UNIVERSIDADE FEDERAL FLUMINENSE

Alan Diêgo Aurélio Carneiro

Sobre Deadlocks e sua Resolução: dos Modelos de Grafos de Espera à Complexidade Computacional

> NITERÓI 2016

UNIVERSIDADE FEDERAL FLUMINENSE

Alan Diêgo Aurélio Carneiro

Sobre Deadlocks e sua Resolução: dos Modelos de Grafos de Espera à Complexidade Computacional

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização

Orientadores: Fábio Protti Uéverton dos Santos Souza

> NITERÓI 2016

Alan Diêgo Aurélio Carneiro

Sobre Deadlocks e sua Resolução: dos Modelos de Grafos de Espera à Complexidade Computacional

> Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização

Aprovada em 16 de março de 2016.

BANCA EXAMINADORA

Prof. Fábio Protti - Orientador, UFF

Prof. Uéverton dos Santos Souza - Orientador, UFF

Prof. Lúcia M. A. Drummond, UFF

Prof. Valmir Carneiro Barbosa, UFRJ

Prof. Lilian Markenzon, UFRJ

Niterói 2016

Aos meus pais, irmãos e minha namorada que, com muito carinho e apoio, não mediram esforços para que eu chegasse até essa etapa de minha vida.

Resumo

Computação distribuída consiste em um conjunto de processos que cooperam entre si e compartilham recursos através de troca de mensagens para a execução de uma dada tarefa. Deadlocks em uma computação distribuída ocorrem quando um conjunto de processos espera indefinidamente por recursos provenientes do mesmo conjunto. Sistemas que efetuam uma computação distribuída são usualmente representados por grafos de espera, onde o comportamento de cada processo é determinado por um modelo de deadlock, que define as regras de como esses processos tornam-se executáveis.

Neste trabalho, primeiramente são revisados os principais modelos de deadlock, e em alternativa ao modelo E-Ou, é proposto um novo modelo de deadlock, de mais simples representação e compatível com estruturas de dados clássicas, os grafos e/ou. Posteriormente, descreve-se formalmente como um modelo de deadlock generaliza um segundo modelo, primando a complexidade computacional como parâmetro. Por consequência, a hierarquia clássica de modelos de deadlock é modificada, e duas novas hierarquias são apresentadas; a primeira leva em consideração apenas tempo computacional, e a segunda considera também a profundidade.

Em seguida, são provadas equivalências e transformações polinomiais entre os grafos de espera nos modelos clássicos de deadlock e os grafos e/ou (grafos xy), os mais conhecidos na literatura. Para tais grafos, são apresentadas duas estruturas genéricas: a primeira, chamada de subgrafo solução, é uma estrutura já conhecida na literatura, que, para grafos de espera em computação distribuída, indica que todos os processos que pertencem ao subgrafo não estão em deadlock; a segunda, chamada de subgrafo não-solução, caracteriza situações de deadlock.

Finalmente, define-se Deadlock-Resolution(a, b) como uma classe de problemas de otimização para resolução de deadlock. Cada problema é indicado pela combinação de dois parâmetros. O primeiro, 'a', indica o modelo de deadlock do grafo de espera. O segundo, 'b', indica o tipo de operação a ser utilizada para a eliminação de deadlocks no grafo. É desenvolvido um estudo sobre a complexidade computacional de cada problema, onde, sempre que possível, são fornecidos algoritmos eficientes para a solução dos problemas. Para o problema Deadlock-Resolution(Ou, Vértice), prova-se sua intratabilidade para instâncias genéricas, e são estudadas diversas classes de grafos que correspondem a características estruturais que tornam o problema intratável.

Palavras-chave: sistemas distribuídos, deadlock, grafos de espera, grafos e/ou, complexidade computacional.

Abstract

Distributed computation consists of a set of independent processors which cooperate with each other and are interconnected by a communication network that supports resource sharing. A deadlock occurs in a distributed computation when a set of processes wait indefinitely for resources from each other. Systems that perform a distributed computation are usually represented by wait-for graphs, where the behavior of each process is determined by a deadlock model, which defines the rules of a distributed computation by determining how processes can become executable.

This work first reviews the main deadlock models, and alternatively to the AND-OR model, proposes a new deadlock model in a simpler representation and compatible with classical data structures, graphs *and/or*. Next, we describe how a deadlock model generalizes a second model, prioritizing the computational complexity as a parameter. Consequently, the classical hierarchy of deadlock models known is modified, and two new hierarchies are shown; the former takes into account only computational time, and the latter also considers depth.

Next, we prove equivalence and polynomial transformations between wait-for graphs in classical deadlock models and and/or graphs (xy graphs), the most known in the literature. For such graphs two generic structures are presented; the first, called *solution* subgraph and already studied in the literature, indicates that all the processes belonging to the subgraph are not in deadlock; the second, called *non-solution* subgraph, characterizes deadlock situations.

Finally, we define Deadlock-Resolution(a, b) as a class of optimization problems for deadlock resolution. Each problem is indicated by the combination of two parameters. The first, 'a', indicates the deadlock model of the wait-for graph. The second, 'b', indicates the type of operation to be used in order to solve all the deadlocks in the input graph. We develop a full study on the computational complexity of each problem, where, when possible, efficient algorithms are provided. For the Deadlock-Resolution(OR, Vertex) problem, besides proving the intractability for general instances, we study several graph classes with structural features that make the problem intractable.

Keywords: distributed systems, deadlock, wait-for graphs, *and/or* graphs, computational complexity.

Lista de Figuras

2.1	Grafo de espera: modelo E	8		
2.2	Grafo de espera: modelo Ou	8		
2.3	Grafo de espera: modelo E-Ou	8		
2.4	Grafo de espera: modelo X-de-Y	8		
2.5	Grafo de espera: modelo X-de-Y disjuntivo.	8		
2.6	Transformação de grafo de espera no modelo E-Ou para grafo $\mathit{e/ou}.$	12		
3.1	l Organização clássica dos modelos de deadlock			
3.2	Exemplo de uma transformação clássica de uma instância no modelo X-de- Y para o modelo E-Ou	17		
3.3	Exemplo de transformação de uma instância no modelo E-Ou para o modelo E-Ou simplificado.	19		
3.4	Comparadores de bits: (a) porta comparadora de bits; (b) comparador utilizando as portas lógicas monótonas	22		
3.5	Circuitos de redes de ordenação: (a) circuito com porta comparadora de bits; (b) rede de ordenação utilizando as portas lógicas monótonas	23		
3.6	Hierarquia dos modelos de deadlock de acordo com seu poder de expressão em tempo polinomial	24		
3.7	Hierarquia dos modelos de deadlock de acordo com seu poder de expressão em tempo polinomial, preservando a ordem de profundidade	25		
3.8	Exemplo de subgrafo solução enraizado em v_1	26		
3.9	Exemplo de subgrafo não-solução	28		
4.1	Grafo e árvore obtida pela contração de CFCs	34		
5.1	Grafo G_F resultante da transformação polinomial da fórmula F	40		

5.2	Grafo ${\cal G}_F$ fortemente conexo resultante da transformação polinomial da			
	fórmula F	42		
5.3	Grafo G_F bipartido resultante da transformação polinomial da fórmula $F..$	44		
5.4	Tipos de vértices em um grafo subcúbico			
5.5	Vértices sumidouro em grafos subcúbicos			
5.6	Fontes em grafos subcúbicos	47		
5.7	Vértices com pelo menos uma aresta de entrada e pelo menos uma de saída.	47		
5.8	Subtipos de vértices A em uma CFC	49		
5.9	Knots restantes após o pré-processamento descrito nos Lemas 5.8, 5.7 e 5.10.	51		
5.10	Refinamento obtido com a Observação 5.12 e Lema 5.13	53		
5.11	Grafo bipartido resultante das contrações de componentes C e knots $K.$.	54		
5.12	2 Exemplo de conjunto M' que satura todos os vértices em K 5			
5.13	Exemplo de conjunto M' que satura $t-1$ vértices. O vértice destacado em			
	vermelho não é saturado por M'	55		

Lista de Tabelas

3.1	Poder de expressão entre os modelos de deadlock. O símbolo \subseteq significa "é	significa "é	
	generalizado por".	20	
4.1	Complexidades em aberto de problemas Deadlock-Resolution (a,b).	32	
4.2	Complexidades de problemas Deadlock-Resolution (a, b)	38	
5.1	Complexidade para diferentes classes de grafos do problema Deadlock-		
	Resolution(Ou, Vértice)	59	
6.1	Complexidade final da classe de problemas Deadlock-Resolution (a, b)	61	

Lista de Algoritmos

1	Detecção de knots	35
2	Deadlock-Resolution(Ou, Arco)	36
3	Deadlock-Resolution(Ou, Sumidouro)	37
4	Algoritmo 2-aproximativo	52
5	Algoritmo resolução subcúbico	58

Sumário

1	Intr	odução	1
	1.1	Conceito de Deadlock	1
	1.2	Objetivos e Contribuição da Dissertação	2
	1.3	Organização	3
2	Con	ceitos Fundamentais	4
	2.1	Sistemas Distribuídos	4
	2.2	Grafos de Espera	6
	2.3	Deadlocks	8
	2.4	Grafos e/ou e Grafos xy	10
3	Rev	isitando Modelos de Deadlock	13
	3.1	Hierarquia dos Modelos de Deadlock	13
		3.1.1 Poder de Expressão em Tempo Polinomial	14
		3.1.1.1 Complexidade de Circuitos	21
	3.2	Caracterizações por Estruturas de Dados	25
		3.2.1 O Modelo X-de-Y	28
	3.3	Considerações Finais	29
4	Pro	olemas sobre Resolução de Deadlock	31
	4.1	A Classe de Problemas Deadlock-Resolution (a, b)	31
	4.2	Modelo E	32
	4.3	Modelo Ou	34

	4.4	Modelo E-Ou e Modelo X-de-Y		
	4.5 Resultados			
5	Dea	dlock-R	esolution(Ou, Vértice)	39
	5.1	NP-D	ificuldade no Modelo Ou	39
	5.2	NP-D	ificuldade para Grafos Fortemente Conexos	41
	5.3	NP-D	ificuldade para Grafos Bipartidos	43
		5.3.1	NP-Dificuldade para Grafos Planares	45
	5.4	Grafo	s Subcúbicos	45
		5.4.1	Fontes e Sumidouros	46
		5.4.2	Refinamento do Grafo	48
	5.4.3 Algoritmo Aproximativo			50
		5.4.4	Resolução em Tempo Polinomial	51
	5.5	Consi	derações Finais	59
6	Con	clusões	e Trabalhos Futuros	60
	Refer	ências		62

Capítulo 1

Introdução

Segundo a definição de Andrew Tanenbaum [58], um sistema distribuído é uma "coleção de computadores independentes que se apresenta ao utilizador como um sistema único e consistente". Outra definição, de George Coulouris [21], diz: "coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita a partilha dos recursos do sistema: hardware, software e dados".

A computação distribuída consiste em utilizar diversos computadores interligados por uma rede de computadores, ou vários processadores trabalhando em conjunto no mesmo computador, para processar cooperativamente uma determinada tarefa de forma coerente e transparente, ou seja, como se apenas um único computador centralizado estivesse executando a tarefa [21]. A união desses diversos computadores com o objetivo de compartilhar a execução de tarefas é conhecida como sistema distribuído.

Uma das principais características dos sistemas distribuídos é o compartilhamento de recursos. Os recursos são compartilhados via comunicação de rede. Tal compartilhamento possibilita vantagens, como o aumento de desempenho, de escalabilidade, etc. Em contrapartida, um problema bastante comum ao compartilhamento de recursos é o *deadlock* [25].

1.1 Conceito de Deadlock

Deadlock é um problema fundamental em projeto de sistemas concorrentes [36]. Apesar de este problema ser amplamente estudado em sistemas de memória compartilhada [18, 23], o problema permanece de difícil solução e possui várias questões em aberto. Algumas destas questões serão exploradas neste trabalho. Um conjunto de processos está em deadlock se cada processo deste conjunto está bloqueado, aguardando resposta de um outro processo desse mesmo conjunto; isto é, os processos não conseguem prosseguir a execução, esperando um evento ou resposta que somente outro processo do próprio conjunto pode enviar. Em outras palavras, uma situação de deadlock é caracterizada pelo impedimento permanente para um conjunto de processos proceder com suas tarefas devido a uma condição que bloqueia pelo menos um recurso essencial a ser adquirido [7].

Deadlock é um fenômeno comum a sistemas onde acontece compartilhamento de recursos, como: sistemas operacionais [59, 63]; cruzamentos de trânsito [18]; linhas férreas [44, 45]; cooperação multi-robôs [64]; dentre outros exemplos.

O foco deste trabalho é estudar problemas combinatórios relativos à resolução de deadlocks em sistemas distribuídos através de intervenções externas, isto é, de algum tipo de operação (usualmente terminação de processos ou de requisições) que geralmente é passo seguinte à detecção de deadlocks.

1.2 Objetivos e Contribuição da Dissertação

Essa dissertação tem como principal objetivo o estudo de problemas de resolução de deadlock, estudando sua complexidade na versão de otimização, considerando os principais modelos de computação distribuída da literatura.

Os demais objetivos e contribuições são:

- 1. Analisar e propor uma reorganização da hierarquia clássica dos modelos de deadlock, dados alguns parâmetros posteriormente explorados.
- Estudar estruturas que garantam condições necessárias e suficientes para a existência de deadlock.
- Desenvolver algoritmos determinísticos e polinomiais para detecção e solução dos deadlocks, sempre que possível.

Embora deadlock seja um problema de grande relevância para a área de sistemas distribuídos, nesta dissertação focamos no estudo de algoritmos sequenciais para os problemas abordados. Essa decisão é baseada no Lema 1.1, que demonstra que o estudo e a classificação da complexidade de problemas computacionais, sem perda de generalidade, pode ser efetuado sobre o ponto de vista sequencial.

Lema 1.1. [28, 3] Uma linguagem L é decidível em tempo sequencial $n^{O(1)}$ se e somente se L é decidível em tempo paralelo $n^{O(1)}$ com $n^{O(1)}$ processadores.

O lema acima conduz ao seguinte corolário:

Corolário 1.2. A menos que P = NP, um problema NP-completo não pode ser decidível em tempo $n^{O(1)}$, independentemente do número de processadores utilizados.

Demonstração. Suponha a utilização de $n^{O(1)}$ processadores. Pelo lema 1.1, se um problema L é decidível polinomialmente em tempo paralelo, também o será em tempo sequencial, a menos que P = NP.

Por outro lado, suponha um número não polinomial de processadores. Para a obtenção da solução de uma instância χ de um problema NP-completo, a alocação de tarefas a um número não polinomial de processadores demanda um tempo não polinomial.

1.3 Organização

O restante deste trabalho é organizado da seguinte maneira. O Capítulo 2 apresenta a revisão bibliográfica. No Capítulo 3 apresentamos estruturas que caracterizam deadlock e a reestruturação da hierarquia de modelos de deadlock a partir de parâmetros restritivos. No Capítulo 4 definimos a classe de problemas Deadlock-Resolution(a, b), e a seguir desenvolvemos estudos de complexidade para esta classe de problemas. No Capítulo 5 desenvolvemos um estudo mais aprofundado da complexidade do problema Deadlock-Resolution(Ou, Vértice). Finalmente, no Capítulo 6, apresentamos nossas conclusões e recomendações para trabalhos futuros.

Capítulo 2

Conceitos Fundamentais

Este capítulo apresenta uma revisão bibliográfica com os conceitos e definições pertinentes aos problemas abordados. O capítulo é dividido em quatro partes. A primeira parte trata da conceituação e funcionamento de uma computação distribuída. A segunda introduz os grafos de espera, que são estruturas de dados utilizadas para representar uma computação distribuída. A terceira apresenta os conceitos e definições de deadlock. Finalmente é abordada a representação da computação distribuída com o uso de estruturas de dados clássicas.

2.1 Sistemas Distribuídos

Sistemas distribuídos compreendem em um conjunto de processadores independentes (coleção de computadores autônomos) interligados por uma rede de comunicação que permite o compartilhamento de recursos [6]. Tais processadores não compartilham fisicamente memória de forma direta; dessa maneira, informações são necessariamente compartilhadas por troca de mensagens através da rede de comunicação.

A arquitetura de um sistema distribuído pode ser abstraída por um grafo G = (V, E), onde V é um conjunto de processadores que executam toda a computação distribuída, e E é um conjunto de canais de comunicação, que funciona como meio de compartilhamento de recursos. Existem duas principais arquiteturas de computação distribuída [5], síncrona e assíncrona, que diferem principalmente pela caracterização do tempo.

Arquitetura Síncrona – É principalmente caracterizada pela existência de um relógio global conhecido por todos os processos. Podemos considerar o relógio como um contador de passos $s \ge 0$, e a troca de mensagens entre processos vizinhos de G ocorre em um

passo de relógio; todas as tarefas dadas a um processo no tempo s são necessariamente terminadas ao iniciar o passo s + 1 [5].

Arquitetura Assíncrona – É caracterizada pela inexistência de um relógio global: cada processo possui seu próprio relógio local, o qual é totalmente independente dos demais. A troca de mensagens entre processos ocorre em tempo finito (garantia de entrega) porém indeterminado. Um processo somente executa alguma ação ao receber mensagem de um vizinho; tal ação pode incluir o envio de mensagens. Pelo menos um processo deve ter algum tipo de inicio espontâneo ou intervenção externa para iniciar a computação, conforme explicado em [4, 16, 22].

O modelo síncrono oferece algumas vantagens devido à existência de um relógio global, porém usualmente tal representação não ocorre na prática; portanto, a partir deste ponto, ao falar de computação distribuída ou sistema distribuído, será assumido o modelo assíncrono.

Em um sistema distribuído, o compartilhamento de recursos é realizado necessariamente por trocas de mensagens. O grafo G = (V, E) que representa uma arquitetura distribuída é insuficiente para a representação de uma computação distribuída em andamento. Uma aresta $v_i v_j$ existe em E se existe um canal direto de comunicação entre v_i e v_j ; entretanto, tal abstração não nos fornece nenhuma informação precisa sobre as trocas de mensagens, fundamentais para o estudo de deadlocks. Assim, faremos uso de grafos de espera, que serão apresentados na próxima seção.

Um conjunto de processos está em deadlock se cada processo deste conjunto está bloqueado, aguardando resposta de um outro processo desse mesmo conjunto; isto é, os processos não conseguem prosseguir a execução, esperando um evento ou resposta que somente outro processo do próprio conjunto pode enviar. Atreya et al. [4] apontam o deadlock, propriedade estável em uma computação distribuída, como um problema fundamental em sistemas distribuídos. Uma propriedade é dita estável se, existindo para um estado global Ψ , também existirá em qualquer estado global subsequente a Ψ .

O principal objetivo deste trabalho é estudar problemas combinatórios relativos à resolução de deadlocks. Nosso estudo é inspirado em sistemas distribuídos; no entanto, os problemas abordados não se restringem a esse cenário, sendo aplicáveis a quaisquer cenários onde deadlocks possam ocorrer. Para tal, consideraremos os grafos de espera de um sistema distribuído, definidos a seguir.

2.2 Grafos de Espera

Barbosa e Benevides [9] definem grafos de espera como estruturas de análise e abstração de sistemas distribuídos. Estes grafos, melhor detalhados adiante, são estruturas dinâmicas, ou seja, mudam de acordo com as requisições e respostas do sistema. Podemos desconsiderar a natureza da dependência entre os nós da rede, isto é, não será relevante para os propósitos deste estudo o que faz um nó esperar por outro e sim se a espera ocorre, pois o estudo é direcionado à existência de deadlocks, que como vimos é uma propriedade estável.

Uma vez que ocorre deadlock em um grupo de processos, apenas por meio de intervenção externa (seguida da detecção do deadlock) podemos solucioná-lo. Sendo assim, podemos desconsiderar alguns aspectos, e trabalhar apenas com um estado global do sistema durante a computação (chamado de *snapshot*) [15]. Sempre que posteriormente nos referirmos a um grafo de espera G, este corresponderá a um grafo estático referente a um snapshot do sistema.

Definiremos formalmente o grafo de espera como um grafo direcionado G = (V, E), onde V é o conjunto de processos (vértices) e E o conjunto de requisições (arcos). Um arco $v_i v_j$ de um processo v_i para v_j existe em E se v_i espera um sinal de v_j .

Conceitos e Notações Adicionais. Dado um vértice $v_i \in V$, seja D_i o conjunto de vértices que descendentes de v_i em G (vértices alcançáveis por v_i , incluindo o próprio) e A_i conjunto de vértices ancestrais de v_i em G (vértices que alcançam v_i , incluindo o próprio). Denotamos por $O_i \subseteq D_i$ o conjunto de vértices imediatamente descendentes de v_i em G (descendentes à distância um de v_i) e $I_i \subseteq A_i$ é o conjunto de ancestrais imediatos de v_i em G (ancestrais à distância um de v_i). Vértices em $D_i \setminus A_i$ são chamados de subordinados de v_i em G.

Sistemas distribuídos são representados por grafos de espera atrelados a um modelo de dependência. Modelos de dependência proporcionam abstração das regras que governam o aguardo de um processo para sua execução. Existem cinco modelos clássicos de espera na literatura:

Modelo E: Um processo p_i somente torna-se executável quando toda requisição (p_i, p_j) é atendida, isto é, todas as requisições (p_i, p_j) forem satisfeitas, e portanto removidas do grafo de espera corrente, tornando p_i um vértice sumidouro. Este modelo (Figura 2.1) é caracterizado por situações onde a conjunção de respostas de todas as requisições são necessárias para execução de p_i [13, 36, 49].

- **Modelo Ou:** Neste modelo, para tornar um processo p_i executável, basta que uma requisição (p_i, p_j) seja atendida; neste caso todas as demais requisições são desconsideradas, tornando p_i um vértice sumidouro. O modelo (Figura 2.2) é caracterizado por situações de característica disjuntiva onde apenas uma resposta é necessária para p_i [13, 36, 49].
- Modelo E-Ou: Neste modelo, existem $t_i \geq 1$ subconjuntos de O_i associados a p_i . Tais conjuntos são denotados por $O_i^1, \dots, O_i^{t_i}$ onde $O_i = \{O_i^1 \cup \dots \cup O_i^{t_i}\}$. Para que o processo p_i se torne executável, basta que todas as requisições (p_i, p_j) de pelo menos um subconjunto de O_i sejam atendidas; neste caso todas as demais requisições são desconsideradas, tornando p_i um vértice sumidouro. Este modelo (Figura 2.3) é caracterizado por processos p_i em que a conjunção de recursos recebida por cada grupo O_i é equivalente [8, 13, 36, 49].
- **Modelo X-de-Y:** Neste modelo, cada processo p_i é associado a um par de inteiros (x_i, y_i) . Para tornar um processo p_i executável, basta que x_i das y_i requisições (p_i, p_j) sejam atendidas $(y_i$ indica a quantidade total de requisições (p_i, p_j) não atendidas); neste caso todas as demais requisições são desconsideradas, tornando p_i um vértice sumidouro. Este modelo (Figura 2.4) é caracterizado por situações onde p_i requisita mais do que ele realmente precisa e espera somente a quantidade x_i de recursos necessários [12, 13, 36].
- Modelo X-de-Y Disjuntivo: Neste modelo, para caca processo p_i existem $u_i \geq 1$ pares de inteiros, denotados por $(x_i^1, y_i^1), \dots, (x_i^{u_i}, y_i^{u_i})$. Estes pares de inteiros são correspondentes a subconjuntos Q_i^j de O_i , onde $O_i = \{Q_i^1 \cup \dots \cup Q_i^{u_i}\}$ e $y_i^1 = |Q_i^1|, \dots, y_i^{u_i} = |Q_i^{u_i}|$. Para que um processo p_i se torne executável, basta que, p_i receba concessão de recursos de x_i^1 dos y_i^1 processos em Q_i^1 , ou x_i^2 dos y_i^2 processos em Q_i^2 , e assim por diante. Assume-se que se $Q_i^j \subseteq Q_i^k$ então $x_i^j \leq x_i^k$. Este modelo é caracterizado por situações similares ao modelo X-de-Y, e generaliza tal modelo. Adicionalmente, permite disjunção [13, 36].



Figura 2.1: Grafo de espera: modelo E.



Figura 2.3: Grafo de espera: modelo E-Ou.



Figura 2.2: Grafo de espera: modelo Ou.



Figura 2.4: Grafo de espera: modelo X-de-Y.



Figura 2.5: Grafo de espera: modelo X-de-Y disjuntivo.

2.3 Deadlocks

Informalmente, dizemos que um conjunto de processos S está em deadlock quanto todo $i \in S$ está à espera de que se realize alguma condição que somente pode se tornar ver-

dadeira por ação de um ou mais membros do próprio conjunto S. Classicamente, há três abordagens principais para o tratamento de deadlocks [52]:

- Prevenir a Ocorrência de Deadlocks: Consiste em identificar uma condição C tal que ∃ Deadlock → C. Uma vez identificada C (que é necessária para que ocorra deadlock), basta proibir a ocorrência de C para que sejam evitados os deadlocks.
- Evitar a Ocorrência de Deadlocks: Consiste em, ao haver novos pedidos por recursos, realizar uma simulação (em geral por algoritmo de bloqueio) para verificar o risco de que ocorra deadlock.
- Detectar a Ocorrência de Deadlocks: Consiste em identificar uma condição C tal que C → ∃ Deadkock. Uma vez identificada C (que é suficiente para que haja deadlock), basta verificar a ocorrência de C para detectar os deadlocks.

Das três abordagens, evitar a ocorrência de deadlocks é considerada uma estratégia não prática, demanda muito tempo e mensagens apenas para saber se uma requisição de recurso é segura (não gerará um deadlock). Assim, evitar a ocorrência de deadlocks e detectar deadlocks são as abordagens mais viáveis. Nesse trabalho abordaremos a estratégia de detecção de deadlocks.

Embora o princípio básico de deadlock independa de modelo, sua caracterização apresenta distinções de acordo com o modelo de espera do grafo em análise; sendo assim, apresentaremos as estruturas conhecidas na literatura que fornecem condições necessárias e suficientes para a existência de deadlock em um grafo de espera [7]:

- **Modelo E:** Um deadlock em um grafo G no modelo E existe se, e somente se, G contém um ciclo.
- Modelo Ou: Um deadlock em um grafo G no modelo Ou existe se, e somente se, G contém uma componente fortemente conexa C de cardinalidade |C| > 1 onde não há caminhos de um vértice de C para algum vértice de $G[V \setminus C]$, isto é, uma componente fortemente conexa sem saídas com pelo menos dois vértices, chamada *knot*. Note que não há caminhos de um knot para um sumidouro.
- Modelo E-Ou: Um deadlock em um grafo G no modelo E-Ou existe se, e somente se, G contém um *b-knot*: um subgrafo G' fortemente conexo tal que para cada vértice $v_i \in V(G')$, pelo menos um vértice de cada subconjunto de O_i também pertence a G'.

- **Modelo X-de-Y:** Um deadlock em um grafo G no modelo X-de-Y existe se, e somente se, G contém um (x - y)-knot: um subgrafo G' fortemente conexo tal que para cada vértice $v_i \in V(G')$, pelo menos $(y_i - x_i + 1)$ nós do conjunto O_i também pertencem a G'.
- Modelo X-de-Y Disjuntivo: Um deadlock em um grafo G no modelo X-de-Y disjuntivo existe se, e somente se, G contém um subgrafo G' fortemente conexo tal que, para cada vértice $v_i \in V(G')$, pelo menos $(y_i^k - x_i^k + 1)$ de cada subconjunto Q_i^k em O_i também pertence a G'.

Uma vez detectado o deadlock em uma computação distribuída, se faz necessária alguma ação adicional à detecção, geralmente alguma intervenção externa, para resolvê-lo (conhecido como *resolução* de deadlocks).

Observação: Os sumidouros em G são vértices que não dependem de nenhum vértice em G, isto é, estão prontos para executar suas tarefas. Assim é fácil ver que:

 \nexists sumidouros em $G \rightarrow \exists Deadlock.$

2.4 Grafos e/ou e Grafos xy

Neste trabalho utilizaremos grafos e, ou, e/ou, e xy, estruturas mais conhecidas na literatura [56, 57, 55], que chamaremos de *convencionais*. Grafos e/ou surgiram na década de 60 inseridos no domínio de inteligencia artificial [43, 51]. Desde então, têm sido aplicado com sucesso em diversas áreas [10, 14, 19, 24, 32, 33, 37, 38, 39, 42].

Um grafo e/ou, em princípio, consiste em um digrafo acíclico G que possui uma fonte e tal que cada vértice $v \in V(G)$ possui um rótulo $f(v) \in \{e, ou\}$. Em tais digrafos, arcos representam relações de dependência entre os vértices: um vértice rotulado 'e' depende de todos os seus vizinhos de saída (dependência conjuntiva), enquanto que um vértice rotulado 'ou' depende de apenas um vizinho de saída (dependência disjuntiva). Grafos epossuem apenas vértices com rótulos 'e', e grafos ou apenas vértices com rótulos 'ou'. Em grafos xy, cada vértice v_i possui um par de inteiros x_i - y_i que indicam que v_i depende de x_i de seus y_i vizinhos de saída. Embora para diversas aplicações são considerados apenas grafos acíclicos, para a nossa análise serão considerados grafos direcionados quaisquer.

Para a representação dos sistemas distribuídos são utilizados os grafos de espera. Discutiremos agora as transformações polinomiais dos grafos de espera em cada modelo para os grafos convencionais equivalentes. De fato, as transformações de grafos de espera nos modelos E, Ou e X-de-Y para grafos convencionais não alteram os tamanhos de instância. Os grafos de espera possuem em sua definição o conjunto O_i relacionado a um processo v_i , portanto, para a transformação, basta que v_i receba um rótulo correspondente ao modelo E, Ou, ou X-de-Y.

No caso do grafo de espera para o modelo E-Ou, para cada $O_i = \{O_i^1, O_i^2, \dots, O_i^q\}$, temos relações disjuntivas entre os subconjuntos, e conjuntivas entre os elementos de cada conjunto; portanto não é possível uma representação precisa utilizando apenas um rótulo.

Dado um grafo de espera W = (P, E) no modelo E-Ou, a transformação para grafo $e/ou \ G = (V, E)$ (Figura 2.6) é apresentada em [49] e obtida seguindo os passos abaixo.

- 1. Para cada processo p_i em P, criar um vértice v_i em V.
 - (a) Se $|O_i| > 1$, associar a v_i um rótulo $f(v_i) = ou$.
 - (b) Caso contrário, associar a v_i um rótulo $f(v_i) = e$ e adicionar arestas de v_i para o vértice que representa um processo em O_i .
- 2. Para cada conjunto $|O_i| > 1$, criar q vértices, $o_i^1, o_i^2, \ldots, o_i^q$ em V que chamaremos de artificiais, para cada subconjunto em $O_i = \{O_i^1, O_i^2, \ldots, O_i^q\}$.
 - (a) Associar a cada vértice v_i criado um rótulo $f(v_i) = e$.
 - (b) Criar arcos direcionados de v_i para todos os q vértices $o_i^1, o_i^2, \ldots, o_i^q \in V$.
 - (c) Para cada vértice artificial o_i^j , relacionado a um subconjunto O_i^j de tamanho k, criar k arcos direcionados de o_i^j para todos os vértices em V correspondentes aos vértices contidos em O_i^j .

Na figura 2.6, podemos destacar os vértices P_1 e P_4 , onde o vértice P_1 com apenas um subconjunto em O_i depende então da resposta de todos os seus vizinhos de saída para se tornar executável. O vértice P_4 possui dois subconjuntos com um vértice ($\{P_3\}$ e $\{P_5\}$), dessa forma, basta que apenas um desses conjuntos atenda P_4 para que este se torne executável.



Figura 2.6: Transformação de grafo de espera no modelo E-Ou para grafoe/ou.

Capítulo 3

Revisitando Modelos de Deadlock

No capítulo anterior, apresentamos os grafos de espera e os modelos de deadlock, onde através deles pode-se representar uma computação distribuída. A interação entre processos é determinada pelo modelo de deadlock; foram apresentados cinco principais modelos. Neste capitulo, definiremos formalmente regras que indicam que um modelo de deadlock A generaliza um modelo B dados critérios de eficiência bem definidos. Dessa forma, indicamos que a atual hierarquia é inadequada, e portanto é proposta uma reestruturação hierárquica dos modelos, onde duas novas hierarquias são apresentadas. Posteriormente, demonstramos que a existência de um subgrafo solução S, uma estrutura bem conhecida na literatura, garante que todo vértice pertencente a S não se encontra em deadlock. Além disso, definimos uma estrutura genérica que caracteriza deadlock em diversos modelos.

3.1 Hierarquia dos Modelos de Deadlock

Na literatura, os cinco modelos conhecidos de deadlock são geralmente classificados em uma hierarquia em que um modelo é dito generalizar o anterior quando contém como casos especiais todas as possíveis condições de espera do outro. Por exemplo, o modelo X-de-Y generaliza o modelo E, com $x_i = y_i$ e o modelo Ou com $x_i = 1$ para todos os $v_i \in V$. Além disso, o modelo E-Ou também generaliza o modelo E, fazendo $t_i = 1$ para todo $v_i \in V$, e o modelo Ou, fazendo $|O_i^1| = \cdots = |O_i^{t_i}| = 1$ (cada subconjunto em O_i possui um vizinho distinto de v_i). Apesar desta capacidade dos modelos X-de-Y e E-Ou de generalizarem os modelos E e Ou, eles não são considerados equivalentes um ao outro. Vários trabalhos afirmam que o modelo E-Ou é mais geral do que o modelo X-de-Y, enquanto que o inverso não é verdadeiro. Estes trabalhos assumem que o modelo E-Ou expressa qualquer condição do modelo X-de-Y: basta que para todo vértice $v_i \in V$ seja criado um conjunto O_i onde $|O_i^1| = \cdots = |O_i^{t_i}| = x_i$ (onde $t_i = \begin{pmatrix} y_i \\ x_i \end{pmatrix}$); em outras palavras, são criados subconjuntos de vértices contendo todas as possíveis combinações dos diferentes vértices de saída de v_i com tamanho x_i . Para concluir, observe que o modelo E-Ou e o modelo X-de-Y disjuntivo são considerados equivalentes entre si na literatura. A Figura 3.1 ilustra como os modelos de deadlock são classicamente organizados.



Figura 3.1: Organização clássica dos modelos de deadlock.

Como podemos observar, a transformação acima de uma instância do modelo X-de-Y para uma instância do modelo E-Ou demanda para cada vértice a criação de $t_i = \begin{pmatrix} y_i \\ x_i \end{pmatrix}$ conjuntos com x_i vértices. Tal transformação, apesar de computacionalmente possível, é inviável na prática.

Todas as transformações clássicas citadas até o momento estão completamente corretas. Porém, as transformações principalmente contemplam se qualquer instância de um modelo A pode ser transformada em uma instância de um modelo B mantendo as relações de dependência (não é considerado a viabilidade computacional). Portanto, visto que nem todas as transformações clássicas podem ser executadas em tempo polinomial, queremos reorganizar a hierarquia de modelos considerando a viabilidade prática das transformações, isto é, o tempo computacional de qualquer transformação de uma instância de um modelo em outro deve ser eficiente (polinomial). Dessa forma sentimos a necessidade de definir formalmente generalização entre modelos.

3.1.1 Poder de Expressão em Tempo Polinomial

Nesta subseção adicionamos mais formalismo para a análise da hierarquia dos modelos de deadlock. Começamos por observar que, se *eficiência no tempo de transformação* é incluída como um critério adicional a ser preenchido por um determinado modelo de deadlock para expressar as condições de espera de uma rede descrita em outro modelo, então

a hierarquia anteriormente proposta é realmente inadequada. Por isso, propomos uma reestruturação da hierarquia dos modelos de deadlock, a fim de satisfazer a propriedade de que cada modelo deve expressar as condições de espera de seus casos especiais em tempo polinomial. São necessários alguns conceitos e definições.

Poder de expressão. Dizemos que um modelo A que representa as regras de uma computação possui o mesmo poder de expressão de um segundo modelo B se, para qualquer instância do modelo B, é possível transformá-la em uma instância do modelo A de forma eficiente, assim como o inverso. Analogamente, um modelo A tem um poder estritamente maior de expressão que um modelo B se é possível transformar qualquer instância do modelo B em uma instância do modelo A de forma eficiente, porém não o inverso. **Tempo computacional eficiente.** Em geral, um algoritmo projetado para resolver um problema é considerado *eficiente* se tal algoritmo é executável em tempo polinomial, pois algoritmos não-polinomiais não são geralmente aplicáveis na prática, especialmente para grandes instâncias.

Ao analisar o poder de expressão de um modelo de deadlock e se ele contém como casos especiais todas as possíveis condições de espera de outro modelo, na verdade estamos verificando se existe um algoritmo de redução que recebe como entrada um grafo de espera G em que a computação funciona de acordo com um modelo de deadlock específico, e gera um grafo de espera H, equivalente em computação, que funciona de acordo com um outro modelo de deadlock. Logicamente, H deve conter todas as condições de espera expressas em G. Note que todas as relações transitivas de G devem ser representadas de forma compatível em H, i.e., se um processo v_i depende de um processo v_j e v_j depende de v_k , então v_i depende de v_k .

Tais reduções podem ser muito úteis na prática – é sempre interessante saber se é possível traduzir um grafo de espera construído a partir de um modelo de deadlock complexo em outro grafo de espera equivalente que opera em um modelo mais simples. No entanto, para tais reduções serem eficazes, este cálculo tem de ser realizado em tempo polinomial.

A partir deste momento, temos toda a base necessária para introduzir a nossa definição de redução entre os modelos de deadlock.

Definição 3.1. Redução entre Modelos de Deadlock – Dados dois modelos distintos de deadlock $A \in B$, B generaliza A em tempo polinomial se existe um algoritmo executável em tempo polinomial que tem como entrada um grafo de espera G = (V, E) cuja computação funciona de acordo com o modelo de deadlock A, e fornece como saída um grafo de espera G' = (V', E') cuja computação funciona de acordo com o modelo de deadlock B, de forma que:

- 1. Existe uma função injetora $f: V \to V'$;
- 2. O grafo de espera G' transitivamente contém todas as possíveis condições de G, isto
 é:
 - (a) Se, para que todos os processos em S₁ ⊆ V sejam liberados de sua condição de espera, é necessário/suficiente que todos os processos em S₂ ⊆ V sejam liberados, então, para que todos os processos em f(S₁) ⊆ V' sejam liberados de sua condição de espera, é necessário/suficiente que todos os processos em f(S₂) ⊆ V' sejam liberados, em algum instante.
 - (b) Reciprocamente, se para que todos os processos em $f(S_1)$ sejam liberados de sua condição de espera é necessário/suficiente que todos os processos em $f(S_2)$ sejam liberados de sua condição de espera, então, para que todos os processos em S_1 sejam liberados de sua condição de espera, é necessário/suficiente que todos os processos em S_2 sejam liberados, em algum instante.

A definição 3.1 assume que dependência entre processos é uma relação transitiva. Adicionalmente, ela permite que G' possua alguns vértices auxiliares (processos) que não existem em G; no entanto, o número de vértices auxiliares criados deve necessariamente ser polinomial. A possibilidade de serem utilizados vértices auxiliares leva em consideração que esses vértices são equivalentes à criação de *processos artificiais* no sistema.

Além disso, se um modelo de deadlock B generaliza em tempo polinomial o modelo de deadlock A, então o modelo B possui pelo menos o mesmo poder de expressão que o modelo A, e em tempo polinomial todas as condições de espera expressas no modelo Apodem ser expressas por B.

Lema 3.2. A transformação clássica do modelo X-de-Y para o modelo E-Ou não é realizada em tempo polinomial.

Demonstração. Na transformação clássica, o modelo E-Ou expressa uma condição geral do modelo X-de-Y da seguinte maneira. Para todo vértice $v_i \in V$, é criado um conjunto $O_i = \{O_i^1, O_i^2, \dots, O_i^{t_i}\}$, onde $|O_i^1| = \dots = |O_i^{t_i}| = x_i$ e $t_i = \binom{y_i}{x_i}$. Isto é, os subconjuntos contêm todas as possíveis combinações dos diferentes vértices de saída de v_i com tamanho x_i . No entanto, como tal transformação deve ser realizada para qualquer grafo de espera, se $x_i = O(n)$ e $y_i = O(n)$, onde n = |V(G)|, o tempo de execução da transformação é $O(n^n)$.



Figura 3.2: Exemplo de uma transformação clássica de uma instância no modelo X-de-Y para o modelo E-Ou.

A Figura 3.2 ilustra a transformação de uma instância no modelo X-de-Y para o modelo E-Ou a partir de um vértice v_1 . Note que para instâncias onde os valores de x_i e y_i são altos (por exemplo $x_i = \frac{n}{2}$ e $y_i = n$) a transformação se torna exponencial.

Agora, inspirados pela representação de Ryang [49], definiremos um novo modelo de deadlock baseado no modelo E-Ou, como segue.

Definição 3.3. Modelo E-Ou simplificado – Existem dois tipos de processos, E e Ou. Um processo E torna-se sumidouro apenas quando seu estado de espera é liberado por todos os processos em O_i , enquanto que, para um processo v_j do tipo Ou se tornar sumidouro, basta que sua condição de espera seja atendida por apenas um processo de O_j .

Neste momento, temos 6 diferentes modelos de deadlock. Através de reduções de tempo polinomial entre os modelos, propomos uma reestruturação da hierarquia dos modelos de deadlock em função do poder de expressão em tempo polinomial.

Lema 3.4.

- 1. O modelo Ou não generaliza o modelo E.
- 2. O modelo E não generaliza o modelo Ou.
- 3. O modelo E-Ou simplificado generaliza em tempo polinomial os modelos E e Ou.
- 4. O modelo E-Ou generaliza em tempo polinomial o modelo E-Ou simplificado.

Demonstração. (1) Para mostrar que o modelo Ou não generaliza o modelo E, simplesmente considere uma árvore binária completa enraizada T com cardinalidade maior que um. Uma condição necessária (mas não suficiente) para que a raiz seja liberada de sua espera é que uma folha envie resposta ao seu pai. Uma condição necessária e suficiente seria que todas as folhas enviem mensagens para os seus pais. Tais condições de espera não podem ser expressas pelo modelo Ou, porque se um caminho direcionado entre dois vértices $f(v_i)$ e $f(v_j)$ é criado, onde v_i e v_j são folhas de T, isto iria representar uma condição de espera que não existe em T; e se tais caminhos não existem, então existem caminhos de nós Ou da raiz até $f(v_i)$ que não passam por nenhum vértice $f(v_j)$ (onde v_j é folha). Portanto, o envio de uma mensagem de $f(v_i)$ para o seu pai é suficiente (mas não necessário) para liberar a raiz, o que não ocorre no modelo E.

(2) Reciprocamente, o modelo E não generaliza o modelo Ou. Assuma que T é construída sob o modelo Ou. Novamente, se um caminho direcionado entre $f(v_i)$ e $f(v_j)$ é criado, onde v_i e v_j são folhas de T, então seria representada uma condição de espera que não existe em T; e se tais caminhos não existem, então o envio de uma mensagem de concessão partindo de $f(v_i)$ para seus vizinhos de entrada é necessário (mas não suficiente) para que a raiz seja liberada em algum momento, o que não ocorre no modelo Ou.

(3) É fácil verificar que o modelo E-Ou simplificado generaliza em tempo polinomial os modelos E e Ou, já que correspondem a casos especiais do modelo E-Ou simplificado onde todos os vértices são rotulados com apenas um tipo, E ou Ou.

(4) O modelo E-Ou simplificado é um caso especial do modelo E-Ou, já que, se no máximo um dentre $O_i^1, \dots, O_i^{t_i}$ não é vazio para algum processo $v_i \in V$ então v_i é equivalente a um vértice do tipo E. Similarmente, se todos os conjuntos não vazios em $O_i^1, \dots, O_i^{t_i}$ possuem apenas um filho para algum $v_i \in V$ então v_i é equivalente a um vértice do tipo Ou.

Note que apesar do modelo E não generalizar o modelo Ou e vice-versa, existem situações onde ambos coincidem, como em casos de grafos de espera com grau máximo de saída um.

O resultado seguinte demonstra que o modelo E-Ou pode facilmente ser simplificado para um modelo de manipulação mais fácil.

Lema 3.5. O modelo E-Ou simplificado generaliza em tempo polinomial o modelo E-Ou.

Demonstração. Dado um grafo de espera G = (V, E) no modelo E-Ou, criaremos um grafo

de espera equivalente G' = (V', E') no modelo E-Ou simplificado. A redução é mostrada a seguir:

- a) Defina $V' = V, E' = \emptyset$, e atribua rotulo Ou a todos os vértices em V'.
- b) Para cada subconjunto O_i^j associado a um vértice v_i em G, crie um vértice auxiliar w_i^j com rótulo E em V'.
- c) Para cada processo $v_i \in V' \cap V$, adicione um arco a E' do vértice v_i ao vértice w_i^j , $1 \leq j \leq t_i$.
- d) Para cada vértice w_i^j , adicione a E' arcos de w_i^j a cada vértice em $O_i^j \cap V'$.

Na construção acima, informalmente, cada conjunto O_i^j encapsulado em um vértice v_i é substituído por um vértice auxiliar w_i^j que opera de forma similar. Portanto, todas as condições de espera são preservadas em G'.



Figura 3.3: Exemplo de transformação de uma instância no modelo E-Ou para o modelo E-Ou simplificado.

A Figura 3.3 ilustra a transformação de uma instância no modelo E-Ou para o modelo E-Ou simplificado a partir de um vértice v_1 . Note que mesmo se os subconjuntos em O_i possuírem intersecção não vazia, a transformação continua executável em tempo polinomial.

A seguir, trataremos do modelo X-de-Y e do modelo X-de-Y disjuntivo.

Lema 3.6.

- 1. O modelo X-de-Y generaliza em tempo polinomial o modelo E-Ou simplificado.
- 2. O modelo X-de-Y disjuntivo generaliza em tempo polinomial o modelo X-de-Y.
- 3. O modelo X-de-Y generaliza em tempo polinomial o modelo X-de-Y disjuntivo.

Demonstração. (1) No modelo X-de-Y, um vértice v_i do tipo E é equivalente a um vértice com $x_i = y_i$, e um vértice Ou é equivalente a um vértice com $x_i = 1$.

(2) O modelo X-de-Y é um caso especial do modelo X-de-Y disjuntivo, onde para cada vértice v_i existe apenas um subconjunto não vazio Q_i^j .

(3) Similarmente ao Lema 3.5, o modelo X-de-Y generaliza o modelo X-de-Y disjuntivo ao criar vértices auxiliares com rótulo (x_i^j, y_i^j) para cada subconjunto Q_i^j .

Na tabela 3.1 é apresentado um comparativo entre o poder de expressão dos modelos de deadlock discutidos até o momento. Sabemos que o modelo E não generaliza o modelo Ou e vice-versa; entretanto, não são completamente disjuntos, já que equivalem em instâncias que possuem apenas vértices com grau de saída no máximo um. Os modelos E-Ou e E-Ou e E-Ou simplificado generalizam os modelos E e Ou. Os modelos E-Ou e E-Ou e E-Ou simplificado generalizam os modelos E e Ou. Os modelos X-de-Y e X-de-Y disjuntivo possuem o mesmo poder de expressão. Finalmente, os modelos E-Ou e E-Ou simplificado são generalizados pelo modelo X-de-Y.

Generalização de Modelos de Deadlock			
Е	¢	Ou	
Ou	É	E	
Ε	\subset	E-Ou Simplificado	
Ou	\subset	E-Ou Simplificado	
E-Ou Simplificado	\subseteq	E-Ou	
E-Ou	\subseteq	E-Ou Simplificado	
E-Ou	\subseteq	X-de-Y	
E-Ou Simplificado	\subseteq	X-de-Y	
X-de-Y	\subseteq	X-de-Y Disjuntivo	
X-de-Y Disjuntivo	\subseteq	X-de-Y	

Tabela 3.1: Poder de expressão entre os modelos de deadlock. O símbolo \subseteq significa "é generalizado por".

Finalmente, podemos focar nossa atenção na seguinte indagação:

(*) "O modelo E-Ou generaliza em tempo polinomial o modelo X-de-Y?"

De acordo com o Lema 3.2, uma transformação bem conhecida do modelo X-de-Y para o modelo E-Ou demanda tempo exponencial no pior caso. Por isso, para responder a questão (*), introduziremos alguns conceitos de complexidade de circuitos.

3.1.1.1 Complexidade de Circuitos

De acordo com [61], um *circuito booleano* com n bits de entrada é um grafo direcionado acíclico onde todo vértice (usualmente chamado de *porta* neste contexto) pode ser: (a) um vértice de entrada, cujo grau de entrada é 0, representado por um dos n bits de entrada; (b) uma porta AND; (c) uma porta OR; (d) uma porta NOT. Uma dessas portas é designada como porta de saída do circuito. Note que tal circuito computa uma função de n entradas. Apesar de classicamente definidos como grafos direcionados acíclicos, para os propósitos deste trabalho, os circuitos booleanos não serão restringidos a grafos acíclicos, consideraremos um circuito booleano como um grafo direcionado qualquer.

O tamanho de um circuito é o número de portas que ele possui, e sua profundidade é dada pelo caminho máximo entre uma entrada e a porta de saída. *Circuitos booleanos* monótonos são circuitos que possuem apenas portas do tipo AND e OR.

Observação 3.7. Podemos interpretar um circuito booleano monótono como um sistema distribuído que opera de acordo com o modelo E-Ou simplificado, onde os vértices de entrada (no circuito) são sumidouros no grafo de espera. Uma saída '1' de um vértice do circuito é equivalente ao envio de uma mensagem de concessão do processo associado.

Tal como definido em [53], existem dois principais conceitos em complexidade de circuitos: (i) complexidade de tamanho do circuito de uma função booleana F, que consiste no tamanho mínimo de qualquer circuito que compute F; (ii) complexidade da profundidade de uma função booleana F, que é a profundidade mínima de qualquer circuito que compute F.

Outros conceitos importantes ao estudo de complexidade de circuitos são *portas th*reshold, funções threshold e circuitos threshold.

Uma porta threshold g com n entradas pode ser descrita por uma porta que consiste em uma sequência w_1, \dots, w_t de números reais chamados de *pesos* e um limiar t. As portas threshold computam as *funções threshold*, definidas da seguinte forma:

$$f_t(x_1, \cdots, x_n) = \begin{cases} 1, & \text{se} \quad \sum_1^n w_i x_i \ge t; \\ 0, & \text{caso contrário.} \end{cases}$$

Quando $t = \lfloor n/2 \rfloor + 1$ e os pesos são unitários, g é chamada de *porta majoritária*, e f_t função majoritária. Um circuito threshold é um circuito que contém portas threshold.

Observação 3.8. Podemos interpretar um circuito threshold monótono como um sistema distribuído que opera de acordo com o modelo X-de-Y. Por outro lado, a condição de espera de um processo em tal modelo é, de fato, uma função threshold.

Já que circuitos threshold monótonos podem ser vistos como um sistema distribuído que opera de acordo com o modelo X-de-Y, basta verificarmos se tais circuitos podem ser transformados em circuitos booleanos monótonos, ou seja, circuitos que operam de acordo com o modelo E-Ou simplificado. Transformações deste tipo são conhecidas na literatura, como poderemos observar adiante.

Circuitos de Redes de Ordenação

Um circuito de rede de comparação é um circuito tal que para quaisquer n bits de entrada, as saídas são monotonamente ordenadas (utilizando apenas portas AND e OR). Tais circuitos são representados por grafos direcionados acíclicos, possuem n entradas (bits a serem ordenados) e n saídas (bits ordenados). A ordenação é feita por um comparador (figura 3.4) que possui duas entradas e duas saídas (duas arestas de entrada e duas arestas de saída). A profundidade da rede é dada pelo maior caminho entre as entradas e as saídas da rede.

A Figura 3.4 (a) apresenta uma porta comparadora, com dois bits de entrada ($x \in y$) e duas saídas, uma com o valor mínimo entre ambos, e outra com o valor máximo. A Figura 3.4 (b) é a representação em nível de portas lógicas do comparador, que mostra que podemos fazer essa comparação utilizando apenas duas portas logicas monótonas (uma porta AND que fornece o menor valor entre os bits comparados, e uma porta OR que fornece o maior).



Figura 3.4: Comparadores de bits: (a) porta comparadora de bits; (b) comparador utilizando as portas lógicas monótonas.

Na Figura 3.5 são apresentados dois circuitos equivalentes, que ilustram a implementação em circuitos de redes de comparação do método da bolha. A Figura 3.5 (a) mostra um circuito de ordenação com cinco bits de entrada utilizando porta comparadora de bits. A Figura 3.5 (b) mostra um circuito monótono de ordenação com cinco bits de entrada utilizando portas lógicas monótonas.



Figura 3.5: Circuitos de redes de ordenação: (a) circuito com porta comparadora de bits; (b) rede de ordenação utilizando as portas lógicas monótonas.

Lema 3.9. Um circuito booleano monótono que computa uma função threshold f com n variáveis pode ser construído em tempo polinomial.

Demonstração. Para se construir tal circuito, basta primeiramente construir um circuito de rede de ordenação para as n entradas, e em seguida verificar a saída que se encontra no limiar t.

Lema 3.10. [1, 30, 46, 60] Uma função threshold f com n variáveis pode ser computada por um circuito booleano monótono de tamanho polinomial e profundidade $O(\log n)$.

Neste momento, temos todo o embasamento necessário para responder a questão (*).

Teorema 3.11. O modelo E-Ou generaliza em tempo polinomial o modelo X-de-Y.

Demonstração. Um dado grafo de espera G no modelo X-de-Y pode ser transformado em tempo polinomial em um grafo G' no modelo E-Ou simplificado, substituindo cada processo p de G, por uma sub-rede (com processos artificiais) associada a um circuito booleano monótono de tamanho polinomial que computa a função threshold equivalente às condições de espera do processo p. Ou seja, cada processo p_i possui uma condição de espera $x_i - y_i$ que equivale a uma função threshlhold associada, que pode ser transformada em um circuito booleano. Podemos utilizar a rede análoga a este circuito para substituir o processo com rótulo $x_i - y_i$ por uma subrede contendo apenas nós E e Ou (portas booleanas monótonas). Tal circuito booleano monótono pode ser obtido construindo, por exemplo, um circuito (auxiliar) de rede de ordenação como o do método da bolha, o qual possui profundidade $O(n^2)$; alternativamente, há duas formas radicalmente diferentes de se construir tais circuitos com profundidade $O(\log n)$: a primeira usa uma construção de circuitos de ordenação descrita em [1, 46], e a segunda utiliza métodos probabilísticos [30, 60].

O Teorema 3.11 encerra o primeiro nível de nossa análise sobre o poder de expressão em tempo polinomial na generalização de modelos de deadlock. A Figura 3.6 ilustra a hierarquia dos modelos de deadlock sugerida a partir da nossa análise. Como pode-se verificar, os modelos E-Ou e X-de-Y de fato possuem o mesmo poder de expressão, mesmo levando em consideração tempo e tamanho polinomial como restrições.



Figura 3.6: Hierarquia dos modelos de deadlock de acordo com seu poder de expressão em tempo polinomial.

Para finalizar a análise, os circuitos construídos em [1, 30, 46, 60] para uma função threshold com n variáveis possuem profundidade $O(\log n)$. Utilizando tais construções em um sistema distribuído como descrito no Teorema 3.11, amplia-se por um fator $O(\log n)$ o comprimento dos caminhos da rede, o que implica no fato de que algumas mensagens que levavam tempo O(1) para serem recebidas na rede original agora demandarão tempo $O(\log n)$ para serem recebidas na rede resultante.
A esta altura, o leitor pode então perguntar se é possível realizar tal transformação em tempo polinomial mantendo a ordem de profundidade da rede original. Infelizmente, a resposta a esta questão é negativa, uma vez que é bem conhecido na literatura (veja Lema 3.13) que algumas funções threshold, como a função majoritária, não podem ser expressas por circuitos booleanos com profundidade constante e tamanho polinomial.

Definição 3.12. [2] A classe AC^0 consiste dos circuitos de profundidade O(1) e tamanho polinomial, onde as portas lógicas AND e OR podem possuir número irrestrito de entradas.

Lema 3.13. [48, 54, 26] Uma função majoritária de n bits não pode ser computada por um circuito em AC^0 de tamanho subexponencial.

Assim, em nossa análise sobre o poder de expressão de um modelo de deadlock, se além da computação em tempo polinomial também adicionarmos preservação da ordem de profundidade da rede como um critério requerido, obtemos a hierarquia ilustrada na Figura 3.7, que é bastante diferente da atual organização de modelos de deadlock na literatura.



Figura 3.7: Hierarquia dos modelos de deadlock de acordo com seu poder de expressão em tempo polinomial, preservando a ordem de profundidade.

3.2 Caracterizações por Estruturas de Dados

Nesta seção iremos utilizar uma estrutura de dados bem conhecida, os grafos e/ou, como ferramenta para caracterizar de forma precisa deadlocks no modelo E-Ou simplificado. Grafos de espera no modelo E-Ou simplificado são, de fato, grafos e/ou que permitem ciclos e não possuem restrição quanto à quantidade de nós fonte.

Definição 3.14. Dado um grafo $e/ou \ G = (V, E) \ e$ um vértice $s \in V$, um subgrafo solução enraizado em s é um subgrafo acíclico H = (V', E') que satisfaz as seguintes propriedades:

1. $s \in V'$.

- 2. Para cada vértice não sumidouro v_i em V':
 - (a) Se v_i é um vértice com rótulo 'e' então cada arco de saída de v_i pertence a E'.
 - (b) Se v_i é um vértice com rótulo 'ou' então exatamente um arco de saída pertence a E'.

Lema 3.15. Seja s um vértice de um grafo de espera G = (V, E) no modelo E-Ou simplificado. Se G possui um subgrafo solução enraizado em s então s não está em deadlock.

Demonstração. A relação de dependência de um vértice em um grafo e/ou é equivalente à condição de espera de um vértice no modelo E-Ou simplificado, i.e., às regras que descrevem como um vértice se torna sumidouro. A partir da Definição 3.15, um subgrafo solução indica uma forma de s ser liberado de sua condição de espera, iniciada por vértices sumidouros. Consequentemente, torna-se claro que vértices em um subgrafo solução ao final receberão mensagens de concessão suficientes, tornando-se portanto aptos a realizar suas tarefas.



Figura 3.8: Exemplo de subgrafo solução enraizado em v_1 .

A Figura 3.8 apresenta um grafo G de espera no modelo E-Ou simplificado (possui apenas vértices de tipo E e Ou). Destacado em azul, tem-se um exemplo de subgrafo solução enraizado em v_1 .

Agora, descreveremos uma estrutura em grafos e/ou que provê uma condição necessária e suficiente para a existência de deadlock.

Definição 3.16. Dado um grafo $e/ou \ G = (V, E) \ e$ um vértice $s \in V$, um subgrafo não-solução de G é um subgrafo fortemente conexo H = (V', E') que satisfaz as seguintes propriedades:

- 1. $s \in V'$.
- 2. Para cada vértice v_i em V':
 - (a) Se v_i é um vértice com rótulo 'e' então exatamente um vértice de O_i pertence a V'.
 - (b) Se v_i é um vértice com rótulo 'ou' então todos os vértices de O_i pertencem a V'.

A definição de subgrafo não-solução leva à caracterização de deadlocks no grafo de espera do modelo E-Ou simplificado.

Teorema 3.17. Um grafo de espera G no modelo E-Ou simplificado contém vértices em deadlock se, e somente se, G possui um subgrafo não-solução.

Demonstração. Seja S um conjunto de vértices em um subgrafo não-solução de um grafo de espera. Todo vértice em S está esperando por mensagens de outro vértice em S. Porém, para qualquer vértice v_i com rótulo 'ou', já que $O_i \subseteq S$, v_i esperará indefinidamente por uma mensagem de concessão. Portanto, S é um conjunto de vértices em deadlock.

Seja D um conjunto minimal de vértices que caracterizam um deadlock. Por definição, todos os vértices em D estão bloqueados, esperando por mensagens de concessão de outros vértices em D. Portanto, todos os vértices com rotulo 'e' possuem pelo menos um de seus vizinhos de saída em D. Para um vértice com rótulo 'ou', todos os seus vizinhos de saída estão em D. Como D é minimal, não existe um par de vértices u, w em D tais que u não possua caminho direcionado para w e vice-versa, caso contrário, pelo menos um deles, u ou w, poderia ser removido de D de tal forma que alguns vértices de D ainda caracterizariam o deadlock, contradizendo a minimalidade de D. Por outro lado, se D contém um vértice u que não é alcançável por todos os vértices em D, D-u ainda conteria vértices que caracterizariam o deadlock, caso contrário, todos os vértices em D-u seriam alcançáveis por u, contradizendo a minimalidade de D. Portando, pode-se concluir que D induz um subgrafo fortemente conexo em G.



Figura 3.9: Exemplo de subgrafo não-solução.

A Figura 3.8 apresenta um grafo G de espera no modelo E-Ou simplificado (possui apenas vértices de tipo E e Ou). Destacado em vermelho, tem-se um exemplo de subgrafo não-solução.

3.2.1 O Modelo X-de-Y

Claramente, grafos xy generalizam os grafos e/ou [57]. Tais grafos são equivalentes aos grafos de espera do modelo X-de-Y: cada nó v_i de um grafo xy possui como rótulo x_i - y_i , significando que v_i depende de x_i de seus y_i vizinhos de saída.

Definição 3.18. Dado um grafo G = (V, E) e um vértice $s \in V$, um subgrafo solução enraizado em s de G é um subgrafo acíclico H = (V', E') que satisfaz as seguintes propriedades:

- 1. $s \in V'$.
- Para cada vértice não sumidouro v_i em V', exatamente x_i de seus y_i arcos de saída pertencem a E'.

Definição 3.19. Dado um grafo xy G = (V, E) e um vértice $s \in V$, um subgrafo nãosolução de G é um subgrafo fortemente conexo H = (V', E') que satisfaz as seguintes propriedades:

- 1. $s \in V'$.
- 2. Para cada vértice não sumidouro v_i em V', pelo menos $y_i x_i + 1$ de seus y_i arcos de saída pertencem a E'.

Corolário 3.20. Dado um grafo de espera G no modelo X-de-Y, G possui vértices em deadlock se e somente se G possui um subgrafo não-solução.

Demonstração. Seja S um conjunto de vértices em um subgrafo não-solução de um grafo de espera G no modelo X-de-Y. Todo vértice em S está esperando por mensagens de outro vértice em S. Porém, para qualquer vértice com rótulo $x_i - y_i$, pelo menos $y_i - x_i + 1$ de seus vizinhos de saída pertencem a S. Logo, v_i esperará indefinidamente por uma mensagem de concessão. Portanto, S é um conjunto de vértices em deadlock.

Seja D um conjunto minimal de vértices que caracterizam um deadlock. Por definição, todos os vértices em D estão bloqueados, esperando por mensagens de concessão de outros vértices em D. Portanto, todo vértice em D possui $y_i - x_i + 1$ de seus y_i vizinhos de saída em D. Como D é minimal, não existe um par de vértices u, w em D tais que u não possua caminho direcionado a w e vice-versa, caso contrário, pelo menos um dentre u ou w poderia ser removido de D de tal forma que alguns vértices de D ainda caracterizariam o deadlock, contradizendo a minimalidade de D.

Por outro lado, se D contem um vértice u que não é alcançável por todos os vértices em D, D-u ainda conteria vértices que caracterizariam o deadlock, caso contrário, todos os vértices em D-u seriam alcançáveis por u, contradizendo a minimalidade de D. Portando, pode-se concluir que D induz um subgrafo fortemente conexo em G.

3.3 Considerações Finais

Neste capítulo foi desenvolvido um estudo completo a respeito de modelos de deadlock e sua hierarquia. Apresentamos a hierarquia clássica da literatura, onde apontamos que a transformação clássica conhecida de uma instância do modelo X-de-Y para o modelo E-Ou é inviável. Introduzimos uma definição formal e apropriada para avaliar quando um modelo de deadlock generaliza outro, onde é visada a viabilidade computacional na transformação de instâncias entre diferentes modelos de deadlock. Também é proposto um novo modelo de deadlock, o modelo E-Ou simplificado, como um modelo alternativo ao modelo E-Ou clássico. Assim, duas novas hierarquias foram apresentadas.

Finalmente, apontamos condições necessárias e suficientes para a existência de deadlock em grafos de espera em termos de subgrafos especiais.

Capítulo 4

Problemas sobre Resolução de Deadlock

Neste capítulo, trataremos da complexidade computacional para a resolução de deadlocks. Consideramos que os deadlocks ou estruturas que o caracterizam são dados por algum algoritmo de detecção (tais algoritmos são conhecidos na literatura). Temos então o interesse em saber o quão eficientemente pode-se resolver um deadlock, isto é, dada alguma intervenção a ser feita no sistema em deadlock, fazê-la com o menor impacto possível. Definiremos então a classe de problemas Deadlock-Resolution(a, b) como notação para os problemas de resolução de deadlock. A notação Deadlock-Resolution(a, b) possui dois parâmetros: o primeiro, a, indica a qual modelo de deadlock o grafo de entrada pertence; o segundo, b, indica o tipo de intervenção externa a ser utilizada. Posteriormente, são apresentadas as complexidades de todos os problemas aqui levantados, separados em seções por modelo de deadlock.

4.1 A Classe de Problemas Deadlock-Resolution(a, b)

Como deadlock é uma propriedade estável em sistemas distribuídos, uma vez que ele ocorre devem ser feitas intervenções para solucioná-lo. A complexidade de cada problema combinatório varia de acordo com o modelo e a intervenção externa. Os tipos de intervenção externa abordados neste trabalho são dadas a seguir:

- Arco: A intervenção é dada por remoção de arcos. Para um dado grafo G, devemos encontrar o número mínimo de arcos a serem removidos de G de forma a torná-lo livre de deadlock. A remoção de um arco em um sistema distribuído equivale a desfazer uma requisição.
- 2. Vértice: A intervenção é dada por remoção de vértices. Para um dado grafo G,

devemos encontrar o número mínimo de vértices a serem removidos de G de forma a torná-lo livre de deadlock. A remoção de um vértice em um sistema distribuído equivale a eliminar um processo.

3. Sumidouro: A intervenção é dada por transformar vértices em processos executáveis, i.e., para um determinado vértice v, devemos remover todos os arcos de saída, transformando-o em sumidouro. Para um dado grafo G, devemos encontrar o menor número de vértices a serem transformados em sumidouros de forma a tornar G livre de deadlock.

De forma mais precisa, o parâmetro a se refere aos modelos de deadlock (E, Ou, E-Ou e X-de-Y), e o parâmetro b se refere ao tipo de intervenção externa (Arco, Vértice e Sumidouro). Estamos interessados em descobrir a complexidade computacional de todos os doze problemas combinatórios resultantes, como pode ser visto na Tabela 6.1.

Deadlock-Resolution(a, b)						
	Е	Ou	E-Ou	X-de-Y		
Arco	?	?	?	?		
Vértice	?	?	?	?		
Sumidouro	?	?	?	?		

Tabela 4.1: Complexidades em aberto de problemas Deadlock-Resolution(a, b).

4.2 Modelo E

Para determinar se há deadlock em um grafo G no modelo E, é necessária e suficiente a existência de ciclos. Caso exista algum ciclo em um dado grafo G, podemos determinar a existência de deadlock em G em tempo linear uma vez que, utilizando uma busca em profundidade, procedimento linear, podemos determinar a existência de ciclos em qualquer tipo de grafo. Ao saber com precisão a exata estrutura que caracteriza deadlock no modelo E, é possível reescrever os três problemas Deadlock-Resolution(E,b) de forma mais específica:

1. Deadlock-Resolution(E, Arco): dado um grafo G = (V, E) no modelo E, encontrar o número mínimo de arcos formando um conjunto E' tal que $G[E \setminus E']$ seja acíclico, isto é, um grafo livre de deadlock.

- Deadlock-Resolution(E, Vértice): dado um grafo G = (V, E) no modelo E, encontrar o número mínimo de vértices formando um conjunto S tal que G[V\S] seja acíclico, isto é, um grafo livre de deadlock.
- 3. Deadlock-Resolution(E, Sumidouro): dado um grafo G = (V, E) no modelo E, encontrar o número mínimo de vértices formando um conjunto S tal que ao resolver todos os vértices de S (transformá-los em sumidouros, tornando-os executáveis), o grafo resultante $G[V \setminus S]$ seja acíclico, isto é, um grafo livre de deadlock.

Neste ponto, temos todas as ferramentas para obter a complexidade computacional de cada problema, e é fácil verificar que o seguinte teorema é correto:

Teorema 4.1.

- (a) Deadlock-Resolution(E, Arco) é NP-Difícil.
- (b) Deadlock-Resolution(E, Vértice) é NP-Difícil.
- (c) Deadlock-Resolution(E, Sumidouro) é NP-Difícil.

Demonstração. (a) Dado um grafo G = (V, E) no modelo E, encontrar o menor número de arestas a serem removidas para o tornar livre de deadlock corresponde a escolher um conjunto mínimo E' de arestas de forma que $G[E \setminus E']$ seja acíclico. Tal problema é exatamente equivalente ao Directed Feedback Arc Set Problem, problema bem conhecido na literatura e provado NP-Difícil em [34].

(b) Dado um grafo G = (V, E) no modelo E, encontrar o menor número de vértices a serem removidos para o tornar livre de deadlock corresponde a escolher um conjunto mínimo S de vértices tal que $G[V \setminus S]$ seja acíclico. Tal problema é exatamente equivalente ao *Directed Feedback Vertex Set Problem*, problema bem conhecido na literatura e também provado NP-Difícil em [34].

(c) Provaremos que Direct Feedback Vertex Set \propto Deadlock-Resolution(E, Sumidouro), o que mostra que Deadlock-Resolution(E, Sumidouro) é NP-Difícil. Seja G = (V, E) uma instância de Direct Feedback Vertex Set. Podemos assumir que G é um grafo no modelo E. Seja G' uma instância do Deadlock-Resolution(E, Sumidouro) tal que G' = G. Vamos mostrar que G' possui k vértices em um conjunto S cuja resolução torna G' livre de deadlock se, e somente se, $G[V \setminus S]$ é acíclico.

Suponha que G = (V, E) possua um conjunto S de forma que $G[V \setminus S]$ seja acíclico. Como G = G', podemos então considerar que removendo em G' todos os vértices do conjunto S, teremos que $G'[V' \setminus S]$ é um grafo acíclico. Finalmente, se reinserirmos em $G'[V' \setminus S]$ todos os vértices de S, porém sem suas arestas de saída, teremos inserido sumidouros. Como sumidouros não possuem qualquer aresta de saída, nunca fazem parte de ciclos; portanto, ao converter diretamente em G' os vértices de S em sumidouros, o grafo resultante permanecerá acíclico pois não há arcos de "feedback" entre os vértices de S e vértices de $V' \setminus S$, isto é, o grafo fica livre de deadlock.

Por outro lado, suponha que G' possua um conjunto S de vértices tais que, se transformados em executáveis removendo seus arcos de saída, o grafo resultante é livre de deadlock. Como os vértices resolvidos em G' se tornam sumidouros, eles não proveem ao grafo nenhum arco de retorno (arco que "feche" um ciclo); consequentemente, não participam de nenhum ciclo, e portanto, se removidos, o grafo resultante $G[V \setminus S]$ também será acíclico.

4.3 Modelo Ou

Para determinar se há deadlock em um grafo G no modelo Ou, é necessária e suficiente a existência de um knot, uma componente fortemente conexa sem saídas com cardinalidade dois ou maior. Na literatura existem diversos algoritmos de tempo linear para encontrar Componentes Fortemente Conexas (CFCs) em um grafo direcionado. Cada componente fortemente conexa encontrada pode ser contraída a um único vértice, formando assim um digrafo acíclico D onde sumidouros são knots. Veja a Figura 4.1.



Figura 4.1: Grafo e árvore obtida pela contração de CFCs.

Os knots finalmente podem ser encontrados em tempo linear pelo algoritmo 1. Tal algoritmo utiliza como base o algoritmo de [20], que utiliza busca em profundidade para encontrar as CFCs de um grafo direcionado G. É feita uma contração dos vértices obtendo um digrafo acíclico D, e finalmente forma-se uma lista de vértices que correspondem aos knots, analisando se são sumidouros em D. Podemos assim detectar knots de uma computação distribuída peloo algoritmo a seguir.

Algoritmo 1: Detecção de knots
Entre de
Entrada:
G: Grafo Ou
Saída:
S: Lista de knots
1 início
2 Encontrar as CFC's de G via busca em profundidade
3 Computar D
4 para cada $v \in D$ faça
5 se $ v > 1$ e v é sumidouro então
6 incluir $v \in S$
7 fim se
8 fim para cada
9 retorna S
10 fim

Sabemos agora como encontrar a estrutura minimal que caracteriza o deadlock em um grafo G conexo no modelo Ou. Portanto, mesmo que o grafo não seja conexo, basta aplicar o algoritmo a cada componente conexa do grafo.

Agora, queremos livrar um dado grafo G de quaisquer knots através de remoção de arcos. Para isso, os lemas a seguir serão úteis.

Lema 4.2. Considere H um knot em um grafo G no modelo Ou. O menor número de arestas a serem removidas de H para torná-lo livre de deadlock é igual ao menor grau de saída de H.

Demonstração. Seja H um knot. H deixará de estar em deadlock se, e somente se, para todo vértice v de H houver um caminho direcionado a um sumidouro. Um sumidouro pode ser obtido em H removendo-se de um vértice u todas os seus arcos de saída. O menor número de remoções de arcos para converter um vértice $v \in V(H)$ em sumidouro é igual ao menor grau de saída do subgrafo H. Como H inicialmente é uma componente fortemente conexa, todo vértice de H possui caminho direcionado para v; após a remoção das arestas de saída de v, os caminhos direcionados a v permanecerão inalterados, tornando H livre de deadlock.

Lema 4.3. Seja G um grafo no modelo Ou, e suponha que G está em deadlock. Então, o menor número de arestas a serem removidas de G para torná-lo livre de deadlock é igual à soma dos menores graus de saída nos knots de G.

Demonstração. Consideraremos sem perda de generalidade que o grafo G é conexo. Ao colapsar cada componente fortemente conexa de G em um vértice, obteremos um digrafo acíclico onde os knots originais se converteram em sumidouros. Portanto, resolvendo o deadlock de cada sumidouro, o sistema estará livre de deadlock. Aplicando o Lema 4.2 a cada knot, teremos um sistema livre de deadlock.

Através dos Lemas 4.2 e 4.3, podemos remover todos os knots de um grafo no modelo Ou por meio de remoções de arcos com o algoritmo de tempo polinomial apresentado a seguir (Algoritmo 2).

Algoritmo 2: Deadlock-Resolution(Ou, Arco)
Entrada:
G: Grafo Ou
Saída:
H: Grafo livre de deadlock
1 início
$2 H \leftarrow G$
$S \leftarrow$ lista de knots utilizando o algoritmo de detecção de knots
4 para cada knot em S faça
5 Encontre o vértice v com menor grau de saída em S
6 Remova todos os arcos de saída do vértice $v \in H$
7 fim para cada
s retorne H
9 fim

Como consequência dos Lemas 4.2 e 4.3 e do Algoritmo 2, podemos enunciar:

Teorema 4.4. Deadlock-Resolution(Ou, Arco) pode ser resolvido em tempo polinomial.

Agora trataremos do problema Deadlock-Resolution(Ou, Sumidouro).

Lema 4.5. O número necessário de sumidouros a serem criados para tornar um grafo G livre de deadlock no modelo Ou é igual ao número de knots em G.

Demonstração. Seja H um knot em G, isto é, componente fortemente conexa sem saídas. Por definição, quaisquer dois vértices v, u em H possuem caminhos direcionados entresi, em ambos os sentidos. A resolução de um vértice $v \in v(H)$ torna v um vértice executável (sumidouro), e qualquer vértice $u \in V(H-v)$ terá portanto um caminho direcionado a v. Portanto, exatamente uma resolução de vértice é necessária e suficiente para tornar Hlivre de deadlock. Qualquer componente fortemente conexa C de G com distância 1 a Hterá seus caminhos a H inalterados. Portanto, a solução total do sistema, assim como no Lema 4.3, será a soma das soluções de cada knot em G.

Através do Lema 4.5, dado um grafo G no modelo Ou, podemos remover todos os knots de G por meio de resoluções de vértices (uma para cada knot) com o algoritmo de tempo polinomial apresentado a seguir (Algoritmo 3).

Algoritmo 3: Deadlock-Resolution(Ou, Sumidouro)
Entrada:
G: Grafo Ou
Saída:
G': Grafo livre de deadlock
1 início
$2 G' \leftarrow G$
3 $S \leftarrow$ lista de knots utilizando o algoritmo de detecção de knots
4 para cada $knot \ em \ S$ faça
5 Selecione um vértice v de S aleatoriamente
6 Remova todos os arcos de saída do vértice $v \text{ em } G'$
7 fim para cada
s retorne G'
9 fim

Como consequência, temos:

Teorema 4.6. Deadlock-Resolution(Ou, Sumidouro) pode ser resolvido em tempo polinomial.

4.4 Modelo E-Ou e Modelo X-de-Y

O modelo E-Ou é uma generalização do modelo E e do modelo Ou, portanto toda instância de um problema de resolução de deadlock no modelo E ou no modelo Ou também será uma instância do problema no modelo E-Ou. Desta observação segue o seguinte teorema:

Teorema 4.7. Deadlock-Resolution(E-Ou, Arco), Deadlock-Resolution(E-Ou, Vértice) e Deadlock-Resolution(E-Ou, Sumidouro) são problemas NP-Difíceis. *Demonstração*. Podemos restringir os problemas Deadlock-Resolution(E-Ou,b) para os respectivos problemas Deadlock-Resolution(E, b) no modelo E, considerando apenas instâncias onde todo vértice v terá rótulo f(v) igual a 'e'.

Todo sistema distribuído no modelo E-Ou pode ser transformado em tempo polinomial em um sistema distribuído equivalente no modelo X-de-Y. Logo é fácil ver que:

Teorema 4.8. Deadlock-Resolution(X-de-Y, Arco), Deadlock-Resolution(X-de-Y, Vértice) e Deadlock-Resolution(X-de-Y, Sumidouro) são problemas NP-Difíceis.

Demonstração. Podemos restringir os problemas Deadlock-Resolution(X-de-Y,b) para os respectivos problemas Deadlock-Resolution(E,b) no modelo E, considerando apenas instâncias onde $x_i = y_i$ para todo vértice v_i , compatíveis com o modelo E.

4.5 Resultados

A Tabela 4.2 apresenta a complexidade computacional dos problemas de otimização apresentados até o momento. O problema Deadlock-Resolution(Ou, Vértice) é apresentado no capítulo seguinte.

Deadlock-Resolution(a, b)					
	Е	Ou	E-Ou	X-de-Y	
Arco	NP-D	Р	NP-D	NP-D	
Vértice	NP-D	?	NP-D	NP-D	
Sumidouro	NP-D	Р	NP-D	NP-D	

Tabela 4.2: Complexidades de problemas Deadlock-Resolution(a, b).

Capítulo 5

Deadlock-Resolution(Ou, Vértice)

Em [41] mostra-se que, para um dado grafo de espera G = (V, E) em qualquer modelo de deadlock de uma computação distribuída, a resolução de deadlock é polinomial caso $deg^+(v) \leq 1$ para todo $v \in V$. Em tal situação, o deadlock se resume a um ciclo onde basta a remoção de qualquer vértice que participe do ciclo. Neste capitulo, exploraremos diferentes classes de grafos a fim de descobrir características que tornam o problema Deadlock-Resolution(Ou, Vértice) NP-Difícil ou solucionável em tempo polinomial.

5.1 NP-Dificuldade no Modelo Ou

Nesta subseção, responderemos a questão dobre a complexidade do problema deixado em aberto no capítulo anterior.

Teorema 5.1. Deadlock-Resolution(Ou, Vértice) no modelo Ou é NP-Difícil.

Demonstração. Provaremos que 3-SAT \propto Deadlock-Resolution(Ou, Vértice), o que mostra que Deadlock-Resolution(Ou, Vértice) é NP-Difícil. Seja F uma instância do problema 3-SAT, ou seja, uma fórmula booleana na forma conjuntiva normal, com n variáveis e tendo em cada cláusula no máximo 3 literais. Mostraremos que dado um grafo $G_F = (V, E)$, construído a partir de F, existe um conjunto S de vértices de cardinalidade k = n de forma que $G_F[V \setminus S]$ é livre de deadlock se, e somente se, F for satisfatível. A construção é descrita a seguir:

1. Para cada variável x_i em F, cria-se um ciclo direcionado com dois vértices, Tx_i e Fx_i , em G_F .



Figura 5.1: Grafo G_F resultante da transformação polinomial da fórmula F.

- 2. Para cada cláusula C_j em F, cria-se um ciclo direcionado com três vértices. Cada literal da cláusula C_j terá um vértice correspondente no ciclo.
- 3. Para cada vértice x_i ou $\overline{x_i}$ correspondente a um literal da cláusula C_j , cria-se um arco direcionado ligando x_i ao vértice correspondente Tx_i em caso de literal positivo, ou $\overline{x_i}$ a Fx_i caso contrário.

A Figura 5.1 ilustra o grafo resultante da redução.

Suponha que F possui uma atribuição A que a satisfaça. Podemos construir um conjunto S de vértices com cardinalidade k, de forma que $G'_F = G_F[V \setminus S]$ seja livre de deadlock, conforme explicamos a seguir. Para cada variável de F, selecionaremos um vértice de G_F para fazer parte do conjunto S de acordo com a atribuição em A, onde o vértice selecionado corresponde ao valor oposto da atribuição, isto é, se a variável x_i possui uma atribuição positiva em A, Fx_i será incluído em S, caso contrário Tx_i será incluído em S. Como vimos, cada componente fortemente conexa correspondente a uma variável de Fé composta por apenas dois vértices. Após a remoção de S, irão permanecer exatamente k vértices sumidouros, que serão compatíveis com o valor na atribuição A. Desta forma, já que A satisfaz F, pelo menos um vértice de cada ciclo criado para as cláusulas possuirá caminho para pelo menos um vértice, que corresponde a um valor de atribuição que se encontra em A. Consequentemente, todos os vértices relativos às cláusulas possuirão caminho para pelo menos um sumidouro, e assim, o grafo G'_F será livre de deadlock.

Por outro lado, suponha que G_F possua um conjunto S de vértices de cardinalidade

k tal que $G'_F = G_F[V \setminus S]$ seja livre de deadlock. Podemos construir uma atribuição A para F da seguinte forma. Por construção, todas as componentes fortemente conexas correspondentes a cláusulas possuem saídas para componentes que correspondem a variáveis. Já que para cada variável em F um knot é formado em G_F , são necessárias pelo menos k remoções para livrar G_F de deadlock. Ao remover um vértice de cada knot (um vértice de cada par Tx_i, Fx_i) obtém-se uma atribuição A para a fórmula F onde o valor de cada variável x_i em A é oposto ao valor da variável em S. Consequentemente, o valor de cada variável x_i em A corresponde a um vértice sumidouro em G'_F , e como G'_F é livre de deadlock, cada ciclo que corresponde a uma cláusula terá um vértice literal que possui caminho a um sumidouro. Dessa forma, em cada cláusula em F, existirá um literal que é satisfeito pela atribuição A. Logo, se G_F possui solução com k remoções então F é satisfatível.

5.2 NP-Dificuldade para Grafos Fortemente Conexos

De forma geral, um grafo em deadlock no modelo Ou é composto por várias componentes fortemente conexas, e a estrutura que caracteriza o deadlock são as componentes que não possuem arestas de saída (knots). A primeira questão natural torna-se, portanto, investigar a complexidade do problema Deadlock-Resolution(Ou, Vértice) quando o grafo de entrada é um knot.

Corolário 5.2. Deadlock-resolution(Ou, Vértice) é NP-Difícil, mesmo se o grafo G é fortemente conexo.

Demonstração. Com o Teorema 5.1, provamos que 3-SAT \propto Deadlock-Resolution(Ou, Vértice), o que mostra que Deadlock-Resolution(Ou, Vértice) é NP-Difícil. Provaremos que o teorema é verdadeiro mesmo para G fortemente conexo. Seja F uma instância do problema 3-SAT, ou seja, uma fórmula booleana na forma conjuntiva normal, com nvariáveis e m cláusulas, onde cada cláusula possui no máximo 3 literais. Mostraremos que dado um grafo $G_F = (V, E)$ fortemente conexo, construído a partir de F, existe um conjunto S de vértices de cardinalidade k = n + 1, de forma que $G_F[V \setminus S]$ é livre de deadlock se, e somente se, F for satisfatível. A construção é dada a seguir.

Para a construção de G_F , grafo fortemente conexo, primeiramente utilizamos a construção feita no Teorema 5.1, para construir um subgrafo G_F^1 . A seguir, criamos um vértice universal u tal que todos os vértices em G_F^1 são vizinhos de entrada e saída de u. O grafo resultante é o grafo G_F . A Figura 5.2 exibe esta construção.



Figura 5.2: Grafo G_F fortemente conexo resultante da transformação polinomial da fórmula F.

Suponha que F possui uma atribuição A que a satisfaça. Podemos construir um conjunto S de vértices com cardinalidade k = n + 1, de forma que $G'_F = G_F[V \setminus S]$ seja livre de deadlock. A construção do conjunto S é dada a seguir. Por construção, G_F possui 2n + 3m + 1 vértices. Note que todos os vértices em $G_F - u$ formam um ciclo direcionado com u. Observe que se u não pertencesse a S, qualquer vértice em G'_F garantiria a condição de deadlock, já que k < 2n + 3m. Portanto, inicialmente, adicionamos u a S, e então o grafo $G_F - u$ se resume a G^1_F . Assim, torna-se fácil verificar que o restante do conjunto S pode ser facilmente obtido conforme o Teorema 5.1.

Por outro lado, suponha que o grafo G_F possui um conjunto S de vértices com cardinalidade k = n+1, onde $G_F[V \setminus S]$ é livre de deadlock. Podemos construir uma atribuição A para F da seguinte forma. Note que u sempre estará em S; portanto, a fim de construir a atribuição A, podemos considerar apenas o subgrafo G_F^1 . Logo, a atribuição A pode ser construída da mesma forma que no Teorema 5.1. Neste ponto, é fácil ver que se G_F tem solução com k remoções, então F é satisfativel.

5.3 NP-Dificuldade para Grafos Bipartidos

Teorema 5.3. Deadlock-resolution(Ou, Vértice) é NP-Difícil mesmo se o grafo de entrada G é bipartido, $\Delta(G) \ge 4 \ e \ \Delta(G)^+ = 2.$

Demonstração. Com o auxílio do Teorema 5.1, provamos que 3-SAT-AM3 \propto Deadlock-Resolution(Ou, Vértice), o que mostra que Deadlock-Resolution(Ou, Vértice) é NP-Difícil. Provaremos que o Teorema é verdadeiro mesmo para G bipartido, $\Delta(G) \geq 4$ e $\Delta(G)^+ = 2$. Seja F uma instância do problema 3-SAT-AM3, ou seja, uma fórmula booleana na forma normal conjuntiva, com n variáveis e m cláusulas, onde cada cláusula possui no máximo 3 literais e cada variável ocorre no máximo 3 vezes (pelo menos uma vez negativamente e pelo menos uma vez positivamente). Mostraremos que dado um grafo $G_F = (V, E)$ bipartido com $\Delta(G) \geq 4$, construído a partir de F, existe um conjunto S de vértices de cardinalidade k = n de forma que $G_F[V \setminus S]$ é livre de deadlock se, e somente se, F for satisfatível. A construção é dada a seguir:

- 1. Para cada variável x_i em F, criar um ciclo direcionado com dois vértices, Tx_i e Fx_i em G_F .
- 2. Para cada cláusula C_j em F, criar um ciclo direcionado com seis vértices. Cada literal da cláusula C_j terá um vértice correspondente no ciclo.
- 3. Para cada vértice x_i ou $\overline{x_i}$ correspondente a um literal da cláusula C_j , criar um arco direcionado ligando x_i ao vértice correspondente Tx_i em caso de literal positivo, ou $\overline{x_i}$ a Fx_i caso contrário.

Informação adicional: Salientamos que a bipartição de G_F pode ser obtida da seguinte maneira. Cada ciclo direcionado com seis vértices relativo a uma cláusula possui dois vértices para cada variável, um para o de literal positivo, e outro para o negativo. Desta forma, podemos separar vértices de literais positivos e negativos de todos os ciclos de tamanho seis em duas partições V_1 (os positivos) e V_2 (os negativos). Os ciclos restantes são de tamanho dois e também podem ser incluídos nas partições V_1 e V_2 , uma vez que vértices x_i de literais positivos somente se ligam a vértices Tx_i , e vértices $\overline{x_i}$ de literais negativos somente se ligam a vértices Fx_i . A bipartição é garantida colocando os vértices remanescentes da seguinte maneira: vértices positivos (Tx_i) em V_2 e negativos (Fx_i) em V_1 . A Figura 5.3 exemplifica a construção.



Figura 5.3: Grafo G_F bipartido resultante da transformação polinomial da fórmula F.

Suponha que F possui uma atribuição A que a satisfaça. Podemos construir um conjunto S de vértices com cardinalidade k = n, de forma que $G'_F = G_F[V \setminus S]$ seja livre de deadlock, da seguinte forma. Para cada variável de F, selecionaremos um vértice de G_F para fazer parte do conjunto S de acordo com a atribuição em A, onde o vértice selecionado corresponde ao valor oposto da atribuição, isto é, se a variável x_i possui uma atribuição positiva em A, Fx_i será incluído em S, caso contrário Tx_i será incluído em S. Como cada componente fortemente conexa correspondente ao valor positivo ou negativo de uma variável de F é composta por apenas dois vértices, após a remoção de S irão permanecer exatamente k vértices sumidouros, que serão compatíveis com o valor na atribuição A. Desta forma, já que A satisfaz F, pelo menos um literal de cada ciclo criado referente às cláusulas possuirá caminho para pelo menos um vértice que corresponde ao valor da atribuição de A. Consequentemente, todos os vértices relativos às clausulas possuirão caminho a pelo menos um sumidouro, e assim o grafo G'_F será livre de deadlock.

Por outro lado, suponha que G_F possua um conjunto S de vértices de cardinalidade k = n tal que $G'_F = G_F[V \setminus S]$ seja livre de deadlock. Podemos construir uma atribuição A para F da seguinte forma. Por construção, todas as componentes fortemente conexas correspondentes às cláusulas possuem saídas para as componentes que correspondem às variáveis através de vértices que correspondem a literais; portanto, G_F possui k knots. Já que para cada variável em F um knot é formado em G_F , pelo menos k remoções são necessárias para livrar G_F de deadlock. Ao remover um vértice de cada knot (um vértice

de cada par Tx_i , Fx_i) obtém-se uma atribuição A para a fórmula F tal que o valor de cada variável x_i em A é oposto ao valor da variável em S. Consequentemente, o valor de cada variável x_i em A corresponde a um vértice sumidouro em G'_F , e como G'_F é livre de deadlock, cada ciclo que corresponde a uma cláusula terá um vértice literal que possui caminho a um sumidouro. Da mesma forma, para cada cláusula em F existirá um literal satisfeito pela atribuição A. Logo, se G_F possui solução com k remoções então Fé satisfatível.

5.3.1 NP-Dificuldade para Grafos Planares

Corolário 5.4. Deadlock-resolution(Ou, Vértice) é NP-Difícil mesmo se o grafo G é planar bipartido e $\Delta(G) \ge 4$.

Demonstração. Para provar que Deadlock-resolution(Ou, Vértice) é NP-Difícil mesmo se G é planar, faremos uma redução de 3-SAT-AM3 planar, versão de 3-SAT onde cada variável ocorre no máximo três vezes (pelo menos uma vez negativamente e pelo menos uma vez positivamente), e o grafo bipartido induzido pelas cláusulas e variáveis de F é planar. Este problema foi provado NP-Completo em [47].

A construção do grafo é igual à do Teorema 5.3. Dessa forma, temos dois *gadgets*. O primeiro, um ciclo de seis vértices para cada cláusula. O segundo, um ciclo de dois vértices para cada variável em F. É fácil ver que o grafo construído a partir de F permanece planar, uma vez que o bipartido natural induzido por F também é planar.

5.4 Grafos Subcúbicos

Nas seções anteriores, exploramos classes de grafos de entrada para o problema Deadlock-Resolution(Ou, Vértice). Provamos que para grafos com grau máximo 4 (somando graus de entrada e saída), o problema é NP-Difícil. Além disso, sabemos que o problema para grafos com grau máximo 2 é trivial. Nesta seção iremos explorar a classe de grafos subcúbicos (grafos com grau máximo 3).

Primeiramente, vamos classificar todos os possíveis tipos de vértices no problema Deadlock-Resolution(Ou, Vértice) para grafos subcúbicos. Os tipos de vértices são apresentados na Figura 5.4: vértices fonte (que possuem apenas vizinhos de saída), vértices sumidouro (que possuem apenas vizinhos de entrada), e demais vértices (que possuem pelo menos um vizinho de entrada e um de saída).



Figura 5.4: Tipos de vértices em um grafo subcúbico.

5.4.1 Fontes e Sumidouros

Para eliminar vértices desnecessários em qualquer possível solução ótima, iremos analisar as fontes e os sumidouros, e provar que nenhum vértice destes tipos participa ou influencia em qualquer solução ótima, como explicado na proposição a seguir.

Proposição 5.5. Fontes e sumidouros são desnecessários à remoção de deadlock.



Figura 5.5: Vértices sumidouro em grafos subcúbicos.

Demonstração. Primeiramente, trataremos o caso dos sumidouros (Figura 5.5). Um sumidouro é um vértice pronto para executar suas tarefas no instante de tempo corrente. No modelo Ou, cada processo v_i depende somente da resposta de um processo $v_j \in O_i$; logo, um sumidouro fornecerá a todos os vértices em I_i mensagens de concessão, tornando-os assim novos sumidouros. Consequentemente, todos os vértices em A_i serão liberados em tempo finito. Analisando a estrutura do grafo, qualquer vértice que possua um caminho direcionado a um sumidouro não está em deadlock no sistema; sendo assim, seu descarte remoção não interfere de forma alguma em qualquer solução ótima.

Ao analisar as fontes (Figura 5.6), teremos que todos os caminhos direcionados que partem de uma fonte levam a knots, caso contrário esta fonte possuiria caminho direcionado a um sumidouro e portanto não estaria em deadlock. Seja v_i uma fonte e seja Qum knot em D_i , que será resolvido por remoção de vértices. Após resolver o knot por remoção de vértices, temos duas possibilidades: ou existe um caminho direcionado de v_i



Figura 5.6: Fontes em grafos subcúbicos.

a um sumidouro (e neste caso v_i é irrelevante para a resolução do deadlock) ou todos os caminhos direcionados que partem de v_i continuam levando a knots (note que, por ser fonte, v_i não pode pertencer a nenhum destes knots). Tomamos então um novo knot Q'em D_i que será resolvido por remoção de vértices, e assim por diante. Como o grafo é finito, ao final existirá um caminho direcionado de v_i a um sumidouro (que pode ser inclusive um caminho trivial, isto é, formado apenas por v_i), o que nos permite concluir que v_i é realmente irrelevante para a resolução do deadlock.

Pré-processamento 1. Dada a Proposição 5.5, podemos eliminar sucessivamente do grafo todos os vértices fonte. A seguir, podemos eliminar do grafo todos os vértices que alcançam sumidouros (incluindo os próprios sumidouros). Estas duas operações podem ser repetidas nesta ordem até que não possam ser mais aplicadas. Consequentemente, a partir deste momento, levaremos em consideração que o grafo não possui fontes nem sumidouros.

Como mostra a Figura 5.7, temos portanto três tipos de vértices a serem considerados: tipo A, com um arco de entrada e um de saída; tipo B, com um arco de entrada e dois de saída; e tipo C, com dois arcos de entrada e um de saída.



Figura 5.7: Vértices com pelo menos uma aresta de entrada e pelo menos uma de saída.

5.4.2 Refinamento do Grafo

Na subseção anterior fizemos um primeiro pré-processamento no grafo, onde foi possível efetuar uma redução no grafo removendo vértices dispensáveis. Adotaremos a partir de agora a estratégia de analisar continuamente as características do grafo a fim de estabelecer regras e procedimentos que possam definir vértices específicos que façam parte de uma solução ótima. Dessa forma, iremos sempre, de forma iterativa, construir soluções parciais contidas numa solução ótima.

Primeiramente, estabelecemos a seguinte regra:

Regra 5.6. Dado um knot Q em G, se existe um vértice v_i em Q tal que $Q-v_i$ é livre de deadlock e a remoção de v_i não gera nenhum novo knot em $G! - v_i$ (além dos outros que já existiam) então remova v_i .

Pré-processamento 2. Esta regra pode ser aplicada sempre que possível, pois para resolver Q é necessário realizar pelo menos uma remoção, e além disso, nenhum novo knot é criado no grafo após a remoção.

Trataremos agora de estabelecer meios de identificar vértices que se enquadram na Regra 5.6.

Para resolver o deadlock de um grafo G, primeiramente devemos resolver o deadlock de cada um dos knots em G. No entanto, a remoção de alguns vértices pode gerar novos knots que também precisam ser resolvidos. Sendo assim, nosso objetivo agora é identificar localmente, através de uma análise individual e isolada de cada knot de G, vértices que sem perda de generalidade fazem parte de uma solução ótima.

Observando subgrafos induzidos por conjuntos de vértices C que formam uma CFC em G, podemos classificar os vértices que são localmente do tipo A em G[C] em três categorias (veja Figura 5.8): um vértice é do subtipo A.1 se ele é do tipo A em G[C], porém no grafo original é do tipo C, i.e, possui uma aresta de entrada vinda de uma outra CFC; um vértice é do subtipo A.2 quando o mesmo é do tipo A tanto em G[C] quanto em G; por fim, um vértice é do subtipo A.3 quando o mesmo é do tipo A em G[C], mas em G ele é do tipo B. Note que em um knot nunca haverá vértices do tipo A.3.

É interessante notar que em um grafo subcúbico todo vértice do knot possui no máximo um vizinho externo (entrada). Pois, por definição, um knot é fortemente conexo; logo, qualquer vértice v_i que pertence a um knot necessita de pelo menos uma aresta de entrada e uma de saída neste knot, uma vez que todo knot é uma CFC. Observe também



Figura 5.8: Subtipos de vértices A em uma CFC.

que essa propriedade também é válida para vértices de CFCs quaisquer. No entanto, para esse caso, o vizinho externo será de entrada ou saída.

No lema a seguir é apresentada uma relação entre vértices do tipo $B \in C$.

Lema 5.7. Seja G um grafo subcúbico fortemente conexo. Então, o número de vértices do tipo B é exatamente igual ao número de vértices do tipo C.

Demonstração. Seja G um digrafo. Sabemos que $\sum_{v_i \in V} deg^-(v_i) = \sum_{v_i \in V} deg^+(v_i)$ [50]. Como G é subcúbico e fortemente conexo, então G possui apenas vértices dos tipos $A, B \in C$. Dado que um vértice do tipo A possui apenas uma entrada e um saída, o número de vértices do tipo B deve ser igual ao número de vértices do tipo C.

Neste momento, podemos identificar cenários em que podemos aplicar a Regra 5.6.

Lema 5.8. Seja $Q \in G$ um knot, onde G é subcúbico e Q possui pelo menos um vértice do tipo B. É possível solucionar o deadlock de Q sem criar um novo deadlock em G, com apenas uma remoção.

Demonstração. Dado que Q é fortemente conexo, qualquer sumidouro proveniente da remoção de apenas um vértice o solucionará. Como um vértice do tipo B não tem nenhum vizinho externo, sua remoção não gera novos knots em $G \setminus Q$. Dado um vértice v_i do tipo B em Q, a remoção de v_i não criará um sumidouro somente se o vértice em I_i (vizinho de entrada de v_i) também for do tipo B. Nesse caso repetimos nossa análise para o vértice em I_i . Seguindo sucessivamente esse raciocínio, temos duas possibilidades: ou vamos encontrar um vértice v_j ancestral de v_i que é do tipo B onde o vértice em I_j não é do tipo B (nesse caso, o vértice em I_j torna-se sumidouro ao remover v_j), ou todos os vértices em A_i são do tipo B. Para esse segundo caso deve ser observado que Q é fortemente conexo; logo, todos os vértices de Q estão em A_i , e consequentemente todo vértice de Q é do tipo B, o que é um absurdo pelo Lema 5.7).

Corolário 5.9. Seja $Q \in G$ um knot, onde G é subcúbico e Q possui pelo menos um vértice do tipo C. É possível solucionar o deadlock de Q sem criar um novo deadlock em G, com apenas uma remoção.

Demonstração. Segue diretamente do Lema 5.7 e do Lema 5.8. $\hfill \Box$

Lema 5.10. Dado um knot $Q \in G$, onde G é subcúbico e Q possui pelo menos um vértice do tipo A.2, é possível solucionar o deadlock com apenas uma remoção sem criar um novo deadlock.

Demonstração. Suponha que existe um vértice v_i do tipo A.2 em Q. Vértices do tipo A.2 não possuem ligação externa, e sua remoção em Q somente não o resolve caso I_i corresponda a um vértice do tipo B. Nesse caso, o Lema 5.8 é aplicável.

Proposição 5.11. Seja Q um knot de um grafo subcúbico G, onde a Regra 5.6 não é aplicável. Então, Q é um ciclo direcionado composto apenas por vértices do tipo A.1.

Demonstração. Se a Regra 5.6 não é aplicável, isso significa que a remoção de qualquer vértice de Q ou gera novos knots em G ou não resolve Q. Como Q é um knot, Q não possui vértices do tipo A.3. Usando o Lema 5.8, o Corolário 5.9 e o Lema 5.10, temos que Q não possui vértices dos tipos B, C e A.2. Logo, Q possui apenas vértices do tipo A.1. Como Q é fortemente conexo, Q é um ciclo direcionado.

A Figura 5.9 ilustra um knot que se enquadra no cenário da Proposição 5.11.

5.4.3 Algoritmo Aproximativo

Podemos determinar um limite inferior e superior para o problema sendo estudado, pois para um dado grafo subcúbico G contendo k knots precisamos de pelo menos k remoções, uma em cada knot, para solucionar o deadlock em G (limite inferior).

No pior caso, todos os knots são compostos apenas de vértices do tipo A.1 (Figura 5.9). Neste caso são necessárias no máximo 2k remoções para solucionar o deadlock em G: para cada knot original Q basta efetuar a remoção de um vértice qualquer; e para os knots Q', gerados após essas remoções (no máximo k novos knots), a Regra 5.6 é sempre aplicável, pois todo knot Q' possui um vértice w do tipo A.2 (Lema 5.10). Esse vértice



Figura 5.9: Knots restantes após o pré-processamento descrito nos Lemas 5.8, 5.7 e 5.10.

w, originalmente (antes das primeiras remoções), era um vértice do tipo A.3 pois w era vizinho de entrada de um vértice v que foi removido de um knot original Q.

Até o momento, nesta seção, o proposito principal foi elucidar todos os aspectos do problema Deadlock-Resolution(Ou, Vértice) para grafos subcúbicos. Tal análise nos garante ao menos um algoritmo 2-aproximativo para o problema (Algoritmo 4).

O Algoritmo 4 pode ser executado em tempo O(nm). Porém, se ocultarmos as linhas 8 a 11, o algoritmo continua 2-aproximativo e pode ser executado em tempo O(n + m).

5.4.4 Resolução em Tempo Polinomial

A fim de obter uma solução ótima em tempo polinomial, há mais considerações importantes a serem feitas quanto à estrutura do grafo.

Observação 5.12. Podemos desconsiderar qualquer CFC com mais de uma saída a um knot. Qualquer componente fortemente conexa C que possua duas ou mais saídas para um mesmo knot K garante que K pode ser solucionado com apenas uma remoção sem gerar novo knot, pois basta que o vértice removido seja um dos adjacentes a C.

Lema 5.13. Podemos desconsiderar qualquer CFC C^1 que alcança alguma CFC C^2 que

Algoritmo 4: Algoritmo 2-aproximativo		
Entrada:		
G: Grafo de espera subcúbico no modelo Ou		
Saída:		
S: Lista de vértices a serem removidos		
1 início		
2 $G' \leftarrow G$ – Vértices obtidos pelo Pré-processamento 1		
3 $K \leftarrow$ lista de knots de G' utilizando o algoritmo de detecção de knots		
4 para cada <i>knot Q em K</i> faça		
5 se Q contém um vértice v_i dos tipos A.2, B ou C então		
6 Aplicar a Regra 5.6 a Q inserindo o vértice selecionado em S		
7 senão		
8 se Q possui duas arestas de entrada provenientes de uma CFC, C então		
9 $S \leftarrow S \cup$ um vértice de Q que é adjacente a C		
10 (para Q a Regra 5.6 é aplicável)		
11 senão		
12 Adicionar um vértice qualquer v de Q em S		
13 fim se		
14 fim se		
15 fim para cada		
16 $K' \leftarrow$ lista de knots de $G[V \setminus S]$ utilizando o algoritmo de detecção de knots		
7 para cada $knot Q' em K'$ faça		
s Aplicar a Regra 5.6 a Q' inserindo o vértice selecionado em S		
19 fim para cada		
20 retornar S		
21 fim		

não é knot. Além disso, se C^1 está à distância 1 de um knot Q, a Regra 5.6 é aplicável a Q.

Demonstração. Se uma CFC C^1 alcança uma outra CFC C^2 então C^1 alcança uma CFC C^j que está à distância 1 de um conjunto de knots K_{C^j} . C^j ou é resolvida apenas pela resolução dos knots em K_{C^j} , ou, após a resolução dos knots em K_{C^j} , uma remoção em C^j será necessária e suficiente. Em ambos os casos remoções em C^1 não serão necessárias. Além disso, se C^1 está à distância 1 de um knot Q, o vértice $w \in V(Q)$, que possui um vizinho de entrada em C^1 , pode ser removido de Q, sem riscos de transformar C^1 em knot.

A figura 5.10 ilustra a Observação 5.12 e Lema 5.13. Pela Observação 5.12, podemos resolver o knot formado por um ciclo direcionado de n vértices sem gerar novo knot apenas removendo ou o vértice v_1 ou o vértice v_2 , dessa forma o knot será solucionado e a CFC_3 será resolvida pela aresta azul remanescente. Em relação ao Lema 5.13, podemos notar que



Figura 5.10: Refinamento obtido com a Observação 5.12 e Lema 5.13.

a CFC_2 pode ser desconsiderada para nossa análise já que não possui aresta direta a um knot, dessa forma, sempre será solucionado por CFCs intermediarias (nesse caso CFC_3). Ademais, a CFC_1 possui uma aresta direta ao knot e um caminho direcionado a essa mesma knot através de CFCs intermediárias, podemos simplesmente remover o vértice correspondente a essa aresta direta (em vermelho ligada ao vértice v_3) que a componente será solucionada por CFCs intermediarias (nesse caso CFC_2 e CFC_3).

Pré-processamento 3. Pela Observação 5.12 resolver knots que possuem CFCs com dois caminhos direcionados a um unico knot. Pelo Lema 5.13, podemos remover todas as CFCs que alcançam CFCs que não são knots.

Observação 5.14. O grafo a ser considerado. Pelos lemas já apresentados temos que o grafo a ser analisado, sem perda de generalidade, possui somente knots que são ciclos direcionados de vértices do tipo A.1, e todas as demais CFCs estão à distância um de um knot e não alcançam nenhuma outra CFC que não seja knot.

Observação 5.15. Vértices solucionadores. Após o Pré-processamento 3, resta um grafo particionado em duas coleções de vértices. A primeira é formada pelos knots, e a segunda é formada pelas componentes fortemente conexas com saídas apenas para knots. Nesse grafo, podemos observar que a remoção de um vértice v em um knot garante a existência de um vértice solucionador, vértice que resolve o deadlock da CFC que alcança v, caso ela se torne um knot, e também novos knots, caso removido. Note que a garantia



da existência de solucionadores não ocorre em grafos com grau quatro ou maior.

Figura 5.11: Grafo bipartido resultante das contrações de componentes C e knots K.

Nesse ponto, busca-se encontrar uma forma de resolver deadlock de forma a minimizar o número de solucionadores que precisam de fato serem removidos para que toda a rede seja solucionada. Podemos então contrair os knots e as componentes de forma a obter um grado bipartido $G = (K \cup C, E)$ onde K é um conjunto de vértices obtidos após a contração de cada knot em um único vértice, e C é um conjunto de vértices obtidos pela contração de cada uma das demais componentes em um único vértice (Figura 5.11). Note que cada aresta entre um vértice $k_i \subseteq K$ e outro vértice $c_j \subseteq C$ representa a ligação entre um vértice no knot representado por k_i que o resolve, e um vértice em c_j que garante a existência de solucionador em C_j , que poderá ou não ser utilizado. Portanto, busca-se encontrar um conjunto M' de arestas tal que:

- 1. Cada vértice em K é adjacente a no máximo uma aresta de M' (as arestas selecionadas indicam os vértices dos knots a serem removidos).
- 2. Cada vértice de C possui pelo menos uma aresta que não pertence a M' (arestas que não pertencem a M' indicam maneiras de componentes em C serem resolvidas sem remover vértices internos, "solucionadores").
- 3. M' é máximo.

Na Figura 5.12, um conjunto M' é dado para o grafo $G = (K \cup C, E)$. Já que |M'| = |K|, todos os vértices em K são saturados por M' (por definição, cada vértice

de K é adjacente a no máximo uma aresta de M'). Note que se removermos para cada knot os vértices no grafo original correspondente à aresta de M', todos os knots serão resolvidos; ainda, nenhum vértice solucionador será necessário, já que há uma pelo menos uma aresta (c_i, k_j) em cada componente $c_i \in C$ que não pertence a M'.



Figura 5.12: Exemplo de conjunto M' que satura todos os vértices em K.

Na Figura 5.13, um conjunto M' é dado para o grafo $G = (K \cup C, E)$. Já que |M'| = |K| - x, teremos que |K| - x knots podem ser resolvidos sem necessidade de criar novos knots. Haverá portanto a necessidade de serem removidos vértices nos x knots restantes, o que acarretará a criação de x novos knots, e consequentemente a utilização de x vértices solucionadores na solução.



Figura 5.13: Exemplo de conjunto M' que satura t - 1 vértices. O vértice destacado em vermelho não é saturado por M'.

Seja $B = (K \cup C, E)$ um grafo bipartido proveniente da contração dos knots e das CFC's com ligação direta a knots em um grafo G. O conjunto de arestas M' induz um conjunto S' ótimo de vértices a serem removidos para tornar G (grafo anterior à contração) livre de deadlock. Pois, quanto maior for M', maior é o número de knots que podem ser resolvidos sem gerar novos knots. O conjunto de vértices S' a ser removido é construído da seguinte forma: para cada vértice k_i em K, adjacente a M', será incluído em S' o vértice em G associado à aresta de k_i em M'; para todo vértice $k_i \in K$ onde k_i não é adjacente a M', escolher uma aresta aleatória, e incluir em S' o vértice de G associado a esta aresta; além disso, para a CFC do grafo original que está associada a esta mesma aresta, devemos incluir um de seus solucionadores em S'.

Para obter tal conjunto de arestas, conceitos de emparelhamento se fazem necessários.

Emparelhamento é um problema clássico em Teoria dos Grafos, formalmente definido em sua forma definitiva por Hall [29]. Posteriormente, em [62], foi introduzido o termo "Teorema do Casamento", também conhecido como Teorema de Hall.

Semi-emparelhamento é uma generalização natural do problema clássico de emparelhamento em grafos bipartidos [35]. Seja $G = (K \cup C, E)$ um grafo bipartido com n = |K| + |C| vértices e m = |E| arestas. Em geral consideraremos apenas casos não triviais e grafos conexos. Um semi-emparelhamento M de G é um conjunto de arestas $M \subseteq E(G)$ tal que cada vértice de K é incidente a exatamente uma aresta em M.

Em [27] foi apresentada uma generalização do semi-emparelhamento denominada (f,g)-semi-emparelhamento, que também pode ser resolvida em polinomial. Seja deg(v) o número de arestas incidentes ao vértice v no grafo G. Denotamos $deg_M(v)$ o número de arestas de M incidentes em v. Sejam $f: k \to N \in g: c \to N$ funções. Um (f,g)-semi-emparelhamento em um grafo bipartido $G = (K \cup C, E) \operatorname{com} n = |K| + |C|$ vértices e m = |E| arestas é um conjunto de arestas $M \subseteq E(G)$ tal que cada vértice k_i em K é incidente a no máximo $f(k_i)$ arestas em M, e cada vértice c_j em C é incidente a no máximo $g(c_i)$ arestas em M [11].

Lema 5.16. Dado um grafo bipartido $G = (K \cup C, E)$ e duas funções $f : k \to \mathbb{N}$ e $g : c \to \mathbb{N}$, onde $k \in K$ e $c \in C$, encontrar um (f, g)-semi-emparelhamento máximo de G pode ser feito em tempo polinomial.

Demonstração. Diversos algoritmos utilizando ideias similares às do conhecido algoritmo de Hopcroft-Karp [31] são apresentados na literatura [11, 27, 35]. $\hfill\square$

Finalmente podemos afirmar que o seguinte teorema é verdadeiro.

Teorema 5.17. Deadlock-Resolution(Ou, Vértice) para grafos subcúbicos pode ser resolvido em tempo polinomial.

Demonstração. Primeiramente devemos efetuar todos os pré-processamentos já apresentados, e em seguida construir o grafo bipartido equivalente. Dado o grafo bipartido, o conjunto de arestas M' desejado é obtido por um (1,g)-semi-emparelhamento, onde $g = deg(c_j) - 1$ para todo c_j em C. O Algoritmo para (1,g)-semi-emparelhamento pode ser executado em tempo $O(m\sqrt{n})$.

Se |M'| = |K|, isto significa que para cada knot um vértice foi escolhido para ser removido (indicado pela aresta de M') e cada componente em C será liberada por algum knot resolvido (pois alguma aresta da componente não está em M'). Logo as arestas de M' induzem um conjunto ótimo de vértices a ser removido.

Suponha M' máximo e |M'| = |K| - x, para algum x. Primeiro note que se |M'| = |K| - x então x vértices da partição K não pertencem ao (1, g)-semi-emparelhamento, pois cada vértice de K possui no máximo uma aresta em M'. Observe também que ao tentar adicionar uma aresta de um vértice que ficou de fora de M', saturamos algum vértice da partição C, e cada vértice de fora satura um vértice distinto de C (caso contrário M' não seria máximo). As escolhas dos vértices que ficaram de fora indicam as suas remoções, e a saturação de uma componente indica a criação de um novo knot. Nesse ponto, o número de vértices que fica de fora é igual ao número de componentes que se transformam em knots e ao número de solucionadores que precisam ser removidos. Se temos |M'| = |K| - x, e M' é máximo, então x é mínimo, e a partir de M' podemos construir um conjunto |S'| = |K| + x de vértices a serem removidos de forma que G se torne livre de deadlock.

Por outro lado, se temos um conjunto S' a ser removido, de cardinalidade |K| + x, essas remoções induzem um conjunto de arestas no grafo bipartido que satura x vértices de C, e portanto existe um (1, g)-semi-emparelhamento para o grafo bipartido de tamanho |K| - x.

A partir do Teorema 5.17, elaboramos o Algoritmo 5. O algoritmo tem como entrada um grafo subcúbico G = (V, E) no modelo Ou, e encontra um conjunto mínimo de vértices S tal que $G[V \setminus S]$ é livre de deadlock.

O Algoritmo 5 executa como principais subrotinas os Pré-processamentos 1, 2 e 3 e o

Alg	Algoritmo 5: Algoritmo resolução subcúbico		
Entrada:			
G	E: Grafo de espera subcúbico no modelo Ou		
\mathbf{S}	aída:		
S	: Lista de vértices a serem removidos		
1 ir	nício		
2	$G' \leftarrow G$ – Vértices obtidos pelo Pré-processamento 1		
3	$K \leftarrow$ lista de knots de G' utilizando o algoritmo de detecção de knots		
4	para cada $knot \ Q \ em \ K$ faça		
5	se Q contém um vértice v_i dos tipos A.2, B ou C então		
6	Aplicar a Regra 5.6 a Q inserindo o vértice selecionado em S		
7	senão		
8	$\mathbf{se} \ Q \ possui \ duas \ arestas \ de \ entrada \ provenientes \ de \ uma \ CFC, \ C \ então$		
9	Incluir em S um vértice de Q que é adjacente a C		
10	(para Q a Regra 5.6 é aplicável)		
11	senão		
12	se Q possui uma aresta de entrada proveniente de uma CFC C^1 ,		
	que alcança uma outra CFC C^2 (não é knot) então		
13	Incluir em S um vértice de Q que é adjacente a C^1		
14	(para Q a Regra 5.6 é aplicável)		
15	fim se		
16	fim se		
17	fim se		
18	fim para cada		
19	Desconsidere qualquer CFC C^1 que alcança alguma outra CFC C^2 que não é		
	knot.		
20	20 $G' \leftarrow O$ grafo a ser considerado, após Pré-Processamentos 1, 2 e 3		
21	1 (Veja Observação 5.14)		
22	$2 \qquad B \leftarrow \text{Grafo } G' \text{ contraído}$		
23	23 $M' \leftarrow (1, g)$ -semi-emparelhamento de B		
24	Incluir em S vértices obtidos a partir de M'		
25	retornar S		
26 fi	m		

(1, g)semi-emparelhamento. Tal algoritmo pode ser executado em tempo O(nm).

5.5Considerações Finais

A Tabela 5.1 apresenta os resultados obtidos neste capítulo sobre a complexidade computacional do problema Deadlock-Resolution(Ou, Vértice).

Deadlock-Resolution(Ou, Vértice)				
Instância	Complexidade			
Fracamente conexo	NP-Difícil			
Fortemente conexo	NP-Difícil			
Bipartido, Planar, $\Delta(G) \ge 4 \in \Delta(G)^+ = 2$	NP-Difícil			
$\Delta(G) = 3$	Polinomial			
$\Delta(G) = 2$	Trivial			
$\Delta(G)^+ = 1$	Trivial			

Tabela 5.1: Complexidade para diferentes classes de grafos do problema Deadlock-Resolution(Ou, Vértice).

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho, um estudo completo em modelos de deadlock foi feito, e uma nova hierarquia de modelos clássicos de deadlock é fornecida. Em nossa análise, consideramos *poder de expressão* e *tempo computacional polinomial* como aspectos fundamentais. Para o melhor de nosso conhecimento, esta é uma nova abordagem para este tipo de estudo.

Introduzimos uma definição formal e adequada para caracterizar o fato de que um determinado modelo de deadlock generaliza outro. Também propomos um novo modelo de deadlock, o modelo E-Ou simplificado, que simplifica o modelo E-OU clássico mantendo o mesmo poder de expressão.

Mostramos que a análise clássica de modelos de deadlock é imprecisa no que diz respeito ao tempo computacional, e que os modelos E-Ou, E-Ou simplificado, X-de-Y e X-de-Y disjuntivo têm o mesmo poder de expressão em tempo polinomial. No entanto, ao levar em conta a preservação da ordem de profundidade da rede, observa-se uma ligeira mudança na hierarquia, o que sugere que os modelos X-de-Y e X-de-Y disjuntivo são os modelos mais expressivos .

No Capítulo 4 é definida a classe de problemas Deadlock-Resolution(a, b) e posteriormente estudada a complexidade computacional dos problemas de otimização desta classe. Demonstramos que Deadlock-Resolution(E, Arco) e Deadlock-Resolution(E, Vértice) são equivalentes aos problemas clássicos Direct Feedback Arc Set e Direct Feedback Vertex Set respectivamente; portanto, são também NP-Difíceis. Também é provado que Deadlock-Resolution(E, Sumidouro) é NP-Difícil . Foram encontrados dois casos polinomiais, Deadlock-Resolution(Ou, Arco) e Deadlock-Resolution(Ou, Sumidouro), e mostramos que Deadlock-Resolution(Ou, Vértice) é NP-Completo. Para os modelos restantes, restringindo as instâncias ao Modelo E (modelo mais simples), provamos que Deadlock-
Resolution (E-Ou,b) e Deadlock-Resolution (X-de-Y,b) são NP-Completos. Temos assim a Tabela a seguir.

Deadlock-Resolution(a, b)				
	Ε	Ou	E-Ou	X-de-Y
Arco	NP-D	Р	NP-D	NP-D
Vértice	NP-D	NP-D	NP-D	NP-D
Sumidouro	NP-D	Р	NP-D	NP-D

Tabela 6.1: Complexidade final da classe de problemas Deadlock-Resolution(a, b).

Finalmente, no Capítulo 5, um estudo detalhado sobre a complexidade em diferentes tipos de grafos do problema Deadlock-Resolution(Ou, Vértice) foi realizado. Foi provado que o problema é NP-Difícil para grafos fortemente conexos, grafos bipartidos com grau máximo quatro, grafos planares e grafos bipartidos planares com grau máximo quatro. Além disso, provamos que para grafos subcúbicos, isto é, grafos com grau máximo três, o problema é polinomial. Adicionalmente, foram apresentados algoritmos para a solução do problema para grafos subcúbicos.

Mostra-se promissor o desenvolvimento de algoritmos distribuídos, principalmente nos casos polinomiais, para os problemas Deadlock-Resolution(Ou, Arco) e Deadlock-Resolution(Ou, Sumidouro), uma vez que já é conhecido um algoritmo distribuído para a detecção de knots [17, 40].

A classe de problemas Deadlock-Resolution(E, b) foi amplamente estudada em termos de complexidade parametrizada, com o auxilio dos problemas clássicos Direct Feedback Arc Set e Direct Feedback Vertex Set. Como trabalho futuro, pretende-se estender este tipo de estudo às outras variantes do problemas Deadlock-Resolution(a, b), principalmente para o problema Deadlock-Resolution(Ou, Vértice) (que já apresenta parametrização eficiente).

Referências

- AJTAI, M.; KOMLÓS, J.; SZEMERÉDI, E. An 0 (n log n) sorting network. In Proceedings of the fifteenth annual ACM symposium on Theory of computing (1983), ACM, pp. 1–9.
- [2] ARORA, S.; BARAK, B. Computational complexity: a modern approach. Cambridge University Press, 2009.
- [3] ATALLAH, M. Crc handbook of algorithms and theory of computation.
- [4] ATREYA, R.; MITTAL, N.; KSHEMKALYANI, A. D.; GARG, V. K.; SINGHAL, M. Efficient detection of a locally stable predicate in a distributed system. *Journal of Parallel and Distributed Computing* 67, 4 (2007), 369–385.
- [5] BARBOSA, V. C. Massively Parallel Models of Computation: Distributed Parallel Processing in Artificial Intelligence and Optimisation. Ellis Horwood, 1993.
- [6] BARBOSA, V. C. An Introduction to Distributed Algorithms. MIT Press, 1996.
- [7] BARBOSA, V. C. The Combinatorics of Resource Sharing. In Models for Parallel and Distributed Computation. Springer, 2002, pp. 27–52.
- [8] BARBOSA, V. C.; BENEVIDES, M. R.; OLIVEIRA FILHO, A. L. A priority dynamics for generalized drinking philosophers. *Information Processing Letters* 79, 4 (2001), 189–195.
- [9] BARBOSA, V. C.; BENEVIDES, M. R. F. A graph-theoretic characterization of AND-OR deadlocks. Tech. rep., UFRJ technical report COPPE-ES-472/98, Rio de Janeiro, Brazil, 1998.
- [10] BARNETT, J.; VERMA, T., ET AL. Intelligent reliability analysis. In Artificial Intelligence for Applications, 1994., Proceedings of the Tenth Conference on (1994), IEEE, pp. 428–433.
- [11] BOKAL, D.; BREŠAR, B.; JEREBIC, J. A generalization of hungarian method and hall's theorem with applications in wireless sensor networks. *Discrete Applied Mathematics* 160, 4 (2012), 460–470.
- [12] BRACHA, G.; TOUEG, S. Distributed deadlock detection. Distributed Computing 2, 3 (1987), 127–138.
- [13] BRZEZINSKI, J.; HELARY, J.-M.; RAYNAL, M.; SINGHAL, M. Deadlock models and a general algorithm for distributed deadlock detection. *Journal of parallel and distributed computing* 31, 2 (1995), 112–125.

- [14] CAO, T.; SANDERSON, A. C. And/or net representation for robotic task sequence planning. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 28, 2 (1998), 204–218.
- [15] CHANDY, K. M.; LAMPORT, L. Distributed snapshots: determining global states of distributed systems. ACM Transactions on Computer Systems (TOCS) 3, 1 (1985), 63-75.
- [16] CHANDY, K. M.; MISRA, J.; HAAS, L. M. Distributed deadlock detection. ACM Transactions on Computer Systems (TOCS) 1, 2 (1983), 144–156.
- [17] CIDON, I. An efficient distributed knot detection algorithm. Software Engineering, IEEE Transactions on 15, 5 (1989), 644–649.
- [18] COFFMAN, E. G.; ELPHICK, M.; SHOSHANI, A. System deadlocks. ACM Computing Surveys (CSUR) 3, 2 (1971), 67–78.
- [19] CONRADI, R.; WESTFECHTEL, B. Version models for software configuration management. ACM Computing Surveys (CSUR) 30, 2 (1998), 232–282.
- [20] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Introduction to Algorithms. MIT press, 2009.
- [21] COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. Distributed systems: concepts and design. pearson education, 2005.
- [22] CRISTIAN, F.; FETZER, C. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems 10*, 6 (1999), 642–657.
- [23] DATTA, A.; GHOSH, S. Synthesis of a class of deadlock-free petri nets. Journal of the ACM (JACM) 31, 3 (1984), 486–506.
- [24] DE MELLO, L. S. H.; SANDERSON, A. C. A correct and complete algorithm for the generation of mechanical assembly sequences. *Robotics and Automation, IEEE Transactions on 7*, 2 (1991), 228–240.
- [25] FANTI, M. P.; ZHOU, M. Petri net approaches to deadlock modeling and resolution in automated manufacturing. In Systems, Man and Cybernetics, 2002 IEEE International Conference on (2002), vol. 3, IEEE, pp. 6–pp.
- [26] FURST, M.; SAXE, J. B.; SIPSER, M. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory* 17, 1 (1984), 13–27.
- [27] GALČÍK, F.; KATRENIČ, J.; SEMANIŠIN, G. On computing an optimal semimatching. In *Graph-Theoretic Concepts in Computer Science* (2011), Springer, pp. 250–261.
- [28] GREENLAW, R.; HOOVER, H. J.; RUZZO, W. L. Limits to parallel computation: P-completeness theory, vol. 200. Oxford university press Oxford, 1995.
- [29] HALL, P. On representatives of subsets. J. London Math. Soc 10, 1 (1935), 26–30.

- [30] HOORY, S.; MAGEN, A.; PITASSI, T. Monotone circuits for the majority function. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer, 2006, pp. 410–425.
- [31] HOPCROFT, J. E.; KARP, R. M. An n⁵/2 algorithm for maximum matchings in bipartite graphs. SIAM Journal on computing 2, 4 (1973), 225–231.
- [32] JIMENEZ, P.; TORRAS, C. Speeding up interference detection between polyhedra. In Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on (1996), vol. 2, IEEE, pp. 1485–1492.
- [33] JIMÉNEZ, P.; TORRAS, C. An efficient algorithm for searching implicit and/or graphs with cycles. Artificial Intelligence 124, 1 (2000), 1–30.
- [34] KARP, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. Miller, J. Thatcher, and J. Bohlinger, Eds., The IBM Research Symposia Series. Springer US, 1972, pp. 85–103.
- [35] KATRENIC, J.; SEMANISIN, G. A generalization of hopcroft-karp algorithm for semimatchings and covers in bipartite graphs. *arXiv preprint arXiv:1103.1091* (2011).
- [36] KSHEMKALYANI, A. D.; SINGHAL, M. Efficient detection and resolution of generalized distributed deadlocks. *IEEE Transactions on Software Engineering* 20, 1 (1994), 43–54.
- [37] KUMAR, V.; KANAL, L. N. Parallel branch-and-bound formulations for and/or tree search. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 6 (1984), 768–778.
- [38] LABER, E. S. A randomized competitive algorithm for evaluating priced and/or trees. In STACS 2004. Springer, 2004, pp. 501–512.
- [39] LABER, E. S. A randomized competitive algorithm for evaluating priced and/or trees. Theoretical Computer Science 401, 1 (2008), 120–130.
- [40] MISRA, J.; CHANDY, K. M. A distributed graph algorithm: Knot detection. ACM Transactions on Programming Languages and Systems (TOPLAS) 4, 4 (1982), 678– 686.
- [41] MITCHELL, D. P.; MERRITT, M. J. A distributed algorithm for deadlock detection and resolution. In *Proceedings of the third annual ACM symposium on Principles of distributed computing* (1984), ACM, pp. 282–284.
- [42] MORABITO, R.; PUREZA, V. A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem. Annals of Operations Research 179, 1 (2010), 297–315.
- [43] NILSSON, N. J. Problem-solving methods in. Artificial Intelligence (1971).
- [44] PACHL, J. The deadlock problem in automatic railway operation. Signal und Draht. Vol. 89, no. 1-2 (1997).

- [45] PACHL, J. Deadlock avoidance in railroad operations simulations. In Transportation Research Board 90th Annual Meeting (2011), no. 11-0175.
- [46] PATERSON, M. S. Improved sorting networks witho (logn) depth. Algorithmica 5, 1-4 (1990), 75–92.
- [47] PENSO, L. D.; PROTTI, F.; RAUTENBACH, D.; DOS SANTOS SOUZA, U. Complexity analysis of p3-convexity problems on bounded-degree and planar graphs. *Theo*retical Computer Science 607 (2015), 83–95.
- [48] RAZBOROV, A. A. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes* 41, 4 (1987), 333–338.
- [49] RYANG, D.-S.; PARK, K. H. A two-level distributed detection algorithm of AND/OR deadlocks. Journal of Parallel and Distributed Computing 28, 2 (1995), 149–161.
- [50] SATYANARAYANA, B.; PRASAD, K. S. Discrete Mathematics and Graph Theory. PHI Learning Pvt. Ltd., 2014.
- [51] SIMON, R.; LEE, R. C. On the optimal solutions to and/or series-parallel graphs. Journal of the ACM (JACM) 18, 3 (1971), 354–372.
- [52] SINGHAL, M. Deadlock detection in distributed systems. Computer 22, 11 (1989), 37–48.
- [53] SIPSER, M. Introduction to the theory of computation, pws pub, 1997.
- [54] SMOLENSKY, R. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In Proceedings of the nineteenth annual ACM symposium on Theory of computing (1987), ACM, pp. 77–82.
- [55] SOUZA, U. D. S.; PINHEIRO, R. G. S.; MARTINS, I. C. Métodos heurísticos e exatos para busca de um subgrafo-solução ótimo de um grafo xy. Simpósio Brasileiro de Pesquisa Operacional XLVI (2014).
- [56] SOUZA, U. D. S.; PROTTI, F.; DA SILVA, M. D. Complexidade parametrizada para problemas em grafos E/OU. Pesquisa Operacional para o Desenvolvimento 4, 2 (2012), 160–174.
- [57] SOUZA, U. D. S.; PROTTI, F.; DA SILVA, M. D. Revisiting the complexity of and/or graph solution. Journal of Computer and System Sciences 79, 7 (2013), 1156–1163.
- [58] TANENBAUM, A. S.; VAN STEEN, M. Distributed systems: principles and paradigms, vol. 2. Prentice hall Englewood Cliffs, 2002.
- [59] TANENBAUM, A. S.; WOODHULL, A. S.; TANENBAUM, A. S.; TANENBAUM, A. S. Operating systems: design and implementation, vol. 2. Prentice-Hall Englewood Cliffs, NJ, 1987.
- [60] VALIANT, L. G. Short monotone formulae for the majority function. *Journal of Algorithms 5*, 3 (1984), 363–366.

- [61] VOLLMER, H. Introduction to circuit complexity: a uniform approach. Springer Science & Business Media, 2013.
- [62] WEYL, H. Almost periodic invariant vector sets in a metric vector space. American Journal of Mathematics (1949), 178–205.
- [63] WOODHULL, A. S.; TANENBAUM, A. S. Operating systems design and implementation, 1997.
- [64] YINGYING, D.; YAN, H.; JINGPING, J. Multi-robot cooperation method based on the ant algorithm. In Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE (2003), IEEE, pp. 14–18.