Algoritmos para Resolução de Deadlocks em Grafos Subcúbicos

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

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 são fornecidos algoritmos eficientes para a para o problema Deadlock-Resolution(Ou, Vértice) em grafos subcúbicos.

PALAVRAS CHAVE. Computação Distribuída, Deadlock, Grafos Subcúbicos.

Área Principal: Teoria e Algoritmos em Grafos

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. In this work we provide efficient algorithms for subcubic graphs on the Deadlock-Resolution(Or, Vertex) problem.

KEYWORDS. Distributed Computation. Deadlock. Subcubic Graph.

Main Area: Theory and Algorithms on Graphs

1. Introdução

Segundo a definição em [Tanenbaum e Van Steen, 2002], um sistema distribuído é uma "coleção de computadores independentes que se apresenta ao utilizador como um sistema único e consistente". Outra definição, em [Coulouris et al., 2005], 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".

Uma 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 [Coulouris et al., 2005]. 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* [Fanti e Zhou, 2002].

2. O Conceito de Deadlock

Deadlock é um problema fundamental em projeto de sistemas concorrentes [Kshemkalyani e Singhal, 1994]. Apesar de este problema ser amplamente estudado em sistemas de memória compartilhada [Coffman et al., 1971; Datta e Ghosh, 1984], 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 [Barbosa, 2002].

Atreya et al. [2007] apontam o deadlock como propriedade estável em uma computação distribuída. Uma propriedade é dita estável se, existindo para um estado global Ψ, também existirá em qualquer estado global subsequente a Ψ. O 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.

2.1. Grafos de Espera

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*) [Chandy e Lamport, 1985]. Sempre que posteriormente nos referirmos a um grafo de espera G, este corresponderá a um grafo estático referente a um estado global consistente.

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. Denotaremos $deg^+(v)$ como grau de saída de um vértice v por e $deg^+(v)$ como grau de entrada. O_i denota o conjunto de vértices ao qual v_i espera algum sinal, isto é, existe uma aresta $v_i v_j$ para todo v_j em O_i.

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. Neste trabalho, estudaremos apenas o modelo Ou.

Modelo Ou: 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 é caracterizado por situações de característica disjuntiva onde apenas uma resposta é necessária para p_i [Brzezinski et al., 1995; Kshemkalyani e Singhal, 1994; Ryang e Park, 1995].

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.

O foco deste trabalho é portanto estudar o problema de otimização relativo à resolução de deadlocks em sistemas distribuídos que operam de acordo com o modelo Ou, e, através de remoção de vértices (equivalente à matar um processo no sistema) que geralmente é passo seguinte à detecção de deadlocks, livrar o grafo de knots.

3. Deadlock-Resolution(Ou, Vértice)

Como deadlock é uma propriedade estável em sistemas distribuídos, uma vez que ele ocorre, devem ser feitas intervenções para solucioná-lo. 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.

Neste trabalho abordaremos como parametro a (modelo de deadlock do grafo de entrada) o modelo Ou. Como parametro b (tipo de intervenção externa) remoção de vértices, denotada por *Vértice*.

Problema: Deadlock-Resolution(Ou, Vértice).
Entrada: Grafo de espera G = (V, E) no modelo Ou.
Objetivo: Determinar o número mínimo de vértices que ao serem removidos livra o grafo de deadlock.

Portanto, para um dado grafo G, devemos encontrar o número mínimo de vértices a serem removidos de G de forma a torná-lo livre de knots (componentes fortemente conexas sem saídas compostas por pelo menos dois vértices). A remoção de um vértice em um sistema distribuído equivale a eliminar um processo.

4. Algoritmos para Grafos Subcúbicos

Em [Carneiro et al., 2015] prova-se que para um dado grafo de espera $G = (V, E)$ o problema Deadlock-Resolution(Ou, Vértice) é NP-Difícil . [Mitchell e Merritt, 1984] mostram que a resolução de deadlock é polinomial caso $deg^+(v) \leq 1$. É facil ver que para grafos com grau máximo 2 a solução 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 de no problema Deadlock-Resolution(Ou, Vértice) para grafos subcúbicos. Podemos então separar em três tipos de vértices: 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). Exploraremos na subseção a seguir os vértices fontes e sumidouros.

4.1. Vértices 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 4.1. *Fontes e sumidouros são desnecessários à remoção de deadlock.*

Demonstração. Primeiramente, trataremos o caso dos sumidouros (Figura 1). 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,

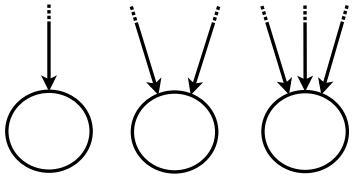

Figura 1: Vértices sumidouro em grafos subcúbicos.

qualquer vértice que possua um caminho direcionado a um sumidouro não está em deadlock no sistema; sendo assim, seu descarte ou remoção não interfere de forma alguma em qualquer solução ótima.

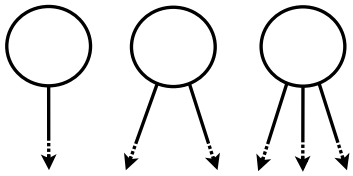

Figura 2: Fontes em grafos subcúbicos.

Ao analisar as fontes (Figura 2), 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 Q um 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 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 4.1, 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 3, 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.

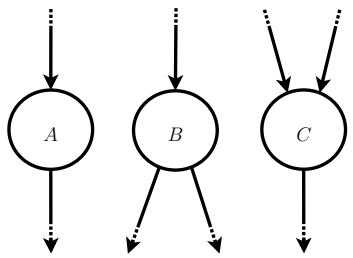

Figura 3: Vértices com pelo menos uma aresta de entrada e pelo menos uma de saída.

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 4.2. *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 4.2.

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 4): 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 $A.3$.

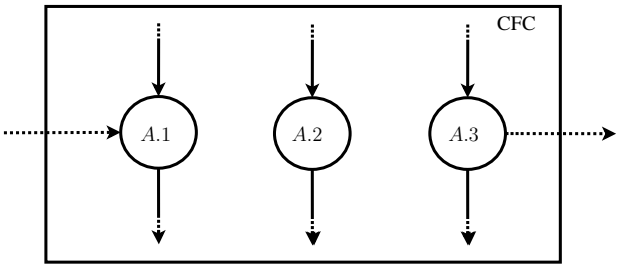

Figura 4: Subtipos de vértices A em uma CFC.

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 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 e C.

Lema 4.3. *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 $\sum_{v_i \in V} deg^-(v_i) = \sum_{v_i \in V} deg^+(v_i)$ [Satyanarayana e Prasad, 2014]. Como G é subcúbico e fortemente conexo, então G possui apenas vértices dos tipos A,B e C. Dado que um vértice do tipo A possui apenas uma entrada e uma 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 4.2.

Lema 4.4. *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 4.3). □

Corolário 4.5. *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 4.3 e do Lema 4.4. □

Lema 4.6. *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 4.4 é aplicável. □

Proposição 4.7. *Seja Q um knot de um grafo subcúbico G, onde a Regra 4.2 não é aplicável. Então, Q é um ciclo direcionado composto apenas por vértices do tipo $A.1$.*

Demonstração. Se a Regra 4.2 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 4.4, o Corolário 4.5 e o Lema 4.6, 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. □

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 $A.1$. 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 4.2 é sempre aplicável, pois todo knot Q' possui um vértice w do tipo $A.2$ (Lema 4.6). Esse vértice 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 1).

Algoritmo 1: Algoritmo 2-aproximativo

Entrada:
G: Grafo de espera subcúbico no modelo Ou
Saída:
S: Lista de vértices a serem removidos

```
1  início
2      G' ← G − {v | v é obtido pelo Pré-processamento 1}
3      K ← 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 4.2 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                  Adicionar a S um vértice vértice v de Q que é adjacente a C
10                 (para Q a Regra 4.2 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' ← lista de knots de G[V \ S] utilizando o algoritmo de detecção de knots
17     para cada knot Q' em K' faça
18         Aplicar a Regra 4.2 a Q' inserindo o vértice selecionado em S
19     fim para cada
20     retornar S
21 fim
```

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 4.8. 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 4.9. *Podemos desconsiderar qualquer CFC C^1 que alcança alguma CFC C^2 que não é knot. Além disso, se C^1 está à distância 1 de um knot Q, a Regra 4.2 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. □

Pré-processamento 3. Pela Observação 4.8 resolver knots que possuem CFCs com dois caminhos direcionados a um unico knot. Pelo Lema 4.9, podemos remover todas as CFCs que alcançam CFCs que não são knots.

Observação 4.10. 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 4.11. 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.*

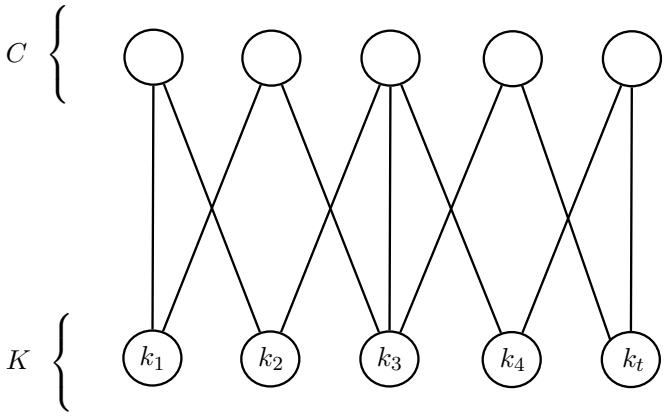

Figura 5: Grafo bipartido resultante das contrações de componentes C e knots K.

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 grafo 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). 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 6, 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

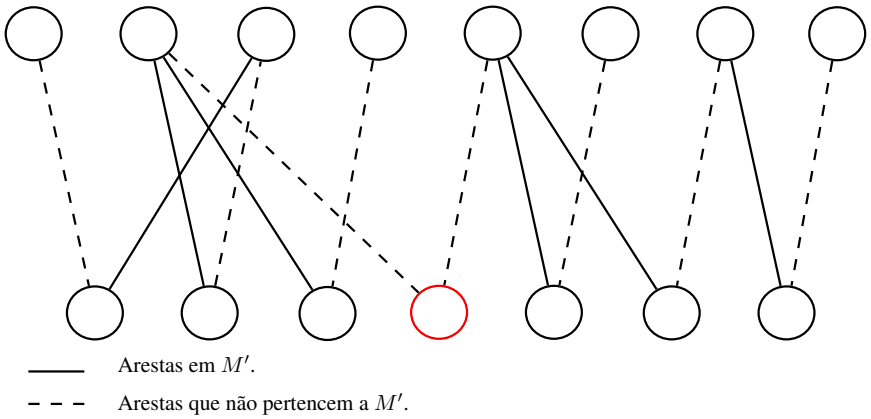

Figura 6: Exemplo de conjunto M' que satura $t-1$ vértices. O vértice destacado em vermelho não é saturado por M'.

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.

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, utilizaremos resultados em semi-emparelhamento.

Semi-emparelhamento é uma generalização natural do problema clássico de emparelhamento em grafos bipartidos [Katrenic e Semanisin, 2011]. Seja $G = (K \cup C,\ E)$ um grafo bipartido com $n = |K| + |C|$ vértices e $m = |E|$ arestas. 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 [Galčík et al., 2011] foi apresentada uma generalização do semi-emparelhamento denominada $(f,\ g)$-semi-emparelhamento, que 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 $f : k \to N$ e $g : c \to N$ funções. Um (f, g)-semi-emparelhamento em um grafo bipartido $G = (K \cup C,\ E)$ 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 $g(c_j)$ arestas em M [Bokal et al., 2012].

Lema 4.12. *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 [Hopcroft e Karp, 1973] são apresentados na literatura [Bokal et al., 2012; Galčík et al., 2011; Katrenic e Semanisin, 2011]. □

Finalmente podemos afirmar que o seguinte teorema é verdadeiro.

Teorema 4.13. *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 4.13, elaboramos o Algoritmo 2. 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.

5. Conclusões

Neste trabalho estudamos com mais profundidade o problema Deadlock-Resolution(Ou, Vértice). O problema de forma genérica foi provado NP-Difícil por [Carneiro et al., 2015]. Provamos também que DFVR no modelo E é NP-Difícil. [Mitchell e Merritt, 1984] mostram que a resolução de deadlock é polinomial caso $deg^+(v) \leq 1$. Portanto exploramos a classe de grafos subcúbicos (grafos com grau máximo 3) onde provamos que o problema é resolvível em tempo polinomial e apresentamos um algoritmo 2-aproximativo e um algoritmo determinístico.

É apresentada na Tabela 1 a complexidade computacional do problema Deadlock-Resolution(Ou, Vértice).

Deadlock-Resolution(Ou, Vértice)	
Instância	Complexidade
Sem Restrições	NP-Difícil
$\Delta(G) = 3$	Polinomial
$\Delta(G) = 2$	Trivial
$\Delta(G)^+ = 1$	Trivial

Tabela 1: Complexidade para diferentes classes de grafos do problema Deadlock-Resolution(Ou, Vértice).

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.

Algoritmo 2: Algoritmo resolução subcúbico

Entrada:
G: Grafo de espera subcúbico no modelo Ou
Saída:
S: Lista de vértices a serem removidos

início
  $G' \leftarrow G - \{v \mid v$ é obtido pelo Pré-processamento $1\}$
  $K \leftarrow$ lista de knots de G' utilizando o algoritmo de detecção de knots
  para cada *knot* Q *em* K **faça**
    se Q *contém um vértice* v_i *dos tipos* $A.2$, B *ou* C **então**
      Aplicar a Regra 4.2 a Q inserindo o vértice selecionado em S
    senão
      se Q *possui duas arestas de entrada provenientes de uma CFC,* C **então**
        Incluir em S um vértice de Q que é adjacente a C
        (para Q a Regra 4.2 é aplicável)
      senão
        se Q *possui uma aresta de entrada proveniente de uma CFC* C^1, *que alcança uma outra CFC* C^2 *(não é knot)* **então**
          Incluir em S um vértice de Q que é adjacente a C^1
          (para Q a Regra 4.2 é aplicável)
        fim se
      fim se
    fim se
  fim para cada
  Desconsidere qualquer CFC C^1 que alcança alguma outra CFC C^2 que não é knot.
  $G' \leftarrow$ O grafo a ser considerado, após Pré-Processamentos 1, 2 e 3
  (Veja Observação 4.10)
  $B \leftarrow$ Grafo G' contraído
  $M' \leftarrow (1, g)$-semi-emparelhamento de B
  Incluir em S vértices obtidos a partir de M'
  retornar S
fim

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

Bokal, D., Brešar, B., e Jerebic, J. (2012). A generalization of hungarian method and hall's theorem with applications in wireless sensor networks. *Discrete Applied Mathematics*, 160(4):460–470.

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*.

Chandy, K. M. e Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75.

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

Coulouris, G. F., Dollimore, J., e Kindberg, T. (2005). *Distributed systems: concepts and design*. pearson education.

Datta, A. e Ghosh, S. (1984). Synthesis of a class of deadlock-free petri nets. *Journal of the ACM (JACM)*, 31(3):486–506.

Fanti, M. P. e Zhou, M. (2002). Petri net approaches to deadlock modeling and resolution in automated manufacturing. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 3, p. 6–pp. IEEE.

Galčík, F., Katrenič, J., e Semanišin, G. (2011). On computing an optimal semi-matching. In *Graph-Theoretic Concepts in Computer Science*, p. 250–261. Springer.

Hopcroft, J. E. e Karp, R. M. (1973). An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231.

Katrenic, J. e Semanisin, G. (2011). A generalization of hopcroft-karp algorithm for semi-matchings and covers in bipartite graphs. *arXiv preprint arXiv:1103.1091*.

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

Mitchell, D. P. e Merritt, M. J. (1984). A distributed algorithm for deadlock detection and resolution. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, p. 282–284. ACM.

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.

Ryang, D.-S. e Park, K. H. (1995). A two-level distributed detection algorithm of AND/OR deadlocks. *Journal of Parallel and Distributed Computing*, 28(2):149–161.

Satyanarayana, B. e Prasad, K. S. (2014). *Discrete Mathematics and Graph Theory*. PHI Learning Pvt. Ltd.

Tanenbaum, A. S. e Van Steen, M. (2002). *Distributed systems: principles and paradigms*, volume 2. Prentice hall Englewood Cliffs.

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.