HECTOR PÉREZ BARANDA

CYTLFGRAPH: UMA EXTENSÃO DO CYTOSCAPE PARA REDES REGULADORAS DE GENES COM LÓGICA LIMIAR

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

Orientador: Ricardo dos S. Ferreira

VIÇOSA – MINAS GERAIS
2020

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
P438c
2020

Perez Baranda, Hector, 1988-
CyTLFGRAPH : uma extensão do Cytoscape para redes reguladoras de genes com lógica limiar / Hector Perez Baranda. – Viçosa, MG, 2020.
75 f. : il. (algumas color.) ; 29 cm.

Orientador: Ricardo dos Santos Ferreira.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.69-75.

1. Redes regulatórias de genes. 2. Multiprocessamento. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22 ed. 004.35

HECTOR PÉREZ BARANDA

CYTLFGRAPH: UMA EXTENSÃO DO CYTOSCAPE PARA REDES REGULADORAS DE GENES COM LÓGICA LIMIAR

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 19 de fevereiro de 2020.

Assentimento:

Hector Pérez Baranda
Autor

Ricardo dos Santos Ferreira
Orientador

À minha companheira Lídia(Lili) que faz com que meu mundo siga em movimento, junto à minha filha Lia que está vindo.

À minha família e aos meus amigos por me fazerem acreditar que sou capaz.

Agradecimentos

Agradeço primeiramente aos meus pais, pelo apoio e incentivo incondicional em todos os momentos da minha vida, e pela força que sempre deram durante este tempo.

A Lidia Raquel de Mesquita Vasconcelos, minha esposa, minha companheira, ofereço um agradecimento especial por ter me dado todo o apoio que necessitava, todo carinho, respeito e amor , e por ter me dado a maior oportunidade de um homem, o de ser pai.

Aos meus amigos Vinicius Sperandio, Fabio Reinoso, Fernando Passe, Laura Perez e Juan Niño e a todos os que conheci nesta viagem, por toda a confiança e ajuda ao longo deste longo caminho.

Aos professores, que se dedicaram tanto para passar o conhecimento formar uma base sólida para cada um de nós

Agradeço muito ao meu orientador Ricardo Santos Ferreira, por sempre me auxiliar na solução dos problemas e por acreditar que seríamos capazes de realizar esta pesquisa.

Agradeço ao coordenador do curso Alcione Paiva pela ajuda e gestões realizada em todo meu processo.

Agradeço a FAPEMIG com o apoio através do projeto "Aceleradores para Redes Reguladores de Genes" - Modalidade: "Edital 001/2018 - Demanda Universal", Processo N. : APQ-01203-18.

Agradeço que o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Agradeço a Universidade Federal de Viçosa e ao Departamento de Informática por fornecerem o suporte necessário para adquirir o título de mestre.

E finalmente, a todos aqueles que, direta ou indiretamente, contribuíram para a execução deste trabalho, os meus sinceros agradecimentos.

"Transportai um punhado de terra todos os dias e fareis uma montanha."
(Confúcio)

Resumo

BARANDA, Hector Pérez, M.Sc., Universidade Federal de Viçosa, fevereiro de 2020. **CyTLFGraph: Uma Extensão do Cytoscape para Redes Reguladoras de Genes com lógica limiar** Orientador: Ricardo dos Santos Ferreira.

Modelos dinâmicos de redes de reguladoras de genes (GRNs) vem sendo utilizados na descrição de fenômenos e processos biológicos complexos, como diferenciação celular e evolução de doenças como o câncer. No entanto, a caracterização topológica e funcional de GRNs reais ainda é muitas vezes parcial. Além disso, o espaço de solução gerado por uma GRN com n vértices (ou genes) é 2^n, onde a busca exaustiva pode não ser viável. O objetivo deste trabalho é aplicar a computação de alto desempenho para explorar o máximo possível do espaço de busca. A característica distintiva deste trabalho em comparação com os esforços anteriores é que ele fornece suporte para modelar o comportamento dinâmico de subsistemas biológicos bem definidos usando várias estratégias de simulação: iterações síncronas ou modelagem híbrida, além de melhorar os tempos de execução, usando as representações booleanas e com lógica limiar das redes reguladores. As ferramentas desenvolvidas também foram incluídas como extensões da ferramenta Cytospace que é uma referência na modelagem de redes e grafos com redes complexas na área de Bioinformática.

Palavras-chave: Redes reguladoras de genes. Computação alto desempenho. Cytoscape.

Abstract

BARANDA, Hector Pérez, M.Sc., Universidade Federal de Viçosa, February, 2020. **CyTLFGraph: A cytoscape application for genetic network analysis** Adviser: Ricardo dos Santos Ferreira.

Dynamical models of genetic regulatory networks are effective in describing complex biological processes and phenomena such as cell differentiation and cancer development. However, the topological and functional characterization of real GRNs is still often partial and exhaustive searching may not be available.The purpose of this work is to apply high-performance computing to expand search space while minimizing time. The distinctive feature of this work compared to previous efforts it's that provides support for modeling the dynamic behavior of well-defined biological subsystems using various simulation strategies like synchronous iterations or hybrid modeling, as well as improving execution times using a threshold representation of the networks using boolean and threshold representation.The tools developed were also included like plugins to the Cytoscape tool, which is a reference in modeling networks and graphs with complex structure in the area of Bioinformatics.

Keywords: Genetic regulatory networks. High-performance computing. Cytoscape

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

INTRODUÇÃO

Na era *big data*, há uma necessidade crescente de modelos matemáticos e computacionais voltados para análise do grande volume de dados [Kitano, 2002]. Considerando os sistemas biológicos, existem várias abordagens que visam inferir, a partir de dados, modelos para um fenômeno de interesse, explorando o espaço com vários modelos possíveis com o objetivo de identificar as características comuns, propriedades e regularidades que sejam consistentes com os dados [Paroni et al., 2016]. A falta de dados confiáveis, muitas vezes, dificulta a inferência de um modelo exato, onde uma análise estatística de conjuntos de modelos pode fornecer aspectos fundamentais sobre o fenômeno de referência, sua origem ou a história evolutiva[Kaneko, 2006].

Uma rede reguladora de genes (*Gene Regulatory Network* - GRN) é uma representação da dinâmica dos sistemas biológicos, como os processos de evolução e divisão celular, fabricação de medicamentos, estudo de câncer, dentre outros. A rede pode ser representada como um grafo onde os nós identificam os genes cujos níveis de expressão são regulados pela ação (ativadora ou supressora) e suas conexões. As interações são representadas pelas arestas que conectam os nós do grafo e por funções booleanas embarcadas nos nós [Carballido et al., 2009]. Um grafo booleano pode modelar uma GRN [Kauffman, 1993, 1969] sendo que:

- Os genes, as entradas e as saídas da GRN são representados como nós;
- Cada nó tem dois estados: ativo ou inativo;
- Um gene recebe os estados dos outros genes através das arestas do grafo. Localmente, o gene tem uma função booleana para avaliar qual será seu novo estado a cada passo de simulação;

- O comportamento da rede é dinâmico e evolui com o tempo. A cada passo, o novo estado de um nó é calculado em função dos estados anteriores dos nós adjacentes.

Dentro deste contexto, envolvendo dados genômicos, os modelos booleanos de GRNs comprovam propriedades de sistemas reais. Através de simulações pode-se gerar previsões qualitativas ou quantitativas sobre o comportamento geral do sistema que pode ser comprovados com experimentos no laboratórios [Shmulevich et al., 2002], onde podemos incluir a simulação de redes regulatórias modeladas por grafos booleanos [Chaos et al., 2006; Albert & Othmer, 2003].

Como já mencionado, os nós da GRN representam os elementos de um sistema, as arestas representam relações regulatórias entre elementos e cada nó é caracterizado por um estado ativo (1) ou inativo (0) [Thomas, 1973]. Assim, uma rede com N nós terá 2^N estados possíveis. Uma mudança no estado de um determinado nó geralmente desencadeia mudanças nos estados dos nós regulados por ele (os nós para os quais ele aponta). Desta forma, o estado da rede passa por uma trajetória dinâmica.

No modelo booleano mais simples assume-se que todas as funções de ativação dos nós são atualizadas ao mesmo tempo, que é chamado de método **síncrono** de atualização [Ruz et al., 2014]. Outra possibilidade é atualizar um ou mais nós por vez em qualquer ordem. Esta abordagem de operação denomina-se atualização **assíncrona**, sendo o mais parecido ao que acontece na vida real [Saadatpour et al., 2010].

1.1 O problema

Desenvolver ferramentas para dar suporte ao estudo das estruturas das GRNs para diferentes organismos auxilia na compreensão do funcionamento biológico dos seres vivos [Trinh et al., 2014a; Carballido et al., 2009], além que possuir muitas aplicações nas áreas da medicina e biotecnologia, no desenvolvimento de medicamentos e desenho de novas variantes de cultivos respetivamente [Huang, 2002; Yamaguchi-Shinozaki & Shinozaki, 2006]. Existem na literatura vários trabalhos que propõem ferramentas de estudo da estabilidade e a resiliência dos GRNs, com representação booleana [Morris et al., 2010; Franke et al., 2010; Trinh et al., 2014a; Truong et al., 2016].

O maior desafio dos modelos booleanos é a complexidade exponencial do espaço de soluções[Kerkhofs & Geris, 2015]. Uma implementação precisa ser executada de

maneira eficiente e extensível para permitir a integração com conhecimento empírico, bem como a exploração dos comportamentos dinâmicos. Portanto, o uso de novas técnicas para análise e representação de GRNs que permitam a busca eficiente e exploratória do maior conjunto de estados da rede sempre está em constante evolução.

Em termos de ferramentas de software na área biológica, especificamente para redes reguladores, podemos destacar o Cytoscape [Shannon et al., 2003; Consortium, 2019] que é um ambiente de software livre que permite agregar várias funcionalidades na modelagem de redes para interação biomolecular. Vale destacar que o artigo "Cytoscape: a software environment for integrated models of biomolecular interaction networks" [Shannon et al., 2003],atualmente, tem mais de 15913 citações [1]. O Cytoscape pode receber colaborações e foi bem aceito pela comunidade. Possui diversas funcionalidades de redes complexas como,por exemplo, cálculo de centralidade de grafos, e coeficientes de agrupamento. Porém, não agrega funcionalidades sobre o comportamento dinâmica das redes reguladoras.

As redes reguladoras são modelos que ainda estão sendo ajustados pelos biólogos. O uso de ferramentas computacionais vem acelerando o processo com as simulações. Novas técnicas e ferramentas devem ser desenvolvidas para o problema de redes reguladoras também poderão ser aplicadas a outros problemas. O uso de computação paralela, nuvem, múltiplos núcleos, GPUs e FPGAs podem também reduzir o tempo de execução e aumentar o espaço a ser investigado. Entretanto, existe uma grande distância entre as ferramentas mais populares como o Cytoscape e as soluções com paralelismo. Por exemplo, os FPGAS e as linguagens de Hardware exigem especialistas em Hardware, as GPUs não são triviais para ser programadas e o uso de computação na nuvem ainda está amadurecendo. Este projeto busca reduzir esta distância oferecendo abstrações que podem ser exploradas por programadores de software em alto nível.

1.2 Objetivos

Como objetivo, o presente trabalho visa fornecer módulos de ferramentas que permitam avaliar e caracterizar as Redes Reguladoras de Genes, incluindo a representação como lógica limiar, reduzir o tempo de execução para uma melhor exploração do espaço de busca dos estados de atração. Especificamente, pretende-se:

- Investigar outros modelos de representação de grafos booleanos.

[1] Medida aferida 10 Dezembro de 2019.

- Propor um modelo de representação e análise de grafos que permita o processamento dos estados de forma paralela e dinâmica sem necessidade de compilação para ajuste no modelo.
- Integração do software/hardware gerado com Software de ampla utilização pelos biólogos como o CytoScape.

1.3 Estrutura da Dissertação

A ordem lógica do trabalho está estruturada em 4 Capítulos:

Capitulo 1: apresenta a situação atual das análises de Redes Reguladoras de Gene, incluindo definições e caraterísticas. Apresenta também os principais problemas para o desenvolvimento do estudo das GRN, além dos objetivos visados para a realização deste trabalho.

Capitulo 2: apresenta o Referencial Teórico, onde dissertamos sobre os fundamentos das redes reguladoras, as ferramentas de software que são referência na área para a representação e as análises das redes reguladoras usadas pelos biólogos. Uma introdução das plataformas e das técnicas utilizadas para o desenvolvimento do trabalho juntamente com uma análise dos trabalhos relacionados ao estudo das redes biológicas também são apresentados.

Capitulo 3: detalha as ações realizadas para a solução proposta, apresentando a técnica de transformação das equações booleanas em equações de lógica limiar, o processo encontrar os atratores, as características e plataformas usadas para a construção da aplicação.

Capitulo 4: os principais resultados e experimentos estão relatados onde se faz uma avaliação da solução proposta. Seis redes reguladoras com caraterísticas distintas são avaliadas.

Capitulo 5: detalha-se em resumo os resultados obtidos durante a realização da pesquisa, além do grupo de funcionalidades a mais que se planejaram acrescentar à solução que se propor.

Capítulo 2

REFERENCIAL TEÓRICO

Este capítulo apresenta os fundamentos das redes reguladores de genes e o estado da arte das ferramentas. Começamos com um exemplo de rede reguladora e o seu diagrama de estados, as funções de atualização, os atratores e as bacias de atração na Seção 2.1. Mostramos a representação por funções limiares e como é feita a transformação de função booleana em uma função limiar na Seção 2.2. Posteriormente, apresentamos as ferramentas que são referência na área para a representação e o análises das redes reguladoras na Seção 2.3. Depois descrevemos brevemente as plataformas de hardware e as ferramentas de modelagem com paralelismo para obter alto desempenho nas Seções 2.4 e 2.6. Finalizamos com os trabalhos relacionados ao estudo e análise da representação das redes reguladoras de genes na Seção 2.7.

2.1 Redes Reguladores de Genes

Um dos primeiros modelos dinâmicos para analisar redes reguladoras foi proposto por [Kauffman, 1969] e é conhecido como Redes Booleanas Aleatórias. Posteriormente, vários modelos foram derivados como o modelo baseados em funções limiares ao invés de funções Booleanas [Darabos et al., 2011], ao estender o modelo proposto em [Li et al., 2004]. Todos os modelos são baseados em grafos de genes onde a influência de um gene sobre outro é descrito pelas arestas e pelas funções de atualização. O comportamento da rede é analisado utilizando o todo o espaço de estados ou um sub-espaço dos estados do grafo.

A seguir iremos apresentar os principais conceitos dessas redes: modelo de atualização, dinâmica das redes reguladoras, os atratores e suas bacias de atração.

2.1.1 Modelo de Atualização

Segundo [Garg et al., 2008] as funções de atualização podem ser classificadas em dois modelos:

- **Modelo de atualização síncrono:** Nesse modelo todos os genes de uma rede alteram seus estados simultaneamente. O novo estado é calculado usando os valores do estado anterior. Apesar dos sistemas biológicos não terem um comportamento síncrono, onde diferentes genes levam diferentes tempos para realizar a transição de um estado para outro, o modelo síncrono é uma boa aproximação com potencial para explorar paralelismo em aceleradores como FPGA e GPUs [Manica et al., 2019; Ferreira & Vendramini, 2010; Campos et al., 2011; Jacob et al., 2012; da Silva et al., 2017; Rosa et al., 2019].

- **Modelo de atualização assíncrono:** Nesse modelo cada gene pode levar um tempo diferente para realizar a transição de um de estado para outro. Esse modelo está mais próximo dos sistemas biológicos. Porém podem demorar mais a convergir para validação de algumas propriedades pois requer várias repetições das simulações para ser ter uma amostragem relevante. Apesar da simulação de uma rede ser sequencial, existe paralelismo que pode ser explorado com aceleradores, fazendo várias simulações simultâneas de diferentes cópias das redes em GPUs ou FPGAs.

O modelo booleano aleatório de Kauffman [Kauffman, 1969] é muito atraente devido à sua simplicidade. Basicamente, consiste em gerar funções aleatórias de atualização para cada gene. Já o modelo com funções limiares pode ser mais fácil para ajustar as funções locais de cada gene, usando uma função global comum para toda a rede como demonstrado em [Darabos et al., 2011; Li et al., 2004].

Uma rede de genes pode ter um comportamento caótico ou ordenado. O limite de uma transição de fase entre os comportamento pode ser ajustado alterando as conexões e as funções de atualização [Aldana, 2003; Ferreira & Vendramini, 2010; Campos et al., 2011]. O sistema deve ser tolerante para suportar pequenas pertubações nos genes sem alterar seu comportamento e ao mesmo tempo ser adaptável para caso receba algumas pertubações mude drasticamente de comportamento se adaptando ao novo cenário.

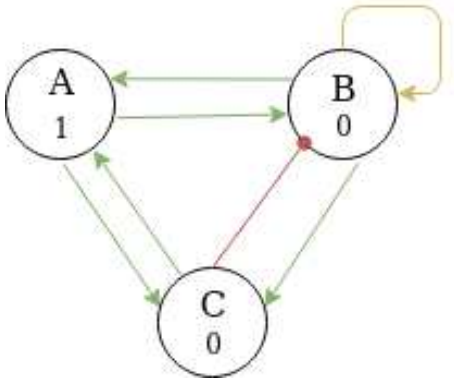


Figura 2.1. Rede reguladora de 3 nós com seus respetiva interações

2.1.2 Função de atualização

O estado de uma rede reguladora é definido pelo conjunto de estados dos genes, ou seja, a configuração que os genes se encontram em um determinado tempo. A Figura 2.1 exibe uma rede composta de apenas 3 genes com suas respectivas interações. Dentro de cada um dos vértices A, B e C está um valor booleano.Neste exemplo, B e C estão inativos (valor 0) e A está ativo (valor 1). Dado um estado, a cada passo de simulação, todas as funções são reavaliadas e a rede muda para próximo estado. A Tabela 2.1 mostra cinco transições de estado da rede reguladora utilizando as seguintes funções booleanas para cada vértice:

$$\begin{aligned} f(A_{i+1}) &= B_i \mid C_i \\ f(B_{i+1}) &= A_i \;\&\; (!\,B_i \;\&\; !\,C_i \mid !\,C_i \;\&\; B_i) \\ f(C_{i+1}) &= A_i \mid B_i \end{aligned} \tag{1}$$

Tabela 2.1. Transição de estados. Abordagem booleana

	Arestas		
tempo	A	B	C
1	0	0	1
2	1	0	0
3	0	1	1
4	1	0	1
5	1	0	1

A Figura 2.1 mostra as arestas ativadoras com a cor verde e as arestas inibidores estão na cor vermelha. Pode-se ver como a rede estava no estado 100 (ABC) e

passou para o estado 011 que também está ilustrado na Tabela 2.1 na transição de $t = 2$ para $t = 3$. Uma rede com n vértice terá 2^n possíveis estados, neste exemplo o numero de estados seria de 8.

No modelo proposto por [Kauffman, 1969] a função de atualização é randômica para cada gene da rede e tem um comportamento síncrono, ou seja, todos os genes são atualizados ao mesmo tempo. No modelo de [Li et al., 2004] a função de atualização é compartilhada por todos os genes e tanto os fatores de ativação e repressão têm o mesmo peso. Este modelo é explicado segundo a equação 2, onde o estado de um gene na próxima interação, $(S_i(t+1))$, será ativo se obtiver a maioria dos genes de entrada ativos, inativo se tiver a maioria dos genes de entrada inativos ou o estado do gene permanecerá inalterado caso o número de ativações e inibições sejam iguais, na equação, a_{ij} vai tomar o valor de 1 para as arestas de ativação e -1 para as arestas de inibição.

$$S_i(t+1) = \begin{cases} 1, & \sum_{j=1}^{n} a_{i,j} S_j(t) > 0 \\ 0, & \sum_{j=1}^{n} a_{i,j} S_j(t) < 0 \\ S_i(t), & \sum_{j=1}^{n} a_{i,j} S_j(t) = 0 \end{cases} \tag{2}$$

Baseados no modelo de [Li et al., 2004], o grafo da rede reguladora apresentado na Figura 2.1 terá uma nova atualização dos estados dos nós. Neste modelo, as arestas verdes representam a interação de ativação que o nó j exerce sobre o nó i e as arestas vermelhas representa a ação de inibição, repressão ou degradação, sobre uma proteína. O modelo de [Li et al., 2004] só representa um subconjunto do espaço pois não temos controle dos pesos nas arestas. Neste exemplo, a função de [Li et al., 2004] gera uma rede reguladora com outro comportamento.

Baseado na rede reguladora do exemplo anterior as equações de cada nó no modelo de [Li et al., 2004] são:

$$\begin{aligned} f(A_{i+1}) &= B_i \ + \ C_i \\ f(B_{i+1}) &= A_i \ - \ (B_i \ + \ C_i) \\ f(C_{i+1}) &= A_i \ + \ B_i \end{aligned} \tag{3}$$

Um exemplo de transição dos estados dessa rede reguladora está representado na Tabela 2.2, começando por um estado inicial, onde as proteínas A e B estão ativas, até encontrar um estado (ou um conjunto) estático que é denominado atrator, para

este exemplo, alcançado no tempo $t = 2$ e se repetindo no $t = 3$.

	Arestas		
tempo	A	B	C
1	1	1	0
2	1	1	1
3	1	1	1

Tabela 2.2. Transição de estados. Abordagem de Li

Na Tabela podemos observar quando o nó C é ativado no tempo $t = 2$, já que a somatória dos nós ativadores ($B = 1, A = 1$) é maior que a dos nós inibidores.

Já no modelo proposto em[Darabos et al., 2011] cada gene tem um peso associado a ligação. A ativação ou a inibição podem ter pesos diferentes. A equação de um gene pode requerer que mais que a maioria de entradas estejam ativas ou inativas para alterar seu estado. O modelo tem um limiar T_i para o gene i, que deve ser atingido para que o gene se torne ativo, sendo w_j o peso correspondente ao nó j e S_j^+ (S_j^-) o estado de um ativador (inibidor), respetivamente. A equação 4 mostram a definição da função de ativação para este modelo para o qual usaremos o termo modelo limiar.

$$S_i(t+1) = \begin{cases} 1, & \sum_{j=1}^{n} w_j S_j^+ > T_i * \left(\sum_{j_1}^{n} w_j S_j^+ + \sum_{j_1}^{n} w_j S_j^- \right) \\ 0, & \sum_{j=1}^{n} w_j S_j^+ < T_i * \left(\sum_{j_1}^{n} w_j S_j^+ + \sum_{j_1}^{n} w_j S_j^- \right) \\ S(t), & \textit{caso contrário} \end{cases} \tag{4}$$

Na ausência de valores específicos reais de T, ou w no sistema, assumimos um valor limiar comum, $T = 0,5$ para todos os genes e um peso idêntico $w = 1$ para todas as conexões, então usando como referencia a rede que já estamos estudando ficariam as equações para cada nó da seguinte forma:

$$\begin{aligned} f(A_{i+1}) &= B_i + C_i > 0.5 * (B_i + C_i) \\ f(B_{i+1}) &= A_i > 0.5 * (B_i + C_i + A_i) \\ f(C_{i+1}) &= A_i + B_i > 0.5 * (A_i + B_i) \end{aligned} \tag{5}$$

Na Tabela 2.3 mostra-se um exemplo de transição dos estados, usando como referencia a rede regulatória da Figura 2.1 e começando com os nós A e B inativos chegando ao mesmo estado estático das funções booleanas.

	Arestas		
tempo	A	B	C
1	0	0	1
2	1	0	0
3	0	1	1
4	1	0	1
5	1	0	1

Tabela 2.3. Transição de estados. Abordagem Darabos

2.1.3 Atratores e Bacias de Atratores

Investigar outros modelos de representação de grafos booleanos.o de estados, caracterizado por um comprimento de ciclo. Os estados estacionários e/ou os ciclos são denominados atratores.

Segundo [Kauffman, 1969], os atratores que são pequenos e estáveis as perturbações, são os de maior importância biológica. Tendo em conta que os diferentes atratores, que representa a dinâmica das redes a partir dos variados estados iniciais, podem-se ver como tipos de células ou formas de desenvolvimento de um organismo unicelular, enquanto a trajetória que levam a estos atratores pode ser vista como os caminhos de diferenciação [Darabos et al., 2011]. Diferentes funções de atualizações podem levar a atratores diferentes.

Qualquer modelo booleano tem um ou vários atratores. Cada atrator está associado a um conjunto de estados que chegam nele. Este conjunto é chamado de sua bacia de atração e o número de estados desse conjunto é definido como o tamanho da bacia de atração.

Para maior entendimento dos conceitos de atrator e bacia de atração, ilustramos o diagrama de estados, na forma de grafo, na Figura 2.2, correspondente com a rede reguladora apresentada na Figura 2.1 e considerado a função de atualização as equações booleanas definidas na equação 1.

Primeiro, apesar de ambos serem representados na forma de grafo, a Figura 2.1 mostra uma rede com três vértices onde cada vértice ou nó da rede pode estar no estado 0 ou 1. Já a Figura 2.2 mostra um diagrama de estados da rede da Figura 2.1. Cada vértice é um estado onde neste exemplo, cada estado (ou vértice E_i) é representando pelos estados dos 3 genes. O estado dos genes A, B e C estão representados dentro de cada vértice do grafo de estados. Por exemplo, o estado E_2 que tem os valores 010 para os seus genes, significa que o gene A e C estão inativos e que o gene B está ativo. Como a rede tem 3 genes, o grafo de estados terá $2^3 = 8$ estados (ou vértices).

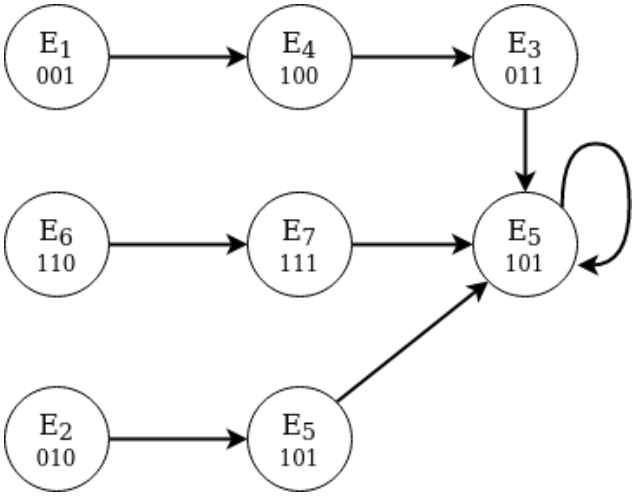


Figura 2.2. Exemplo de atratores e bacias de atração. Abordagem Booleana

A Figura 2.2 mostra o diagrama de estados da rede com equações booleanas que tem apenas um atrator. O atrator tem tamanho 1 e é composto pelo estado E_5. A bacia de atração tem tamanho 7 e é composta pelos estados E_1, E_4, E_3, E_6, E_7 e E_2.

A Figura 2.3 como o grafo de estados assim como os atratores para rede reguladora com a equação 2 do modelo de [Li et al., 2004]. O comportamento gerado é um pouco diferente e podemos observar tanto no diagrama de estados como nos atratores gerados, sendo estes 2 de tamanho 1, E_1 e E_7 respetivamente.

Já usando a proposta de [Darabos et al., 2011], apresentada na equação 4, o comportamento é o mesmo do modelo booleano, mostrando que é possível substituí-lo pelo modelo de limiar.

2.2 Função Limiar

As funções booleanas ficaram populares graças ao pesquisador Claude Shannon [Shannon, 1938] com os circuitos chaveados que evoluíram para as portas lógicas (AND, OR e NOT) na década de 60 como primitivas para descrição de circuitos digitais. As portas AND/OR são casos especiais da porta limiar, que são baseadas no princípio de decisão majoritária ou de limite, o que significa que o valor de saída depende se a soma aritmética dos valores de suas entradas exceder um limite [Mozaffari et al., 2017]. A Figura 2.4 mostra uma porta genérica com n entradas $x_1, \ldots, x_n$. Cada entrada tem um peso associado w_i e a função será modelada pela equação $x_1w_1 + \ldots + x_nw_n > T$. Uma função booleana pode ser transformada em

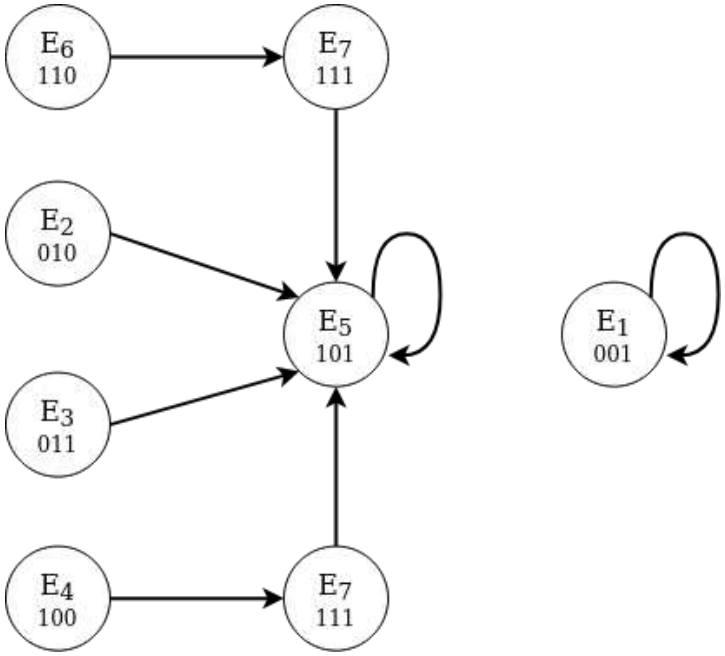


Figura 2.3. Exemplo de atratores e bacias de atração. Abordagem de Li.

uma Função Lógica Limiar ou *Threshold Logic Function* (TLF) em inglês.

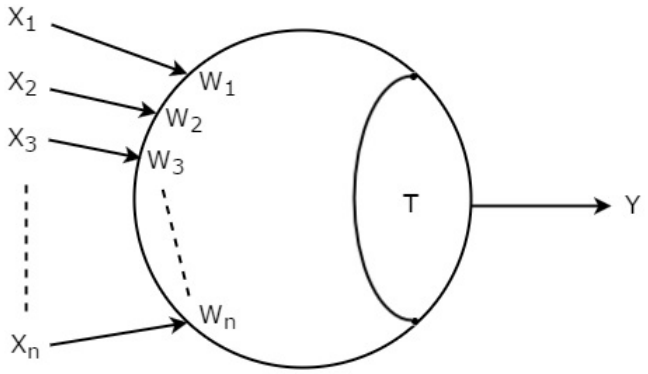


Figura 2.4. Porta threshold. Tem n entradas $[x_1...x_n]$ para cada entrada e uma saida y. Formada por duas partes; os pesos $[w_1 \ ... \ w_n]$ e o peso limite T.

Uma função lógica limiar booleana (TLF) requer um peso, um por variável de entrada. A TLF é completamente representado por um vetor compacto $[w_1, w_2, ..., w_n; T]$. A TLF pode representar uma função booleana complexa, como $f = x_1x_2 \vee x_1x_3 \vee x_2x_3x_4 \vee x_2x_3x_5$ que corresponde com $f = [4, 3, 3, 1, 1; 7]$, que representa a equação $4x_1 + 3x_2 + 3x_3 + x_4 + x_5 \geq 7$. A função é 1 somente quando a soma da maior que (ou igual) que um valor de peso chamado peso limite [Muroga, 1971; Neutzling et al., 2013].

A TLF é formalmente definida como:

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & se \sum_{i=1}^{n} w_i * x_i \geq T \\ 0 & caso\ contrário \end{cases} \quad (5)$$

Para a transformação de uma equação booleana em lógica liminar precisa-se achar os pesos e o peso limiar. Para isso, pode-se usar os parâmetros de Chow[Muroga, 1971; Neutzling et al., 2013] que são um conjunto de parâmetros utilizados para definir a relação entre os pesos de uma TLF. Dada uma função $f(x_1, x_2, ..., x_n)$, o parâmetro Chow de uma variável é definido pelo dobro da diferença do número de entradas para as quais $x_i = 1$ e $f(x_i) = 1$, e o número de entradas para as quais $x_i = 0$ e $f(x_i) = 1$ [Muroga, 1971]. A expressão matemática é:

$$w_i = 2 * (count(x_i) - count(\overline{x_i})), \quad quando : f(x_i) = 1 \quad (6)$$

Onde $count(x_i)$ conta o número de vezes que a função tem o valor 1 quando $x_i = 1$ e $count(\overline{x_i}$ contará o número de vezes que a função tem o valor 1 quando $x_i = 0$. Mas ainda falta calcular qual será o valor do limiar T. Para este cálculo temos que considerar que podem existir equações booleanas não unitárias, estas têm que ser transformadas para se converter em funções limiares, [Neutzling et al., 2013; Muroga, 1971], um exemplo de simples de uma função booleana não unitária seria $f(x, y) = x\overline{y} + \overline{x}y$, o que resultaria na tentativa de calculo dos $w_i = 0$ e não se cumpririam as inequações. Para estas funções é necessário decompor as funções em mais de um nível.

Como explicamos, existe uma relação entre os pesos das entradas e o peso limite da TLF. Esta relação pode-se expressar com um conjunto de inequações, ilustrados na Tabela2.4.

Para um exemplo de função booleana $f = \overline{x_3} \cdot (x_1 \vee x_2)$ usando a equação (6) temos que a função f é verdadeira nas linhas 2, 4 e 6. Nestas linhas a variável x_1 é verdadeira 2 vezes (linhas 4 e 6) e falsa uma vez (linha 2), portanto $W_1 = 2 * (count(x_1) - count(\overline{x_1})) = 2 * (2 - 1) = 2$, sendo que a função $count(x_i)$ irá contar quantas vezes a variável x_i é verdadeira quando a função f é verdadeira e a função $count(\overline{x_i})$ irá contar quando a variável é falsa e a função seja verdadera, fazendo esse calculo para cada entrada finalmente temos que os valores dos pesos seria 2, 2 e -6, respetivamente.

As duas últimas linhas da Tabela 2.4 mostram as ocorrências de x_i verdadeiro e falso quando f é verdadeira.Também a ultima coluna da representa a relação que

	Entrada			Saída	Relação entre
linha	x_1	x_2	x_3	y	W e T
0	0	0	0	0	0 <T
1	0	0	1	0	W_3 <T
2	0	1	0	1	$W_2 \geq$ T
3	0	1	1	0	$W_2 + W_3$ <T
4	1	0	0	1	$W_1 \geq$ T
5	1	0	1	0	$W_1 + W_3$ <T
6	1	1	0	1	$W_1 + W_2 \geq$ T
7	1	1	1	0	$W_1 + W_2 + W_3$ <T
count(x_i)	2	2	0		
count($\overline{x_i}$)	1	1	3		

Tabela 2.4. Tabela verdade da Função $f = \overline{x_3} \cdot (x_1 \vee x_2)$ e pesos da Função Limiar.

tem os pesos (W) e o limite (T) da função limiar.

O valor de T é obtido solucionando as inequações. Neste exemplo podemos usar apenas as linhas 2, 4 e 6 onde a função f é verdadeira. Para achar o valor do limiar calculamos o mínimo do somatório dos pesos W_n no lado esquerdo das inequações em cada uma destas linhas, definido por:

$$T = \min_{j=1}^{linhas} (\sum_{i=1}^{n} W_i), \quad \forall \quad j \quad \text{onde} \quad f(x) = 1 \tag{7}$$

Para a função $f(x_1, x_2, ..., x_n)$ com as três condições e os valores de $W_2 = 2$, $W_1 = 2$ e $W_1 + W_2 = 4$, o valor mínimo é 2, ou seja $T = 2$, então a função limiar será $f = [2, 2, -6; 2]$. Numa simples verificação usando a linha 4, (pode-se usar qualquer outra linha), temos $2x_1 + 2x_2 - 6x_3 = 2 * 1 + 2 * 0 - 6 * 0 = 2 \geq 2$, portanto $f(4 = 100_2) = 1$.

2.3 Cytoscape

Cytoscape é uma plataforma de código aberto para visualização de redes de interação molecular e redes biológicas em geral. Pode ser usado para integrar redes com perfis de expressão genética e outros dados de estados. Cytoscape foi originalmente desenvolvido para pesquisas biológicas, mas com a sua rápida popularização, converteu-se em uma plataforma para análise e visualização de redes complexas, o núcleo do Cytoscape tem várias ferramentas, além de possibilitar extensão com recursos adicionais em forma de aplicativos, ou *plugins* [Shannon et al., 2003].

Os aplicativos de Cytoscape para análises de perfis molecular e de redes tem suporte adicional para vários formatos, novos layouts, utilização de scripts, conexão a bancos de dados e outras. A base de API Cytoscape foi desenvolvida em Java, tem boa documentação e muitas extensões já foram propostas. Um exemplo é o CyREST, uma extensão que expõe estruturas de dados relacionadas à rede Cytoscape e publica as funcionalidades como microsserviços que pode ser chamado por meio de um protocolo *REST*[1] por ferramentas e outras linguagems [Ono et al., 2015]. Um exemplo da interface gráfica e da representação de redes numa aplicação Cytospcape está ilustrado na Figura 2.5.

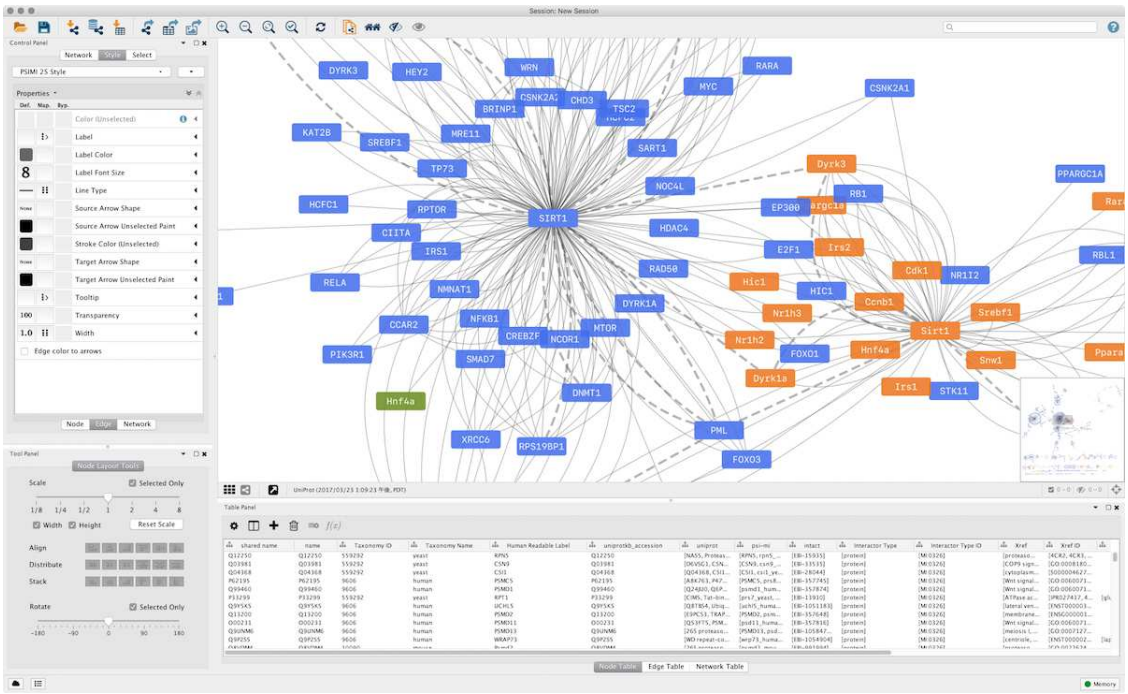


Figura 2.5. Representação de uma Rede de Reguladora de Gene no Cytoscape.

O desenho estrutural da versão 3 do Cytoscape teve como objetivos a escalabilidade, o desempenho e a estabilidade. A padronização vem garantindo a estabilidade das aplicações (extensões), a API e sua modularidade com modelo OSGI, melhorando o desempenho e usabilidade em comparação com a versão 2.X [Consortium, 2019].

O framework OSGI é um conjunto de especificações que definem um sistema de componentes dinâmicos para Java. Essas especificações permitem um modelo de desenvolvimento em que o aplicativo é composto por vários componentes que

[1]Do inglês *representational state transfer* é um estilo de arquitetura de *software* para sistemas hipermídias distribuídos pela *World Wide Web*.

são encapsulados em um pacote (Bundle) [Technology, 2019]. As aplicações ou componentes que vêm em forma de pacotes podem ser instaladas remotamente, iniciadas, paradas, atualizadas e desinstaladas, sem precisar reiniciar.

Por sua vez o pacote é composto por um "JAR" (Java ARchive) com metadados extras, um módulo de Importação usado pelo componente e um módulo exportação que permite a comunicação com outros componentes. Além disso, um Ativador que é lançado toda vez que o Bundle é iniciado ou parado. A Figura 2.6 mostra um diagrama estrutural da arquitetura [Consortium, 2019].

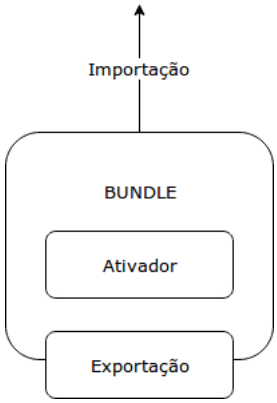


Figura 2.6. Com é composto a arquitetura das aplicações Cytoscape.

As aplicações de Cytoscape 3 são serviços orientado a microkernel [Shannon et al., 2003], que tem como base o framework OSGI e todas os subsistemas estão separados por Bundles de OSGI, permitindo a carga e desativação desses pacotes de forma dinâmica, como já tinha-se explicado anteriormente, formando a seguinte estrutura como mostra a Figura 2.7 .

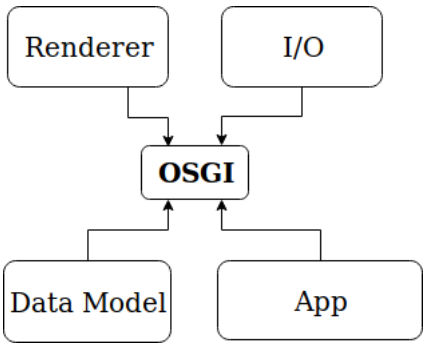


Figura 2.7. Como é composto a arquitetura do Cytoscape.

Exemplo de aplicações que seguem este modelo encontramos a **clusterMaker** [Morris et al., 2011], ferramenta que fornece vários algoritmos e visualizações de

cluster que podem ser usados independentemente ou em combinação para análise de conjuntos de dados biológicos e a **GeneMANIA** [Montojo et al., 2010], que um aplicativo que permite que os usuários construam uma rede composta de interação funcional gene-gene a partir de uma lista de genes, ambas são disponibilizadas pela comunidade de desenvolvedores de Cytoscape. A maioria das aplicações podem ser descarregadas gratuitamente na loja de Aplicações de Cytoscape.

2.4 OpenMP

O OpenMP é uma interface de programação de aplicações (API), para a programação paralela com memória compartilhada em múltiplas plataformas. Permite a adição de concorrência em programas escritos em C, C++ e Fortran com threads de forma transparente para o programador[Board, 2019].

O OpenMP é um modelo de programação portável e escalável que proporciona ao programador uma interface simples e flexível para o desenvolvimento de aplicações paralelas. Fazendo uso de um conjunto de diretivas de compilador, rotinas de bibliotecas e variáveis de ambientes.

2.4.1 Sintaxe Básica

No código de exemplo 2.1, pode ser visto um exemplo de diretivas que são usadas pelo OpenMP para instruir o compilador.

```
#pragma omp <diretiva> [clausula [ , ...] ...]
```

Código 2.1. Diretiva de compilação usada no OpenMP

No exemplo básico de código 2.2 mostra-se como explorar o potencial de paralelismo em uma tarefa que percorre um array e atribui um valor calculado para cada posição, com a diretiva "parallel" indica-se que parte do código vai ser executado concorrentemente usando vários *threads*, além disso a diretiva for orienta ao compilador que cada iteração do loop seja executada num thread diferente, caso não seja usada esta diretiva o loop seria executada por cada thread. Na Figura 2.8 mostra o como foi processado o trecho de código pelo compilador [Board, 2019; Sato, 2002].

```
int *a = new int[LIMIT];
#pragma omp parallel
        {
                #pragma omp for
                for(i = 1;i <= LIMIT;i++)
                a[i] = 2 * i;
```

```
        }
print_vector(a)
```

Código 2.2. Diretiva de compilação usada no OpenMP

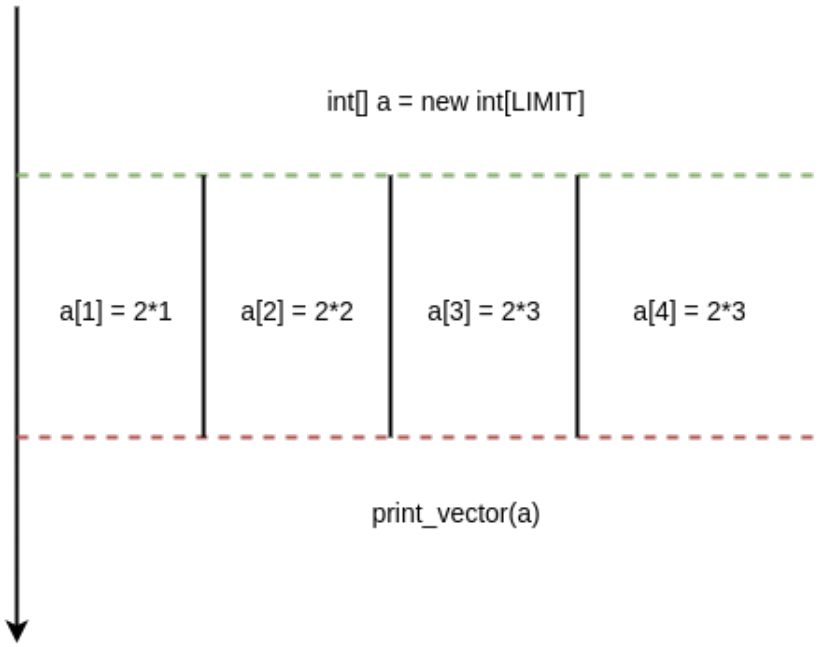


Figura 2.8. Diagrama de execução do código em OpenMP.

2.5 AVX

Desde o processador Pentium MMX, lançado em 1997, os processadores da linha X86 começaram a incorporar instruções vetoriais para manipulação de pacotes de 8, 16 ou 32 bits em unidades aritméticas de 128,256 e 512 bits. A extensão MMX, evoluiu para SSE e posteriormente para AVX. As extensões vetoriais avançadas (AVX) é formada por um conjunto de instruções de 128/256/512 bits desenvolvido pela *Intel* como uma extensão das instruções $x86$ usado em processadores *Intel* e AMD, que melhora o desempenho em novos aplicativos, e alguns existentes, manipulando pacotes de dados vetoriais maiores e usando mais *threads* e núcleos do processador.

As instruções do AVX processam os pacotes de dados vetoriais ao mesmo tempo, em vez de processá-los individualmente. Esses pacotes são chamados de vetores e os vetores AVX podem conter até 512 bits de dados. Os vetores AVX comuns contêm quatro *double* (4 x 64 bits = 256), oito $float$ (8 x 32 bits = 256) ou oito *int* (8 x 32 bits = 256).

O exemplo, a seguir, mostra como podemos utilizar as instruções AVX para executar cálculos em paralelo. Suponha que precisamos multiplicar e somar três vetores de float de 32 bits. Sem AVX, o código mais simples é:

```
produto_vetores(float* a, float* b, float* c, float* d) {

  for(int i=0; i<8; i++) {
    d[i] = a[i] * b[i] + c[i] ;
  }
}
```

Código 2.3. Multiplicaçõ de vetores sem AVX

Agora usando instruções AVX:

```
__m256 produto_vetore_avx(__m256 a, __m256 b, __m256 c, __m256 d) {

  d = _mm256_fmadd_ps(a, b, c);
}
```

Código 2.4. Multiplicação de vetores usando AVX

Neste exemplo a intrusão $__m256$ usa o tipo de dados vetor de 256 bits que encapsula 8 valores de 32 bits, realizado a operação A*B+C (multiplica e soma). Ou seja, o treco poderá ser executado 8 vezes mais rápido.

O exemplo anterior ilustra uma vetorização explícita onde o programador tem que fazer o trabalho de acoplar e usar as instruções. Porém, o compilador GCC pode fazer de forma transparente esse trabalho, só é preciso o uso dos sinalizadores de otimização ($flags$) $-O3$ ou $-ftree-vectorize$. O compilador pesquisará os possíveis laços a vetorizar. veja o exemplo no código 2.5. O código fonte permanece o mesmo, mas o código compilado pelo GCC é completamente diferente.

```
#pragma GCC optimize("O3","unroll-loops","omit-frame-pointer","inline")
#pragma GCC option("arch=native","tune=native","no-zero-upper")
#pragma GCC target("avx")
produto_vetores(float* a, float* b, float* c, float* d) {

  for(int i=0; i<8; i++) {
    d[i] = a[i] * b[i] + c[i];
  }
}
```

Código 2.5. Vetorização automatica pelo compilador GCC

Nesta dissertação apenas exploramos a geração automática do compilador GCC e verificamos se o compilador faz uso de instruções vetoriais AVX no código de máquina gerado.

2.6 CUDA

Ao longo do tempo o uso da GPU foi ampliado, começando com um acelerador gráfico, até se tornar uma ferramenta para processamento vetorial paralelo. Nos últimos anos, as GPUs têm sido combinadas com as CPUs para fornecer uma grande quantidade de poder de cálculo para a computação técnica, chamada computação heterogênea [Cheng et al., 2014; Coorporation, 2019]. O grande salto das GPUs para programação paralela ocorreu em torno de 2006/7 com a introdução da APIs CUDA e OpenCL [Cheng et al., 2014].

A sigla CUDA é uma abreviação para "Arquitetura Unificada de Dispositivos de Computação", que é uma plataforma de compilação que inclui, além do compilador, ferramentas de desenvolvimento criadas pela Nvidia que permite criar programas, numa extensão de C, que são executados nas GPUs da Nvidia. Apesar das abstrações criadas pelo modelo CUDA, existe uma necessidade de conhecer alguns detalhes das arquiteturas das GPUs, a fim de explorá-la bem. De uma forma geral, a interface de programação é simples em comparação, por exemplo, com o desenvolvimento para FPGA[Cheng et al., 2014; Ferreira & Vendramini, 2010; Romao et al., 2012; Campos et al., 2011; Jacob et al., 2012].

Em CUDA, os *threads* são organizados em dois níveis de hierarquia: grade (Grid) e blocos (Block). O grade é forma por um conjunto uni-dimensional ou bi-dimensional de blocos. A Figura 2.9 ilustra uma grade com 6 blocos com duas dimensões, sendo 3 na dimensão X e 2 na dimensão Y. Cada bloco por sua vez pode ter uma, duas ou três dimensões. No exemplo da Figura 2.9, o bloco tem duas dimensões 4x3 totalizando 12 threads. No total serão disparados para cada função (kernel) 6 blocos de 12 threadas totalizando 72 threads. Em CUDA, existem mecanismos explícitos de sincronização e troca de dados entre a CPU e a GPU [Gou & Gaydadjiev, 2013].

2.7 Trabalhos relacionados

Embora a relevância das redes reguladoras na pesquisa tem sido demonstrada nas últimas décadas com trabalhos [Kauffman, 1993; Fumia & Martins, 2013] que estu-

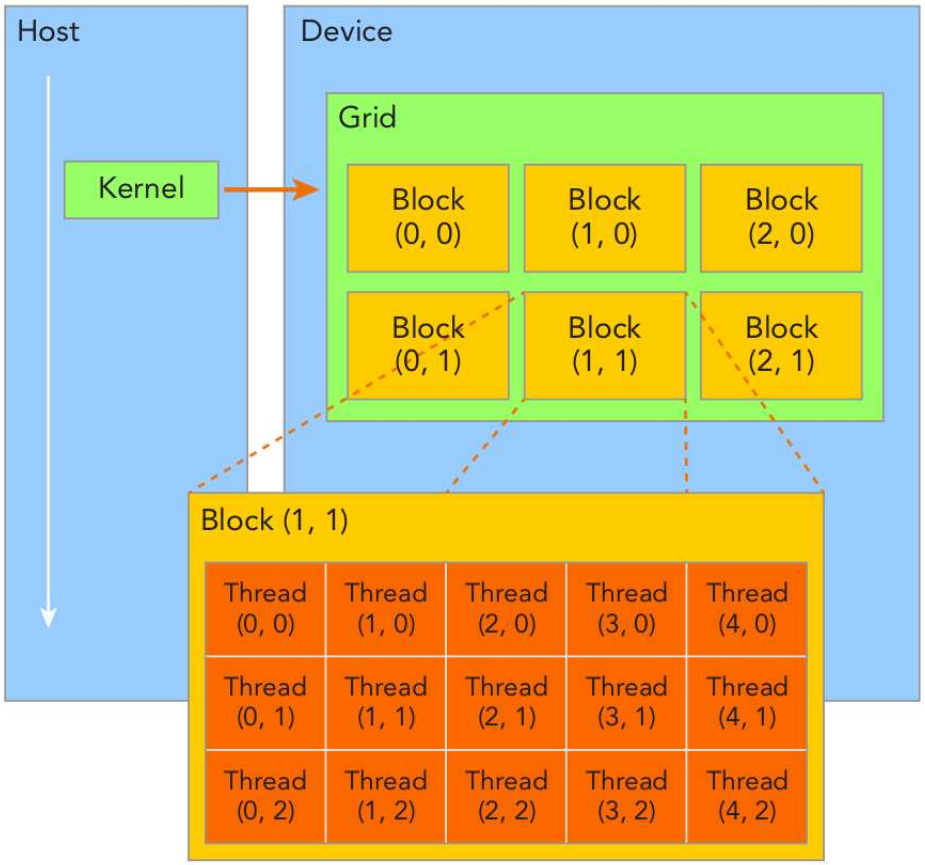


Figura 2.9. Estrutura do Grid interno da GPU.

dam diferentes abordagem de caracterização delas, a maioria não detalham e nem disponibilizam as implementações.

Na tabela 2.5 apresenta um comparativo dos recursos e funcionalidades das ferramentas para simulação dinâmica e análise de modelos qualitativos de GRN, destacando as semelhanças e diferenças.

As duas primeiras linhas mostram quais opções de atualização estão modeladas nas ferramentas: **Síncrono** e/ou **Assíncrono**. Posteriormente, analisando se a ferramenta é capaz de trabalhar com redes reguladoras já criados (lendo de um arquivo) ou possibilita a criação de uma rede de forma aleatória. A linha **Percurso Exaustivo** são para as ferramentas fazem a busca de atratores percorrendo todo o espaço de busca.

As ferramentas podem gerar gráficos estatísticos da transição dos estados, onde se mostra os estados pelos quais passa durante a simulação ou da composição dos atratores, descrevendo o tamanho destes e quais estados o compõe. Na Figura 2.10 vemos uma representação gráfica de composição de um atrator composto por 7 genes

(linhas) e 9 estados (colunas).

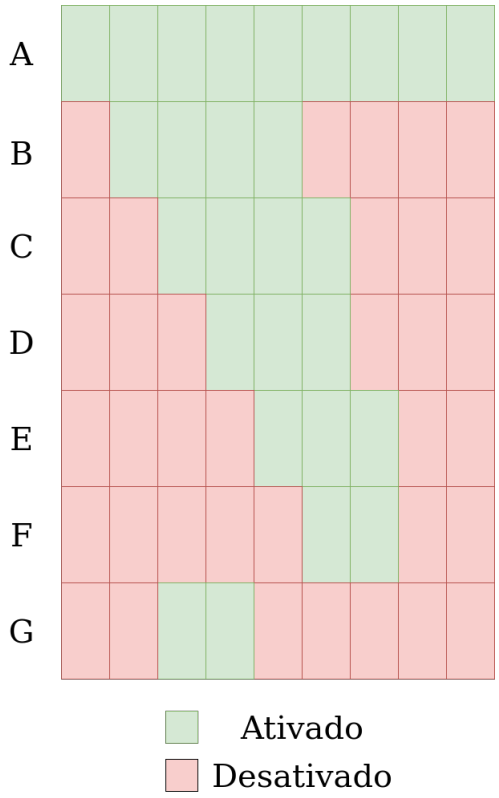


Figura 2.10. Gráfica da Composição de um atrator de uma rede biológica.

A Figura 2.11 mostra uma visão macro do diagrama de estados sem especificar quais são, mas mostrando os pontos estados pelo que passa a rede durante a simulação, até os atratores.

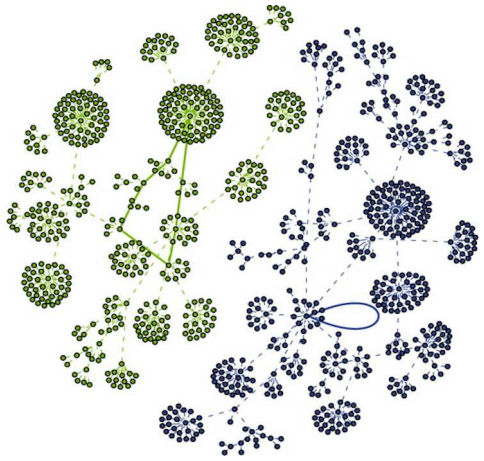


Figura 2.11. Gráfica de transição de estados de uma rede.

As perturbações são mudanças na estrutura do sistema, que são exploradas pelos cientistas para analisar o comportamento do mesmo, estas alterações podem acontecer no sistema biológico real, já seja pela influencia de um ator externo ou por mutações sofridas no organismo. As perturbações são representadas nas redes booleanas na mudança das interações dos nós (arestas), das equações que regem cada um (peso, limite) ou manter constante o valor de um ou mais genes durante a simulação ou parte do tempo de simulação.

Também é analisado se as ferramentas trabalham com rede representadas de forma booleana. Por ultimo analisamos se as ferramentas pode exibir o grafo da rede.

	Ferramentas			
Carateristicas	**BooleanNet**	**BooleSim**	**Boolnet**	**Cabernet**
Síncrono	X	X	X	X
Assíncrono	X		X	
Definição de GRN	X	X	X	X
GRN Aleatorio			X	X
Busca de Atratores	X	X	X	X
Percurso Exaustivo			X	X
Transição dos estados		X	X	X
Transição dos atratores			X	X
Perturbações	X	X	X	X
Representação booleana	X	X	X	
Exibição do Grafo da Rede	X	X	X	X

Tabela 2.5. Comparação das diferentes ferramentas de analises de GRNs.

Booleannet, segundo [Albert et al., 2008] é uma ferramenta que pode simular um modelo booleano a partir de um formato simples de entrada. A principal característica distintiva desta ferramenta, é fornecer suporte para modelar o comportamento dinâmico de subsistemas biológicos, em vez de focar em análise ou modelagem de rede de maior escala. Uma vez tendo as regras, a ferramenta pode empregar várias estratégias de simulação: iterações síncronas, atualizações estocásticas ou modelagem híbrida por meio de um sistema de equações diferenciais lineares. O **Booleannet** permite a integração de mecanismos não booleanos na simulação, expandindo sua aplicabilidade para um domínio mais amplo. Todos os aspectos do processo de simulação podem ser personalizados: os estados dos nós podem ser substituídos em diferentes estágios da operação, as regras de atualização podem ser alteradas e as equações diferenciais podem ser aumentadas ou substituídas. É uma ferramenta que precisa de conhecimento de programação, no caso da lingua-

gem Python, já que a estrutura da rede a testar, nós e equações de atualização, vai embutida no código. O pacote só tem funcionalidades básicas implementadas, inicialização de rede, simulação e visualização de grafos, o resto vai de acordo com a programação realizada pelo usuário.

A ferramenta **R BoolNet** [Müssel et al., 2010] fornece os principais métodos de simulação de grafos booleanos, síncronos, assíncronos e probabilísticos e inclui novas funções para pesquisa de atratores, análise de robustez e binarização. Suporta também a reconstrução de redes a partir de séries temporais, análise de redes especificadas por especialistas humanos, geração de redes aleatórias, perturbação de redes e identificação de atratores. Os genes podem ser temporariamente eliminados e superexpressos. Além disso, são fornecidos métodos convenientes de visualização. O BoolNet se integra a outras ferramentas de modelagem com grafos, como BioTapestry [Longabaugh et al., 2005] e Pajek [Batagelj & Mrvar, 1998]. R Boolnet é uma ferramenta que apresenta um maior número de funcionalidades, porem, ao igual que Booleanet são desenvolvidas com a ideia de ser usadas como complementos de alguma outra ferramenta, não possui uma interfase gráfica amigável para o uso pelo usuário comum. Desenvolvido em R apresenta uma boa base para a criação de gráficos estatísticos no analises das redes, os atratores com suas bacias e a transição dos estados.

BoolSim [Bock et al., 2014] é um simulador que é executado no navegador, portanto não requer download ou instalação. O BooleSim, o simulador de rede booleano, que suporta a importação dos formatos de arquivo de modelo mais comuns, como o usado pelas ferramantas Booleanet, RBoolNet e a importação do formato de intercâmbio interno do biógrafo (JSBGN [Krause et al., 2013]). Permite manipulação fácil através da funcionalidade no clique e visualização das propriedades dinâmicas de uma rede, bem como a exportação do modelo booleano, da visualização gráfica da rede e da série temporal, são algumas das caraterísticas que mais destacam dessa ferramenta. Porém pode ficar em um ciclo infinito na busca de um atrator. É uma ferramenta, simples, com um desenho básico orientada para trabalhar de forma online, sendo uma ferramente web, só apresenta a funcionalidades de simulação. Tem uma interface que permite o acompanhamento da simulação vendo com vai se comportando a transição dos estados, pode exportar o grafo e o gráfico de serie do tempo da transição dos estados.

A ferramenta mais próxima do trabalho proposto, por ser um plugin do Cytoscape que permite análise e simulação de modelos de rede reguladoras, é o **Cabernet** [Paroni et al., 2016], usada para descrever fenômenos e processos biológicos complexos, como a diferenciação e a evolução de câncer. Usando a simulação e analises

dos GRN booleanos, especificamente focados em fragmentos destas quando apenas estiver disponível uma caracterização topológica e funcional parcial da rede. Esta ferramenta permite a formulação de hipótese sobre as partes ausentes de redes reais bem como investigar suas propriedades genéricas. Das ferramentas apresentadas para comparação o Cabernet é a mais completa, enquanto a funcionalidades que propõe, porém ainda não permite o ajuste das funções de atualização dos nó a simulação é feita uma única vez. Posteriormente é apresentado o grafo com os resultados, ficando a ferramenta como um instrumento a mais para mostrar grafos e tabelas.

2.7.1 Exemplos de Redes

A maioria das redes usadas na literatura são descritas com equações booleanas. Além disso, muitas não são redes derivadas de redes biológicas, são apenas redes aleatórias sintéticas usadas para avaliação das técnicas. Um exemplo é o trabalho apresentado em [Shi et al., 2017] que usa uma rede reguladora baseada no câncer de mama, para o analises de predição de posivel desenvolvimento da rede.

Existem trabalhos que propõem o uso de lógica limiar [Darabos et al., 2011], mas usam de redes sintéticas, além de explorar propriedades de grafos livres de escala nos quais alguns vértices concentram a maioria das ligações [Aldana, 2003]. Além de redes sintéticas, o trabalho[Darabos et al., 2011] usa o mesmo valor de limiar para todas as equações, onde o valor é inferido, testando valores e analisando o comportamento da rede com base no regime de funcionamento da rede avaliado com a metodologia *Derida plot* [Derrida & Pomeau, 1986].

2.7.2 Uso da GPU na análise das redes biológicas

A abordagem proposta em [Borelli et al., 2013] implementa um acelerador em GPU para as redes com aceleração de mais de 200 vezes, mas não foram testadas redes reais, e os cálculos são baseados na abordagem da rede genética probabilística, onde o valor para cada estado no instante t só depende dos valores dos preditores no instante anterior de acordo com os sinais de expressão.

Já na abordagem proposta em [Trinh et al., 2014b] fazem uso da CPU e GPU, mostrando um ganho de 2x na versão implementada em GPU, mas propõe uma técnica para analisar a robustez das redes e a estrutura de retroalimentação das redes e não o estudo da dinâmica de evolução da rede nem cálculo de atratores.

Em um trabalho mais recente [Portocarrero Tovar et al., 2019] exploram a possibilidade de achar os atratores com a solução do problema de satisfatibilidade booleana, (SAT), que é conhecido por ser um problema NP completo, mas a abordagem proposta é a transformação do SAT no outro problema, *Hitting Set*, este é resolvido aplicando um algoritmo em paralelo implementado na GPU. Porém, esta abordagem só consegue testar grafos pequenos, levando 3,5 segundo e 9,15 horas para calcular 2 atratores para cada grafo, um de 3 e outro de 4 nós respetivamente.

Em relação ao uso de GPU, trabalhos anteriores do grupo de pesquisa da UFV mostraram o potencial de aceleração [Campos et al., 2011; Jacob et al., 2012], porém as avaliações não usaram redes derivadas de modelos biológicas e foram baseadas em redes sintéticas. Em relação ao uso de FPGA, uma avaliação com rede sintética foi feita em [Ferreira & Vendramini, 2010] e uma avaliação com apenas duas redes biológicas foi realizada em [da Silva et al., 2017]. Neste trabalho iremos avaliar novas técnicas sobre um conjunto amplo de redes. Parte das contribuições foram publicadas no trabalho [Rosa et al., 2019].

2.7.3 Cytoscape como ferramenta de trabalho

Como já foi apresentado o Cytoscape é uma ferramenta usada por biólogos no processamento e representação das redes biológicas e são vários os trabalhos que demostram isso [Cumbo et al., 2014]. Mas a maioria deles só centram no estudo da modularidade a robustez também baseado nas caraterísticas estruturais, como em [Truong et al., 2016] onde a regras que governam a dinâmica da rede são relações de disjunção ou conjunção escolhidas randomicamente com uma distribuição de probabilidade uniforme, também usam um algoritmo paralelo para o calculo da robustez, usando a biblioteca de OpenCl.

Outros trabalhos como [Shi et al., 2017] usam abordagem de *Machine Learning* para a identificação de biomarcadores de câncer biologicamente significativos a partir de dados de expressão de genes. Outros são serviços disponibilizados para ajudar no trabalho diário dos biólogos na previsão de funções proteicas e priorização de genes de doenças [Carlin et al., 2017].

Capítulo 3

CYTLFGRAPH

Um dos objetivos desta dissertação é o desenvolvimento de suporte para simulação de redes reguladoras com lógica limiar e com aceleradores como GPU. As ferramentas desenvolvidas foram projetadas para serem acopladas ao Cytoscape 3 2.3, usando vários linguagens de programação, como C++, Python e Java. As ferramentas são para análise e caracterização dos grafos booleanos/limiar que representam Redes Reguladoras, visando à simulações para o cálculo dos atratores e experimentação com clones.

Uma característica distintiva de nossa abordagem é a conversão do grafo booleano em um grafo onde a função de estado seja representado como se fosse uma TLF, proporcionando ao usuário final a possibilidade e facilidade de testar as mutações fazendo mudanças nos pesos com que influência uma proteína em outra. A TLF pode ser modelada como uma tabela de pesos que pode ser modificada em tempo de execução para ajustes das redes, sem a necessidade de compilação para avaliar uma pequena alteração. Uma função booleana também pode ser modelada como tabela, porém os ajustes e modificações não são tão diretos de serem feitos em comparação com a simplicidade do modelo com pesos e limiares.

Nas próximas seções detalhamos a solução proposta e as implementações desenvolvidas. Começando pela seção 3.1 onde explicamos o processo de conversão das equações booleanas em equações limiares. A seção 3.2 explica as implementações e os cálculos realizados para encontrar os atratores das redes analisadas pela ferramenta proposta nesta dissertação, o CyTLFGraph. A seção 3.3 apresenta os aspectos mais técnicos da implementação. A seção 3.3.1 apresenta as análises robustez e estabilidade das redes.

3.1 Transformação de Funções Booleanas em Lógica Limiar

Nesta seção iremos ilustrar, com um simples exemplo, como uma função booleana pode ser transformada em lógica limiar. Sendo esta a primeira etapa deste trabalho, a transformação das equações booleanas em representação limiar, usando o detalhado a introdução apresentada na seção 2.2.

Os passos para a transformação das equações booleanas em equações limiares estão apresentadas no algoritmo 3.1, onde se recebe uma matriz que representa a tabela de verdade da função booleana. A função limiar requer o armazenamento apenas dos pesos no lugar de uma tabela completa.

```
TLF(table[][]):
    for colum in input_colum{
        for line in table{
            if(exit_colum == true){
                positive = count(cell == 1)
            }
            else{
                negative = count(cell == 0)
            }
            Wi = 2 * (positive-negative)
        }
    }

    for line in table{
        for colum in line{
            if( exit_colum == true){
                line_th = sum(Wi * cell)
            }
        }
    }
    Th = min(line_th)

    return (W,Th)
```

Algoritmo 3.1. Algoritmo de conversão a TLF

O Código 3.1 apresenta o cálculo das funcões limiares. O algoritmo foi descrito mais informalmente na seção 2.2. Através de uma varredura na tabela verdade e nos vetores coluna_entradas e coluna_saidas, o primeira parte tem dois laços é encarregado de calcular o peso associada a cada entrada segundo a equação 6.

A segunda parte tem também dois laços e determina o valor do limiar para essa equação, é de interesse só aquelas linhas onde a evaluação da função seja 1, $f(a, b, c) = 1$, para isso percorremos as equações de cada linha que é a somatório dos valores das variáveis multiplicadas pelos pesos. Por exemplo baseados na Tabela 2.4 da seção 2.2, na linha 010 é $W_2 > T$, na linha 100 é $W_1 > T$, e na linha 110 seria $W_1 + W_2 > T$, o valor do limite T é calculado como o mínimo dos resultados da parte esquerda das inequações, no caso se $W_2 = 2$, $W_1 = 2$ e $W_1 + W_2 = 4$ o valor de $T = 2$.

A tabela de verdade tem m linhas (para um nó com $\log m$ entradas). se percorre cada coluna (n) da matriz, resultando na complexidade $n * m$. para cada laço, sendo no total $2(n * m)$. Agora a conversão para TLF é feita para cada um dos nó (N) do grafo o que levaria a uma complexidade de $N * m * n \equiv O(n^3)$, para todo o processo de conversão da rede reguladora.

3.2 Cálculo de Atratores

No modelo de atualização síncrono, como já mencionado, assume-se que os processos representados como arestas na rede a mesma duração e os estados são atualizados simultaneamente.

Nosso trabalho usa a abordagem síncrona, que apesar de ser uma simplificação oferece mais potencial de paralelismo acelerando as simulações. O trabalho pode ser estendido para a modelagem assíncrona. Apesar do assíncrono ser mais próximo dos modelos biológicos, ter uma simulação relevante demanda um esforço computacional grande para execução de várias repetições.

A cada passo de simulação, a rede muda de estado até encontrar um ciclo de estados estacionário denominados por atratores. Como já mencionado, os atratores estáveis e de menor tamanho são os de maior importância do ponto de vista biológico [Kauffman, 1969, 1993].

Para o cálculo dos atratores existem duas abordagem que podem ser implementadas cada uma com as suas características. A primeira consiste em usar uma estrutura de dado para guardar os últimos X estados, pode-se usar um vetor circular. Após o cálculo do próximo estado verifica se o estado atual está no vetor. Caso esteja, um atrator foi encontrado. Em seguida, a simulação continua para determinar o tamanho do atrator. Esta abordagem tem uma desvantagem de encontrar atratores com o tamanho máximo limitado pelo tamanho do vetor. A Figura 3.1 ilustra um exemplo da abordagem com uma fila de até 5 elementos.

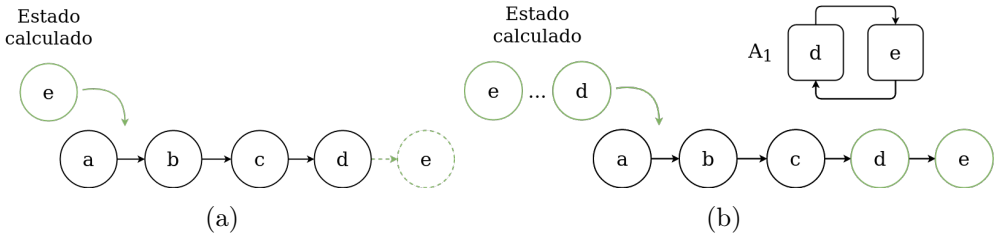


Figura 3.1. Cálculo de atratores usando uma fila circular com até 5 elementos;(a) Inserção de um estado calculado que não está presente na fila; (b) Encontrando um atrator ao localizar o estado d na fila.

A segunda abordagem requer pouca memória (apenas 2 estados) e pode encontrar um atrator de qualquer tamanho. Esta abordagem é baseada no trabalho apresentado por [Bhattacharjya & Liang, 1996], são usadas apenas duas instâncias da rede (I_1, I_2) onde a simulação do algoritmo começa no mesmo estado. Com o passo do tempo a instância I_1 avança um passo na evolução do grafo e a instância I_2 avança dois passos, como ilustrado no diagrma de estados da Figura 3.2 para a rede da Figura 2.1. E_1 é o estado inicial. No primeiro momento calcula-se $I_1(1) = E_2$ e $I_2(2) = E_3$. Como I_2 avança duas vezes mais rápido ira atravessar o ciclo antes de se encontrar com I_2, no exemplo da Figura 3.2 no $t = 2$ é onde $I_1(t) = I_2(2t)$. Tendo o ponto de partida do atrator, para calcular o período o que seria o tamanho deste, deixa-se I_1 estacionaria e avançamos com S_2 com passo 1. Assim quando I_1 e I_2 se encontrarem de novo, $I_1(t_x) = I_2(t_x + p)$ a simulação termina e o numero de passos dados P seria o período desse atrator, no exemplo da Figura 3.2 o período é de 1, correspondente ao estado E_3, também desta forma é possível achar os estados que conformam a bacia desse atrator, chamado de transiente, que no exemplo tem cumprimento 2 e está formado pelos estados E_1 e E_2.

Segue o algoritmo simplificado para encontrar um atrator:

```
For E=0 to N // scan all states
    I1 = I2 = E
    Repeat
        Step(I1) // next state
        Step(I2)
        Step(I2)
    Unitl I1 == I2
```

Algoritmo 3.2. Pseudo Código do cálculo do atrator

Para este código veja-se como a simulação começa por um estado inicial 0 até N número de simulações, podendo ser 2^N. Esta abordagem só é viável para redes

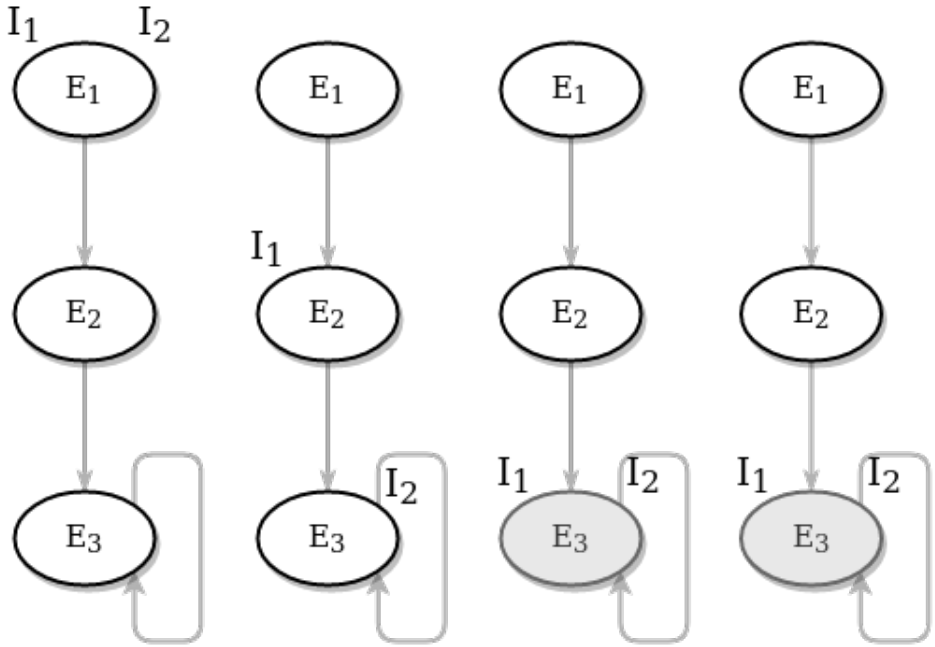


Figura 3.2. Sequencia de passos para o cálculo do atrator

.

com N pequeno. Para redes maiores pode-se fazer uma varredura randômica do espaço de estados até um determinado tempo limite de execução.

3.3 Construção da Aplicação

A solução proposta está implementada em três linguagens de programação e para a interação de informação entre estas é usado a comunicação via transferência de arquivo, tanto para a entrada dos dados iniciais como para a persistência dos dados já calculados, de acordo com isso, o grafo é carregado desde um arquivo de texto onde estão as equações booleanas que regem cada nó, usando um formato de atribuição de variáveis, um exemplo de arquivo de entrada se mostra a continuação no código 3.3.

```
Cas8 = ( (Cas6) and not (cFLIP) )  or ( (FADD) and not (cFLIP) )
MEKK1 = (TRAF)
Cas9 = ( ( (Cas3) and not (AKT)  ) and not (IAP) )
or ( ( (Cas12) and not (AKT)  ) and not (IAP) )
Cas6 = ( ( Cas3  ) and not ( IAP  ) )
RIP = ( TRADD )
Cas3_dummy = ( Cas3 )
Cas3 = ( (APC) and not (IAP) )  or ( (Cas8) and not (IAP) )
Mdm2 = (AKT)  or (p53)
...
```

```
TRADD = (TNFR1)
PIP2 = (GFR)
FADD = (TRADD)
BAD = ( (p53) and not (AKT) )
TRAF2 = (RIP)
NTFK = 1
PSD
```

Código 3.3. Entrada CyTLF, declaração de cada nó em equações booleanas

Veja que no arquivo de entrada cada linha representa a função de atualização de cada nó, este está expressado na esquerda da equação, já na direita a função booleana. No caso de um nó ficar fixo basta colocá-lo no arquivo igualado ao valor desejado se não se designar uma função ou uma variável, o valor por definição será 0.

Numa primeira etapa é realizado uma conversão das equações usando um programa escrito na linguagem Python, e com o serviço que proporciona o Cytoscape (CyRestClient) é montado visualmente o grafo na plataforma. Tanto o resultado da conversão de booleano para equações TLF como as equações originais são armazenadas na tabela dos nós, estrutura do Cytoscape para a descrição dos grafos, de onde depois é recuperada a informação das equações TLF para gerar as entradas da segunda etapa.

Para o cálculo dos atratores fazemos uso da aceleração e potencialidades oferecidas pela computação paralela baseada na CPU e GPU, implementados usando a extensão de OpenMP e CUDA para C/C++ respetivamente, dado que usualmente esses grafos são complexos com grande número de nós e arestas, e para o cálculo do atrator é preciso visitar todos ou a maior quantidades de estados possíveis. O resto do processamento, assim como a integração de todos os dados, é implementado em forma de *Bundles* do *framework* OSGI para oferecer uma aplicação de Cytoscape.

Segue o diagrama das funcionalidades implementadas e do fluxo de ações que podem-se seguir no uso da ferramenta CyTLFGraph:

A Figura 3.4 mostra a interface implementada de como é escolhido a plataforma que será usada para o calculo dos atratores, e na mesma tela já o usuário pode escolher a quantidade de estados que serão visitados. Um adicional que é oferecido pela solução proposta é a possibilidade de determinar o tipo de busca dos atratores: sequencial, quando os estados iniciais seguem a sequencia de 1, no exemplo da Figura 2.1, do total de estados $2^3 = 8$, vai percorrendo os estados 1,2,3 ate quantidade determinada, pode-se determinar os limites para assim fazer uma analises por lotes, ou randômico onde os estados são escolhidos aleatoriamente até se cumprir com a quantidade de simulações desejadas, que seria marcada pelo o usuário.

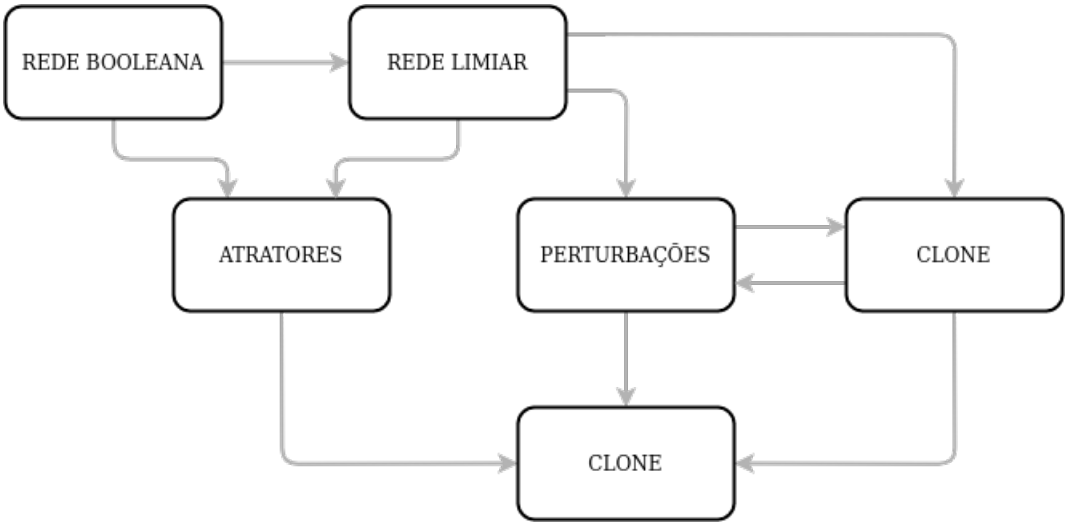


Figura 3.3. Fluxo de funcionalidades no CyTLFGraph

.

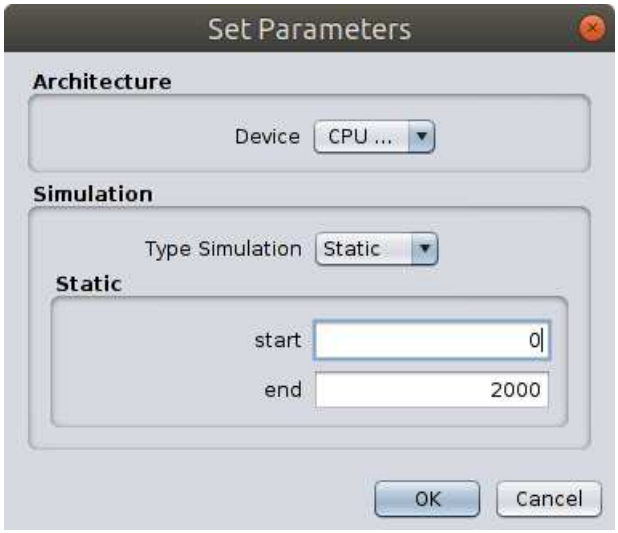


Figura 3.4. Seleção de plataforma no Cytoscape

.

Todos os recursos oferecidos pelo CyTLFGraph são registrados em serviços, herdados da classe NetworkTaskFactory, implementando um padrão *Factory*, Figura 3.5. Tendo em mente que as tarefas demoram no processamento, as tarefas são executadas de forma assíncrona, caraterística oferecida pela classe AbstractTask que cada funcionalidade implementa. Finalmente obtemos os serviços de desmembramento do Cytoscape na classe CyActivator, que estende da classe Abstract CyActivator, funcionando como uma ponte entre o Cytoscape e nossa implementação.

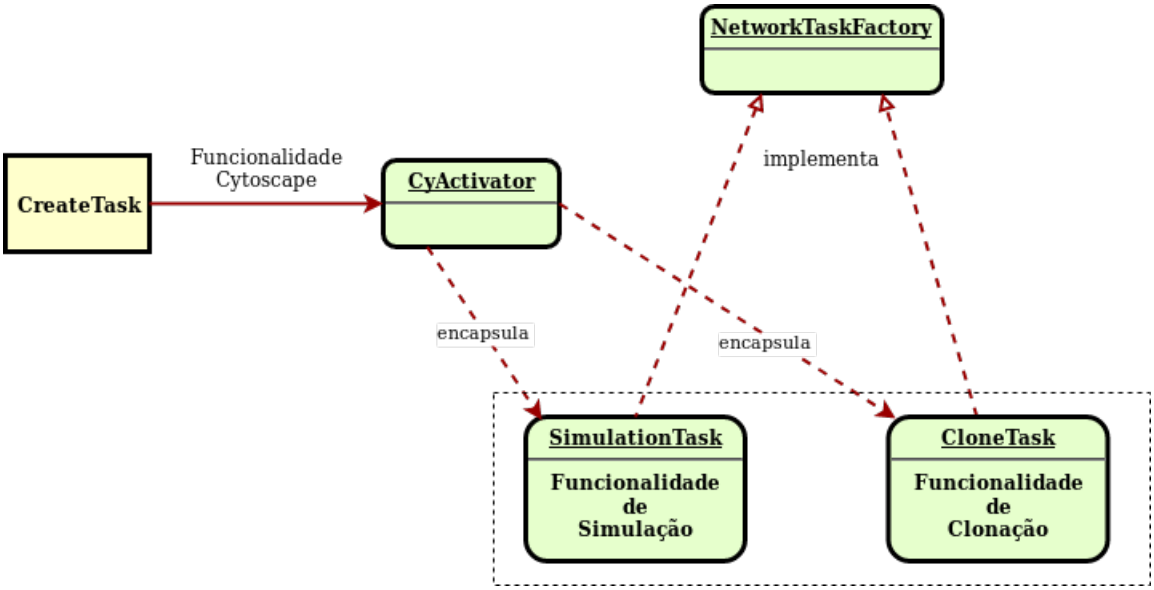


Figura 3.5. Exemplo de implementação do padrão *Factory*

.

3.3.1 Análises de robustez e estabilidade

Os organismos vivos são resistentes a uma variedade de mudanças genéticas, o que segura que funções especificas se mantenham apesar destas alterações externas ou internas, e sendo as redes reguladoras um modelo simples da dinâmica das interações biológicas, é de interesse estudar a caraterística de tolerância a falhas o que seria a robustez destas [Kitano, 2004].

Agora estudar a estrutura das bacias atratoras é interessante, pois pode refletir na estabilidade dos tipos de células à alterações. Por exemplo, a rede se encontra presa em um atrator, um gene muda de estado por algum motivo, a rede passa novamente por vários estados e depois estabiliza novamente no atrator [Darabos et al., 2011]. Tomando de exemplo a rede que ilustra a transição dos estados no capitulo 2, que se encontra no estado estacionário E_5 (101) como se mostra na Figura 2.2. Suponha que o gene C se torne inativo por um motivo externo, a rede irá retornar o estado E_4 (100) e transitará pelo estado E_3 (011) até chegar novamente ao atrator E_5 (101).

Uma característica do CyTLFGraph que proporcionamos é a capacidade de estudar o comportamento do grafo de acordo com as diferentes alterações que podem aparecer, isto é possível com a funcionalidade de clone que a ferramenta apresenta, o que facilita o análises da robustez e estabilidade deste e dos atratores, onde cada copia terá seu próprio resultado.

As modificações dos pesos são mais fáceis e intuitivas para o usuário testar usando a representação TLF das equações que definem cada nó, onde se mostra o nome dos nós atuantes os pesos deles e a inequação que rege o valor final de saída de cada equação, como já se apontou, o valor limiar.

A Figura 3.6 mostra a plataforma de Cytoscape na Seção do Painel de Controle, que se encontra listado o grafo original e os clones deste onde os usuários podem experimentar pequenas alterações nos pesos das arestas, nos valores limiares ou até fixar um nó, visado a analisar o comportamento da rede e deferentes hipóteses, todas estas perturbações são mostradas separadas pelo rótulo *Mutations* e a nomenclatura usada para os clones sera do *nome do* $grafo\_n_o$ *copia*.

Da mesma forma é usada uma identificação para os resultados das simulações, onde atratores estão devidamente nomeados usando o tipo de arquitetura, quantas simulações foram rodadas, no caso se for a simulação de um clone, mostrando o numero que identifica a este. Todo isto baixo a rotulo de *Atractors*, ficando a nomenclatura $Atractors\_arquitetura\_Sn_o$ *simulaesn*$_o$ *do clone*.

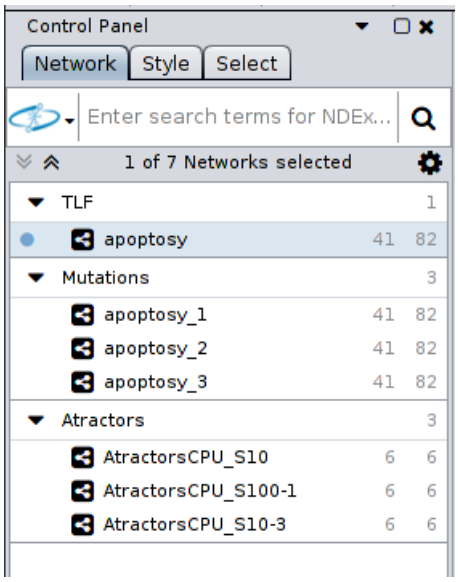


Figura 3.6. Estudo de um GRN com as diversas perturbações

A possibilidade de gerar uma perturbação nas redes analisadas, como ilustrado na Figura 3.7, a possibilidade de fixar o valor de um nó, sendo este valor os únicos possíveis, ativo (1) ou inativo (0), esta fixação afeta de forma direta o comportamento desse nó já que passa a desconsiderar inclusive a equação que poderia estar governando-lo no momento. Caso não seja especificado um valor na hora de fixar a função, este será de 0 por definição.

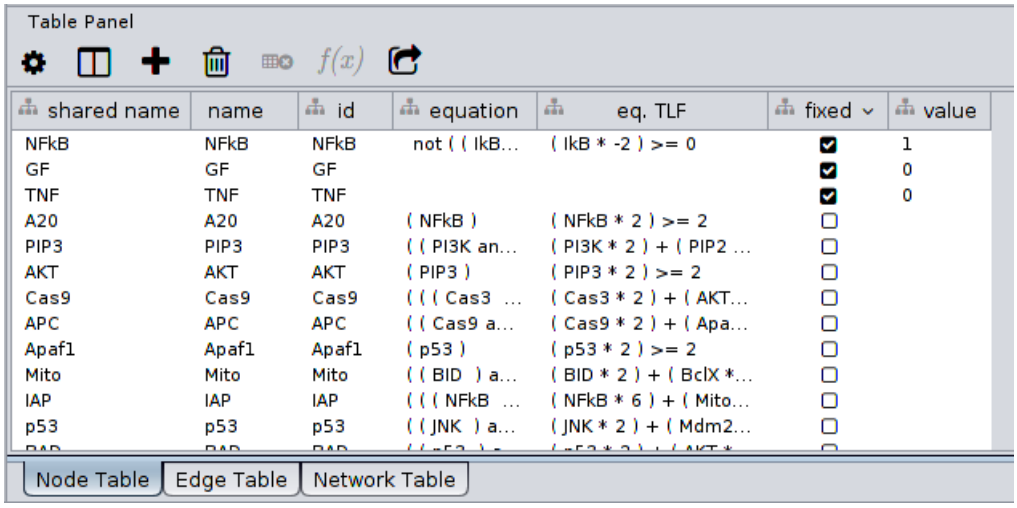
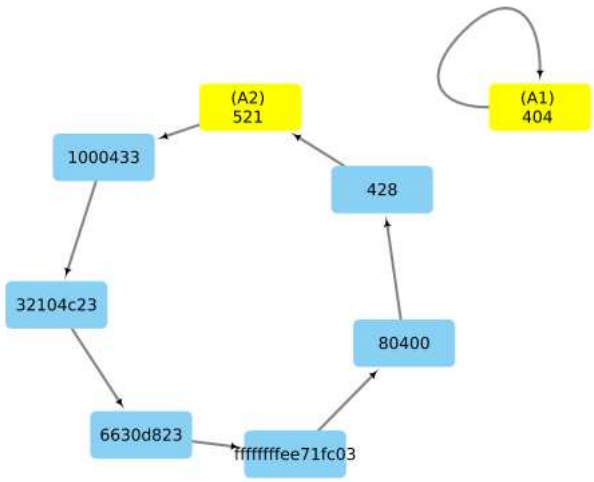


Figura 3.7. Armazenando as equações booleanas e limiares na Tabela de nós

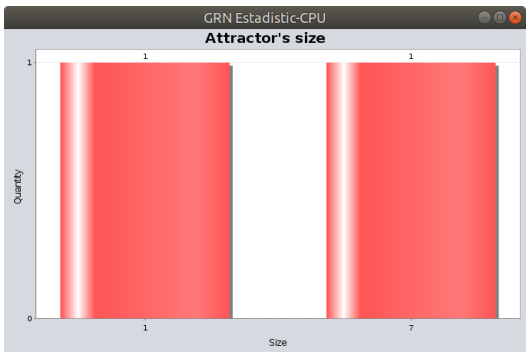
Todas as estatísticas podem ser visualizadas na interface do CyTLFGraph, a distribuição dos atratores o tamanho deles, o tamanho da bacia dele, também a proximidade dos atratores, baseados na distancia de *Hamming*, mesmo que sejam visualmente diferentes pode-se determinar quão perto de mudar estão cada um se for apresentada alguma modificação.

A Figura 3.8 os estados que compõem os atratores são representados em hexadecimal (para evitar números binário muito grandes). O atrator é identificado pelo primeiro estado dele em amarelo, isto, para que seja mais visível a proximidade que poda existir, cada atrator está identificado pelo nome, no primeiro nó que é achado, da mesma forma que este fica marcado no caso de color amarelo. Além dos atratores, usando histograma mostra-se o tamanho dos atratores e as bacias dos atratores na Figura 3.8.

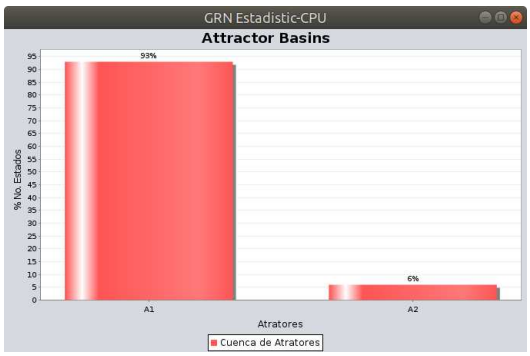
Também nosso aplicativo apresenta outras métricas de rede, como é a classificação de importância dos nós seguindo a proposta do algoritmo PageRank, mesmo que seja um algoritmo para classificação é indexação de paginas web, é aplicado para os grafos, denotando a importância que cada nó tomando em conta a relação dele com a vizinhança e a influencia dos outros nós. Dando ao usuário uma ferramenta a mais para determinar que proteínas ou elementos são mais importantes no organismo estudado, que afeta em alguma medida a evolução do mesmo. O cálculo feito é disponibilizado no Painel da Tabela, coluna chamada *PageRank*, a Figura 3.9 mostra esta funcionalidade, que é acessível, junto as outras, usando o menu referente ao aplicativo.



(a)



(b)



(c)

Figura 3.8. Visualização de Simulação. (a) Grafo da composição dos atratores (b) Estatística de tamanho dos Atratores (c) Estatística das bacias dos Atratores.

3.3.2 Considerações Finais

Neste capítulo foi apresentado todo o referente à construção da solução proposta, que como se mostrou apresenta as funcionalidades mais procurada nos usuários que visam estudar o comportamento das redes reguladoras de genes, além de mostrar uma via de convenção da plataforma Cytoscape em um sistema heterogêneo usando o processamento da CPU e da GPU, trazendo o melhor de ambos mundos. CyTLF-Graph sendo uma aplicação do Cytoscape, permite que seja multiplataforma, não só é usado a linguagem de Java, temos Python e C++, o que a converte em uma aplicação heterogênea. Seguindo com os padrões de implementação ditados para o tipo de aplicação Cytoscape, permite que seja escalável. Tomando como exem-

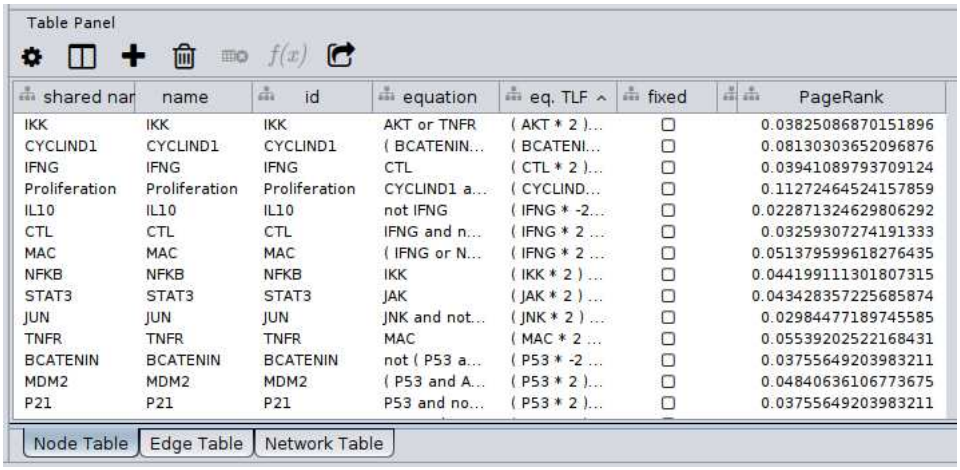


Figura 3.9. Visualização de Simulação. (a) Grafo da composição dos atratores (b) Estatística de tamanho dos Atratores (c) Estatística das bacias dos Atratores.

plo as diferenças mostradas na análise das ferramentas concorrentes, CyTLFGRaph trouxe uma proposta mais completa enquanto analises e visualização das redes Reguladoras, com o alto nível de personalização esta, visa atingir as necessidades dos cientistas. O uso de representação Limiar nas funções de atualização dos nós permite uma maior operabilidade nos testes de alterações da rede, que junto ao uso de aceleradores como a GPU permite que as análises das redes seja mais rápida sem deixar de ser profundo e exaustivo para toda a rede, o que representa um ponto forte na hora de caracterizar o aplicativo.

Capítulo 4

RESULTADOS

Neste capítulo iremos apresentar os experimentos realizados com as ferramentas desenvolvidas para redes reguladoras de genes e acopladas no ambiente Cytoscape.

Na seção 4.1 apresentamos as redes selecionadas para realizar os experimentos. Na seção 4.2 começamos com o primeiro experimento, analisando o custo de instruções para calcular um estado da rede. Estes resultados são apresentados dividindo as abordagens em booleana e limiar em duas subseções: 4.2.1 e 4.2.2, respetivamente. A seção 4.3 testamos a ferramenta com o calculo dos atratores nas redes originais e em perturbações.

4.1 Redes Reguladoras

Para validar mapeamento das funções booelanas em equações TLF e realizar os outros experimentos um conjunto de 6 redes reguladoras associadas a redes biológicas foram selecionadas, A Tabela 4.1 enumera as redes utilizadas nos testes. A seguir descrevemos cada uma das redes selecionadas.

Nome	G
Cholesterol Regulatory Pathway	1
Apoptosis Network	2
T-LGL Survival Network 2011	3
Glucose Repression Signaling 2009	4
Lymphopoiesis Regulatory Network	5
Signal Transduction in Fibroblasts	6

Tabela 4.1. As 6 Redes Reguladoras usadas nos experimentos

A síntese do colesterol (**Cholesterol Regulatory Pathway** Kervizic & Cor-

cos [2008]) desempenha um papel central na dislipidemia e, finalmente, no câncer por intermediários como *mevalonate*, *farnesyl pyrophosphate* e *geranyl geranyl pyrophosphate*. Este é o primeiro modelo biológico de sistemas dinâmicos da sínteses do colesterol humano e vários de seus principais elementos de controle regulatório.

Apoptosis Network [Mai & Liu, 2009] modela os princípios da interação molecular associada à irreversibilidade da apoptose[1] celular e à estabilidade da sobrevivência celular, esta rede booleana integra as vias de sínteses pró-apoptóticas intrínsecas e extrínsecas e as vias de sínteses de transdução de sinal pró-sobrevivência.

T-LGL Survival Network 2011 [Saadatpour et al., 2011] é o modelo de rede booleano que carateriza os linfócitos granulares grandes de células T (T-LGL do inglês), conhecido como leucemia ou câncer do sangre, principalmente estudada para a identificação de possíveis alvos terapêuticos, sendo que nenhuma terapia curativa ainda é conhecida para esta doença.

Glucose Repression Signaling 200[Christensen et al., 2009], representa o modelo lógico de repressão ao fermento da glicose, formalizado como um hipergrafo, foi construído com base em interações regulatórias verificadas e inclui 50 transcrições de genes, 22 proteínas, 5 metabólitos e 118 interconexões. O modelo constitui uma ferramenta primária para armazenar conhecimento regulatório, que é usado na busca de incoerências em hipóteses e avaliando o efeito da exclusão de elementos reguladores envolvidos na repressão à glicose.

Lymphopoiesis Regulatory Networ [Mendoza & Méndez, 2015] é uma rede reguladora de 81 nós, representando vários tipos de moléculas que se regulam entre si durante o processo de linfopoiese, já que existe interesse em conhecer os mecanismos moleculares que controlam a diferenciação das linhagens de células sanguíneas. As interações regulatórias foram inferidas principalmente a partir de dados experimentais publicados. A rede é modelada como um sistema dinâmico contínuo, na forma de um conjunto de equações diferenciais.

Por fim, a **Signal Transduction in Fibroblasts** [Helikar et al., 2008], Por ser muito complexas as redes bioquímicas de tradução de sinal, pelo alto grau de interconectividade, existem especulações de que são capazes de processamento de afirmações, y para testar essa hipótese foi criado um modelo, com os mecanismos lógicos de cada nó descrito completamente para permitir simulação e análise dinâmica. Foi observado que a rede executava classificações nítidas, mesmo diante ruídos adicionais.

[1]Representa a energia que se requer para a espontânea morte de uma célula, com características morfológicas e bioquímicas específicas

Todas estas redes reguladores são grafos e possuem propriedades que podem ser exploradas na modelagem com aceleradores como FPGAs[2] e GPUs. Uma das propriedades é o grau de entrada dos vértices para saber o tamanho das equações. Em geral, a maioria das redes biológicas é caraterizada por ter um grande número de vértices com poucas ligações e poucos vértices com muitas ligações e o grau médio é em torno de 2. Esta propriedade é importante para seleção da estrutura de dados para modelagem dinâmica das redes com TLF.

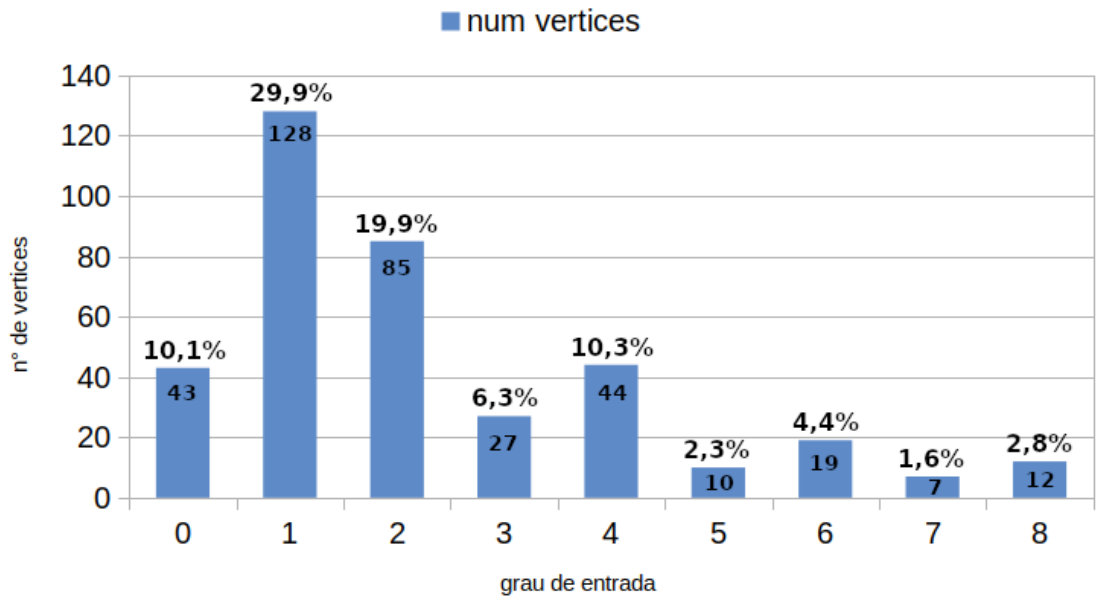


Figura 4.1. Histograma do grau de entrada das 16 redes reguladoras.

A Figura 4.1 mostra o histograma de distribuição do grau de entrada dos vértices nas redes avaliadas. Podemos observar que existe uma grande quantidade de vértices com um grau de entrada pequeno (0, 1 ou 2), onde $10,05+29,91+19,86 = 59,81\%$ dos vértices tem grau de entrada inferior a 3, já mencionado na literatura Aldana [2003]

A Tabela 4.2 caracteriza as 6 redes reguladoras de genes avaliadas. As colunas *Rede*, v e *arestas* mostram o identificador da rede (correspondente a Tabela 4.1), o número de vértices e o número de arestas, respectivamente. A segunda, terceira, quarta e quinta coluna mostram a quantidade de vértices agrupados pelo tamanho do grau de entrada. A coluna $0-1$ mostra o total de vértices com grau de entrada 0 ou 1. Podemos observar que a maioria dos vértices possui grau de entrada igual ou inferior a três. Entretanto, como as redes reguladoras de genes são grafos livre de escala

[2]

Rede	v	Grau de entrada 0-1	2-3	4-7	8+	Total Arestas
1	34	26	8	-	-	43
2	41	23	9	9	-	82
83	60	6	24	20	16	255
4	73	40	31	2	-	107
5	81	47	15	11	14	204
6	139	29	25	38	65	1014

Tabela 4.2. Caracterização dos grafos das redes biológicas

[Aldana, 2003], existe a presença de *hubs* que possuem um grau alto de entrada mesmo para um grafo relativamente pequeno como a rede 3, onde encontramos vértices com mais de 8 entradas.

4.2 Cálculo de um estado

O primeiro experimento realizado foi analisar o custo em instruções, em linguagem de máquina, necessárias para avaliar um estado de cada grafo, para cada uma das arquiteturas alvo. Este experimento foi selecionado por ser a operação básica de vários algoritmos e para verificar o limite inferior para execução dos algoritmos. Os testes feitos na arquitetura CPU foram executados no processador Intel(R) Xeon(R) CPU E5-2630v3 2.4 GHZ com 8 núcleos. O código foi compilado com GCC versão 5.4.0. com -O3 como sinalizador de otimização. Já na arquitetura GPU foi utilizado uma GPU GTX 1070 da Nvidia com 1.920 núcleos e 1,683 GHz.

4.2.1 Função booleana

A Tabela 4.3 tem as colunas CPU e AVX que mostram o número de instruções para o cálculo de estado de cada grafo, em ambos casos foi usado otimizador O3, a vetorização do código AVX foi feita automaticamente pelo compilador GCC, e acrescentando as otimizações *unroll-loops*, *omit-frame-pointer*, *inline*, com as opções *arch=native*, *tune=native*, *no-zero-upper* e com o alvo AVX. A coluna GPU mostra as instruções aferidas com a ferramenta *nvprof* na versão CUDA 10.1.

Neste experimento se observou que para os grafos de equações booleanas o uso de otimizações como o O3 e a vetorização do código não têm efeito na redução do número de instruções, sendo que o compilador não reconhece o laço(*loop*) automaticamente das equações booleanas e sim aquele que tinha o cálculo aritmético.

rede	Número de Instruções Booleano			
	Arestas	CPU	AVX	GPU
1	36	185	185	195
2	43	384	384	356
3	62	630	630	432
4	75	464	464	650
5	83	664	664	618
6	141	2387	2387	2612

Tabela 4.3. Número de instruções geradas para as equações booleanas.

Observando a Tabela 4.3, como se comentou, que devido a que o compilador não vetorizar os laços para as equações booleanas, os valores nas colunas CPU e AVX são os mesmos. Já na GPU, os resultados são próximos, porém uma GPU irá executar centenas a milhares de *threads* em paralelo. Medir o número de instruções dá uma ideia do mapeamento do conjunto de equações no *assembly* e o custo em cálculos das expressões.

O desempenho das arquiteturas de CPU e GPU, para as redes foi comparado olhando o numero de estados calculados e o tempo de execução. Para cada rede foram avaliadas um conjunto de 10 milhões de estados. O tempo de execução das soluções com CPU (incluindo $OpenMP$ e AVX) foi aferido com a biblioteca *chrono*. O tempo de execução da GPU foi medido com nvprof. A versão OMP faz uso de vários threads com uma implementação simples. Vale destacar que a maioria dos processadores oferece suporte para OMP. Portanto se seu processador possui múltiplo núcleos, o OMP é uma alternativa de aceleração com pequenas modificações no código.

rede	Mega Estados/Segundos Implementação Booleana		
	CPU	OMP	GPU
1	74	174	8569
2	37	101	3625
3	17	51	3760
4	26	98	857
5	17	56	701
6	4	18	275
A	1,0	3,1	83,4
G	1,0	3,3	68,8

Tabela 4.4. Cálculo de Estados usando as equações booleanas nas diversas plataformas: CPU, GPU.

A Tabela 4.4 mostra o número de estados avaliados em Mega estados por segundo. A coluna *rede* mostra o identificador da rede reguladora. Podemos observar que o pior desempenho é da CPU simples com apenas 1 *thread* na coluna CPU. Não foi colocada uma coluna AVX já que tem o mesmo numero de instruções que a implementação CPU. As duas últimas linhas da Tabela 4.4 mostram o ganho de desempenho de todas as implementações booleanas em relação a sua versão CPU simples. Foram calculadas as médias geométricas dos ganhos de desempenho (linha G) e a média aritmética dos ganhos de desempenho (linha A).

No segundo grupo de implementações temos a paralelização do cálculo com OMP para 8 *threads*. Podemos ver um ganho médio de 3x tanto na média aritmética, quanto na geométrica. Finalmente, a versão GPU apresenta um ganho de 65-85x com a versão booleana.

4.2.2 Função Limiar

O número de instruções e desempenho também foi avaliado para a variante de função limiar.

	Número de Instruções Limiar			
rede	Arestas	CPU	AVX	GPU
1	36	255	185	195
2	43	419	162	783
3	62	872	351	1396
4	75	356	186	1016
5	83	623	322	1724
6	141	2063	997	6303

Tabela 4.5. Número de instruções geradas para as equações limiares.

Segundo a Tabela 4.5 na maioria dos casos o número de instruções para processar as equações na GPU é bem maior que para a versão CPU. Já a versão limiar vetorizada pelo compilador GCC possui a maior redução, em media de até 2x em relação à CPU. Apesar de executar mais instruções, com a GPU dispara milhares de *threads* em paralelo, o tempo de execução é menor.

Um trecho do código assembler X86 da CPU gerado para o calculo de um estado de uma rede, se apresenta no Código 4.2 em comparação com o código mapeado em *assembly* ptx da implementação feita na GPU, mostrado no Código 4.1.

```
        xor     r9d, r9d
        test    rdx, rdx
        je      .L1
```

```
.L5:
        mov     rax, QWORD PTR [rdi+r9*8]
        mov     rcx, rax
        mov     r8, rax
        mov     r10, rax
        shr     rcx, 14
        sal     r8, 52
        and     ecx, 2
        sar     r8, 63
        lea     r8, [rcx+r8*2]
        cmp     r8, 2
        sbb     rcx, rcx
        not     rcx
        and     ecx, 16384
        cmp     r8, 2
        sbb     r8, r8
        shr     r10, 17
        sal     rax, 32
        and     r10d, 2
        sar     rax, 63
        and     r8, -16384
        lea     rax, [r10+rax*2]
        add     r8, 147456
        cmp     rax, 1
        cmova   rcx, r8
        mov     QWORD PTR [rsi+r9*8], rcx
        add     r9, 1
        cmp     rdx, r9
        jne     .L5
.L1:
        rep ret
```

Código 4.1. Código *assembly* X86 mapeado da implementação na CPU

```
        mov.u32         %r1, %tid.x;
        mov.u32         %r2, %ntid.x;
        mov.u32         %r3, %ctaid.x;
        mad.lo.s32      %r4, %r2, %r3, %r1;
        cvt.u64.u32     %rd1, %r4;
        setp.ge.u64     %p1, %rd1, %rd4;
        @%p1 bra        BB0_2;

        cvta.to.global.u64      %rd5, %rd2;
        shl.b64         %rd6, %rd1, 3;
        add.s64         %rd7, %rd5, %rd6;
```

```
    ld.global.u64   %rd8, [%rd7];
    shr.u64         %rd9, %rd8, 14;
    and.b64         %rd10, %rd9, 2;
    bfe.u64         %rd11, %rd8, 11, 1;
    neg.s64         %rd12, %rd11;
    and.b64         %rd13, %rd12, -2;
    add.s64         %rd14, %rd10, %rd13;
    setp.gt.u64     %p2, %rd14, 1;
    selp.u32        %r5, 1, 0, %p2;
    mul.wide.u32    %rd15, %r5, 16384;
    shr.u64         %rd16, %rd8, 17;
    and.b64         %rd17, %rd16, 2;
    bfe.u64         %rd18, %rd8, 31, 1;
    neg.s64         %rd19, %rd18;
    and.b64         %rd20, %rd19, -2;
    add.s64         %rd21, %rd17, %rd20;
    setp.gt.u64     %p3, %rd21, 1;
    selp.u32        %r6, 1, 0, %p3;
    mul.wide.u32    %rd22, %r6, 131072;
    or.b64          %rd23, %rd15, %rd22;
    cvta.to.global.u64      %rd24, %rd3;
    add.s64         %rd25, %rd24, %rd6;
    st.global.u64   [%rd25], %rd23;

    ret;
```

Código 4.2. Código textitessembly PTX mapeado da implementação na GPU

Nas figura se mostram como para uma mesma tarefa a realizar, o segmento de código das instruções muda de acordo com a arquitetura, e como para a *GPU* isto traze um aumento do numero de instruções, mas a diferença está na quantidades de *threads* disparados.

Um exemplo claro é quando o algoritmo pega o valor armazenado no vetor de estados iniciais para calcular o próximo estado desse nó,

```
V = init_rand[i]
```

no *assembly* X86 a instrução é:

```
mov     rax, QWORD PTR [rdi+r9*8]
```

o que significa o movimento do valor que esta no segmento de memoria $(rdi + r9 * 8)$ para o registro rax. Já na arquitetura GPU com o assembler PTX a mesma operação leva 3 instruções usando 5 registros:

```
shl.b64         %rd6, %rd1, 3;
```

```
add.s64         %rd7, %rd5, %rd6;
ld.global.u64   %rd8, [%rd7];
```

Na primeira instrução faz um corrimento para a esquerda de 3 posições do registro de 64 bits, $rd1$, logo depois faz a somatória do corrimento com o registro que representa o ultimo endereço de memoria acessado, guardado no registro $rd5$, essa soma é um inteiro signado de 64 bit ($.s64$), com isso já pode acessar à memoria para pegar o valor, sendo este um inteiro não signado de 64 bit ($.u64$).

	Mega Estados/Segundos Implementação Limiar				
rede	CPU	AVX OMP	OMP	AVX	GPU
1	49	197	174	76	2685
2	25	103	133	48	2644
3	13	40	59	24	1436
4	19	78	83	20	898
5	13	42	42	14	617
6	5	26	26	5	171
A	1,00	4,1	4,5	1,0	51,1
G	1,00	3,9	4,3	1,0	60,5

Tabela 4.6. Cálculo de Estados usando as equações limiares nas diversas plataformas: CPU, GPU.

A Tabela 4.6 mostra os Mega estados por segundo que as redes com equações booleanas conseguiram. Podemos observar que o pior desempenho é da CPU com apenas 1 *thread* nas colunas CPU e AVX. Da mesma forma que nas equações booleanas as duas últimas linhas da tabela mostram o ganho de desempenho de todas as implementações em relação a versão CPU simples limiar, com o cálculo da média geométrica dos ganhos de desempenho (linha G) e a média aritmética dos ganhos de desempenho (linha A).

Já na paralelização do cálculo com OMP o ganho foi de até 4x em ambas métricas, a media aritmética e a media geométrica, sempre seguindo a mesma caraterística de 8 *threads*. Finalmente, como no analises booleano, a versão GPU teve um ganho entre 50-60x

4.2.3 Comparação de Desempenho

O gráfico da Figura 4.2 compara o desempenho das arquiteturas. Observamos três grupos em desempenho. O primeiro grupo formado pelas soluções em GPU, se-

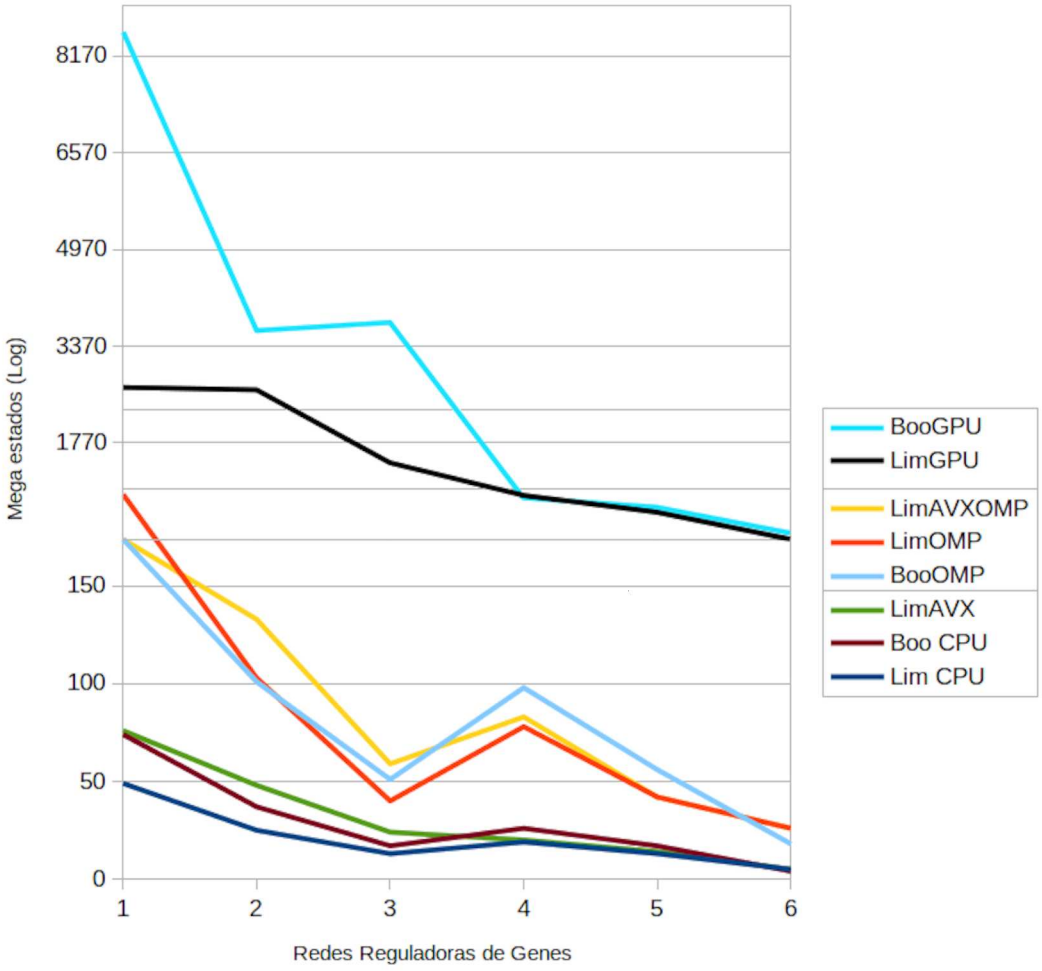


Figura 4.2. Desempenho das arquiteturas.

guindo pelo grupo das implementações com OpenMP e com o desempenho pior as implementações em CPU, mesmo com a vetorização gerada pelo compilador GCC.

A versão limiar é melhor em 43% das vezes que a booleana. Tendo um diferencial mais bem importante, a versão limiar pode ser facilmente adaptada para ter um comportamento dinâmico no qual pode-se explorar variações nos pesos para ajustar uma rede biológica que está sendo modelada e comparada com dados experimentais [Davidson & Levin, 2005]. A versão limiar que é vetorizada pelo compilador tem um desempenho semelhante à versão booleana sem vetorização, enquanto a versão booleana não foi automaticamente vetorizada pelo compilador GCC. A versão limiar vetorizada melhora o número de instruções geradas, mas que nem sempre repercute de forma positiva no desempenho do cálculo de atratores.

4.3 Cálculo dos atratores

Nesta seção realizamos o experimento com o cálculo de atratores.

Inicialmente iremos descrever a rede biológica CAC[Lu et al., 2015], que é uma representação booleana do câncer de cólon associado à colite CAC (pelas siglas em inglês). O grafo da rede está ilustrado na Figura 4.3.

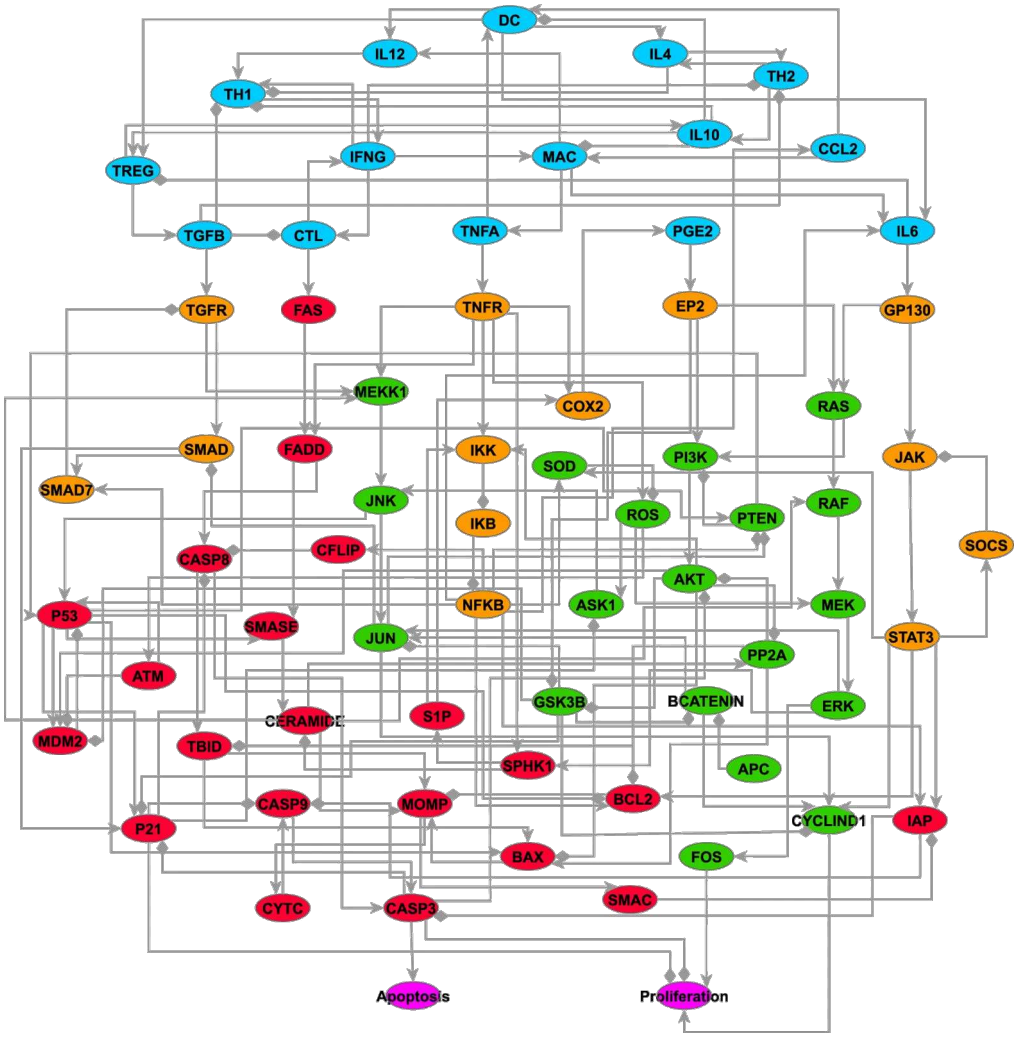


Figura 4.3. Representação do câncer de cólon associado à colite.

Toda a rede CAC tem 70 nós e 153 arestas. Que pode ser dividido em duas partes: a parte das células epiteliais intestinais, que contém nós que representam componentes de sinalização intracelular e a parte do microambiente imune, que contém os nós que representam células imunes, citocinas e quimiocinas. Também nesse trabalho modelaram *Proliferation* e *Apoptosis* como dois nós de saída, para resumir os efeitos biológicos finais da sinalização inflamatória.

Mas considerando que o tamanho do espaço de estado é 2^{70} e que a identificação do atrator é um problema de NP-completo, rastrear todos os atratores é inviável. Uma solução é a simplicação do modelo com uma nova rede, como proposto em [Saadatpour et al., 2010], mantendo o comportamento a longo prazo do modelo dinâmico. A rede resultante tem 21 nós com 39 arestas incluindo o nó *Proliferation*.

Para compara o desempenho da ferramenta CyTLFgraph usaremos a ferramenta proposta em [Lu et al., 2015], denominada **SimpleBol**. **SimpleBol** é um pacote em python para simulação e analises de modelos booleanos dinâmicos, semelhante ao **Booleanet**, porém é bem mais simples para o usuário sem experiencia em programação.

	SimplesBool	CyTLFGraph
GUI		X
Busca de Atratores	X	X
Percorrido Exaustivo	X	X
Transição dos estados	X	
Transição dos atratores		X
Representação booleana	X	X
Método síncrono	X	X
Método assíncrono	X	
Perturbações	X	X

Tabela 4.7. Comparação entra SimplesBool e CyTLFGraph

A Tabela 4.7 compara o CyTLFGraph e o SimpleBol. Ambos têm características muito semelhantes, como fácil uso e entendimento dos processos, que possuem gráficos booleanos com nomenclatura semelhante como entrada, busca de atratores e até realizam estudos sistemáticos de perturbações de nós, podendo ser estas únicas e múltiplas.

Usando a mesma rede biológica de 21 nós, foram testadas algumas caraterísticas das ferramentas, como se mostra na Tabela 4.8.

	SimpleBool	CYTLFGraph	
		CPU	GPU
tempo (seg)	71,32	6,53	2,28
atratores	24*	12	14*
estados visitados	2097152	2097152	2097152

Tabela 4.8. Analises de desempenho entre SimplesBool e o CyTLFGraph

A Tabela mostra o ganho no tempo de execução do **CyTLFGraph** é de $11x$, foi analisado a mesma quantidade de estados iniciais, 2097152 que corresponde com

o numero total de estados que o grafo pode alcançar, 21 nós $2^{21} = 2097152$, os dados confrontados são das execuções nas arquiteturas CPU e GPU.

Vale destacar que o **SimplesBool** realiza uma busca dos atratores de forma peculiar, ele pode sim, percorrer de forma exaustiva todos os estados do grafo como estado inicial para achar um atrator, porém o usuário é quem determina quantos passos o sistema da a partir do punto de saída, além de determinar quantas rodadas de teste fará. Portanto, para redes booleanas grandes, o uso de todos os estados iniciais possíveis pode ser extremamente lento ou ter problemas de memória.

Na Tabela 4.8, também pode-se ver que na ferramenta **SimplesBool** o calculo de atratores é maior, mas isso acontece por conta que o algoritmo não detecta quando por cada rodada se acha um atrator se este existe, já que sendo os atratores, as vezes, um ciclo de estados, nem sempre o ponto de entrada deles vai ser o mesmo, por isso na saída da solução podemos achar atratores repetidos, no caso testado o número real do atratores achados depois de ser feita uma limpeza nos dados são 12. Na implementação em GPU, cada *thread* calcula um atrator de forma simultânea, podem ocorrer repetições que são posteriormente filtradas.

A Figura 4.4 mostra os 12 atratores do resultado da simulação no CyTLFGraph, destacando a visualização do grafo que representa os atratores, os nós desse grafo representam os valores dos estados, convertidos em hexadecimal, permitindo uma melhor visualização das diferenças dos genes.

Histogramas dos atratores também é apresentado pela aplicação, veja Figura 4.5, mostrando caraterísticas simples como o tamanho dos atratores e e a distribuição dos estados por atrator.

As pertubações e clones Também foram testadas com a possibilidade de fazer pequenas mudanças na estrutura da rede. O CAC com 21 nós, teve o valor fixado do gene ($P21$) a 1, representando a garantia de permaneça do mesmo no sistema, para ver o comportamento, a estabilidade e a robustez da rede.

	SimpleBool	CYTLFGraph	
		CPU	GPU
tempo (seg)	31,75	7,35	3,25
atratores	24*	2	2
estados visitados	1048576	2097152	2097152

Tabela 4.9. Desempenho de uma mutação simulada no SimpleBool e CyTLFGraph

A Tabela 4.9 mostra o resultado da simulação após uma mudança no comportamento da rede, alterando o valor de um gene a 1. A ferramenta SimpleBool apaga

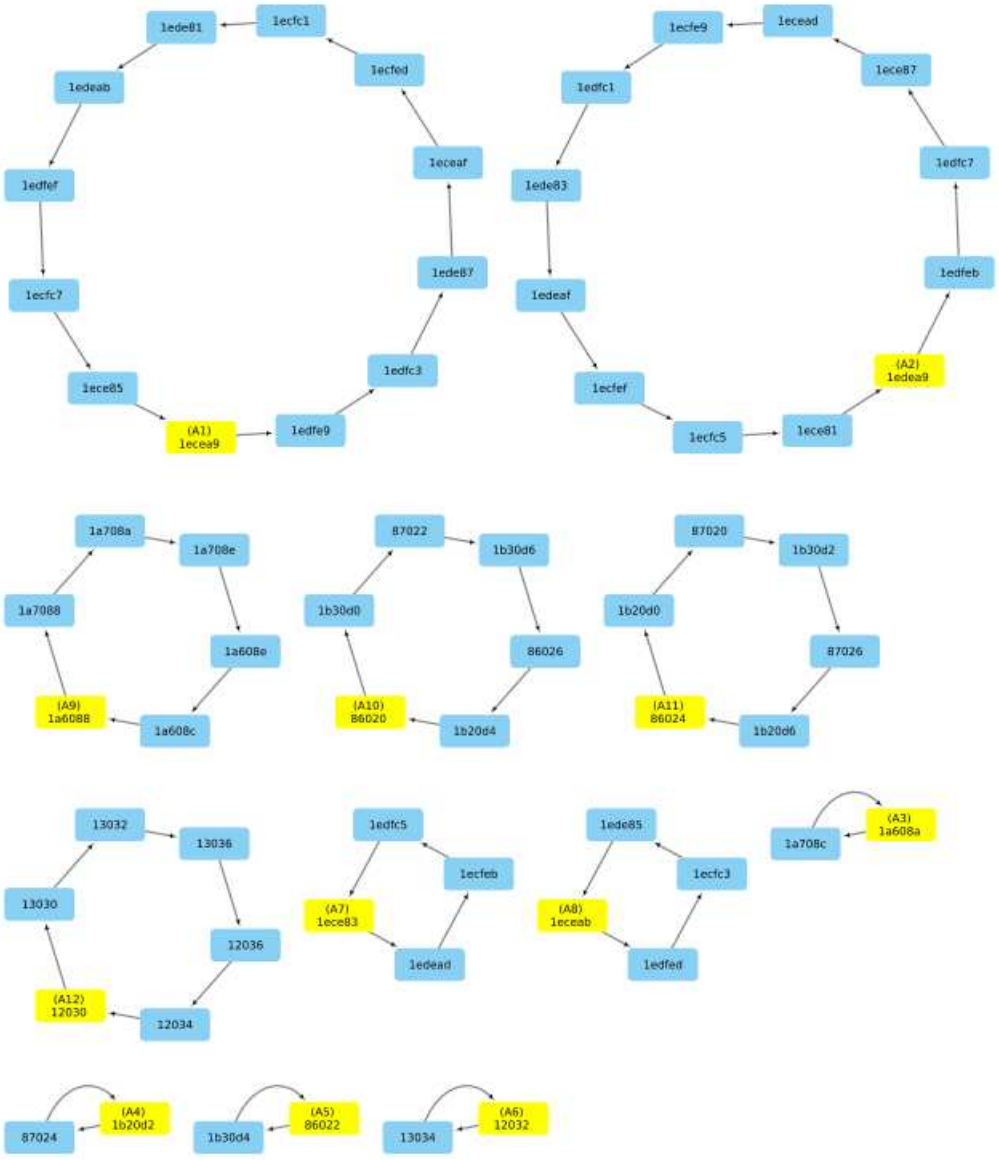
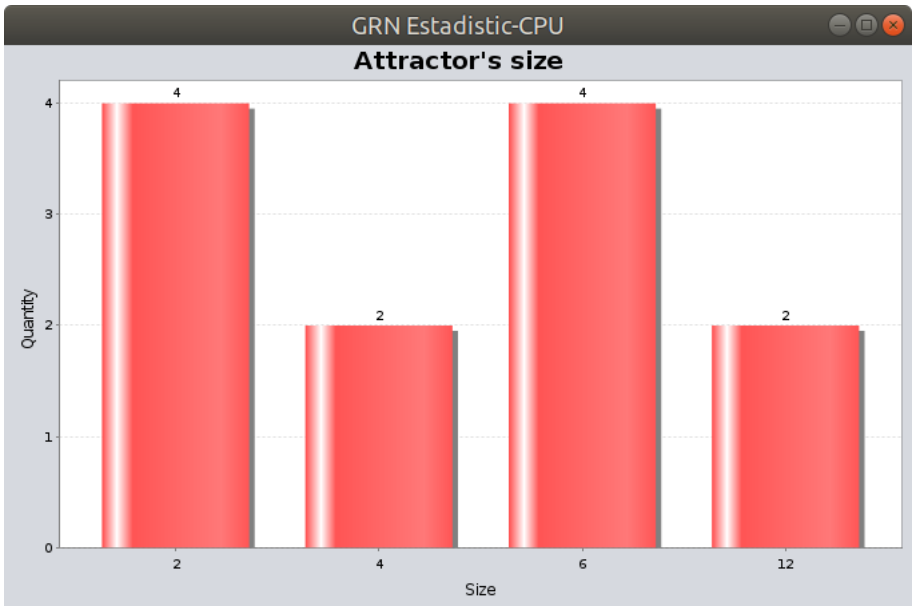


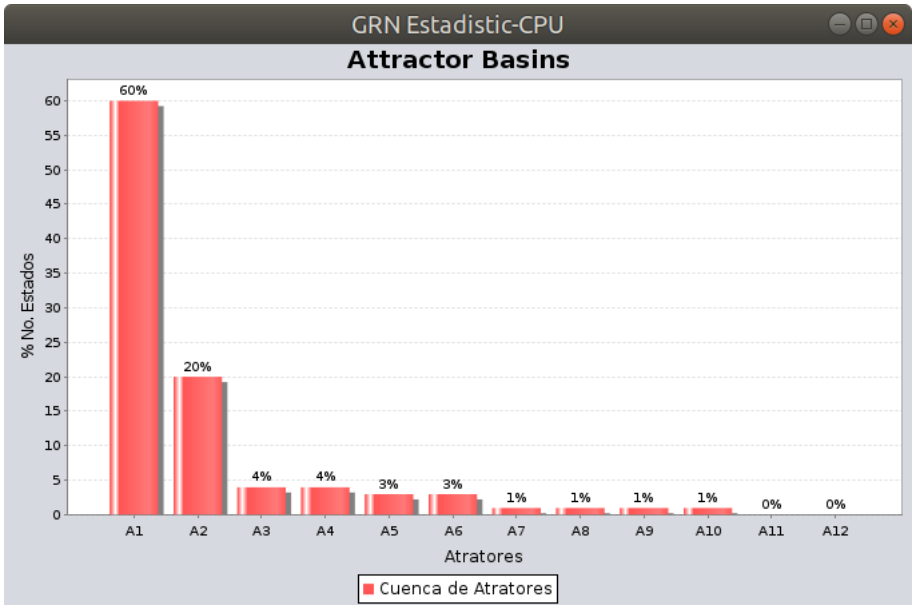
Figura 4.4. Resultado da simulação da rede no CyTLFGraph

o gene da simulação o que leva ao calculo de $2^20 = 1048576$ estados iniciais, também o numero de atratores varia notavelmente. Já a perturbação no aplicativo CyTLFGraph pode ser avaliada como uma mudança observando o número e caraterísticas dos atratores.

A Figura 4.6 mostra novamente os histogramas do CyTLFGraph para a simulação da perturbação, onde se observa que os dois atratores tem o mesmo tamanho, porém um deles tem uma maior bacia de atração.

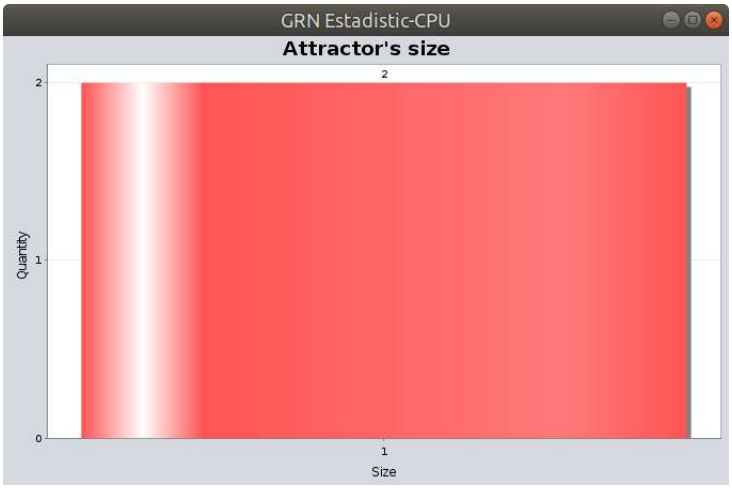


(a)

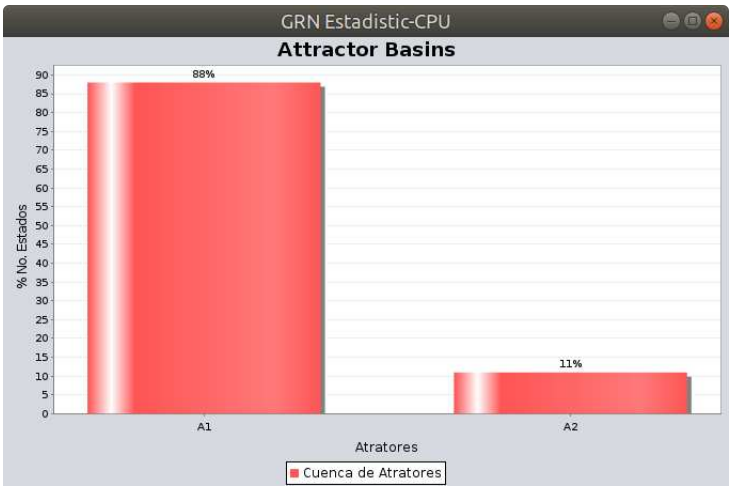


(b)

Figura 4.5. (a) Histograma das bacias dos Atratores.(b) Histograma de tamanho dos Atratores



(a)



(b)

Figura 4.6. Resultado da simulação da perturbação (a) Estatística das bacias dos Atratores.(b) Estatística de tamanho dos Atratores

Capítulo 5

CONCLUSÕES GERAIS E TRABALHOS FUTUROS

Neste trabalho foi apresentada uma nova ferramenta para o analises de redes de regulação biológica, integrada na plataforma Cytoscape e aproveitando o potencial de paralelismo oferecido pela programação paralela, já seja na CPU usando $OpenMP$, vetorização AVX, ou no uso da GPU, acelerando as simulações, em até 4x as implementações de $OpemMP$ e AVX, em relação à implementação mais simples da CPU com apenas 1 *thread* de execução e em 50x com a versão GPU, o que ajuda na ampliação do espaço de busca dos atratores.

O uso de uma abordagem limiar para o trabalho e cálculo das funções de atualização dos nós é um facilitador para a exploração do usuário na dinâmica das redes biológicas, com só ajustes nos pesos, sem a necessidade de recompilação para testar o comportamento do organismo. Uma aplicação é a avaliação de pertubações que podem alterar a estrutura da rede seguida da avaliação dos clones. O uso da programação paralela diminui o tempo de processamento, aumenta o número de estados atingidos nas análises das redes reguladoras.

A vinculação de uma ferramenta como o Cytoscape, neste trabalho, possibilita a expansão e a massividade do uso desta abordagem, centralizando o estudo e análises dos GRNs numa plataforma que tem um conjunto de profissionais, em crescimento, desenvolvendo e dando suporte cada dia.

5.1 Trabalhos Futuros

Como trabalhos futuros temos a ampliação de um grupo de funcionalidades na ferramenta, como o estudo de comportamento de cada nó da rede, computando as

mudanças de valores (quantas vezes ativo ou inativo) ao longo da simulação. A incorporação de analises para determinar o regime de execução da rede, para ter uma visão da transição entre entre as fases em um regime critico, usando os gráficos de Derrida.

Como parte da ampliação de recursos, uma proposta é vincular a ferramenta com os recursos oferecido pelo Serviço Web da Amazon (AWS), como o processamento na nuvem usando instancias de computadores com maior desempenho, ou a hospedagem do aplicativo na nuvem para ter um acesso via API REST, usando a ferramenta de CyREST, trazendo um maior desacoplamento da solução proposta.

https://www.overleaf.com/project/5d2f244811a5bf571dd4af1d

Referências Bibliográficas

Albert, I.; Thakar, J.; Li, S.; Zhang, R. & Albert, R. (2008). Boolean network simulations for life scientists. *Source code for biology and medicine*, 3(1):16.

Albert, R. & Othmer, H. G. (2003). The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in drosophila melanogaster. *Journal of theoretical biology*, 223(1):1--18.

Aldana, M. (2003). Boolean dynamics of networks with scale-free topology. *Physica D: Nonlinear Phenomena*, 185(1):45--66.

Batagelj, V. & Mrvar, A. (1998). Pajek program for large network analysis. connection. *Project home page at http://vlado. fmf. uni-lj. si/pub/networks/pajek*.

Bhattacharjya, A. & Liang, S. (1996). Median attractor and transients in random boolean nets. *Physica D: Nonlinear Phenomena*, 95(1):29--34.

Board, O. A. R. (2019). Openmp enabling hpc since 1997. In *OpenMP enabling HPC since 1997*.

Bock, M.; Scharp, T.; Talnikar, C. & Klipp, E. (2014). Boolesim: an interactive boolean network simulator. *Bioinformatics*, 30(1):131--132.

Borelli, F. F.; de Camargo, R. Y.; Martins, D. C. & Rozante, L. C. (2013). Gene regulatory networks inference using a multi-gpu exhaustive search algorithm. *BMC bioinformatics*, 14(18):S5.

Campos, R. R.; Ferreira, R.; Vendramini, J. C. G.; Martins, M. L. et al. (2011). Simulation of scale free gene regulatory networks based on threshold functions on gpu. In *2011 Simpasio em Sistemas Computacionais*, pp. 11--11. IEEE.

Carballido, J. A.; Ponzoni, I. & Gallo, C. A. (2009). Computación evolutiva y aprendizaje automático para la inferencia, modelado y simulación de redes regulatorias de genes. In *XI Workshop de Investigadores en Ciencias de la Computación*.

Carlin, D. E.; Demchak, B.; Pratt, D.; Sage, E. & Ideker, T. (2017). Network propagation in the cytoscape cyberinfrastructure. *PLoS computational biology*, 13(10):e1005598.

Chaos, A.; Aldana, M.; Espinosa-Soto, C.; de León, B. G. P.; Arroyo, A. G. & Alvarez-Buylla, E. R. (2006). From genes to flower patterns and evolution: dynamic models of gene regulatory networks. *Journal of Plant Growth Regulation*, 25(4):278--289.

Cheng, J.; Grossman, M. & McKercher, T. (2014). . John Wiley & Sons.

Christensen, T. S.; Oliveira, A. P. & Nielsen, J. (2009). Reconstruction and logical modeling of glucose repression signaling pathways in saccharomyces cerevisiae. *BMC systems biology*, 3(1):7.

Consortium, C. (2019). Cytoscape. In *Cytoscape*.

Coorporation, N. (2019). Cuda toolkit. In *CUDA Toolkit*.

Cumbo, F.; Paci, P.; Santoni, D.; Di Paola, L. & Giuliani, A. (2014). Giant: a cytoscape plugin for modular networks. *PloS one*, 9(10):e105001.

da Silva, L. B.; Almeida, D.; Nacif, J. A. M.; Sánchez-Osorio, I.; Hernández-Martínez, C. A. & Ferreira, R. (2017). Exploring the dynamics of large-scale gene regulatory networks using hardware acceleration on a heterogeneous cpu-fpga platform. In *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1--7. IEEE.

Darabos, C.; Di Cunto, F.; Tomassini, M.; Moore, J. H.; Provero, P. & Giacobini, M. (2011). Additive functions in boolean models of gene regulatory network modules. *PloS one*, 6(11):e25110.

Davidson, E. & Levin, M. (2005). Gene regulatory networks. *Proceedings of the National Academy of Sciences*, 102(14):4935--4935.

Derrida, B. & Pomeau, Y. (1986). Random networks of automata: a simple annealed approximation. *EPL (Europhysics Letters)*, 1(2):45.

Ferreira, R. & Vendramini, J. C. G. (2010). Fpga-accelerated attractor computation of scale free gene regulatory networks. In *2010 International Conference on Field Programmable Logic and Applications*, pp. 550--555. IEEE.

Franke, R.; Theis, F. J. & Klamt, S. (2010). From binary to multivalued to continuous models: the lac operon as a case study. *Journal of integrative bioinformatics*, 7(1):172--190.

Fumia, H. F. & Martins, M. L. (2013). Boolean network model for cancer pathways: predicting carcinogenesis and targeted therapy outcomes. *PloS one*, 8(7):e69008.

Garg, A.; Di Cara, A.; Xenarios, I.; Mendoza, L. & De Micheli, G. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917--1925.

Gou, C. & Gaydadjiev, G. N. (2013). Addressing gpu on-chip shared memory bank conflicts using elastic pipeline. *International Journal of Parallel Programming*, 41(3):400--429.

Helikar, T.; Konvalina, J.; Heidel, J. & Rogers, J. A. (2008). Emergent decision-making in biological signal transduction networks. *Proceedings of the National Academy of Sciences*, 105(6):1913--1918.

Huang, S. (2002). Rational drug discovery: what can we learn from regulatory networks? *Drug Discovery Today*, 7(20):s163--s169.

Jacob, V. V.; de Resende Ferreira, C. & Ferreira, R. (2012). Gpu optimization techniques applied to scale free gene regulatory networks based on threshold function. In *2012 13th Symposium on Computer Systems*, pp. 57--64. IEEE.

Kaneko, K. (2006). . Springer.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437--467.

Kauffman, S. A. (1993). . OUP USA.

Kerkhofs, J. & Geris, L. (2015). A semiquantitative framework for gene regulatory networks: increasing the time and quantitative resolution of boolean networks. *PloS one*, 10(6):e0130033.

Kervizic, G. & Corcos, L. (2008). Dynamical modeling of the cholesterol regulatory pathway with boolean networks. *BMC systems biology*, 2(1):99.

Kitano, H. (2002). Computational systems biology. *Nature*, 420(6912):206.

Kitano, H. (2004). Biological robustness. *Nature Reviews Genetics*, 5(11):826--837.

Krause, F.; Schulz, M.; Ripkens, B.; Flöttmann, M.; Krantz, M.; Klipp, E. & Handorf, T. (2013). Biographer: web-based editing and rendering of sbgn compliant biochemical networks. *Bioinformatics*, 29(11):1467--1468.

Li, F.; Long, T.; Lu, Y.; Ouyang, Q. & Tang, C. (2004). The yeast cell-cycle network is robustly designed. *Proceedings of the National Academy of Sciences*, 101(14):4781--4786.

Longabaugh, W. J.; Davidson, E. H. & Bolouri, H. (2005). Computational representation of developmental genetic regulatory networks. *Developmental biology*, 283(1):1--16.

Lu, J.; Zeng, H.; Liang, Z.; Chen, L.; Zhang, L.; Zhang, H.; Liu, H.; Jiang, H.; Shen, B.; Huang, M. et al. (2015). Network modelling reveals the mechanism underlying colitis-associated colon cancer and identifies novel combinatorial anticancer targets. *Scientific reports*, 5:14739.

Mai, Z. & Liu, H. (2009). Boolean network-based analysis of the apoptosis network: irreversible apoptosis and stable surviving. *Journal of Theoretical Biology*, 259(4):760--769.

Manica, M.; Polig, R.; Purandare, M.; Mathis, R.; Hagleitner, C. & Rodríguez Martínez, M. (2019). Accelerated analysis of boolean gene regulatory networks via reconfigurable hardware. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–1. ISSN .

Mendoza, L. & Méndez, A. (2015). A dynamical model of the regulatory network controlling lymphopoiesis. *Biosystems*, 137:26--33.

Montojo, J.; Zuberi, K.; Rodriguez, H.; Kazi, F.; Wright, G.; Donaldson, S. L.; Morris, Q. & Bader, G. D. (2010). Genemania cytoscape plugin: fast gene function predictions on the desktop. *Bioinformatics*, 26(22):2927--2928.

Morris, J. H.; Apeltsin, L.; Newman, A. M.; Baumbach, J.; Wittkop, T.; Su, G.; Bader, G. D. & Ferrin, T. E. (2011). clustermaker: a multi-algorithm clustering plugin for cytoscape. *BMC bioinformatics*, 12(1):436.

Morris, M. K.; Saez-Rodriguez, J.; Sorger, P. K. & Lauffenburger, D. A. (2010). Logic-based models for the analysis of cell signaling networks. *Biochemistry*, 49(15):3216--3224.

Mozaffari, S. N.; Tragoudas, S. & Haniotakis, T. (2017). A new method to identify threshold logic functions. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 934--937. IEEE.

Muroga, S. (1971). Threshold logic and its applications. *Center of Digital philosophy*.

Müssel, C.; Hopfensitz, M. & Kestler, H. A. (2010). Boolnet—an r package for generation, reconstruction and analysis of boolean networks. *Bioinformatics*, 26(10):1378--1380.

Neutzling, A.; Martins, M. G.; Ribas, R. P. & Reis, A. I. (2013). An efficient method to threshold logic functions identification. In *28TH SOUTH SYMPOSIUM ON MICROELECTRONICS. Proceedings*. sn.

Ono, K.; Muetze, T.; Kolishovski, G.; Shannon, P. & Demchak, B. (2015). Cyrest: Turbocharging cytoscape access for external tools via a restful api. *F1000Research*, 4.

Paroni, A.; Graudenzi, A.; Caravagna, G.; Damiani, C.; Mauri, G. & Antoniotti, M. (2016). Cabernet: a cytoscape app for augmented boolean models of gene regulatory networks. *BMC bioinformatics*, 17(1):64.

Portocarrero Tovar, C. R.; Araújo, E.; Carastan-Santos, D.; Martins, D. C. & Rozante, L. (2019). Finding attractors in biological models based on boolean dynamical systems using hitting set. In *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*, pp. 235–239. ISSN 2159-5410.

Romao, O. C.; de Souza Amorim, L. E.; Ferreira, R. S.; de Araujo Possi, M. & de Paiva Oliveira, A. (2012). Multiagent systems modeling using gpus–a case study of the human immune system. In *2012 13th Symposium on Computer Systems*, pp. 234--241. IEEE.

Rosa, W.; Baranda, H.; Canesche, M.; Menezes, M.; Bragança, L.; Magalhaes, S.; Nacif, J. A. & Ferreira, R. (2019). Simulação de redes reguladoras de genes com lógica booleana e limiar em plataformas alto desempenho. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pp. 334--345. SBC.

Ruz, G. A.; Goles, E.; Montalva, M. & Fogel, G. B. (2014). Dynamical and topological robustness of the mammalian cell cycle network: A reverse engineering approach. *Biosystems*, 115:23--32.

Saadatpour, A.; Albert, I. & Albert, R. (2010). Attractor analysis of asynchronous boolean models of signal transduction networks. *Journal of theoretical biology*, 266(4):641--656.

Saadatpour, A.; Wang, R.-S.; Liao, A.; Liu, X.; Loughran, T. P.; Albert, I. & Albert, R. (2011). Dynamical and structural analysis of a t cell survival network identifies novel candidate therapeutic targets for large granular lymphocyte leukemia. *PLoS computational biology*, 7(11):e1002267.

Sato, M. (2002). Openmp: parallel programming api for shared memory multiprocessors and on-chip multiprocessors. In *15th International Symposium on System Synthesis, 2002.*, pp. 109--111. IEEE.

Shannon, C. E. (1938). A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713--723.

Shannon, P.; Markiel, A.; Ozier, O.; Baliga, N. S.; Wang, J. T.; Ramage, D.; Amin, N.; Schwikowski, B. & Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498--2504.

Shi, X.; Banerjee, S.; Chen, L.; Hilakivi-Clarke, L.; Clarke, R. & Xuan, J. (2017). Cynetsvm: A cytoscape app for cancer biomarker identification using network constrained support vector machines. *PloS one*, 12(1):e0170482.

Shmulevich, I.; Dougherty, E. R.; Kim, S. & Zhang, W. (2002). Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261--274.

Technology, O. (2019). The dynamic module system for java. In *The Dynamic Module System for Java*.

Thomas, R. (1973). Boolean formalization of genetic control circuits. *Journal of theoretical biology*, 42(3):563--585.

Trinh, H.-C.; Le, D.-H. & Kwon, Y.-K. (2014a). Panet: a gpu-based tool for fast parallel analysis of robustness dynamics and feed-forward/feedback loop structures in large-scale biological networks. *PloS one*, 9(7):e103010.

Trinh, H.-C.; Le, D.-H. & Kwon, Y.-K. (2014b). Panet: a gpu-based tool for fast parallel analysis of robustness dynamics and feed-forward/feedback loop structures in large-scale biological networks. *PloS one*, 9(7).

Truong, C.-D.; Tran, T.-D. & Kwon, Y.-K. (2016). Moro: a cytoscape app for relationship analysis between modularity and robustness in large-scale biological networks. *BMC systems biology*, 10(4):122.

Yamaguchi-Shinozaki, K. & Shinozaki, K. (2006). Transcriptional regulatory networks in cellular responses and tolerance to dehydration and cold stresses. *Annu. Rev. Plant Biol.*, 57:781--803.