Resolução de Deadlocks: Complexidade e Tratabilidade Parametrizada

Alan Diêgo Aurélio Carneiro
Universidade Federal Fluminense - Instituto de Computação
aaurelio@ic.uff.br

Fábio Protti
Universidade Federal Fluminense - Instituto de Computação
fabio@ic.uff.br

Uéverton dos Santos Souza
Universidade Federal Fluminense - Instituto de Computação
usouza@ic.uff.br

RESUMO

Deadlocks em uma computação distribuída ocorrem quando um conjunto de processos espera indefinidamente por recursos provenientes do mesmo conjunto. Neste trabalho estudamos ações a serem tomadas após a detecção de deadlocks, mais precisamente, dado um grafo G representando uma computação distribuída em deadlock governada por um certo modelo de deadlock $\mathbb{M}$, investigamos a complexidade de problemas de deleção de vértices/arcos que visam obter um conjunto mínimo que, ao ser removido, torna G um grafo livre de deadlock. Primeiramente, é desenvolvido um estudo sobre a complexidade de problemas combinatórios relativos à resolução de deadlocks em grafos ponderados onde mostramos a equivalência com a resolução de deadlocks em grafos não ponderados. Posteriormente, provamos que deleção de vértices no modelo OU é Tratável por Parâmetro Fixo (FPT) quando perguntado se existe um subconjunto S de tamanho no máximo k onde G possui componentes fortemente conexas com tamanho limitado por r ou uma quantidade l de vértices com no máximo k vizinhos de saída. Finalmente, dois algoritmos FPT são apresentados.

PALAVRAS CHAVE. Computação Distribuída, Deadlock, FPT.

Área Principal: Teoria e Algoritmos em Grafos

ABSTRACT

A deadlock occurs in a distributed computation when a group of processes wait indefinitely for resources from each other. In this paper we study actions to be taken after deadlock detection, more precisely, given a graph G representing a deadlocked state of a distributed computation governed by a certain deadlock model $\mathbb{M}$, we investigate the complexity of vertex/arc deletion problems that aim at finding minimum vertex/arc subsets whose removal turns G into a deadlock-free graph. First, we develop a study of the complexity of combinatorial problems for weighted graphs where we show their equivalence with deadlock resolution problems on unweighted graphs. Furthermore, we prove that the vertex deletion problem in the OR model is Fixed-Parameter Tractable (FPT) when it is asked whether there is a subset S of size at most k such that either the size of the largest strongly connected component in G is bounded by r or G has exactly l vertices with k or less out-neighbors. Finally, two FPT algorithms are presented.

KEYWORDS. Distributed Computation. Deadlock. FPT.

Main Area: Theory and Algorithms on Graphs

1. Introdução

Sistemas distribuídos são sistemas de memória compartilhada [Barbosa, 1996] e compreendem um conjunto de processadores independentes (coleção de computadores autônomos) interligados por uma rede de comunicação que permita o compartilhamento de recursos. Tais processadores não compartilham fisicamente memória de forma direta; dessa forma, informações são necessariamente compartilhadas por trocas de mensagens através da rede de comunicação.

Os sistemas distribuídos são representados por grafos direcionados e seguem algum modelo como regra de troca de mensagens. Os modelos são divididos em síncronos e assíncronos [Barbosa, 1993]. No modelo síncrono [Tanenbaum e Van Steen, 2007] há um relógio com tempo global a todos os processadores, e um limite superior para a troca de mensagens entre pares de processos. Já no modelo assíncrono [Chandy et al., 1983; Cristian e Fetzer, 1999; Atreya et al., 2007] não há quaisquer tipos de restrições temporais entre as trocas de mensagens, não há relógio em comum, e não há memória compartilhada.

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, uma vez que o estudo é direcionado a deadlocks, que é uma propriedade estável destes modelos de sistemas. Uma propriedade é dita estável se, existindo para um tempo Ψ, também existirá em qualquer tempo subsequente a Ψ.

Deadlock é um fenômeno comum a sistemas onde acontece compartilhamento de recursos, como: sistemas operacionais [Tanenbaum et al., 1987; Woodhull e Tanenbaum, 1997]; cruzamentos de trânsito [Coffman et al., 1971]; linhas férreas [Pachl, 1997, 2011]; cooperação multi-robôs [Yingying et al., 2003]; dentre outros exemplos.

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. O objetivo deste trabalho é estudar problemas combinatórios relativos à resolução de deadlocks em sistemas distribuídos; para tal, consideraremos os grafos de espera de um sistema distribuído, definidos a seguir.

1.1. Grafos de Espera

Barbosa e Benevides [Barbosa e Benevides, 1998] definem grafos de espera como estruturas de análise e abstração de sistemas distribuídos. Os grafos de espera são dinâmicos, ou seja, mudam de acordo com as trocas de requisições e respostas do sistema. Novamente podemos desconsiderar alguns aspectos, e trabalhar apenas com o grafo em um instante específico de tempo (chamado de *snapshot*), visto que deadlocks existentes no grafo nesse instante implicam na sua existência em instantes subsequentes.

Definiremos 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 existe de um processo v_i para v_j se e somente se v_i fez a uma solicitação a v_j que não foi atendida até o instante corrente. Para cada processo $v_i \in V$, denotaremos por $O_i \subseteq V$ o conjunto de vértices dos quais v_i aguarda resposta.

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 quatro modelos clássicos de espera na literatura [Brzezinski et al., 1995; Kshemkalyani e Singhal, 1994]:

Modelo E: Um processo v_i torna-se executável quando toda requisição (v_i, v_j) é atendida, isto é, as requisições (v_i, v_j) foram satisfeitas, sendo portanto removidas do grafo de espera corrente, o que torna v_i um vértice sumidouro. Este modelo (Figura 1) é caracterizado por situações onde a conjunção de respostas de todas as requisições são necessárias para execução de v_i.

Modelo OU: Neste modelo, para tornar um processo v_i executável, basta que uma requisição (v_i, v_j) seja atendida; neste caso todas as demais requisições são desconsideradas, tornando v_i um vértice sumidouro. O modelo (Figura 2) é caracterizado por situações de característica disjuntiva onde apenas uma resposta é necessária para v_i.

Modelo E-OU: Neste modelo, existem $t_i \geq 1$ subconjuntos de O_i associados a v_i. Tais conjuntos são denotados por $O_i^1, \cdots, O_i^{t_i}$ onde $O_i = O_i^1 \cup \cdots \cup O_i^{t_i}$. Para que o processo v_i se torne executável, basta que todas as requisições (v_i, v_j) de pelo menos um subconjunto de O_i sejam atendidas. Neste caso todas as demais requisições são desconsideradas, tornando v_i um vértice sumidouro. Esse modelo (Figura 3) é caracterizado por processos v_i em que a conjunção de recursos recebida por cada grupo O_i é equivalente.

Modelo X-de-Y: Neste modelo, para tornar um processo v_i executável, basta que x_i requisições (v_i, v_j) sejam atendidas, e neste caso todas as demais requisições são desconsideradas, tornando v_i um vértice sumidouro. Este modelo (Figura 4) é caracterizado por situações onde v_i requisita mais do que ele realmente precisa e espera somente x_i requisições.

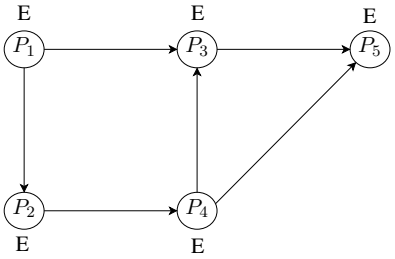

Figura 1: Grafo de Espera: Modelo E.

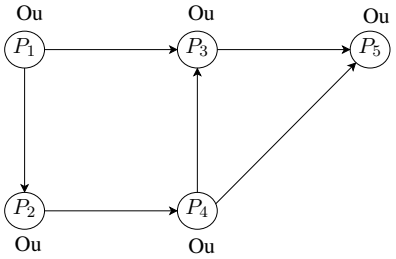

Figura 2: Grafo de Espera: Modelo OU.

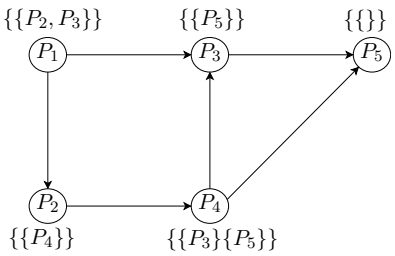

Figura 3: Grafo de Espera: Modelo E-OU.

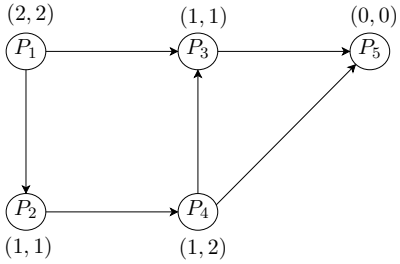

Figura 4: Grafo de Espera: Modelo X-de-Y.

Grafos de espera com pesos em sistemas distribuídos indicam o grau de prioridade de um processo ou requisição. A utilização de prioridade em processos ou requisições é amplamente utilizada no contexto da computação distribuída [Kanrar e Chaki, 2010; Choudhary et al., 1989] com aplicações em balanceamento de carga, escalonamento do uso de CPU, entre outros. Utilizaremos $P(x)$ para denotar o peso de um arco ou vértice x.

1.2. Deadlocks

Uma condição necessária para o surgimento de deadlocks, segundo [Barbosa, 1996], é a existência de ciclos no grafo de espera associado ao sistema. Nós em deadlock sempre estão à espera de pelo menos um nó pertencente a algum ciclo. Mais especificamente, deadlocks ocorrem se, e somente se, o grafo de espera contém um knot. Um knot é definido como uma estrutura genérica minimal que caracteriza o deadlock.

Embora o princípio básico de um knot independa do modelo, sua caracterização apresenta distinções de acordo com o modelo de espera do grafo em análise; sendo assim, denominamos E-

knot, OU-knot, (E-OU)-knot e (X-Y)-knot os knots associados aos modelos E, OU, E-OU e X-de-Y, respectivamente [Barbosa, 2002]:

E-Knot: Deadlock em um grafo G no modelo E existe se, e somente se, G contém um ciclo; ciclos são chamados E-Knots.

OU-Knot: Deadlock em um grafo G no modelo OU existe se, e somente se, G contém uma componente fortemente conexa C 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.

(E-OU)-Knot: Deadlock em um grafo G no modelo E-OU existe se, e somente se, G contém 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'.

(X-Y)-Knot: Um deadlock em um grafo G no modelo E/Ou existe se, e somente se, G contém um subgrafo G' fortemente conexo tal que para cada vértice $v_i \in V(G')$, pelo menos $(y_i - x_i + 1)$ nós do conjunto O_i também pertencem a G'.

Segundo Singhal [1989], existem três abordagens principais para o tratamento de deadlocks:

1. **Prevenir a Ocorrência de Deadlocks:** Consiste em identificar uma condição C tal que $\exists\ Deadlock \rightarrow C$. Uma vez identificada C (que é necessária para que ocorra deadlock), basta proibir a ocorrência de C para que sejam **evitados** os deadlocks.

2. **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.

3. **Detectar a Ocorrência de Deadlocks:** Consiste em identificar uma condição C tal que $C \rightarrow \exists\ Deadkock$. Uma vez identificada C (que é suficiente para que haja deadlock), basta verificar a ocorrência de C para **detectar** os deadlocks.

Adotaremos para este trabalho a terceira, na qual permite-se que o deadlock ocorra, e, como deadlock é uma propriedade estável em sistemas distribuídos, uma vez que ele ocorre, é necessária uma intervenção externa para solucioná-lo. Quando nos referimos à resolução de deadlock, considera-se que o deadlock foi previamente detectado.

2. A Classe de Problemas P–λ–DELETION($\mathbb{M}$)

Definimos P–λ–DELETION($\mathbb{M}$) como um problema de otimização genérico para resolução de deadlocks em grafos de espera com prioridades, isto é, a prioridade de um processo ou requisição é dada por um peso. O parâmetro λ indica o tipo de deleção a ser utilizada para tornar o grafo de entrada G livre de deadlock, e $\mathbb{M} \in$ {E, OU, X-DE-Y, E-OU} indica o modelo de deadlock do grafo de espera G. As operações de deleção consideradas neste trabalho são dadas abaixo:

1. **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 um conjunto de vértices de custo mínimo 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. **Saídas:** A intervenção consiste em 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 um conjunto de vértices de custo mínimo a serem transformados em sumidouros de forma a tornar G livre de deadlock.

A complexidade computacional de doze problemas combinatórios em grafos sem pesos nas arestas são apresentadas em [Carneiro et al., 2015]. Assim, já que grafos sem pesos são um caso particular dos grafos com pesos, podemos ver na Tabela 1 que apenas dois problemas, P–ARCO–DELETION(OU) e P–SAÍDAS–DELETION(OU) permanecem em aberto.

P–λ–DELETION($\mathbb{M}$)

	E	OU	E-OU	X-de-Y
Arco	NP-D	?	NP-D	NP-D
Vértice	NP-D	NP-D	NP-D	NP-D
Saídas	NP-D	?	NP-D	NP-D

Tabela 1: Complexidade parcial para P–λ–DELETION($\mathbb{M}$).

3. Complexidade Computacional

Em [Carneiro et al., 2015], são apresentados algoritmos polinomiais para ARCO–DELETION(OU) e SAÍDAS–DELETION(OU). Esta sessão tem como principal foco mostrar que a resolução de deadlock no modelo OU, tendo como entrada grafos com pesos, é equivalente à resolução de deadlock em grafos sem pesos, o que mostra que os problemas ARCO–DELETION(OU) e SAÍDAS–DELETION(OU) permanecem polinomiais mesmo para grafos com pesos. Já que λ–DELETION($\mathbb{M}$) é um caso particular de P–λ–DELETION($\mathbb{M}$), as demonstrações a seguir salientam reduções de problemas em grafos com pesos G para grafos sem pesos G'. É considerado que os pesos são polinomiais em relação ao tamanho da entrada.

3.1. Prioridade de Processos

Iniciaremos a análise com grafos de espera com pesos em vértices, isto é, uma computação distribuída onde os vértices têm prioridades. Tais grafos afetam diretamente operações "Vértice" e "Saídas" e são indiferentes em relação à operação "Arco".

Teorema 3.1. *Seja G uma instância do problema P–λ–DELETION(OU). Podemos reduzir G em tempo polinomial a uma instância G' de λ–DELETION(OU) se para todo vértice $v_i \in G$, $P(v_i) = poly(n)$.*

Demonstração. Provaremos que P–λ–DELETION(OU) $\propto$ λ–DELETION(OU). Seja G uma instância de P–λ–DELETION(OU), isto é, um grafo de espera em um modelo OU tal que os vértices possuem prioridade/custo associado. Construímos a partir de G um grafo G' tal que G' possui um conjunto de vértices S' de cardinalidade k que ao serem removidos o tornam livre de deadlock se, e somente se, G possui um conjunto de vértices S tal que $\sum_{i=1}^{|S|} P(v_i) = k$, e, ao serem removidos, tornam G livre de deadlock. A construção é descrita a seguir:

1. Inicie G' com uma cópia de G, isto é, para todo vértice $v_i \in V(G)$, existe um vértice $v_i \in V(G')$, e, para toda aresta $(v_iv_j) \in E(G)$, existe uma aresta $(v_iv_j) \in E(G')$.

2. Para cada vértice $v_i \in V(G)$ que possui custo $p = P(v_i)$ é aplicado o gadget descrito a seguir (Figura 5). Crie $2(p-1)$ vértices $\{Wv_i^1, Uv_i^1, Wv_i^2, Uv_i^2, \ldots, Wv_i^{p-1}, Uv_i^{p-1}\}$ tais que todo par de vértices Wv_i^j, Uv_i^j forme um ciclo direcionado.

3. Para cada vértice Uv_i^j em G' crie uma aresta direcionada a v_i.

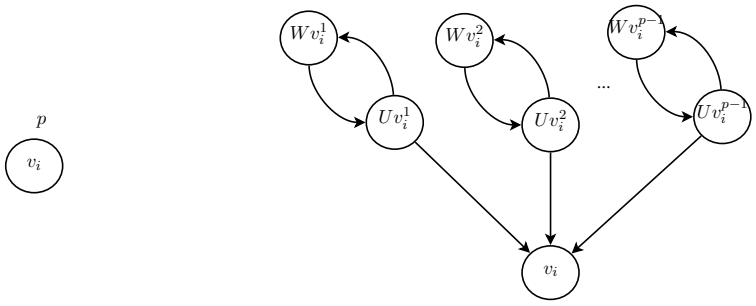

Figura 5: Vértice $v_i \in G$ com prioridade p e subgrafo induzido pelo gadget em G'.

Suponha que G possui um conjunto de vértices S tal que $\sum_{i=1}^{|S|} P(v_i) = k$ e $G[V \setminus S]$ é livre de deadlock. Podemos construir um conjunto de vértices S' com cardinalidade k em G', de forma que $G'[V' \setminus S']$ seja livre de deadlock. A construção do conjunto S' é dada a seguir. Inicia-se com $S' = S$; após isso, para cada vértice v_i em S e $p = P(v_i)$, inclua em S', o conjunto de vértices $\{Uv_i^1, Uv_i^2, \ldots, Uv_i^{p-1}\}$. É fácil ver que o conjunto S' possui cardinalidade k. Desta forma, já que $G[V \setminus S]$ é livre de deadlock, e nenhuma condição de espera adicional é atribuída a um vértice v_i ou qualquer vértice alcançável por v_i, temos que ou o vértice é removido ou em algum momento será liberado em $G'[V' \setminus S]$. Dessa forma, restam em deadlock no grafo $G'[V' \setminus S]$ apenas os ciclos direcionados dos pares de vértices Wv_i, Uv_i. Como cada ciclo Wv_i, Uv_i para o qual $v_i \notin S'$ possui caminho a v_i, garantidamente o par Wv_i, Uv_i será liberado por v_i, caso contrário $G[V \setminus S]$ não é livre de dedlock. Para um ciclo Wv_i, Uv_i tal que $v_i \in S'$, Uv_i também é removido, convertendo assim Wv_i em sumidouro. Portanto $G'[V' \setminus S']$ é livre de deadlock.

Por outro lado, suponha que G' possui um conjunto de vértices S' de cardinalidade k tal que $G'[V' \setminus S']$ é livre de deadlock. Podemos construir um conjunto de vértices S tal que $\sum_{i=1}^{|S|} P(v_i) = k$, de forma que $G[V \setminus S]$ é livre de deadlock. Seja H o conjunto de vértices $\{Wv_i^1, Uv_i^1, \ldots, Wv_i^{p-1}, Uv_i^{p-1}\}$ criados em G' a partir de cada vértice v_i em G. Como $G'[V' \setminus S']$ é livre de deadlock e por construção nenhum vértice em $G'' = G'[V' \setminus H]$ alcança vértices de H, ao fazer $S = S' \setminus H$ temos que $G''[V'' \setminus S]$ é livre de deadlock. Já que $G''[V'' \setminus S]$ é livre de deadlock, $G[V \setminus S]$ também será. Finalmente provaremos que $\sum_{i=1}^{|S|} P(v_i) = k$. Note que, em G', todo vértice do tipo v_i removido gera $P(v_i) - 1$ novos ciclos direcionados com dois vértices. Sendo assim, exatamente uma remoção em cada ciclo se torna necessária e suficiente. Portanto, para cada vértice v_i, $P(v_i) - 1$ vértices adicionais estarão em S'. Logo, cada vértice v_i implica em $P(v_i)$ vértices em S', onde $\sum_{i=1}^{|S|} P(v_i) = k$. □

3.2. Prioridade de Requisições

Mostraremos que P–ARCO–DELETION(OU) é equivalente a ARCO–DELETION(OU) e portanto solucionável em tempo polinomial. Já que o algoritmo para solucionar deadlock proposto por [Carneiro et al., 2015] se baseia apenas na quantidade de arestas de saída de um potencial vértice a ser transformado em sumidouro, uma abordagem simples seria permitir um multigrafo como entrada para o problema ARCO–DELETION(OU); entretanto, a utilização de multigrafos para representação de sistemas não é abordada na literatura. Isto posto, apresentaremos um procedimento que reduz um grafo com pesos a um grafo sem pesos nas arestas. A redução é formalmente apresentada a seguir.

Teorema 3.2. *Seja G uma instância do problema P–ARCO–DELETION(OU). Podemos reduzir G em tempo polinomial a uma instância G' de λ–DELETION(OU) se para toda aresta $e = (v_i v_j) \in G$, $P(e) = poly(m)$.*

Demonstração. Construímos a partir de G um grafo G' tal que G' possui um conjunto de vértices S' de cardinalidade k que ao serem removidos tornam G' livre de deadlock se, e somente se, G possui um conjunto de vértices S tal que $\sum_{i=1}^{|S|} P(v_i) = k$ e, ao serem removidos, tornam G livre de deadlock. A construção é descrita a seguir:

1. Para cada vértice v_i em G crie um vértice v_i em G'.
2. Para cada aresta $e = (v_i v_j)$:
   (a) Se $P(e) = 1$ crie uma aresta $(v_i v_j)$ em G'.
   (b) Caso contrário, crie um subgrafo completo com $P(e)+2$ vértices. Crie $P(e)-1$ arestas de v_i a vértices do subgrafo completo. Crie uma aresta de um vértice do subgrafo completo a v_j (Figura 6).

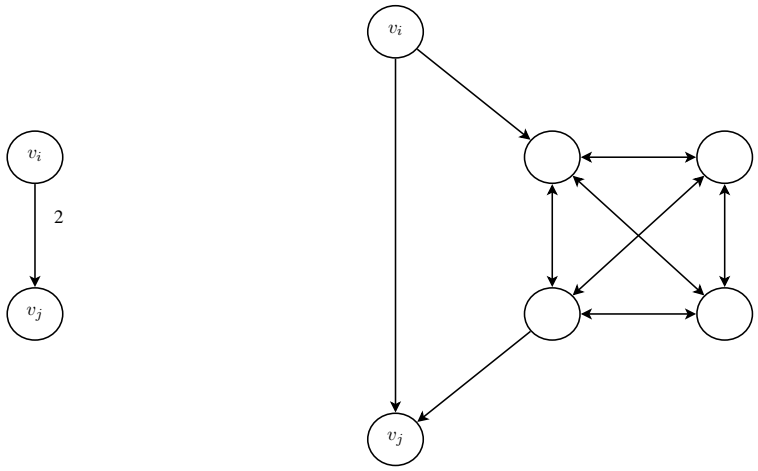

Figura 6: Aresta $(v_i, v_j) \in G$ com prioridade p e subgrafo induzido pelo gadget em G'.

O restante da prova, por ser análoga à do Teorema 3.1, será deixada ao leitor. □

Dos Teoremas 3.1 e 3.2, podemos nos estender aos modelos que englobam o modelo OU, e finalmente podemos provar a equivalência entre as classes de problemas λ–DELETION($\mathbb{M}$) e P–λ–DELETION($\mathbb{M}$).

Corolário 3.3.

a) Seja G' uma instância do problema λ–DELETION($\mathbb{M}$). Podemos reduzir G em tempo polinomial a uma instância G' de P–λ–DELETION($\mathbb{M}$) para grafos com pesos em vértices ou arestas.

b) Seja G uma instância do problema P–λ–DELETION(E-OU). Podemos reduzir G em tempo polinomial a uma instância G' de λ–DELETION(E-OU) se para todo vértice $v_i \in G$, $P(v_i) = poly(n)$.

c) Seja G uma instância do problema P–λ–DELETION(E-OU). Podemos reduzir G em tempo polinomial a uma instância G' de λ–DELETION(E-OU) se para todo vértice $v_i \in G$, $P(v_i) = poly(n)$.

d) Seja G uma instância do problema P–λ–DELETION(X-DE-Y). Podemos reduzir G em tempo polinomial a uma instância G' de λ–DELETION(X-DE-Y) se para todo vértice $v_i \in G$, $P(v_i) = poly(n)$.

e) Seja G uma instância do problema P–λ–DELETION(X-DE-Y). Podemos reduzir G em tempo polinomial a uma instância G' de λ–DELETION(X-DE-Y) se para todo vértice $v_i \in G$, $P(v_i) = poly(n)$.

Demonstração.

a) Trivial.

b) A construção do grafo G' segue diretamente da construção utilizada no Teorema 3.1. Adicionalmente, os vértices Uv_i^j terão os rótulos $\{\{v_i\}\{Wv_i^j\}\}$. O restante da prova de (b) segue diretamente do Teorema 3.1.

c) Análogo a (b).

d) A construção do grafo G' segue diretamente da construção utilizada no Teorema 3.1. Adicionalmente, os vértices Uv_i^j terão os rótulos $x = 1$. O restante da prova de c) segue diretamente do Teorema 3.1.

e) Análogo a (d).

□

Neste ponto, como resultado direto dos Teoremas 3.1 e 3.2 e do Corolário 3.3, completamos as complexidades em aberto da Tabela 1. Os resultados são apresentados a seguir.

P–λ–DELETION($\mathbb{M}$)				
	E	OU	E-OU	X-de-Y
Arco	NP-D	P	NP-D	NP-D
Vértice	NP-D	NP-D	NP-D	NP-D
Saídas	NP-D	P	NP-D	NP-D

Tabela 2: Complexidade computacional para P–λ–DELETION($\mathbb{M}$).

4. Tratabilidade Parametrizada para VERTEX–DELETION(OU)

A Teoria da Complexidade Parametrizada foi proposta e desenvolvida por Downey e Fellows [Rodney e Michael, 1999; Flum e Grohe, 2006; Niedermeier, 2006] e surgiu como alternativa promissora para se trabalhar com problemas NP-difíceis. Tais problemas podem ser descritos da seguinte forma: dado um objetivo x e um inteiro não negativo k, "x tem alguma propriedade que depende de k?". Na complexidade parametrizada, k é chamado de parâmetro. O interesse em tais parâmetros se deve ao fato de que, em muitos casos, somente uma pequena faixa de valores é realmente importante na prática. Logo, o parâmetro k pode ser considerado pequeno em comparação com o tamanho de x. Sendo assim, a intratabilidade (aparente) desses problemas no caso geral pode ser indevidamente pessimista [dos Santos e Souza, 2015].

Nessa seção apresentaremos dois algoritmos cuja complexidade exponencial depende apenas de certos aspectos da entrada. Tais algoritmos são denominados *Tratáveis por Parâmetro Fixo* (ou simplesmente algoritmos FPT). Ambos os algoritmos são para o problema VERTEX–DELETION(OU) considerando dois parâmetros em cada algoritmo.

4.1. Tamanho da Maior Componente Fortemente Conexa

Utilizamos a técnica de árvores de busca de altura limitada. Tal método tem sido amplamente utilizado e fornece poderosos algoritmos parametrizados. A técnica consiste em tentar criar uma solução factível ao problema fazendo uma série de decisões, que tipicamente marcam, removem ou rotulam elementos de algum conjunto ou estrutura a cada chamada recursiva reduzindo o tamanho da instância até esta ter uma resposta facilmente computável ou até mesmo trivial.

O problema VERTEX–DELETION(OU) foi provado NP-Difícil em [Carneiro et al., 2015]. A prova utiliza uma transformação do problema 3-SAT e o grafo resultante da transformação é um grafo onde as componentes fortemente conexas tem tamanho máximo 3. Posteriormente, em [Carneiro et al., 2016], é mostrado que VERTEX–DELETION(OU) é NP difícil mesmo para grafos planares, bipartidos, com $\Delta(G) = 4$ e $\Delta^+(G) = 2$. A prova utiliza uma transformação do problema 3-SAT, e, o grafo resultante da transformação é um grafo no qual as componentes fortemente conexas possuem tamanho máximo 6. Portanto exploramos a seguir o parâmetro r que corresponde ao tamanho da maior componente fortemente conexa no grafo de entrada. Definimos formalmente o problema a seguir:
VERTEX–DELETION(OR, r, k)
Entrada: Um grafo de espera $G = (V, E)$ no modelo OR, um par de inteiros positivos r e k.
Parâmetros: r e k.
Pergunta: Existe um conjunto de vértices $S \subseteq V$ de tamanho no máximo k tal que $G[V \setminus S]$ é um grafo livre de knots?

Seja (G, r, k) uma instância do problema VERTEX–DELETION(OU). Em cada passo da árvore de busca, testaremos por "branching" a remoção de todos os vértices no knot com menor tamanho. Nosso algoritmo é baseado nas seguinte observações:

i) Todos os knots em um grafo G podem ser encontrados e enumerados em tempo linear.

ii) O menor número possível de remoções para livrar um grafo de knots é equivalente ao número de knots no grafo, isto é, no mínimo uma remoção em cada knot de G deve ocorrer para torná-lo livre de deadlock.

O Algoritmo 1, abaixo da raiz, em cada subárvore procede com o parâmetro $k - 1$, e o algoritmo recursivo cria no máximo r chamadas com o parâmetro k novamente reduzido em uma unidade. O padrão se repete até que $k \leq 0$ ou um conjunto de vértices de tamanho máximo k (que pode ser obtido subindo a árvore de recursão) seja encontrado. Já que encontrar um knot em G (linha 5) e verificar se o grafo é livre de knots (linha 7) podem ser executadas em tempo linear $O(n)$, o Algoritmo 1 é executável em tempo $O(r^k n)$.

4.2. Analisando o Grau dos vértices

Também em [Carneiro et al., 2015] é apresentado um algoritmo polinomial para VERTEX–DELETION(OU), e como mencionado na seção anterior, o problema é NP-Difícil mesmo quando $\Delta(G) = 4$ e $\Delta^+(G) = 2$. Portanto a seguinte pergunta torna-se natural: dado um grafo de espera G no modelo OU, pode-se com no máximo k remoções livrar G de knots caso G tenha l vértices com grau máximo de saída k? Definimos o problema a seguir.
VERTEX–DELETION(OU, l, k)
Entrada: Um grafo de espera $G = (V, E)$ no modelo OU, um par de inteiros positivos l e k.
Parâmetros: l e k.
Pergunta: Existe um conjunto de vértices $S \subseteq V$ de tamanho no máximo k tal que $G[V \setminus S]$ é um grafo livre de knots?

Propomos um Algoritmo para tal problema que utiliza uma observação adicional às utilizadas no Algoritmo 1:

Algoritmo 1: VERTEX–DELETION(G, r, k)

```
1  inicio
2      se k ≤ 0 então
3          retornar "não";
4      fim se
5      Q ← Conjunto de vértices do menor knot de G;
6      para cada vértice v_i ∈ Q hacer
7          se G − v_i é livre de knots então
8              retornar "sim";
9          senão
10             VERTEX–DELETION(G − v_i, r, k − 1);
11         fim se
12     fin para cada
13 fin
```

iii) É necessária a criação de sumidouros através de remoção de vértices. Portanto, para converter um vértice promissor em sumidouro, requer-se a remoção de todos os seus vértices de saída.

Considerando (iii) e tendo que o conjunto de vértices a ser removido deve ser limitado por k, podemos então encontrar um conjunto de vértices H tal que todo vértice $v_i \in H$ satisfaz $\delta^+(v_i) \leq k$. Assim enumera-se todos os subconjuntos de H, e testa-se para todos os subconjuntos $H' \subseteq H$ tal que $H' = \{v_1, v_2, \ldots, v_z\}$ se $\sum_{j=0}^{z} \delta^+(v_j) \leq k$ (conjunto de vértices promissores). O conjunto de vértices S a ser removido é criado a partir dos subconjuntos H': para cada vértice $v_j \in H'$, incluímos em S a vizinhança de v_j. Finalmente testa-se exaustivamente todos os conjuntos S obtidos por subconjuntos H' tal que $G[V \setminus S]$ é livre de knots. O algoritmo é dado a seguir.

Algoritmo 2: VERTEX–DELETION(G, l, k)

```
1  inicio
2      se l ≤ 0 ou k ≤ 0 então
3          retornar "não";
4      fim se
5      H ← Conjunto de vértices v_i tal que δ+(v_i) ≤ k;
6      para cada Subconjunto H' ⊆ H hacer
7          se Σ_{j=0}^{z} δ(v_j) ≤ k então
8              S ← N+(v_j) ∀ v_j ∈ H';
9              se G[V \ S] é livre de knots então
10                 retornar "sim";
11             fim se
12         fim se
13     fin para cada
14     retornar "não";
15 fin
```

A corretude do Algoritmo 2 se dá pelo fato de que, ao testar exaustivamente todos os conjuntos obtidos por vértices promissores, são testados exaustivamente todos os conjuntos de vértices de tamanho k que podem gerar sumidouros. Obter o conjunto de vértices promissores (linha 5) pode ser feito em tempo linear $O(n)$. Enumerar todos os subconjuntos de H (linha 6) pode ser feito em tempo 2^l. Para todos os subconjuntos de vértices promissores são criados conjuntos S

de tamanho limitado por k e testa-se se $G[V \setminus S]$ é livre de deadlock (as operações nas linhas 7 a 9 podem ser executadas em tempo $O(n+m)$). Portanto o Algoritmo 2 é executável em tempo $O(2^l(n+m)+n)$.

5. Conclusões

Neste trabalho definimos a classe P–λ–DELETION($\mathbb{M}$). Primeiramente, mostramos que P–λ–DELETION($\mathbb{M}$) é equivalente a λ–DELETION($\mathbb{M}$), provando assim a complexidade de todos os doze problemas (Tabela 3). Ainda, como resultado direto dos Teoremas 3.1 e 3.2 e Corolário 3.3, algoritmos polinomiais tanto quanto algoritmos que abordem a intratabilidade dos problemas na versão sem pesos podem ser igualmente utilizadas em grafos com pesos e vice-versa.

P–λ–DELETION($\mathbb{M}$)				
	E	OU	E-OU	X-de-Y
Arco	NP-D	P	NP-D	NP-D
Vértice	NP-D	NP-D	NP-D	NP-D
Saídas	NP-D	P	NP-D	NP-D

Tabela 3: Complexidade computacional para P–λ–DELETION($\mathbb{M}$).

Finalmente, estudamos com mais profundidade o problema VÉRTICE–DELETION(OU). O problema de forma genérica foi provado NP-Difícil por [Carneiro et al., 2015]. O grafo resultante da redução possui apenas componentes fortemente conexas pequenas e, portanto, um parâmetro promissor a um algoritmo FPT. O Algoritmo 1 é executável em tempo $O(r^k n)$ e utiliza como parâmetro k e r, onde r indica o tamanho da maior componente fortemente conexa. Uma das abordagens de tratamento de deadlock mais utilizadas é não permitir que ciclos se formem no grafo; dessa forma, independente do modelo utilizado, um deadlock jamais existirá. Tendo em mãos um algoritmo que resolve deadlock de forma exponencial em relação a uma característica controlada permite uma prevenção de deadlock mais permissiva, isto é, pode-se proibir apenas a existência de ciclos maiores que r, garantindo assim que a maior componente fortemente conexa terá tamanho r, e sempre que um deadlock for detectado pode ser tratado em tempo viável. Na literatura, em [Carneiro et al., 2016] é apresentado um algoritmo polinomial para grafos subcúbicos, e é provado que para grafos com grau máximo 4 o problema correspondente é NP-Completo. No Algoritmo 2, executável em tempo $O(2^l(n+m)+n)$, exploramos como parâmetro k e l, que indica a quantidade de vértices com grau de saída menor que k. Tal algoritmo se comporta bem para grafos densos, também tendo aplicação em prevenção de deadlock; porém é necessário o controle global da quantidade de vértices com grau máximo k.

Referências

Atreya, R., Mittal, N., Kshemkalyani, A. D., Garg, V. K., e Singhal, M. (2007). Efficient detection of a locally stable predicate in a distributed system. *Journal of Parallel and Distributed Computing*, 67(4):369–385.

Barbosa, V. C. (1993). *Massively Parallel Models of Computation: Distributed Parallel Processing in Artificial Intelligence and Optimisation*. Ellis Horwood.

Barbosa, V. C. (1996). *An Introduction to Distributed Algorithms*. MIT Press.

Barbosa, V. C. (2002). The Combinatorics of Resource Sharing. In *Models for Parallel and Distributed Computation*, p. 27–52. Springer.

Barbosa, V. C. e Benevides, M. R. F. (1998). A graph-theoretic characterization of AND-OR deadlocks. Technical report, UFRJ technical report COPPE-ES-472/98, Rio de Janeiro, Brazil.

Brzezinski, J., Helary, J.-M., Raynal, M., e Singhal, M. (1995). Deadlock models and a general algorithm for distributed deadlock detection. *Journal of parallel and distributed computing*, 31 (2):112–125.

Carneiro, A. D. A., Protti, F., e dos Santos Souza, U. (2015). Complexidade de resolução de deadlocks em grafos de espera de sistemas distribuídos. *XLVII Simpósio Brasileiro de Pesquisa Operacional*.

Carneiro, A. D. A., Protti, F., e dos Santos Souza, U. (2016). Algoritmos para resolução de deadlocks em grafos subcúbicos. *XLVIII Simpósio Brasileiro de Pesquisa Operacional*.

Chandy, K. M., Misra, J., e Haas, L. M. (1983). Distributed deadlock detection. *ACM Transactions on Computer Systems (TOCS)*, 1(2):144–156.

Choudhary, A. N., Kohler, W. H., Stankovic, J. A., e Towsley, D. (1989). A modified priority based probe algorithm for distributed deadlock detection and resolution. *IEEE Transactions on Software Engineering*, 15(1):10–17.

Coffman, E. G., Elphick, M., e Shoshani, A. (1971). System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78.

Cristian, F. e Fetzer, C. (1999). The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657.

dos Santos, V. F. e Souza, U. d. S. (2015). Uma introdução à complexidade parametrizada.

Flum, J. e Grohe, M. (2006). Parameterized complexity theory, volume xiv of texts in theoretical computer science. an eatcs series.

Kanrar, S. e Chaki, N. (2010). Fapp: A new fairness algorithm for priority process mutual exclusion in distributed systems. *JNW*, 5(1):11–18.

Kshemkalyani, A. D. e Singhal, M. (1994). Efficient detection and resolution of generalized distributed deadlocks. *IEEE Transactions on Software Engineering*, 20(1):43–54.

Niedermeier, R. (2006). Invitation to fixed-parameter algorithms.

Pachl, J. (1997). The deadlock problem in automatic railway operation. *Signal und Draht. Vol. 89, no. 1-2*.

Pachl, J. (2011). Deadlock avoidance in railroad operations simulations. In *Transportation Research Board 90th Annual Meeting*, number 11-0175.

Rodney, G. D. e Michael, R. (1999). Fellows: Parameterized complexity. *Monogr. Comput. Sci. Springer, New York*, 87.

Singhal, M. (1989). Deadlock detection in distributed systems. *Computer*, 22(11):37–48.

Tanenbaum, A. e Van Steen, M. (2007). *Distributed Systems*. Pearson Prentice Hall.

Tanenbaum, A. S., Woodhull, A. S., Tanenbaum, A. S., e Tanenbaum, A. S. (1987). *Operating systems: design and implementation*, volume 2. Prentice-Hall Englewood Cliffs, NJ.

Woodhull, A. S. e Tanenbaum, A. S. (1997). Operating systems design and implementation.

Yingying, D., Yan, H., e Jingping, J. (2003). Multi-robot cooperation method based on the ant algorithm. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, p. 14–18. IEEE.