ALGORITMO EFICIENTE DE ENUMERAÇÃO DOS CONJUNTOS DE ENLACES VIÁVEIS EM REDES SEM FIO

Guilherme Iecker Ricardo

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores:
José Ferreira de Rezende
Valmir Carneiro Barbosa

Rio de Janeiro

Setembro de 2016

ALGORITMO EFICIENTE DE ENUMERAÇÃO DOS CONJUNTOS DE ENLACES VIÁVEIS EM REDES SEM FIO

Guilherme Iecker Ricardo

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA 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.

Examinado por:

_____
Prof. José Ferreira de Rezende

_____
Prof. Valmir Carneiro Barbosa

_____
Prof. Marcia Rosana Cerioli

_____
Prof. Daniel Ratton Figueiredo

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2016

Ricardo, Guilherme Iecker
Algoritmo Eficiente de Enumeração de Conjuntos de Enlaces Viáveis em Redes Sem Fio/ Guilherme Iecker Ricardo. - Rio de Janeiro: UFRJ/ Escola Politécnica, 2016.
X, 57 p.: il.; 29,7 cm.
Orientadores: José Ferreira de Rezende / Valmir Carneiro Barbosa
Projeto de Graduação - UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2016.
Referências Bibliográficas: p. XX-YY.
1. Introdução 2. Fundamentos Teóricos 3. Trabalhos Relacionados 4. Algoritmos para Enumeração de Matchings 5. Resultados 6. Conclusão. I. Rezende, José Ferreira II. Barbosa, Valmir Carneiro III. Universidade Federal do Rio de Janeiro, Escola Politécnica, Engenharia de Computação e Informação IV. Algoritmo Eficiente para Enumeração de Matchings em Grafos Genéricos.

Aos meus pais

Agradecimentos

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

ALGORITMO EFICIENTE DE ENUMERAÇÃO DOS CONJUNTOS DE ENLACES VIÁVEIS EM REDES SEM FIO

Guilherme Iecker Ricardo

Setembro/2016

Orientadores:
José Ferreira de Rezende
Valmir Carneiro Barbosa

Curso: Engenharia de Computação e Informação

/////

Palavras-chave: Redes de Computadores, Otimização Combinatória, Teoria dos Grafos, Projeto de Algoritmos

Abstract of Undergraduate Project presented to Escola Politécnica/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

EFFICIENT ALGORITHM FOR ENUMERATING FEASIBLE SETS OF LINKS IN WIRELESS NETWORKS

Guilherme Iecker Ricardo

September/2016

Advisors:
José Ferreira de Rezende
Valmir Carneiro Barbosa

Course: Computer and Information Engineering

//Abstract

Keywords: Computer Networks, Combinatory Optimization, Graph Theory, Algorithms Design

Sumário

Lista de Abreviaturas

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nesse capítulo, vamos apresentar intuitivamente através de exemplos e, depois, definir formalmente o problema que motivou esse trabalho.

Capítulo 2

Enumeração de Conjuntos de Enlaces Viáveis

Nesse capítulo, será apresentado o problema de enumeração de conjuntos de enlaces viáveis em redes sem fio.

Capítulo 3

Trabalhos Relacionados

Capítulo 4

Algoritmo Iterativo

O algoritmo iterativo é a primeira alternativa à busca de combinações de enlaces viáveis na árvore de combinações que vimos no capítulo 2. Trata-se de um algoritmo bastante simples e que, ainda assim, consegue apresentar bons resultados.

4.1: Codificando Combinações Usando Inteiros

A primeira abstração que devemos fazer é pensar nas 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, vamos apresentar essa modelagem fazendo uso de uma linguagem mais formal, visando permitir algumas manipulações matemáticas importantes ao funcionamento do algoritmo.

Considere o grafo direcionado G=(V,E). Podemos enumerar as arestas em E como e_0, e_1, e_2, ..., e_{m-1}. Seja B um número inteiro na base 2 com m bits. Podemos enumerar os bits em B como b_0, b_1, b_2, ..., b_{m-1}. Considere uma combinação de arestas C tal que C [está contido em] E. Dizemos que B codifica C quando:

b_i = 1 se, e somente se, e_i [pertence a] C

= 0, caso contrário.

Podemos aplicar essa ideia de codificação em toda a árvore de combinações que apresentamos no capítulo anterior. Ao fazer isso, todos os 2^m vértices da árvore serão codificados em inteiros.

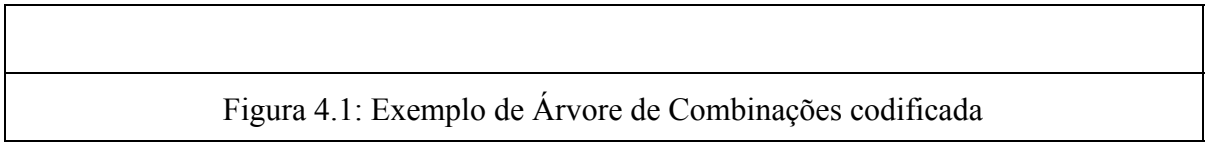

Figura 4.1: Exemplo de Árvore de Combinações codificada

Como exemplo, na Figura 4.1, apresentamos a 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.

Ao realizar essa codificação, podemos observar duas consequências, que em um primeiro momento podem parecer triviais, mas são o arcabouço para podermos escrever o algoritmo. A primeira está relacionada a como percorrer e a segunda a como "podar" a árvore.

4.2: Percorrendo a Árvore de Combinações

No exemplo da Figura 4.1, vimos que, para uma rede com 4 enlaces, temos uma árvore de combinações com 16 vértices. Partimos da raiz (C=[vazio] ou B=0) e vamos até a folha de maior altura (C=E, ou B=15). De forma geral, para um grafo com m enlaces, temos uma árvore com 2^m vértices, partindo de B=0 e chegando a $B=2^{m-1}$.

Na árvore codificada, a ordem que consideramos os enlaces 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, podemos 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 4.2: Exemplo de como percorrer a Árvore de Combinações através da contagem

Na Figura 4.2, vemos como a árvore do exemplo anterior é percorrida através da contagem. Se compararmos com a ordem em que os vértices da árvore são visitados usando os algoritmos clássicos apresentados no capítulo anterior, podemos ver que trata-se de uma nova ordem, que somente é possível devido a codificação aqui introduzida.

4.3: "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, vamos usar o método de busca baseado na contagem apresentado na seção anterior para detectar alguns padrões que nos ajudarão a "podar" também de maneira eficiente. Mais uma vez, a correspondência (bijeção) 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 bi podem nos fornecer mais informações úteis. Vamos pensar no caso em que b_i* é o bit ativo menos significativo (ou seja, é o bit 1 mais a direita). Nesse caso, o índice i, além de servir de referência ao enlace $e_{i,}$ também representa o número de filhos que B tem na árvore, ou seja, quantas combinações são consequências diretas da atual. Em termos de bits, i representa os números que são resultantes da alternagem dos bits 0 a 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 descedentes.

No capítulo 2, chamamos de "poda" o ato de ignorar os descendentes de uma combinação não viável ao percorrer uma árvore. Para o presente caso, como estamos realizando uma contagem, vamos usar o termo "salto". Portanto, uma vez que uma

combinação codificada B não for viável, vamos realizar um “salto” sobre todos os seus descendentes.

Como nós sabemos o número de descendentes de uma combinação B através de b_i*, caso seja necessário realizar um “salto” na contagem por B não ser viável, basta incrementar a contagem com 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.

Figura 4.3: Exemplo de contagem com “saltos”

Na Figura 4.3, apresentamos um exemplo de cenário em que os “saltos” ocorrem. Na rede do exemplo, sabemos 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, não estamos listando as combinações de enlaces viáveis, apenas estamos mostrando a sequência de enlaces visitados na contagem.

4.4: Decodificando um Conjunto de Enlaces

Mesmo que agora nós possamos percorrer a árvore e “saltar” sobre os descendentes de uma combinação inviável, não podemos testar a viabilidade de um conjunto de enlaces usando apenas um número inteiro. Precisamos decodificar o número inteiro em um conjunto de enlaces para conhecermos os nós da rede envolvidos e, com isso, realizar os testes. A seguir, apresentamos um algoritmo recursivo para decodificar um inteiro em um conjunto de enlaces.

Algoritmo Decodificador
Entrada: Número inteiro B, número inteiro I, conjunto de arestas E
Saída: Conjunto de enlaces C

1. Q <- B / 2
2. R <- B%2

3. Se R=1, então
4. C<-C [união] e_I
5. Se Q>0, então
6. I <-I + 1
7. Decodificador(Q, I, G)

4.4.1: 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 conseguirmos capturar os enlaces pertencentes ao conjunto.

Nesse algoritmo temos 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. A variável Q nos indica quando devemos parar o algoritmo. Se Q>0, continuamos o processo incrementando o valor de I e verificando a pertinência dos enlaces com novos índices I. Se Q=0, o processo de conversão é finalizado.

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

4.4.2: Análise de Complexidade

Complexidade de Espaço

Entrada

B O(1)

I O(1)

E O(m)

Saída

	C	O(m)
Auxiliares		
	Q	O(1)
	R	O(1)
Total		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 tempo como O(m).

4.5: Descrição do Algoritmo

Agora que sabemos como percorrer a árvore de combinações, "saltar" sobre os descendentes das combinações inviáveis e decodificar os inteiros para realizar os testes de interferência, podemos descrever formalmente do algoritmo.

Algoritmo Iterativo de Enumeração de Conjuntos de Enlaces Viáveis
Entrada: Grafo direcionado G=(V,E)
Saída: Lista dos conjuntos dos enlaces viáveis F

1. F ← 0 (vazio)
2. C ← 0 (vazio)
3. B ← 0
4. I ← 0
5. Enquanto $B < 2^m - 1$, faça
6. C ← Decodificador(B, 0, E(G))

7. Se VIÁVEL(C), então
8. F ← F [união com] {B}
9. I ← 1
10. Senão
11. I ← 2^i
12. B ← B + I

4.5.1: Prova de Corretude

De acordo com o que foi provado na seção 3.2, sabemos que o laço instanciado na linha 5 e o incremento feito na linha 12 representam uma contagem e, de fato, faz 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 3.3 são baseados na variável I, usada como incremento à variável B. Pelo algoritmo, I somente possui dois valores possíveis.

I=1, se B é viável

I=2^m, caso contrário

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

Pela característica do problema, como mencionado nos capítulos anteriores, temos certeza que todos os 2^i descendentes de uma combinação inviável 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.

4.5.2: Análise de Complexidade

Complexidade de Espaço

Entrada

	G	O(m+n)
Saída		
	F	$O(2^m)$
Auxiliares		
	C	O(m)
	B	O(1)
	I	O(1)
Funções		
	viável(C)	O(X)
	decodifica(B, G)	O(m)
Total		(2^m)

Complexidade de Tempo

As linhas 1-4, 7-12 são O(1). A função de decodificação na linha 6 é O(m). O laço principal das linhas 5-12 é $O(2^m m)$ pois a cada uma das 2^m iterações, a função de decodificação é chamada com complexidade de O(m).

A complexidade de tempo total é $O(2^m m)$.

4.6: Detalhes de Implementação

>> falar das limitações

4.7: Conclusão

Capítulo 5

Algoritmo Recursivo

Capítulo 6

Teste de Padrões e Paralelismo

Capítulo 7

Resultados e Discussões

Capítulo 8

Conclusão e Trabalhos Futuros

Referências Bibliográficas