Análise do Desempenho Computacional dos Algoritmos de Simulação Estocástica para Redes de Regulação Gênica

Rafael de Souza Terra

JUIZ DE FORA

NOVEMBRO, 2019

Análise do Desempenho Computacional dos Algoritmos de Simulação Estocástica para Redes de Regulação Gênica

Rafael de Souza Terra

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Itamar Leite de Oliveira

JUIZ DE FORA

NOVEMBRO, 2019

Análise do Desempenho Computacional dos Algoritmos de Simulação Estocástica para Redes de Regulação Gênica

Rafael de Souza Terra

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Itamar Leite de Oliveira
Professor Doutor

Heder Soares Bernadino
Professor Doutor

Raul Fonseca Neto
Professor Doutor

JUIZ DE FORA

29 DE NOVEMBRO, 2019

Aos meus pais, pelo apoio e sustento.
Aos meus amigos, principalmente o João Gabriel Malaguti.
Ao professor Itamar Leite por todos esses anos de orientação e aos outros professores por todo o conhecimento a mim passado.

Resumo

Com o avanço da tecnologia e o crescente interesse no estudo de redes de regulação gênica, diversos algoritmos utilizando diferentes abordagens para a simulação estocástica dessas redes foram desenvolvidos. Decorre então o questionamento de qual algoritmo presente na literatura utilizar e por que utilizá-lo na simulação de uma determinada rede de regulação gênica. A escolha do algoritmo de simulação estocástica pode ser decisiva para conseguir ou não realizar a simulação de uma rede gênica. Para auxiliar nessa escolha o presente trabalho busca realizar a implementação dos principais algoritmos de simulação estocástica presentes na literatura utilizando a linguagem de programação *C++11* e o estudo deles, verificando o desempenho de cada algoritmo em relação a redes de diferentes graus de acoplamento. Esse trabalho ainda propõe a implementação de uma estrutura de dados utilizando o conceito de lista circular no algoritmo de simulação estocástica com *delay* verificando o seu desempenho em relação ao algoritmo com outras estruturas como a *heap* e a lista ordenada.

Os algoritmos de simulação estocástica apresentados nesse trabalho obtiveram um resultado semelhante entre si com o *Direct Method* se destacando na maioria dos testes. A lista circular apresentou uma melhora de desempenho em relação à lista ordenada, porém a *heap* teve os melhores resultados.

Palavras-chave: Simulação estocástica, redes de regulação gênica, lista circular.

Abstract

With the improvement of the technology and the increasing interest in the study of gene-regulatory networks, several algorithms using different approaches to the stochastic simulation of these networks have been developed. It follows the questioning of which algorithm in the literature we should use and why we should use it in the simulation of a certain gene-regulatory network. Choosing the best algorithm is a very important factor to be able to simulate or not a gene-regulatory network. To help in this choice the present work aims to realize the implementation of the main stochastic simulation algorithms in the literature using the programming language *C++11* and their study, checking the performance of each algorithm regarding networks of distinct degrees of coupling. This work also presents the implementation of a data structure using the circular list's concept in the delayed stochastic simulation algorithm verifying it's performance concerning the one with other structures like the heap and the sorted list.

The stochastic simulation algorithms presented in this work got similar results with the Direct Method highlighting itself in the majority of the tests. The circular list showed an improvement of the performance when compared with the ordered list, but the heap got the best results.

Keywords: Stochastic simulation, gene-regulatory networks, circular list.

Agradecimentos

Aos meus pais pelo apoio em todos os momentos da minha vida e por me ajudar a perseguir uma carreira acadêmica, mesmo enfrentando vários desafios pela vida.

Ao professor Itamar por todos esses anos de orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos meus amigos, especialmente o João Gabriel Malaguti por me ajudar nos momentos de necessidade.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o meu enriquecimento pessoal e profissional.

"You never fail until you stop trying".

Albert Einstein

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DDG	*Delayed Dependency Graph*
DDR3	*Double Data Rate Type 3*
DG	*Dependency Graph*
DM	*Direct Method*
DNA	*Deoxyribonucleic Acid*
EDO	Equações diferenciais ordinárias
FRM	*First Reaction Method*
MNRM	*Modified Next Reaction Method*
NRM	*Next Reaction Method*
ODM	*Optimized Direct Method*
RAM	*Random Acess Memory*
RM	*Rejection Method*
RNA	*Ribonucleic Acid*
SDM	*Sorting Direct Method*
SNRM	*Simplified Next Reaction Method*
SSA	*Stochastic simulation algorithm*
TAD	Tipo abstrato de dados

1 Introdução

1.1 Apresentação do tema

Redes bioquímicas como as redes de regulação gênica são caracterizadas com um comportamento temporal complexo que, na maioria das vezes, é difícil de ser compreendido (Pahle, 2009). Existem duas principais maneiras de simular uma rede bioquímica: de maneira determinística e de maneira estocástica. A simulação determinística é realizada utilizando equações diferenciais ordinárias (EDO) para calcular os níveis de concentração dos compostos que participam da rede ao decorrer do tempo; já a simulação estocástica é realizada calculando a variação do número de moléculas da rede ao longo do tempo.

O algoritmo de simulação estocástica utilizado para realizar a simulação de redes bioquímicas foi desenvolvido graças aos trabalhos de Gillespie (1976, 1977). A cada passo esse algoritmo calcula a reação μ que ocorrerá e o intervalo de tempo τ em que ela ocorre, sendo ineficiente em termos de tempo de execução. Outras abordagens que visam resolver essa deficiência foram desenvolvidas, entre as mais relevantes estão o *Optimized Direct Method* (Cao et al., 2004), *Next Reaction Method* (Gibson e Bruck, 2000) e *Modified Next Reaction Method* (Anderson, 2007).

1.2 Problema

Os algoritmos de simulação estocástica são utilizados diariamente para a simulação de redes bioquímicas importantes e que podem possuir um grande volume de dados. Com a existência de várias implementações diferentes para os algoritmos de simulação estocástica alguns questionamentos aparecem, entre eles: a eficiência da implementação (ou seja, qual possui o menor tempo de execução para vários tipos de redes diferentes), a escolha do algoritmo para cada problema e a complexidade associada aos métodos (às vezes é preferível um algoritmo mais básico para a solução do problema).

1.3 Hipótese

O algoritmo de simulação estocástica com *delay* (Bratsun et al., 2005) (também conhecido como *Rejection Method*) utiliza em sua implementação uma lista ordenada para o armazenamento dos produtos com atraso. Se essa implementação for realizada utilizando uma estrutura de dados com complexidade assintótica menor do que a lista ordenada, o algoritmo terá um menor tempo de execução em relação à execução utilizando a lista ordenada.

1.4 Justificativa

No início do desenvolvimento dos algoritmos de simulação estocástica o algoritmo desenvolvido por Gillespie (1976) não era capaz de realizar a simulação de redes bioquímicas que possuíssem um vasto número de espécies e reações, isso fez com que Gillespie e outros pesquisadores buscassem soluções para esse problema. Escolher o algoritmo mais eficiente nem sempre é a melhor opção, visto que geralmente quanto mais eficiente o algoritmo, mais complexa é a sua implementação. Dependendo da rede bioquímica a ser simulada, pode ser melhor ao pesquisador utilizar um algoritmo com menor eficiência em relação aos demais, mas que possua uma implementação mais simples.

1.5 Objetivos

1.5.1 Objetivos gerais

Para responder aos questionamentos propostos anteriormente o presente trabalho tem dois objetivos: implementar e analisar o desempenho dos principais algoritmos de simulação estocástica publicados na literatura e também provar a hipótese apresentada implementando uma estrutura de dados baseada na lista circular para o armazenamento de produtos com atraso no algoritmo *Rejection Method*

1.5.2 Objetivos específicos

Implementar, na mesma linguagem, os algoritmos citados abaixo. Além disso implementar outros componentes necessários para a simulação, como: uma estrutura para armazena a mudança na quantidade das espécies durante o decorrer da simulação e também a simulação em lote. Para analisar o desempenho dos algoritmos, seus tempos de execução serão comparados frente a um conjunto de testes com redes com baixo e alto nível acoplamento.

- *Direct Method* (Gillespie, 1977);
- *Modified Next Reaction Method*;
- *Next Reaction Method*;
- *Optimized Direct Method*;
- *Rejection Method*;
- *Sorting Direct Method* (McCollum et al., 2006);
- *First Reaction Method* (Gillespie, 1976);
- *Simplified Next Reaction Method* (Silva, 2014).

Para provar a hipótese será desenvolvida uma lista circular que busca reduzir o custo computacional das operações de inserção e remoção de produtos com *delay*. Com o intuito de verificar a eficiência dessa lista será implementada também uma *heap*, por ser uma estrutura conhecida pelo seu bom desempenho. Com isso os testes serão realizados comparando o tempo de execução do *Rejection Method* utilizando as três estruturas (lista ordenada, lista circular e *heap*).

2 Fundamentação teórica

2.1 Introdução

Neste capítulo são apresentados os conceitos necessários para o entendimento do trabalho. A seção 2.2 aborda os conceitos biológicos envolvidos na simulação estocástica de redes de regulação gênica, sendo eles: genes, regulação da expressão gênica e redes de regulação gênica. Na seção 2.3 estão os conceitos computacionais envolvidos no desenvolvimento dos algoritmos de simulação.

2.2 Conceitos biológicos

2.2.1 Gene

As informações hereditárias de todos os seres vivos do planeta ficam armazenadas em moléculas de DNA, nas quais estão no formato de dupla hélice. Cujas fitas são compostas por longas cadeias de polímeros pareadas e não ramificadas formadas sempre pelos mesmos quatro tipos de nucleotídeos - adenina, timina, guanina e citosina - que são ligados entre si por meio de ligações fosfodiéster (Figura 2.1). Os genes correspondem a cerca de 10% do DNA e podem ser definidos como a sequência de DNA que contém a informação necessária para realizar a síntese de uma molécula de RNA, que caso seja do tipo mensageira, é capaz de sintetizar uma proteína (Alberts et al., 2010; De Robertis; Hib, 2006). Os nucleotídeos associados com o RNA são: adenina, citosina, uracila e guanina. Para ambas as moléculas a guanina e a citosina são nucleotídeos complementares; para o DNA a adenina é complementar com a timina e para o RNA a adenina é complementar com a uracila.

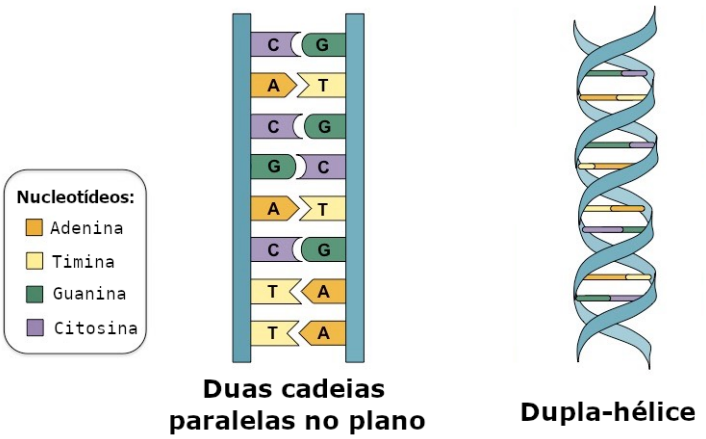

Figura 2.1: Esquema do DNA. Adaptado de Cornell (2016)

2.2.2 Regulação da expressão gênica

Marcus (2018) define a expressão gênica como "(...) o processo pelo qual a sequência de DNA é transcrita em um produto genético como uma proteína ou RNA". Apesar da maioria das células terem o mesmo DNA e possuírem funções básicas como a obtenção de energia, cada célula possui genes que são expressos na mesma mas que podem não ser expressos por outras células. A expressão desses genes pode ocorrer devido a diversos fatores, como mudanças à presença de elementos sinalizadores no ambiente celular e a regulação transcricional (Fowler et al., 2013; Linden, 2012).

Redes de regulação gênica são redes que além de serem responsáveis por controlar os processos metabólicos da célula, controlam várias funções da mesma como estado, forma e função. Essas redes são inferidas a partir de dados de uma expressão gênica e possuem um vasto grau de interação entre vários componentes heterogêneos onde, muitas vezes, apresentam um comportamento temporal complexo (Emmert-Streib et al., 2014; Pahle, 2009; Ramsey et al., 2005).

Um exemplo de rede de regulação gênica é a rede obtida da síntese de proteína a partir do DNA. A síntese de moléculas de RNA utilizando moléculas de DNA recebe o nome de transcrição. Na transcrição os nucleotídeos timina, uracila, adenina e citosina são unidos e pareados de acordo com a ordem dos nucleotídeos complementares do DNA. Essa união não ocorre espontaneamente, ela é catalisada por enzimas chamadas RNA polimerases. O processo de transcrição inicia-se com a ligação da RNA polimerase ao

DNA na direção da extremidade 5' → extremidade 3' por meio de uma sequência chamada promotor, com isso a polimerase determina a separação das duas cadeias do DNA e deixa exposto o primeiro nucleotídeo que vai ser lido. A polimerase, além de realizar a catalisação, desliza sobre o DNA separando os nucleotídeos do DNA para a leitura, unindo-os após a leitura. Quando a polimerase encontra a sequência de término presente na extremidade 3' do DNA ela é liberada e a transcrição é finalizada, tendo como produto o RNA, que também é chamado de transcrito primário (De Robertis; Hib, 2006).

Para sintetizar uma proteína a partir do RNA são necessárias três etapas (iniciação, elongação e terminação) que juntas compõem o processo de tradução. Na iniciação ocorre o acoplamento do RNA mensageiro à menor subunidade do ribossomo (organela presente no citoplasma), a união do primeiro RNA transportador ao códon de início da proteína e a junção das duas subunidades do ribossomo. A elongação é marcada por todas as reações que ocorrem desde a formação da primeira ligação peptídica (entre os aminoácidos) até a integração do último aminoácido à proteína. Na terminação o ribossomo se separa do RNA mensageiro e suas duas subunidades se separam (Amabis; Martho, 1997).

Durante o processo da síntese de proteínas existem seis etapas nas quais a expressão gênica pode ser regulada. A célula pode realizar o controle de diversas ações, como: quando e como um gene é transcrito, como o transcrito de RNA é processado, o controle de quais RNA mensageiros são enviados ao citoplasma e a suas respectivas localizações, de quais RNA mensageiros são traduzidos, de quais moléculas de RNA mensageiro serão desestabilizadas no citoplasma e de quais moléculas de proteínas serão ativadas, desativadas ou seccionadas após sua produção. A Figura 2.2 ilustra os momentos em que esse controle é possível (Alberts et al., 2010).

2.3 Conceitos computacionais

2.3.1 Tabela *Hash*

A Tabela *Hash* é uma estrutura de dados efetiva para a implementação de dicionários, armazenando e ordenando os elementos de acordo com seu valor. Essa tabela generaliza

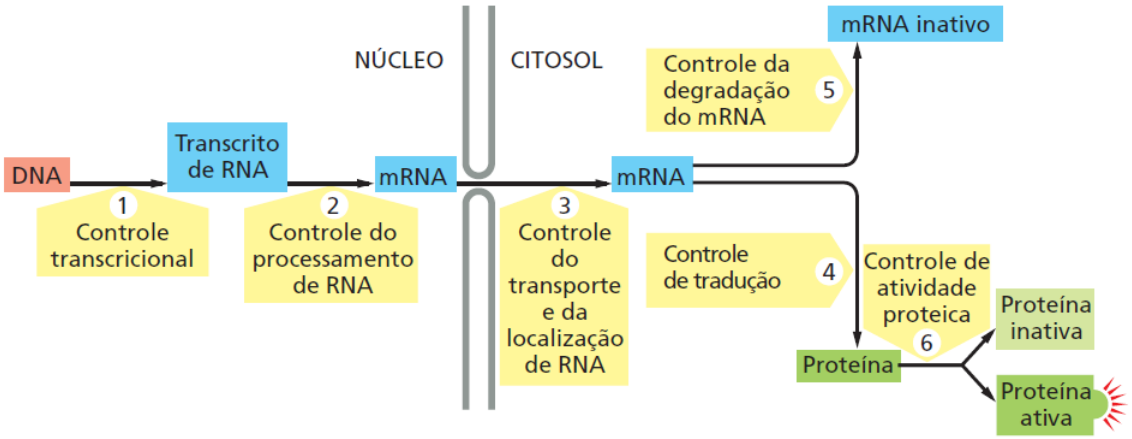

Figura 2.2: Seis momentos nas quais a expressão gênica pode ser regulada. Fonte: Alberts et al. (2010).

a noção de um vetor comum, geralmente utilizando um vetor de tamanho proporcional ao número de elementos armazenados e utilizando uma função para gerar os índices do vetor. Mesmo que procurar um elemento em uma tabela *hash* pode demorar tanto quanto uma lista, ou seja $\theta(n)$, na média ela é capaz de realizar uma pesquisa em $O(1)$ (Cormen et al., 2009).

As funções utilizadas para gerar os índices da tabela *hash* são chamadas de funções de *hashing*. Uma boa função de *hashing* é aquela que distribui os elementos de maneira uniforme sem que haja colisão, ou seja, nenhum elemento tenha um índice igual a outro elemento que já esteja armazenado, mas isso é praticamente impossível para certas tabelas. Existem várias funções de *hashing* conhecidas na literatura, tais como o método da divisão e o método da multiplicação. Sendo k o valor do elemento e m o tamanho de uma tabela, no método da divisão k é mapeado em um dos m espaços utilizando como índice o resto da divisão de k por m. Já no método da multiplicação ocorre a multiplicação de k por uma constante A, tal que $0 < A < 1$, a parte fracionária do produto é extraída e multiplicada por m, após isso esse resultado é arredondado e utilizado como índice (Cormen et al., 2009).

2.3.2 Grafo

Um grafo $G = (V, E)$ é uma estrutura composta pelos conjuntos V e E, tal que os elementos de E são formados por subconjuntos de 2 elementos de V. Os elementos de V

são chamados vértices do grafo G e os de E são chamados arestas (Diestel, 2000).

Os grafos podem ser classificados em não direcionados ou direcionados (dígrafos). Sendo G(V, E) um grafo não direcionado, se G possui a aresta (v, w), tanto v é adjacente com w quanto w é adjacente com v, já em um dígrafo essa propriedade não é verdadeira, com isso a aresta (v, w) apresenta apenas uma direção de v para w. Um exemplo de grafo não direcionado pode ser visto na Figura 2.3 (Szwarcfiter, 1998).

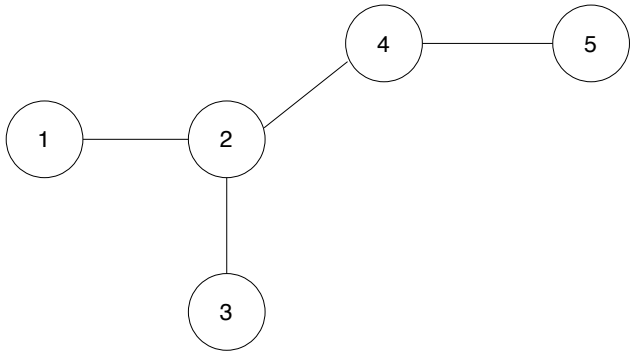

Figura 2.3: Grafo $G = (V, E)$, onde $V = \{1, 2, 3, 4, 5\}$ e $E = \{\{1, 2\}, \{2, 3\}, \{2, 4\}, \{4, 5\}\}$.

2.3.3 *Heap* binária

A *Heap* binária é um vetor de objetos que representa uma árvore binária na qual todos os níveis, exceto o último, estão necessariamente completos. Considerando um vetor de tamanho n e um índice i, tal que $1 \leq i \leq n$, o índice do elemento pai é $i/2$, o do filho à esquerda $2i$ e o do filho à direita $2i + 1$, como exibe a Figura 2.4 (Cormen et al., 2009).

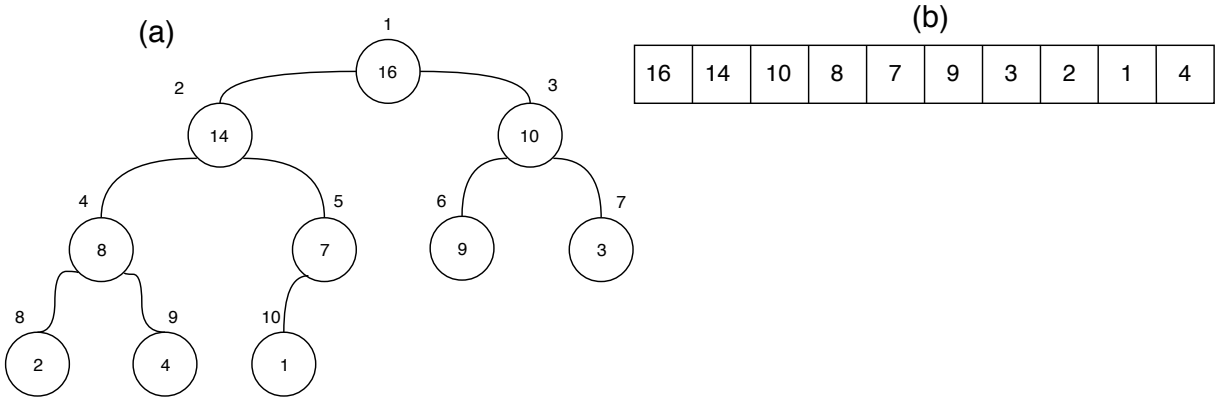

Figura 2.4: Árvore binária em *(a)* e o vetor da *heap* correspondente em *(b)*. Adaptado de Cormen et al. (2009).

Uma *heap* binária pode ser categorizada em *max-heap* ou *min-heap*. A *max-heap* satisfaz a propriedade de que, para cada elemento o valor do elemento pai é maior do que o de seus filhos, já na *min-heap* o valor do elemento pai é menor do que o de seus filhos (Cormen et al., 2009).

2.4 Simulação de redes de regulação gênica

Nessa seção são apresentados os conceitos de simulação determinística e estocástica, além de uma descrição detalhada sobre os algoritmos de simulação estocástica apresentados nesse trabalho. Para a melhor compreensão dos algoritmos é necessário o conhecimento de algumas constantes, entre elas:

- N: número de espécies da rede gênica;
- M: número de reações da rede gênica;
- a_0: somatório de todas as propensões da rede em um tempo t;
- t: tempo atual de simulação;
- T: tempo total de simulação.

2.4.1 Simulação determinística

A simulação determinística é calculada utilizando um conjunto de equações diferenciais ordinárias (EDOs) acopladas, em que para cada espécie molecular existe uma EDO no conjunto. Cada equação representa a alteração da concentração de uma espécie em função da concentração de todas as outras espécies de acordo com os coeficientes estequiométricos e constantes das reações que envolvem essa espécie (Gillespie, 1977; Klipp et al., 2016).

2.4.2 Simulação estocástica

A abordagem determinística é amplamente utilizada para a simulação de redes bioquímicas, porém apresenta problemas em alguns casos, entre eles: não ser capaz de considerar as flutuações dos números de moléculas de subsistemas que possuem poucas partículas, não

poder descrever sistemas bi- ou multi-estáveis corretamente e ainda necessitar de estocasticidade, onde a sua ausência pode gerar resultados indesejados. O uso da simulação estocástica é uma solução para esses casos, ela trata as constantes como probabilidades por unidade de tempo, na forma de um passeio aleatório de Markov, comandado por uma única EDO chamada Equação Mestra. Devido a isso essa simulação apresenta como desvantagem a demanda de muito poder computacional para sua realização (Pahle, 2009; Gillespie, 1977).

2.4.3 Algoritmos de simulação estocástica

Para contornar o problema da grande necessidade de poder computacional para realizar a simulação estocástica, Gillespie (1976) desenvolveu um método que utiliza uma aproximação à Equação Mestra usando elementos de Monte Carlo, baseada na Função de Densidade da Probabilidade da Reação, presente na equação 2.1. Essa função determina a probabilidade $P(\tau, \mu|x, t)d\tau$ que a partir de um estado x em um tempo t, a próxima reação no sistema que irá ocorrer no intervalo de tempo $[t + \tau, t + \tau + d\tau]$, nesta função os tempos estão exponencialmente distribuídos. Esse método ficou conhecido como *Stochastic Simulation Algorithm* (SSA) (Gillespie, 1976; Pahle, 2009).

$$P(\tau, \mu|x, t) = a_\mu(x) exp\bigg(- \sum_{\mu=1}^{M} a_\mu(x)\tau \bigg) \tag{2.1}$$

Gillespie (1976) propôs o *First Reaction Method* (FRM), o algoritmo calcula para cada reação i um tempo τ_i e seleciona a reação μ, sendo ela a que possui o menor tempo (τ_μ) entre todos os tempos absolutos e executa a reação. Um dos problemas desse algoritmo é a necessidade de atualizar todos tempos de disparo em cada iteração, assim o algoritmo tem uma complexidade assintótica igual a *O(M)*, onde M é o número de reações da rede. O funcionamento geral desse algoritmo pode ser visto na Figura 2.5 e de forma detalhada no Algoritmo 1, presente no anexo.

Outra contribuição de Gillespie (1977) é o algoritmo *Direct Method* (DM), que utiliza dois números aleatórios de uma distribuição uniforme no intervalo $(0, 1)$ em cada iteração para obter o índice μ da próxima reação da rede e também o tempo τ em que

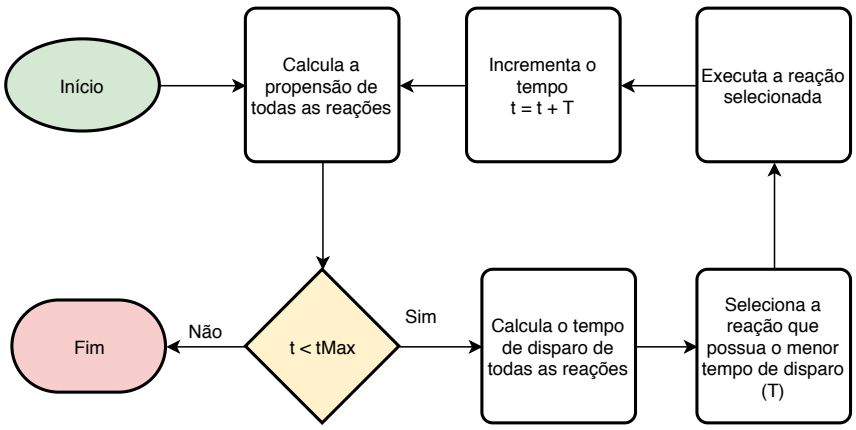

Figura 2.5: Fluxograma do algoritmo *First Reaction Method*. Adaptado de Silva (2014).

ela ocorre (Figura 2.6 e Algoritmo 2). Apesar do DM também apresentar complexidade $O(M)$ ele é mais eficiente do que o FRM, pois enquanto o DM gera 2 números aleatórios por iteração o FRM gera M números.

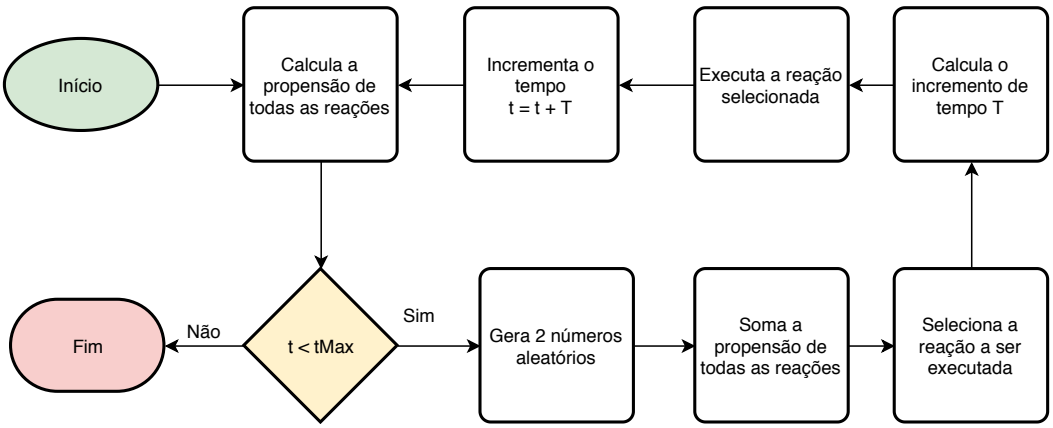

Figura 2.6: Fluxograma do algoritmo *Direct Method*. Adaptado de Silva (2014).

Analisando o FRM, Gibson e Bruck (2000) elaboraram um novo algoritmo chamado *Next Reaction Method* (NRM), que pode ser visto na Figura 2.7 e no Algoritmo 3. No NRM os $M - 1$ tempos absolutos que não foram utilizados na iteração atual são modificados para reuso nas próximas iterações. O NRM utiliza duas estruturas para aumentar o seu desempenho: a fila de prioridade indexada para armazenar os tempos e o Grafo de Dependência (DG) para auxiliar o cálculo da propensão.

Uma fila de prioridades indexada utiliza uma árvore de pares ordenados (i, τ_i), comumente implementada como uma *heap*, onde i é o número da reação e τ_i é o tempo absoluto de quando a reação i ocorre, e uma lista que o $i-$ésimo elemento é um ponteiro para a posição da árvore que contém (i, τ_i), a árvore tem a propriedade de cada pai é

menor do que seus filhos (Gibson e Bruck, 2000).

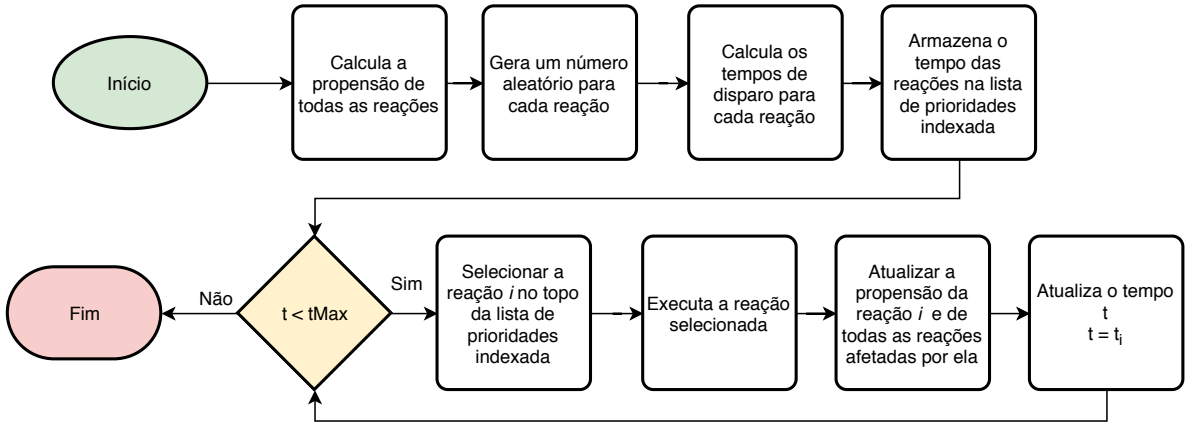

Figura 2.7: Fluxograma do algoritmo *Next Reaction Method*. Adaptado de Silva (2014).

O Grafo de Dependência é uma estrutura que indica quais propensões devem ser recalculadas a partir de uma reação disparada na iteração atual. Sendo $G(V, E)$ um grafo orientado onde V é o conjunto de reações da rede R_i o conjunto de reagentes e P_i o conjunto de produtos da reação i, seguem as definições (Gibson e Bruck, 2000):

- $DependeDe(a_i) = R_i$ é o conjunto de substâncias que afetam o valor de a_μ;
- $afeta(i) = R_i \cup P_i$ é o conjunto de substâncias que alteram a quantidade quando a reação i é executada;
- Existe uma aresta orientada de v_i para v_j, onde v_i e $v_j \in V$, se e somente se, $afeta(v_i) \cap DependeDe(a_{v_j}) \neq \emptyset$, onde a aresta orientada de v_i para v_i deve ser criado caso não exista.

Com o Grafo de Dependência, o cálculo da propensão é reduzido de M vezes a cada iteração para no máximo K, onde K é o maior grau do grafo. Devido a todas essas modificações o NRM possui uma complexidade assintótica igual a $O(K * log(M))$. Um exemplo de DG pode ser visto na Figura 2.8.

Outros dois algoritmos de simulação estocástica baseados no *Direct Method* são o *Optimized Direct Method* (ODM) (Cao et al., 2004) e o *Sorting Direct Method* (SDM) (McCollum et al., 2006). O ODM (Figura 2.10 e Algoritmo 4) foi desenvolvido com o intuito de melhorar o desempenho do *Direct Method* nas simulações de redes de regulação gênica com muitas reações. Cao et al. (2004) afirmam que nesse tipo de sistema o DM tem

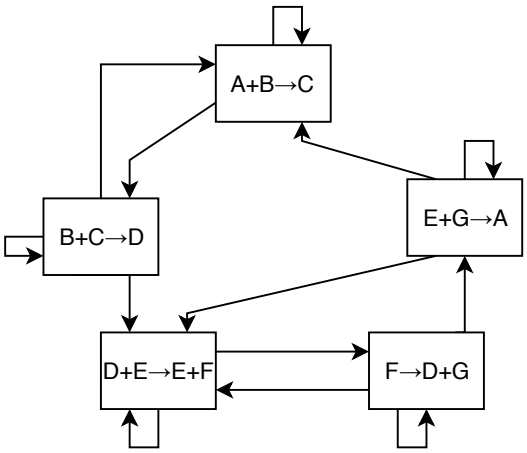

Figura 2.8: Grafo de dependência, onde cada seta indica que uma reação i afeta a reação j. Adaptado de Gibson e Bruck (2000).

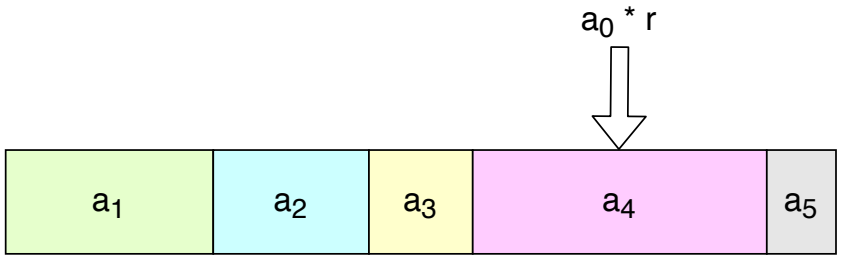

Figura 2.9: Método da roleta, onde a reação μ é escolhida a partir do somatório de todas as propensões (a_0) multiplicado por um número aleatório r, onde $r \in [0, 1]$. Adaptado de Silva (2014).

um gasto computacional elevado para realizar a seleção das reações devido a sua escolha de reação, que é feita utilizando o método da roleta (Figura 2.9). Como nesse tipo de sistema certas reações podem ter uma taxa de disparo muito maior que outras, colocá-las nas posições iniciais no vetor de propensões faz com que o gasto computacional utilizado para realizar a escolha de uma reação seja reduzido.

O SDM (Figura 2.11 e Algoritmo 5) busca melhorar o *Direct Method* utilizando uma abordagem semelhante à utilizada no ODM, porém não realiza uma pré simulação para ordenar as reações. Como essa pré ordenação pode ser custosa dependendo do modelo, o SDM realiza a ordenação durante a própria simulação. Cada vez que uma reação é executada ela é movida para uma posição em direção ao início do vetor de reações, reduzindo assim o custo da busca pela reação para a próxima vez que essa ela for executada (McCollum et al., 2006).

Anderson (2007) desenvolveu uma modificação do NRM chamada *Modified Next*

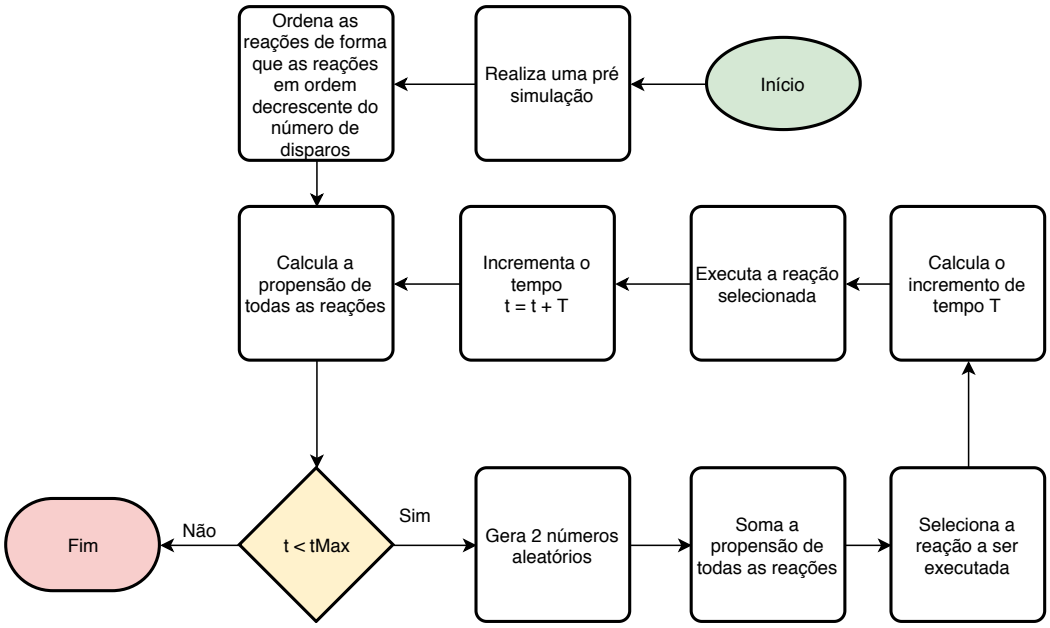

Figura 2.10: Fluxograma do algoritmo *Optimized Direct Method*.

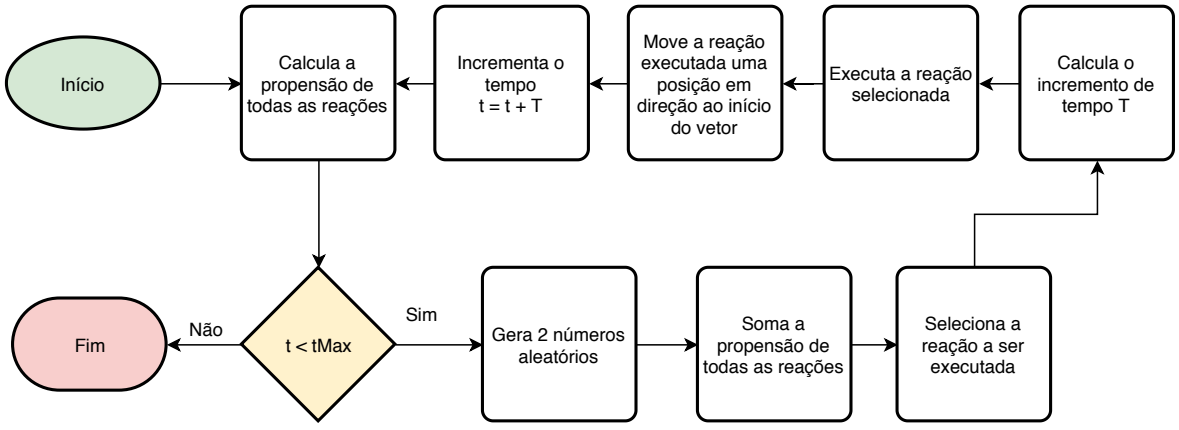

Figura 2.11: Fluxograma do algoritmo *Simplified Direct Method*.

Reaction Method (MNRM) (Figura 2.13 e Algoritmo 6), onde os tempos de inicialização das reações químicas são representados como disparos de processos de Poisson com tempos internos fornecidos por funções de propensões integradas, com isso o MNRM não precisa da *heap*, mas tem que recalcular o tempo de disparo de todas as reações a cada iteração. Outra deficiência do MNRM é que ele pode obter um desempenho equivalente ao NRM em sistemas com muitas reações (Silva, 2014).

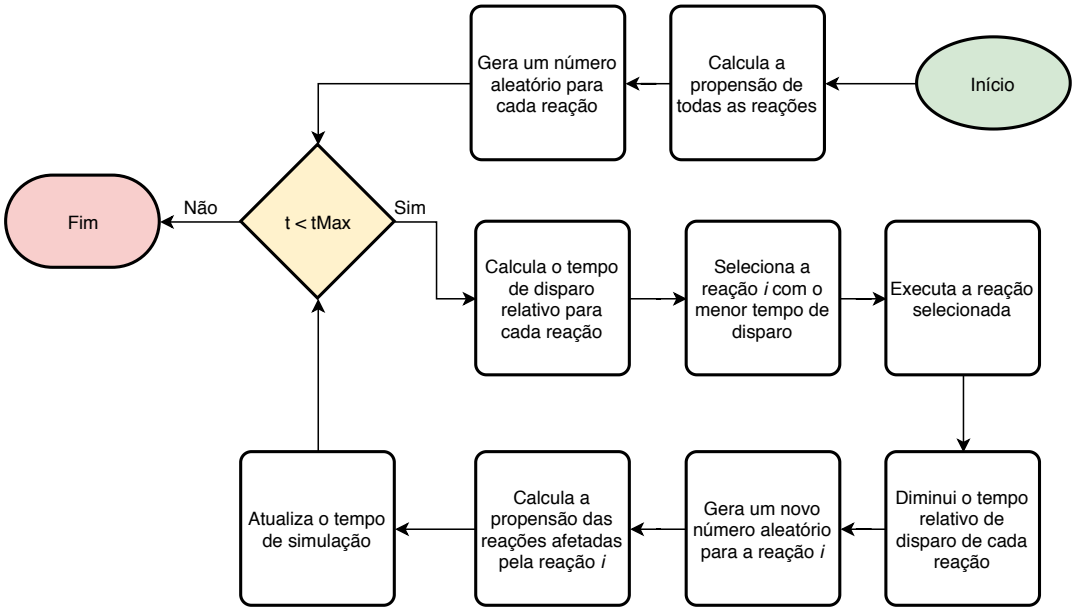

Figura 2.12: Fluxograma do algoritmo *Modified Next Reaction Method*. Adaptado de Silva (2014).

Para resolver a ineficiência do MNRM quando utilizado em redes com muitas reações Silva (2014) desenvolveram o algoritmo *Simplified Next Reaction Method* (SNRM) (Figura Figura **??** e Algoritmo 7) que é baseado no NRM. O SNRM não apresenta exceções para o cálculo dos tempos de disparo da reação e possui tempos de disparo atualizados apenas para as reações afetadas pela reação executada. Contudo o SNRM não utiliza tempos relativos, com isso precisa guardar o tempo U da última atualização do tempo de disparo de cada reação. Sendo U_i o tempo da última atualização de propensão da reação i, t o tempo atual da simulação e P_i o primeiro tempo interno após T_i, onde T_i segue a fórmula de atualização vista na equação 2.2, a fórmula de recálculo do tempo de disparo se torna a fórmula da 2.3 (Silva, 2014).

$$T_i = T_i + a_i(t - U_i) \tag{2.2}$$

$$\tau = \frac{P_i - T_i}{a_i} + t \tag{2.3}$$

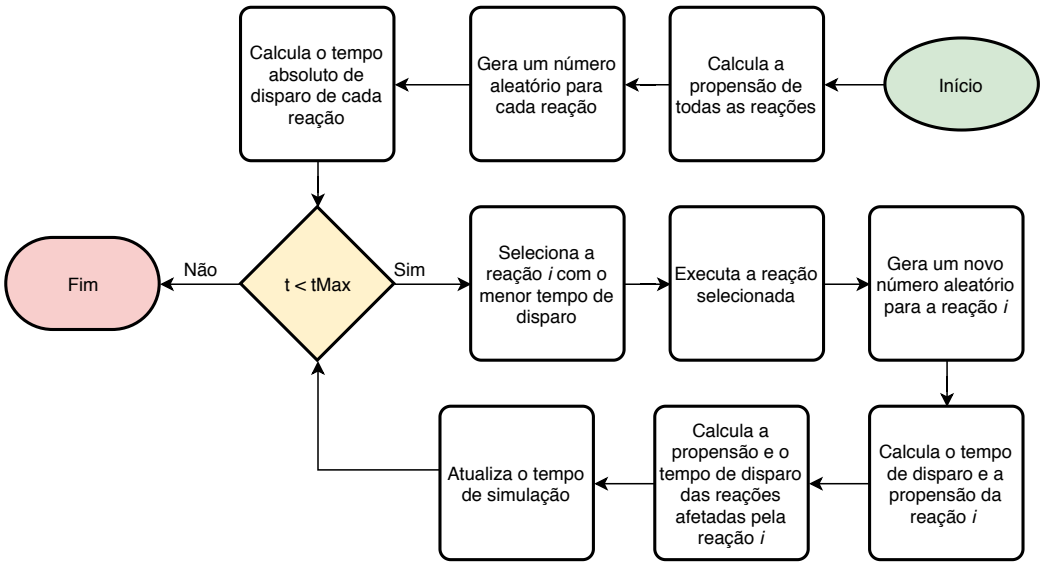

Figura 2.13: Fluxograma do algoritmo *Simplified Next Reaction Method*. Adaptado de Silva (2014).

Apesar dos algoritmos de simulação estocástica propostos por Gillespie (1976, 1977) e suas modificações considerarem as expressões gênicas como processos instantâneos, essas expressões podem possuir processos com mais de um passo. Com o avanço nos estudos do efeito do *delay* na simulação estocástica de redes de regulação gênica, Bratsun et al. (2005) desenvolveram a primeira versão do SSA com *delay* chamado de DSSA, ou apenas *Rejection Method* (RM), que é baseado no DM e utiliza uma lista para salvar os produtos que estejam em atraso, ou seja, produtos que tenham *delay* em uma reação disparada. Por causa dessa lista o RM pode ter um desempenho pior quando comparado ao DM (Ribeiro, 2010). Uma visão geral de seu funcionamento pode ser visto na Figura 2.14.

Outra contribuição de Silva (2014) foi o Grafo de Dependência para Reações com *Delay*, ou *Delayed Dependency Graph* (DDG), que foi baseado no DG. O DDG consiste em um dígrafo G(V, E) no qual V é a união entre o conjunto de substâncias e o conjunto das reações, e E sendo o conjunto de arestas direcionadas da substância i (s_i) para a reação j (r_j), satisfazendo a restrição $s_i \cap Reagentes(r_j) \neq \emptyset$ (Silva, 2014). Sendo a equação 2.4 um modelo simplificado do exemplo apresentado na Figura 2.8, a construção do DDG

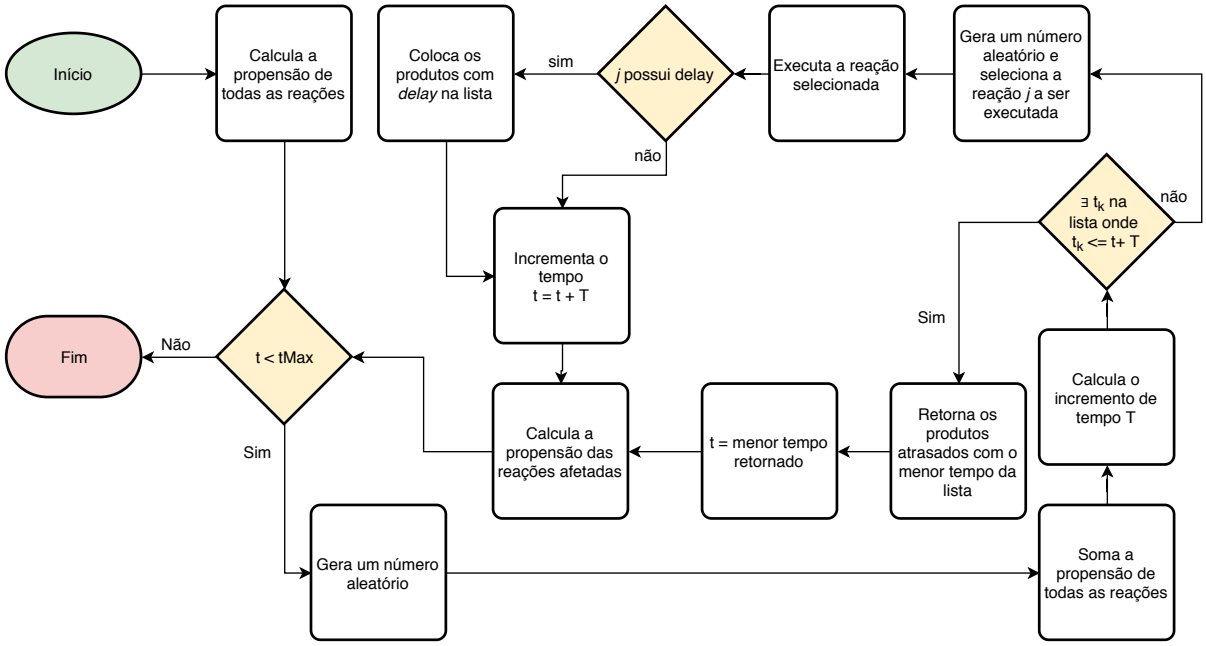

Figura 2.14: Fluxograma do algoritmo *Rejection Method*. Adaptado de Silva (2014).

equivalente pode ser vista na Figura 2.15.

$$\begin{aligned}
&\text{Reação 1: } A + B \rightarrow C \\
&\text{Reação 2: } B + C \rightarrow D \\
&\text{Reação 3: } D + E \rightarrow E + F \\
&\text{Reação 4: } F \rightarrow D + G \\
&\text{Reação 5: } E + G \rightarrow A
\end{aligned} \tag{2.4}$$

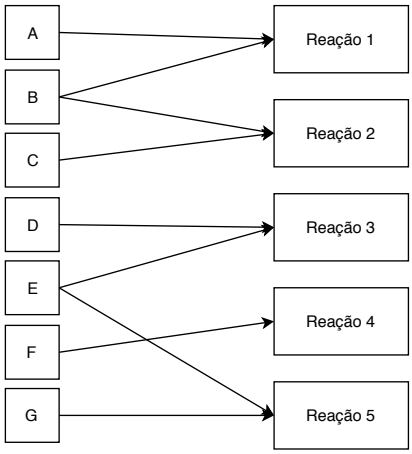

Figura 2.15: Exemplo de DDG para o modelo da equação 2.4.

3 Metodologia

Este trabalho foi realizado utilizando uma abordagem descritiva na qual os principais algoritmos de simulação estocástica presentes na literatura foram implementados e comparados entre si para verificar o desempenho de cada um em relação a redes de regulação gênica com diferentes graus de acoplamento. O algoritmo de simulação estocástica com *delay* foi implementado utilizando três diferentes estruturas de dados - lista ordenada, *heap* e lista circular. Todo o desenvolvimento dos algoritmos foi realizado utilizando a linguagem de programação C++11. Como o Grafo de Dependência proposto por Gibson e Bruck (2000) garante um aumento significativo na eficiência dos algoritmos, ele foi utilizado em todos os SSA, com exceção do *First Reaction Method*, pois ele necessita calcular todos os elementos do vetor de tempos a cada iteração.

O *Rejection Method* foi implementado utilizando, além do grafo de dependências, o grafo de dependências para reações com atraso (DDG). Com isso, se não houverem produtos com atraso na lista as propensões são atualizadas utilizando o DG, mas quando um produto com atraso for retornado, o cálculo das propensões das reações afetadas é feito utilizando o DDG.

3.1 Hierarquia de classes

Os algoritmos de simulação estocástica foram organizados em uma hierarquia de classes flexível, que é composta por uma classe chamada *SSA* na qual estão presentes os métodos como o cálculo da propensão e a atualização da matriz estequiométrica que são herdados pelas subclasses, cada qual um algoritmo de simulação estocástica diferente (Figura 3.1).

3.2 Estrutura para a compactação dos dados

Durante a simulação de uma rede um algoritmo necessita salvar a quantidade das espécies em cada iteração, tendo duas escolhas: armazenar os dados na memória principal ou na

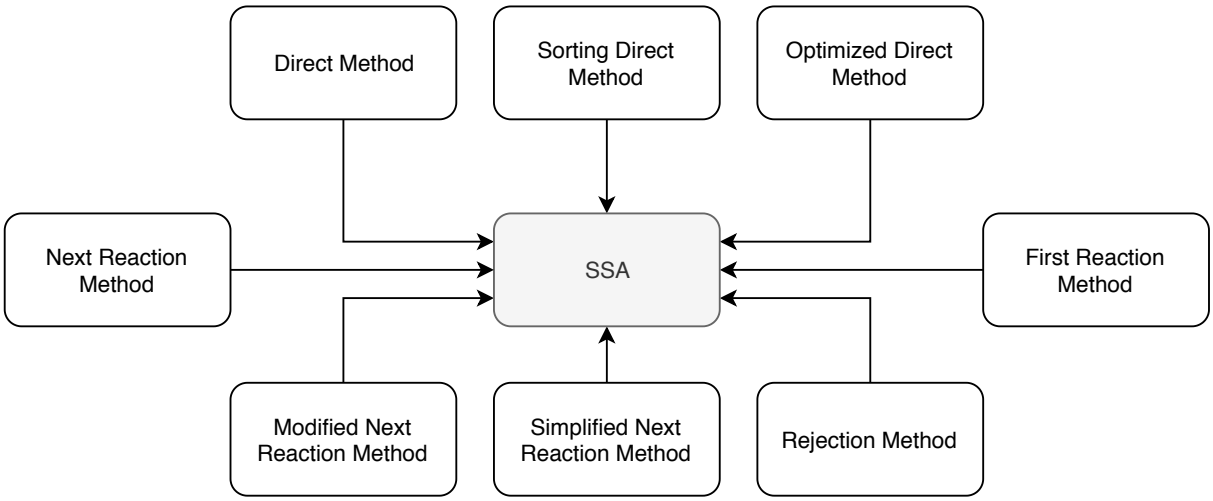

Figura 3.1: Diagrama de classes simplificado

memória secundária. Salvar na memória principal pode ser um problema em simulações longas ou com modelos complexos, pois essas simulações geram um volume massivo de dados podendo exceder o limite da memória e salvar os dados na memória secundária durante a simulação é um processo mais lento, fazendo com que o desempenho da simulação seja prejudicado. O exemplo a seguir ilustra a quantidade massiva de dados gerados durante a simulação estocástica de uma rede gênica, seja uma rede com m espécies e uma simulação de duração T e que, devido ao nível de acoplamento da rede, a simulação tenha iterações com incremento do tempo de simulação na ordem de 10^{-4} segundos. Ao final da simulação seria necessário armazenar cerca de N elementos, conforme a equação 3.1. Para contornar esse problema foi desenvolvido um tipo abstrato de dados (TAD) que recebeu o nome de Log, que armazena apenas as espécies que tiveram suas quantidades alteradas em relação às iterações anteriores.

$$N = \frac{T}{10^{-4}} \times m \tag{3.1}$$

O Log consiste em uma estrutura de dados que contem um vetor de inteiros que guarda as quantidades de todas as espécies no último tempo de simulação salvo e uma lista encadeada de ponteiros, onde cada ponteiro aponta para uma estrutura de dados contendo o tempo de simulação na iteração em que foi criado e uma estrutura de mapeamento que armazena as quantidades das espécies. A inserção de uma quantidade em um nó do Log

é melhor detalhada na Figura 3.2.

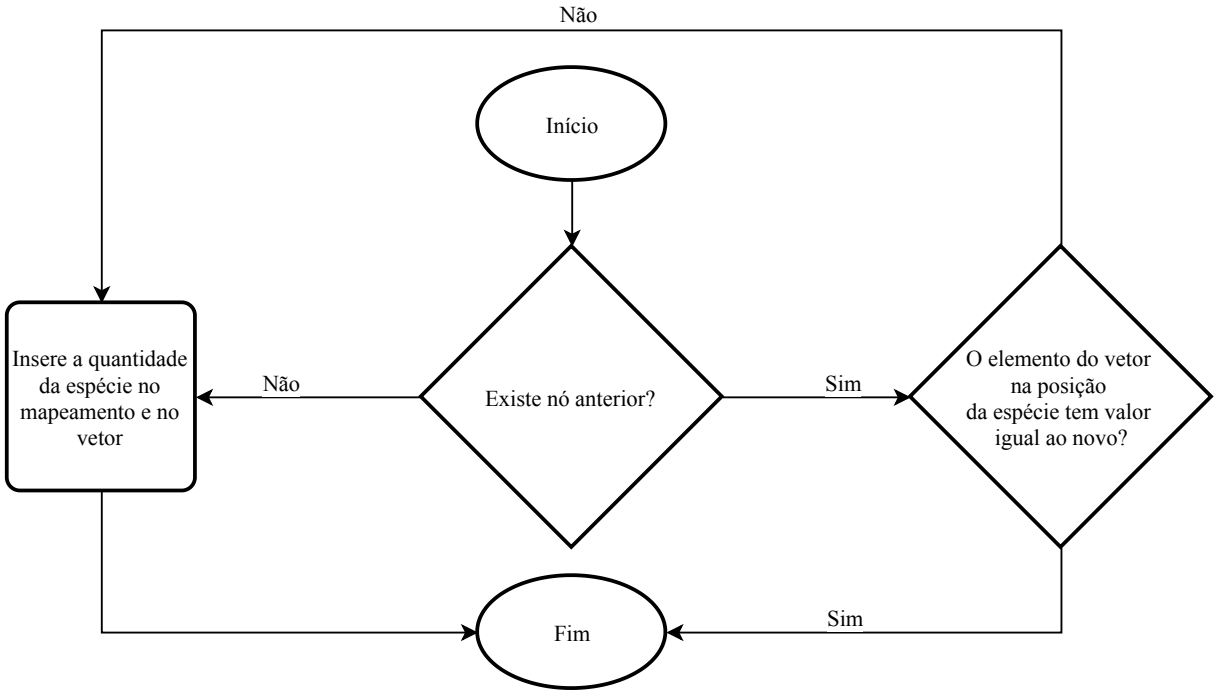

Figura 3.2: Fluxograma da inserção de uma espécie em um nó do Log.

3.3 Lista circular

O *Rejection Method* necessita realizar duas operações na estrutura que armazena os produtos com *delay* durante uma simulação: inserir um produto e retirar todos os produtos com tempo τ de forma que $\tau \in [t, t + \theta]$, onde t é o tempo atual da simulação e θ é o incremento de tempo calculado na iteração atual. A lista circular ordenada tem como característica, quando comparada com a lista ordenada, a realização dessas duas operações com um menor gasto computacional. Ao inserir um produto com *delay* na lista ordenada, a mesma tem que realocar todos os outros elementos após o índice selecionado para a inserção do produto atual, já a lista circular proposta nesse trabalho não necessita passar por todos os elementos. Apesar da operação de inserção de um produto com *delay* na lista circular possuir uma complexidade assintótica equivalente à complexidade da lista ordenada (Tabela 3.1), o comportamento geral do algoritmo possui um desempenho melhor do que a lista ordenada.

A inserção de um produto com *delay* na lista circular se dá de forma bem simples e eficiente. Para inserir um elemento na lista, primeiro é verificado e ajustado o tamanho

Tabela 3.1: Complexidade assintótica das operações de inserção e remoção de um produto com *delay* em cada estrutura.

Estrutura	Inserir elemento	Remover
Lista ordenada	$O(n)$	$O(n)$
Lista circular	$O(n)$	$O(1)$
Heap	$O(log(n))$	$O(log(n))$

da lista, depois verifica-se se o elemento é menor que o primeiro ou maior que o último, caso não seja nem um e nem outro é feita uma busca para encontrar a posição do elemento a ser inserido a partir do último até que o elemento a ser inserido seja menor que o i e maior que o da $i+1$, sendo movidos apenas $N-i$ elementos, onde N é a quantidade de elementos na lista, a Figura 3.3 exemplifica a inserção de dois produtos na lista. A remoção de elementos se dá de forma similar, como o *Rejection Method* sempre remove todos os elementos iguais ao menor, o algoritmo percorre removendo todos esses elementos e no final atribui ao próximo elemento diferente dos removidos a posição de primeiro elemento, sendo assim não são feitas realocações dos elementos na lista e apresentando uma remoção com complexidade assintótica $O(K)$, onde K é o número de produtos removidos.

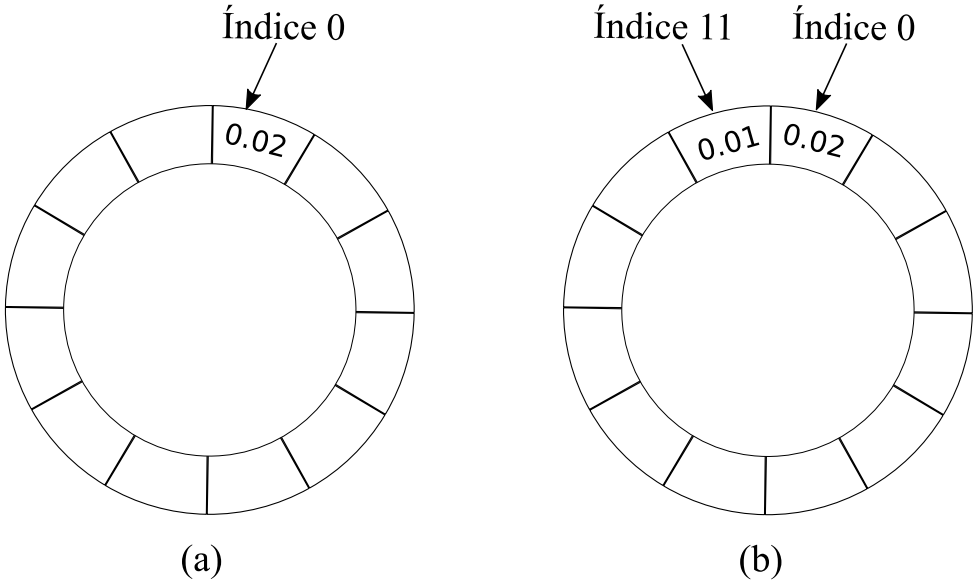

Figura 3.3: Exemplo de inserção de produto com *delay* na lista circular. Em (a) é exemplificada a inserção do produto com valor $0,02$ em uma lista vazia, já em (b) é exibido a inserção do produto com valor $0,01$ na lista resultante da Figura (a).

4 Resultados

Para a obtenção de resultados foram feitos testes em cada algoritmo utilizando redes de regulação gênica com baixo e alto nível de acoplamento, onde cada rede foi simulada mil vezes. Os algoritmos foram comparados de forma empírica de acordo com o tempo gasto com a simulação. O número elevado de simulações de cada rede foi necessário para garantir a convergência do tempo de simulação, porém como trata-se de um teste empírico os resultados podem sofrer alterações de acordo com o estado do dispositivo no qual os testes foram realizados.

Todas as simulações foram realizadas em um computador com os seguintes requisitos: sistema operacional *Elementary OS 5.0* 64 bits, 6GB de memória RAM DDR3 e um processador Intel i5-2320. Os testes foram realizados nesse dispositivo com o intuito de simular uma experiência de uso real, porém foi feita a verificação de existência de processos em segundo plano para evitar variações nos valores das simulações.

Para realizar a simulação estocástica das redes utilizadas nos testes algumas constantes foram utilizadas, entre elas a quantidade inicial das espécies, constantes de velocidades das reações, tempo de simulação e *delay*. A mudança em qualquer uma dessas constantes afeta o resultado da simulação e seus valores foram escolhidos de forma a facilitar a compreensão dos resultados.

4.1 Redes de regulação gênica com baixo nível de acoplamento

Redes com baixo nível de acoplamento tem como característica o baixo nível de dependência entre suas reações, com isso o uso do Grafo de Dependências tem seu impacto reduzido (mas ainda possui um desempenho considerável em relação a métodos que não o utilizam), pois os algoritmos não necessitam calcular a propensão de todas as reações da redes. Assim o tempo de simulação passa a ser influenciado principalmente pela quan-

tidade de números aleatórios utilizados em cada iteração dos algoritmos.

A seguinte rede foi projetada para ter um baixo nível de acoplamento, porém um grande número de reações. Para os testes foram utilizados $N = 100$, constante de velocidade ($k_1 = 1$), todas as espécies com quantidade inicial igual a 1000 e o tempo máximo de simulação igual a 100 segundos.

$$\begin{aligned} &S_i \xrightarrow{k_1} S_j \\ &i = 1, 3, 5, .., (N-1) \text{ e } j = 2, 4, 6, ..., N \end{aligned} \tag{4.1}$$

Tabela 4.1: Resultado das simulações do modelo fracamente acoplado.

Algoritmo	Tempo médio (s)	Desvio padrão
DM	0,0869	0,0010
FRM	1,8079	0,0066
MNRM	0,1075	0,0014
NRM	**0,0860**	0,0007
ODM	0,1471	0,0014
SDM	0,0873	0,0004
SNRM	**0,0860**	0,0006

Como o *First Reaction Method* não utiliza o Grafo de Dependências e necessita gerar M números aleatórios durante cada iteração seu resultado foi muito pior em comparação aos outros. Os outros algoritmos obtiveram resultados bem próximos, com o SNRM e o NRM ficaram com os melhores resultados. Analisando esses resultados concluise que em redes fracamente acopladas a melhor solução seria utilizar o SNRM devido a sua facilidade de implementação em relação ao NRM.

Para o teste dos métodos com *delay* a rede foi modificada para incluir um *delay* (d_1) com valor igual a $0,1$. Ficando no formato da equação 4.2.

$$\begin{aligned} &S_i \xrightarrow{k_1} S_j(d_1) \\ &i = 1, 3, 5, .., (N-1) \text{ e } j = 2, 4, 6, ..., N \end{aligned} \tag{4.2}$$

Analisando os resultados é possível notar a melhora de desempenho da lista circular em relação à lista ordenada, mesmo assim a *heap* ainda conseguiu o melhor resultado entre as estruturas.

Tabela 4.2: Resultado das simulações do algoritmo *Rejection Method* utilizando a rede fracamente acoplada.

Algoritmo	Tempo médio (s)	Desvio padrão
RM-OL	1,4210	0,0323
RM-CL	0,8642	0,0186
RM-H	**0,1389**	0,0008

4.2 Redes de regulação gênica com alto nível de acoplamento

Redes de regulação gênica com alto nível de acoplamento possuem um grande número de dependências entre as reações, com isso algoritmos que utilizam o grafo de dependências possuem um desempenho muito melhor do que os que não o utilizam.

O primeiro modelo de rede com alto nível de acoplamento utilizado para os testes é chamado de *Colloidal Aggregation*, proposto por Ramaswamy e Sbalzarini (2011). Esse modelo é composto pelo conjunto de equações presentes em 4.3. Para a realização dos testes a constante N foi utilizada com valor 7, todas as espécies tem quantidade inicial 1 e cada reação possui constante de velocidade k_1 igual a 1 e um tempo de simulação de 100 segundos.

$$\begin{aligned} S_n + S_m &\xrightarrow{k_1} S_{n+m},\ n+m = 2,3,...,N, \\ S_{n+m} &\xrightarrow{k_1} S_n + S_m\ n+m = 1,2,3,... \end{aligned} \tag{4.3}$$

Tabela 4.3: Resultado das simulações do modelo *Colloidal Aggregation*

Algoritmo	Tempo médio (s)	Desvio padrão
DM	**0,0031**	0,0001
FRM	0,0084	0,0003
MNRM	0,0034	0,0002
NRM	0,0056	0,0003
ODM	0,0040	0,0002
SDM	**0,0031**	0,0001
SNRM	0,0040	0,0002

Como o número de reações da rede é pequeno o FRM não teve um resultado tão discrepante em relação outros algoritmos, mesmo obtendo o pior resultado entre eles. Os melhores algoritmos foram o SDM e o DM. Para uma rede como essa, com um pequeno

número de reações e com um nível de acoplamento alto (porém não tão alto a ponto de se tornar um grafo completo) todos os algoritmos apresentaram resultados semelhantes. Assim a escolha do algoritmo não influencia muito na simulação (desde que ele utilize o DG).

Para a realização das simulações com *delay* com base no *Colloidal Aggregation Model* as reações foram modificadas para o formato da equação 4.4, onde d_1 é equivale ao *delay*, com valor 1 nesse caso.

$$\begin{aligned} S_n + S_m &\xrightarrow{k_1} S_{n+m},\ n+m = 2,3,...,N, \\ S_{n+m} &\xrightarrow{k_1} S_n(d_1) + S_m(d_1)\ n+m = 1,2,3,... \end{aligned} \tag{4.4}$$

Tabela 4.4: Resultado das simulações do algoritmo *Rejection Method* utilizando o modelo *Colloidal Aggregation* modificado.

Algoritmo	Tempo médio (s)	Desvio padrão
RM-OL	0,0034	0,0002
RM-CL	**0,0032**	0,0002
RM-H	**0,0032**	0,0002

As simulações do *Rejection Method* tiverem resultados parecidos nas três estruturas de dados, onde a lista circular obteve um resultado igual à *heap*, nesse teste a lista ordenada também obteve um resultado bem próximo ao das outras estruturas.

A segunda rede de regulação gênica utilizada para os testes consiste na rede da equação 4.5, a qual seu nível de acoplamento é ainda maior do que o *Colloidal Aggregation*. Para as simulações dessa rede foram adotados as constantes com os seguintes valores: N igual a 50 reações e k_1 com valor 1, tempo de simulação equivalente a um segundo e quantidade inicial das espécies igual a 1.

$$S_n \xrightarrow{k_1} S_1 + S_2 + ... + S_{n-1},\ n = 1...N \tag{4.5}$$

Nessa simulação é possível notar a grande eficiência do DG nos algoritmos de simulação estocástica. O FRM ficou com um resultado muito pior do que os outros e novamente o DM obteve um resultado melhor do que os outros algoritmos.

Para a realização dos testes com os algoritmos de simulação estocástica com *delay*,

Tabela 4.5: Resultado das simulações da rede fortemente acoplada

Algoritmo	Tempo médio (s)	Desvio padrão
DM	**0,1440**	0,0697
FRM	0,8577	0,4576
MNRM	0,1645	0,0829
NRM	0,1732	0,0843
ODM	0,1487	0,0763
SDM	0,1444	0,0742
SNRM	0,1602	0,0788

foram adicionados *delays* d_1 com valor $0,1$ em todos os produtos.

$$S_n \xrightarrow{k_1} S_1(d_1) + S_2(d_1) + ... + S_{n-1}(d_1),\ n = 1...N \tag{4.6}$$

Tabela 4.6: Resultado das simulações do algoritmo *Rejection Method* utilizando a rede fortemente acoplada.

Algoritmo	Tempo médio (s)	Desvio padrão
RM-OL	0,7269	0,5370
RM-CL	0,1458	0,0988
RM-H	**0,0210**	0,0079

Para esse teste a *heap* apresentou o melhor resultado, com a lista circular logo após. Essa rede possui um grande número de produtos em suas reações fazendo com que o número de inserções seja alto, influenciando no desempenho da lista ordenada e na lista circular.

5 Considerações finais

5.1 Conclusão

O presente trabalho apresentou um estudo sobre os principais algoritmos de simulação estocástica presentes na literatura, analisando seu desempenho quando desenvolvidos na mesma linguagem e executados em um mesmo dispositivo.

A maioria dos algoritmos apresentados nesse trabalho foram desenvolvidos em uma época onde os recursos computacionais eram escassos, porém atualmente temos recursos abundantes para a realização dessas simulações. Com isso, a análise do desempenho dos algoritmos mostra que quando implementados com estruturas como o Grafo de Dependências os algoritmos têm um desempenho muito similar entre si. De acordo com os testes o DM obteve o melhor resultado na maioria das simulações. Para um pesquisador que não possua um conhecimento aprofundado em computação, pode ser vantajoso utilizar o DM ou até variações dele como o SDM, pela simplicidade de implementação quando comparado ao SNRM que também obteve resultados muito bons em relação aos demais.

Esse trabalho também apresentou uma estrutura de lista circular com o intuito de melhorar o desempenho do algoritmo *Rejection Method* utilizando a lista ordenada. A lista circular obteve um melhor tempo de execução em relação à lista ordenada, porém não conseguiu superar a *Heap*, que também foi implementada para motivos de comparação. Apesar dos resultados a lista circular ainda se mostra uma boa opção para ser utilizada no lugar da lista ordenada e da *heap* devido a sua fácil implementação.

Além dos resultados de análise e do desenvolvimento da lista circular, o presente trabalho obteve como produto de seus objetivos uma ferramenta de código aberto que agrega todos os algoritmos aqui citados, tornando-se uma ferramenta funcional capaz de realizar simulações individualmente e em lote, armazenar a dinâmica das reações no decorrer do tempo e ainda exportar esses resultados de maneira a facilitar o compreendimento do usuário. Essa ferramenta pode ser encontrada em seu repositório no *GitHub*[1].

[1] `https://github.com/rafaelstjf/STOCBIONET`

5.2 Trabalhos futuros

Como trabalhos futuros, pretende-se melhorar ainda mais a lista circular, desenvolver uma estrutura que consiga um resultado melhor do que o apresentado utilizando a *heap* e também aprimorar a ferramenta gerada como produto desse trabalho, adicionando novos métodos, possibilidade de utilização de novas cinéticas, simulações com volume variável e uma interface gráfica para melhorar a usabilidade do usuário.

Bibliografia

Alberts, B.; Johnson, A.; Lewis, J.; Morgan, D.; Raff, M.; Roberts, K.; Walter, P.; Wilson, J. ; Hunt, T. **Biologia molecular da célula**. Artmed Editora, 2010.

Amabis, J.; Martho, G. Do gene à proteína. **Atualidades Biológicas, Editora Moderna**, 1997.

Anderson, D. F. A modified next reaction method for simulating chemical systems with time dependent propensities and delays. **The Journal of chemical physics**, v.127, n.21, p. 214107, 2007.

Bratsun, D.; Volfson, D.; Tsimring, L. S. ; Hasty, J. Delay-induced stochastic oscillations in gene regulation. **Proceedings of the National Academy of Sciences**, v.102, n.41, p. 14593–14598, 2005.

Cao, Y.; Li, H. ; Petzold, L. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. **The journal of chemical physics**, v.121, n.9, p. 4059–4067, 2004.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. ; Stein, C. **Introduction to algorithms**. MIT press, 2009.

Cornell, B. **Dna structure**. `https://ib.bioninja.com.au/standard-level/topic-2-molecular-biology/26-structure-of-dna-and-rna/dna-structure.html`, 2016. Online; acessado em 28 de Outubro de 2018.

De Robertis, E.; Hib, J. **Bases da biologia celular e molecular**. In: Bases da biologia celular e molecular. 2006.

Diestel, R. **Graph theory, electronic edition 2000 ed**, 2000.

Emmert-Streib, F.; Dehmer, M. ; Haibe-Kains, B. Gene regulatory networks and their applications: understanding biological and medical problems in terms of networks. **Frontiers in cell and developmental biology**, v.2, p. 38, 2014.

Fowler, S.; Roush, R.; Wise, J. ; Stronck, D. **Concepts of Biology**. OpenStax College, Rice University, 2013.

Gibson, M. A.; Bruck, J. Efficient exact stochastic simulation of chemical systems with many species and many channels. **The journal of physical chemistry A**, 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**, v.22, n.4, p. 403–434, 1976.

Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. **The journal of physical chemistry**, v.81, n.25, p. 2340–2361, 1977.

Klipp, E.; Liebermeister, W.; Wierling, C.; Kowald, A. ; Herwig, R. **Systems biology: a textbook**. John Wiley & Sons, 2016.

Linden, R. **Algoritmos genéticos (3a ediçao)**. Ciência Moderna, 2012.

Marcus, F. **Bioinformatics and systems biology: collaborative research and resources**. Springer Science & Business Media, 2008.

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**, v.30, n.1, p. 39–49, 2006.

Pahle, J. Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. **Briefings in bioinformatics**, v.10, n.1, p. 53–64, 2009.

Ramsey, Stephen e Orrell, D. e. B. H. Dizzy: stochastic simulation of large-scale genetic regulatory networks. **Journal of bioinformatics and computational biology**, v.3, n.02, p. 415–436, 2005.

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**, v.134, n.1, p. 014106, 2011.

Ribeiro, A. S. Stochastic and delayed stochastic models of gene expression and regulation. **Mathematical biosciences**, v.223, n.1, p. 1–11, 2010.

Silva, C. d. L. F. d. **Novos algoritmos de simulação estocástica com atraso para redes gênicas**. 2014. Dissertação de Mestrado - Universidade Federal de Juiz de Fora (UFJF).

Szwarcfiter, J. L. Grafos e algoritmos computacionais, 2a edição. **Editora Campus**, 1988.

I Pseudocódigo dos algoritmos de simulação estocástica

Algoritmo 1: Pseudocódigo do algoritmo *First Reaction Method*. Sendo τ o vetor de tempos absolutos das reações.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

```
1  Início;
2  t ← 0;
3  repita
4  |  para i ← 1 to M faça
5  |  |  calcula a_i;
6  |  |  Gera R_1 aleatoriamente por meio de uma distribuição uniforme no
   |  |  intervalo (0, 1);
7  |  |  τ_i ← −ln(R_1) /a_i(t);
8  |  fim
9  |  Selecione μ tal que ele seja o índice do menor elemento de τ;
10 |  D(t + τ_μ)←D(t) +E_μ;
11 |  t ← t + τ_μ;
12 até t < T;
13 Fim;
```

Algoritmo 2: Pseudocódigo do algoritmo *Direct Method*.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

1 *Início*;
2 $\mathsf{t} \leftarrow 0$;
3 **repita**
4 | **para** $i \leftarrow 1$ **to** M **faça**
5 | | *calcula* a_i;
6 | **fim**
7 | $a_0 \leftarrow \sum_{i=1}^{M} a_i$;
8 | *Gera* R_1 *e* R_2 *aleatoriamente por meio de uma distribuição uniforme no intervalo* $(0, 1)$;
9 | $\theta \leftarrow -\texttt{ln}(R_1)\ / a_0(t)$;
10 | *Selecione* μ *tal que* $\sum_{k=1}^{\mu-1} a_k(t) < R_2 a_0(t) \leq \sum_{k=1}^{\mu} a_k(t)$;
11 | $\mathsf{D}(\mathsf{t} + \theta) \leftarrow \mathsf{D}(t) + E_\mu$;
12 | $\mathsf{t} \leftarrow \mathsf{t} + \theta$;
13 **até** $\mathsf{t} < \mathsf{T}$;
14 *Fim*;

Algoritmo 3: Pseudocódigo do algoritmo *Next Reaction Method*. Sendo, P a fila de prioridades indexada, t_1 o tempo na qual uma determinada reação i tem sua propensão igual a zero pela primeira vez, t_2 o tempo em que a reação i deixa de ser 0 e $a_{i,velho}$ a última propensão diferente de zero da reação i.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

```
1  Início;
2  Gera o grafo de dependência DG;
3  para i ← 1 to M faça
4  |  calcula a_i;
5  |  Gera R_1 aleatoriamente por meio de uma distribuição uniforme no
   |  intervalo (0,1);
6  |  τ_i ← − ln(R_1) /a_i(t);
7  |  Armazena o valor de t_i em P;
8  fim
9  t ← 0;
10 repita
11 |  Seleciona μ tal que seja o índice do menor elemento de P;
12 |  D(τ_μ) ←D(t) +E_μ;
13 |  t ← τ_μ;
14 |  para cada aresta (μ, i) DG faça
15 |  |  Calcula a_i;
16 |  |  se i ≠ μ então
17 |  |  |  se a_i = 0 então
18 |  |  |  |  τ_i ← (a_{i,velho}/a_i)(τ_i − t_1) + t_2;
19 |  |  |  senão
20 |  |  |  |  τ_i ← (a_{i,velho}/a_i)(τ_i − t )+ t;
21 |  |  |  fim
22 |  |  senão
23 |  |  |  Gera R_2 aleatoriamente por meio de uma distribuição uniforme
   |  |  |  no intervalo (0,1);
24 |  |  |  τ_i ← (−ln(R_2)/a_i(t))+ t;
25 |  |  fim
26 |  fim
27 até t < T;
28 Fim;
```

Algoritmo 4: Pseudocódigo do algoritmo *Optmized Direct Method*.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

Início;
Gera o grafo de dependência DG;
Realiza uma simulação com tempo menor do que T;
Ordena as reações de modo que a reação mais frequente esteja no índice 1 *e a menos frequente no índice M*;
para $i \leftarrow 1$ **to** M **faça**
    calcula a_i;
fim
$a_0 \leftarrow \sum_{i=1}^{M} a_i$;
$\mathsf{t} \leftarrow 0$;
repita
    Gera R_1 *e* R_2 *aleatoriamente por meio de uma distribuição uniforme no intervalo* $(0, 1)$;
    $\theta \leftarrow -\mathtt{ln}(R_1)\ / a_0(t)$;
    Selecione μ *tal que* $\sum_{k=1}^{\mu-1} a_k(t) < R_2 a_0(t) \leq \sum_{k=1}^{\mu} a_k(t)$;
    $\mathsf{D}(\mathsf{t} + \theta) \leftarrow \mathsf{D}(t) + E_\mu$;
    $\mathsf{t} \leftarrow \mathsf{t} + \theta$;
    para cada *aresta* (μ, i) *em DG* **faça**
        Calcula a_i;
    fim
    $a_0 \leftarrow \sum_{i=1}^{M} a_i$;
até $\mathsf{t} < \mathsf{T}$;
Fim;

Algoritmo 5: Pseudocódigo do algoritmo *Sorting Direct Method*. Sendo RSO o vetor para a ordem de procura da reação.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

```
1  Início;
2  Gera o grafo de dependência DG;
3  para i ← 1 to M faça
4  |  RSO_i ← i;
5  fim
6  para i ← 1 to M faça
7  |  calcula a_i;
8  fim
9  a_0 ← Σ_{i=1}^{M} a_i;
10 t ← 0;
11 repita
12 |  Gera R_1 e R_2 aleatoriamente por meio de uma distribuição uniforme no
   |  intervalo (0,1);
13 |  θ ← −ln(R_1) /a_0(t);
14 |  seletor ← R_2 a_0(t);
15 |  para i ← 1 to M faça
16 |  |  seletor ← seletor −a[ RSO[ i ] ];
17 |  |  se seletor ≤ 0 então
18 |  |  |  RSOIndice ← i;
19 |  |  |  para;
20 |  |  senão
21 |  |  fim
22 |  fim
23 |  μ ← RST[ RSOIndice ];
24 |  D(t + θ) ← D(t) + E_μ;
25 |  se μ ≠ 1 então
26 |  |  Troca os valores de RSO_μ e RSO_(μ−1);
27 |  senão
28 |  fim
29 |  t ← t + θ;
30 |  para cada aresta (μ, i) em DG faça
31 |  |  Calcula a_i;
32 |  fim
33 |  a_0 ← Σ_{i=1}^{M} a_i;
34 até t < T;
35 Fim;
```

Algoritmo 6: Pseudocódigo do algoritmo *Modified Next Reaction Method*.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

```
1  Início;
2  Gera o grafo de dependência DG;
3  para i ← 1 to M faça
4  |  L_i ← 0;
5  |  calcula a_i;
6  |  Gera R_1 aleatoriamente no intervalo (0,1);
7  |  P_i ← − ln(R_1);
8  fim
9  repita
10 |  para i ← 1 to M faça
11 |  |  Δt_i ← (P_i- L_i)/ a_i;
12 |  fim
13 |  Δt_μ ← min{Δt } ;
14 |  D(τ_μ) ←D(t) +E_μ;
15 |  t ← t + Δt_μ;
16 |  para i ← 1 to M faça
17 |  |  L_i ← L_i + a_i*Δt_μ;
18 |  fim
19 |  Gera R_2 aleatoriamente no intervalo (0,1);
20 |  P_μ ← P_μ + ln(1/R_2);
21 |  para cada aresta (μ,i) DG faça
22 |  |  Calcula a_i;
23 |  fim
24 até t < T;
25 Fim;
```

Algoritmo 7: Pseudocódigo do algoritmo *Simplified Next Reaction Method*. Sendo Z o tempo total de simulação e H a fila de prioridades indexada.

Entrada: Tempo de simulação (T), estequiometria (E), constantes de velocidade das reações (K) e quantidade inicial das espécies (S)
Saida : Dinâmica de estados (D)

1 *Início*;
2 *Gera o grafo de dependência DG*;
3 **para** $i \leftarrow 1$ **to** M **faça**
4 $T_i \leftarrow 0$;
5 *calcula* a_i;
6 *Gera* R_1 *aleatoriamente no intervalo* $(0, 1)$;
7 $P_i \leftarrow -$ `ln(`R_1`)`;
8 $U_i \leftarrow$ t;
9 $\tau_i \leftarrow (P_i - T_i)/(a_i +$ t$)$;
10 *Armazena o valor de* τ_i *em* H ;
11 **fim**
12 t $\leftarrow 0$;
13 **repita**
14 *Seleciona* μ *tal que seja o índice do menor elemento de* H;
15 D(τ_μ) $\leftarrow$D(t) $+E_\mu$;
16 t $\leftarrow \tau_\mu$;
17 *Gera* R_2 *aleatoriamente no intervalo* $(0, 1)$;
18 $P_\mu \leftarrow -$ `ln(`R_2`)`;
19 **para cada** *aresta* (μ, i) *DG* **faça**
20 $T_i \leftarrow T_i + a_i$ (t- U_i);
21 $U_i \leftarrow$ t;
22 *Calcula* a_i;
23 $\tau_i \leftarrow (P_i - T_i)/(a_i +$ t$)$;
24 *Atualiza a posição de* τ_i *em* H;
25 **fim**
26 **até** t $<$ Z;
27 *Fim*;