Universidade Federal
do Rio de Janeiro
Escola Politécnica

ALGORITMOS PARA ENUMERAÇÃO DE CONJUNTOS DE ENLACES VIÁVEIS EM REDES SEM FIO

Guilherme Iecker Ricardo

Projeto de Graduação apresentado ao Curso de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientadores: José Ferreira de Rezende
Valmir Carneiro Barbosa

Rio de Janeiro
Setembro de 2016

ALGORITMOS PARA ENUMERAÇÃO DE CONJUNTOS DE ENLACES VIÁVEIS EM REDES SEM FIO

Guilherme Iecker Ricardo

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

Prof. José Ferreira de Rezende, D.Sc.

Eng. Raphael de Melo Guedes, D.Sc.

Prof. Daniel Raton Figueiredo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2016

Ricardo, Guilherme Iecker

Algoritmos para Enumeração de Conjuntos de Enlaces Viáveis em Redes Sem Fio/Guilherme Iecker Ricardo. – Rio de Janeiro: UFRJ/POLI – COPPE, 2016.

VIII, 20 p.: il.; 29, 7cm.

Orientadores: José Ferreira de Rezende

Valmir Carneiro Barbosa

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Computação e Informação, 2016.

Referências Bibliográficas: p. 20 – 20.

1. Redes Sem Fio. 2. Projeto de Algoritmos. 3. Otimização Combinatória. I. de Rezende, José Ferreira *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Computação e Informação. III. Título.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

SINR	Signal to Interference plus Noise Ratio, p. 5
TIP	Teste de Interferência Primária, p. 5
TIS	Teste de Interferência Secundária, p. 6

Capítulo 1

Introdução

O problema de enumeração de conjuntos de enlaces viáveis em redes sem fio surgiu de uma proposta de solução ao problema de escalonamento de enlaces. Neste capítulo, apresentamos o problema de escalonamento de enlaces como motivação, de onde surgiu a necessidade de encontrar e listar os conjuntos de enlaces viáveis, e algumas definições importantes ao longo deste trabalho.

Uma rede mesh é composta por nós que podem se comunicar diretamente através de enlaces de comunicação. Os nós são os componentes da rede onde ocorre a computação e eventualmente desejam se comunicar. Um enlace de comunicação, ou simplesmente enlace, é um componente da rede que representa uma conexão física ou lógica entre dois nós. Uma transmissão ocorre quando um nó, chamado de transmissor, envia mensagens a outro nó, chamado de receptor, através de um enlace de comunicação. Em um dado período de tempo, um enlace está ativo quando há uma transmissão em curso. Em redes sem fio, os nós compartilham o meio de transmissão. Por isso, quando existe um cenário com mais de um enlace ativo, é comum que os nós transmissores de um enlace causem interferência nos nós receptores de outro enlace. Em alguns casos, essa interferência pode atrapalhar ou mesmo inviabilizar a troca de mensagens na rede. Portanto, faz-se necessário um mecanismo para organizar os enlaces no tempo com o intuito de reduzir o efeito de tais interferências.

Um conjunto de enlaces ativos é dito viável quando todos os enlaces que o compõem passam nos testes de interferência. No próximo capítulo, os testes serão explicados com detalhes. Por enquanto, podemos pensar que passar nos testes significa que tais enlaces causam pouca ou nenhuma interferência entre si. Considerando um período de tempo T no qual diversas transmissões estão acontecendo, T pode ser particionado em um conjunto de slots de tempo $S = \{s_1, s_2, \ldots, s_t\}$ tal que $s_1 \cap s_2 \cap \ldots \cap s_t = \emptyset$ e $s_1 \cup s_2 \cup \ldots \cup s_t = T$. O problema de escalonamento de enlaces consiste em encontrar maneiras de distribuir os enlaces ativos de uma rede em S, de forma que os conjuntos de enlaces em cada slot sejam viáveis e o tamanho de S seja o menor possível.

Existem diversas estratégias para abordar esse problema, algumas delas estão em [1, 2]. Nesses casos, tais estratégias baseiam-se em heurísticas que, apesar de serem executadas em tempo factível, podem não apresentar os melhores resultados. Visando desenvolver um algoritmo para resolver o problema de escalonamento de enlaces usando programação linear e encontrar o melhor resultado, é fundamental que todos os conjuntos de enlaces viáveis sejam enumerados para serem usados como restrições de um problema de programação linear. Dada uma rede *mesh*, como a descrita anteriormente, podemos analisá-la em um dado instante t. Em t, podemos ver quais enlaces estão estabelecidos e em qual direção a transmissão está ocorrendo. A disposição dos enlaces em cada instante t pode ser interpretada como um grafo direcionado $G = (V, E)$. Nesse grafo, V é o conjunto de nós (vértices) e E é o conjunto de enlaces que possuem uma direção de transmissão específica (arestas direcionadas).

Um conjunto de enlaces C que desejamos analisar é um subconjunto de E, ou seja $C \subset E$. O problema de enumeração de conjuntos de enlaces viáveis consiste em desenvolver um algoritmo capaz de encontrar todos os conjuntos C, tal que C é viável, com o menor tempo para que possa ser utilizado em aplicações práticas. Caso G tenha m enlaces, então teremos um total de 2^m conjuntos a serem testados. Com isso, a complexidade de tempo de um algoritmo ingênuo que realiza os testes de interferência sequencialmente para cada conjunto de enlaces possível é $O(2^m)$, ou seja, para m suficientemente grande, ele não termina em tempo factível. Por isso, é preciso que algum método mais inteligente seja adotado.

Nesse trabalho, vamos modelar esse problema usando a ideia de árvore de combinação que vai nos oferecer propriedades fundamentais para obtermos menores complexidades de tempo. Depois, vamos apresentar dois algoritmos que usam essa propriedade, realizam os testes de maneira mais inteligente e conseguem enumerar todos os conjuntos de enlaces viáveis em tempo factível. Vamos comparar seus desempenhos e analisar as técnicas de implementação e otimização adotadas. Finalmente, apresentamos uma breve conclusão sobre o projeto como um todo e propostas de trabalhos futuros.

1.1 Organização do Texto

No Capítulo 1, os principais conceitos e

Capítulo 2

Modelagem

Neste capítulo, é descrito um novo modelo para o problema proposto o qual será a base para os algoritmos introduzidos nos Capítulos 3 e 4. Em seguida, é apresentado como as interferências foram modeladas, como os testes de interferência são definidos e uma propriedade decorrente da modelagem que nos ajudará a garantir menores complexidades de tempo.

2.1 Árvore de Combinações

Seja um grafo direcionado $G = (V, E)$ que representa uma rede mesh sem fio com seus dispositivos e enlaces de comunicação ativos. O conjunto de arestas E representa todas as possibilidades de enlaces ativos para essa rede. É possível definir um subconjunto C de E escolhendo quais enlaces estão ativos ou não. Com isso, tem-se que o menor tamanho de C é zero, ou seja, não há enlaces ativos ($C = \emptyset$), e o maior tamanho de C é o próprio E com todos os enlaces possíveis ativos ($C = E$). Entre esses dois extremos, existem 2^m combinações de enlaces possíveis, que são subconjuntos de E definidos pela escolha de quais enlaces estão ativos.

Sobre essas combinações de enlaces é possível definir uma árvore de combinações. Se $m_c = |C|$ então para uma altura $h = m_c$ da árvore, o número de conjuntos possíveis com tamanho m_c é dado por $C(h, m)$. Nessa árvore, parte-se da combinação vazia ($m_c = 0$) como raiz ($h = 0$). Os filhos de qualquer vértice C da árvore são obtidos combinando C com um enlace $i \in E$, tal que $i \ni C$.

A ordem em que os enlaces são alocados nos conjuntos não é relevante nesse caso, de forma que se $C = \{e_1, e_2\}$ e $C' = \{e_2, e_1\}$, então $C \equiv C'$, ou seja, C e C' são equivalentes. Se as combinações equivalentes fossem consideradas na árvore, o número de conjuntos a ser testado iria desnecessariamente aumentar muito. Por isso, ao construir a árvore, apenas uma das combinações equivalentes é incluída. É apresentado na Figura 2.1 um exemplo de árvore de combinações com a estrutura descrita para $m = 3$.

No contexto de árvore de combinações, é importante definir também os descendentes e os ancestrais de um vértice. Um descendente de um vértice u em uma árvore é qualquer vértice v que pode ser alcançado a partir de u por algum caminho na árvore. Analogamente, u é ancestral de v se, e somente se, v é descendente de u. A abstração de árvore de combinações permite que os conjuntos de enlaces sejam percorridos conforme a estrutura da árvore. Com isso é possível desenvolver métodos sistemáticos para buscar e avaliar a viabilidade das combinações de enlaces. Os métodos mais famosos como Busca em Largura e Busca em Profundidade em árvores podem ser considerados nesse caso. Entretanto, sua utilização exige a construção prévia da árvore que apresenta alta complexidade tanto de tempo, quanto de espaço ($O(2^m)$ em ambos os casos).

Os algoritmos que serão apresentados nos próximos capítulos apenas usam a ideia de árvore de combinações em seus projetos. Não há construção da árvore para realizar buscas e isso representa uma diminuição considerável na complexidade de espaço. Além disso, eles fazem uso da propriedade apresentada na próxima seção que por sua vez ajuda a reduzir a complexidade de tempo.

2.2 Modelos de Interferência

No capítulo anterior, o teste de interferência foi descrito de maneira intuitiva e pouco detalhada. Nessa seção, eles serão definidos formalmente e uma propriedade decorrente da natureza desses testes será apresentada. Essa propriedade é importante pois permitirá a confecção de algoritmos de enumeração com melhores desempenhos no futuro.

2.2.1 Modelo de Interferência Primária

Existem duas características dos enlaces que limitam a comunicação entre os nós em uma rede sem fio:

1. Enlaces *half-duplex*: Em canais *half-duplex*, os rádios dos dispositivos que compõem a rede não podem transmitir e receber mensagens ao mesmo tempo. Por isso, um nó pode apenas ser ou transmissor ou receptor em um dado cenário.

2. Enlaces dedicados: Apesar das mensagens serem transmitidas em várias direções e, portanto, alcançando diversos nós, elas são direcionadas a apenas um nó específico.

Ao modelar uma rede em que os enlaces possuam essas duas características, os nós apenas assumem papel de transmissor ou de receptor em, no máximo, um enlace.

Alguns casos que possuem ou não as características I e II são ilustrados na Figura 2.2.

Seja i um enlace de um conjunto $C \subset E$. O nó transmissor e o nó receptor de i são, respectivamente, s_i e r_i. Dizemos que C passa no **Teste de Interferência Primária** (TIP) se, e somente se, $\forall i, j \in C, s_i \neq s_j \& s_i \neq r_j \& r_i \neq s_j \& r_i \neq r_j$.

2.3 Modelo de Interferência Secundária

Como mencionado no capítulo anterior, o meio de transmissão é compartilhado, então o nó transmissor em um enlace pode interferir nos nós receptores de outros enlaces. Contudo, existe um limite de interferência aceitável que é baseado na SINR (*Signal to Interference plus Noise Ratio*) dos nós receptores. Antes de definir a SINR, algumas outras definições se fazem necessárias.

A primeira delas é a Potência de Recepção $RP(s, r)$ que é a potência com que um sinal transmitido por um nó transmissor s a uma potência de transmissão TP é recebido em um nó r seguindo o modelo de propagação. Matematicamente,

$$RP(s, r) = \frac{TP}{(\frac{d_{sr}}{d_0})^{\alpha}} \tag{2.1}$$

onde α é o coeficiente e d_0 é a distância de referência do modelo de propagação.

Com isso, dado um nó receptor r_i, consegue-se calcular a potência de recepção em r_i em duas situações distintas:

- quando o sinal é transmitido por s_i, ou seja, é a potência de recepção dentro do próprio enlace i.
- quando o sinal é transmitido por s_j, tal que $j \neq i$, ou seja, é a potência de recepção de um sinal transmitido em um outro enlace j. Nesse caso, a potência de recepção de tais sinais é chamada de interferência.

Denomina-se Interferência Total $I(i, C)$ a soma das interferências que os nós transmissores de todos os outros enlaces de C exercem sobre o nó receptor do enlace i. Ou seja,

$$I(i, C) = \sum_{j \neq i} RP(s_j, r_i) \tag{2.2}$$

Finalmente, a $SINR(i, C)$ é a razão entre a potência de recepção em r_i referente a transmissão no enlace i e a interferência causada pelo ruido do ambiente γ e a interferência total dos outros enlaces no conjunto C.

$$SINR(i, C) = \frac{RP(s_i, r_i)}{\gamma + I(i, C)} \tag{2.3}$$

Dado um conjunto de enlaces C e tendo calculado $SINR(i, C)$, $\forall i \in C$, deve-se comparar os valores encontrados com uma constante β, que representa um valor numérico para o limite de interferência tolerado. Se a interferência for muito alta, o valor do denominador na Equação 2.3 irá aumentar, fazendo o valor da $SINR(i, C)$ diminuir. Portanto, β é um limite inferior, tal que, se $SINR(i, C) \geq \beta$, $\forall i \in C$, então C passa no Teste de Interferência Secundária (TIS).

Nesse trabalho, os seguintes valores para as constantes foram usados: $\alpha = 4.0$, $\beta = YYY$ e $\gamma = XXX$.

2.3.1 Inviabilidade Hereditária

Dados os modelos de interferência apresentados, se um conjunto de enlace C passar em ambos os testes, então diz-se que C é viável. Entretanto, nesta subseção, será analisado o que acontece quando C não é viável. Em um primeiro cenário, assume-se que C não passou no TIP. Nesse caso, pelo menos um nó de C está participando de mais de um enlace, o que é proibido. Seja um conjunto C', tal que $C' = C \cup i$, onde $i \in E$. A adição do novo enlace i em C pode: (**i**) conectar dois nós contidos em C; (**ii**) conectar um nó existente em C a um novo nó; e (**iii**) incluir dois novos nós conectados por i.

Nas três situações descritas anteriormente, a adição do novo enlace i para formar C' não muda o fato de que C não é viável, independentemente do efeito que i cause no conjunto original C. Consequentemente, é possível notar que C' também não é viável.

Em um segundo cenário, assume-se que C passou no TIP, mas não passou no TIS. Nesse caso, $SINR(i, C) < \beta$, para algum enlace i. Seja um conjunto C', tal que, $C' = C \cup j$, onde $j \in E$ e C' também passa no TIP. A adição de um novo enlace ao conjunto C para formar C', apenas fará aumentar a interferência nos enlaces já contidos em C. Mesmo que a contribuição na interferência total seja pequena, podendo até ser desprezada, a

adição de um novo enlace não muda o fato de que C não é viável. Consequentemente, é possível notar que C' também não é viável.

Os dois cenários apresentados garantem a seguinte propriedade: se um conjunto C não é viável, independentemente de qual teste de interferência ele falhou, então qualquer conjunto C', tal que $C \subset C'$ também não é viável. Usando o modelo de árvore de combinações apresentado na seção anterior, se uma combinação C da árvore de combinações não é viável, então todos os seus descendentes na árvore também não são viáveis. Por isso, essa propriedade é denominada Inviabilidade Hereditária.

Devido a Inviabilidade Hereditária, no processo de busca e verificação de viabi-

lidade de todas as combinações de enlaces de uma rede, sabe-se que, ao encontrar qualquer combinação não viável, não é necessário testar a viabilidade de nenhum de seus descendentes. O ato de não testar os descendentes de uma combinação não viável pode ser chamado de "podar" a árvore.

2.4 Algoritmo de Viabilidade

Dado o modelo de interferência descrito na seção 2.2, o algoritmo de viabilidade é puramente baseado nos testes de interferência primária e secundária. Algoritmos para os testes serão descritos formalmente e serão combinados para gerar o algoritmo de viabilidade final.

2.4.1 Algoritmo para o Teste de Interferência Primária

A seguir, o algoritmo para o Teste de Interferência Primária, TIP, é formalizado:

Algoritmo 1: Algoritmo TIP

```
1 input: Conjunto de enlaces C
2 foreach i ∈ C do
3     foreach j ∈ C, j ≠ i do
4         if (( s_i == s_j ) || ( s_i == r_j ) || ( r_i == s_j ) || ( r_i == r_j )) then
5             return FALSE
6 return TRUE
```

Prova de Corretude

O TIP apenas formaliza o que foi definido na subseção 2.2.1. Portanto, está correto.

Análise da Complexidade

Para o pior caso, $|C| = m$, então a complexidade de tempo é $O(m^2)$. A complexidade de espaço é definida pelo maior tamanho de C possível, portanto, $O(m)$.

2.4.2 Algoritmo para o Teste de Interferência Secundária

A seguir, o algoritmo para o Teste de Interferência Secudária, TIS, é formalizado:

Prova de Corretude

O TIS apenas formaliza o que foi definido na subseção 2.2.2. Portanto, está correto.

Algoritmo 2: Algoritmo TIS

```
1 input: Conjunto de enlaces C
2 foreach i ∈ C do
3 |   if (( SINR(i,C) < β )) then
4 |   |   return FALSE
5 return TRUE
```

Análise da Complexidade

Como o cálculo da $SINR(i, C)$ é $O(m)$, devido o laço definido na linha 2 iterar sobre no máximo m enlaces, então a complexidade de tempo é $O(m^2)$. Analogamente ao TIP, a complexidade de espaço é $O(m)$.

2.4.3 Algoritmo para o Teste de Viabilidade

O algoritmo para testar a viabilidade de um conjunto de enlaces é simplesmente a junção dos algoritmos anteriores conforme a descrição abaixo:

Algoritmo 3: Algoritmo VIAVEL

```
1 input: Conjunto de enlaces C
2 if (TIP(C)) || (TIS(C)) then
3 |   return TRUE
4 else
5 |   return FALSE
```

Prova de Corretude

Análise da Complexidade

2.5 Exemplo

O grafo G=(V,E) da Figura 2.3 representa os nós e os enlaces de uma rede mesh sem fio. Deseja-se encontrar os conjuntos de enlaces viáveis dessa rede.

Nesse exemplo, $E = \{1, 2, 3, 4\}$, ou seja, $m = 4$. Portanto, existem $2^4 = 16$ combinações de enlaces que são representados na árvore de combinações da Figura 2.4. Uma Busca em Profundidade será executada para percorrer os vértices da árvore que serão verificados usando o algoritmo VIÁVEL. A ordem em que os vértices são visitados é $\{\{\}, \{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}, \{1, 3\}, \{1, 3, 4\}, \{1, 4\}, \{2\}, \{2, 3\}, \{2, 3, 4\}, \{2, 4\}, \{3\}, \{3, 4\}, \{4\}\}$ e pode ser verificada também na Figura 2.4.

A Tabela 2.1 mostra o resultado dos testes para cada combinação e a Tabela 2.2 mostra os motivos pelos quais as combinações falharam os testes.

Finalmente, os conjuntos de enlaces viáveis obtidos são: $\{\{\}, \{1\}, \{2\}, \{2,4\}, \{3\}, \{3,4\}, \{4\}\}$. No próximo capítulo, o primeiro algoritmo de enumeração é introduzido. Trata-se de um algoritmo iterativo que, apesar de seguir a estrutura da árvore, não precisa instanciá-la como foi feito nesse exemplo. No capítulo 4, é apresentada uma outra versão desse algoritmo que, devido sua natureza recursiva, permite que muitos dos cálculos realizados nos testes de interferência sejam poupados.

Capítulo 3

Algoritmo Iterativo

3.1 Introdução

O projeto do algoritmo iterativo é baseado na ideia da árvore de combinações e nos modelos de interferência apresentados.

3.2 Codificando Combinações em Inteiros

A primeira abstração a ser feita é considerar as combinações de enlaces como números binários. Nesses números, cada bit está relacionado à pertinência de um enlace específico àquela combinação. Consequentemente, também é possível representar a combinação de enlaces como um número inteiro sem sinal resultado da conversão do número na base 2 para a base 10. A seguir, essa modelagem é apresentada fazendo uso de uma linguagem mais formal, visando permitir algumas manipulações matemáticas importantes ao funcionamento do algoritmo.

Seja o grafo direcionado $G = (V, E)$. As m arestas em E são enumeradas como $e_0, e_1, e_2, ..., e_{m-1}$. Seja B um número inteiro na base 2 com m bits. Os bits em B são enumerados como $b_0, b_1, b_2, ..., b_{m-1}$. Seja uma combinação de arestas C tal que $CinE$. O número $B(C)$ codifica a combinação C quando:

$$b_i = \begin{cases} 1 & \text{se, e somente se, } e_i \in C \\ 0 & \text{se, e somente se, } e_i \notin C \end{cases}$$

Essa ideia de codificação é estendida para toda a árvore de combinações. Ao fazer isso, todos os 2^m vértices da árvore serão codificados em inteiros.

Na Figura 3.1 é ilustrada uma árvore de combinações para uma rede com $E = \{a, b, c, d\}$, ou seja, $m = 4$. Na imagem, todos os $2^4 = 16$ vértices possuem sua identificação principal na primeira linha, o valor codificado na base 2 na linha de baixo e o valor convertido para a base 10 entre parênteses na última linha.

É possível verificar duas consequências diretas da codificação, que em um primeiro momento podem parecer triviais, mas são o arcabouço do projeto do algoritmo. A primeira está relacionada a maneira de percorrer a árvore e a segunda está relacionada a maneira de "podar" a árvore.

3.3 Percorrendo a Árvore Iterativamente

No exemplo da Figura 3.1, foi visto que, para uma rede com 4 enlaces, tem-se uma árvore de combinações com 16 vértices. A árvore é estruturada da raiz ($C = \emptyset$ ou $B(C) = 0$) à sua folha de maior altura ($C = E$ ou $B(C) = 15$). De forma geral, para um grafo com m enlaces, a árvore possui 2^m vértices, partindo de $B = 0$ e chegando a $B = 2^m - 1$.

Na árvore codificada, a ordem em que os enlaces são considerados é muito importante porque os índices dos bits de $B(C)$ são estritamente relacionados aos enlaces e, consequentemente, cada combinação C é codificada unicamente por um número B.

Sabendo disso, é possível percorrer a árvore de combinações simplesmente realizando uma contagem que vai de 0 até $2^m - 1$. Ou seja, todo vértice codificado $B > 0$ é alcançado a partir da soma de um outro nó codificado mais 1.

Figura 3.2 mostra como a árvore do exemplo anterior é percorrida através da contagem. Comparado com o método de busca adotado na seção 2.3, é importante ressaltar que a ordem em que os vértices são visitados é totalmente diferente. Essa nova ordem é consistente com uma contagem que somente é alcançável devido a codificação aqui introduzida.

3.4 "Podando" a Árvore de Combinações

Diferente do caso anterior, a vantagem em codificar as combinações para a atividade de "poda" da árvore não é tão trivial. Contudo, o método de busca baseado na contagem será usado para detectar alguns padrões que ajudam a "podar" a árvore de maneira eficiente. Mais uma vez, a correspondência entre E e B é muito importante e sua ordem, uma vez arbitrada, deve ser mantida no decorrer do processo.

Dado um inteiro B, seus bits b_i podem fornecer mais informações úteis além da pertinência do enlace e_i ao conjunto C. Considere o caso em que b_i^* é o bit ativo menos significativo, ou seja, é o bit 1 mais à direita. Nesse caso, o índice i de b_i^* também representa o número de filhos que B tem na árvore, ou seja, quantas combinações são consequências diretas, ou descendentes diretos, de C. Em termos de bits, i representa os números que são resultantes da variação dos bits 0 à direita de b_i^* em B.

Isso significa que b_i^* tem muito a dizer sobre os descendentes de B. Especificamente, se essa ideia for aplicada de maneira recursiva aos filhos B' de B, seus b_i^*'s vão indicar o número de netos de B e assim por diante. Ao final da recursão, podemos ver que uma combinação C codificada em B tal que b_i^* é o bit ativo menos significativo tem exatamente 2^i–1 descendentes.

No capítulo 2, baseado na ideia de inviabilidade hereditária, o termo "podar" a árvore foi definido como o ato de ignorar os descendentes de uma combinação não viável ao percorrer uma árvore. Para o presente caso, como uma contagem está sendo feita, um termo mais adequado seria "saltar". Portanto, uma vez que uma combinação codificada $B(C)$ não for viável, um "salto" na contagem será realizado sobre todos os seus descendentes.

Como o número de descendentes de uma combinação B é facilmente determinável usando b_i^*, caso seja necessário realizar um "salto" na contagem devido a inviabilidade de B, basta incrementar a contagem com o número de descendentes mais 1. Ou seja, se B não é viável, a próxima combinação a ser testada será $B + 2^i$, onde i é o índice do bit ativo menos significativo.

Um exemplo de cenário em que os "saltos" ocorrem é ilustrado na Figura 3.3. Na rede do exemplo, sabe-se que as combinações 4 e 12 não são viáveis. Por isso, todos os descendentes de 4 e de 12 serão ignorados na contagem, resultando na sequência $0, 1, 2, 3, 4, 8, 9, 10, 11, 12$. Até esse ponto, as combinações de enlaces viáveis ainda não estão sendo listadas, apenas está sendo mostrada a sequência de enlaces visitados na contagem.

3.5 Decodificando um Conjunto de Enlaces

Mesmo que agora seja possível percorrer a árvore e "saltar" sobre os descendentes de uma combinação inviável, não pode-se testar a viabilidade de um conjunto de enlaces usando apenas um número inteiro. É preciso decodificar o número inteiro em um conjunto de enlaces para que os nós da rede envolvidos sejam conhecidos e, finalmente, permitir a realização dos testes. A seguir, um algoritmo recursivo para decodificar um inteiro em um conjunto de enlaces é apresentado. Vale ressaltar que o processo de decodificação é o responsável por não ser preciso instanciar toda uma árvore de combinações. Com ele, é suficiente conhecer o conjunto de enlaces E e os índices dos bits de B para realizar os testes de interferência.

Algoritmo 4: Algoritmo DECODIFICADOR

```
1 input: Número inteiroB, número inteiro I, conjunto de arestas E(G)
2 Q ← B/2
3 R ← B%2
4 if (R = 1) then
5 |  C ← C ∪ {e_I}
6 if (Q > 0) then
7 |  I ← I + 1
8 |  DECODIFICADOR(Q, I, E(G))
```

3.5.1 Algoritmo Decodificador

3.5.2 Prova de Corretude

O algoritmo apresentado nada mais é do que uma versão recursiva do algoritmo padrão de conversão de um número na base 10 para a base 2 com uma modificação para permitir a captura dos enlaces pertencentes ao conjunto.

Nesse algoritmo existem duas variáveis de controle, Q e R, que são, respectivamente, o quociente e o resto da divisão inteira entre os números B e 2. Se $R = 1$, temos $b_I = 1$ e, consequentemente, o enlace e_I pertence à combinação C decodificada a partir do inteiro B. Reciprocamente, se $b_I = 0$, então o enlace e_I não pertence a C. A variável Q controla a parada do algoritmo. Enquanto $Q > 0$, o processo continua incrementando o valor de I e verificando a pertinência dos enlaces com novos índices I. Quando $Q = 0$, o processo de conversão é finalizado.

Quando $R = 1$, então a condição na linha 3 é satisfeita e o enlace com índice I é adicionado ao conjunto decodificado na linha 4. Portanto, ao final da recursão, C será o conjunto de enlaces decodificado apropriadamente.

3.5.3 Análise de Complexidade

Complexidade de Espaço

O(m)

Complexidade de Tempo

Todas as linhas do algoritmo são $O(1)$. Contudo, como trata-se de um algoritmo recursivo, a função Decodificador será chamada o número de vezes equivalente ao índice do bit ativo mais significativo. No pior caso, teremos $m - 1$ chamadas da função, o que define sua complexidade de tempo como $O(m)$.

3.6 Descrição do Algoritmo

Finalmente, agora que os métodos de como percorrer a árvore de combinações, "saltar" sobre os descendentes das combinações inviáveis e decodificar os inteiros para realizar os testes de interferência são conhecidos, é possível descrever o algoritmo iterativo para enumeração de conjuntos de enlaces viáveis.

3.6.1 Algoritmo Iterativo para Enumeração de Conjuntos de Enlaces Viáveis (IECEV)

Algoritmo 5: Algoritmo IECEV

```
1  input: Grafo direcionado G = (V, E)
2  output: Conjuntos de Enlaces Viáveis F
3  F ← ∅
4  C ← ∅
5  B ← 0
6  I ← 0
7  while B < 2^m do
8      C ← DECODIFICADOR(B, 0, E(G))
9      if VIAVEL(C) then
10         F ← F ∪ B
11         I ← 1
12     else
13         I ← 2^i
14     B ← B + I
```

3.6.2 Prova de Corretude

De acordo com o que foi provado na seção 3.2, o laço instanciado na linha 5 e o incremento feito na linha 12 representam uma contagem e, de fato, fazem o algoritmo percorrer todas as combinações de enlaces. Caso uma combinação seja viável, ela passará no teste da linha 7 e será adicionada ao resultado final na linha 8.

Os "saltos" demonstrados na seção 4.3 são baseados na variável I, usada como incremento à variável B. Pelo algoritmo, I somente possui dois valores possíveis.

$$I = \begin{cases} 1 & \text{se, e somente se, } B \text{ é viável} \\ 2^i & \text{caso contrário} \end{cases}$$

Essa seleção de valores é gerenciada pela condicional das linhas 7 e 10.

Pela característica do problema, como mencionado nos capítulos anteriores, todos os 2^i descendentes de uma combinação inviável certamente também são inviáveis.

Portanto, os "saltos" apenas tem o papel de agilizar o processo e não interferem no resultado final do algoritmo.

Conclui-se que, de fato, o algoritmo retorna todos os conjuntos de enlaces viáveis.

3.6.3 Análise de Complexidade

Complexidade de Espaço

$O(2^m)$

Complexidade de Tempo

As linhas 1-4, 7-12 são $O(1)$. A função de decodificação na linha 6 é $O(m)$ e o teste de viabilidade na linha 7 é $O(m^2)$. O laço principal das linhas 5-12 é $O(2^m m^2)$ pois a cada uma das 2^m iterações (no pior caso), a função de decodificação e o teste de viabilidade são chamados com complexidade de $O(m) + O(m^2) = O(m^2)$. De maneira geral, a complexidade de tempo total é $O(2^m m^2)$. Contudo, esse valor de complexidade é muito exagerado uma vez que, devido aos saltos, a contagem cobrirá apenas uma parte das 2^m combinações. Nesse caso, o números de conjuntos avaliados na contagem é $O(|F|)$, onde $|F|$ é a quantidade conjuntos viáveis da rede. Além disso, como será discutido no Capítulo 5, o m^2 pode ser simplificado por n^2. Portanto, a complexidade é $O(|F|n^2)$.

3.7 Conclusão

Capítulo 4

Algoritmo Recursivo

4.1 Introdução

Neste capítulo, será introduzido um novo algoritmo para encontrar os conjuntos de enlaces viáveis em uma rede sem fio. Esse algoritmo é baseado nos mesmos modelos apresentados no Capítulo 2, portanto ele possui algumas similaridades como o Algoritmo Iterativo. Como será mostrado nas próximas seções, o diferencial consiste em uma nova maneira de percorrer a árvore, que permitirá que os cálculos da SINR dos enlaces em uma dada combinação sejam reutilizados por seus descendentes. Finalmente, o algoritmo será detalhado e sua complexidade analisada.

4.2 Percorrendo a Árvore Recursivamente

A grande diferença com relação ao Algoritmo Iterativo está na forma em que a árvore é percorrida. No caso iterativo, faz-se uma contagem, ou seja, visita-se os vértices da árvore somando 1 ao seu valor codificado anterior. O caso recursivo não é tão simples e essa seção será dedicada ao seu entendimento.

Como foi brevemente mencionado no capítulo anterior, é possível saber quantos descendentes tem um conjunto codificado B. Para isso, basta encontrar o bit ativo menos significativo b_i^* (o bit 1 mais à direita) e o número de bits 0 depois de b_i^* é o número de filhos de B. Como consequência disso, é possível notar que as folhas da árvore possuem os seus bits menos significativos ativos, ou seja, se B é ímpar, então B é folha. Um caso especial surge ao analisar $B = 0$. Nesse caso, não existem bits ativos e, por convenção, o número de filhos de $B = 0$ é $|B|$.

Além de saber quantos filhos tem a combinação B, conhecer b_i^* também nos permite alcançar os filhos de B. Para adicionar mais um enlace em C e, com isso, descobrir um possível filho de B na árvore, basta que um dos bits 0 depois de b_i^* seja alternado para 1. De maneira geral, para fazer essa alternância, basta realizar a

soma $B+2^i$, tal que, $i < i^*$. Consequentemente, para alcançar todos os filhos de uma combinação B basta fazer $B + 2^i$, $\forall i$, tal que, $0 \leq i \leq i^*$. Portanto, uma pequena contagem é feita para alternar todos os bits 0 depois de b_i^* em uma combinação B com o intuito de encontrar os filhos de B.

De maneira análoga, os netos de B podem ser encontrados por meio da aplicação da técnica descrita nos filhos de B. Em geral, para encontrar todos os descendentes de uma combinação B, basta aplicar recursivamente a técnica em cada descendente de B. Quando a recursão alcançar uma folha, essa instância se encerra. Se a técnica descrita for aplicada para $B = 0$, então todos os seus descendentes serão alcançados e, portanto, toda a árvore de combinações será visitada. Um exemplo de como percorrer uma árvore com $m = 4$ é apresentado na Figura 4.1.

4.3 "Podando" a Árvore de Combinações

"Podar" uma árvore de combinações que esteja sendo percorrida usando o método da seção anterior é muito mais simples do que "saltar" em uma contagem. Quando uma folha $B' = B + 2^i$ na árvore é alcançada, como ela não tem filhos, a técnica não será mais aplicada e instância da busca que visitou tal folha é encerrada, de forma que a busca continua em $B'' = B + 2^i + 1$. O mesmo acontece quando uma combinação B' visitada não é viável. Nesse caso, todos os descendentes de B' são ignorados e a busca continua em B'.

Dada essa situação, é importante ressaltar que, nem sempre irá existir um B'', ou seja, um vértice irmão de B'. Caso isso venha a acontecer, significa que todos os descendentes de um vértice já foram visitados (ou ignorados) e ele é o último em seu ramo com altura h, ou seja, o último filho de B. Isso fará com que sua instância da recursão se encerre e autoriza o seu pai na árvore a visitar algum outro filho, se houver.

4.4 Reaproveitando Cálculos

No algoritmo anterior, era fundamental que houvesse um processo de decodificação do número B em um conjunto C para que os testes de interferência fossem realizados. No caso da busca recursiva, quando um novo vértice é visitado, o enlace e_i correspondente ao índice i do bit alternado é adicionado em C. Ao ser adicionado em C, a porção de interferência causada e sofrida por e_i é atualizada. A viabilidade do conjunto C é testada toda vez que um novo enlace é adicionado. Simetricamente, depois de visitar todos os seus descendentes, e_i é removido de C.

Ao realizar o procedimento de adição e remoção dos enlaces descrito, dado um conjunto viável B, para testar a viabilidade de seus filhos na árvore, basta considerar

a contribuição de interferência do enlace adicionado a B. Isso significa que todo o cálculo de SINR feito de 0 até B não precisa ser refeito ao testar os filhos de B.

4.5 Descrição do Algoritmo

Até o momento, uma nova ideia de como percorrer a árvore foi apresentada. Essa ideia permite que a árvore seja "podada" e dispensa a necessidade de decodificação, o que, intuitivamente, representa um ganho na complexidade de tempo em relação ao Algoritmo Iterativo. O Algoritmo Recursivo é apresentado a seguir.

4.5.1 Algoritmo Recursiva para Enumeração de Conjuntos de Enlaces Viáveis (RECEV)

Entrada: Grafo direcionado G=(V,E), Inteiro X, Conjunto de Enlaces Atual C
Saída: Lista dos conjuntos dos enlaces viáveis F

4.5.2 Prova de Corretude

Como mostrado na seção 4.2, se $X = 0$, então toda a árvore é percorrida chamando o algoritmo RECEV recursivamente para todos os descendentes. A condicional na linha 1 e 4 fazem o tratamento do caso especial em que $X = 0$. A chamada recursiva da função só é feita quando o conjunto C se mostra viável ou quando $limite > 0$. Se C não é viável então RECEB não será chamada para seus descendentes, o que não problema devido a Inviabilidade Hereditária. Logo, ignorar os descendentes de uma combinação não viável não altera o resultado do algoritmo, apenas agiliza o processo. As linhas 6 e 11 garantem que qualquer enlace que seja adicionado a C também seja removido. Finalmente, na linha 8, se C é viável, então é adicionado ao conjunto F e, por isso, F contém todos os conjuntos de enlaces viáveis. Portanto, o algoritmo está correto.

4.5.3 Análise da Complexidade

Complexidade de Espaço

$O(2^m)$

Complexidade de Tempo

As linhas 1-4, 7-12 são $O(1)$. A função de decodificação na linha 6 é $O(m)$ e o teste de viabilidade na linha 7 é $O(m^2)$. O laço principal das linhas 5-12 é $O(2^m m^2)$ pois a cada uma das 2^m iterações (no pior caso), a função de decodificação e o teste

de viabilidade são chamados com complexidade de $O(m) + O(m^2) = O(m^2)$. De maneira geral, a complexidade de tempo total é $O(2^m m^2)$.

Contudo, esse valor de complexidade é muito exagerado uma vez que, devido aos saltos, a contagem cobrirá apenas uma parte das 2^m combinações. Nesse caso, o números de conjuntos avaliados na contagem é $O(|F|)$, onde $|F|$ é a quantidade conjuntos viáveis da rede. Além disso, como será discutido no Capítulo 5, o m^2 pode ser simplificado por n^2. Portanto, a complexidade é $O(|F|n^2)$.

4.6 Conclusão

Referências Bibliográficas

[1] CISCO. "Cisco Visual Networking Index: Forecast and Methodology, 2013–2018". . http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html, jun. 2014.

[2] INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. "TOPODATA - Banco de Dados Geomorfométricos do Brasil". . http://www.dsr.inpe.br/topodata/acesso.php. Acessado em fevereiro de 2015.