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.

Prof. Valmir Carneiro Barbosa, D.Sc.

Eng. Raphael de Melo Guedes, 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, 29 p.: il.; 29, 7cm.

Orientadores: José Ferreira de Rezende

Valmir Carneiro Barbosa

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

Bibliografia: p. 29 – 29.

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.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

SINR	Signal to Interference plus Noise Ratio, p. 7
TIP	Teste de Interferência Primária, p. 6
TIS	Teste de Interferência Secundária, p. 8

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, é apresentado 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.

Além disso, serão apresentados a descrição formal para o problema de enumaração de conjuntos viáveis, o objetivo deste trabalho, e qual a metodolgia usada para alcançá-lo. Finalmente, é descrito como o restante do texto está organizado.

1.1 Motivação

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 signi-

fica 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 $S = \{s_1, s_2, \ldots, s_t\}$. 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 s_i sejam viáveis e o número de slots 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.

1.2 Descrição do Problema

Deseja-se verificar a viabilidade de todos os conjuntos de enlaces possíveis em uma rede através de testes de interferência. 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 a rede tenha m enlaces, então existe um total de 2^m conjuntos a serem testados. Com isso, a complexidade de tempo de um algoritmo que usa força bruta pura para explorar todos os conjuntos possíveis e testar sua viabilidade é $O(2^m)$. Isto é, o tempo de excução de tal algoritmo é tão grande que torna-se impraticável utilizá-lo em aplicações reais. Se a complexidade do teste também for considerada, o tempo de execução aumenta ainda mais.

1.3 Objetivo

Baseado na ineficiência das técnicas que usam força bruta, o objetivo deste trabalho é introduzir algoritmos que consigam resolver o problema de enumeração de conjuntos de enlaces viáveis com complexidades de tempo menores. Além disso, também será estudado o tempo real de execução dos algoritmos em diferentes cenários.

1.4 Metodologia

Nesse trabalho, o problema será modelado usando a ideia de árvore de combinações, que oferece propriedades fundamentais para se obter menores complexidades de

tempo. Baseando-se em tais propriedades, dois algoritmos que usam as propriedades provenientes da modelagem serão projetados e implementados. Além disso, outras técnicas para otimização serão estudadas e aplicadas.

Uma vez que os algoritmos são implementados, suas eficiências serão verificadas através de experimentos em redes simuladas. Um estudo do número de conjuntos viáveis e como ele varia com o tamanho da rede será feito para estudar as complexidades computacionais dos algoritmos. Finalmente, os tempos de execução serão avaliados.

1.5 Organização do Texto

No Capítulo 2, é apresentada a modelagem da rede na forma de um grafo, como as interferências são modeladas e como modelar os conjuntos de enlaces possíveis na forma de uma árvore de combinações. Como reunir tais modelos para projetar algoritmos e algumas vantagens e desvantagens.

Nos capítulos 3 e 4, o projeto dos algoritmos são apresentados. Os passos dos projetos são detalhados desde as técnicas fundamentais até a sua descrição propriamente dita. Finalmente, as vantagens e desvantagens das abordagens propostas são discutidas.

No capítulo 5, algumas técnicas de otimização adicionais são apresentadas. Depois, é feito um breve resumo das técnicas de implementação e ambientes utilizados. Conclui-se o capítulo descrevendo as simulações e analisando os resultados obtidos.

No capítulo **??**, as considerações finais são feitas e alguns trabalhos decorrentes desse projeto são propostos.

Capítulo 2

Modelagem

2.1 Introdução

2.2 Modelando Redes usando Grafos

Para modelar uma rede qualquer usando um grafo $G = (V, E)$, basta fazer as seguintes representações:

- Os nós da rede são os vértices do grafo, ou seja constituem o conjunto V. Por convenção, $|V| = n$
- Os enlaces da rede são as arestas do grafo, ou seja constituem o conjunto E. Por convenção, $|E| = m$

Para esse trabalho, é importante que as arestas de G sejam direcionadas já que os enlaces especificam a direção da comunicação. A figura abaixo mostra um exemplo de como tal modelagem pode ser feita para uma rede bem simples com $n = 5$ e $m = 4$.

Modelar uma rede como um grafo é bastante útil pois permite pegar emprestado algumas propriedades úteis da Teoria dos Grafos e fazer diversas manipulações matemáticas. Especificamente, nesse trabalho, a abstração em grafos também ajudou muito na etapa de implementação dos algoritmos.

2.3 Árvore de Combinações

Seja um grafo direcionado $G = (V, E)$ conforme aquele utilizado para modelar uma rede na seção anterior. O conjunto de arestas E representa todos os enlaces ativos para essa rede. É possível definir um subgrafo $C = (V, E)$ de G, tal que $V(C) = V(G)$ e $E(C)' \subseteq E(G)$. Com isso, tem-se que o menor tamanho de $E(C)$ é zero, ou

seja, não há enlaces ativos ($E(C) = \emptyset$), e o maior tamanho de $E(C)$ é o próprio $E(G)$ com todos os enlaces possíveis ativos ($E(C) = E(G)$). Entre esses dois extremos, existem 2^m combinações de enlaces possíveis, que são subconjuntos de G definidos pela escolha de quais enlaces estão ativos.

Sobre essas combinações de enlaces é possível definir uma árvore de combinações $A = (V, E)$. Os vértices da árvore são todas os subconjuntos possíveis de G. As arestas da árvore ligam qualquer subconjunto em uma altura h a outro em uma altura $h + 1$ que pode ser obtido através da adição de um enlace de G. Em A, o tamanho de um vértice C a uma altura h na árvore é tal que $m_c = h$.

O número de conjuntos possíveis com tamanho $h = m_c$ é dado pela combinação de m tomados m_c a m_c. Nessa árvore, parte-se da combinação vazia ($m_c = 0$) como raiz ($h = 0$). Os filhos de qualquer vértice $E(C)$ da árvore são obtidos combinando C com um enlace $i \in E$, tal que $i \ni C$. 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.

2.4 Modelos de Interferência

No capítulo 1, 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 o desenvolvimento de algoritmos de enumeração com melhores desempenhos.

2.4.1 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, alcançarem diversos nós, elas são direcionadas a 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. A figura abaixo mostra algumas redes com e sem essas características.

2.4.2 Teste de Interferência Primária

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. C passa no **Teste de Interferência Primária** (TIP) se, e somente se, $\forall i, j \in C, s_i \neq s_j \& s_i \neq r_j \& r_i \neq s_j \& r_i \neq r_j$.

Descrição do Algoritmo

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 anterior. Portanto, o algoritmo 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, $O(m)$.

2.4.3 Modelo de Interferência Secundária

Como mencionado no capítulo 1, 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.

A Potência de Recepção $RP(s, r)$ de uma transmissão entre s e r 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 ruído 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 $SINR(i, C)$, $\forall i \in C$, compara-se 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.

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

2.4.4 Teste de Interferência Secundária

No modelo de interferência secundária, 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).

Descrição do Algoritmo

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
```

Prova de Corretude

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

Análise da Complexidade

Como itera-se sobre todos os enlaces de C para calcular $SINR(i, C)$, sua complexidade 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 $O(m)$.

2.4.5 Viabilidade de Conjuntos

Dados os modelos de interferência apresentados, se um conjunto de enlace C passar em ambos os testes, então diz-se que C é viável. O algoritmo para testar a viabilidade de um conjunto de enlaces é simplesmente a junção dos algoritmos anteriores.

Descrição do Algoritmo

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

A condicional da linha 2 garante que ambos os testes são executados e, somente ao passar necessariamente nos dois, um conjunto C é classificado como viável. Portanto, o algoritmo está correto.

Análise da Complexidade

O pior caso para o teste de viabilidade é quando o conjunto é viável, ou seja, todas as iterações do TIP e do TIS ocorrem. Para o maior tamanho de C possível, temos que a complexidade de tempo desse algoritmo é $O(m) + O(m^2) = O(m^2)$.

2.4.6 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 viabilidade 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.

2.5 Exemplo com Busca em Profundidade

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: $\{\{\}, \{1\}, \{2\}, \{2, 4\}, \{3\}, \{3, 4\}, \{4\}\}$.

2.6 Vantagens e Desvantagens

No pior caso, o processo de enumeração descrito na seção anterior percorre todos os 2^m vértices da árvore, aplicando VIAVEL em cada conjunto de enlaces. Isso representa uma complexidade de $O(2^m m^2)$. Na prática, as redes possuem uma densidade grande. Por isso, é um exagero assumir que 2^m combinações serão testadas, pois certamente haverão muitos conjuntos não viáveis.

Seja F o conjunto de combinações viáveis obtido por meio de um processo de enumeração como o da seção anterior. Certamente, $|F|$ combinações foram testadas, caso contrário não fariam parte de F. Na maioria dos casos, os conjuntos não viáveis são "podados"de forma que só um primeiro conjunto é testado. Supondo o pior caso, sempre depois de um conjunto viável com maior altura em um ramo da árvore ser testado, encontra-se um conjunto não viável. Nesse caso, o número de combinações testadas é igual a $|F| + |U|$, onde U é o conjunto das primeiras combinações não viáveis testadas depois de uma viável. É possível ver que $|F| \approx |U|$ e, portanto, o número de conjuntos testados é $2|F|$.

Existe, então, um bom candidato a substituir a porção exponencial da complexidade do algoritmo mencionado. Como $O(|F|)$ conjuntos serão testados (viáveis ou não), a complexidade pode ser alterada para $O(|F|m^2)$. Até o momento, $|F|$ é desconhecido, mas certamente varia em função do tamanho da rede. No Capítulo 5, uma função que represente os valores de $|F|$ usando m e n como parâmetros é aproximada, definindo melhor o valor da complexidade.

Mesmo com uma potencial redução da complexidade de tempo ao utilizar o valor $|F|$, a complexidade de espaço desse algoritmo ainda é exponencial. Isso significa que sua implementação está limitada a pequenos valores de m.

2.7 Conclusão

Nesse capítulo, todo o problema foi modelado usando diferentes abstrações com o intuito de obter um procedimento sistemático para enumerar combinações viáveis. Devido a essa modelagem, a complexidade de tempo encontrada em algoritmos que usam força bruta pode ser modificada para um valor diferente, que será estudado em capítulos futuros. Ainda assim, a complexidade de espaço continua exponencial, inviabilizando sua utilização na a maioria das aplicações.

Capítulo 3

Projeto do Algoritmo Iterativo

3.1 Introdução

No Capítulo 2, uma abordagem foi utilizada para enumerar os conjuntos viáveis no exemplo proposto. Essa abordagem, possui elevada complexidade computacional e, por isso, não representa um bom candidato a ser implementado. Nesse capítulo, o algoritmo projetado será a primeira alternativa a essa abordagem.

O algoritmo introduzido aqui é composto por três partes básicas: codificação/decodificação, busca, e "poda". Ao reunir esses elementos, obtém-se um método iterativo para enumerar os conjuntos viáveis. Cada uma dessas partes será detalhada antes da descrição do algoritmo ser apresentada. Ao final, será feita uma análise de sua complexidade e comparação com o desempenho da abordagem anterior.

3.2 Representando Combinações Usando Inteiros

3.2.1 Codificação

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 $C \in E$. O número B codifica a combinação C quando:

$$b_i = \begin{cases} 1, \text{ sse } e_i \in C \\ 0, \text{ caso contrário} \end{cases}$$

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.

3.2.2 Decodificação

Para conhecer os enlaces pertencentes a uma combinação e realizar o teste de viabilidade apropriadamente, é essencial que tal número possa ser decodificado em uma combinação de enlaces. 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.

Descrição do Algoritmo

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))
```

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.

Análise de Complexidade

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 $O(m)$. Para o processo de decodificação é preciso armazenar, no máximo, $m - 1$ enlaces e, portanto, sua complexidade de espaço é O(m).

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 $B = 0$) à sua folha de maior altura ($C = E$ ou $B = 15$). 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 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.

A Figura XXX mostra como a árvore do exemplo anterior é percorrida através da contagem. Comparado com o método de busca adotado no capítulo anterior, 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, uma combinação C que foi codificada em B tal que b_i^* é o bit ativo menos significativo tem exatamente 2^i–1 descendentes na árvore.

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 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 Algoritmo Iterativo para Enumeração de Conjuntos de Enlaces Viáveis

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.

Descrição do Algoritmo

Algoritmo 5: Algoritmo ITERATIVO

```
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
```

Prova de Corretude

De acordo com o que foi provado na seção XXX, 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 XXX 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.

Devido a inviabilidade hereditária, 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 e, portanto, está correto.

Análise de Complexidade

O processo de decodificação apresentado é responsável por tornar desnecessário o armazenamento de todas as combinações de enlaces possíveis, como precisava ser feito na abordagem do capítulo anterior. Portanto, a complexidade de espaço é reduzida de $O(2^m)$ para $O(m)$.

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 $O(2^m m^2)$.

Analogamente ao que foi explicado no capítulo anterior, esse valor de complexidade é muito exagerado e também cabe fazer a substituição de 2^m por $|F|$. Portanto, a complexidade é $O(|F|m^2)$.

3.6 Conclusão

Nesse capítulo, uma ideia de como percorrer a árvore em busca de combinações viáveis foi formalizada na forma de um algoritmo iterativo. Diferente da abordagem do capítulo anterior, devido ao processo de decodificação, apenas o conjunto de enlaces precisa ser armazenado, reduzindo consideravelmente sua complexidade de espaço. Contudo, sua complexidade de tempo continua a mesma do caso anterior.

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 introduzido no Capítulo 3.

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 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.

Além disso, a SINR já é calculada no momento em que um novo enlace é adicionado ao conjunto. Logo, para esse caso, a complexidade do algoritmo TIS é reduzida. Como as SINR já estão calculadas, basta iterar sobre os enlaces de C, fazendo a complexidade de tempo diminuir de $O(m^2)$ para $O(m)$. Como será detalhado no Capítulo 5, o tempo do TIS também será reduzido para $O(m)$. Consequentemente, o algoritmo VIAVEL para o caso recursivo terá complexidade $O(m) + O(m) = O(m)$.

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 Recursivo para Enumeração de Conjuntos de Enlaces Viáveis

Descrição do Algoritmo

Algoritmo 6: Algoritmo RECURSIVO

```
1  input: Grafo direcionado G = (V, E), Conjunto de Enlaces C, Inteiro X
2  output: Conjuntos de Enlaces Viáveis F
3  if X = 0 then
4      for i = 0 to m − 1 do
5          RECURSIVO(G, C, X + 2^i)
6  else
7      limite ← log2(X& (X − 1))
8      C ← C ∪ {e_limite}
9      if VIAVEL(C) then
10         F ← F ∪ C
11         for i = 0 to limite − 1 do
12             RECURSIVO(G, C, X + 2^i)
13         C ← C e_limite
```

Prova de Corretude

Como mostrado na seção 4.2, se $X = 0$, então toda a árvore é percorrida chamando o algoritmo RECURSIVO 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 RECURSIVO 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.

Análise da Complexidade

Assim como no caso do algoritmo ITERATIVO, a complexidade de espaço é $O(m)$.

A complexidade de tempo desse algoritmo é parecida com a do algoritmo ITERATIVO. Contudo, como houveram algumas otimizações decorrentes do reaproveitamento dos cálculos da SINR, sua complexidade é um pouco menor. Houve uma redução de $O(|F|m^2)$ para $O(|F|m)$.

4.6 Conclusão

Apesar de uma concepção um pouco mais complexa, o algoritmo RECURSIVO obteve melhor complexidade de tempo em comparação com todas as técnicas apresentadas até agora. Tudo isso, devido a reutilização dos cálculos proporcionada por sua natureza recursiva.

Capítulo 5

Implementação e Resultados

5.1 Introdução

Nos capítulos anteriores, os algoritmos ITERATIVO e RECURSIVO foram introduzidos trazendo menores complexidades do que a força bruta pura. Neste capítulo, será estudado o que tais complexidades representam em termos de recursos computacionais reais, ou seja, quanto de espaço em memória e tempo de processamento eles precisam.

Além disso, detalhes de implementação relevantes como teconologias e ambientes adotados são descritos. Finalmente, todo o plano de simulações será detalhado e os resultados são analisados.

5.2 Otimizações

Antes de falar da implementação, é importante mostrar algumas ideias que ajudaram a reduzir ainda mais a complexidade dos algoritmos.

5.2.1 Acoplamentos em Grafos

Em Teoria dos Grafos, um acoplamento M de um grafo $G = (V, E)$ é definido como um subconjunto de $E(G)$, tal que, para qualquer par de arestas $(i, j) \in M$, i e j não são adjacentes, ou seja, não compartilham vértices. Um acoplamento máximo M^* é aquele com o maior número possível de arestas.

A partir dessas definições, é possível extrair duas propriedades fundamentais:

1. Para qualquer M em $G = (V, E)$, $g(v) \leq 1, \forall v \in V(G)$
2. Para qualquer M^* em $G = (V, E)$, $|M^*| = \lfloor n/2 \rfloor$

5.2.2 Limitando a Busca

O Modelo de Interferência Primária, devido algumas características, não permite que nós da rede estejam participando de mais de um enlace de comunicação ao mesmo tempo. Logo, em uma rede modelada por um grafo $G = (V, E)$, um conjunto de arestas $C \subset E(G)$ é viável se, e somente se, as arestas de C não comparilham vértices, ou seja, não são adjacentes entre si.

Dada a definição na seção anterior, um conjunto viável C é um acoplamento M de G. Portanto, o maior tamanho possível para um conjunto viável C^* é o tamanho de um acoplamento máximo M^*, ou seja, $|C^*| = |M^*| = \lfloor n/2 \rfloor$. Com essa informação, é possível "podar"todos os vértices C da árvore de combinações que tenham $|C| > n/2$, conforme exemplificado na Figura 5.1.

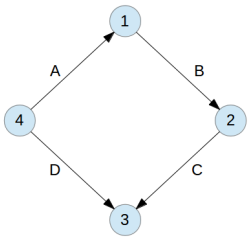

Figura 5.1: Grafo com $n = m = 4$

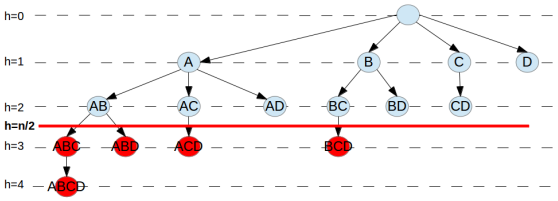

Figura 5.2: Árvore "podada em $n/2$

Usando essa técnica, a complexidade para percorrer a árvore cai de $O(2^m)$ para $(2^{n/2})$.

5.2.3 Reformulando o TIP

Como consequência da definição de acoplamento, é possível reescrever o algoritmo 1 com uma menor complexidade.

Descrição do Algoritmo

Algoritmo 7: Algoritmo TIP-G

```
1 input: Conjunto de enlaces C
2 foreach i ∈ C do
3     if ((g(s_i) > 1)||(g(s_i) > 1)) then
4         return FALSE
5 return TRUE
```

Prova de Corretude

Pela propriedade 1, se, dado um conjunto $C \in E(G)$, $\forall v \in V(G), g(V) \leq 1$, então C é um acoplamento e, portanto, C é viável. Isso garante que as arestas de C não compartilham vértices, ou seja, os nós do conjunto apenas participam de um enlace. Então, para testar a Interferência Primária, não é necessário verificar se em cada par de enlaces existem nós compartilhados. Basta verificar que o grau dos vértices nos enlaces de C são menores que 2. Como essa verificação é feita na linha 3, o algoritmo está correto.

Análise de Complexidade

A complexidade de espaço é O(m). Devido a otimização anterior, $m \leq n/2$, então a complexidade é reduzida a $O(n/2) = O(n)$.

Como todos os enlaces de C são avaliados, então a complexidade é O(m). Contudo, sabemos que $|C| \leq n/2$, portanto, a complexidade de espaço é reduzida para $O(n)$

5.3 Detalhes de Implementação

Todos os algoritmos aqui apresentados, bem como as estruturas de dados subjacentes, foram implementados usando a liguagem de programação C++ sob o paradigma de Orientação a Objetos. O ambiente de desenvolvimento foi composto do sistema operacional Ubuntu 16.04 64 bits com o compilador GCC 5.4.0-6, usando o editor Atom para produção do código. O hardware utilizado foi um processador Intel Core i5-3337U com 6GB de RAM.

A primeira porção de código produzido foi referente a modelagem da rede. As classes Node, Link e Network representam os nós, enlaces e a rede, respectivamente. Além dos dados relevantes aos cálculos efetuados, como posição dos nós, potência de recepção, entre outros, vários métodos também foram criados, como geração aleaória da posição dos nós, cálculo da distância entre nós e assim por diante. O objeto da classe Network é servir de entrada para os algoritmos.

Para geração das redes, três parâmetros são utilizados: (i) potência de transmissão TP; (ii) número de nós N; (iii) tamanho do lado A da área quadrada onde os nós serão dispostos. A primeira coisa a ser feita é distribuir os N nós aleatoriamente dentro da área definida por A. A distância entre todos os pares de nós é então calculada. Se a distância entre os nós for menor que um limite D, então um enlace é criado para conectá-los usando a equação 2.1.

Devido algumas facilidades para gerenciar variáveis compartilhadas, os algoritmos ITERATIVO e RECURSIVO também foram codificados como classes, Iterative

e Recursive, respectivamente. Ambas as classes possuem o método construtor e o método principal que implementa o algoritmo propriamente dito, além de alguns outros métodos secundários para manipulação dos dados.

A arquitetura do ambiente descrito é o principal limitante dos algoritmos. Como os processadores são de 64 bits, o maior número inteiro nativo oferecido pela linguagem C++ é $2^{64}-1$. Portanto, até o momento, não é possível aplicar esses algoritmos a redes com mais de 64 enlaces. Devido a essa limitação, alguns cuidados também tiveram que ter sido tomados. Por exemplo, em C++, em uma contagem, ao exceder o número $2^{64}-1$, a contagem é retomada do 0. Se nenhum mecanismo para detectar essa situação fosse implantado, ao executar os algoritmos para $m = 64$, o programa iria entrar em loop infinito. Um outro caso é baseado na inicialização da variável *limite*. No algoritmo ITERATIVO, $limite = 2^m$, logo, se $m = 64$, então $limite = 0$ e, portanto, o programa seria encerrado antes mesmo de entrar no loop principal. Mais uma vez, um mecanismo teve de ser adotado para contornar essa situação.

5.4 Simulações e Resultados

Verificação da variação da quantidade de conjuntos de enlaces viáveis

É importante verificar a variação da quantidade de conjuntos de enlaces viáveis resultantes da execução dos algoritmos em função do tamanho da rede para melhor determinar a sua complexidade computacional. Para isso, 30 redes diferentes para cada valor de m e n foram geradas. Mais uma vez, $n = \{16, 32, 64, 128\}$.

Em cada uma dessas redes, o número de conjuntos viáveis foi computado. A média dos valores encontrados foi utilizada para gerar uma curva e, com isso, poder aproximar a complexidade dos algoritmos.

Assim como no caso anterior, as curvas se acentuam com o aumento de n. Isso significa que a variação do número de conjuntos viáveis torna-se mais sensível ao aumento do número de enlaces a medida que maiores valores de m são utilizados.

O método de regressão linear foi utilizado para interpolar as curvas subjacentes aos gráficos da figura anterior. Usando esse método, foi possível encontrar uma função de f(m) que melhor se aproxima dos pontos, ou seja, que melhor descreve o comportamento do número de conjuntos em função de m. As funções encontradas para os diferentes valores de n são mostradas da tabela a seguir.

Em todos os casos, $f_n(m)$ é $O(n^2)$. Logo, é possível afirmar que $O(|F|) = O(n^2)$. Conclui-se que, as complexidades dos algoritmos ITERATIVO e RECURSIVO são, respectivamente, $O(|F|n^2) = O(n^2n^2) = O(n^4)$ e $O(|F|n) = O(n^2n) = O(n^3)$. Para o algoritmo RECURSIVO, a complexidade de tempo é menor do que a força bruta pura, que é $O(|F|n^2) = O(n^2n^2) = O(n^4)$.

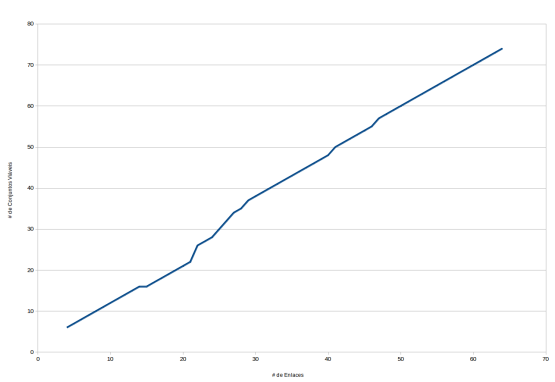

Figura 5.3: Variação de $|F|$ para $n = 16$

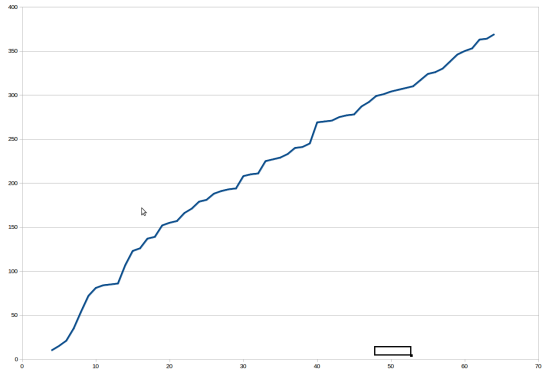

Figura 5.4: Variação de $|F|$ para $n = 32$

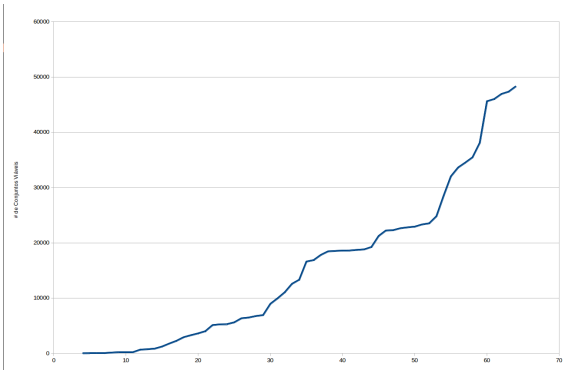

Figura 5.5: Variação de $|F|$ para $n = 64$

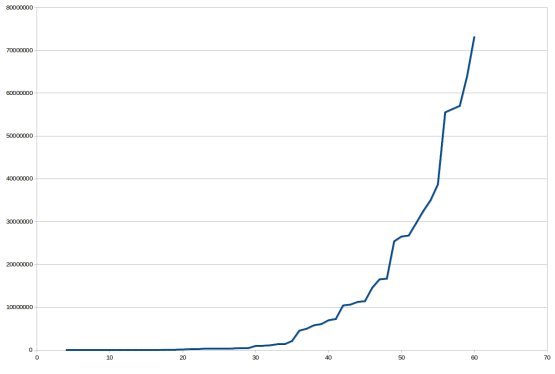

Figura 5.6: Variação de $|F|$ para $n = 128$

Complexidades

Agora que o valor de $O(|F|)$ foi encontrado, pode-se resumir as complexidades computacionais dos algoritmos apresentados nesse trabalho.

Verificação do tempo de execução real dos algoritmos

Para alcançar esse objetivo, um programa foi criado para variar o valor do lado da área A com o intuito de simular redes de 4 a 64 enlaces. Além do tempo de execução de cada algoritmo para as 60 redes geradas, também se mostrou importante entender o efeito da variação do número de nós. Os valores utilizados para n são $\{16, 32, 64, 128\}$.

Então, as redes foram simuladas com diferentes valores de m e n e o tempo de execução de ambos os algoritmos foi medido. Para evitar ruídos provenientes de variações computacionais aleatórias, 30 execuções de cada algoritmo foram feitas para cada cenário, seguindo a regra de [Citar o cara que determinou esse número]. Ao final, o tempo médio das execuções foi utilizado para gerar uma curva e, com isso, poder analisar o tempo real.

Tabela 5.1: Funções Aproximadas.

n	$f_n(m)$
16	$f_{16}(m) = am^2 + bm + c$
32	$f_{32}(m) = am^2 + bm + c$
64	$f_{64}(m) = am^2 + bm + c$
128	$f_{128}(m) = am^2 + bm + c$

Tabela 5.2: Complexidades Computacionais.

Algoritmo	Tempo	Espaço
Força Bruta Pura	$O(2^m m^2)$	$O(2^m)$
BP com "Poda"	$O(m^4)$	$O(2^m)$
ITERATIVO	$O(m^4)$	$O(m)$
RECURSIVO	$O(m^3)$	$O(m)$
ITERATIVO + Otimizações	$O(n^4)$	$O(m)$
RECURSIVO + Otimizações	$O(n^3)$	$O(m)$

O primeiro fato a ser notado é que o algoritmo recursivo obteve menores tempos de execução para todos os valores de n. Isso reflete a menor complexidade de tempo teórica que algoritmo RECURSIVO tem em comparção com o algoritmo ITERATIVO. Em ambos os algoritmos, com o aumento de n, suas curvas se aproximam cada vez mais de uma exponencial, confirmando suas complexidades computacionais.

5.5 Conclusão

Nesse capítulo, a redução da complexidade computacional de O(m) para O(n) em algoritmos que iteram sobre os enlaces de um conjunto foi justificado através de técnicas de otimização. Em seguida, foi descrito um panorama sobre as técnicas de implementação e ambiente utilizados.

Finalmente, usou-se o resultado das simulações para entender qual o impacto das complexidades computacionais teóricas quando os algoritmos são utilizados na prática. Além disso, também foi possível verificar o tamanho de —F— em função de m, que permitiu concluir a superioridade dos algoritmos aqui apresentados em relação aos algoritmos ingênuos e baseados em força bruta.

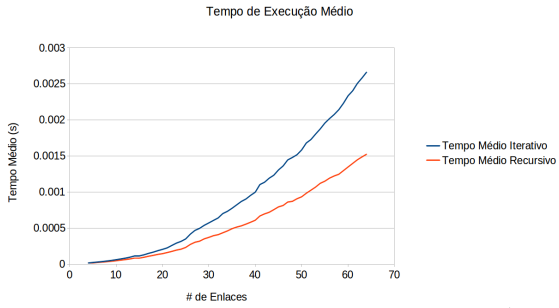

Figura 5.7: $n = 16$

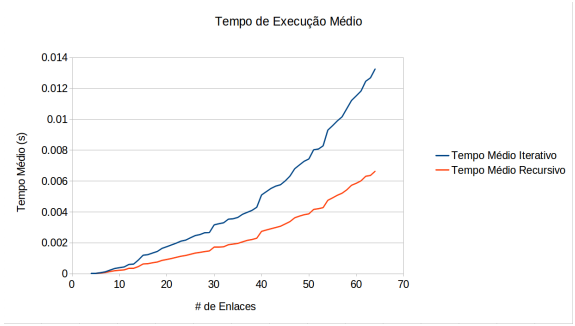

Figura 5.8: $n = 32$

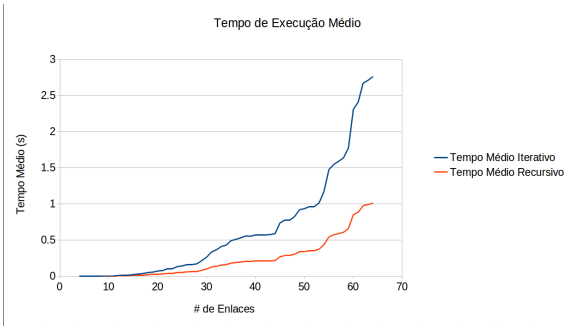

Figura 5.9: $n = 64$

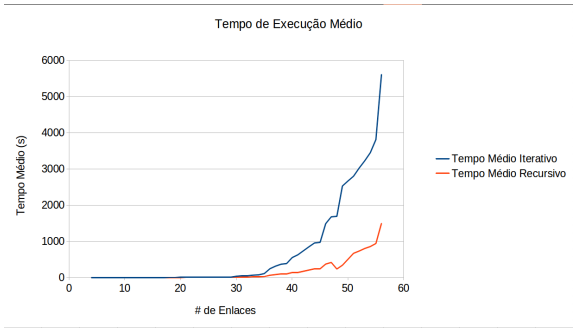

Figura 5.10: $n = 128$

Bibliografia

[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.