PROCESSOS K-REVERSÍVEIS EM GRAFOS

Leonardo Inácio Lima de Oliveira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Valmir Carneiro Barbosa
Fábio Protti

Rio de Janeiro
Dezembro de 2012

PROCESSOS K-REVERSÍVEIS EM GRAFOS

Leonardo Inácio Lima de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Fábio Protti, D.Sc.

Prof. Jayme Luiz Szwarcfiter, Ph.D.

Prof. Luerbio Faria, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2012

Oliveira, Leonardo Inácio Lima de

Processos k-reversíveis em grafos/Leonardo Inácio Lima de Oliveira. – Rio de Janeiro: UFRJ/COPPE, 2012.

IX, 63 p. 29, 7cm.

Orientadores: Valmir Carneiro Barbosa

Fábio Protti

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2012.

Referências Bibliográficas: p. 62 – 63.

1. Processos k-reversíveis. 2. Sistemas dinâmicos em grafos. 3. Processos que operam com limiares. 4. Autômatos celulares. 5. Configurações jardim-do-éden. I. Carneiro Barbosa, Valmir *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

"A melhor maneira de prever o futuro é inventá-lo." - Alan Kay

Agradecimentos

Agradeço especialmente à minha mãe Kátia pelo seu amor incondicional, por sempre ter acreditado no meu sucesso e por tudo o que fez por mim ao longo de toda a minha vida.

Agradeço a meu pai Ernesto pelo apoio e por me proporcionar uma excelente formação.

Agradeço à minha namorada Gabriela por todo o incentivo, carinho e compreensão.

Agradeço aos meus orientadores Valmir Barbosa e Fábio Protti pelo suporte essencial para a conclusão desse trabalho. As discussões sobre a pesquisa ao longo desses meses sempre foram fonte de inspiração e novas perspectivas e objetivos. A disponibilidade, os conhecimentos transmitidos e os sábios conselhos por parte de ambos nunca serão esquecidos.

Agradeço aos membros da banca pelas sugestões que tornaram esse trabalho melhor.

Agradeço ao Programa de Engenharia de Sistemas e Computação(PESC/COPPE/UFRJ) pelo alto nível de ensino proporcionado e pela excelente estrutura que possui.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior(CAPES) pelo suporte financeiro.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PROCESSOS K-REVERSÍVEIS EM GRAFOS

Leonardo Inácio Lima de Oliveira

Dezembro/2012

Orientadores: Valmir Carneiro Barbosa
Fábio Protti

Programa: Engenharia de Sistemas e Computação

Uma variedade de modelos para estudar a disseminação de opinião consideram os aspectos dos processos dinâmicos em grafos. Nesse trabalho, apresentamos os processos k-reversíveis. Em tais processos, cada vértice do grafo possui um estado dentre dois estados possíveis a cada instante de tempo discreto. A mudança de estado de cada vértice ocorre apenas se esse vértice possuir pelo menos k vizinhos com estado oposto ao seu próprio estado naquele passo de tempo.

Ao longo dessa dissertação, estudamos os processos k-reversíveis e suas relações com outros processos que operam com limiares. Apresentamos resultados conhecidos na literatura relacionados ao comprimento do transiente e do período nesses processos. Desenvolvemos uma ferramenta específica para os processos reversíveis, que foi essencial na obtenção de demonstrações alternativas para esses mesmos resultados e para a demonstração de alguns limites mais justos. Estudamos o problema de determinar se uma configuração inicial é uma configuração jardim-do-éden em processos k-reversíveis. Para esse problema, obtemos resultados inéditos para a dificuldade do problema e também algoritmos polinomiais para o caso onde o grafo é uma árvore e para o caso dos processos 2-reversíveis em grafos onde o grau máximo de um vértice é três.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

K-REVERSIBLE PROCESSES IN GRAPHS

Leonardo Inácio Lima de Oliveira

December/2012

Advisors: Valmir Carneiro Barbosa
Fábio Protti

Department: Systems Engineering and Computer Science

Several models proposed to study opinion dissemination consider aspects of dynamic processes in graphs. In this work, we present the k-reversible processes. In such processes, each vertex in the graph has one of two possible states at each discrete time step. Each vertex changes its state if and only if it has at least k neighbours in a state different than its own.

In this thesis, we study the k-reversible processes and how they are related to other threshold processes. We present known results found in the literature related to the length of the transient and period in this processes. We have developed a specific tool to study reversible processes that was a useful tool to, alternatively, prove those known results and to obtain better upper bounds. We also study the problem of determining whether a given configuration is a garden-of-eden configuration in such processes. We present new results for this problem, including its hardness and efficient algorithms when the graph is a tree. We also present an efficient algorithm for 2-reversible processes when the maximum degree of a vertex in the graph is bounded by three.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Seja G um grafo simples, finito e não-direcionado. Definimos que um *processo k-reversível* em G é um processo iterativo, onde em cada passo de tempo discreto t, cada vértice de G possui um determinado estado. Os estados de cada vértice são apenas números inteiros que pertencem ao conjunto $\{-1, +1\}$ e são alterados em cada passo de tempo se e somente se o vértice possuir pelo menos k vértices vizinhos com estado oposto ao estado que possui naquele passo de tempo, onde k é um número inteiro positivo.

A motivação para o estudo de tais processos pode ser relacionada com a análise da disseminação de opinião em redes sociais. Por exemplo, suponha que o grafo modele uma rede, onde cada vértice representa uma pessoa e cada aresta entre dois vértices representa que aquelas pessoas se conhecem. Vamos supor que o estado -1 representa que a pessoa tem opinião contrária em relação a algum assunto, e o estado $+1$ representa que a pessoa tem opinião a favor sobre esse mesmo assunto. Um processo k-reversível, portanto, é uma abordagem para modelar a disseminação de opinião quando as pessoas são fortemente influenciadas pela opinião da sociedade onde estão inseridas. Note que nesse modelo, estamos assumindo que todas as pessoas agem de forma igual. Uma modelagem mais complexa poderia, por exemplo, assumir limiares distintos para cada pessoa, bem como limiares que sejam calculados através de uma função do número de pessoas conhecidas.

Podemos observar que um processo k-reversível é, na verdade, um exemplo de *sistema dinâmico em grafos*. Mais precisamente, um processo k-reversível é um exemplo do que conhecemos por *autômato celular generalizado*, que é a extensão do conceito de *autômato celular* para grafos quaisquer com funções de atualização dos estados dos vértices podendo ser distintas para cada vértice. De maneira gral, podemos definir formalmente o conceito de sistema dinâmico em grafos pela seguinte tripla:

- Um grafo G com conjunto de vértices $V = \{v_1, v_2, \ldots, v_n\}$ e conjunto de arestas

E. Para cada vértice $v_i \in V$, temos um número inteiro $x_i \in Q$ (por exemplo, $Q = \{-1, +1\}$) associado. O grafo não é necessariamente simples e não-direcionado, mas é sempre finito.

- Uma função $F : Q^n \to Q^n$ calculada a partir da função $F_i : Q^n \to Q^n$ ou da função $f_i : Q \to Q$ para cada vértice v_i:

$$F_i(x_1, x_2, \ldots, x_n) = (x_1, x_2, \ldots, x_{i-1}, f_i(x_i), x_{i+1}, \ldots, x_n)$$

  O cálculo de F depende do esquema de atualização utilizado.

- Um esquema de atualização para utilizar os valores F_i e calcular o valor da função $F : Q^n \to Q^n$. Dois dos métodos mais utilizados são:

  - Síncrono, onde o sistema dinâmico é denominado *autômato celular generalizado*:

$$F(x_1, x_2, \ldots, x_n) = (f_1(x_1), f_2(x_2), \ldots, f_n(x_n))$$

  - Sequencial, onde o sistema dinâmico é denominado *sistema dinâmico sequencial*:

$$F(x_1, x_2, \ldots, x_n) = F_{p(1)} \circ F_{p(2)} \circ \ldots \circ F_{p(n)},$$

    onde p é uma permutação da numeração dos vértices de G.

  A função F é aplicada no vetor x em cada instante de tempo discreto t, onde cada componente x_i contém o estado do vértice v_i no tempo t.

Em um processo k-reversível, a função de atualização é aplicada simultaneamente a todos os vértices do grafo. Ou seja, os estados considerados para a aplicação da função são todos eles obtidos da aplicação anterior da função. Poderíamos também definir um processo k-reversível sequencial, mas nesse caso a motivação sobre influência social perderia o sentido, visto que, no mundo real, não conhecemos uma ordem fixa de como as pessoas mudam de opinião. Note, que para o caso sequencial, a escolha da ordem de atualização dos estados dos vértices influencia drasticamente as configurações de estados subsequentes da dinâmica.

O estudo de sistemas dinâmicos em grafos, assim como o estudo de autômatos celulares, é um estudo multidisciplinar abrangendo diversas áreas, como óptica [1], redes neurais [2], termodinâmica estatística [3], disseminação de opinião [4] e disseminação de doenças [5].

Na área de computação distribuída, também temos diversos estudos e modelos de sistemas dinâmicos, como, por exemplo, o modelo de processos majoritários, onde cada vértice muda de estado se e somente se pelo menos metade de seus vizinhos tem

estados diferentes [6]. Uma aplicação desse modelo é dada em [7] para manutenção de consistência de dados.

No que se refere à análise de processos k-reversíveis, grande parte do estudo realizado envolve o problema do MENOR CONJUNTO CONVERSOR em tais processos, e muitos resultados interessantes sobre esse problema podem ser encontrados no trabalho de Dreyer [8]. Nesse mesmo trabalho, Dreyer apresenta alguns resultados importantes para certos limites superiores para o comprimento do período nesses processos e também para o comprimento do transiente. Todos esses limites apresentados, no entanto, utilizam a redução dos chamados *processos que operam com limiares*, que são estudados amplamente por Goles e Olivos em [9, 10]. E, portanto, os resultados apresentados se baseiam em limites mais gerais e em sua maioria utilizando demonstrações algébricas, de forma que nenhum estudo específico para o caso de processos k-reversíveis foi realizado até o momento utilizando uma abordagem que utilize características próprias da dinâmica desses processos. Uma parte dessa dissertação é dedicada justamente à obtenção de uma abordagem alternativa que utilize a dinâmica desses processos para demonstrar alguns resultados já conhecidos e também para melhorar alguns limites obtidos.

Um problema básico que também aparece em sistemas dinâmicos em grafos é o chamado EXISTÊNCIA DE CONFIGURAÇÃO PREDECESSORA. Como o nome diz, dado um sistema dinâmico e uma configuração de estados qualquer, queremos determinar a existência de uma configuração predecessora para tal configuração. Esse problema foi estudado por Sutner [11] em autômatos celulares e demonstrado ser um problema NP-Completo. Resultados de NP-Completude existem também para alguns sistemas dinâmicos sequenciais, como pode ser visto em [12], bem como alguns algoritmos polinomiais para algumas classes de grafos, como as árvores. Uma extensão óbvia para esse problema é contar o número de configurações predecessoras. Novamente, para sistemas dinâmicos sequenciais, temos resultados em [13]. O caso de autômatos celulares generalizados parece ter menos atenção, e, nessa dissertação estudamos esse problema nos processos k-reversíveis, obtendo resultados similares de NP-Completude e algoritmos polinomiais para árvores.

Continuamos essa dissertação dividindo o conteúdo em mais 4 capítulos organizados da seguinte forma:

- No Capítulo 2, definimos formalmente o que é um processo k-reversível e sua relação com processos que operam com limiares. Apresentamos outros processos similares e também o problema conhecido como MENOR CONJUNTO CONVERSOR. Finalizamos o capítulo apresentando o operador monotônico decrescente denominado na literatura como *função de energia* e alguns dos principais resultados obtidos em trabalhos relacionados que utilizam essa função como ponto essencial de suas demonstrações. Os resultados apresentados

são, principalmente, limites superiores para o comprimento do transiente e do período em processos que operam com limiares e também a aplicação desses resultados diretamente nos processos k-reversíveis.

- No Capítulo 3, apresentamos o que chamaremos de *função de energia para processos k-reversíveis*. Essa função é um operador monotônico crescente que concebemos para o caso restrito de processos k-reversíveis. Por ser específica para esse caso, a função é mais fácil de ser compreendida e permite demonstrações mais simples e limites superiores mais justos para o comprimento do transiente e do período em processos k-reversíveis. Nesse capítulo, utilizamos essa função para demonstrar alguns desses limites, como, por exemplo, o comprimento máximo do período em tais processos. O capítulo contém ainda uma conjectura para o limite superior justo do transiente no caso restrito de processos 2-reversíveis em árvores.

- No Capítulo 4, tratamos o problema de determinar a existência de uma configuração predecessora em processos k-reversíveis. Esse problema é bastante estudado em *autômatos celulares*, mas pouco estudado em grafos. Nesse capítulo, mostramos que esse problema é NP-Completo para processos k-reversíveis e mostramos também a existência de algoritmos polinomiais para o caso onde o grafo é uma árvore e para o caso onde $\Delta(G) \leq 3$ em processos 2-reversíveis. Para o caso onde o grafo é uma árvore, mostramos o algoritmo polinomial que fornece uma configuração predecessora e um algoritmo, também polinomial, que conta a quantidade total de configurações predecessoras.

- Finalmente, no Capítulo 5, apresentamos um resumo do trabalho realizado e possíveis trabalhos futuros.

Capítulo 2

Processos k-Reversíveis

Nesse capítulo, introduziremos formalmente o conceito de processos k-reversíveis e outros processos similares em grafos. Fazemos um estudo geral de trabalhos anteriores e seus principais resultados, principalmente os resultados associados aos comprimentos do transiente e do período em tais processos.

2.1 O Modelo

Seja $G(V, E)$ um grafo simples com n vértices e m arestas, onde cada vértice v possui grau $d(v)$. Associamos a cada vértice um número inteiro que denominaremos como sendo o *estado* do vértice, que será sempre -1 ou $+1$. Chamamos de processo k-reversível em G a dinâmica onde em cada unidade de tempo discreto t, cada vértice muda de estado se e somente se possuir pelo menos k vizinhos com estado oposto ao seu próprio estado no tempo t. Apesar de não se restringir apenas a grafos conexos, por simplicidade, toda a análise realizada nesse trabalho assumirá que os grafos são conexos. Toda a análise para grafos desconexos é similar, bastando assumir cada componente conexa de G separadamente.

O processo k-reversível ocorre na forma de uma *dinâmica síncrona*, onde as alterações de estados dos vértices são realizadas simultanemante nos instantes de tempo 0, 1, 2, ... Dessa forma, para definir o estado de cada vértice no passo de tempo $t + 1$, é necessário considerar os estados de seus vizinhos, além do estado do próprio vértice, no tempo t.

Seja $V(G) = \{v_1, v_2, \ldots, v_n\}$. Denominaremos por $x(t)$ o vetor onde a i-ésima posição $x_i(t)$ contém o estado do vértice v_i no tempo t. Definimos também o vetor $op(t)$, tal que a i-ésima posição $op_i(t)$ é a quantidade de vizinhos com estado oposto a $x_i(t)$ no passo de tempo t. Essas definições terão esse mesmo significado ao longo de todo o trabalho. Com essas definições, podemos descrever a dinâmica do processo k-reversível da seguinte forma:

$$x_i(t+1) = \begin{cases} +1, & \text{se } x_i(t) = +1 \text{ e } op_i(t) < k \\ +1, & \text{se } x_i(t) = -1 \text{ e } op_i(t) \geq k \\ -1, & \text{se } x_i(t) = +1 \text{ e } op_i(t) \geq k \\ -1, & \text{se } x_i(t) = -1 \text{ e } op_i(t) < k \end{cases} \tag{2.1}$$

Note que o vetor $x(0)$ pode ser escolhido arbitrariamente, e, a partir dele, o processo se desenvolve. Qualquer vetor que descreva o estado dos vértices em uma determinada unidade de tempo, chamaremos de *configuração*. Em particular, chamaremos $x(0)$ de *configuração inicial*. Observe também que o fato de termos uma configuração inicial $x(0)$ não implica na não-existência de uma outra configuração S pela qual seja possível obter $x(0)$ após algum número finito de passos de tempo.

A Figura 2.1 ilustra a dinâmica de um processo 2-reversível em um grafo P_5. Para melhor compreensão, os vértices coloridos com a cor preta representam vértices com estado $+1$, e os vértices coloridos com a cor branca representam os vértices com estado -1. Nesse trabalho, qualquer figura que ilustre um grafo e a configuração dos estados dos vértices adotará este padrão.

Observando a Figura 2.1, vemos que em $x(0)$, apenas os vértices v_2, v_3 e v_4 possuem pelo menos 2 vizinhos com estados opostos, e, portanto, são os únicos que mudam de estado. Como descrito anteriormente, a alteração dos estados ocorre simultaneamente nos vértices que terão seus estados modificados. Dessa forma, em $x(1)$, apenas v_3 possui pelo menos 2 vizinhos com estados opostos ao seu próprio estado, e é o único vértice que muda de estado. Finalmente, em $x(2)$, todos os vértices possuem mesmo estado e o processo chega em um ponto fixo, onde a configuração não se altera mais. Observe que, nesse caso, a configuração final é uma configuração onde todos os vértices possuem mesmo estado. Essa situação não ocorre sempre, ou seja, uma configuração que não se altera pode não ser uma configuração onde todos os vértices possuem mesmo estado. Por exemplo, supondo o mesmo processo 2-reversível e o mesmo grafo P_5 mas com outra configuração inicial como ilustrado na Figura 2.2, vemos que a configuração inicial nunca se altera, mas nem todos os vértices possuem o mesmo estado.

2.2 Processos que Operam com Limiares

Um processo que opera com limiar é um processo definido completamente por uma matriz quadrada A com dimensões $d \times d$ e um vetor b com d elementos que chamaremos de *vetor limiar*. A dinâmica de atualização dos estados dos vértices em tais processos pode ser definida de maneira geral em função da matriz A e do vetor b da

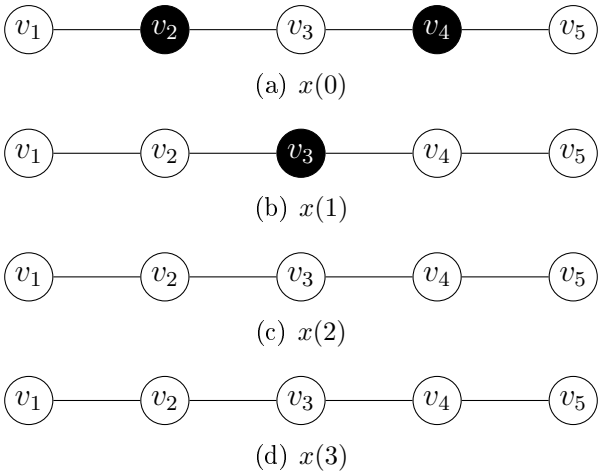


Figura 2.1: Exemplo de configurações $x(0)$, $x(1)$, $x(2)$ e $x(3)$ de um processo 2-reversível em um P_5.

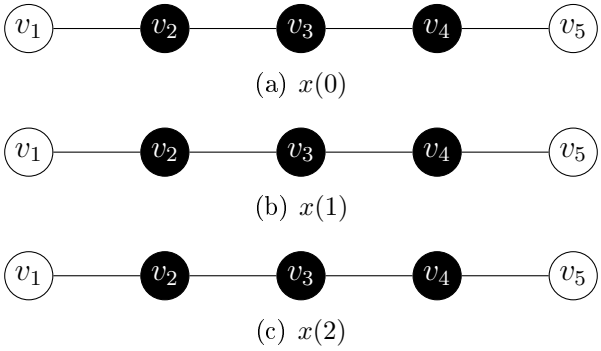


Figura 2.2: Outro exemplo de configurações $x(0)$, $x(1)$, $x(2)$ de um processo 2-reversível em um P_5. Configurações nunca se alteram nesse caso.

seguinte forma:

$$x_i(t+1) = \begin{cases} +1, & \text{se } \sum_{j=1}^{n} a_{ij}x_j(t) - b_i \geq 0 \\ 0, & c.c. \end{cases} \tag{2.2}$$

Note que os estados que utilizamos nessa definição foram os números inteiros 0 e $+1$. No entanto, os estados associados não precisam ser necessariamente os valores 0 e $+1$. Os valores para os estados são importantes e úteis apenas para facilitar a prova de alguns resultados, como veremos posteriomente. Podemos, portanto, sem perda alguma na definição, utilizar a definição de processos que operam com limiar assumindo valores de estados -1 e $+1$:

$$x_i(t+1) = \begin{cases} +1, & \text{se } \sum_{j=1}^{n} a_{ij}x_j(t) - b_i \geq 0 \\ -1, & c.c. \end{cases} \tag{2.3}$$

Processos que operam com limiares são bastante úteis para modelar diversos sistemas dinâmicos em grafos, sejam eles síncronos ou paralelos. E de fato, os processos k-reversíveis podem ser modelados dessa forma. Para isso, definimos a matriz A como em (2.4) e o vetor b como sendo o vetor nulo. Com esses parâmetros, a dinâmica descrita em (2.3) é equivalente a dinâmica de um processo k-reversível, como pode ser visto em [8]. Observe que a matriz A definida é a matriz de adjacência do grafo com a diagonal principal modificada.

$$A_{ij} = \begin{cases} 1 & \text{se } i \neq j \text{ e } (v_i, v_j) \in E(G) \\ 0 & \text{se } i \neq j \text{ e } (v_i, v_j) \notin E(G) \\ 2k - d(v_i) - 1 & \text{se } i = j \end{cases} \tag{2.4}$$

Um processo bastante similar ao processo k-reversível é o chamado *processo k-irreversível*. A diferença de um processo k-irreversível para um processo k-reversível é que, no primeiro tipo de processo, uma vez que um veŕtice atinga estado $+1$, esse vértice nunca mais altera seu estado. Dessa forma, é um processo que também possui motivação para o estudo de disseminação de opinão em redes sociais, além de ser útil também no estudo de disseminação de doenças. Note que, nesse aspecto, o processo k-irreversível é até mais interessante do que os processos k-reversíveis, já que para a disseminação de doenças estamos interessados em saber o total de indivíduos que a doença pode atingir dentro de um determinado intervalo de tempo. Nesses casos, não existe muito sentido considerar a cura das pessoas baseando-se no limiar de pessoas não contaminadas.

Assim como os processos k-reversíveis, os processos k-irreversíveis também são casos específicos de processos que operam com limiares. Essa demonstração pode ser realizada utilizando a matriz A como sendo a matriz de adjacência do grafo, o vetor $b = (k, k, k, \ldots, k)$ e utilizando os valores 0 e $+1$ para os estados [8].

Outro tipo de processo que também pode ser modelado como um processo que opera com limiar é o processo k_1-k_2-reversível, que é o processo onde os vértices mudam do estado 0 para $+1$ se e somente se possuirem pelo menos k_1 vizinhos com estado $+1$, e mudam do estado $+1$ para 0 se e somente se possuirem pelo menos k_2 vizinhos com estado 0. A ideia de limiares distintos para a transição de estados dos vértices faz sentido e pode ser útil na modelagem de situações onde a aceitação de uma afirmação, produto ou opinião ocorre com mais facilidade ou até mesmo dificuldade do que a negação, por exemplo.

Para demonstrar que os processos k_1-k_2-reversíveis são casos específicos de processos que operam com limiares utilizamos uma ideia semelhante à utilizada na prova para os processos k-reversíveis e para os processos k-irreversíveis.

Teorema 2.1. *Processos k_1-k_2-reversíveis são casos específicos de processos que operam com limiar.*

Demonstração. Definimos a matriz A:

$$A_{ij} = \begin{cases} 1 & \text{se } i \neq j \text{ e } (v_i, v_j) \in E(G) \\ 0 & \text{se } i \neq j \text{ e } (v_i, v_j) \notin E(G) \\ k_1 + k_2 - d(v_i) - 1 & \text{se } i = j \end{cases} \tag{2.5}$$

Definimos o vetor b como $(k_1, k_1, k_1, \ldots, k_1)$. Seja $x(t)$ uma configuração qualquer. Então, precisamos mostrar que caso $x_i(t) = 0$, então $x_i(t+1) = +1$ se e somente se v_i possui pelo menos k_1 vizinhos com estado $+1$. Da mesma forma, caso $x_i(t) = +1$, então precisamos mostrar que $x_i(t+1) = 0$ se e somente se v_i possui pelo menos k_2 vizinhos com estado 0.

Denominaremos $n_i^+(t)$ como a quantidade de vizinhos de v_i com estado $+1$, e $n_i^0(t)$ como a quantidade de vizinhos de v_i com estado 0.

- Caso $x_i(t) = 0$: utilizando (2.2), temos $x_i(t+1) = +1$ se e somente se $\sum_{j=1}^{n} a_{ij} x_j(t) \geq b_i$. Observe que como $x_i(t) = 0$, o valor do somatório é igual a $n_i^+(t)$, pois todas as outras parcelas do somatório são anuladas pelos valores de estado 0, e, portanto, concluimos que $x_i(t+1) = +1$ se e somente se $n_i^+(t) \geq k_1$.

- Caso $x_i(t) = +1$: utilizando (2.2), temos $x_i(t+1) = 0$ se e somente se $\sum_{j=1}^{n} a_{ij} x_j(t) < b_i$. Note que, nesse caso:

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j(t) &= n_i^+(t) + k_1 + k_2 - n_i^+(t) - n_i^0(t) - 1 \\ &= k_1 + k_2 - n_i^0(t) - 1 \end{aligned} \tag{2.6}$$

  Como $b_i = k_1$, para $\sum_{j=1}^{n} a_{ij} x_j(t) < b_i$, então é necessário que $n_i^0(t) \geq k_2$. Concluímos, portanto, que $x_i(t+1) = +1$ se e somente se $n_i^0(t) \geq k_2$.

□

Como existem diversos resultados na literatura para processos que operam com limiares [9, 10], podemos utilizar todos esses resultados nesses processos mais específicos.

2.3 O Problema do Menor Conjunto Conversor

Seja C um subconjunto de $V(G)$ e defina uma configuração X tal que todos os vértices de C possuam estado $+1$ e todos os vértices em $V(G) \setminus C$ possuam o estado -1. O conjunto C é um conjunto conversor de G se e somente se a dinâmica de um processo k-reversível tendo como configuração inicial a configuração X produz em uma quantidade finita de passos a configuração onde todos os vértices de $V(G)$ possuem estado $+1$.

Podemos usar essa mesma definição de conjunto conversor em diversos processos dinâmicos em grafos, entre eles, os processos k-irreversíves e k_1-k_2-reversíveis. Em cima dessa definição, uma questão que logo surge é a cardinalidade do menor conjunto conversor para um determinado grafo G. Esse problema é conhecido como Menor Conjunto Conversor. Na Figura 2.1, por exemplo, vemos que os vértices v_1, v_3 e v_5 formam um conjunto conversor para o grafo em um processo 2-reversível, além de formarem o conjunto conversor de menor cardinalidade. Claramente, o conjunto conversor trivial é o próprio conjunto $V(G)$. Note também que, para um processo k-reversível, qualquer conjunto conversor é função não só da estrutura do grafo em questão, como também do limiar k.

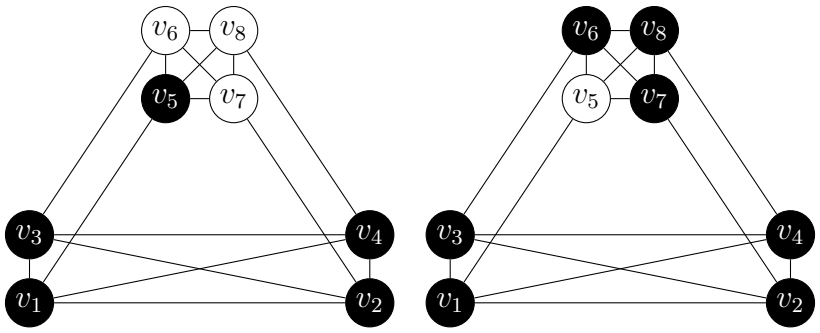
Outra questão interessante é determinar se um dado conjunto C é um conjunto conversor sem a necessidade de simular todo o processo k-reversível. Ou seja, determinar propriedades estruturais para qualquer conjunto conversor em um grafo G de acordo com o valor k. Podemos ainda perguntar qual é o menor conjunto conversor que converte G no menor número de passos. Todas essas questões, apesar de simples, mostram-se bastante difíceis de responder para casos gerais, e a maioria dos resultados existentes restrigem a classe de grafos para análise e obtenção de algoritmos polinomiais.

O Menor Conjunto Conversor é um problema NP-Difícil em processos k-reversíveis e k-irreversíveis para $k \geq 3$. Esse resultado é demonstrado por Dreyer em [8]. Para $k = 1$, o problema é trivial para os dois casos. Quando o processo é reversível, qualquer aresta entre vértices com estados distintos gera a impossibilidade da convergência dos estados dos vértices para um único estado em comum, pois, como $k = 1$, a dinâmica do processo faz com que esses dois vértices alternem seus estados indefinidamente, e, portanto, concluímos que o único conjunto conversor para esse caso é o próprio conjunto $V(G)$. Para processos irreversíveis, quando $k = 1$, a existência de uma aresta entre vértices com estados distintos faz com que todo os vértices do grafo fiquem com o mesmo estado após um determinado número de passos, já que estamos tratando apenas grafos conexos, e cada vértice ao atingir o estado $+1$, permanece nesse estado, e, portanto, nesse caso o menor conjunto possui cardinalidade 1 e é formado por qualquer vértice do grafo. Para $k = 2$, o problema

é NP-Difícil para o caso dos processos irreversíveis [14].

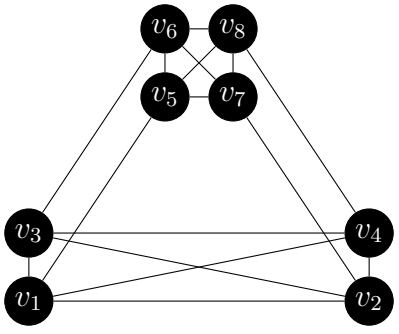
Esse problema ilustra algumas características que deixam claro a maior complexidade da dinâmica do processo k-reversível frente ao processo k-irreversível. Por exemplo, se C é um conjunto conversor de G para um processo k-irreversível, então, claramente, qualquer super-conjunto de C também será um conjunto conversor de G. O mesmo, no entanto, não se pode afirmar para os processos k-reversíveis. Um exemplo é ilustrado pela Figura 2.3 e pela Figura 2.4. Na Figura 2.3, vemos uma configuração inicial e a dinâmica subsequente para um processo 2-reversível que resulta na conversão total do grafo. A Figura 2.4 contém o mesmo grafo com configuração inicial onde os vértices com estado +1 são os mesmos da figura anterior com a adição de mais um vértice. Para esse caso, o processo 2-reversível termina com comportamento periódico.

Outra característica interessante é que qualquer conjunto conversor C para um processo k-reversível em G é necessariamente um conjunto conversor para o processo k-irreversível em G [8].



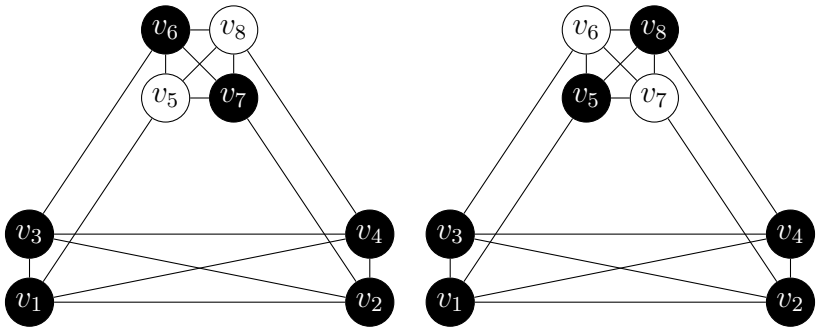
(a) $x(0)$: Conjunto conversor formado pelos vértices v_1, v_2, v_3, v_4, v_5

(b) $x(1)$

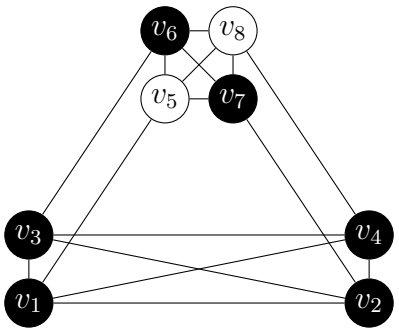


(c) $x(2)$

Figura 2.3: Processo 2-reversível a partir de um conjunto conversor.



(a) $x(0)$ (b) $x(1)$



(c) $x(2)$

Figura 2.4: Processo 2-reversível a partir de um super-conjunto de um conjunto conversor.

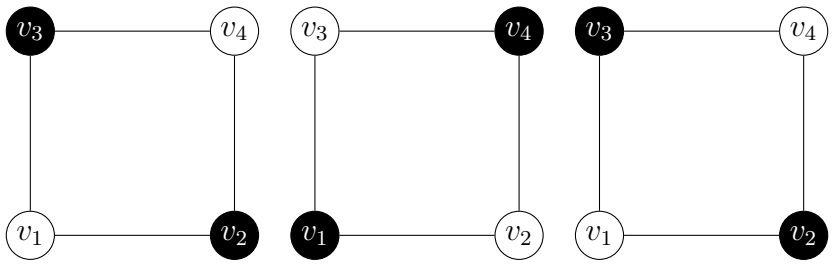
2.4 Transiente e Período

Em um processo k-reversível, o número de configurações em G é finito, e, portanto, dada uma configuração qualquer, após um número finito de passos, a configuração obtida é uma configuração que já foi alcançada anteriormente. Como em um processo k-reversível $x(t)$ é função de $x(t-1)$ para todo $t \geq 1$, uma vez que alcançamos uma configuração já vista, o processo entra em um comportamento periódico. Isso nos leva a dois atributos importantes desses processos: o transiente e o período de uma configuração.

Denominamos como período de uma configuração inicial o tamanho do ciclo de configurações alcançado a partir dessa configuração, e cada configuração pertecente ao ciclo chamaremos de *configuração periódica*. O transiente é definido como a quantidade de passos ocorridos até uma configuração presente em um ciclo de configurações ocorrer, dada a configuração inicial $x(0)$. De forma equivalente, o transiente é definido como o número de configurações que ocorrem fora do ciclo de configurações a partir de $x(0)$. Formalmente, definimos o comprimento do período $p(x(0))$ e o comprimento do transiente $\tau(x(0))$ como dois números inteiros tais que:

$$x(t + p(x(0))) = x(t) \text{ para qualquer } t \geq \tau(x(0))$$
$$x(t + q) \neq x(t) \text{ para qualquer } t < \tau(x(0)) \text{ ou } q < p(x(0))$$

Observando novamente a Figura 2.1, vemos que a configuração $x(2)$ é ponto fixo, ou seja, a partir de $x(2)$ não é gerada outra configuração diferente. Nesse caso, temos um comportamento periódico de tamanho 1. Como existem duas configurações ocorrendo antes da configuração $x(2)$, o transiente possui tamanho 2. Definimos assim, $p(x(0)) = 1$ e $\tau(x(0)) = 2$.



(a) $x(0)$ (b) $x(1)$ (c) $x(2)$

Figura 2.5: Exemplo de configurações periódicas de um processo 2-reversível em um C_4.

Na Figura 2.5, temos um C_4 com configuração inicial $x(0)$ para um processo 2-reversível. Observe que $x(2) = x(0)$, e, portanto, a própria configuração inicial $x(0)$ já é uma configuração periódica. Dessa forma, $p(x(0)) = 2$ e $\tau(x(0)) = 0$.

Teorema 2.2. *[15] Para processos que operam com limiares, caso a matriz A seja simétrica, o comprimento do período de uma configuração qualquer é no máximo 2.*

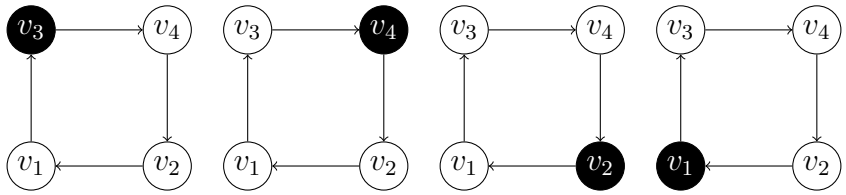
O Teorema 2.2 nos fornece diretamente o limite para o comprimento do período de processos k-reversíveis, k-irreversíveis e k_1-k_2-reversíveis. O Corolário 2.1 apresenta apenas o resultado para processos k-reversíveis, mas a ideia é idêntica para os outros processos.

Corolário 2.1. *Para processos k-reversíveis, o comprimento do período de uma configuração qualquer é no máximo 2.*

Demonstração. Utilizando o Teorema 2.2 e dado que um processo k-reversível é um caso específico de um processo que opera com limiar com matriz A simétrica, temos que o comprimento do período de uma configuração em um processo k-reversível também é no máximo dois. □

É interessante notar que esse resultado somente se aplica a processos k-reversíveis devido ao fato de definirmos esses processos sobre grafos não-direcionados. Caso contrário, por exemplo, se o processo fosse definido sobre grafos direcionados e cada vizinho mudasse de estado se e somente se possuísse pelo menos k vizinhos de entrada com estados opostos, o processo não seria um caso específico de um processo que

opera com limiar com a matriz A simétrica. De fato, se fosse um digrafo, o período de uma configuração pode ser maior que dois, como na Figura 2.6. Note que nesse exemplo assumimos $k = 1$, e o comprimento do período é 4.



(a) $x(0)$ (b) $x(1)$ (c) $x(2)$ (d) $x(3)$

Figura 2.6: Processo k-reversível em um digrafo pode possuir comprimento de período maior que 2; nesse exemplo, $k = 1$

A demostração do Teorema 2.2 dada por Goles e Olivos em [9] utiliza uma invariante algébrica e é uma prova bastante complexa. Já em [15], temos uma prova alternativa através da utilização de um operador monotônico que é denominado função de energia. Essa função é definida como:

$$E(x(t)) = -\frac{1}{2}\sum_{i=1}^{n} x_i \sum_{j=1}^{n} a_{ij}x_j + \sum_{i=1}^{n} b_i x_i \tag{2.7}$$

A função $E(x(t))$ é bastante similar à energia associada a redes de *Hopfield* [2] e é uma função de *Lyapunov*. Essa função é utilizada para provar diversos resultados associados ao comprimento do transiente e do período de processos que operam com limiares e processos majoritários. Modelos de funções similares são extensivamente estudados em Física.

Uma prova alternativa para o Teorema 2.2 é dada por Poljak e Sura [16]. Apesar de ser um prova independente, também é uma prova apenas algébrica que fornece pouco entendimento sobre a influência da dinâmica nos tamanhos do transiente e do período. No entanto, a técnica utilizada em [15] é de grande utilidade para demonstração de estabilidade de diversos sistemas, e serviu de inspiração para os resultados obtidos nesse trabalho.

Seja $T(A, b)$ o transiente máximo de uma configuração em um processo que opera com limiar que possui como parâmetros a matriz A e o vetor b:

Teorema 2.3. *[15] Se A é uma matriz simétrica, então $T(A, b) \leq \sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}| + 2\sum_{i=1}^{n} b_i$.*

Corolário 2.2. *Em um processo k-reversível, o transiente máximo é limitado por $2m + n(2\Delta(G) - 2)$.*

Demonstração. Utilizando o Teorema 2.3 e sabendo as definições do vetor b e da matriz A quando temos um processo k-reversível:

$$T(A,b) \leq 2m + \sum_{i=1}^{n} |2k - d(v_i) - 1|.$$

O valor máximo de cada parcela $|2k-d(v_i)-1|$ ocorre quando $k = \Delta(G)$ e $d(v_i) = 1$. Não consideramos $k > \Delta(G)$, pois nesse caso, qualquer configuração inicial para G já é periódica. Logo:

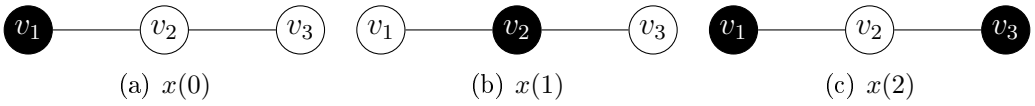
$$\begin{aligned} T(A,b) &\leq 2m + \sum_{i=1}^{n} |2k - d(v_i) - 1| \\ &\leq 2m + \sum_{i=1}^{n} (2\Delta(G) - 2) \\ &= 2m + n(2\Delta(G) - 2). \end{aligned} \tag{2.8}$$

□

Vemos, portanto, que o comprimento do transiente não cresce exponencialmente, apesar de existir um número exponencial de configurações possíveis. Em [8], é demonstrado um limite mais justo para o comprimento do transiente. O limite apresentado é $(4k-4)n$, porém a demonstração possui um erro nas contas realizadas. A partir de outros limites conhecidos na literatura, Dreyer conclui (2.9):

$$\begin{aligned} T(A,b) &\leq \frac{1}{2}(2m + \sum_{i=1}^{n} |2k - d(v_i) - 1| + 3(2k-2)n - n) \\ &\leq \frac{1}{2}(2m + 2kn - 2m - n + 6kn - 6n - n) \\ &= \frac{1}{2}(8kn - 8n) \\ &= (4k-4)n. \end{aligned} \tag{2.9}$$

É fácil observar o erro existente na passagem da primeira para a segunda linha da desigualdade, onde o somatório do módulo das expressões é tido como sendo menor ou igual ao somatório das expressões sem o operador de módulo. Esse erro acarreta em um limite para o transiente mais justo, porém, equivocado. Suponha o caso de um processo 1-reversível em um P_3, como na Figura 2.7. Pela fórmula, teríamos que não existe transiente, porém, a figura nos dá um exemplo que claramente possui transiente.



(a) $x(0)$ (b) $x(1)$ (c) $x(2)$

Figura 2.7: Exemplo de configurações $x(0)$, $x(1)$, $x(2)$ de um processo 1-reversível em um P_3. Note que existe transiente de comprimento 1.

Note, no entanto, que caso $2k > \Delta(G)$, (2.9) está correto. Porém, para esse caso, utilizando o Teorema 2.3, obtemos um resultado melhor contido no Corolário 2.3. Para ser mais preciso, o corolário oferece um resultado melhor caso $k > 1$. Se $k = 1$, o único grafo possível é um P_2, onde um processo 1-reversível claramente não possui transiente, em conformidade com a expressão $(4k-4)n$, que resulta no valor 0. Pelo resultado obtido no Corolário 2.3, limitamos o transiente a 1, e, portanto, para esse caso, o limite é pior. Para $k > 1$, é fácil ver que temos um limite mais justo.

Corolário 2.3. *Em um processo k-reversível, se* $2k > \Delta(G)$ *então o comprimento máximo do transiente é limitado por* $(2k-1)n$.

Demonstração.

$$\begin{aligned} T(A,b) &\leq 2m + \sum_{i=1}^{n} |2k - d(v_i) - 1| \\ &= 2m + 2kn - 2m - n \\ &= 2kn - n \\ &= (2k-1)n. \end{aligned} \tag{2.10}$$

□

Questões interessantes sobre o transiente de processos k-reversíveis giram em torno da determinação de limites superiores mais apertados que os apresentados nesse capítulo. Com relação ao período, podemos perguntar se é possível determinar o comprimento do período de uma configuração de forma mais eficiente que a simulação do processo k-reversível.

Um ponto bastante interessante sobre os resultados já obtidos é o fato de, apesar da existência de 2^n configurações de estados, um simples algoritmo de simulação possui complexidade de tempo polinomial para detectar as configurações que fazem parte do período e para determinar o transiente de uma determinada configuração. Mais precisamente, assumindo o valor demonstrado pelo Teorema 2.3 como sendo o valor máximo para o comprimento do transiente, e notando que um passo da dinâmica pode ser executado em tempo $\mathrm{O}(n+m)$, concluímos que um algoritmo simples de simulação possui complexidade $O((n+m)m+(n+m)n\Delta(G)) = \mathrm{O}(m^2 + n^2m)$.

Capítulo 3

Função de Energia em Processos k-Reversíveis

No Capítulo 2, apresentamos a função de energia utilizada para provar diversos resultados importantes em processos que operam com limiares. Os principais resultados obtidos são limites, nem sempre justos, para o comprimento máximo do transiente e do período que ocorrem em tais processos. Com essa mesma definição de função de energia, outros resultados importantes também são demonstrados, como, por exmplo, para os casos onde os processos ocorrem de forma sequencial, ou seja, quando existe uma permutação que determina a ordem pela qual a alteração dos estados dos vértices será realizada.

Nesse capítulo, introduzimos uma função mais adequada e intuitiva para processos k-reversíveis. A ideia é que com essa nova função possamos obter resultados mais específicos para esses processos e também maior entendimento sobre a dinâmica do processo, visto que a função de energia que foi apresentada trata o caso mais genérico de processos que operam com limiares e não é intuitiva para entender processos mais específicos como os processos k-reversíveis. Com essa nova função, pretendemos ter mais entendimento sobre esses processos e obter limites mais justos e demonstrações mais óbvias para os comprimentos máximos do transiente e do período.

Ainda nesse capítulo, enunciaremos uma conjectura de um limite superior justo para o transiente de processos 2-reversíveis em árvores. Para essa conjectura, mostramos inclusive o método de construção das árvores que atingem tal limite.

3.1 Energia em Processos k-Reversíveis

Inicialmente vamos definir dois conjuntos que serão referenciados ao longo desse capítulo e serão utilizados em basicamente todas as demonstrações. Esses conjuntos

chamaremos de $S_1(t)$ e $S_2(t)$:

$$S_1(t) = \{v_i \mid x_i(t+1) \neq x_i(t)\}$$
$$S_2(t) = \{v_i \mid x_i(t+1) = x_i(t)\}$$

Alternativamente, podemos definir:

$$S_1(t) = \{v_i \mid op_i(t) \geq k\}$$
$$S_2(t) = \{v_i \mid op_i(t) < k\}$$

Como podemos ver, definimos os conjuntos de tal forma que o conjunto $S_1(t)$ contém todos os vértices que tiveram seus estados alterados entre as configuraçãos $x(t)$ e $x(t+1)$. Por sua vez, o conjunto $S_2(t)$ contém todos os vértices que mantiveram seus estados entres essas configurações.

Utilizando esses dois conjuntos, formulamos uma função que denominaremos *função de energia para processos k-reversíveis* e a denotaremos por $E(t)$. Essa função se baseia na função de energia apresentada em (2.7) e é definida da seguinte forma:

$$E(t) = \sum_{i \in S_1(t)} (op_i(t) - k) + \sum_{i \in S_2(t)} (k - op_i(t)). \tag{3.1}$$

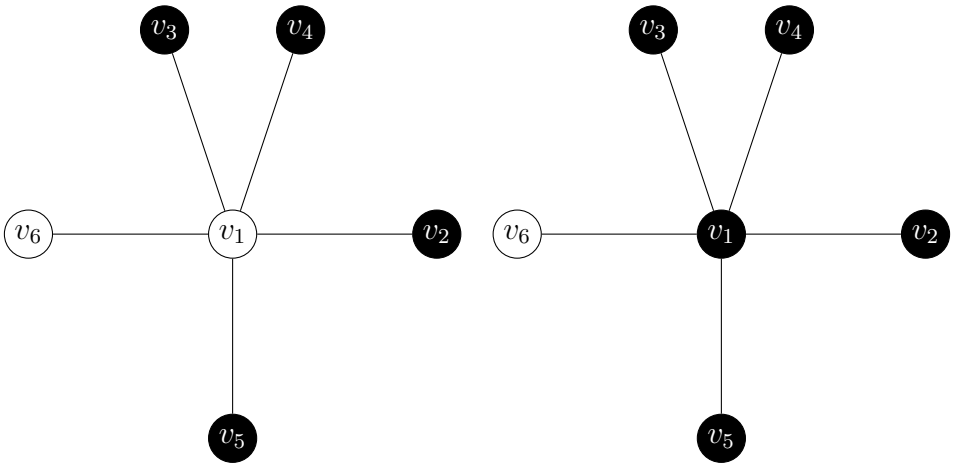
Essa nova função de energia pode ser entendida como um *saldo* de vizinhos que sobraram ou faltaram para que ocorressem mais mudanças de estados a partir da configuração $x(t)$. No primeiro somatório, cada parcela $(op_i(t) - k)$ representa a quantidade de vizinhos a mais com estado oposto que v_i possui para mudar de estado, visto que v_i está no conjunto $S_1(t)$. Já no segundo somatório, cada parcela $(k - op_i(t))$ representa o número de vizinhos com estado oposto que eram necessários a mais para v_i mudar seu estado entre as configuraçãos $x(t)$ e $x(t+1)$.

Observe a Figura 3.1 que contém um grafo G, que é um grafo estrela com 6 vértices, e as configurações $x(0)$ e $x(1)$ ilustradas em um processo 3-reversível. Para o cálculo de $E(0)$, temos os seguintes valores:

- $op_1(0) = 4$, $op_2(0) = 1$, $op_3(0) = 1$, $op_4(0) = 1$, $op_5(0) = 1$ e $op_6(0) = 0$.
- $S_1(0) = \{v_1\}$ e $S_2(0) = \{v_2, v_3, v_4, v_5, v_6\}$.

Utilizando esses valores e a Definição 3.1, obtemos $E(0) = 12$. Para o cálculo de $E(1)$, temos:

- $op_1(1) = 1$, $op_2(1) = 0$, $op_3(1) = 0$, $op_4(1) = 0$, $op_5(1) = 0$ e $op_6(1) = 1$.



(a) $x(0)$: $E(0) = 12$ (b) $x(1)$: $E(1) = 16$

Figura 3.1: Exemplo de configurações $x(0)$ e $x(1)$ de um processo 3-reversível em um grafo estrela.

- $S_1(1) = \emptyset$ e $S_2(1) = V(G)$.

Novamente, utilizando os valores obtidos e a Definição 3.1, obtemos $E(1) = 16$.

Pela definição, é possível observar que a função de energia nunca assume valores negativos. O primeiro somatório claramente nunca assume valores negativos visto que todos os vértices em $S_1(t)$ possuem $op_i(t) \geq k$. Para o segundo somatório, todos os veŕtices em $S_2(t)$ possuem $op_i(t) < k$, e, portanto, a menos que o conjunto seja vazio, o valor desse somatório é maior que zero.

Uma outra definição equivalente para essa função de energia é dada por:

$$E(t) = \sum_{i \in S_1(t)} (op_i(t+1) - k) + \sum_{i \in S_2(t)} (k - op_i(t+1)). \tag{3.2}$$

Apesar de possuir uma forma similar à Definição 3.1, apenas olhando para essa nova definição não é claro que ela assuma apenas valores não-negativos e muito menos que as duas definições sejam equivalentes. Como exemplo, vamos usar a Definição 3.2 para o cálculo de $E(0)$ e $E(1)$ para o caso da Figura 3.1. Temos:

- $op_1(1) = 1$, $op_2(1) = 0$, $op_3(1) = 0$, $op_4(1) = 0$, $op_5(1) = 0$ e $op_6(1) = 1$.
- $S_1(0) = \{v_1\}$ e $S_2(0) = \{v_2, v_3, v_4, v_5, v_6\}$.

Como em $x(1)$ não há vértices com mais de um vizinho com estado oposto, então sabemos que $x(2) = x(1)$, e assim calculamos:

- $op_1(2) = 1$, $op_2(2) = 0$, $op_3(2) = 0$, $op_4(2) = 0$, $op_5(2) = 0$ e $op_6(2) = 1$.
- $S_1(1) = \emptyset$ e $S_2(1) = V(G)$.

Obtemos, assim, $E(0) = 12$ e $E(1) = 16$, que são exatamente os valores que havíamos encontrado anteriormente com a Definição 3.1.

Ao contrário da definição anterior, essa nova versão não possui, à primeira vista, um significado claro, pois, apesar de mantermos os mesmos conjuntos $S_1(t)$ e $S_2(t)$ para o cálculo, também utilizamos os parâmetros $op_i(t+1)$. Inclusive, devido a isso, podemos agora ter valores negativos nas parcelas, como acontece no exemplo utilizado ao calcular $E(0)$ com a parcela referente ao vértice v_1.

Por essa dificuldade, a primeira versão será a mais utilizada ao longo do capítulo e será utilizada como definição padrão. Mas a segunda versão será primordial em alguns resultados para os quais necessitaremos de algumas observações que utilizam ambas as versões.

Antes de prosseguir, no entanto, precisamos demonstrar a equivalência das funções.

Lema 3.1. *As Definições 3.1 e 3.2 para o cálculo da função de energia em processos k-reversíveis são equivalentes.*

Demonstração. Sejam $|S_1(t)|$ e $|S_2(t)|$ as cardinalidades dos conjuntos $S_1(t)$ e $S_2(t)$, respectivamente. Reescrevemos (3.1):

$$E(t) = \sum_{i \in S_1(t)} op_i(t) - \sum_{i \in S_2(t)} op_i(t) - |S_1(t)|k + |S_2(t)|k. \tag{3.3}$$

Reescrevemos também (3.2):

$$E(t) = \sum_{i \in S_1(t)} op_i(t+1) - \sum_{i \in S_2(t)} op_i(t+1) - |S_1(t)|k + |S_2(t)|k.$$

Dessa forma, como a expressão $-|S_1(t)|k + |S_2(t)|k$ aparece nas duas fórmulas, precisamos mostrar que:

$$\sum_{i \in S_1(t)} op_i(t) - \sum_{i \in S_2(t)} op_i(t) = \sum_{i \in S_1(t)} op_i(t+1) - \sum_{i \in S_2(t)} op_i(t+1).$$

Vamos definir os seguintes conjuntos $A(t)$, $B(t)$ e $C(t)$:

- $A(t) = \{(v_i, v_j) \mid (v_i, v_j) \in E(G), v_i \in S_1(t), v_j \in S_1(t) \text{ e } x_i(t) \neq x_j(t)\}$
- $B(t) = \{(v_i, v_j) \mid (v_i, v_j) \in E(G), v_i \in S_2(t), v_j \in S_2(t) \text{ e } x_i(t) \neq x_j(t)\}$
- $C(t) = \{(v_i, v_j) \mid (v_i, v_j) \in E(G) \setminus (A(t) \cup B(t)) \text{ e } x_i(t) \neq x_j(t)\}$

Assim, $A(t)$ é o conjunto de todas arestas entre vértices de $S_1(t)$ que possuem estados distintos no passo de tempo t, $B(t)$ o conjunto de todas as arestas entre vértices de $S_2(t)$ que possuem estados distintos no passo de tempo t, e $C(t)$ o conjunto de todas as outras arestas entre vértices que possuam estados distintos no passo de tempo t.

Sejam $|A(t)|$, $|B(t)|$ e $|C(t)|$ as cardinalidades de tais conjuntos. Não é difícil ver que $\sum_{i \in S_1(t)} op_i(t)$ é igual a $2|A(t)|+|C(t)|$. Para verificar esse resultado basta observar que para dois vértices vizinhos v_i e v_j com estados opostos no passo de tempo t, a aresta entre eles é contabilizada tanto por $op_i(t)$ quanto por $op_j(t)$. Se ambos os vértices pertencem ao conjunto $S_1(t)$, então essa aresta será contabilizada duas vezes no somatório. Caso apenas um dos vértices pertença ao conjunto $S_1(t)$, então a aresta será contabilizada apenas uma vez no somatório. De maneira semelhante, temos $\sum_{i \in S_2(t)} op_i(t)$ igual a $2|B(t)| + |C(t)|$. Logo, temos que:

$$\sum_{i \in S_1(t)} op_i(t) - \sum_{i \in S_2(t)} op_i(t) = 2|A(t)| - 2|B(t)|. \tag{3.4}$$

Definimos agora os conjuntos $A'(t)$, $B'(t)$ e $C'(t)$:

- $A'(t) = \{(v_i, v_j) \mid (v_i, v_j) \in E(G), v_i \in S_1(t), v_j \in S_1(t) \text{ e } x_i(t+1) \neq x_j(t+1)\}$
- $B'(t) = \{(v_i, v_j) \mid (v_i, v_j) \in E(G), v_i \in S_2(t), v_j \in S_2(t) \text{ e } x_i(t+1) \neq x_j(t+1)\}$
- $C'(t) = \{(v_i, v_j) \mid (v_i, v_j) \in E(G) \setminus (A'(t) \cup B'(t)) \text{ e } x_i(t+1) \neq x_j(t+1)\}$

A única diferença desses conjuntos para os conjuntos $A(t)$, $B(t)$ e $C(t)$ é que, nesses, consideramos estados distintos no tempo $t+1$, em vez do tempo t. Com isso podemos escrever $\sum_{i \in S_1(t)} op_i(t+1)$ como $2|A'(t)| + |C'(t)|$. E podemos também escrever $\sum_{i \in S_2(t)} op_i(t+1)$ como $2|B'(t)| + |C'(t)|$. E, portanto:

$$\sum_{i \in S_1(t)} op_i(t+1) - \sum_{i \in S_2(t)} op_i(t+1) = 2|A'(t)| - 2|B'(t)|.$$

A Figura 3.2 ilustra os conjuntos de arestas $A(t)$, $B(t)$ e $C(t)$ nas cores azul, vermelha e preta, respectivamente. Note que inúmeras outras arestas podem existir no grafo, mas somente as arestas entre vértices com estados distintos no tempo t são representadas nos conjuntos. A Figura 3.3 ilustra os conjuntos de arestas $A'(t)$, $B'(t)$ e $C'(t)$ nas cores azul, vermelha e preta, respectivamente.

Observe que, na verdade, os conjuntos $A(t)$ e $A'(t)$ são iguais. Isso acontece por que todos os vértices que estão em $S_1(t)$ irão mudar de estado no passo t. Sendo assim, se v_i e v_j são dois vértices de $S_1(t)$ com $x_i(t) \neq x_j(t)$, então, certamente,

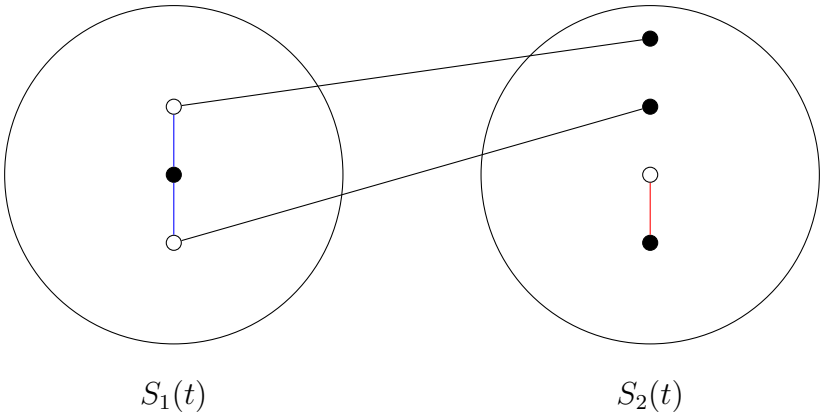


Figura 3.2: Exemplo de conjuntos de arestas $A(t), B(t)$ e $C(t)$.

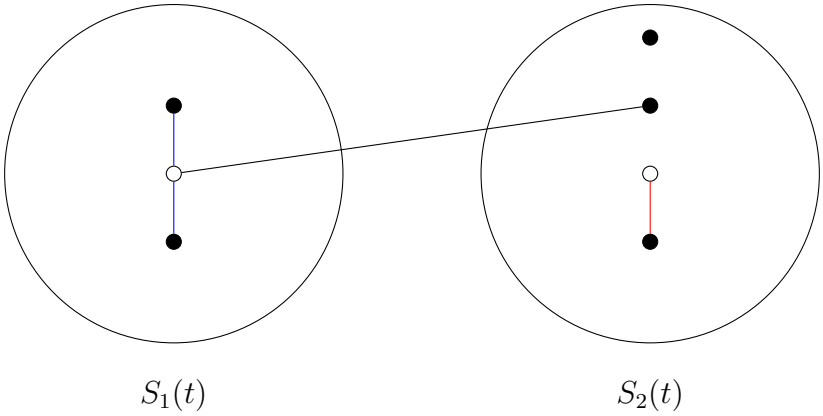


Figura 3.3: Exemplo de conjuntos de arestas $A'(t), B'(t)$ e $C'(t)$.

temos $x_i(t+1) \neq x_j(t+1)$. O mesmo vale para os conjuntos $B(t)$ e $B'(t)$. Para dois vértices v_i e v_j em $S_2(t)$ com $x_i(t) \neq x_j(t)$, teremos, certamente, $x_i(t+1) \neq x_j(t+1)$. Dessa forma, temos finalmente que:

$$\sum_{i \in S_1(t)} op_i(t+1) - \sum_{i \in S_2(t)} op_i(t+1) = 2|A(t)| - 2|B(t)|.$$

Concluímos assim, que (3.1) e (3.2) são equivalentes. □

Como já foi visto no Capítulo 2, a função de energia definida em (2.7) para o caso geral de processos que operam como limiares é uma função monôtonica decrescente. Essa característica é de grande ajuda na demonstração de que o comprimento máximo do período nesses processos é dois.

Veremos que a monotonicidade também é uma característica da função de energia específica para processos k-reversíveis. Nesse caso, a função é monôtonica crescente, já que não temos o sinal negativo nos somatórios.

Lema 3.2. *A função $E(t)$ definida em (3.1) é um operador monotônico crescente.*

Demonstração. Vamos incluir a notação $\Delta E(t)$, que será a variação de energia do tempo t para o tempo $t+1$:

$$\Delta E(t) = E(t+1) - E(t)$$

Podemos usar a Definição 3.1 para reescrever $E(t+1)$, e a Definição 3.2 para reescrever $E(t)$:

$$\Delta E(t) = \sum_{i \in S_1(t+1)} (op_i(t+1) - k) + \sum_{i \in S_2(t+1)} (k - op_i(t+1)) - \sum_{i \in S_1(t)} (op_i(t+1) - k) - \sum_{i \in S_2(t)} (k - op_i(t+1)).$$

Iremos mostrar que $\Delta E(t) \geq 0$. Para demonstrar isso, analisaremos a contribuição em $\Delta E(t)$ de cada vértice de G. Observando a fórmula expandida de $\Delta E(t)$ é possível separar quatro casos para cada vértice v_i:

- $v_i \in S_1(t)$ e $v_i \in S_1(t+1)$.
  - Nesse caso, a contribuição de v_i é dada por $op_i(t+1) - k - op_i(t+1) + k = 0$.
- $v_i \in S_2(t)$ e $v_i \in S_2(t+1)$.
  - Nesse caso, a contribuição de v_i é dada por $k - op_i(t+1) - k + op_i(t+1) = 0$.
- $v_i \in S_1(t)$ e $v_i \in S_2(t+1)$.
  - Nesse caso, a contribuição de v_i é dada por $k - op_i(t+1) - op_i(t+1) + k = 2(k - op_i(t+1))$.
  - Como $k > op_i(t+1)$, então, $2(k - op_i(t+1)) > 0$.
- $v_i \in S_2(t)$ e $v_i \in S_1(t+1)$.
  - Nesse caso, a contribuição de v_i é dada por $op_i(t+1) - k - k + op_i(t+1) = 2(op_i(t+1) - k)$.
  - Como $op_i(t+1) \geq k$, então, $2(op_i(t+1) - k) \geq 0$.

Note que nenhum vértice em G contribui com valores negativos em $\Delta E(t)$. Logo, concluímos que $\Delta E(t) \geq 0$.

□

A partir da demonstração do Lema 3.2 podemos observar também que sempre que $\Delta E(t) > 0$, então $\Delta E(t) \geq 2$. Observamos também que a variação de energia é sempre um número par.

Ainda observando a demonstração desse lema, podemos ver que caso não exista nenhum vértice na terceira ou quarta possibilidades de comportamentos enunciados na demonstração, então a configuração $x(t)$ é uma configuração periódica, e o período possui comprimento no máximo dois.

Suponha, por exemplo, que um vértice v_i esteja em $S_1(t)$ e em $S_1(t+1)$; logo, $x_i(t) \neq x_i(t+1)$ e $x_i(t+1) \neq x_i(t+2)$, e portanto $x_i(t) = x_i(t+2)$. Caso o vértice v_i esteja em $S_2(t)$ e em $S_2(t+1)$, temos que $x_i(t) = x_i(t+1)$, e, por sua vez, $x_i(t+1) = x_i(t+2)$. Logo se todos os vértices se encontram nessas duas possibilidades, estamos em uma configuração periódica, tal que o comprimento do período é no máximo dois. Nesse caso, se existir pelo menos um vértice no primeiro comportamento, então a configuração pertence a um período de comprimento dois, caso contrário, se todos os vértices estiverem no segundo comportamento, então a configuração pertence a um período de comprimento um.

Para qualquer vértice v_i tal que $v_i \in S_1(t)$ e $v_i \in S_2(t+1)$, temos $x_i(t) \neq x_i(t+1)$ e $x_i(t+1) = x_i(t+2)$. E para qualquer vértice v_i tal que $v_i \in S_2(t)$ e $v_i \in S_1(t+1)$, então temos $x_i(t) = x_i(t+1)$ e $x_i(t+1) \neq x_i(t+2)$. Sendo assim, a única situação onde a configuração é periódica e o período possui comprimento dois ocorre quando todos os vértices se encontram em um dos dois primeiros comportamentos descritos na demonstração do Lema 3.2.

3.2 A Função de Energia como Ferramenta de Demonstração

A função de energia para processos k-reversíveis, como veremos, é bastante útil para provarmos de maneira mais intuitiva o limite para o comprimento máximo do período de uma dinâmica de um processo k-reversível. Além disso, fornece limites interessantes também para o comprimento máximo do transiente de tais processos.

Teorema 3.1. *Para qualquer grafo simples G e qualquer configuração inicial $x(0)$ de um processo k-reversível, $p(x(0)) \leq 2$.*

Demonstração. Pelo Lema 3.2, sabemos que a função $E(t)$ é monotônica crescente. Pela Definição 3.1 e pelo fato de G ser finito, sabemos também que $E(t)$ não pode crescer indefinidamente, ou seja, existe um passo de tempo t_{max} tal que $E(t) = E(t_{max})$ para todo $t > t_{max}$.

Seja um grafo G e uma configuração inicial $x(0)$ para um processo k-reversível. Definimos t_0 como sendo um instante de tempo grande o suficiente tal que $t_0 > \tau(x(0))$ e $t_0 > t_{max}$. É importante notar que essa situação ocorre para todos os processos k-reversíveis, já que um processo pode ocorrer por tanto tempo quanto se queira. Vamos supor, ainda, que $p(x(0)) > 2$.

Como estamos assumindo que a função de energia já alcançou o valor máximo $E(t_{max})$, para todo $t > t_0$ não existe vértice v_i tal que $v_i \in S_1(t)$ e $v_i \in S_2(t+1)$, pois dessa forma, como vimos na demonstração do Lema 3.2, isto implicaria $\Delta E(t) > 0$, contradizendo a hipótese de que a função já havia atingido seu valor máximo. Também é claro que não pode ser o caso onde em um tempo $t > t_0$, para todo vértice v_i, ocorra $v_i \in S_1(t)$ e $v_i \in S_1(t+1)$ ou ocorra $v_i \in S_2(t)$ e $v_i \in S_2(t+1)$, pois dessa forma todos os vértices estariam em comportamento periódico de comprimento no máximo dois, como já foi visto anteriormente e como estamos supondo não ser o caso. Concluímos que, para termos $p(x(0)) > 2$, certamente existe pelo menos um vértice v_i tal que $v_i \in S_2(t)$ e $v_i \in S_1(t+1)$ para que $x_i(t) \neq x_i(t+2)$. Mais precisamente, existe um vértice v_i tal que $op_i(t) < k$ e $op_i(t+1) = k$. Note que é necessária a condição $op_i(t+1) = k$, pois caso a condição fosse $op_i(t+1) > k$, teríamos um aumento de no mínimo duas unidades na variação da função de energia.

Mas como $v_i \in S_1(t+1)$, a partir do tempo $t+1$ v_i necessariamente continuará em $S_1(t+2)$, $S_1(t+3)$, $S_1(t+4)$ e assim por diante, pois caso esteja em algum $S_2(t')$, para algum $t' > t+1$, então, como já vimos na demonstração do Lema 3.2, o valor da energia aumentaria, contradizendo a hipótese $t > t_{max}$. Dessa forma, temos $x_i(t) \neq x_i(t+2)$ e $x_i(t'+2) = x_i(t')$, para todo $t' > t$. Portanto, v_i entrou em comportamento periódico somente no tempo $t > t_0$, contradizendo a hipótese inicial $t_0 > \tau(x(0))$.

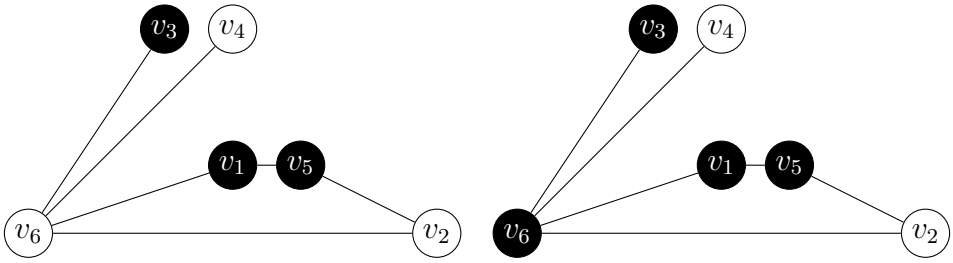
Sendo assim, só existem vértices com comportamento periódico de comprimento no máximo dois, e, portanto, concluímos que $p(x(0)) \leq 2$, para qualquer configuração $x(0)$ e grafo G.

□

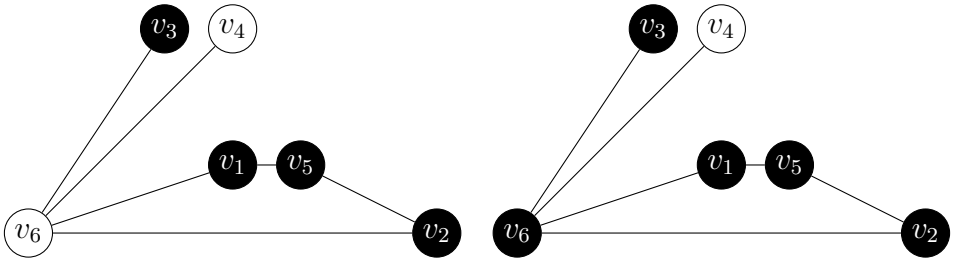
Para o estudo do comprimento máximo do transiente em um processo k-reversível a partir de uma configuração inicial $x(0)$, devemos notar que não podemos ter $\Delta E(t) > 0$, para $t \geq \tau(x(0))$. Note que, se isso ocorresse, teríamos $E(t+2) > E(t)$, mas isso é impossível, visto que $x(t+2) = x(t)$, e, portanto, o cálculo de $E(t+2)$ e $E(t)$ resultam no mesmo valor.

Concluímos assim que o comprimento do transiente possui alguma relação com o valor máximo de energia $E(t_{max})$. Se $E(t)$ fosse um operador monotônico estritamente crescente, teríamos um resultado direto limitando o comprimento máximo do transiente ao valor $E(t_{max})$. Na verdade, o resultado limitaria o comprimento do transiente ao valor $\frac{E(t_{max})}{2}$, já que o crescimento da função ocorre em passos de no mínimo duas unidades. Mas, como podem existir intervalos de tempo onde $\Delta E(t) = 0$ e $t < \tau(x(0))$, como na Figura 3.4, então essa afirmação não pode ser feita. Nessa figura, por exemplo, que ilustra um processo 2-reversível, observe que $\Delta E(0) = 0$, mas $\tau(x(0)) = 3$. O problema então é termos grandes intervalos de

tempo com a variação de energia igual a zero mas ainda longe de uma configuração periódica.



(a) $x(0)$: $E(0) = 6$ (b) $x(1)$: $E(1) = 6$



(c) $x(2)$: $E(2) = 8$ (d) $x(3)$: $E(3) = 10$

Figura 3.4: Configurações $x(0)$, $x(1)$, $x(2)$ e $x(3)$ para um processo 2-reversível.

Podemos observar, no entanto, que existe apenas uma situação onde em algum intervalo de uma unidade de tempo a função de energia mantém seu valor constante sem que a configuração atual seja de fato uma configuração periódica. E essa situação ocorre somente quando existe no mínimo um vértice v_i tal que $v_i \in S_2(t)$ e $v_i \in S_1(t+1)$ com $op_i(t+1) = k$. Logo, podemos concluir que o limite máximo de passos tal que a função de energia permaneça com o mesmo valor é igual a $n-1$. Essa situação extrema ocorreria, por exemplo, se existirem os conjuntos $S_1(t')$, $S_1(t'+1)$, $\ldots$, $S_1(t'+n-1)$ e o conjuntos $S_2(t')$, $S_2(t'+1)$, $\ldots$, $S_2(t'+n-1)$ tais que:

- $|S_1(t')| = 1$
- $|S_2(t')| = n-1$
- $S_1(t') \subsetneq S_1(t'+1) \subsetneq \ldots \subsetneq S_1(t'+n-1)$
- $S_2(t') \supsetneq S_2(t'+1) \supsetneq \ldots \supsetneq S_2(t'+n-1)$
- Se $v_i \in S_2(j) \cap S_1(j+1)$, então $op_i(j+1) = k$.

Note que, dessa forma, teríamos exatamente $n-1$ passos que terminariam em uma configuração periódica com todos os vértices em $S_1(t'+n-1)$. Isso nos dá diretamente um limite superior igual a $\frac{E(t_{max})(n-1)}{2}$ para o comprimento máximo do transiente. Mas para sermos um pouco mais precisos, devemos considerar apenas

$n-2$ passos, pois o último passo resultaria em uma configuração periódica. Apenas para o valor de energia máximo devemos considerar os $n-1$ passos, pois, obviamente, não há mais para onde aumentar. Portanto, podemos limitar o comprimento máximo do transiente a $\frac{(E(t_{max})-2)(n-2)}{2} + (n-1)$.

Esse limite no entanto está longe de ser um limite justo. Mesmo sem uma análise mais detalhada, parece ser bastante improvável que exista alguma configuração onde a função de energia cresça somente em passos de duas unidades, e permaneça em cada valor possível por $n-2$ passos até alcançar o valor $E(t_{max})$ e permanecer nesse valor por $n-1$ passos até atingir a configuração periódica. E isso de fato não acontece. Apesar de a função poder ficar com valor constante por alguns intervalos de tempo, o crescimento ao longo do tempo até o valor máximo não chega nem perto do extremo de demorar $\frac{(E(t_{max})-2)(n-2)}{2} + (n-1)$ passos. Um limite mais justo é demonstrado pelo Teorema 3.2.

Teorema 3.2. *Para qualquer grafo simples G e qualquer configuração inicial $x(0)$ de um processo k-reversível, $\tau(x(0)) \leq E(t_{max}) + n - 1$.*

Demonstração. Observando cada passo onde $E(t)$ permanece constante, já verificamos que existe pelo menos um vértice v_i tal que $v_i \in S_2(t)$ e $v_i \in S_1(t+1)$ com $op_i(t+1) = k$. Mas para essa situação ocorrer, as possibilidades são bem poucas e podem ser separadas em dois casos:

- v_i pertence a todos os conjuntos $S_2(0)$, $S_2(1)$, ..., $S_2(t)$.
- Existe algum tempo $0 < t' < t$ tal que $v_i \in S_1(t')$ e $v_i \in S_2(t'+1)$, $v_i \in S_2(t'+2)$, $\ldots v_i \in S_2(t)$.

Note que apenas no segundo caso o vértice v_i contribuiu em algum momento anterior para o aumento do valor da função em pelo menos duas unidades, pois temos que $v_i \in S_1(t')$ e $v_i \in S_2(t'+1)$. Sendo assim, é possível concluir que, excetuando-se os vértices que estão no primeiro caso, todos os outros vértices que contribuem para a manutenção do valor da função constante após uma unidade de tempo, também contribuíram para o aumento do valor dessa mesma função em pelo menos duas unidades em algum passo anterior de tempo. Dessa forma, ainda desconsiderando todos os vértices que estão no primeiro caso, vemos que a média do crescimento da energia ao longo do tempo é de no mínimo uma unidade por passo de tempo, mas na maioria dos casos esse aumento será ainda maior, pois o mais comum é termos aumentos significativos entre cada passo de tempo.

Voltando a considerar os vértices do primeiro caso, vemos que esses vértices contribuem para a manutenção do valor de energia constante sem terem contribuído com o aumento da função em algum passo anterior de tempo. Mas essa situação pode

ocorrer somente uma vez para cada vértice, que é justamente no passo de tempo t. Após t, qualquer contribuição de um vértice para a manutenção do valor da função terá necessariamente como custo em algum passo anterior o aumento desse valor em pelo menos duas unidades. Podemos então limitar o valor máximo do transiente a $E(t_{max}) + |S_2(0)|$.

Mas se $|S_2(0)| = n$, então a configuração inicial já é uma configuração periódica e, portanto, o comprimento do transiente nesse caso é zero. Logo, de maneira mais geral, podemos limitar o valor máximo do transiente a $E(t_{max}) + n - 1$. □

O resultado do Teorema 3.2 ainda não deixa muito claro o limite máximo do transiente em processos k-reversíveis, pois ainda não vimos a relação de $E(t_{max})$ com os parâmetros do grafo e com o parâmetro k.

Corolário 3.1. *Para qualquer grafo simples G e qualquer configuração inicial $x(0)$ de um processo k-reversível, $\tau(x(0)) \leq n(\Delta(G) + 1) - 1$.*

Demonstração. Sabemos que se $k > \Delta(G)$, o comprimento do transiente de qualquer configuração é igual a 0. Então, assumindo $k \leq \Delta(G)$, e usando a Definição 3.1 da função de energia, temos que o valor máximo que $E(t)$ pode assumir ocorre quando $k = \Delta(G)$ e $S_2(t) = V(G)$. Sendo assim, $E(t_{max}) = n\Delta(G)$. Note que em (3.1) o valor máximo de cada parcela nunca é maior que $\Delta(G)$. Utilizando o Teorema 3.2, temos:

$$\begin{aligned} \tau(x(0)) &\leq E(t_{max}) + n - 1 \\ &\leq n\Delta(G) + n - 1 \\ &\leq n(\Delta(G) + 1) - 1. \end{aligned} \tag{3.5}$$

□

Note que o Corolário 3.1 nos fornece um limite melhor para o transiente de uma configuração em um processo k-reversível do que o limite obtido pelo Corolário 2.2. De maneira semelhante ao que fizemos no Capítulo 2, analisamos o caso onde $2k > \Delta(G)$.

Corolário 3.2. *Para qualquer grafo simples G e qualquer configuração inicial $x(0)$ de um processo k-reversível tal que $2k > \Delta(G)$, vale $\tau(x(0)) \leq (k + 1)n - 1$.*

Demonstração. É fácil ver que se $2k > \Delta(G)$, nenhuma parecela dos somatórios em (3.1) é maior que k, e, portanto a energia máxima é nk. Utilizando o Teorema 3.2, concluímos:

$$\begin{aligned}\tau(x(0)) &\leq nk + n - 1 \\ &= (k+1)n - 1. \end{aligned} \tag{3.6}$$

□

Novamente, através da utilização da função de energia, obtemos um limite mais justo para o comprimento do transiente. Observe que o Corolário 3.2 fornece um limite menor do que o limite obtido pelo Corolário 2.3 para todo $k > 1$.

3.3 Árvores

Nessa seção, mostramos um limite justo para o valor máximo da função de energia quando o grafo é uma árvore. Com isso, o limite superior para o comprimento do transiente em processos k-reversíveis também fica mais apertado. Na última parte da seção enunciamos uma conjectura para limite máximo justo do transiente em processos 2-reversíveis em árvores.

Teorema 3.3. *Quando G é uma árvore, $E_{max} = nk$.*

Demonstração. Usando (3.3) e (3.4):

$$E(t) = 2(|A(t)| - |B(t)|) + k(|S_2(t)| - |S_1(t)|).$$

Queremos maximizar $E(t)$. Note que se $|S_2(t)| = n$ e $|S_1(t)| = 0$, para maximizar $E(t)$ assumimos que todos os vértices em $S_2(t)$ possuem o mesmo estado, e, portanto, $|B(t)| = 0$. Como $S_1(t)$ não possui nenhum vértice, $|A(t)| = 0$:

$$\begin{aligned} E(t) &= 2(|A(t)| - |B(t)|) + k(|S_2(t)| - |S_1(t)|) \\ &= 2(0 - 0) + k(n - 0) \\ &= nk. \end{aligned} \tag{3.7}$$

Generalizando, vamos supor $|S_1(t)| = w$, $|S_2(t)| = n - w$ e $w > 0$. Note que para maximizar o valor da função devemos sempre considerar todos os vértices em $S_2(t)$ com mesmo estado, para sempre termos $|B(t)| = 0$. De maneira semelhante, devemos considerar $|A(t)| = |S_1(t)| - 1$, que é o maior número possível de arestas entre vértices de $S_1(t)$, pois G é uma árvore. Sendo assim, o valor máximo possível para $E(t)$ é:

$$\begin{aligned} E(t) &= 2(|A(t)| - |B(t)|) + k(|S_2(t)| - |S_1(t)|) \\ &= 2(w-1) + k(n-w-w) \\ &= 2w - 2 + kn - 2kw \\ &= kn + 2w - 2 - 2kw \\ &= kn + w(2-2k) - 2. \end{aligned} \tag{3.8}$$

Para termos $E(t) > nk$, devemos ter $w(2-2k) > 2$. Mas, para qualquer $k \geq 1$, $w(2-2k) \leq 0$. Sendo assim, caso $w > 0$, não existe configuração $x(t)$ tal que $E(t) > nk$. Dessa forma, concluímos que o valor máximo de energia para árvores ocorre quando todos os vértices possuem o mesmo estado. E esse valor é igual a nk. □

Corolário 3.3. *Para processos k-reversíveis onde o grafo G é uma árvore, e para qualquer configuração inicial x, vale $\tau(x) \leq (k+1)n - 1$.*

Demonstração. O resultado segue diretamente dos Teoremas 3.2 e 3.3:

Para qualquer configuração inicial x, $\tau(x) \leq E(t_{max}) + n - 1$. Como $E(t_{max}) \leq nk$:

$$\begin{aligned} \tau(x) &\leq nk + n - 1 \\ &\leq (k+1)n - 1. \end{aligned} \tag{3.9}$$

□

O Corolário 3.3 fornece um limite superior melhor para árvores do que o limite dado pelo Corolário 3.1, já que ambos os resultados só fazem sentido se $\Delta(G) \geq k$. Como k é uma constante, o segundo corolário oferece um limite linear no número de vértices para o comprimento do transiente, enquanto o corolário anterior para o caso geral demonstra um limite que pode ser quadrático.

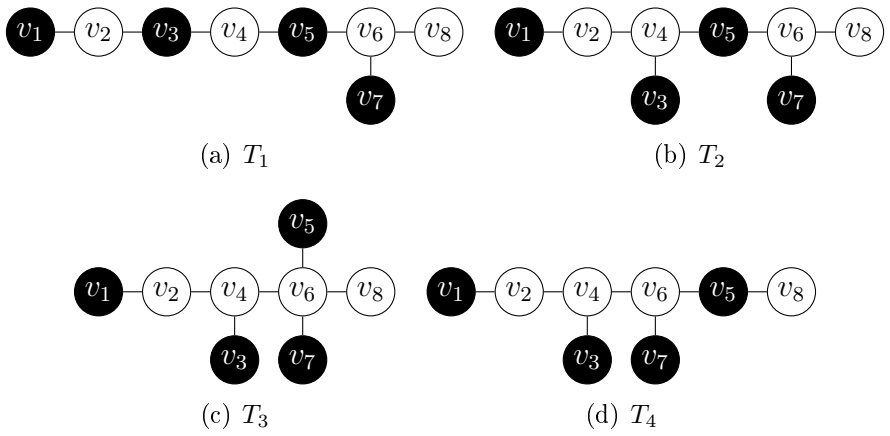
Apesar de um limite melhor, acreditamos que o limite ainda não é justo. Para o caso dos processos 2-reversíveis, apresentamos uma conjectura para o limite superior justo bem como as configurações que alcançam tal limite.

Conjectura 3.1. *Para processos 2-reversíveis onde o grafo G é uma árvore e possui $n \geq 5$ vértices, para qualquer configuração inicial x, $\tau(x) \leq n - 3$. Além disso, sempre existe uma árvore e uma configuração inicial x tais que $\tau(x) = n - 3$.*

Observe que a Conjectura 3.1 fornece um limite justo para processos 2-reversíveis, consideravelmente melhor do que poderíamos obter utilizando o Corolário 3.3, já que pelo corolário tínhamos $\tau(x) \leq 3n - 1$.

A conjectura tem como base testes computacionais realizados para todas as árvores com no máximo 20 vértices. Em nenhum caso foi obtido transiente maior que $n - 3$. A partir dos testes realizados, observamos um padrão estrutural para as árvores que possuíam configurações com transiente de comprimento $n - 3$ e desenvolvemos um algoritmo que constrói tais árvores com suas respectivas configurações iniciais.

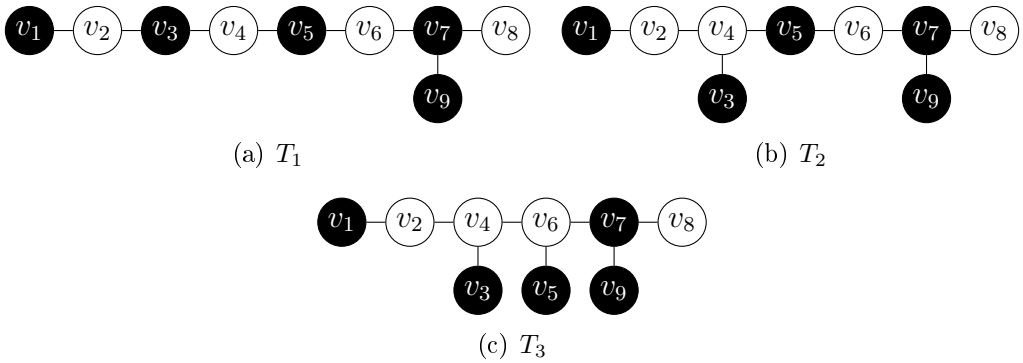
A Figura 3.5 contém todas as árvores geradas pelo Algoritmo 1 quando n é um número par, no caso o número 8. A Figura 3.6 contém todas as árvores geradas quando n é um número ímpar, no caso o número 9. Para os testes computacionais realizados, o algoritmo construiu todas as árvores e configurações iniciais que fornencem transiente de comprimento $n - 3$. A partir desse resultado apresentamos outra conjectura:



(a) T_1 (b) T_2

(c) T_3 (d) T_4

Figura 3.5: Árvores geradas pelo Algoritmo 1 para $n = 8$.



(a) T_1 (b) T_2

(c) T_3

Figura 3.6: Árvores geradas pelo Algoritmo 1 para $n = 9$.

Conjectura 3.2. *O Algoritmo 1 gera todas as árvores de n vértices e configurações x tais que $\tau(x) = n - 3$ em processos 2-reversíveis.*

Algoritmo 1:

Entrada: Número de vértices n das árvores a serem geradas
Saída: Árvores T_j com suas respectivas configurações iniciais X_j tal que $\tau(X_j) = n - 3$, em um processo 2-reversível.

início
  $V \longleftarrow \{v_1, v_2, \ldots, v_n\}$;
  $E \longleftarrow \emptyset$;
  $j \longleftarrow 1$;
  para $i \leftarrow 1$ **até** $n - 2$ **faça**
    $E \longleftarrow E \cup (v_i, v_{i+1})$;
    se $i = n - 2$ **então**
      $E \longleftarrow E \cup (v_i, v_{i+2})$;
  para $i \leftarrow 1$ **até** n **faça**
    se i *é ímpar* **então**
      $x[i] \longleftarrow 1$;
    senão
      $x[i] \longleftarrow 0$;
  $T_j \longleftarrow G(V, E)$;
  $X_j \longleftarrow x$;
  $j \longleftarrow j + 1$;
  para $i \leftarrow 3$ **até** $n - 3$ **faça**
    se i *é ímpar* **então**
      $E \longleftarrow E \setminus (v_i, v_{i-1})$;
      $E \longleftarrow E \cup (v_{i-1}, v_{i+1})$;
      $T_j \longleftarrow G(V, E)$;
      $X_j \longleftarrow x$;
      $j \longleftarrow j + 1$;
    se $i = n - 3$ *e* n *é par* **então**
      $E \longleftarrow E \setminus (v_{i+1}, v_{i+3})$;
      $E \longleftarrow E \cup (v_i, v_{i+3})$;
      $T_j \longleftarrow G(V, E)$;
      $X_j \longleftarrow x$;
      $j \longleftarrow j + 1$;
fim

Caso a Conjectura 3.2 esteja correta então existem exatamente $\frac{n}{2}$ árvores com configurações iniciais que resultam em transientes de comprimento $n-3$ caso n seja par, e $\frac{n-1}{2}-1$ dessas árvores caso n seja ímpar. Para chegarmos a esse resultado, observe que na linha 14 do algoritmo temos 1 árvore. Dentro do laço da linha 17, somente geramos árvores em metade dos $n-5$ passos, sendo que se n for par, geramos 1 árvore a mais. Então para o caso n par, a quantidade de árvores geradas é igual a $1+\frac{n-5}{2}+1$. Como $n-5$ é ímpar, temos que a quantidade de árvores geradas é igual a $1+\frac{n}{2}-2+1=\frac{n}{2}$. Para o caso n ímpar a fórmula é $1+\frac{n-5}{2}$, mas nesse caso $n-5$ é par, e, portanto, a quantidade de árvores é igual a $1+\frac{n-1}{2}-2=\frac{n-1}{2}-1$.

Capítulo 4

O Problema da Existência de Configuração Predecessora

O estudo de muitos problemas computacionais envolvendo processos dinâmicos em grafos já foram estudados previamente em autômatos celulares. Um desses problemas é o chamado Existência de Configuração Predecessora, que, por simplicidade, nesse capítulo, chamaremos de Ecp.

Dada uma configuração de estados Y, o Ecp consiste em determinar a existência de uma outra configuração Y' tal que a dinâmica do processo em G com configuração Y' associada produza a configuração Y na próxima unidade de tempo. Note que o termo *configuração predecessora* nesse contexto significa uma configuração imediatamente anterior. Esse termo será usado ao longo de todo o capítulo com esse significado.

O problema é bem amplo e não se restringe apenas a processos síncronos e grafos não-direcionados, como o os processos k-reversíveis. Podemos considerar processos sequenciais, grafos direcionados, processos majoritários e qualquer outra dinâmica que podíamos definir sobre autômatos celulares.

As configurações que não possuem predecessoras são conhecidas como *configurações jardim-do-éden* pelo fato de serem as configurações que dão origem a todas as outras configurações possíveis. Esse problema foi demonstrado ser NP-Completo para autômatos celulares finitos por Sutner [11] e Green [17]. As demonstrações realizadas mostram a existência de pelo menos uma função de atualização para o autômato tal que o problema seja NP-Completo, e, portanto esses resultados não se aplicam diretamente para provar a NP-completude de uma dinâmica específica.

Moore [18] e Myhill [19] demonstraram o *Teorema do Jardim do Éden* que prova uma condição suficiente e necessária para que um autômato celular possua configurações jardim-do-éden.

Nesse capítulo, vemos que para um processo k-reversível, o Ecp também é um problema NP-completo, incluindo o caso onde o grafo associado pertence à classe

de grafos bipartidos, para todo $k \geq 2$. Mostramos também que para a classe das árvores o problema está em P, e conseguimos não só determinar a existência de uma configuração predecessora, como conseguimos construir alguma das possíveis configurações e contar quantas existem. Finalizamos o capítulo mostrando que um caso restrito para o qual o problema também está em P é o caso onde o processo é 2-reversível e $\Delta(G) \leq 3$.

Nesse trabalho, o problema de determinação da existência de uma configuração predecessora em processos k-reversíveis será denominado ECP(k). A configuração que servirá como entrada para o problema será denominada sempre por Y. A Seção 4.1 trata da NP-completude do ECP(k), mesmo quando o grafo em questão é bipartido. Em seguida, na Seção 4.2 tratamos o problema para a classe das árvores, onde apresentamos algoritmos para resolver o problema em tempo polinomial. Concluímos o capítulo com a Seção 4.3, onde mostramos um algoritmo polinomial para o ECP(2) em grafos onde o grau de um vértice é no máximo três.

4.1 NP-Completude do Problema ECP(k)

Problema de Decisão ECP(k)

INSTÂNCIA: Um grafo G simples, finito e não-direcionado com n vértices $\{v_1, v_2, \ldots, v_n\}$. Associado ao grafo existe uma configuração de estados Y, onde $Y(v_i)$ representa o estado do vértice v_i e $Y(v_i) \in \{-1, +1\}$.

PERGUNTA: Existe uma configuração Y' associada a G tal que a dinâmica de um processo k-reversível gera a configuração Y após exatamente uma unidade de tempo?

Nessa seção, mostramos a NP-completude do ECP(k), para todo $k \geq 2$, através de uma redução polinomial realizada a partir do problema de satisfatibilidade 3SAT EXATAMENTE-2. O problema 3SAT EXATAMENTE-2 é uma variante do problema 3SAT, onde dado um conjunto de cláusulas onde cada cláusula é composta por exatamente três literais, a questão consiste em determinar se existe uma atribuição de valores para as variáveis de tal forma que cada cláusula possua exatamente dois literais verdadeiros, e, consequentemente, exatamente um literal falso.

Lema 4.1. *O problema* 3SAT EXATAMENTE-2 *é NP-Completo.*

Demonstração. O problema está trivialmente em NP, já que podemos verificar se uma valoração para as variáveis satisfaz todas as cláusulas à medida que as lemos. A prova de NP-completude se dá pela redução polinomial a partir do problema 3SAT EXATAMENTE-1, onde deve-se determinar se é possível obter uma valoração para as

variáveis tal que cada cláusula seja satisfeita por exatamente um literal verdadeiro. Esse problema é NP-Completo, como pode ser visto em [20].

A redução é bastante simples e consiste apenas na inversão de todos os literais em todas as cláusulas de uma instância S do 3Sat Exatamente-1 resultando em uma instância S' do problema 3Sat Exatamente-2. Em (4.1) temos o exemplo de uma transformação de uma instância S para uma instância S'. Note que as variáveis são exatamente as mesmas e só os literais foram alterados.

$$\begin{aligned} S &= (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3) \\ S' &= (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \end{aligned} \tag{4.1}$$

Caso S seja satisfatível, qualquer solução de S claramente satisfaz S', já que pela construção que fizemos, S' possuirá, nesse caso, exatamente dois literais positivos. Da mesma forma, caso S' seja satisfatível, então qualquer solução de S' satisfaz S, já que pela construção, S possuirá nesse caso exatamente um literal positivo.

□

Teorema 4.1. *O* Ecp(k) *é NP-Completo, para todo* $k \geq 2$.

Demonstração. O problema está claramente em NP, visto que, dada uma configuração Y', a verificação se Y' é de fato uma configuração predecessora de Y é feita apenas realizando um passo da dinâmica do processo k-reversível em G com configuração Y', e comparando a configuração resultante com a configuração Y. A realização da dinâmica possui complexidade $\mathrm{O}(n+m)$, e a comparação da configuração resultante com a configuração Y possui complexidade $\mathrm{O}(n)$. Pelo Lema 4.1, sabemos que 3Sat Exatamente-2 é NP-Completo. Para provar que o Ecp(k) é NP-Completo, mostramos que 3Sat Exatamente-2 se reduz polinomialmente a Ecp(k).

Seja S uma instância qualquer do 3Sat Exatamente-2 com M cláusulas $c_1, c_2, \ldots, c_M$ e N variáveis $x_1, x_2, \ldots, x_N$. Iremos construir uma instância a partir de S associada ao Ecp(k), ou seja, construiremos um grafo G e uma configuração Y que será a configuração alvo de estados para os vértices em $V(G)$.

O conjunto de vértices $V(G)$ será formado por:

- $\{x_i, \neg x_i, \mid$ para toda variável x_i de $S\}$
- $\{z_i, z'_i, \mid$ para toda variável x_i de $S\}$
- $\{u_{i,1}, \ldots, u_{i,2k-3} \mid$ para toda variável x_i de $S\}$

- $\{p_{i,1}, \ldots, p_{i,2k-3} \mid$ para toda variável x_i de $S\}$
- $\{w_{i,1}, \ldots, w_{i,k-2} \mid$ para toda variável x_i de $S\}$
- $\{w'_{i,1}, \ldots, w'_{i,k-2} \mid$ para toda variável x_i de $S\}$
- $\{c_i, c'_i \mid$ para toda cláusula c_i de $S\}$
- $\{b_{i,1}, \ldots, b_{i,k-2} \mid$ para toda cláusula c_i de $S\}$
- $\{b'_{i,1}, \ldots, b'_{i,k-1} \mid$ para toda cláusula c_i de $S\}$

Os vértices x_i e $\neg x_i$ chamaremos de vértices literais, enquanto os vértices c_i e c'_i chamaremos de vértices cláusulas.

O conjunto de arestas $E(G)$ conterá as arestas:

- $\{(x_i, z_i), (x_i, z'_i), (\neg x_i, z_i), (\neg x_i, z'_i) \mid$ para toda variável x_i de $S\}$
- $\{(x_i, u_{i,1}), \ldots, (x_i, u_{i,2k-3}) \mid$ para toda variável x_i de $S\}$
- $\{(\neg x_i, p_{i,1}), \ldots, (\neg x_i, p_{i,2k-3}) \mid$ para toda variável x_i de $S\}$
- $\{(z_i, w_{i,1}), \ldots, (z_i, w_{i,k-2}) \mid$ para toda variável x_i de $S\}$
- $\{(z'_i, w'_{i,1}), \ldots, (z'_i, w'_{i,k-2}) \mid$ para toda variável x_i de $S\}$
- $\{(c'_j, b'_{j,1}), \ldots, (c'_j, b'_{j,k-1}) \mid$ para toda cláusula c_j de $S\}$
- $\{(c_j, b_{j,1}), \ldots, (c_j, b_{j,k-2}) \mid$ para toda cláusula c_j de $S\}$
- $\{(c_j, x_i), (c'_j, x_i) \mid$ literal x_i ocorre na cláusula $c_j\}$
- $\{(c_j, \neg x_i), (c'_j, \neg x_i) \mid$ literal $\neg x_i$ ocorre na cláusula $c_j\}$

Para finalizar a construção, definimos a configuração alvo de estados Y da seguinte forma:

- $Y(x_i) = +1$, $Y(\neg x_i) = +1$, para $1 \leq i \leq n$.
- $Y(z_i) = +1$, $Y(z'_i) = -1$, para $1 \leq i \leq n$.
- $Y(u_{i,j}) = +1$, $Y(p_{i,j}) = +1$, para $1 \leq i \leq n$ e $1 \leq j \leq k-1$.
- $Y(u_{i,j}) = -1$, $Y(p_{i,j}) = -1$, para $1 \leq i \leq n$ e $k \leq j \leq 2k-3$.
- $Y(w_{i,j}) = -1$, $Y(w'_{i,j}) = +1$, para $1 \leq i \leq n$ e $1 \leq j \leq k-2$.
- $Y(c_i) = +1$, $Y(c'_i) = -1$, para $1 \leq i \leq m$.

- $Y(b_{i,j}) = -1$, para $1 \leq i \leq m$ e $1 \leq j \leq k-2$.
- $Y(b'_{i,j}) = -1$, para $1 \leq i \leq m$ e $1 \leq j \leq k-1$.

A Figura 4.1 ilustra a construção do grafo associado a uma instância S composta por apenas uma cláusula c_1 e três variáveis x_1, x_2 e x_3. Nesse caso, o valor de k considerado é 3. Assuma c_1 definida da seguinte forma:

$$c_1 = x_1 \vee \neg x_2 \vee \neg x_3$$

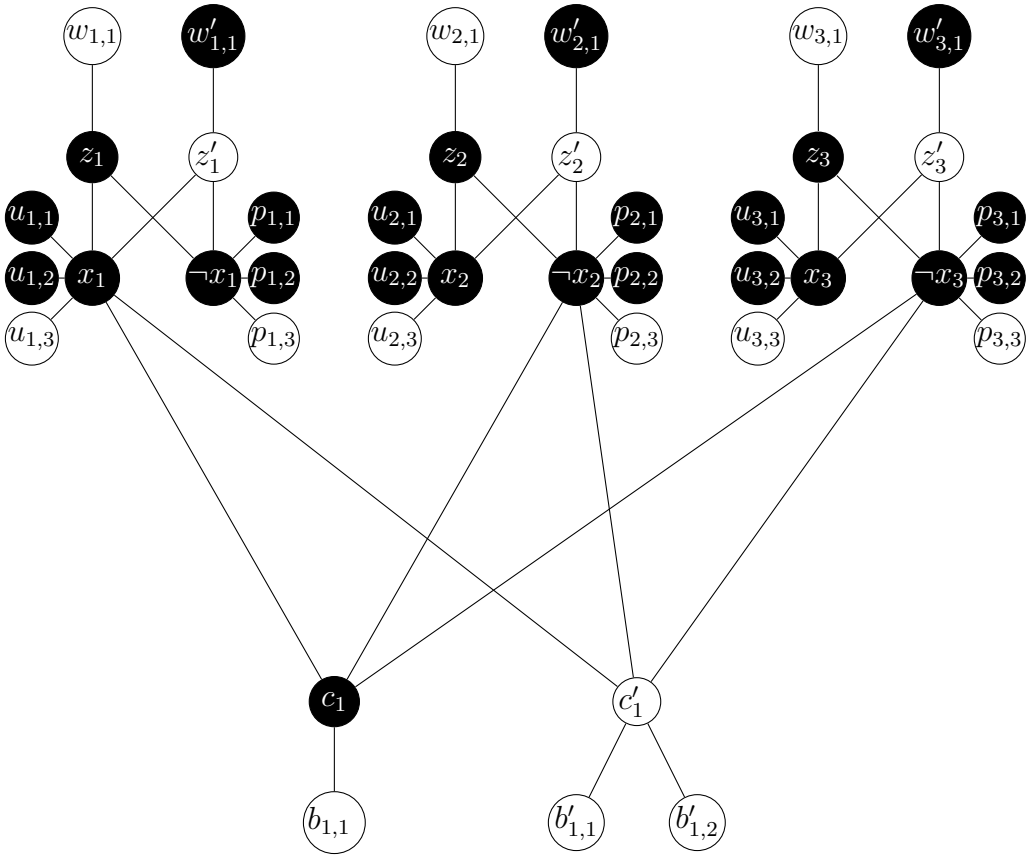


Figura 4.1: Exemplo de grafo construído para prova da NP-Completude do Ecp(3).

Observamos que a redução é feita em tempo polinomial, visto que para cada variável x_i de S são criados exatamente $6k-6$ vértices. Por sua vez, cada cláusula é responsável pela criação de exatamente $2k-1$ vértices. Dessa forma, temos que o total de vértices criados a partir da instância S é igual a $N(6k-6) + M(2k-1)$. A quantidade total de arestas criadas que são associadas a uma variável x_i de S é exatamente $6k-6$, enquanto que a quantidade total de arestas criadas associadas as cláusulas de S é exatamente $2k+3$. Logo o total de arestas do grafo construído é $N(6k-6) + M(2k+3)$. Como o valor k é constante, a construção do grafo é realizada em tempo linear no tamanho da instância S.

Após a redução polinomial ter sido realizada, precisamos demonstrar que S é satisfatível se e somente se a configuração Y possui pelo menos uma configuração predecessora. Iniciaremos a demonstração provando que se S é satisfatível então Y possui uma configuração predecessora.

A partir de uma solução para S, essa configuração predecessora de Y, que chamaremos de Y', pode ser obtida da seguinte forma:

- $Y'(x_i) = +1$, se o literal x_i for verdadeiro.
- $Y'(x_i) = -1$, se o literal x_i for falso.
- $Y'(\neg x_i) = -Y'(x_i)$
- $Y'(z_i) = +1$, $Y'(z'_i) = -1$ para $1 \leq i \leq n$.
- $Y'(u_{i,j}) = Y'(p_{i,j}) = +1$ para $1 \leq i \leq n$ e $1 \leq j \leq k-1$.
- $Y'(u_{i,j}) = Y'(p_{i,j}) = -1$ para $1 \leq i \leq n$ e $k \leq j \leq 2k-3$.
- $Y'(w_{i,j}) = -1$, $Y'(w'_{i,j}) = +1$ para $1 \leq i \leq n$ e $1 \leq j \leq k-2$.
- $Y'(c_i) = +1$, $Y'(c'_i) = +1$ para $1 \leq i \leq m$.
- $Y'(b_{i,j}) = -1$ para $1 \leq i \leq m$ e $1 \leq j \leq k-2$.
- $Y'(b'_{i,j}) = -1$ para $1 \leq i \leq m$ e $1 \leq j \leq k-1$.

A Figura 4.2 ilustra a configuração Y' predecessora da configuração Y para o grafo ilustrado na Figura 4.1. Como exemplo, os literais x_1, $\neg x_2$ e x_3 são considerados verdadeiros.

Pela construção realizada de G e observando a configuração Y' descrita acima, é possível ver que cada vértice x_i e $\neg x_i$ possui pelo menos k vértices vizinhos com estado $+1$ em Y' e exatamente um vértice vizinho com estado -1 em Y', sendo esse único vizinho o vértice z'_i. Essa condição é suficiente para garantir que todo vértice x_i e $\neg x_i$, pela dinâmica do processo, no próximo passo de tempo, assuma estado $+1$, independente de sua atribuição em Y'. Note que, na configuração Y, esse é o estado que esses vértices possuem.

Todos os vértices $u_{i,j}$, $p_{i,j}$, $w_{i,j}$, $w'_{i,j}$, $b_{i,j}$ e $b'_{i,j}$ possuem apenas um vizinho, e, dessa forma permanecem com o mesmo estado assumido em Y', que por sua vez é exatamente o mesmo estado em Y, como foi definido. Os vértices z_i e z'_i possuem exatamente $k-1$ vizinhos com atribuição oposta em Y', já que os vértices x_i e $\neg x_i$ possuem estados mutuamente opostos. Sendo assim, tanto z_i como z'_i mantêm os seus estados no próximo passo da dinâmica e, consequentemente, seus estados ficam de acordo com a configuração Y.

Nenhum vértice c_i pode possuir mais do que um vértice literal v vizinho tal que $Y'(v)$ seja -1, pois, dessa forma, a dinâmica resultará no próximo passo em uma configuração com c_i possuindo estado -1. Como assumimos que a instância S é satisfatível, então há exatamente dois literais verdadeiros para cada cláusula c_i de S, e pela construção realizada para Y', isso garante que c_i possua exatamente um vizinho literal com estado -1 em Y'. Todo vértice c'_i precisa de pelo menos um vértice literal vizinho tal que o estado seja -1 em Y', para que no passo seguinte c'_i assuma estado -1. Como a instância S é satisfatível, a construção de Y' nos garante que c'_i possuirá exatamente um vértice literal vizinho com estado -1.

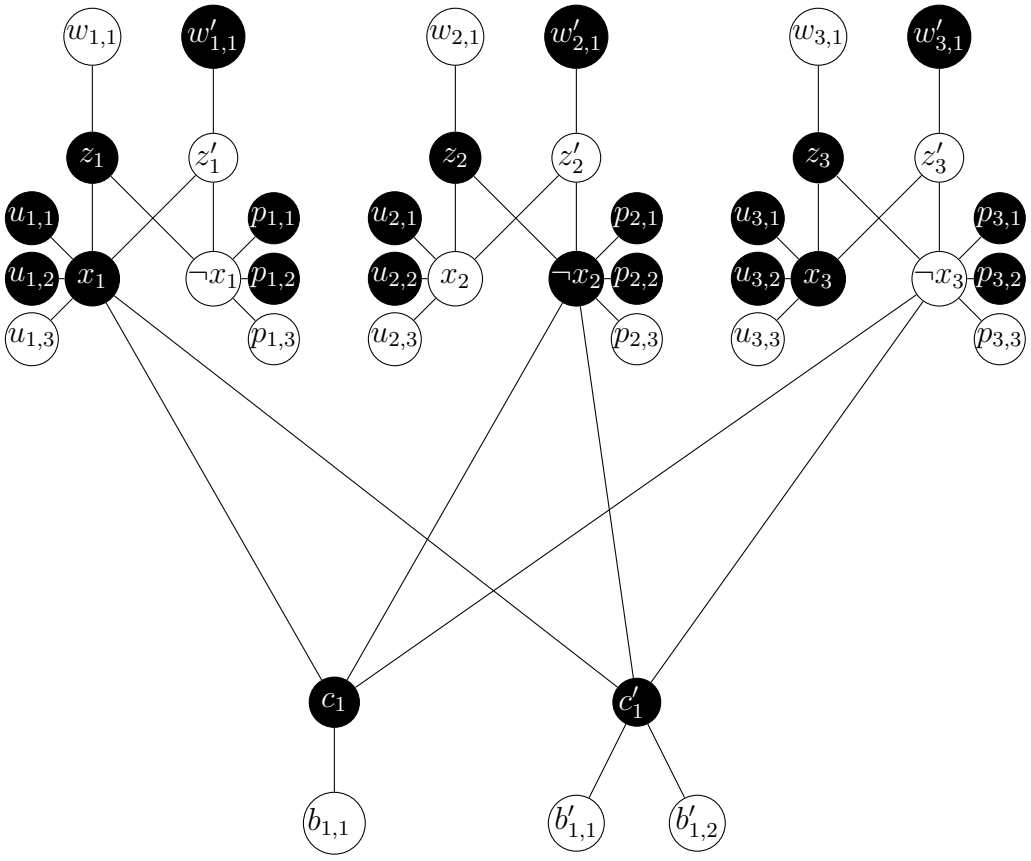


Figura 4.2: Exemplo de configuração predecesssora Y' da configuração Y.

Dessa forma, após um passo, a dinâmica do processo garante que a configuração Y' resulte na configuração Y, sempre que existir uma solução para o problema de satisfatibilidade associado.

Reciprocamente, podemos demonstrar que se existe uma configuração predecessora para o grafo construído com a configuração de estados Y, então S é satisfatível.

Em qualquer configuração predecessora, os estados dos vértices $u_{i,j}$, $p_{i,j}$, $w_{i,j}$, $w'_{i,j}$, $b_{i,j}$ e $b'_{i,j}$ deverão ser os mesmos estados que esses vértices possuem na configuração alvo Y devido ao fato de possuirem apenas um vizinho, e portanto não mudarem de estado em nenhum momento.

Os estados dos vértices z_i e z'_i em uma configuração predecessora também serão os mesmos da configuração Y. Caso o estado de z_i na configuração predecessora seja -1, então necessariamente os estados de x_i e $\neg x_i$ nessa configuração são iguais a $+1$, e consequentemente a dinâmica do processo k-reversível resultaria no estado $+1$ para z'_i na configuração imediatemante posterior, diferente do que ocorre na configuração Y, onde o estado de z'_i é -1. O raciocínio é análogo para o caso onde o estado de z'_i é $+1$. Os estados dos vértices z_i e z'_i em uma configuração predecessora de Y forçam que o estado de x_i seja diferente do estado de $\neg x_i$ em uma possível configuração predecessora, visto que se ambos possuírem o mesmo estado, então a dinâmica do processo resultaria em z_i e z'_i com o mesmo estado no próximo passo.

Para cada vértice c_i, se o estado desse vértice em uma configuração predecessora de Y for -1, então como em Y, c_i possui estado $+1$, sabemos que um número positivo de vértices literais vizinhos deve possuir estado $+1$. Mas, dessa forma, todos esses vizinhos passariam a possuir estado -1 na próxima configuração. Sendo assim, em uma configuração predecessora de Y o estado de cada c_i é necessariamente $+1$. Além disso, essa condição força que pelo menos dois de seus vizinhos literais também possuam estado $+1$, pois caso contrário c_i receberia estado -1 no próximo passo da dinâmica, o que também contradiz a configuração Y.

Cada vértice c'_i compartilha os mesmos vizinhos literais com o vértice c_i. Como alguns desses vizinhos em uma configuração predecessora possuem estado $+1$, então c'_i deve possuir estado $+1$, pois caso contrário, esses vértices vizinhos passariam a ter estado -1 na próxima configuração, contradizendo a configuração Y. Junto com a restrição imposta por c_i, os três vértices literais associados a cláusula c_i devem possuir uma configuração onde dois veŕtices possuem estado $+1$ e outro vértice possui estado -1 na configuração predecessora.

Logo, a partir de qualquer configuração predecessora de Y, podemos associar para cada literal das cláusulas o valor verdadeiro caso o vértice literal no grafo possua estado $+1$, e o valor falso caso contrário. Como foi visto, a construção nos garante que dois literais opostos não possuirão valores iguais e que cada cláusula possuirá exatamente dois literais positivos e um literal negativo. Conclui-se então que se Y possui uma configuração predecessora então S é satisfatível.

Concluímos que Y possui configuração predecessora se e somente se S é satisfatível.

□

Diretamente, a partir da construção realizada para a prova do Teorema 4.1, obtemos o resultado de NP-Completude do ECP(k) para a classe de grafos bipartidos.

Corolário 4.1. *Quando G é um grafo bipartido, o* ECP(k) *ainda é NP-Completo, para todo $k \geq 2$.*

Demonstração. O grafo construído no Teorema 4.1 é bipartido com partes $A = \{x_i, \neg x_i, b_{i,j}, b'_{i,j}, w_{i,j}, w'_{i,j}\}$ e $B = \{c_i, c'_i, u_{i,j}, p_{i,j}, z_i, z'_i\}$. □

4.2 ECP(k) em árvores

Em teoria de grafos, é bastante comum problemas muito difíceis possuírem soluções com complexidades de tempo polinomiais quando restringimos o problema para algumas classe de grafos. Um exemplo clássico é o problema de determinar a menor *cobertura de arestas por vértices*, que é um problema NP-Difícil [20], porém, caso o grafo seja bipartido, existe algoritmo de tempo polinomial que o resolve [21]. Esse caminho, no entanto, não pode ser seguido para o problema ECP(k), como demonstrado pelo Corolário 4.1.

Uma outra classe de grafos bastante estudada para elaborar algoritmos mais eficientes para problemas difíceis é a classe das árvores. Como vimos no Capítulo 2, o problema de determinar o MENOR CONJUNTO CONVERSOR em processos k-irreversíveis é NP-Difícil; porém, em árvores esse problema está em P [14]. E nesse caso, onde G é uma árvore, o ECP(k) pode ser resolvido em complexidade de tempo polinomial. Além disso, não só é possível determinar se uma configuração Y possui uma configuração predecessora, como também é possível contar quantas configurações predecessoras existem em tempo polinomial.

Nesse capítulo, consideramos T uma árvore enraizada em um vértice arbitrário *raiz*. Para cada vértice v de T, o vértice pai de v será denominado pai_v, o conjunto de vértices filhos de v será denominado $filhos_v$. Qualquer sub-árvore de T será denominada T_u, onde u é o vértice raiz dessa sub-árvore. Dessa forma, podemos denominar a árvore T também por T_{raiz}.

A Subseção 4.2.1 apresenta o algoritmo que resolve, em tempo polinomial, o ECP(k) em árvores para todo k. Enquanto a Subseção 4.2.2 apresenta o algoritmo de contagem de todas as configurações predecessoras. Note que o segundo algoritmo claramente pode ser utilizado para resolver o ECP(k), porém o primeiro algoritmo é consideravelmente mais eficiente na sua complexidade de tempo, e por isso é apresentado nesse trabalho.

4.2.1 Um Algoritmo Polinomial para Resolver o ECP(k) em Árvores

Começamos essa seção definindo uma função simples para auxílio na explicação do algoritmo. Definimos a função $vstate(alvo, atual, p, k)$ como sendo a função que determina a possibildade de um vértice qualquer no estado *atual* possuir no próximo instante de tempo o estado *alvo*, dado que possui exatamente p vizinhos

com estado diferente de *atual*. O valor da função é *verdadeiro* caso a dinâmica desejada seja possível em um processo k-reversível. Caso contrário, o valor da função é *falso*. A função pode ser calculada facilmente da seguinte forma:

$$vstate(alvo, atual, p, k) = \begin{cases} falso & \text{se } atual = alvo \text{ e } p \geq k \text{ ou} \\ & \text{se } atual \neq alvo \text{ e } p < k. \\ verdadeiro & c.c. \end{cases}$$

A complexidade de tempo do cálculo dessa função é O(1), visto que são necessárias apenas duas comparações para o seu cálculo. Podemos ver que, de maneira geral, a utilização dessa função se resume a verificar se uma determinada transição de estados de um vértice é uma transição válida de acordo com as regras do processo k-reversível.

Seja v um vértice de T. Suponha que pai_v, quando existir, possua estado c em uma configuração predecessora de Y. Definimos a função $fstate(v, c)$ como sendo o estado que v deve possuir nessa configuração predecessora de tal forma que na subárvore T_v, após uma unidade de tempo, todos os vértices estejam com os estados que possuem na configuração Y. Caso ambos os estados sejam possíveis para o vértice v, o valor da função é o estado que pai_v, quando existir, possui na configuração Y, de forma que no caso onde v é o vértice *raiz*, o valor da função será, por padrão, $+1$. Caso nenhum estado seja possível, a função possui valor ∞.

Como o vértice *raiz* não possui pai, para não haver confusão, assumimos que a função *fstate* nesse caso sempre é chamada com os parâmetros *raiz* e 0. Logo, $fstate(raiz, 0)$ é simplesmente o estado que *raiz* deve possuir em uma configuração predecessora de Y. Portanto, se $fstate(raiz, 0)$ for diferente de ∞, então a configuração Y possui configuração predecessora, caso contrário, tal configuração não existe.

Quando os dois estados são possíveis para $fstate(v, c)$, definimos que a função deva possuir o estado $Y(pai_v)$. E essa escolha nunca é prejudicial, visto que não importa o estado que está sendo atribuído a pai_v na configuração predecessora: quanto maior o número de vizinhos com estado $Y(pai_v)$, "mais fácil" é para a dinâmica do processo k-reversível fazer com que pai_v possua estado $Y(pai_v)$ no próximo passo. Da forma que calculamos $fstate(v, c)$, estamos maximizando o número de vizinhos que favorecem o alcance do estado $Y(pai_v)$, e, claramente, se esse número máximo de vizinhos com estado $Y(pai_v)$ não é suficiente, então um número menor também não será.

O problema agora consiste em calcular $fstate(raiz, 0)$ de maneira eficiente e correta. Para calcular $fstate(v, c)$, vamos assumir que para todo filho f de v, $fstate(f, +1)$ e $fstate(f, -1)$ estejam corretamente calculados. Ou seja, conhece-

mos os estados de todos os filhos de v tanto para o caso onde o estado de v na configuração predecessora é $+1$, como também quando o estado de v é -1. Avaliando todos os valores $\mathit{fstate}(f, st)$, chamaremos de w o número de filhos de v tal que $\mathit{fstate}(f, st)$ seja $-st$. Somaremos 1 em w caso c seja $-st$. Ou seja, w é o número de vizinhos de v com estado oposto ao estado de v em uma configuração predecessora. Sendo assim, caso $\mathit{vstate}(Y(v), st, w, k)$ seja verdadeiro podemos afirmar que o estado st para v permite que a sub-árvore T_v esteja com uma configuração de acordo com a configuração Y após uma unidade de tempo quando pai_v possui estado c na configuração predecessora. Vamos denominar esse estado como sendo um estado válido. Note que o cálculo quando v for uma folha considera apenas o estado de pai_v, que é o valor c. Dessa forma, esse caso não depende de valor calculado pela função previamente e é calculado corretamente de maneira trivial.

Um ponto importante é que caso pelo menos um filho f de v possua $\mathit{fstate}(f, +1) = \infty$, então o estado $+1$ já não pode ser válido para v, pois se a sub-árvore T_f não possui uma configuração de acordo com a configuração Y após uma unidade de tempo, então obviamente a sub-árvore T_v não possuirá. A análise para a atribuição do estado -1 em v é análoga. Dado que já sabemos quais são os estados válidos para v quando pai_v possui estado c, a função *fstate* é calculada da seguinte forma:

$$\mathit{fstate}(v, c) = \begin{cases} Y(pai_v) & \text{se ambos os estados são válidos.} \\ +1 & \text{se somente o estado } +1 \text{ é válido.} \\ -1 & \text{se somente o estado } -1 \text{ é válido.} \\ \infty & c.c. \end{cases}$$

Portanto, dado que sabemos quais são os estados válidos, $\mathit{fstate}(v, c)$ é determinado em O(1). Para determinar quais são os estados válidos, dado que $\mathit{fstate}(f, +1)$ e $\mathit{fstate}(f, -1)$ estão calculados para todo filho f de v, precisamos apenas contabilizar o número de vizinhos com estado diferente do estado em que está sendo verificada a validade para v, para então chamar a função *vstate*. O custo para fazer isso é $2d(v)$, pois precisamos fazer isso para os dois estados; logo, a complexidade de tempo para o cálculo é $O(d(v))$.

É possível calcular *fstate* para toda a árvore T a partir de um algoritmo recursivo, semelhante a uma busca em profundidade, onde ao visitarmos um vértice v com pai_v possuindo estado c, calculamos recursivamente $\mathit{fstate}(f, +1)$ e $\mathit{fstate}(f, -1)$ para todo filho f de v, e ao voltarmos da recursão, temos todos os valores necessários para o cálculo de $\mathit{fstate}(v, c)$ da maneira que descrevemos anteriormente. Podemos, então apenas calcular $\mathit{fstate}(raiz, 0)$ recursivamente, e caso o valor calculado seja diferente de ∞ concluímos que existe uma configuração predecessora para Y.

Algoritmo 2: $calcfstate(v, c)$

```
1  início
2      se fstate[v, c + 1] ≠ NIL então
3          retorna fstate[v, c + 1];
4      contador ⟵ 0 ;
5      rotuloTeste ⟵ Y(pai_v) ;
6      enquanto contador ≠ 2 faça
7          contador ⟵ contador + 1;
8          w ⟵ 0;
9          ret ⟵ verdadeiro;
10         se c = −rotuloTeste então
11             w ⟵ w + 1 ;
12         para cada f ∈ filhos_v faça
13             se calcfstate(f, rotuloTeste) = −rotuloTeste então
14                 w ⟵ w + 1 ;
15             se calcfstate(f, rotuloTeste) = ∞ então
16                 ret ⟵ falso;
17         se vstate(Y(v), rotuloTeste, w, k) = verdadeiro e ret ≠ falso então
18             fstate[v, c + 1] ⟵ rotuloTeste ;
19             retorna rotuloTeste;
20         rotuloTeste ⟵ −rotuloTeste ;
21     fstate[v, c + 1] ⟵ ∞ ;
22     retorna ∞;
23 fim
```

O Algoritmo 2 implementa exatamente essa abordagem recursiva. Mantemos uma tabela *fstate* que ao final do algoritmo conterá os valores da função. Essa tabela é inicializada com valor NIL para todos os vértices e estados, indicando que a função ainda não foi calculada para aqueles parâmetros.

Para cada vértice, o algoritmo primeiramente tenta atribuir o estado $Y(pai_v)$ para v, e caso seja um estado válido, atribuímos esse valor na tabela e o retornamos. Caso contrário, o estado oposto a $Y(pai_v)$ é testado. Caso nenhum estado seja válido, atribuímos ∞ na tabela. Note que, devido ao uso da tabela, todo acesso a uma posição da tabela incrementa o valor de c em 1, para não tentarmos acessar uma posição de valor negativo quando c for igual a -1. O uso da tabela é de grande importância para que o algoritmo possua complexidade de tempo polinomial. Se não fosse feita a verificação do valor calculado na linha 2 do algoritmo, o que teríamos seria um algoritmo de *backtracking* de complexidade de tempo $O(2^n)$.

Para ver como o algoritmo se comportaria em termos de complexidade de tempo sem o uso da tabela, basta analisarmos o caso onde T é um P_n e $k = 2$. Seja $H(n)$ o número de operações do algoritmo em um P_n. Nesse caso, então, se escolhermos a

raiz da árvore como sendo um dos dois vértice de grau 1, podemos calcular o número de operações do algoritmo recursivo no pior caso da seguinte forma:

$$H(n) = 2H(n-1) + a, \text{ onde } a \text{ é uma constante e } H(1) = a.$$

Essa análise considera que em cada vértice o algoritmo testará a validade dos dois estados. Note que essa situação ocorre sempre que não existe nenhuma configuração predecessora. Além disso, a análise aproveita a estrutura de um P_n poder ser definida facilmente a partir da estrutura de um P_{n-1} com a adição de apenas mais um vértice e uma aresta. Sendo assim, podemos calcular $H(i)$ como função de qualquer $H(j)$, para todo $j < i$. Para isso, substituímos iterativamente $H(i-1)$ em $H(i)$:

$$\begin{aligned} H(i) &= 2H(i-1) + a \\ &= 4H(i-2) + 3a \\ &= 8H(i-3) + 7a \\ &\dots \\ &= 2^j H(i-j) + (2^j - 1)a \end{aligned} \tag{4.2}$$

Usando (4.2) para escrever $H(n)$ em função de $H(1)$, obtemos:

$$H(n) = 2^{n-1}H(1) + (2^{n-1} - 1)a$$

Finalmente, substituindo $H(1)$:

$$\begin{aligned} H(n) &= 2^{n-1}a + (2^{n-1} - 1)a \\ &= 2^{n-1}a + 2^{n-1}a - a \\ &= 2^n a - a \end{aligned} \tag{4.3}$$

Portanto, a partir de (4.3), vemos que $\mathrm{O}(H(n)) = \mathrm{O}(2^n)$.

Para evitar esse consumo exponencial de tempo, utilizamos a tabela para o algoritmo visitar cada vértice no máximo quatro vezes. Essa técnica onde guardamos os resultados de um algoritmo recursivo lento visando utilizá-los como uma espécie de *memória cache* é conhecida como *memoization* [22].

Suponha que fizemos a seguinte chamada $calcfstate(v, c)$ pela primeira vez. Logo, pelo algoritmo, no pior caso, chamaremos $calcfstate(f, +1)$ e $calcfstate(f, -1)$ para

cada filho f de v, e enfim calculando $fstate[v, c+1]$. Para cada filho f de v, foram feitas duas visitas, e qualquer outra chamada para $calcfstate(v, c)$ não resultará em mais nenhuma visita aos filhos de f, pois o algoritmo retornará na linha 3. Para a chamada $calcfstate(v, -c)$, novamente chamaremos, no pior caso, $calcfstate(f, +1)$ e $calcfstate(f, -1)$ para cada filho f de v, o que incrementa para quatro o número de visitas para todo f. Após essa chamada, qualquer visita ao vértice v não acarretará mais nenhuma visita aos seus filhos, pois temos ambos os valores *fstate(v,c)* e *fstate(v,-c)* armazenados na tabela. Concluímos que, para qualquer vértice, os filhos desse vértice são visitados no máximo quatro vezes. Além disso, a raíz da árvore é visitada exatamente uma vez, pois o algoritmo realiza apenas uma chamada para tal vértice. Portanto, qualquer vértice da árvore recebe no máximo quatro visitas durante o algoritmo. Podemos ver também que cada aresta é percorrida também no máximo quatro vezes, duas vezes para cada estado de teste. Logo, a complexidade de tempo do Algoritmo 2 é $O(n+m)$. Como em uma árvore, $m = n-1$, a complexidade de tempo fica $O(n)$.

Para finalizar, o Algoritmo 3 implementa a ideia recursiva para recuperar a configuração predecessora de Y dado que a tabela *fstate* já foi calculada. O algoritmo apenas percorre a árvore novamente acessando os valores na tabela *fstate* de acordo com o vértice que está visitando e com o estado atribuído ao seu pai. Ao atribuirmos um estado para um vértice, só existe uma opção de chamada recursiva a se fazer. Sendo assim, precisamos apenas seguir pela recursão e atribuindo o estado calculado na tabela *fstate*, à medida que visitamos os vértices. O algoritmo apenas percorre a árvore atribuindo estados para os vértices, e, portanto, possui complexidade de tempo $O(n)$.

Algoritmo 3: $buildState(v, c)$

```
1 início
2 |  y[v] = fstate[v, c + 1] ;
3 |  para cada f ∈ filhos_v faça
4 |  |  buildState(f, y[v]);
5 fim
```

4.2.2 Um Algoritmo de Tempo Polinomial para Contar o Número de Configurações Predecessoras

O Algoritmo 2 em conjunto com o Algoritmo 3 fornece uma maneira eficiente de produzirmos uma configuração predecessora para uma configuração Y qualquer em um árvore, se tal configuração existir. No entanto, a quantidade de configurações predecessoras pode ser maior que apenas uma configuração, e o nosso algoritmo

sempre retorna a mesma configuração para uma determinada configuração Y. Note que não sabemos o número máximo de configurações predecessoras que podem existir. Então, um algoritmo que retorne todas essas configurações, a princípio, pode possuir complexidade de tempo exponencial. Suponha, por exemplo, uma árvore que seja um grafo construído da seguinte forma:

- Um vértice v_1 conectado a $x > 1$ vértices $v_2, v_3, \ldots, v_{x+1}$.
- Cada vértice v_i, com $2 \leq i \leq x + 1$, conectado a dois vértices d_{i-1} e e_{i-1}.

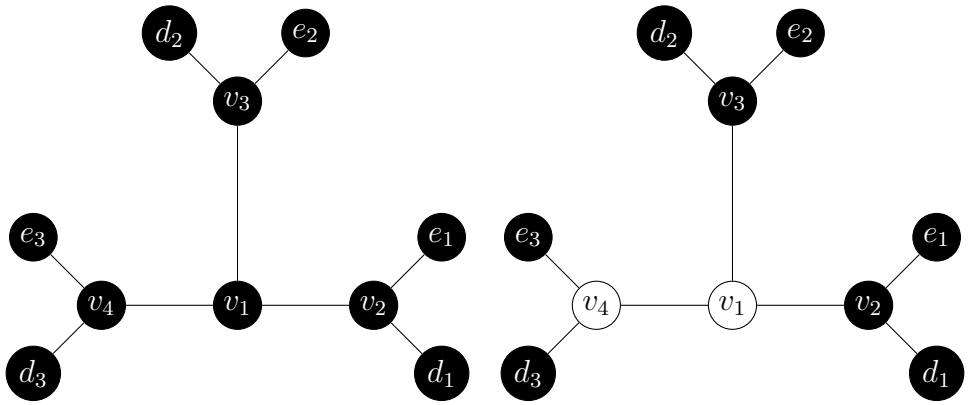
A árvore, portanto, possui $3x + 1$ vértices. Vamos assumir um processo 2-reversível e a configuração Y como sendo a configuração onde todos os vértices possuem estado $+1$. Nesse caso, qualquer configuração onde o vértice v_1 possua estado -1 e pelo menos dois dos vértices $v_2, v_3, \ldots, v_{x+1}$ possuam estado $+1$ é uma configuração predecessora de Y. Note que todos os vértices d_i e e_i devem possuir estado $+1$ em uma configuração predecessora de Y, já que possuem grau 1. Portanto, um limite inferior para o número de configurações predecessoras para Y em uma árvore com $n = 3x + 1$ é dado por:

$$\begin{aligned} \sum_{i=2}^{x} \binom{x}{i} &= \sum_{i=0}^{x} \binom{x}{i} - \sum_{i=0}^{1} \binom{x}{i} \\ &= 2^x - x - 1 \end{aligned} \tag{4.4}$$

A Figura 4.3 ilustra a construção dessa árvore com $x = 3$ e a configuração Y. Ilustramos, também, algumas das possíveis configurações predecessoras de Y para um processo 2-reversível. Em todas as configurações predecessoras ilustradas, o vértice v_1 possui estado -1, e, portanto, são configurações contabilizadas por (4.4).

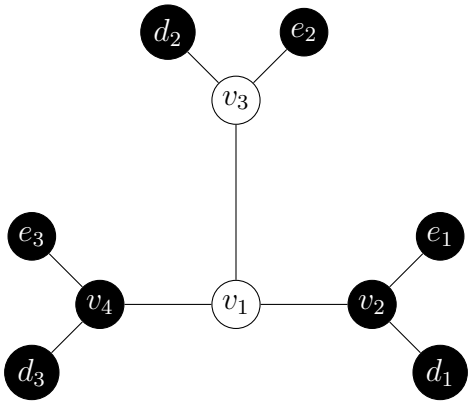
O limite calculado é um limite inferior apenas para termos noção do número exponencial de configurações possíveis. Consideramos apenas as configurações onde o vértice v_1 possui estado -1. Mas existem também configurações onde o estado é $+1$, o que só torna o limite ainda mais alto. Logo, não é possível construir um algoritmo que possua complexidade de tempo polinomial para recuperar todas as possíveis configurações predecessoras mesmo quando o grafo é uma árvore. No entanto, a tarefa de apenas contá-las possibilita a construção de um algoritmo de complexidade de tempo $O(n^2)$, para todo k.

A ideia do algoritmo se baseia no uso de uma função parecida com a função *fstate* utilizada no algoritmo apresentado anteriormente. Porém, nesse algoritmo será utilizada uma função mais robusta, que não apenas conterá o estado que o vértice deve possuir em uma configuração predecessora, mas conterá tanto o número



(a) *Configuração Y*

(b) *Configuração predecessora Y′*



(c) *Configuração predecessora Y″*

Figura 4.3: Exemplo de árvore e configuração Y com possível número exponencial de configurações predecessoras.

de configurações predecessoras da sub-árvore quando o vértice possui estado $+1$, como o número de configurações predecessoras da sub-árvore quando o vértice possui estado -1. Assim, definimos a função $cfstate(v, c)$ como sendo um par ordenado onde o primeiro elemento do par é o número de configurações predecessoras que a sub-árvore T_v possui quando pai_v possui estado c e v possui estado $+1$, e o segundo elemento do par é o número de configurações predecessoras que a sub-árvore T_v possui quando pai_v possui estado c e v possui estado -1.

Assim como a função *fstate*, quando v for a raiz da árvore, assumimos que a nova função possuirá os parâmetros v e 0. Logo, o número total de configurações predecessoras da sub-árvore T_v, quando pai_v possui estado c, é a soma dos dois elementos do par ordenado $cfstate(v, c)$, e no caso onde v é a raiz, a soma dos dois elementos do par ordenado $cfstate(v, 0)$.

A forma natural de calcular $cfstate(v, c)$ é bem simples. Sem perda de generalidade, suponha que estamos calculando o primeiro elemento do par $cfstate(v, c)$ e que para esse caso seja necessário que pelo menos u filhos de v possuam estado st em qualquer configuração predecessora. Por simplicidade, o primeiro elemento do par

$cfstate(v, c)$ será denominado $cfstate(v, c)_{+1}$, enquanto o segundo elemento do par será denominado $cfstate(v, c)_{-1}$. Sendo assim, uma forma de realizar esse cálculo é a seguinte:

$$cfstate(v, c)_{+1} = \sum_{X \in C_v^u} calc(X, st)$$

onde C_v^u é o conjunto que contém todos os subconjuntos de filhos do vértice v, tal que cada subconjunto possui pelo menos u elementos. E $calc(X, st)$ é definido da seguinte forma:

$$calc(X, st) = \begin{cases} 1 & \text{se } X = \emptyset \\ \prod_{f \in X} cfstate(f, +1)_{st} \cdot \prod_{f \in V(G) \setminus X} cfstate(f, +1)_{-st} & \text{c.c.} \end{cases}$$

Ou seja, apenas testamos todas as possibilidades de estados para os filhos de v tal que existam pelo menos u desses vértices com estado $+1$. Para cada uma dessas possibilidades calculamos o número total de configurações predecessoras, multiplicando a quantidade de configurações predecessoras para cada sub-árvore. O valor de $cfstate(v, c)_{+1}$ é o somatório do total obtido para cada uma das possibilidades.

Note que o cálculo de $cfstate(v, c)_{-1}$, também supondo que em uma configuração predecessora pelo menos u filhos de v possuam estado st, pode ser realizado de maneira análoga, apenas redefinindo $calc(X, st)$:

$$calc(X, st) = \begin{cases} 1 & \text{se } X = \emptyset \\ \prod_{f \in X} cfstate(f, -1)_{st} \cdot \prod_{f \in V(G) \setminus X} cfstate(f, -1)_{-st} & \text{c.c.} \end{cases}$$

Utilizando essas definições, podemos construir um algoritmo recursivo para calcular $cfstate(v, c)$. No entanto, o problema dessa abordagem é que é necessário iterar sobre totas as configurações possíveis para os filhos de v. O número total de tais configurações é $O(2^{d(v)-1})$, e, portanto, essa maneira de calcular o valor da função resulta em um algoritmo com complexidade de tempo exponencial. Podemos facilmente verificar que o total de configurações possíveis é $O(2^{d(v)-1})$. Por exemplo, suponha $u = 1$:

$$
\begin{aligned}
|C_v^1| &= \sum_{i=1}^{d(v)-1} \binom{d(v)-1}{i} \\
&= \sum_{i=0}^{d(v)-1} \binom{d(v)-1}{i} - \sum_{i=0}^{0} \binom{d(v)-1}{i} \\
&= 2^{d(v)-1} - 1
\end{aligned}
\tag{4.5}
$$

No entanto, é possível evitar o consumo exponencial de tempo utilizando a técnica conhecida como *programação dinâmica* [22]. Através dessa técnica, buscamos reduzir um problema em sub-problemas mais simples para então calcular a resposta para o problema original a partir das respostas para os sub-problemas. Queremos, nesse caso, calcular $cfstate(v, c)$ sem ter que iterar sobre todas as configurações possíveis de estados para os filhos de v.

Para isso, vamos assumir que os filhos de v estão ordenados: $filho_{v,0}$, ..., $filho_{v,d(v)-2}$, caso v seja um vértice interno da árvore. A ordenação, caso v seja a raiz da árvore, é: $filho_{v,0}$, ..., $filho_{v,d(v)-1}$. Para não ser necessário diferenciar entre o vértice raiz e os vértices internos, iremos denominar a quantidade de filhos de um vértice v por $d'(v)$.

Definamos a função $g_v^{rt}(i, j)$ como o número total de configurações predecessoras para a sub-árvore T_v quando v possui estado rt nessas configurações, tal que dentre os vértices $filho_{v,0}, filho_{v,1}, \ldots, filho_{v,i-1}$, exatamente j desses veŕtices possuem estado $+1$. Similarmente, a $h_v^{rt}(i, j)$ como o número total de configurações predecessoras para a sub-árvore T_v quando v possui estado rt nessas configurações, tal que dentre os vértices $filho_{v,0}, filho_{v,1}, \ldots, filho_{v,i-1}$, exatamente j desses veŕtices possuem estado -1.

Logo, para calcular $cfstate(v, c)_{rt}$ podemos utilizar essas funções:

- Se para a existência de configurações predecessoras na sub-árvore T_v for necessário que pelo menos u filhos de v possuam estado $+1$, fazemos:

$$
cfstate(v, c)_{rt} = \sum_{i=u}^{d'(v)} g_v^{rt}(d'(v), i).
$$

- Se para a existência de configurações predecessoras na sub-árvore T_v for necessário que pelo menos u filhos de v possuam estado -1, fazemos:

$$
cfstate(v, c)_{rt} = \sum_{i=u}^{d'(v)} h_v^{rt}(d'(v), i).
$$

O cálculo de $g_v^{rt}(i,j)$ pode ser definido recursivamente da seguinte forma:

$$g_v^{rt}(i,j) = \begin{cases} 0 & \text{se } i = 0 \text{ e } j > 0 \\ 1 & \text{se } i = 0 \text{ e } j = 0 \\ g_v^{rt}(i-1,j) \cdot a_i + g_v^{rt}(i-1,j-1) \cdot b_i & \text{se } i > 0 \text{ e } j > 0 \\ g_v^{rt}(i-1,j) \cdot a_i & \text{se } i > 0 \text{ e } j = 0 \end{cases}$$

onde estamos chamando $a_i = \mathit{cfstate}(\mathit{filho}_{v,i}, rt)_{-1}$ e $b_i = \mathit{cfstate}(\mathit{filho}_{v,i}, rt)_{+1}$, apenas para simplicidade na fórmula.

Similarmente, o cálculo de $h_v^{rt}(i,j)$ também pode ser definido recursivamente:

$$h_v^{rt}(i,j) = \begin{cases} 0 & \text{se } i = 0 \text{ e } j > 0 \\ 1 & \text{se } i = 0 \text{ e } j = 0 \\ h_v^{rt}(i-1,j) \cdot b_i + h_v^{rt}(i-1,j-1) \cdot a_i & \text{se } i > 0 \text{ e } j > 0 \\ h_v^{rt}(i-1,j) \cdot b_i & \text{se } i > 0 \text{ e } j = 0 \end{cases}$$

Como podemos ver, o cálculo de $g_v^{rt}(i,j)$ consiste em testar as duas possibilidades de estado para o vértice $\mathit{filho}_{v,i-1}$, se possível. Separamos em 4 casos:

- $i = 0$ e $j > 0$: Nesse caso, não existem vértices a serem considerados e devem existir $j > 0$ vértices com estado $+1$. Logo, não existe solução.
- $i = 0$ e $j = 0$: Esse é o único caso que existe solução quando $i = 0$, já que não existem vértices a serem considerados mas também não deve existir nenhum vértice com estado $+1$.
- $i > 0$ e $j = 0$: Dentre os vértices $\mathit{filho}_{v,0}, \ldots, \mathit{filho}_{v,i-1}$, nenhum deles deve possuir estado $+1$. Sendo assim, calculamos todas as configurações predecessoras para as sub-árvores enraizadas nos vértices $\mathit{filho}_{v,0}, \ldots, \mathit{filho}_{v,i-2}$ com todos esses vértices possuindo estado -1, que pode ser calculado por $g_v^{rt}(i-1,j)$, e multiplicamos pelo número de configurações da sub-árvore $T_{\mathit{filho}_{v,i-1}}$ com o vértice $\mathit{filho}_{v,i-1}$ também possuindo estado -1, que é calculado por $\mathit{cfstate}(\mathit{filho}_{v,i-1}, rt)_{-1}$.
- $i > 0$ e $j > 0$: Para esse caso somamos o número de configurações predecessoras para as i primeiras sub-árvores considerando o vértice $\mathit{filho}_{v,i-1}$ com estado $+1$ como o número de configurações predecessoras considerando esse vértice com estado -1. Note que se assumimos que $\mathit{filho}_{v,i-1}$ possui estado $+1$, então dentre os vértices $\mathit{filho}_{v,0}, \ldots, \mathit{filho}_{v,i-2}$, exatamente $j-1$ desses vértices devem

possuir estado $+1$. Caso contrário, e consideramos $\mathit{filho}_{v,i-1}$ com estado -1, então dentre os $i-1$ primeiros filhos de v, ainda é necessário que j desses vértices possuam estado $+1$.

A análise para $h_v^{rt}(i,j)$ é omitida por ser análoga.

Agora, observe que a forma que calculamos g_v^{rt} e h_v^{rt} é bem semelhante à fórmula recursiva utilizada para calcular coeficientes binomiais:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

A aplicação pura e simples da recursão, portanto, resulta em um algoritmo onde o número de operações cresce muito rapidamente. E, para evitar isso, podemos novamente apenas utilizar a técnica conhecida como *memoization*. Ou, podemos calcular esses valores de maneira iterativa. Como o cálculo de g_v^{rt} e h_v^{rt} já ocorre dentro de uma função recursiva, calcular essas funções de maneira iterativa evita maior *overhead* de novas chamadas recursivas, e na prática acaba sendo mais eficiente e também mais compacto de implementar. A ideia é também mantermos uma tabela e preenchê-la em uma ordem tal que quando necessitarmos calcular $g_v^{rt}(i,j)$, todos os valores necessários para o cálculo já estejam corretamente armazenados na tabela. Da forma que definimos a recursão, podemos simplesmente iterar em ambos os parâmetros da função partindo do menor valor para o maior valor.

Para decidir qual das duas funções utilizar para um determinado vértice, basta verificar o valor de $Y(v)$. Utilizamos g_v^{rt} caso $Y(v)$ seja $+1$, e utilizamos h_v^{rt} caso $Y(v)$ seja -1. Suponha que $Y(v)$ seja $+1$. Então, se em uma configuração predecessora de Y, o vértice v possuir estado -1, então, dependendo do valor do estado de pai_v, é necessário que v possua k ou $k-1$ filhos com estado $+1$ na configuração predecessora. No caso onde v possua estado $+1$ na configuração predecessora, dependendo do valor do estado de pai_v, é necessário que v possua $d(v)-k+1$ ou $d(v)-k$ filhos com estado $+1$. Note que tendo esse número mínimo de filhos com estado $+1$, não é necessária preocupação com número de filhos com estado -1. A análise para o caso onde $Y(v)$ é igual a -1 é análoga.

Além de escolher qual das duas funções utilizar, temos ainda que calcular o número u. Formalmente, dado um vértice v e que pai_v possua estado c na configuração predecessora, definimos a função $\mathit{limiar}_v(\mathit{atual}, c, k)$ como sendo o menor número de filhos com estado $Y(v)$ que v deve possuir na configuração predecessora para que no próximo passo v possua estado $Y(v)$, dado que v possui na configuração predecessora estado *atual*.

$$limiar_v(atual, c, k) = \begin{cases} min(d(v) - k + 1, 0) & \text{se } Y(v) = atual \text{ e } c \neq Y(v) \\ min(d(v) - k, 0) & \text{se } Y(v) = atual \text{ e } c = Y(v) \\ k & \text{se } Y(v) \neq atual \text{ e } c \neq Y(v) \\ k - 1 & \text{se } Y(v) \neq atual \text{ e } c = Y(v) \end{cases}$$

onde definimos $min(a, b)$ como uma função que retorna o valor mínimo entre os inteiros a e b. O uso da função min foi necessário, pois não faz sentido dizer que um determinado vértice deve possuir pelo menos um número negativo de filhos com determinado estado na configuração predecessora.

O Algoritmo 4 contém a implementação da ideia descrita. Note que o algoritmo também é recursivo, assim como o algoritmo que apenas determina a existência de configuração predecessora, e calcula o valor da função recursivamente para os filhos, para então determinar o valor para o pai. Assim como o Algoritmo 2, armazenamos *cfstate* em uma tabela para o algoritmo não possuir complexidade de tempo exponencial. Note que não utilizamos uma função g_v^{rt} e h_v^{rt} explicitamente, mas utilizamos a tabela *tab* para esse fim, apenas verificando $Y(v)$ para decidir a ordem da multiplicação dos termos. A complexidade de tempo em cada vértice v é da ordem $O(d'(v)^2)$ nas duas iterações das linhas 22 e 23 do algoritmo. Fazendo o somatório para todos os vértices da árvore, isso é claramente $O(n^2)$, pois sabemos que $\sum_v d(v)^2 \leq (\sum_v d(v))^2 = 4m^2$. Como $m = n - 1$, $\sum_v d(v)^2$ é $O(n^2)$. Um exemplo onde esse limite é atingido ocorre quando a árvore em questão é uma estrela de n vértices, enraizando a árvore no vértice de grau $n - 1$. Nesse caso, o algoritmo atinge n^2 passos.

4.3 Caso Particular para Processos 2-reversíveis

Nessa subseção, mostramos que o ECP(2) está em P quando $\Delta(G) \leq 3$. Esse resultado, apesar de bastante restrito, é interessante pois utiliza uma outra técnica de demonstração que pode ser útil para casos mais gerais e também por cobrir a interessante classe de grafos cúbicos, como o grafo de *Petersen*, o grafo de *Heawood*, o *Tutte* $12-Cage$, além de inúmeros grafos cordais.

A demonstração utilizada para provar esse resultado reduz o problema a um problema de satisfatibilidade onde cada cláusula possui no máximo 2 literais. Esse problema é conhecido como 2-SAT e e está em P, podendo ser resolvido em tempo $O(N + M)$, onde N é o número de variáveis e M o número de cláusulas [23].

Algoritmo 4: $contafstate(v, c)$

```
1  início
2      se cfstate[v, c + 1] ≠ NIL então
3          retorna cfstate[v, c + 1];
4      d'(v) ⟵ d(v) − 2;
5      se v = raiz então
6          d'(v) ⟵ d(v) − 1;
7      contador ⟵ 0 ;
8      rotuloTeste ⟵ +1 ;
9      enquanto contador ≠ 2 faça
10         contador ⟵ contador + 1;
11         i ⟵ 0;
12         para cada f ∈ filhos_v faça
13             par[i] ⟵ contafstate(f, rotuloTeste) ;
14             i ⟵ i + 1;
15         se v = raiz então
16             l ⟵ limiar_v(rotuloTeste, ∞, k);
17         senão
18             l ⟵ limiar_v(rotuloTeste, c, k);
19         tab[0, 0] ⟵ 1;
20         para i ← 1 até d'(v) faça
21             tab[0, j] ⟵ 0;
22         para i ← 1 até d'(v) faça
23             para j ← 0 até d'(v) faça
24                 se Y(v) = +1 então
25                     se j > 0 então
26                         tab[i, j] ⟵ tab[i−1, j]×par[i]_− + tab[i−1, j−1]×par[i]_+;
27                     senão
28                         tab[i, j] ⟵ tab[i − 1, j] × par[i]_−;
29                 senão
30                     se j > 0 então
31                         tab[i, j] ⟵ tab[i−1, j]×par[i]_+ + tab[i−1, j−1]×par[i]_−;
32                     senão
33                         tab[i, j] ⟵ tab[i − 1, j] × par[i]_+;
34         se rotuloTeste = +1 então
35             cfstate[v, c + 1]_+ ⟵ Σ_{i=l}^{d'(v)} tab[d'(v), i] ;
36         senão
37             cfstate[v, c + 1]_− ⟵ Σ_{i=l}^{d'(v)} tab[d'(v), i] ;
38         rotuloTeste ⟵ −rotuloTeste ;
39     retorna cfstate[v, c + 1];
40 fim
```

A partir da configuração Y, estamos interessados em criar um conjunto de cláusulas S tal que S seja satisfatível se e somente se Y possui configuração predecessora. Para cada vértice v, criaremos os literais x_v e $\neg x_v$. Em uma solução para S, definimos que caso x_v seja *verdadeiro*, então, na configuração predecessora de Y, v possui estado $+1$. Caso contrário, v possui estado -1. Da mesma forma, se em uma configuração predecessora de Y, v possui estado $+1$, então x_v possui valor *verdadeiro* em S. Caso contrário, $\neg x_v$ possui valor *verdadeiro*.

Considerando apenas os grafos com $\Delta(G) \leq 3$, podemos montar um conjunto de cláusulas a partir de Y da seguinte forma:

Para cada vértice v tal que $Y(v) = +1$:

- Caso $d(v) = 1$: Na configuração predecessora v deve possuir o mesmo estado que na configuração Y, pois estamos considerando os processos 2-reversíveis, e, dessa forma, vértices com grau 1 nunca possuem alteração em seus estados. Portanto, adicionamos a seguinte cláusula:

  - x_v

  Com essa cláusula adicionada a S, temos que se S for satisfatível então x_v é *verdadeiro*, e, portanto, numa configuração predecessora de Y, v possui estado $+1$.

- Caso $d(v) = 2$ com vizinhos u e w: Se na configuração predecessora Y', v possui estado $+1$, então, em Y', pelo menos um dos vizinhos também possui estado $+1$. Se em Y', v possui estado -1, então os dois vizinhos possuem estado $+1$ em Y'. Dessa forma, podemos criar as seguintes cláusulas:

  - $\neg x_v \rightarrow x_u \equiv x_v \vee x_u$
  - $\neg x_v \rightarrow x_w \equiv x_v \vee x_w$
  - $x_v \rightarrow x_u \vee x_w \equiv \neg x_v \vee x_u \vee x_w$

  Analisando essas três cláusulas, vemos que quando x_v é *verdadeiro* então x_u ou x_w também tem que ser *verdadeiro* para que as três cláusulas sejam satisfeitas. E quando x_v é *falso*, então x_u e x_w devem possuir valor *verdadeiro*. Então, podemos simplificar mais e criamos apenas as cláusulas:

  - $x_v \vee x_u$
  - $x_v \vee x_w$
  - $x_u \vee x_w$

- Caso $d(v) = 3$ com vizinhos w, u e z: Se na configuração predecessora Y', v possui estado $+1$, então, em Y', pelo menos dois dos vizinhos também possuem

estado $+1$. Se, em Y', v, possui estado -1, então pelo menos dois dos três vizinhos possuem estado $+1$ em Y'. Dessa forma, podemos criar as seguintes as cláusulas:

- $\neg x_v \rightarrow x_u \vee x_w \equiv x_v \vee x_u \vee x_w$
- $\neg x_v \rightarrow x_u \vee x_z \equiv x_v \vee x_u \vee x_z$
- $\neg x_v \rightarrow x_w \vee x_z \equiv x_v \vee x_w \vee x_z$
- $x_v \rightarrow x_u \vee x_w \equiv \neg x_v \vee x_u \vee x_w$
- $x_v \rightarrow x_u \vee x_z \equiv \neg x_v \vee x_u \vee x_z$
- $x_v \rightarrow x_w \vee x_z \equiv \neg x_v \vee x_w \vee x_z$

É fácil ver que para satisfazer as seis cláusulas não é importante o valor que x_v assume. Sendo assim, podemos criar apenas as seguintes cláusulas:

- $x_u \vee x_w$
- $x_u \vee x_z$
- $x_w \vee x_z$

Para cada vértice v tal que $Y(v) = -1$, a análise é análoga:

- Caso $d(v) = 1$, criamos a cláusula:
  - $\neg x_v$
- Caso $d(v) = 2$ com vizinhos w e u, criamos as cláusulas:
  - $\neg x_v \vee \neg x_u$
  - $\neg x_v \vee \neg x_w$
  - $\neg x_u \vee \neg x_w$
- Caso $d(v) = 3$ com vizinhos w, u e z, criamos as cláusulas:
  - $\neg x_u \vee \neg x_w$
  - $\neg x_u \vee \neg x_z$
  - $\neg x_w \vee \neg x_z$

A construção do conjunto de cláusulas S nos dá diretamente que quando Y possuir alguma configuração predecessora, então S é satisfatível. Agora, supondo que S seja satisfatível, precisamos mostrar que Y possui configuração predecessora.

Podemos construir, como já explicado anteriormente, uma configuração Y' a partir de uma solução para S, onde o vértice v possui estado $+1$ se e somente se

nessa solução x_v é *verdadeiro*, e possui estado -1 se e somente se, nessa solução, x_v é *falso*.

Vamos supor, por contradição, que tal configuração não é uma configuração predecessora de Y. Ou seja, existe pelo menos um vértice v que não atinge o estado correspondente na configuração Y. Analisando o caso onde $Y(v) = +1$:

- $Y'(v) = +1$: Se for esse o caso, para v não atingir o estado $Y(v)$, então v possui pelo menos dois vizinhos com estado -1. Logo, v necessariamente é um vértice com pelo menos grau 2.

  Supondo $d(v) = 2$, então, se existem tais vizinhos u e w com estado -1, teríamos a cláusula $x_u \vee x_w$ não satisfeita, mas como estamos considerando Y' a partir de uma solução para S, isso não é possível.

  Supondo $d(v) = 3$, então, de maneira semelhante, existirá pelo menos uma cláusula não satisfeita.

- $Y'(v) = -1$: Se for esse o caso, para v não atingir o estado $Y(v)$, então v possui, no máximo, um vizinho com estado $+1$. Note que $d(v)$ não pode ser igual a 1, pois, como $Y(v) = +1$, então temos em S a cláusula x_v, que, se satisfeita, implica em $Y'(v) = +1$.

  Supondo $d(v) = 2$ com vizinhos u e w, então uma das cláusulas $x_v \vee x_w$ ou $x_v \vee x_u$ não será satisfeita. Já se $d(v) = 3$, como temos cláusulas de dois literais envolvendo os três vizinhos de v, pelo menos uma dessas cláusulas não será satisfeita.

O caso onde $Y(v) = -1$ é análogo, e, portanto, concluímos que se S é satisfatível então a configuração Y' obtida a partir de um solução para S é necessariamente uma configuração predecessora de Y. Pois, caso não seja, temos uma contradição com o fato de S ser satisfatível.

A partir dos n vértices do grafo, criamos n variáveis e, no pior caso, $3n$ cláusulas. Cada cláusula possui no máximo dois literais. Logo, criamos uma instância do 2-SAT. É possível resolver o 2-SAT em complexidade de tempo $O(N+M)$, onde N é o número de variáveis e M o número de cláusulas. Sendo assim, resolvemos o ECP(2) em complexidade de tempo $O(n)$ quando $\Delta(G) \leq 3$.

Poderíamos estender a classe de grafos, aumentando o limite superior para $\Delta(G)$. Mas nesse caso, o número de literais em cada cláusula não seria mais limitado a no máximo dois no caso geral.

Capítulo 5

Conclusões

Nesse capítulo, fazemos uma breve revisão dos problemas estudados nesse trabalho, bem como os resultados obtidos. Uma lista de possíveis trabalhos futuros é sugerida ao final.

Nessa dissertação, apresentamos os processos k-reversíveis em grafos. Sobre esses processos, estudamos o comportamento das configurações de estados ao longo do tempo. Em particular, estudamos limites superiores para o comprimento do transiente e para o comprimento do período em tais processos, e também as chamadas *configurações jardim-do-éden*, que são configurações relacionadas com o problema conhecido como Existência de Configuração Predecessora. No estudo do transiente e do período, apresentamos resultados existentes na literatura e, além disso, para alguns desses resultados, utilizamos provas alternativas que em alguns casos fornencem limites superiores mais justos. Para o caso dos processos k-reversíveis, denominamos Existência de Configuração Predecessora como Ecp(k) e todos os resultados apresentados para esse problema são inéditos.

No Capítulo 2, mostramos os resultados existentes na literatura para *processos que operam com limiares* envolvendo os limites superiores para o comprimento do transiente e comprimento do período. Como os processos k-reversíveis são casos específicos desse tipo de processo, podemos utilizar tais resultados para obter diretamente limites superiores para os processos k-reversíveis. O principal resultado diz que para qualquer k e qualquer grafo, o comprimento máximo do período é dois. Esse resulado é particularmente surpreendente pois se sustenta para uma varieadade de processos que operam com limiares, bastando apenas que a matriz associada a tal processo seja simétrica. Logo, é muito simples e eficiente identificar se uma determinada configuração é periódica, bastando apenas simular o processo a partir daquela configuração, e concluindo em complexidade $O(n + m)$ se tal configuração é periódica. Esse limite é justo e é bastante simples termos exemplos de configurações iniciais que possuem períodos de tamanho 1 ou 2. No Capítulo 3, mostramos uma prova alternativa própria para processos k-reversíveis através de um operador

monotônico que chamamos de *função de energia para processos k-reversíveis.*

O estudo do Capítulo 2 contém, também, limites para o comprimento do transiente nos processos k-reversíveis. Mostramos que o comprimento do transiente é no máximo $2m + n(2\Delta(G) - 2)$ a partir dos resultados obtidos para processos que operam com limiares. No Capítulo 3, utilizando o nosso operador monotônico mostramos que esse limite pode ser reduzido para $n(\Delta(G) + 1) - 1$. No entanto, não encontramos configurações que possuíssem comprimentos que atingem o limite superior obtido, e não podemos afirmar que esses limites são justos. Na verdade, os limites, da forma que foram calculados, intuitivamente parecem não ser justos.

Ainda no Capítulo 3, para o caso das árvores, apresentamos uma conjectura para o limite superior justo em processos 2-reversíveis. Nesse caso, a conjectura diz que o comprimento do transiente é no máximo $n - 3$, e também que todas as árvores que possuem configurações que atingem transiente com esse comprimento possuem uma forma iterativa de construção. Dessa forma, em conjunto com a conjectura, apresentamos o algoritmo que constrói todas essas árvores.

O Capítulo 4 é dedicado ao ECP(k). Nesse capítulo, mostramos que é um problema NP-Completo para o caso geral, utilizando a redução a partir do problema 3-SAT-EXATAMENTE-2. A partir dessa demonstração, obtemos também diretamente que o problema é NP-Completo para a classe de grafos bipartidos. Assim, como diversos problemas em teoria dos grafos, esse problema pode ser resolvido eficientemente quando a classe de grafos é a classe das árvores. Dessa forma, nesse capítulo, apresentamos um algoritmo que determina em tempo $O(n)$ se uma configuração é uma configuração jardim-do-éden. O algoritmo se baseia em um algoritmo *backtracking* que possui complexidade de tempo exponencial, e utilizamos a técnica conhecida como *memoization* para reduzir essa complexidade de tempo para complexidade polinomial.

Uma variação do problema, que surge naturalmente, é contar o número de configurações predecessoras, que obviamente é um problema NP-Completo para o caso geral. De maneira similar, mostramos um algoritmo polinomial de complexidade $O(n^2)$ para o caso das árvores. É também um algoritmo de *backtracking* cuja complexidade de tempo a partir das técnicas conhecidas como *programação dinâmica* e *memoization*, conseguimos reduzir para $O(n^2)$.

Uma outra técnica que pode ser útil para a prova de algoritmos de tempo polinomial para casos restritos do ECP(k) é a redução do problema para algum problema de satisfatibilidade que possa ser resolvido em tempo polinomial. Utilizando essa técnica, reduzimos o problema quando $\Delta(G) \leq 3$ e $k = 2$. Esse problema é bem restrito, mas abrange a interessante classe dos grafos cúbicos, que incluem, por exemplo, o famoso grafo de *Petersen*. Nesse caso, conseguimos reduzir o problema a uma instância do problema 2-SAT que é um problema clássico da literatura e conhecido

por ser um problema de satisfatibilidade resolvido em tempo polinomial. Essa forma de demonstração, no entanto, não parece ser muito útil caso o interesse seja contar o número de configurações predecessoras.

As técnicas utilizadas no Capítulo 4 foram aplicadas somente ao caso dos processos k-reversíveis, mas podem ser adaptadas para diversos outros processos em grafos. O estudo de classes de grafos restritos parece ser bem promissor na obtenção de algoritmos de tempo polinomial.

O estudo apresentado nesse trabalho deixa algumas questões em aberto, como por exemplo:

- Dada uma configuração inicial em um processo k-reversível, é possível obter um algoritmo mais eficiente que a simples simulação para determinar se uma configuração possui comprimento do período 1 ou 2? A questão é obter algum algoritmo que determine essa característica a partir da estrutura do grafo, da configuração inicial e do limiar k, ou provar que não existe algoritmo mais eficiente.

- Obter um limite superior justo para o comprimento do transiente, ou encontrar exemplos que atinjam os limites apresentados nessa dissertação.

- Demonstrar a validade da conjectura para o limite do comprimento do transiente apresentada no Capítulo 3, bem como demonstrar se todas as árvores construídas pelo algoritmo são as únicas que possuem o transiente máximo.

- Estudar o ECP(k) para outras classes de grafos buscando obter algoritmos de tempo polinomial em tais classes.

- Provar que o problema de contar o número de configurações predecessoras quando $\Delta(G) \leq 3$ e $k = 2$ é #P-completo ou mostrar um algoritmo que o resolva em tempo polinomial.

Referências Bibliográficas

[1] GHIGLIA, D. C., MASTIN, G. A., ROMERO, L. A. "Cellular automata method for phase unwrapping", *Journal of the Optical Society of America A*, v. 4, n. 1, pp. 267–280, 1987.

[2] HOPFIELD, J. J. "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences of the USA*, v. 79, n. 8, pp. 2554–2558, abr. 1987.

[3] ADLER, J. "Bootstrap percolation", *Physica A*, v. 171, pp. 453–470, 1991.

[4] DEGROOT, M. H. "Reaching a consensus", *Journal of the American Statistical Association*, v. 69, pp. 167–182, 1974.

[5] MIKLER, A. R., VENKATACHALAM, S., ABBAS, K. "Modeling infectious diseases using global stochastic cellular automata", *Journal of Biological Systems*, v. 13, n. 4, pp. 421–439, 2005.

[6] LUCCIO, F., PAGLI, L., SANOSSIAN, H. "Irreversible dynamos in butterflies". In: *Proceedings of the Sixth Colloquium on Structural Information and Communication Complexity*, 1999.

[7] PELEG, D. "Size bounds for dynamic monopolies", *Discrete Applied Mathematics*, v. 86, pp. 263–273, 1998.

[8] DREYER JUNIOR, P. A. *Application and Variations of Domination in Graphs*. Tese de Ph.D., The State University of New Jersey, New Brunswick, New Jersey, Estados Unidos, 2000.

[9] GOLES, E., OLIVOS, J. "Periodic behavior of binary threshold functions and applications", *Discrete Applied Mathematics*, pp. 93–105, 1985.

[10] GOLES, E., OLIVOS, J. "The convergence of symmetric threshold automata", *Information and Control*, v. 51, pp. 98–104, 1981.

[11] SUTNER, K. "On the computational complexity of finite cellular automata", *Journal of Computer and System Sciences*, v. 50, pp. 87–97, fev. 1995.

[12] BARRETT, C., MARATHE, M. V., RAVI, S. S., et al. "Gardens of Eden and Fixed Points in Sequential Dynamical Systems". In: *Proceedings of Discrete Models: Combinatorics, Computation, and Geometry (DM-CCG)*, pp. 95–110, Discrete Mathematics and Theoretical Computer Science (DMTCS), 2001.

[13] TOSIC, P. T. *Modeling and Analysis of the Collective Dynamics of Large-Scale Multi-Agent Systems: A Cellular and Network Automata based Approach.* Relatório técnico, 2006.

[14] CENTENO, C. C., DOURADO, M. C., PENSO, L. D., et al. "Irreversible conversion of graphs", *Theoretical Computer Science*, v. 412, pp. 3693–3700, 2011.

[15] GOLES-CHACC, E., FOGELMAN-SOULIE, F., PELLEGRIN, D. "Decreasing Energy Functions as a Tool for Studying Threshold Networks", *Discrete Applied Mathematics 12*, pp. 261–277, 1985.

[16] POLJAK, S., SURA, M. "On periodical behavior in societies with symmetric influences", *Combinatorica*, v. 3, pp. 119–121, 1983.

[17] GREEN, F. "NP-Complete Problems in Cellular Automata", *Complex Systems*, v. 1, n. 3, pp. 453–474, 1987.

[18] MOORE, E. F. "Machine models of self-reproduction". In: *Proc. Symp. Applied Mathematics 14*, pp. 17–33, 1962.

[19] MYHILL, J. "The converse of Moore's Garden-of-Eden theorem". In: *Proceedings of the American Mathematical Society 14*, pp. 685–686, 1963.

[20] GAREY, M. R., JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[21] BONDY, J. A., MURTY, U. S. R. *Graph Theory with Applications.* North Holland p. 74, 1976.

[22] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1990.

[23] ASPVALL, B., PLASS, M. F., TARJAN, R. E. "A linear-time algorithm for testing the truth of certain quantified boolean formulas", *Information Processing Letters*, v. 8, n. 3, pp. 121–123, mar. 1979.