Sincronismo e Estabilidade em redes dinâmicas Discretas Acopladas - Blog

Carlos Reynaldo Portocarrero Tovar

2018

1 Resume

Os processos de estabilização e sincronização em redes de entidades dinâmicas interagentes são quase onipresentes na natureza e desempenham um papel muito importante em muitos contextos diferentes, da biologia à sociologia. Redes Dinâmicas Discretas Acopladas (RDDA) são uma classe de modelos para redes de entidades dinâmicas interagentes, que incluem redes Boolianas acopladas, e que apresentam um amplo leque de potenciais aplicações, principalmente em Biologia de Sistemas. Apesar da sua importância, existem relativamente poucos estudos focados em estabilidade e sincronização envolvendo essa classe específica de modelos, em particular estudos baseados em abordagens computacionais. O objetivo desse projeto consiste em desenvolver um método computacionalmente eficiente que, dada uma RDDA como entrada, seja capaz de responder se ela contém ou não estados de sincronização estáveis, bem como identificá-los.

2 Conceitos iniciais

2.1 Redes Dinâmicas Discretas

Uma Rede Dinâmica Discreta (RDD) e uma representação de

Uma Rede Dinâmica Discreta (RDD) esta composta por um conjunto de variáveis $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ que assumem valores num domínio finito X e sejam as funções $F = [f_1, f_2, \ldots, f_n]$ tal que a função $f_i : X^n \to X$ determina o estado da variável x_i. Se o domínio finito X é Booliano $0, 1$ e as funções f_i são aplicadas simultaneamente, chamamos esse sistema de Rede Booliana Síncrona. A dinâmica de uma Rede Booliana pode ser representada pelo *Grafo de Transições de Estados* (GTE) $G = (V, E)$, onde $V = X^n$ e $(x, y) \in E$ se e somente se $F^j(x) = y$. Dizemos que $x \in X^n$ é um *atrator pontual* (ou *ponto fixo* ou *estado estacionário*) se $F^j(x) = x$. Se existe algum $k \geq 1$ tal que $(F^j)^k(x) = x$, então dizemos que x é *periódico* e denominamos o conjunto $\{x, (F^j)^1(x), (F^j)^2(x), \ldots, (F^j)^k(x)\}$ de *atrator periódico*.

2.2 Problema de Satisfazibilidade Booliana - *SAT*

Na teoria da complexidade computacional, o problema de satisfazibilidade booliana (SAT) foi o primeiro problema identificado como pertencente à classe de complexidade NP-completo. O problema de satisfazibilidade booliana é o problema de determinar se existe uma determinada valoração para as variáveis de uma determinada fórmula booliana tal que esta valoração satisfaça esta fórmula em questão. Por exemplo, tomando x_1, x_2, x_3, x_4 como as variáveis boolianas e a expressão $(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$, caso exista uma atribuição de valores de verdade para as variáveis da fórmula que torne a fórmula avaliada VERDADEIRA, esta fórmula é considera satisfazível, em contrapartida se nenhuma atribuição levou a uma avaliação da fórmula como verdadeira, ela é considerada insatisfazível.

2.3 Forma Normal Conjuntiva - *CNF*

Na lógica booliana, uma fórmula está na forma normal conjuntiva (CNF) se é uma conjunção de cláusulas, onde uma cláusula é uma disjunção de literais. Qualquer formula booliana pode-se converter a CNF, um exemplo disto é:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \tag{1}$$

Minterminos e Maxiterminos são utilizados para reescrever uma função lógica em uma forma padronizada no sentido de obter-se uma simplificação da mesma. Esta simplificação é traduzida na redução do número de portas do circuito lógico que implementa tal função. Isto é feito através de manipulação algébrica da função lógica, claro, sem alteração do valor lógico da mesma. A função é escrita na forma de uma Soma de Produtos (Minterminos), onde cada termo possui todas as variáveis (A, B e C) complementadas (negadas) ou não, ou na forma de um Produto de Somas (Maxiterminos), onde cada fator contém a soma de todas as variáveis (complementadas ou não). Há uma exigência de que todas as variáveis devem aparecer em cada produto (no caso de Minterminos) e em cada soma (no caso dos Maxiterminos).

- Minterminos - são termos somente com AND (termos PRODUTO).
- Maxiterminos - são termos somente com OR (termos SOMA).

2.4 Funcionamento do MiniSAT

A maioria de SAT-Solvers o que fazem e procurar os valores das variáveis que pertencem a uma clausula, que fazem a esta clausula ser Falso o Negativo e dizer Não-Satisfazível, lembrando que a função booliana debe estar no 3-CNF. Uma veis achados os valores que fazem a clausula Não-Satisfazível, este valor e jogado fora da solução, e é criado um arvore com as ramas dos valores que fazem Satisfazível a clausula, e de novo e avaliada do mesmo jeito a seguinte clausula gerando um arvore de possíveis caminhos os quais podem ser paralelizados e quem achar a solução primeiro, mostra sua resposta e o algoritmo, termina. [1]

$FB = (X_1 \vee \neg X_2 \vee X_3) \wedge (X_4 \vee X_5 \vee \neg X_6) \wedge (\neg X_1 \vee X_7 \vee X_8) X1 = 0, X2 = 1, X3 = 0$

2.5 O Problema *Hitting Set*

Dada uma coleção E de subconjuntos de V, o problema do *Hitting Set* é encontrar o subconjunto $S \subseteq V$ de tamanho K que intercepta (acerta) cada conjunto em E. Se considerarmos que E define um hiper-grafo em V (onde cada conjunto em E constitui uma hiper-relação), então vemos que o problema do Hitting Set é equivalente ao problema de cobertura de vértices em hiper-grafos; esse problema é NP-difícil.

3 Artigo da Professora Duvrova

Neste artigo e apresentado um método para encontrar atratores numa rede biológica, esta rede tem a particularidade de que as transições genicas são representadas como funciones boolianas , esto gera as vezes que a transições fiquem em estados repetitivos , os que chamamos de "atratores", a gente pode ver na figura uma representação da rede biológica e um gráfico das transições.

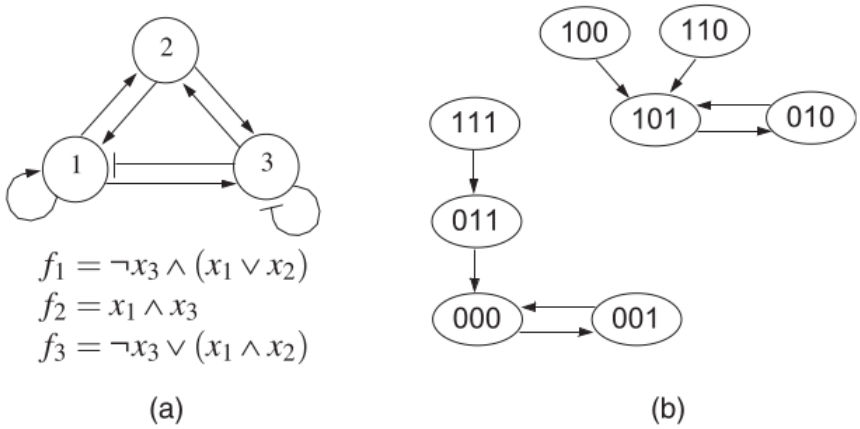

Fig. 1. (a) A Boolean network with three nodes. Arrows indicate activatory regulation and blunt-ends represent inhibitory regulation. (b) Its state-transition graph. Each state is of type (x_1, x_2, x_3).

3.1 Descrição do Método

O artigo descreve um algoritmo que se basa em gerar uma sequencia de transições F de estado na forma de operações boolianas, depois calculamos mediante SAT se existe algum conjunto de estados que satisfaz F e sim estos estados que são retornados pelo SAT-solver encontra-se dos estados iguais ,

isso quer diz que foi encontrado um atractor. Aqui podemos observar como se construí a sequencia de transições F.

$$\begin{aligned} T(s, s^+) &= (x_1^+ \leftrightarrow \neg x_3 \wedge (x_1 \vee x_2)) \wedge (x_2^+ \leftrightarrow x_1 \wedge x_3) \\ &\quad \wedge (x_3^+ \leftrightarrow \neg x_3 \vee (x_1 \wedge x_2)). \\ T(s^+, s^{++}) &= (x_1^{++} \leftrightarrow \neg x_3^+ \wedge (x_1^+ \vee x_2^+)) \wedge (x_2^{++} \leftrightarrow x_1^+ \\ &\quad \wedge x_3^+) \wedge (x_3^{++} \leftrightarrow \neg x_3^+ \vee (x_1^+ \wedge x_2^+)). \end{aligned}$$

So, the unfolding $T(s, s^+) \wedge T(s^+, s^{++})$ is

$$\begin{aligned} &(x_1^+ \leftrightarrow \neg x_3 \wedge (x_1 \vee x_2)) \wedge (x_2^+ \leftrightarrow x_1 \wedge x_3) \wedge (x_3^+ \leftrightarrow \neg x_3 \\ &\quad \vee (x_1 \wedge x_2)) \wedge (x_1^{++} \leftrightarrow \neg x_3^+ \wedge (x_1^+ \vee x_2^+)) \wedge (x_2^{++} \leftrightarrow x_1^+ \wedge x_3^+) \\ &\quad \wedge (x_3^{++} \leftrightarrow \neg x_3^+ \vee (x_1^+ \wedge x_2^+)). \end{aligned}$$

Depois de isto os estados atratores são guardados e são retirados da sequencia de transições F usando uma propriedade booliana , que faz já não aceitar a solução inicial do SAT-solver. Quando o SAT-solver não pode achar um novo caminho então não pode satisfazer F que quer dizer que já achou todos os atratores. Podemos ver o Conjunto dos atractores ,expressado nesta forma.

$$\begin{aligned} A(s^0) = &((x_1^0 \leftrightarrow 0) \wedge (x_2^0 \leftrightarrow 0) \wedge (x_3^0 \leftrightarrow 0)) \\ &\vee ((x_1^0 \leftrightarrow 0) \wedge (x_2^0 \leftrightarrow 0) \wedge (x_3^0 \leftrightarrow 1)). \end{aligned}$$

After finding the second attractor $(101, 010)$, $A(s^0)$ becomes

$$\begin{aligned} A(s^0) = &((x_1^0 \leftrightarrow 0) \wedge (x_2^0 \leftrightarrow 0) \wedge (x_3^0 \leftrightarrow 0)) \\ &\vee ((x_1^0 \leftrightarrow 0) \wedge (x_2^0 \leftrightarrow 0) \wedge (x_3^0 \leftrightarrow 1)) \\ &\vee ((x_1^0 \leftrightarrow 1) \wedge (x_2^0 \leftrightarrow 0) \wedge (x_3^0 \leftrightarrow 1)) \\ &\vee ((x_1^0 \leftrightarrow 0) \wedge (x_2^0 \leftrightarrow 1) \wedge (x_3^0 \leftrightarrow 0)). \end{aligned}$$

O algoritmo começa especificando o tamanho dos nodos n que representam o cumprimento do caminho , depois temos uma variável onde guarda se encontro um atractor ou não uma variável que salvara o estado da busca (se for ou não encontrado um atrator), depois declara a variável F onde gera e guarda a sequencia dos estados de transição, apos disto tem um bloco repetitivo *while* que vai se repetir até não encontrar mais caminhos, o que é, até achar todos os estados atratores, primeiramente ele procura estados iguais a o topo do caminho , e se consegue encontrá-los salva todos os estados intermédios na repetição (esses são parte do atrator). Depois disso tira os estados atratores do F caso não encontrar um atractor dobra o tamanho da transição, uma vez que foram

encontrados todos os atratores, a sequência de transição F ao ser avaliada pelo SAT-solver da resultado falso , isso quer dizer que já não tem mas atratores então termina o algoritmo.

Algorithm 1. An algorithm for finding attractors in a Boolean network with n nodes and the transition relation T

$i = n$
attractor_is_found = False
$A(s^0) = 0$ /* $A(s^0)$ is the characteristic function representing the set of all states of currently identified attractors expressed in terms of variables of the state s^0 */
$F = T_{-i...0}$ /* $T_{-i...0}$ is the unfolding of T from the time step $-i$ to the time step 0 defined by (2) */
while Sat(F) **do**
  for $(j = -1; j \geq -i; j--)$ **do**
    if $s^j = s^0$ **then**
      for $(p = 0; p > j; p--)$ **do**
        $A(s^0) = A(s^0) \vee (s^0 \leftrightarrow c_p)$ /* $c_p \in \{0,1\}^n$ is an assignment of the variables of the state s^p returned by the SAT-solver; $s^0 \leftrightarrow c_p$ is defined by (3) */
      end for
      $F = F \wedge \neg A(s^0)$
      attractor_is_found = True
      break
    end if
  end for
  if attractor_is_found **then**
    attractor_is_found = False
  else
    $F = F \wedge T_{-2*i...-i}$
    $i = 2 * i$
  end if
end while

O método parte da premissa que os estados dos vértices da RDD são deter-

minados por funciones boolianas, com os valores dos estados anteriores:

$$x_i^t = f(x_1^{t-1}, ..., x_n^{t-1}) \tag{2}$$

Para apresentar o funcionamento do método, usamos uma Rede Booliana Discreta de 3 vértices x_1, x_2, x_3, os quais interatuam mediante as seguintes funções de transição:

- $x_1^1 = (x_1^0 \vee x_2^0) \wedge (x_1^0 \vee x_3^0))$
- $x_2^1 = x_1^0 \vee x_2^0 \vee \neg x_3^0$
- $x_3^1 = \neg x_1^0 \vee \neg x_2^0 \vee \neg x_3^0$

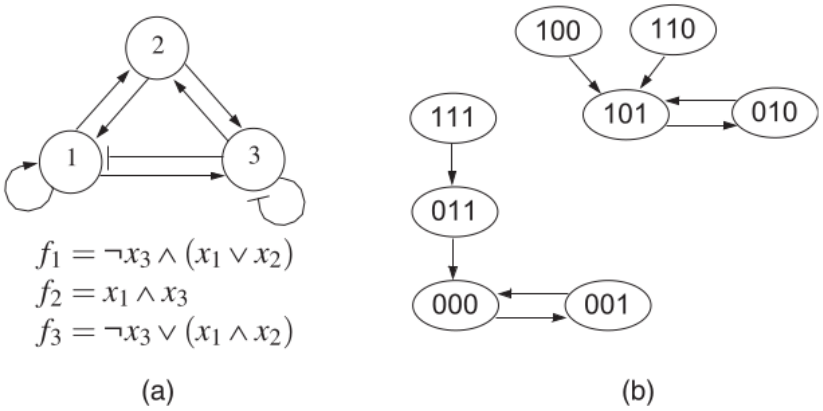

Fig. 1. (a) A Boolean network with three nodes. Arrows indicate activatory regulation and blunt-ends represent inhibitory regulation. (b) Its state-transition graph. Each state is of type (x_1, x_2, x_3).

Figura 1: Red Booliana

Para representar as transições de estados é gerada una gerada una Função Booliana F seguindo a seguinte formula:

$$T(0,1) = \tag{3}$$

Ao usar essa formula dentro de nosso exemplo gera a seguinte formula booliana F :

$$F_{0..1} = (X_1^1 \leftrightarrow ((X_1^0 \vee X_2^0) \wedge (X_1^0 \vee X_3^0))) \wedge (X_2^1 \leftrightarrow (X_1^0 \vee X_2^0 \vee \neg X_3^0)) \wedge (X_3^1 \leftrightarrow (\neg X_1^0 \vee \neg X_2^0 \vee \neg X_3^0))$$

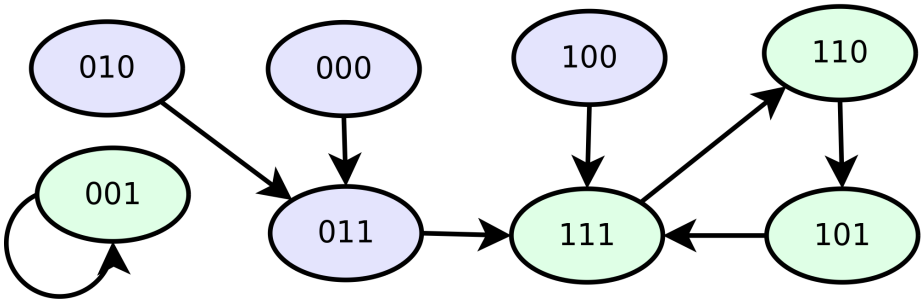

Figura 2: Gráfico de Transição de Estados de uma Rede Booliana Discreta

3.2 Algoritmo

Result: atractorsList
atractorsList = [ ] .....(1);
dataRed = funtions of Red.....(1);
numberTransitions = number of vertices .....(1);
booleanFunction = createBooleanFuntion(numeroTransiciones, dataRed, atractorsList).....(n*m);
estadosAtractores = HitingSetSolver(booleanFunction);
atractorsList = atractorsList union estadosAtractores ;
while *estadosAtractores != 0* **do**
  atractorsList = atractorsList union estadosAtractores;
  numberTransitions = number of vertices x 2 ;
  booleanFunction = createBooleanFuntion(numeroTransiciones, dataRed,atractorsList);
  estadosAtractores = HitingSetSolver(booleanFunction);
  atractorsList = atractorsList union estadosAtractores ;
end
print atractorsList;

Algorithm 1: Algoritmo para procurar Atratores usando Hitting Set Solver

4 Método para procurar Atractores em Redes Boolianas usando *Hitting Set Problem*

Nosso objetivo nesta parte e criar um algoritmo computacionalmente eficiente para procurar *Atractores* em Redes Boolianas usando um *Hitting Set Solver* paralelo [3] para resolver *Satisfazibilidade* dentro uma metodologia baseada em Funções Boolianas [4]. O método cria uma Função Booliana que descreve as transições entre os estados, depois as cláusulas são transformadas a subconjuntos para serem processados pelo *Hitting Set* Solver e resolver a Satisfazibilidade das variáveis, si tiver dentro das respostas, dois estados repetidos quer dizer que achamos um *atrator*. Esperamos reduzir a constante da Função de Com-

plexidade Computacional do Tempo, pois o problema de Procura de Atratores é NP-Completo.

4.1 Redução Polinomial dos Problemas *SAT* para *Hitting Set*

Os Problemas SAT e Hitting Set são problemas NP-Completos as entradas de um podem ser convertidas a em entradas do outro problema, isto e chamado de *Redução Polinomial*. O motivo desta parte e comprovar se do mesmo jeito que e usado o SAT pra achar caminhos no algoritmo do Duvrova, se pode usar um Hitting Set solver numa sequencia booliana para achar atratores. Dentro da revisão da documentação não se acho que Hitting Set poda modelar um problema de satibilidade, o que si foi encontrado foi uma redução polinomial feita entre vários tipos de problemas NP-completos, no ordem SAT -> 3SAT -> Vertex Cover -> Hitting Set o qual pode gerar o maior costo computacional e complexidade a la solução de achar atratores nas redes.O Hitting Set e Vertex Cover sao chamados de problemas equivalentes. Para poder trabalhar mais ordenadamente possível utilizaremos o exemplo do artigo do Duvrova que usa 3 variaveis boolianas e acharemos os atractores do jeito sistemático mas agora com um solucionador de Hitting Set.

4.2 Solução Planteada

Fazendo revisão de a bibliografia encontrou-se a redução polinomial entre os problemas 3-Sat ao Vertex Cover, o qual e feito gerando uma ligação entre as variables e as seus opostos, este principio foi usado para a conversao entre SAT a Hitting Set, esto dentro do SAT representaria $(x_1 \vee \neg x_1)$ o qual não modifica a Função Booliana F, esto a nivel do Hitting Set representaria un subconjunto adicional por cada variavel na entrada do problema, esto garante que a resposta minima do hitting set seja o numero de variaveis.

O primeiro passo e converter as funções booliana na forma CNF.

- $f1 = -x_3 \wedge (x_1 \vee x_2)$
- $f2 = x_1 \wedge x_3$
- $f3 = (x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3)$

Como segundo passo e gerar a formula F que representa as transições, para representar os saltos entre estados S_0 e S_2 seria igual a isto :

$$F_{0..2} = T(s^0, s^1) \wedge T(s^1, s^2)$$

$$F_{0..2} = (x_1^1 \leftrightarrow -x_3^0 \wedge (x_1^0 \vee x_2^0)) \wedge (x_2^1 \leftrightarrow x_1^0 \wedge x_3^0) \wedge (x_3^1 \leftrightarrow (x_1^0 \vee x_2^0) \wedge (x_2^0 \vee x_3^0)) \wedge$$
$$(x_1^2 \leftrightarrow -x_3^1 \wedge (x_1^1 \vee x_2^1)) \wedge (x_2^2 \leftrightarrow x_1^1 \wedge x_3^1) \wedge (x_3^2 \leftrightarrow (x_1^1 \vee x_2^1) \wedge (x_2^1 \vee x_3^1))$$

$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee x_4 \vee x_1), n = 4, m = 3$

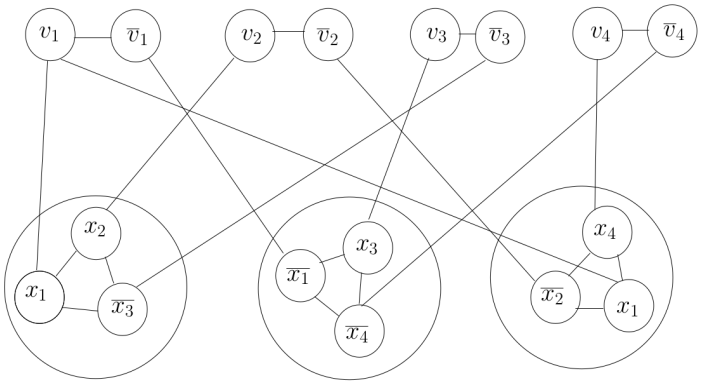

Figura 3: 3SAT Vertex Cover

Ainda a formula Booliana F não pode ser resoluta pelo solucionador Hitting Set ,esta formula tem que se converter no CNF, gerando esta formula Booliana :

$$\begin{aligned} CNF - F_{0..2} =& (\neg X_1^0 \vee X_3^0 \vee X_1^1) \wedge (X_1^1 \vee X_3^0 \vee \neg X_2^0) \wedge \\ & (\neg X_1^1 \vee X_1^0) \wedge (\neg X_1^1 \vee X_1^0 \vee X_2^0) \wedge \\ & (\neg X_1^0 \vee \neg X_3^1 \vee \neg X_2^0) \wedge (X_3^0 \vee X_3^1 \vee X_2^0) \wedge \\ & (\neg X_1^0 \vee X_3^0 \vee X_2^1) \wedge (X_1^0 \vee \neg X_2^1) \wedge \\ & (X_3^0 \vee \neg X_2^1) \wedge (\neg X_1^0 \vee X_3^0 \vee X_3^1) \wedge (X_3^0 \vee X_3^1) \end{aligned}$$

Neste ponto já pode ser usado o solucionador Hitting Set para resolver a Satisfazibilidade da formula booliana, todas as clausulas C vão representar os subconjuntos S', o que precisamos e ter em conta o tamanho de resposta do Hitting Set , osea o quantidade de elementos da Solução do Hitting set , tem que ser a mesma que a quantidade de estados da rede booliana.

O que faz esto possível e a ideia que de cada Clausula C pelo menos uma das variáveis ter que ser 1 isto faz que toda a Formula Booliana F seja Verdadeira.Nesse sentido precisamos encontrar um conjunto de clausuras que tenham pelo menos uma clausula em cada associação dada, em outras palavras um subconjunto formado pelas inversões de cada clausula é uma solução para nosso problema.

Resolvendo o outro ponto da problemática e o segundo rodamento do algoritmo quando já se tem os atratores, somente tem que aplicar a mesma formula $F = F \wedge \neg A(s^0)$ com os valores das variáveis negadas.

O Hitting Set Solver tem que ser chamado passando as clausulas como se fossem os subconjuntos e também tem que se enviar a quantidade v que repre-

senta a quantidade de variáveis geradas pelas $t = 2$ transições, O que vai gerar a seguinte entrada para $v = 9$ é:

$$\begin{aligned} C_{0..2} = (\neg X_1^0, X_3^0, X_1^1), (X_1^1, X_3^0, \neg X_2^0), (\neg X_1^1, X_1^0), (\neg X_1^1, X_1^0, X_2^0), \\ (\neg X_1^0, \neg X_3^1, \neg X_2^0), (X_3^0 \vee X_3^1 \vee X_2^0), (\neg X_1^0 \vee X_3^0 \vee X_2^1), \\ (X_1^0, \neg X_2^1), (X_3^0, \neg X_2^1), (\neg X_1^0, X_3^0, X_3^1), (X_3^0, X_3^1) \end{aligned} \quad (4)$$

As probas do Hitting Set na resolução de Satisfazibilidade deram certo para quantidades pequenas de variáveis.

4.3 Análises de Complexidade do Hitting Set

- O problema de Achar o Mínimo Hitting Set e fácil é equivalente ao problema de Dualizar Hiper-grafos de Larga escala. Achar um Mínimo Hitting Set es fácil mais achar todos os minimal hitting set es difícil, pois tem que fazer checks exponencialmente. Os estudos falam que para pequenos espácios de procura os resultados são rápidos, mas quando aumentar demoram muito. O algoritmo para dualização de large-escale Hiper-grafos tem uma complexidade de $N^{O(\log N)}$, onde N e o espaco de entrada mas o espaco de saida.Te muitos metodos para achar o Minimo Hitting Set mas tudos crecem exponencialmente. [8].

- O Hitting Set Problem esta no grupo dos problemas NP-Completo , mas o Minimal Hitting Set Problem fica no grupo dos problemas NP-Hard os quais são pelo menos tão difíceis quanto os problemas mais difíceis em NP [10]. Tudos os algoritmos exatos para resolver problemas NP-Hard sao exponenciales[5]

- No tipo de abordagem parcial onde pode o não achar a solução, Um algoritmo seguindo o caminho determinístico teria uma complexidade de $O(m * \sqrt{3} + n^2)$, onde n e o numero de subconjuntos e m e o numero de elementos, algums abordagens aleatórios tem tempos de $O(n^2)$, $O(m\sqrt{n})$, $O(n^{5/2} \log n)$

4.4 Problemas na Resolução de Satisfazibilidade pelo Hitting Set

- Um algoritmo para resolver HSP com el caminho determinístico poderia obter $O(m\sqrt{n} + n^2)$ onde n e o numero de subconjuntos e m o numero de elementos. Os abordagens Euristicos podem nao llevar ao soluçao, e sus tempos são, $O(n^2)$ e $O(m\sqrt{n})$. Temos tambem um algoritmo aproximado que resolve em $O(n^{5/2} \log n)$

- O Problema de achar o Minimal Hitting Set é classificado como NP-Hard e como minimo a Soluçao do mesmo tem complexidade NP-Completo

- O problema nao e achar uma solução valida para o HSP , o problema e determinar que nao existe solução. muitos HSP solver usam internamente um SAT solver, o qual e muito estudado e otimizado.

5 Analise de Complexidade do Algoritmo to Find Attractors with Hitting Set

5.1 Tamanho de Entrada para o Hitting Set Solver

O tamanho de entrada para o Programa do Danilo e a quantidade de subconjuntos S e a quantidade de elementos da solução R.Os arquivos de entrada da Professora Duvrova representam os genes desde jeito:

- Variáveis - Variaveis Asociadas
- Linhas da Funcao Booliana - Resultado da Funcao

Onde cada Línea tem um resultado, para formar o DNF As líneas tem que se juntar com o operador $\vee$ e depois esta formula booliana tem que se converter para o CNF. A quantidade de Líneas e similar a quantidade de genes associados.Chamaremos de n a quantidade de genes, a quantidade de líneas (Funciones boolianas) que tem resultado 1 poderia ser igual no pior caso a metade de genes, mas na maioria depende só de a quarta parte. o que da a complexidade parcial e

$$\Theta(n) = n * (n/2) * (1/2) * (1/2)$$

Depois disto já temos as funções associadas a cada gene, mas para procurar atratores temos que criar a Função Booliana F onde vai se juntar as transições que se produz no tempo T, na media só e preciso uma Função onde $T = 2n$, no conjunto com a formula anterior daria uma complexidade total de

$$\Theta(n) = n * (n/2) * (1/2) * (1/2) * (2 * n)$$

$$\Theta(n) = n^3/4$$

A pregunta que Danilo fez foi sobre a quantidade de variáveis que vai receber seu programa do "Hitting Set", esta quantidade de variáveis é descrita em seu caso médio pela formula:

$$\Theta(n) = (n * (2^n/tpa))$$

Onde$n$ e a quantidade de genes e o 2^n e a máxima quantidade de possíbeis estados dos genes, a variável tpa e o tamanho promédio do atrator o que vai variar de acordo a tamanho dos atratores, nesta tabela da pra perceber como vai aumentando o numero de variáveis de acordo as transições para um numero de genes $n = 3$.

Tabela 1: Quantidade de Variáveis

Transições	Estados	Numero Variáveis
1	000	3
2	001	6
3	010	9
4	011	12
5	100	15
6	101	18
7	110	21
8	111	24
9	100	27

5.2 Análises do Complexidade do HSP-Solver do Danilo

O HSP-Solver do Danilo e uma implementação optimizada e em Paralelo do Algoritmo do Steinbach e Postohoff [11]. Em ela ele cria lista onde guarda os valores dos subconjuntos e depois cria uma lista enlanzada o um dicionario com o qual executa o algoritmo.A complexidade deste algoritmo e de $O(((\binom{|X|}{k})))$. Onde X e o numero de clausulas que se traduz em o numero de elementos da lista enlanzada, e o k que e o numero de elementos da união dos conjuntos. Este fato faz que a Complexidade do Algoritmo dependa direitamente do numero de subconjuntos.

6 Procura de Atractores Locais em RDDAs

Como primeiro passo para achar a estabilidade e dinamismo nas RDDAs, a gente tem que procurar os Atractores Locais para cada RDDA, para isto temos dois opções que avaliamos:

- Entender o Conjunto das RDDAs como uma só Rede Booliana, e depois achar os atractores e ao final dividir os estados para cada RDDA. Esta opção e a mais real e exata pois permite achar os atratores ao mesmo tempo incluindo sua dinâmica.
- Procurar os Atratores individualmente por cada RDDA. Para os valores dos sinais de acoplamento estos ficariam fixos, então teríamos um grupo de atratores por cada combinação de sinal de acoplamento possível, os atratores achados seriam aproximações da dinâmica real.

6.1 Problema de achar Satisfazibilidade com valores Constantes

O MiniSat e a maioria de SAT-Solvers utilizam o método de substituição como já vimos nos conceitos iniciais, e depois utiliza um tipo de arvore onde vai

guardando as possíveis soluções. Por este fato os SAT-Solvers não podem o não tem implementado la possibilidade de ter valores constantes.

6.2 Complexidade CNF

Um conjunto importante de problemas em complexidade computacional envolve encontrar atribuições para as variáveis de uma fórmula booleana expressa em Forma Normal Conjuntiva, de tal forma que a fórmula é verdadeira. O problema k-SAT é o problema de encontrar uma atribuição satisfatória a uma fórmula booleana expressa em CNF na qual cada disjunção contém no máximo k variáveis. O 3-SAT é NP-completo (como qualquer outro problema k-SAT com k$>$ 2), enquanto o 2-SAT é conhecido por ter soluções em tempo polinomial. Como consequência, a tarefa de converter uma fórmula em um DNF, preservando a satisfatibilidade, é NP-difícil; duplamente, converter em CNF, preservando a validade, também é NP-difícil; portanto, a conversão de preservação de equivalência em DNF ou CNF é novamente NP-difícil. Desse jeito fala la na Wikipedia.

6.3 Conversão a CNF

O método tradicional [9, 2] para transformar uma formulação Booleana em CNF se basa em a aplicação de regras de equivalências lógicas:

1. eliminação de equivalência
2. eliminação de implicâncias
3. eliminação da dupla negação
4. leis de De Morgan
5. lei distributiva

Este método e o utilizado em este trabalho. No entanto, em alguns casos, essa transformação em CNF pode levar a uma explosão exponencial da fórmula [6], podendo produzir uma formulação em CNF com 2^n cláusulas o qual e a complexidade de espaço, mas o algoritmo tem uma complexidade de tempo polinomial.

Existem outros algoritmos que poderiam ser usados para nesse trabalho: O algoritmo proposto por Quine–McCluskey, que serve para minimização de Funciones Booleanas, além de devolver a formulação Booleana in CNF. Esse algoritmo não é usado neste abordagem pois tem uma complexidade exponencial respeito ao numero de variáveis. [7]. O Algoritmo proposto por Tseytin também permite a conversão, mas deixa a formulação de saída em 3-CNF adicionando variáveis adicionais. O comprimento da formulação de saída é linear ao tamanho da formulação inicial [12].

7 Procura de Campos Atratores em RDDAs

Nesta seção descrevemos as características da implementação do algoritmo para achar com RDDAs.

7.1 Gerador de RDDAs

Para puder testar o algoritmo, foi criado um programa gerador de RDDAs, ele recebe como parâmetros:

- Número de RDDs que formaram a RDDA.
- Número de variáveis que terram as RDDs
- Número de sinais de acoplamento que terá cada uma das RDDs.
- Número máximo e mínimo de Clausulas que definem o estado de cada variável.
- Número máximo e mínimo de literais dentro das cláusulas que definem o estado de cada variável.

8 Anexos

8.1 Detalhes da Implementação do Algoritmo com Sinais de acoplamento em conjunto

Os sinais de acoplamento Y, de acordo com o escrito, são apenas um por relação entre os RDDs. Isto é, se houver uma conexão entre a rede r e a rede j, entrando em j, não importa quantas variáveis a rede r tenha em seu conjunto de saída $\mathcal{S}^j$, somente um sinal de acoplamento chegará. Isto pode-se ver no gráfico 4. Esta escolha gera as observações:

- Tem pouca flexibilidade na interação do sinal de acoplamento com as variáveis de entrada $\mathcal{E}^j$ da rede objetivo.
- Não se tem correspondência entre a sinal de acoplamento Y e as variáveis de saída da rede de onde vem.

Para exemplificar isto vamos usar a representação de uma RDDA no gráfico 4 mas somente usaremos 3 RDDs, as redes serão r, u, j. Sendo j a rede onde chegam os sinais de acoplamento y_r, y_u. Cada uma das redes tera 5 variáveis, as quais são respetivamente $\mathcal{X}^r = \{x_1, x_2, x_3, x_4, x_5\}$, $\mathcal{X}^u = \{x_6, x_7, x_8, x_9, x_{10}\}$, $\mathcal{X}^r = \{x_{11}, x_{12}, x_{13}, x_{14}, x_{15}\}$. Já que a rede j tem dois co-vizinhas, terá 2 sinais de acoplamento y_u, y_r. As quais são formadas por as 2 variáveis de saída de cada rede, x_6, x_7 da rede u e x_1, x_2 da rede r, respetivamente.

Exemplificamos o problema 1 de este abordagem, si a gente quiser que o jeito de relação entre as variáveis de r e u seja por cada uma das variáveis das sinais de saída $\mathcal{S}_u$, assim :

$$f(x_4^{11}) = f(x_{11..15}, x_6)$$
$$f(x_4^{12}) = f(x_{11..15}, x_7)$$
$$f(x_4^{13}) = f(x_{11..15}, y_r)$$

esto no poderia ser feito si somente a sinal de acoplamento y_u chega ate a rede r.

Exemplificando o problema 2, se tem que ao momento de reduzir as variáveis de saída $\mathcal{S}_u$ em só uma sinai de acoplamento, se perdem os valores iniciais das variáveis de saída $\mathcal{S}_u$, o qual fara inexato o calculo de campos atratores na fase 2 da pesquisa, pois não se taram os valores que relacionam uma rede com outra, deixamos aqui um exemplo com valores na tabela 8.1.

x6	x7	h_r	y_r
0	0	$\wedge$	0
0	1	$\wedge$	0
1	0	$\wedge$	0
1	1	$\wedge$	1

Tabela 2: Tabela com os valores das variáveis de saída da rede u

Ao momento de procurar os atratores locais de j, somente ter que se executar duas vezes o algoritmo: uma veis com o valor $y_r = 1$ e outro com $y_r = 0$, então teremos dois listas de atratores $\mathcal{A}_j^1$ e $\mathcal{A}_j^0$. No caso de $\mathcal{A}_j^1$ pode facilmente identificar a que valor (11) de x_6 e x_7, mas no caso de $\mathcal{A}_j^0$ não se pode identificar a qual valor $(00, 01, 10)$ de x_6 e x_7 pertence, o qual representa um problema maior ao momento de calcular os campos atratores.

Este problema fica ainda maior quando são 3 as variáveis de saída $\mathcal{S}_r$ que conformam o sinal de acoplamento y_r, neste caso utilizaremos x_6, x_7, x_8, respetivamente. A tabela de possíveis valores para estas variáveis é:

x6	x7	x8	h_r	y_r
0	0	0	$\wedge$	0
0	0	1	$\wedge$	0
0	1	0	$\wedge$	0
0	1	1	$\wedge$	0
1	0	0	$\wedge$	0
1	0	1	$\wedge$	0
1	1	0	$\wedge$	0
1	1	1	$\wedge$	1

Igual que o anterior exemplo ao momento de achar os atratores locais, também executaremos duas vezes o algoritmo, uma veis para $y_r = 1$ e outra para $y_r = 0$. Para o caso de $y_r = 0$. O problema e ainda maior pois não pudemos conhecer o qual valor $\{000, 001, 010, 011, 100, 101, 110\}$ das variáveis de saída ,

corresponde os atratores da lista $\mathcal{A}_j^0$. Isto impossibilita e faz inexato o calculo de os campos atratores.

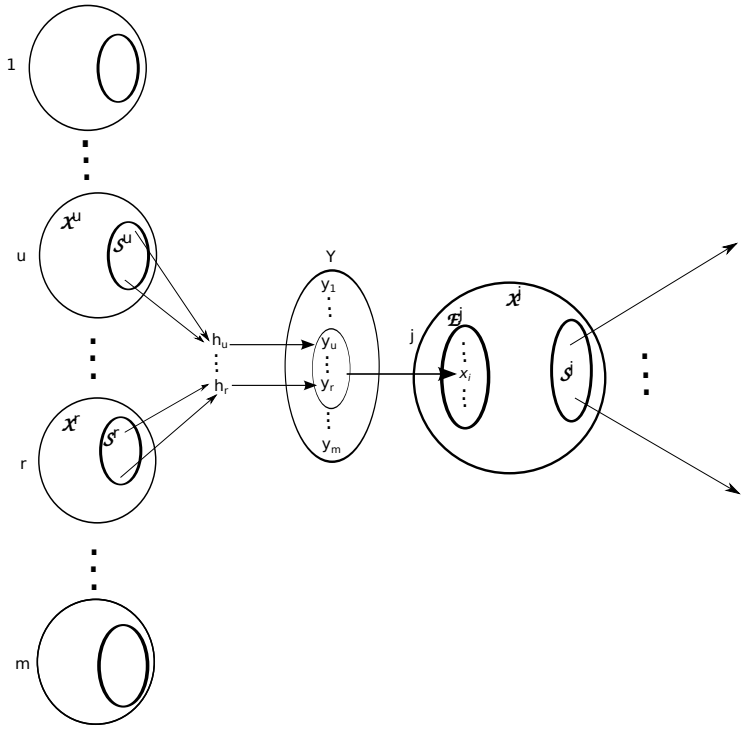

Figura 4: Uma representação gráfica da forma pela qual as RDDs $u, \ldots, r$ (que são convizinhas de j) exercem influência sobre j.

8.2 Detalhes da Implementação do Algoritmo com Sinais de acoplamento individuais

Em resposta ao abordagem anterior se determino usar dentro da rede local r direitamente os valores das variáveis de saída $\mathcal{S}_u$, isto tem duas avantajes:

- Flexibilidade da interação entre as variáveis de saída $\mathcal{S}^u$ e as variáveis de entrada $\mathcal{E}^r$
- Exatidão na relação de variáveis no reconhecimento de campos atratores.

Mas tem uma desvantagem, Tem que ser feitas mais corridas de algoritmo têm que ser feitas para achar os atratores locais.

Respeito a flexibilidade de este abordagem os exemplo da secção anterior com 2 variáveis de saída, podem ser feitos normalmente, tomando em conta que a função de acoplamento h_r e a função $\wedge$, daria a equivalência de $y_r = x_6 \wedge x_7$:

$$f(x_4^{11}) = f(x_{11..15}, x_6)$$
$$f(x_4^{12}) = f(x_{11..15}, x_7)$$
$$f(x_4^{13}) = f(x_{11..15}, y_r)$$
$$f(x_4^{13}) = f(x_{11..15}, x_6 \wedge x_7)$$

As dois ultimas formulações são equivalentes, então este abordagem pode representar o abordagem da secção anterior sim problemas. Ista equivalência também pode ser representada do jeito gráfico, na imagem 5.

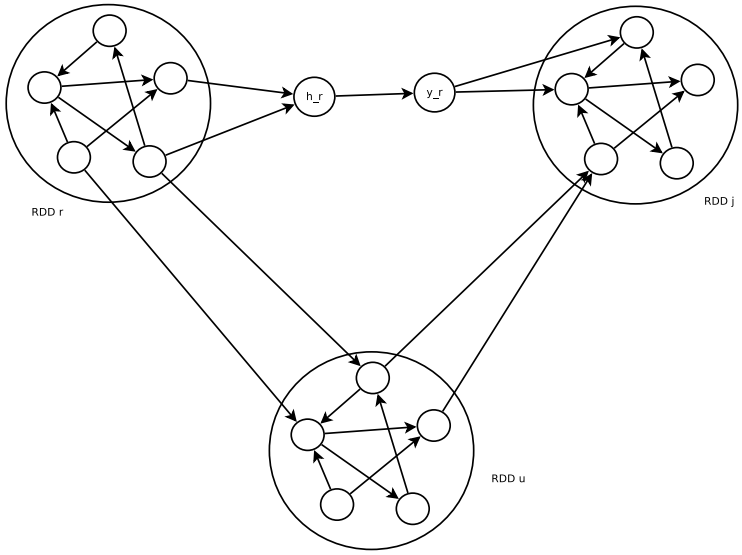

Figura 5: Equivalência entre os abordagens.

Referências

[1] T. Akutsu. *Algorithms for Analysis, Inference, and Control of Boolean Networks*. World Scientific Publishing Co Pte Ltd, 2018.

[2] A. R. Bradley and Z. Manna. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media, 2007.

[3] D. Carastan-Santos, R. Y. de Camargo, D. C. Martins Jr, S. W. Song, and L. C. Rozante. Finding exact hitting set solutions for systems biology applications using heterogeneous gpu clusters. *Future Generation Computer Systems*, 67:418–429, 2017.

[4] E. Dubrova and M. Teslenko. A sat-based algorithm for finding attractors in synchronous boolean networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 8(5):1393–1399, 2011.

[5] M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[6] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[7] E. J. McCluskey Jr. Minimization of boolean functions. *Bell system technical Journal*, 35(6):1417–1444, 1956.

[8] K. Murakami and T. Uno. Efficient algorithms for dualizing large-scale hypergraphs. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–13. SIAM, 2013.

[9] S. Russell and P. Norvig. Artificial intelligence: A modern approach prentice-hall. *Englewood cliffs, NJ*, 26, 1995.

[10] L. Shi and X. Cai. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 64–67. IEEE, 2010.

[11] B. Steinbach and C. Posthoff. Sources and obstacles for parallelization—a comprehensive exploration of the unate covering problem using both cpu and gpu. *GPU Computing with Applications in Digital Logic*, page 63, 2012.

[12] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.