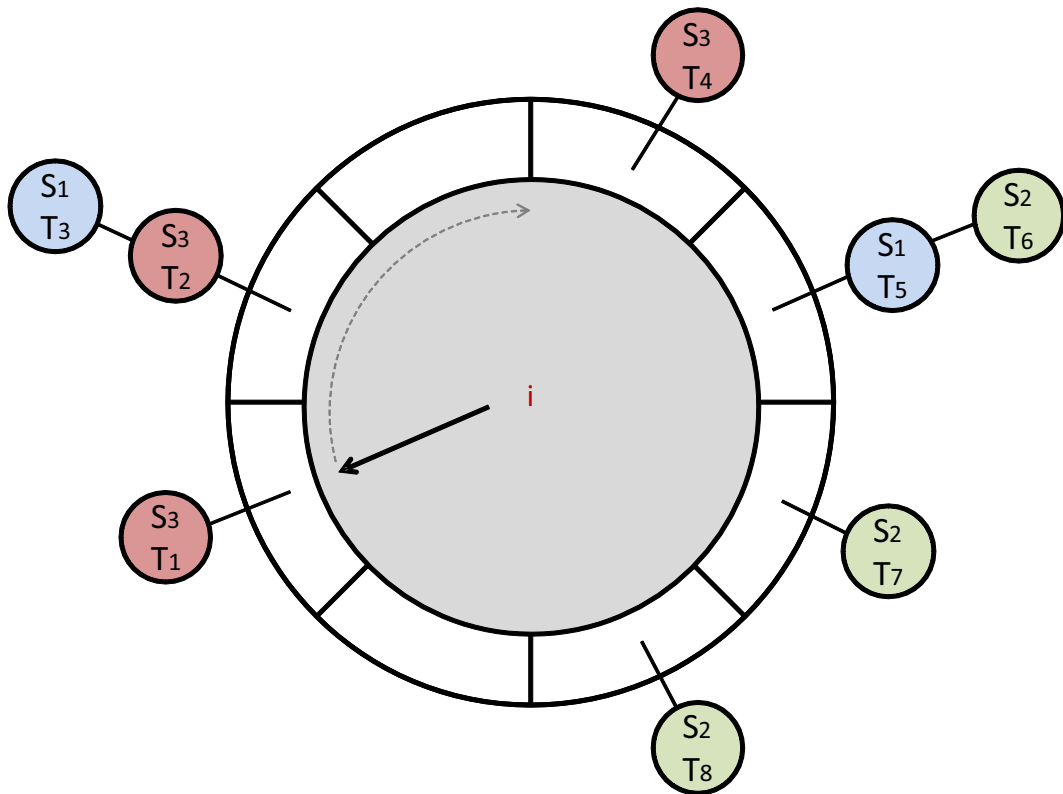


Figura 3.6: Estrutura de dados lista ordenada por *hashing* com 8 espécies S inseridas. Apenas para facilitar o entendimento dessa estrutura de dados, os índices dos tempos de retorno T estão ordenados de acordo com a ordem em que essas espécies serão retiradas. A estrutura dados é uma lista circular cujo tamanho é proporcional ao tamanho do maior atraso existente no modelo. Essa lista é percorrida no sentido horário e os produtos em espera são distribuídos na lista pela diferença entre o seu tempo de agendamento e o tempo atual da simulação. Como essa diferença não pode ser maior que o tempo do maior atraso, a utilização de uma lista circular suficientemente pequena de forma que consiga representar somente a diferença de tempo do maior atraso já é o necessário para que o algoritmo consiga percorrer a lista e retirar os produtos em espera em ordem. As colisões são resolvidas por listas encadeadas com elementos ordenados.

4 RESULTADOS

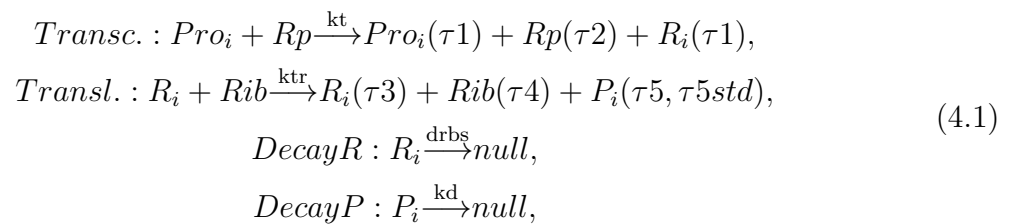
Os resultados dos testes das contribuições do presente trabalho são apresentados e discutidos neste capítulo. O DDG é a primeira contribuição cujos resultados dos testes são apresentados. Em seguida, os resultados para o SNRM com atraso e, posteriormente, os resultados para o algoritmo RM utilizando a lista ordenada por *hashing* para a lista de produtos em espera.

Como os algoritmos utilizados trabalham com atrasos, o número de atualizações de espécies por segundo foi utilizado no lugar do número de reações por segundo. Dessa forma, tanto o disparo de uma reação quanto o retorno de um produto atrasado são considerados para medir os desempenhos dos algoritmos nas simulações.

Todos os testes foram executados em um microcomputador com processador Intel Core i7 com 8GB de memória RAM e sistema operacional Microsoft Windows 7 Ultimate. Os algoritmos foram codificados na linguagem de programação C#.

4.1 DDG

Para testar o ganho de desempenho proporcionado pela utilização do DDG em substituição ao DG no retorno de produtos em espera, foi utilizado o algoritmo RM em duas implementações: RM + DG e RM + DG + DDG. O modelo utilizado foi baseado no da equação 2.3, com a diferença de não possuir autoregulação, o que permite a utilização do modelo para um número ilimitado de genes. Esse novo modelo está representado no conjunto de equações 4.1. Os grafos de dependências para esse modelo encontram-se na figura 4.1.



A palavra *null* à direita das duas últimas reações do modelo 4.1 significa que os reagentes dessas reações serão degradados e que os seus produtos não são relevantes para o modelo.

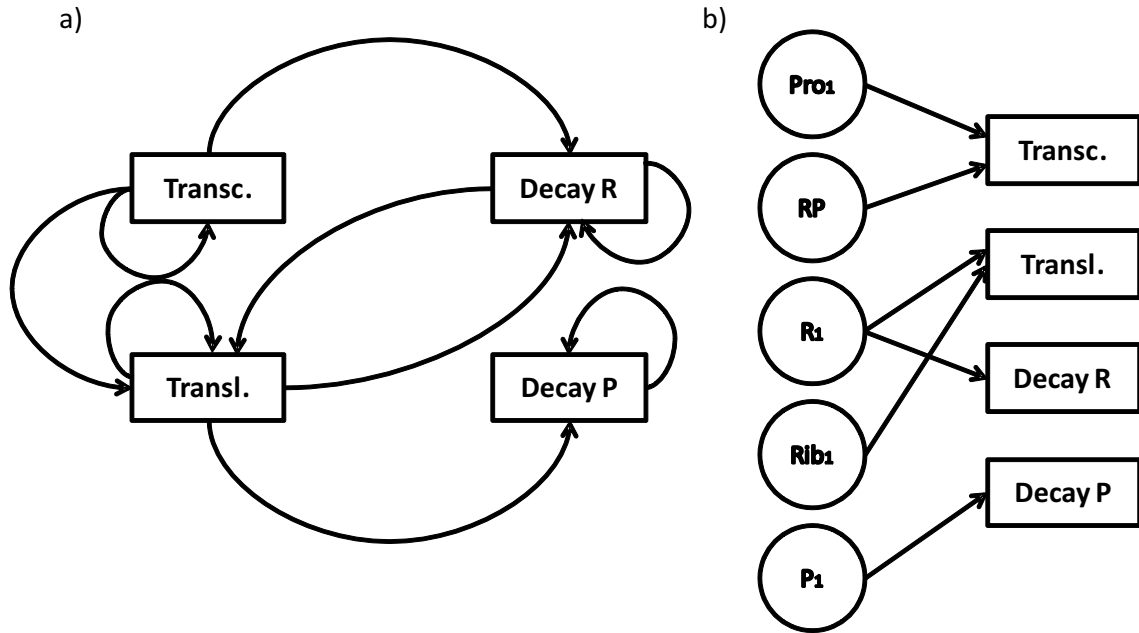


Figura 4.1: Grafos de Dependências para o Modelo 4.1. (a) DG (grafo de dependências) e (b) DDG (grafo de dependências para reações com atrasos).

Para o modelo 4.1, foram executadas 4 mil simulações para ambos algoritmos: RM + DG e RM + DG + DDG. Essas 4 mil simulações foram divididas em quatro grupos de mil simulações, sendo o primeiro grupo com a rede do modelo 4.1 com 4 genes ($i = 4$). Os outros três grupos de mil simulações foram executados com esse mesmo modelo para 10 genes, 16 genes e 22 genes, respectivamente. O número de atualizações por segundo do número de espécies de cada grupo de simulações foi obtido para cada algoritmo e, posteriormente, foi utilizado para calcular o ganho percentual do algoritmo RM + DG + DDG em relação ao algoritmo RM + DG. O resultado dessa comparação é apresentado na figura 4.2.

A utilização do DDG melhorou o desempenho do algoritmo RM em aproximadamente 40% no modelo com menos reações (4 genes) e em mais de 60% para o modelo com mais reações (22 genes). Para o modelo apresentado, quanto maior o número de reações, melhor foi o desempenho do RM + DG + DDG quando comparado ao desempenho do RM + DG.

Como o DDG pode ser utilizado por todos os algoritmos apresentados nesse trabalho, outros resultados da utilização do DDG em conjunto com outros algoritmos são apresentados nas próximas subseções.

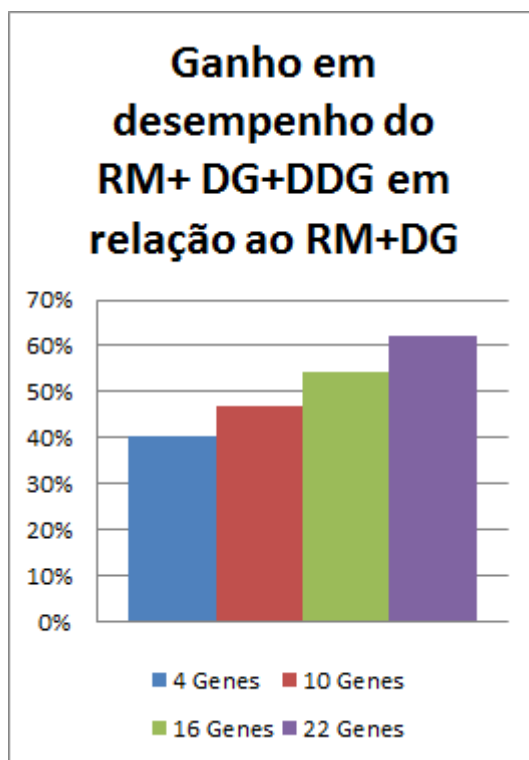


Figura 4.2: Ganho em desempenho do algoritmo RM + DG + DDG em relação ao algoritmo RM + DG. O gráfico apresenta o ganho percentual do número de atualizações de espécies por segundo do algoritmo que utiliza o grafo DDG em relação ao algoritmo que não utiliza esse grafo.

4.2 NRM SIMPLIFICADO PARA REAÇÕES COM ATRASO

Essa seção apresenta o resultado de diversas simulações de três modelos diferentes, sendo um modelo fracamente acoplado e dois modelos fortemente acoplados. Foram testados os algoritmos *Rejection Method* (algoritmo 2.8, página 63), *Modified Next Reaction Method for Delays* (algoritmo 2.9, página 64) e *Simplified Next Reaction Method for Delay* (algoritmo 3.2, página 73). Todos os algoritmos foram testados em suas versões que implementam somente o DG e nas versões que implementam o DG e o DDG.

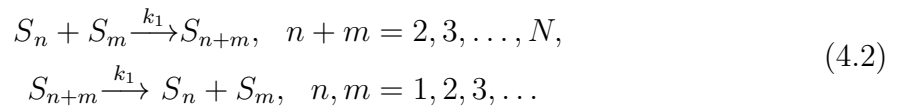
Para evitar que os resultados das simulações fossem afetados pelo posicionamento das reações, o algoritmo *Rejection Method* foi implementado utilizando-se a estratégia de ordenação das reações mais frequentes proposta por McCollum et al. (2006). Com essa estratégia, a reação executada troca de posição com a reação que possui um índice anterior ao índice da executada. Dessa forma, as reações mais executadas tendem a ficar

no início do vetor que contém as reações, o que pode aumentar a velocidade de execução do algoritmo.

Para os testes dos algoritmos, foram escolhidos modelos com um grande número de reações. O objetivo da utilização desses modelos é testar os algoritmos em redes mais complexas e que demandem uma grande capacidade de processamento. É esperado que estratégias mais simples como a do algoritmo RM sejam mais eficientes em redes biológicas com poucas reações.

4.2.1 UMA REDE FORTEMENTE ACOPLADA

O primeiro modelo usado para teste dos algoritmos chama-se *Colloidal Aggregation Model*. Apresentado por Ramaswamy e Sbalzarini (2011), esse modelo é fortemente acoplado e composto pelas seguintes reações:



Para esse modelo, consideramos que todas as espécies possuem o valor inicial 1 e a constante de velocidade k_1 também possui o valor 1. Todas as reações com índice par foram consideradas reações com atraso do tipo *consuming* com o tempo de retorno de 0.1 segundos. O grau de saída máximo para esse grafo é $3N - 7$, onde N é o número de espécies do modelo. A tabela 4.1 apresenta as 24 reações desse modelo para $N = 7$, sendo que o grafo de dependências DG para o modelo apresentado nessa tabela é exibido na figura 4.3. Já a tabela 4.2 apresenta as mesmas reações da tabela 4.1, somente acrescentando os atrasos. O grafo de dependências DDG para o modelo exibido na tabela 4.2 é mostrado na figura 4.4

$n + m$	Ida	Volta
2	1) $S_1 + S_1 \rightarrow S_2$	2) $S_2 \rightarrow S_1 + S_1$
3	3) $S_1 + S_2 \rightarrow S_3$	4) $S_3 \rightarrow S_1 + S_2$
4	5) $S_1 + S_3 \rightarrow S_4$	6) $S_4 \rightarrow S_1 + S_3$
	7) $S_2 + S_2 \rightarrow S_4$	8) $S_4 \rightarrow S_2 + S_2$
5	9) $S_1 + S_4 \rightarrow S_5$	10) $S_5 \rightarrow S_1 + S_4$
	11) $S_2 + S_3 \rightarrow S_5$	12) $S_5 \rightarrow S_2 + S_3$
6	13) $S_1 + S_5 \rightarrow S_6$	14) $S_6 \rightarrow S_1 + S_5$
	15) $S_2 + S_4 \rightarrow S_6$	16) $S_6 \rightarrow S_2 + S_4$
	17) $S_3 + S_3 \rightarrow S_6$	18) $S_6 \rightarrow S_3 + S_3$
7	19) $S_1 + S_6 \rightarrow S_7$	20) $S_7 \rightarrow S_1 + S_6$
	21) $S_2 + S_5 \rightarrow S_7$	22) $S_7 \rightarrow S_2 + S_5$
	23) $S_3 + S_4 \rightarrow S_7$	24) $S_7 \rightarrow S_3 + S_4$

Tabela 4.1: Reações para modelo da equação 4.2 (*Colloidal Aggregation*) com $N = 7$. Para esse caso, são geradas 24 reações considerando-se as duas reações do modelo (2 sentidos). Não considera atrasos no modelo.

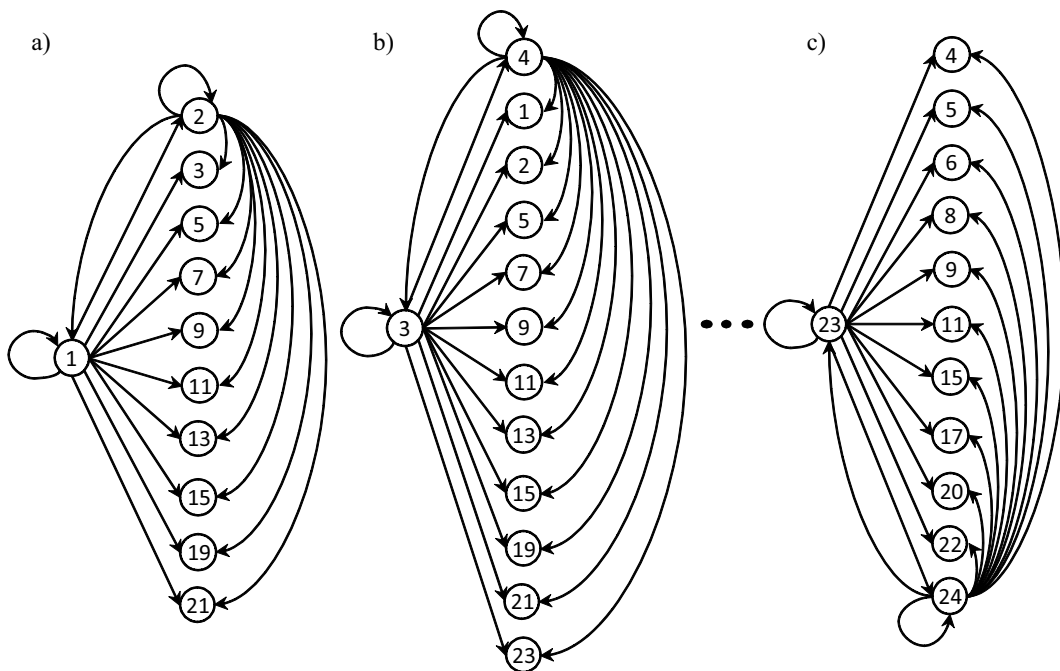


Figura 4.3: Partes do DG para modelo da equação 4.2 (*Colloidal Aggregation*) com $N = 7$. Os nós dos grafos foram repetidos para se obter uma melhor visualização das arestas. Foram construídos DG's para 6 reações: (a) reações 1, 2; (b) reações 3, 4 e (c) reações 23 e 24. Observa-se nestes grafos o alto acoplamento entre as reações. O grau de acoplamento (grau máximo de saída), $3N - 7$, para $N = 7$ é $3 \times 7 - 7 = 14$ para este modelo de reações. Observar que a reação 3 afeta 13 reações, pouco mais da metade do número total de reações. Esse grafo é usado, considerando modelos com reações com atrasos que utilizam o DDG, apenas quando uma reação é disparada.

$n + m$	Ida	Volta
2	1) $S_1 + S_1 \rightarrow S_2$	2) $S_2 \rightarrow S_1(d1) + S_1(d1)$
3	3) $S_1 + S_2 \rightarrow S_3$	4) $S_3 \rightarrow S_1(d1) + S_2(d1)$
4	5) $S_1 + S_3 \rightarrow S_4$	6) $S_4 \rightarrow S_1(d1) + S_3(d1)$
	7) $S_2 + S_2 \rightarrow S_4$	8) $S_4 \rightarrow S_2(d1) + S_2(d1)$
5	9) $S_1 + S_4 \rightarrow S_5$	10) $S_5 \rightarrow S_1(d1) + S_4(d1)$
	11) $S_2 + S_3 \rightarrow S_5$	12) $S_5 \rightarrow S_2(d1) + S_3(d1)$
6	13) $S_1 + S_5 \rightarrow S_6$	14) $S_6 \rightarrow S_1(d1) + S_5(d1)$
	15) $S_2 + S_4 \rightarrow S_6$	16) $S_6 \rightarrow S_2(d1) + S_4(d1)$
	17) $S_3 + S_3 \rightarrow S_6$	18) $S_6 \rightarrow S_3(d1) + S_3(d1)$
7	19) $S_1 + S_6 \rightarrow S_7$	20) $S_7 \rightarrow S_1(d1) + S_6(d1)$
	21) $S_2 + S_5 \rightarrow S_7$	22) $S_7 \rightarrow S_2(d1) + S_5(d1)$
	23) $S_3 + S_4 \rightarrow S_7$	24) $S_7 \rightarrow S_3(d1) + S_4(d1)$

Tabela 4.2: Reações para modelo da equação 4.2 (*Colloidal Aggregation*) com $N = 7$ e atraso $d1 = 0.1$ para as reações pares. São 24 reações considerando-se as duas reações do modelo (2 sentidos). Apenas as reações de “volta” possuem atrasos.

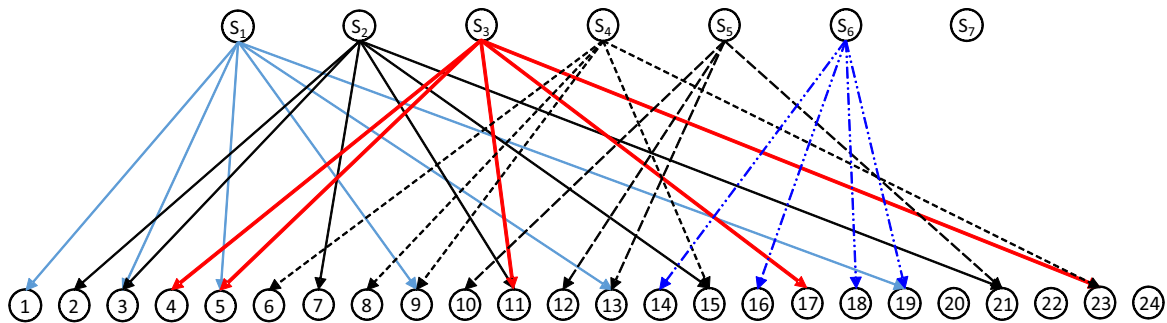


Figura 4.4: DDG para modelo da equação 4.2 (*Colloidal Aggregation*) com atraso para as reações pares e $N = 7$. Todas as reações estão na tabela 4.2. A espécie S_7 foi deixada apenas para ilustração já que ela não ocorre com atraso em nenhuma reação do modelo, sendo assim – como todos os reagentes do modelo – ela nunca será inserida na lista de espera. Como visto, por exemplo, as propensidades das reações 4, 5, 11, 17 e 23 são atualizadas quando o produto S_3 retorna da lista de espera em um momento qualquer. O grau de acoplamento (grau máximo de saída) é 6 para este modelo de reações ($N = 7$). Nenhum arco atinge as reações 20, 22 e 24, pois nenhum de seus reagentes é um produto com atraso em qualquer outra reação.

Utilizando o modelo 4.2, foi criada uma rede com 3.048 reações. A figura 4.5 apresenta o resultado da execução de cada um dos algoritmos para o modelo 4.2.

A eficiência do SNRM é afetada pelo grau de acoplamento do grafo de dependências. Para um modelo fortemente acoplado, como o Modelo 4.2, é esperado um fraco desempenho do algoritmo DSNRM em comparação com o RM. Essa expectativa foi parcialmente confirmada pelo DSNRM + DG ter apresentado um desempenho levemente inferior ao desempenho do RM + DG. Porém, mesmo para esse modelo fortemente acoplado, o algoritmo DSNRM + DG + DDG obteve um desempenho superior ao RM + DG + DDG. Isso indica que apesar da utilização do DDG favorecer todos os algoritmos apresentados, a importância desse grafo para o DSNRM é ainda maior, pois o cálculo das propensidades nesse algoritmo está diretamente relacionado à necessidade de alterações na fila de prioridades das reações, que é um processo que consome muitos recursos quando se utiliza o algoritmo *Simplified Next Reaction Method*.

Como ambos os grafos (DG e DDG) do modelo 4.2 são fortemente acoplados, o desempenho do DDG foi levemente superior ao do DG para o retorno da lista de produtos atrasados (ver figura 4.5). Para modelos em que o DG é um grafo fortemente acoplado e o DDG é fracamente acoplado, o desempenho dos algoritmos que utilizam DG + DDG tende a ser muito superior ao desempenho dos algoritmos que utilizam somente o DG.

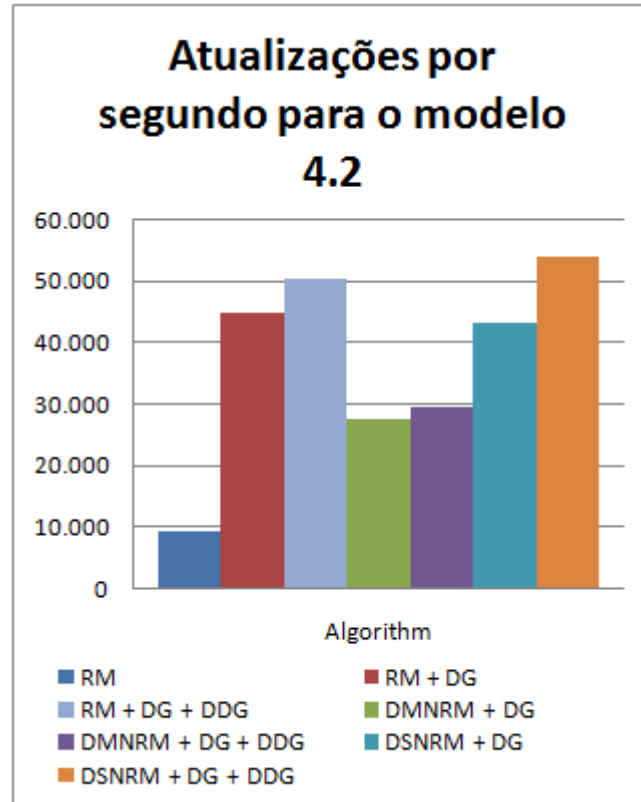
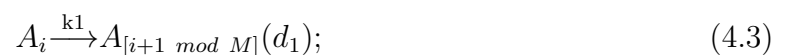


Figura 4.5: Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.2 (*Colloidal Aggregation Model*). Os algoritmos mencionados foram testados em suas versões que utilizam o grafo DG e nas que utilizam o grafo DG em conjunto com o grafo DDG.

4.2.2 UMA REDE FRACAMENTE ACOPLADA

O segundo modelo testado apresenta um conjunto com M reações onde toda reação i ($i < M$) gera o substrato para a reação $i + 1$ enquanto a reação $i = M$ gera o substrato para a reação 1. Esse modelo é fracamente acoplado, pois o grafo de dependências das reações possui somente duas arestas saindo de cada reação, sendo que uma dessas arestas representa a própria reação.

Com i no intervalo $[1 \dots 1.000]$, obtivemos 1.000 reações. O modelo encontra-se na equação 4.3.



Para o modelo 4.3, considere: $A_i=10$, $k_1=1$, $d_1=1$. O DG e o DDG para esse modelo encontram-se na figura 4.6. A figura 4.7 apresenta o resultado da execução de cada

algoritmo para esse modelo.

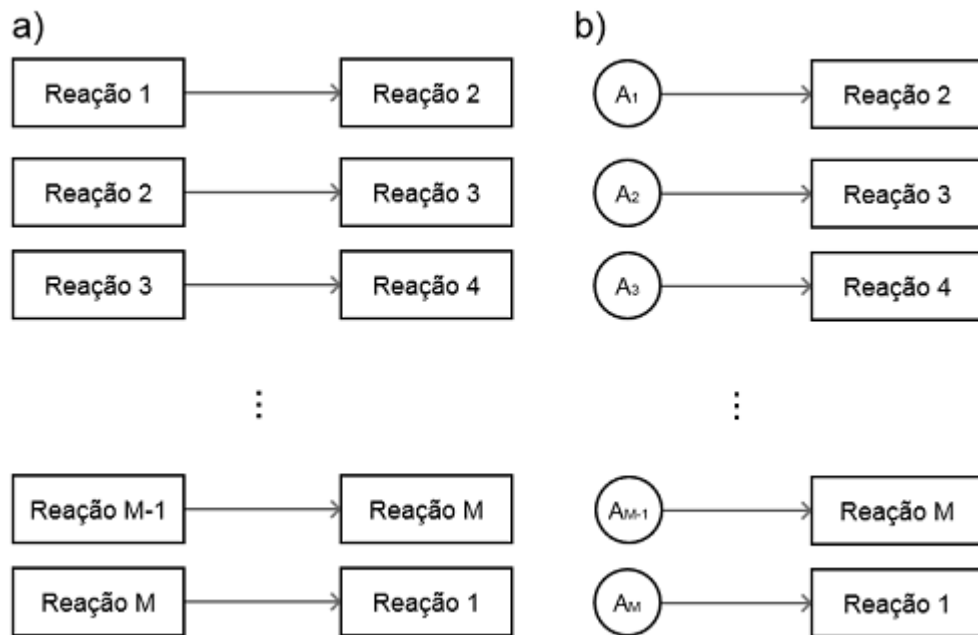


Figura 4.6: Grafos de dependências para o modelo 4.3. (a) apresenta o DG e (b) apresenta o DDG.

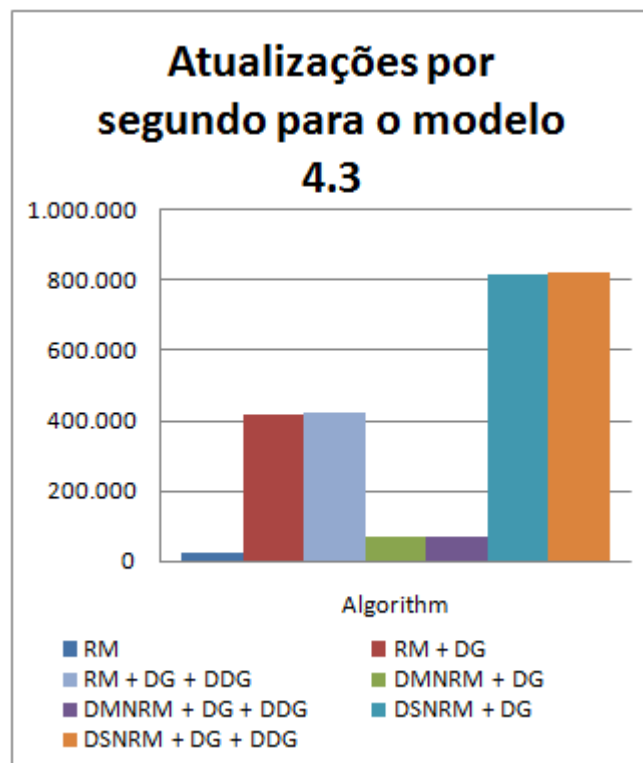


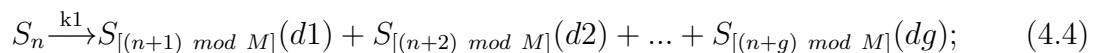
Figura 4.7: Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.3 (rede fracamente acoplada). Os algoritmos foram testados em suas versões que utilizam o grafo DG e nas que utilizam o grafo DG em conjunto com o grafo DDG.

Para um modelo fracamente acoplado, como o Modelo 4.3, é esperado um bom desempenho do algoritmo SNRM em comparação com o RM. Essa expectativa foi confirmada pelo DSNRM + DG ter realizado quase o dobro do número de atualizações por segundo que o RM + DG.

O modelo 4.3 representa o pior caso para o DDG, onde ele torna-se equivalente ao DG, o que explica a proximidade do desempenho dos algoritmos que utilizaram o DG e dos que utilizaram DG + DDG. É importante destacar que, no pior caso, o DDG é igual ao DG. Devido a isso, recomenda-se fortemente a utilização do DDG em todas as simulações com atrasos.

4.2.3 MODELO FORTEMENTE ACOPLADO EM REDES COM MÚLTIPLOS ATRASOS

O terceiro modelo testado é um modelo circular. A principal característica desse modelo é que o número de produtos com atraso cresce na mesma razão que o acoplamento. Nesse modelo, quando um determinado produto em espera é retornado, existe somente uma reação que possui a espécie retornada como substrato. Isso ocorre devido à utilização de múltiplos atrasos com valores diferentes e à característica do modelo de possuir somente um substrato e mais de um produto. Dessa forma, mesmo as reações estando fortemente acopladas, o retorno de um produto será um processo simples e rápido. Para os testes, geramos 1000 reações ($M = 1000$, $n = (1..1000)$). Para trabalhar com diferentes graus de acoplamento, foi utilizado $g = [1..2]$ para a simulação 1, $g = [1..6]$ para a simulação 2 e $g = [1..10]$ para a simulação 3. Tal modelo de reações é exibido na equação 4.4.



Para esse modelo, consideramos que todas as espécies possuem o valor inicial 40 e a constante de velocidade k_1 possui o valor 1. Todas as reações foram consideradas *consuming delayed reactions* com atraso de $g/10 + 4$ segundos.

As figuras 4.8, 4.9 e 4.10 apresentam o resultado da execução de cada algoritmo para o modelo 4.4 com g igual a $[1..2]$, $[1..6]$ e $[1..10]$, respectivamente.

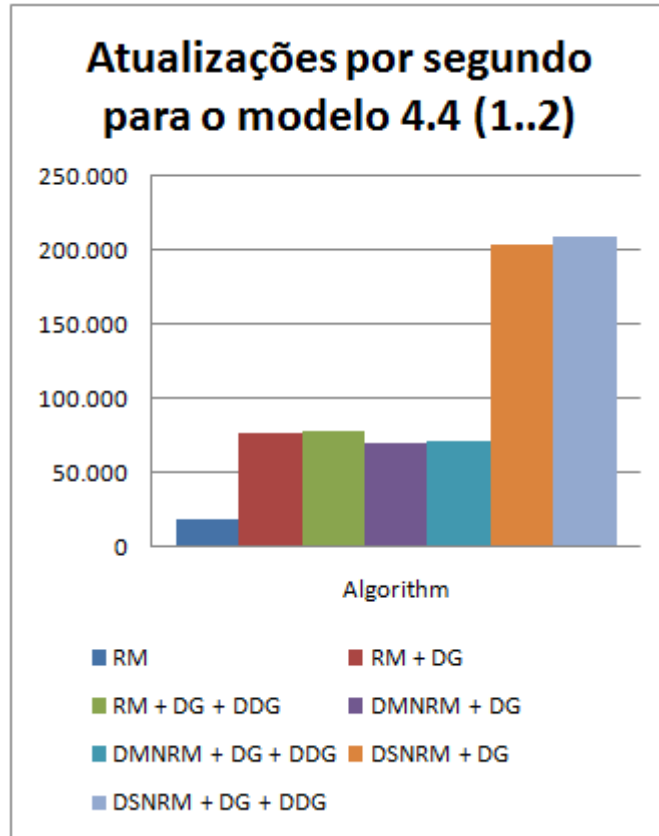


Figura 4.8: Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.4 (rede fortemente acoplada com múltiplos atrasos) com acoplamento de grau 2. Os algoritmos foram testados em suas versões que utilizam o grafo DG e nas que utilizam o grafo DG em conjunto com o grafo DDG.

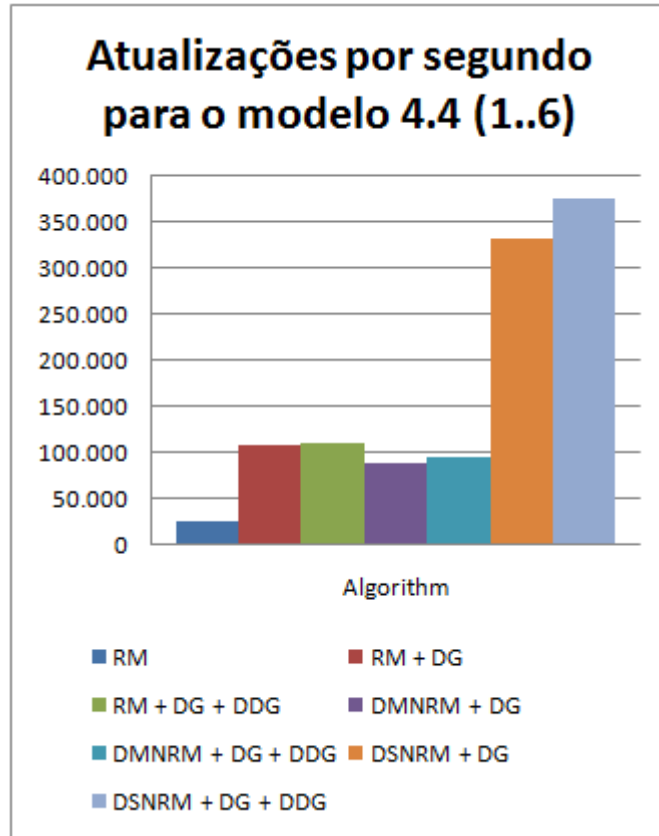


Figura 4.9: Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.4 (rede fortemente acoplada com múltiplos atrasos) com acoplamento de grau 6. Os algoritmos foram testados em suas versões que utilizam o grafo DG e nas que utilizam o grafo DG em conjunto com o grafo DDG.

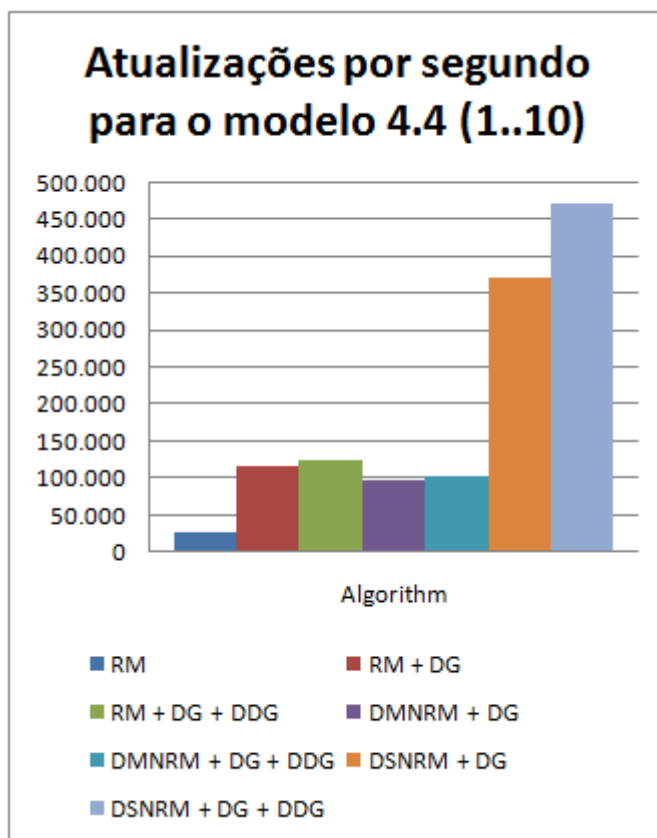


Figura 4.10: Comparação do desempenho dos algoritmos RM, DMNRM e DSNRM para o modelo 4.4 (rede fortemente acoplada com múltiplos atrasos) com acoplamento de grau 10. Os algoritmos foram testados em suas versões que utilizam o grafo DG e nas que utilizam o grafo DG em conjunto com o grafo DDG.

O modelo cujos resultados foram apresentados no gráfico da figura 4.10 é mais acoplado que o modelo utilizado no gráfico da figura 4.8. É possível observar que o aumento do acoplamento influenciou positivamente o desempenho do algoritmo DSNRM. Avaliando melhor o modelo e os algoritmos, é possível concluir que o aumento do grau de acoplamento para reações com múltiplos atrasos é prejudicial ao DSNRM se cada produto em espera for substrato de várias reações. Um forte acoplamento entre reações não indica se existirá um forte acoplamento na direção produtos-substratos. Logo, é possível concluir que nem todos os modelos fortemente acoplados são prejudiciais ao DSNRM, mas somente modelos fortemente acoplados com o grafo DDG denso.

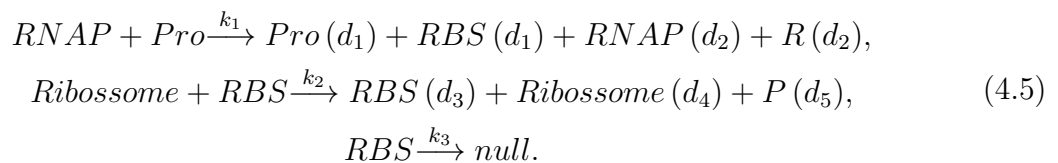
4.3 LISTA ORDENADA POR *HASHING*

Para os testes de desempenho da lista ordenada por *hashing*, o algoritmo RM foi utilizado. Duas implementações desse algoritmo foram testadas, sendo que a primeira utiliza uma *heap* para controle da lista de produtos em espera e a segunda utiliza a lista ordenada por *hashing* para essa mesma função.

Dois modelos foram usados para testar o desempenho computacional entre essas duas abordagens. O primeiro modelo, proposto por Zhu et al. (2007), é composto por três reações (conjunto de equações 4.5). Esse modelo, que possui múltiplos atrasos, descreve a transcrição e a tradução para um gene de um organismo procarioto (uma bactéria). As espécies possuem os seguintes significados:

- RNAP: RNA polimerase;
- Pro: promotor do gene;
- RBS: sítio de ligação do ribossomo (*ribosome binding site*);
- R: mRNA (transcrito);
- Ribosome: ribossomo;
- P: proteína traduzida.

A palavra *null* da última reação do modelo 4.5 significa que o reagente será degradado por enzimas e que o seu produto não é relevante para o modelo. É possível observar que nesse modelo não há nenhuma regulação da expressão do gene.



O segundo modelo testado é o modelo 4.2, apresentado na página 84. Esse modelo foi utilizado para gerar uma rede com 1.012 reações.

Os resultados obtidos nos experimentos encontram-se nas figuras 4.11 e 4.12.

O desempenho para o modelo de inspiração biológica 4.5 foi bastante positivo para o algoritmo RM + *Hashing*. Já o resultado para o modelo 4.2 foi levemente positivo para o mesmo algoritmo. Isso é esperado devido ao modelo 4.2 não possuir valores diferentes para atrasos. Essa característica, somada ao curto intervalo de tempo entre cada uma das reações do modelo, contribuem para que vários produtos em espera sejam inseridos em

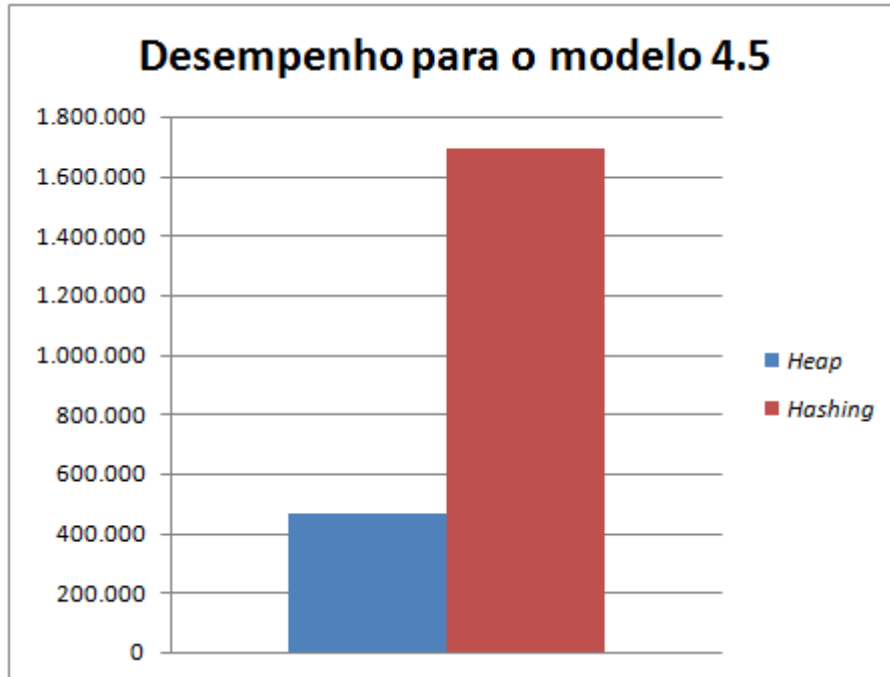


Figura 4.11: Comparação do número de atualizações por segundo para o modelo 4.5 (proposto por Zhu et al. (2007)) entre o algoritmo RM + *Heap* e o algoritmo RM + *Hashing*.

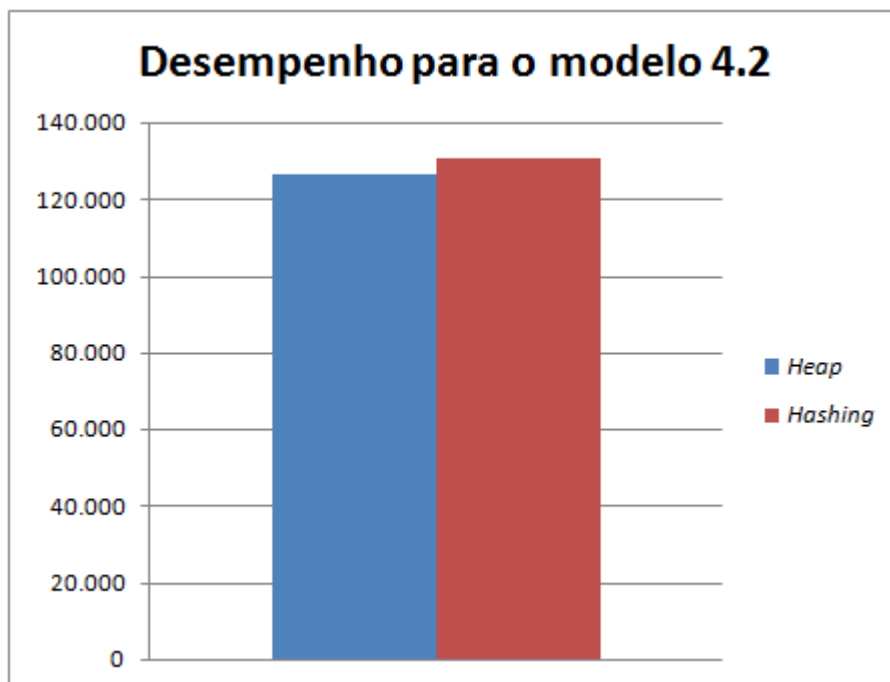


Figura 4.12: Comparação do número de atualizações por segundo para o modelo 4.2 (*Colloidal Aggregation Model*) entre o algoritmo RM + *Heap* e o algoritmo RM + *Hashing*.

um mesmo índice da lista ordenada por *hashing*, provocando várias colisões. Isso pode ser resolvido aumentando-se o tamanho da lista de produtos em espera.

5 CONCLUSÕES

Para modelos com poucas reações, o algoritmo RM tende a apresentar um melhor desempenho que os algoritmos MNRM e SNRM. Já o algoritmo MNRM tende a ter um desempenho melhor em modelos totalmente acoplados. O algoritmo SNRM tende a apresentar melhores resultados em modelos com muitas reações e fracamente acoplados, mas esse algoritmo também pode apresentar desempenho superior ao do RM e ao MNRM em alguns modelos fortemente acoplados, conforme mostrado nos resultados.

Em alguns testes, o grafo DDG aumentou o desempenho das simulações em mais de 60%, como pode ser visto na seção 4.1. Como no pior caso o DDG é exatamente igual ao DG, consideramos que a utilização do DDG em simulações estocásticas de redes de regulação gênica com atrasos é altamente recomendável.

Ao utilizar o DDG, uma nova definição para o grafo DG pode ser feita, sendo que nessa nova definição, todas as espécies que possuem atraso seriam desconsideradas pelo DG. Uma possibilidade de trabalho futuro seria fazer essa alteração no DG e testá-la com diferentes modelos.

Como a implementação do NRM exige diversos controles extras para tratar as exceções à fórmula de reaproveitamento de números aleatórios, é comum algumas implementações de NRM gerarem um novo número aleatório ao invés de tratar essas exceções, o que diminui o desempenho do NRM. Como o SNRM é um algoritmo que não possui exceções a serem tratadas, a sua implementação é mais simples e direta que a do NRM, o que pode trazer maior confiabilidade para o programador e um desempenho igual ou superior ao do NRM.

Para os modelos testados, a lista ordenada por *hashing* mostrou-se uma estrutura de dados mais eficiente que a *heap* para o controle da lista de produtos em espera. Porém a definição do tamanho dessa lista é fundamental para uma melhor distribuição dos produtos em espera, o que contribui diretamente para o desempenho da lista. O mal desempenho da lista ordenada por *hashing* exibido na figura 4.12 é devido à essa lista possuir um tamanho pequeno para tal modelo, causando assim um fraco desempenho por causa da quantidade de colisões. Como a definição matemática do melhor tamanho da lista pode ser difícil de ser obtida, uma sugestão de trabalho futuro pode ser a criação de uma heurística para a definição do tamanho dessa lista.

REFERÊNCIAS

ALBERTS, B.; JOHNSON, A.; LEWIS, J.; RAFF, M.; ROBERTS, K.; WALTER, P. **Biologia Molecular da Célula**. 5. ed. Porto Alegre: Artmed, 2010. 1728 p.

ANDERSON, D. F. A Modified Next Reaction Method for Simulating Chemical Systems with Time Dependent Propensities and Delays. **The Journal of Chemical Physics**, AIP Publishing, v. 127, n. 21, 2007.

ARKIN, A.; ROSS, J.; MCADAMS, H. H. Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage λ -infected Escherichia Coli Cells. **Genetics**, Genetics Society of America, v. 149, n. 4, p. 1633–1648, 1998.

BARABASI, A. L.; OLTVAI, Z. N. Network Biology: Understanding the Cell's Functional Organization. **Nature Reviews Genetics**, Nature Publishing Group, v. 5, n. 2, p. 101–113, 2004.

BRASS, P. **Advanced Data Structures**. Nova Iorque: Cambridge University Press, 2008. 474 p.

BRATSUN, D.; VOLFSO, D.; TSIMRING, L. S.; HASTY, J. Delay-induced Stochastic Oscillations in Gene Regulation. **Proceedings of the National Academy of Sciences of the United States of America**, National Academy of Sciences of the United States of America, v. 102, n. 41, p. 14593–14598, 2005.

CARAVAGNA, G. **Formal Modeling and Simulation of Biological Systems with Delays**. Tese (Doutorado) — Departamento de Informática, Universidade de Pisa, Pisa, 2011.

CELES, W.; CERQUEIRA, R.; RANGEL, J. L. **Introdução a Estruturas de Dados: com técnicas de programação em C**. Rio de Janeiro: Campus, 2004. 308 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms**. 2. ed. Cambridge: MIT Press, 2001. 1180 p.

- CORRERA, T. C.; FRANK, A. de C. A Equação-Mestra: Atingindo o Equilíbrio. **Química Nova**, SciELO Brasil, v. 34, n. 2, p. 346–353, 2011.
- ELOWITZ, M. B.; LEVINE, A. J.; SIGGIA, E. D.; SWAIN, P. S. Stochastic Gene Expression in a Single Cell. **Science**, American Association for the Advancement of Science, v. 297, n. 5584, p. 1183–1186, 2002.
- GARDNER, T. S.; CANTOR, C. R.; COLLINS, J. J. Construction of a Genetic Toggle Switch in Escherichia Coli. **Nature**, Nature Publishing Group, v. 403, n. 6767, p. 339–342, 2000.
- GIBSON, M. A.; BRUCK, J. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. **The Journal of Physical Chemistry**, ACS Publications, v. 104, n. 9, p. 1876–1889, 2000.
- GILLESPIE, D. T. A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. **Journal of Computational Physics**, Elsevier, v. 22, n. 4, p. 403–434, 1976.
- GILLESPIE, D. T. Exact Stochastic Simulation of Coupled Chemical Reactions. **The Journal of Physical Chemistry**, ACS Publications, v. 81, n. 25, p. 2340–2361, 1977.
- GILLESPIE, D. T. **Markov Processes: an introduction for physical scientists**. San Diego: Elsevier, 1991. 581 p.
- GILLESPIE, D. T. Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems. **The Journal of Chemical Physics**, AIP Publishing, v. 115, n. 4, p. 1716–1733, 2001.
- HOLLEY, R. W.; APGAR, J.; EVERETT, G. A.; MADISON, J. T.; MARQUISEE, M.; MERRILL, S. H.; PENSWICK, J. R.; ZAMIR, A. Structure of a Ribonucleic Acid. **Science**, American Association for the Advancement of Science, v. 147, n. 3664, p. 1462–1465, 1965.
- HOVATTA, I.; KIMPPA, K.; LAINE, M. M.; LEHMUSSOLA, A.; PASANEN, T.; SAARELA, J.; SAARIKKO, I.; SAHARINEN, J.; TIIKKAINEN, P.; TOIVANEN, T.; TOLVANEN, M.; TUIMALA, J.; VIHINEN, M.; WONG, G. **DNA Microarrays Data Analysis**. 2. ed. Helsinki: CSC - Scientific Computing, 2005. 163 p.

- KAMPEN, N. G. V. **Stochastic Processes in Physics and Chemistry**. 3. ed. Amsterdam: North-Holland Personal Library, 2007. 474 p.
- KARLEBACH, G.; SHAMIR, R. Modelling and Analysis of Gene Regulatory Networks. **Nature Reviews Molecular Cell Biology**, v. 9, n. 10, p. 770–780, 2008.
- KARP, G. **Cell and Molecular Biology: concepts and experiments**. Nova Iorque: John Wiley & Sons, 2010. 837 p.
- KLIPP, E.; HERWIG, R.; KOWALD, A.; WIERLING, C.; LEHRACH, H. **Systems Biology in Practice: concepts, implementation and application**. Nova Iorque: John Wiley & Sons, 2005. 486 p.
- MCADAMS, H. H.; ARKIN, A. Stochastic Mechanisms in Gene Expression. **Proceedings of the National Academy of Sciences of the United States of America**, National Academy of Sciences of the United States of America, v. 94, n. 3, p. 814–819, 1997.
- MCCOLLUM, J. M.; PETERSON, G. D.; COX, C. D.; SIMPSON, M. L.; SAMATOVA, N. F. The Sorting Direct Method for Stochastic Simulation of Biochemical Systems with Varying Reaction Execution Behavior. **Computational Biology and Chemistry**, Elsevier, v. 30, n. 1, p. 39–49, 2006.
- MESELSON, M.; STAHL, F. W. The replication of DNA in Escherichia Coli. **Proceedings of the National Academy of Sciences of the United States of America**, National Academy of Sciences of the United States of America, v. 44, n. 7, p. 671–682, 1958.
- METROPOLIS, N.; ULAM, S. The Monte Carlo Method. **Journal of the American Statistical Association**, Taylor & Francis Group, v. 44, n. 247, p. 335–341, 1949.
- NELSON, D. L.; LEHNINGER, A. L.; COX, M. M. **Lehninger Principles of Biochemistry**. 5. ed. Nova Iorque: Macmillan, 2008. 1294 p.
- OLIVEIRA, I. L. **Análise Computacional de Redes Metabólicas com Regulação**. Tese (Doutorado) — Departamento de Engenharia Química e Alimentos, Universidade Federal de Santa Catarina, Florianópolis, 2008.

- PAHLE, J. Biochemical Simulations: stochastic, approximate stochastic and hybrid approaches. **Briefings in Bioinformatics**, Oxford University Press, v. 10, n. 1, p. 53–64, 2009.
- PALSSON, B. O. **Systems Biology: properties of reconstructed networks**. San Diego: Cambridge University Press, 2006. 336 p.
- RAMASWAMY, R.; GONZÁLEZ-SEGREDO, N.; SBALZARINI, I. F. A New Class of Highly Efficient Exact Stochastic Simulation Algorithms for Chemical Reaction Networks. **The Journal of Chemical Physics**, AIP Publishing, v. 130, n. 24, p. 244104, 2009.
- RAMASWAMY, R.; SBALZARINI, I. F. A Partial-propensity Variant of the Composition-rejection Stochastic Simulation Algorithm for Chemical Reaction Networks. **The Journal of Chemical Physics**, AIP Publishing, v. 132, n. 4, p. 44102, 2010.
- RAMASWAMY, R.; SBALZARINI, I. F. A Partial-propensity Formulation of the Stochastic Simulation Algorithm for Chemical Reaction Networks with Delays. **The Journal of Chemical Physics**, AIP Publishing, v. 134, n. 1, p. 14106, 2011.
- RIBEIRO, A. S. Stochastic and Delayed Stochastic Models of Gene Expression and Regulation. **Mathematical Biosciences**, Elsevier, v. 223, n. 1, p. 1–11, 2010.
- ROBERTIS, E. D.; HIB, J. **De Robertis: Bases da Biologia Celular e Molecular**. Rio de Janeiro: Guanabara Koogan, 2001. 418 p.
- ROUSSEL, M. R.; ZHU, R. Validation of an Algorithm for Delay Stochastic Simulation of Transcription and Translation in Prokaryotic Gene Expression. **Physical Biology**, IOP Publishing, v. 3, n. 4, p. 274, 2006.
- SOBOL, I. M. **A Primer for the Monte Carlo Method**. Boca Raton: CRC Press, 1994. 246 p.
- SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de Dados e seus Algoritmos**. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos, 1994. 334 p.
- TAYLOR, H. M.; KARLIN, S. **An Introduction to Stochastic Modeling**. 3. ed. San Diego: Academic Press, 1998. 646 p.

WATSON, J.; CRICK, F. A Structure for Deoxyribose Nucleic Acid. **Nature**, v. 171, n. 4356, p. 737–738, 1953.

ZHU, R.; RIBEIRO, A. S.; SALAHUB, D.; KAUFFMAN, S. A. Studying Genetic Regulatory Networks at the Molecular Level: Delayed Reaction Stochastic Models. **Journal of Theoretical Biology**, Elsevier, v. 246, n. 4, p. 725–745, 2007.