UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Algoritmos e Estruturas de Dados III
Trabalho Prático 2 – Uaibucks

Orlando Enrico Liz Silvério Silva

Belo Horizonte
21 de novembro de 2017

1 Introdução

No domínio da Ciência da Computação, A Teoria dos Grafos é o estudo do objeto matemático denominado grafo e a relação entre seus vértices e arestas. Qualquer cenário em que seja preciso examinar uma estrutura similar a uma rede provavelmente envolve um problema que pode ser modelado a partir de um grafo. Eles são usados para representar redes de comunicação, organização de dados, fluxos de computação, dentre outros. O desenvolvimento de algoritmos que lidam com diversos tipos de grafos é de grande importância para a resolução de problemas que temos atualmente.

Dentre esses problemas, alguns se destacam por serem NP-Completo. Isso significa que só podem ser resolvidos em tempo polinomial utilizando uma máquina não-determinística. Porém, existem aproximações com um ferramental matemático indicando o quão próximo elas são da solução exata. Sendo assim, este trabalho tem como objetivo desenvolver uma solução exata e uma heurística para um problema NP-Completo de grafos.

2 Solução do Problema

Esta seção tem como objetivo apresentar a solução exata e a heurística implementada. Para tanto, inicialmente será introduzida a modelagem proposta, ilustrando as etapas da resolução do problema.

2.1 Modelagem do Problema

A entrada consiste, na primeira linha, de dois inteiros positivos: N ($1 \leq N \leq 30$ para o algoritmo exato e $1 \leq N \leq 10^5$ para a heurística), indicando o número de esquinas de uma determinada cidade e M ($1 \leq M \leq N(N-1)/2$)) indicando o número de pares de esquinas vizinhas. Cada esquina é identificada por IDs que vão de 1 a N e, cada uma das N linhas seguintes possui um inteiro natural d ($d \geq 0$) indicando a demanda de clientes da i-ésima esquina. Seguem então M linhas, cada uma contendo dois inteiros x e y ($1 \leq x \leq N$ e $1 \leq y \leq N$ e $x \neq y$) indicando que as esquinas x e y são vizinhas.

Sendo assim, as esquinas podem ser modeladas em um grafo como vértices e as esquinas vizinhas de cada esquina estabelecem uma conexão por aresta e demonstram que o grafo é não direcionado. As demandas de cada filial indicam que o grafo possui peso nos vértices.

O problema então consiste em encontrar subconjunto de esquinas (vértices) para abrir novas filiais de modo que duas filiais não estejam em esquinas vizinhas e a demanda seja maximizada. Ou seja, deve-se escolher o conjunto independente com peso máximo.

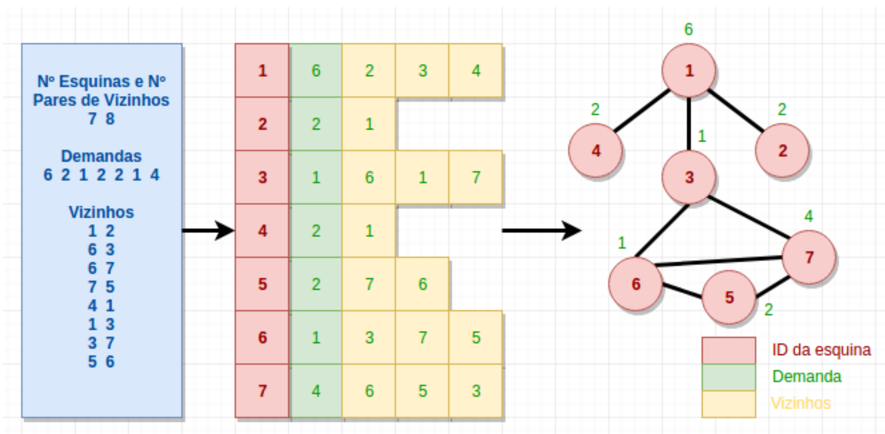

Figura 1 – Exemplo de entrada e modelagem em Grafo

.

Seja um grafo $G = (V, A)$: um conjunto independente de vértices V_{IND} de G é um subconjunto de V em que não existe nenhuma aresta entre qualquer par de elementos de V_{IND}. Porém, ainda deve ser levado em conta os pesos dos vértices que, para o problema, deve ser máximo.

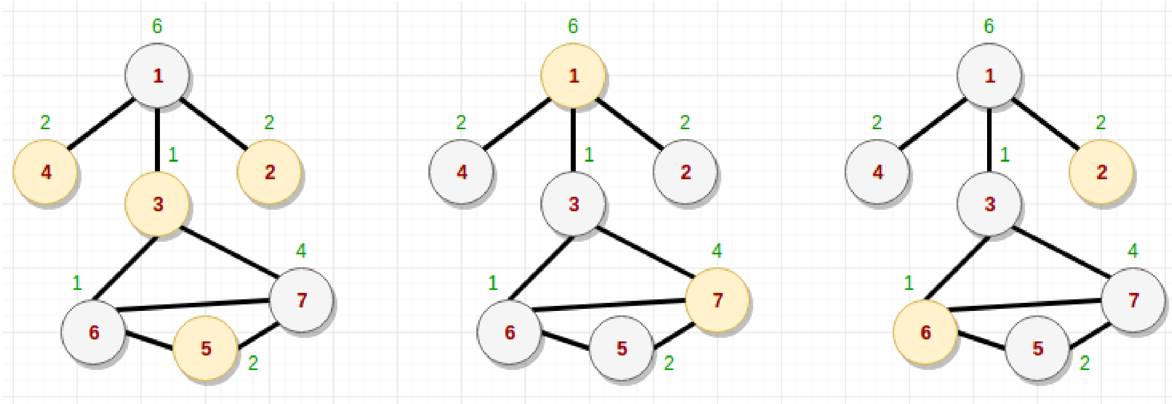

Figura 2 – Exemplo de conjuntos independentes de vértices com base na entrada anterior
.

2.2 NP-Completo

Pode-se então reformular o problema em questão, indicado como NP-difícil por Bryan como um problema de decisão. Dado um grafo $G = (V, A)$ é possível encontrar um conjunto de vértices independentes com $|V| > x$ tal que x seja constante?

Um problema de decisão π que seja NP-difícil pode ser mostrado ser NP-completo exibindo um algoritmo não-determinístico polinomial para π. Para provar que ele é NP-completo é necessário:

- **Mostrar que ele está em NP**: uma maneira é encontrar um algoritmo determinístico polinomial para verificar que uma dada solução é válida.

Algorithm 1 Verificação de Solução

```
for i ∈ {0, ..., |V|} do
    for j ∈ {i, ..., |V|} do
        if vizinhos(i, j) then
            insucesso
        end if
    end for
end for
if |V| > x then
    sucesso
end if
```

- **Mostrar que um problema NP-completo conhecido pode ser polinomialmente transformado para ele**: Dada uma fórmula $3CNF$ determinada ϕ ($CNF - SAT$ é sabidamente NP-Completo) o objetivo é construir um grafo G de tal forma que ele tenha um conjunto independente de tamanho x se e somente se ϕ é satisfazível. Ele deve ser construído em tempo polinomial.
  Fazendo a redução, G terá um vértice para cada literal de uma cláusula. Então, são conectados os 3 literais da cláusula para formar um triângulo. O conjunto independente irá escolher, no máximo, um vértice de cada cláusula, que irá corresponder ao literal ao qual foi atribuído o valor verdadeiro. Conecta-se 2 vértices se são literais complementares para

garantir que literais participantes do conjunto independente não tenham conflitos. Sendo assim, pode-se escolher uma atribuição que torne os x literais verdadeiros e satisfaça ϕ.

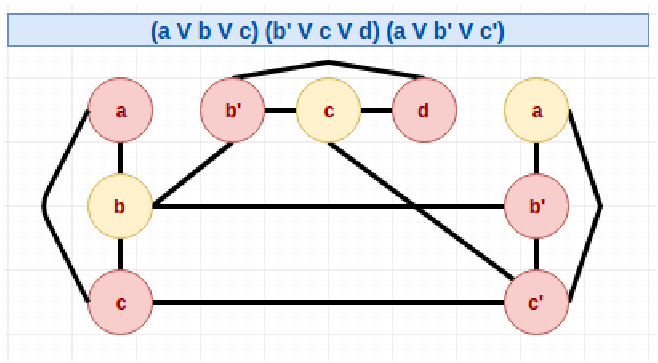

Figura 3 –Modelagem da Redução

.

Portanto, está provado que o problema é NP-completo.

2.2.1 Algoritmo Exato

Para tornar o algoritmo exato mais rápido, os grafos são representados como uma cadeia de bits (*bitmask*) aliado a um arranjo de uma estrutura criada para conter o peso de cada vértice assim como os seus vizinhos (representado por uma cadeia de bits também).

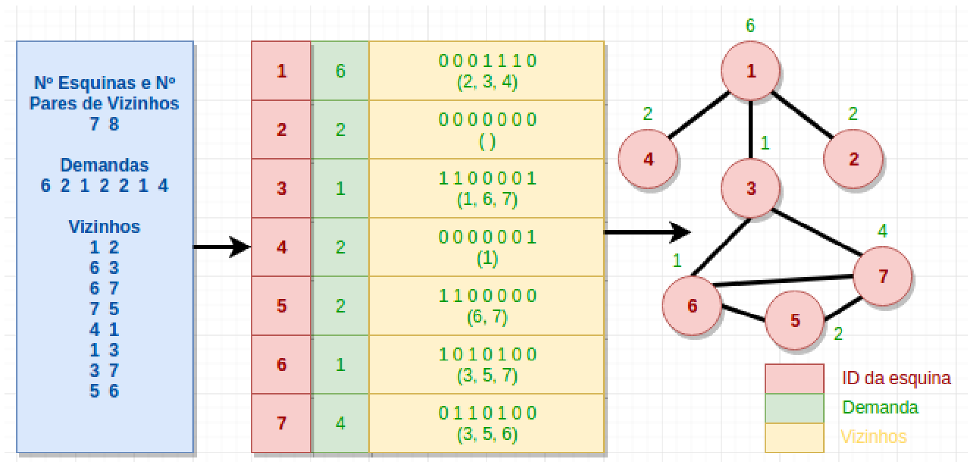

Figura 4 – *Estrutura Utilizada no Algoritmo Exato.*

Como cada um dos n vértices podem ou não estar na solução final, são 2^n possibilidades de conjuntos a serem testadas. A ideia do algoritmo consiste basicamente em percorrer cada subconjunto achando todos os conjuntos independentes possíveis. Uma vez que pede-se a maior demanda possível, comparações são feitas entre os subconjuntos válidos para obter, ao final, o conjunto independente de peso máximo. No caso de múltiplas soluções ótimas, a que possuir o maior número de esquinas é selecionada.

2.2.2 Heurística

Na heurística, a estrutura utilizada para o algoritmo exato sofreu modificações. Como não há como usar *bitmask* uma vez que o número de vértices pode ser 10^5 e cada bit deve representar um vértice, os vizinhos são representados por uma lista de adjacências. A *struct esquina* teve o campo *vizinhas* removido e o campo *id* acrescentado. Um vetor de N posições dessa *struct* serve para localizar e armazenar os pesos de cada esquina. Além disso, foi criado um vetor de tamanho N para servir como uma validação dos vértices aptos a compor o conjunto independente. Se na posição x dele, por exemplo, o valor for 0, significa que o vértice x pode entrar no conjunto. Se for 1 é inválido.

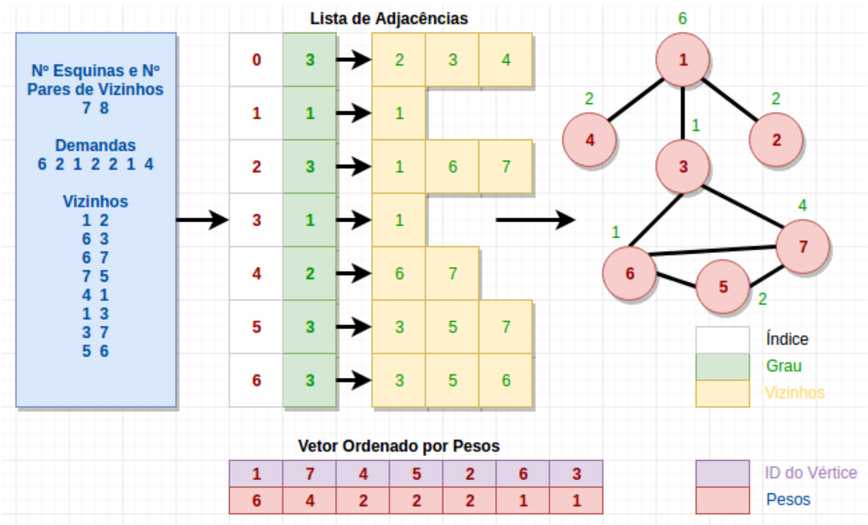

Figura 5 – *Estrutura Utilizada na Heurística.*

Uma estratégia gulosa foi adotada com o intuito de conseguir uma aproximação. Ela consiste, basicamente, em ordenar os vértices de maneira decrescente utilizando como parâmetro o peso. Uma vez ordenados a solução escolhe a esquina v de maior peso para compor o conjunto solução desde que passe na validação citada anteriormente e elimina a possibilidade de inserção de seus vizinhos que são adjacentes.

2.3 Organização do Código

A organização do código, em termos de arquivos, ficou do seguinte modo:

- **exato.h e exato.c**: fornece uma *struct esquina* que possui o peso e um inteiro representando os vizinhos. Além disso, possui as operações de inicialização e a função que acha o conjunto independente de peso máximo.
- **heuristica.h e heuristica.c**: fornece uma *struct esquina* que possui o identificador do vértice e seu peso e uma *struct* que permite a construção da lista de adjacências de cada vértice. Além disso, possui as funções responsáveis pelo *Heap Sort*, funções de manipulação da lista de adjacência e a função com uma solução aproximada do problema.
- **mainE.c**: contem a solução para o problema utilizando o algoritmo exato.
- **mainH.c**: contem a aproximação do problema.

3 Análise de Complexidade

Nesta seção serão apresentadas as análises de complexidade de tempo e espaço dos algoritmos implementados.

3.1 Complexidade Temporal

3.1.1 Algoritmo Exato

Tanto a alocação quanto a liberação do vetor do tipo *esquina* de tamanho n ocorre em $O(n)$. Então, inicializar o valor correspondente aos vizinhos de cada vértice é $O(n)$ também. Já a função responsável pela solução exata testa todos os 2^n subconjuntos. Para cada um deles é necessário verificar se cada vértice pode se juntar ao conjunto que está sendo construído e, se for, a soma de seus pesos é feita percorrendo os n bits (cada um respresentando um vértice). Logo, no pior caso, a complexidade temporal é $O(n^2 2^n)$ ou $O^*(2^n)$. A consulta de um vértice no conjunto é feita em $O(1)$ já que são operações lógicas com bits.

3.1.2 Heurística

Tanto a alocação quanto a liberação do vetor do tipo *esquina* de tamanho n ocorre em $O(n)$. Para inicializar o valor correspondente aos graus de cada vértice é $O(n)$ também.

O algoritmo de ordenação escolhido para essa solução é o *Heap Sort*. A complexidade do *constroiHeap* é $O(\log n)$. A complexidade de todo o processo contando as chamadas é $O(n \log n)$.

Já a função responsável pela aproximação testa cada um dos n vértices apenas verificando se é possível ou não incluir no conjunto final sem que ele deixe de ser independente. Logo, a complexidade é $O(n)$. Se um vértice está apto a se juntar a solução aproximada um vetor de tamanho N é percorrido para adicionar seus vizinhos como vértices que não devem ser incluídos. Então, no pior caso, o vértice de maior peso (primeiro a ser conferido) se conecta a todos os outros vértices e, portanto, só ele irá compor o resultado e, nesse caso, o vetor restrição será percorrido em $O(n)$.

Com relação a lista de adjacência que representa um grafo $G(V, A)$, o loop externo é feito $O(|V|)$ vezes. Mesmo que não existam arestas para cada iteração do loop externo, um número constante de operações é executado ($O(1)$). O loop interno é executado uma vez para cada aresta, ou seja, $O(grau(V))$ vezes. Somando tudo, a complexidade é $O(|V| + |A|)$.

3.2 Complexidade Espacial

3.2.1 Algoritmo Exato

O vetor do tipo *esquina* que possui os pesos e um inteiro representando os vizinhos de cada esquina tem tamanho n. Logo, possui complexidade de espaço $O(n)$.

3.2.2 Heurística

O vetor do tipo *esquina* que possui os pesos e um inteiro representando os vizinhos de cada esquina tem tamanho n. Logo, possui complexidade de espaço $O(n)$. Já o *Heap Sort* possui, no pior caso, complexidade de espaço $O(1)$.

Como o valor máximo da entrada para a heurística é definido como 10^5 é necessário um algoritmo mais eficiente. Apesar do *Quicksort* ser mais rápido na maioria dos casos, sua complexidade temporal, na pior situação é $O(n^2)$ enquanto a espacial é $O(\log n)$. Já o *Heap Sort* é sempre $O(n \log n)$ na temporal e $O(1)$ na espacial.

A complexidade é dada por $O(|V| + |A|)$. No pior caso, o grafo é denso e tem-se $N(N-1)/2$ arestas. Portanto, a complexidade espacial da lista de adjacência é $O(n^2)$.

4 Análise Experimental

Para analisar o comportamento dos algoritmos desenvolvidos, foi implementado um gerador de testes para verificar o que ocorre com o crescimento do número de esquinas (n) com o número de vizinhos variando.

A Figura 7 ilustra o comportamento assintótico analisado experimentalmente para o crescimento do número de vértices (n) no intervalo $[3, 30]$. Como esperado, o comportamento do algoritmo exato confere com a complexidade encontrada anteriormente ($O(2^n n^2)$) enquanto o algoritmo aproximado tem um tempo polinomial.

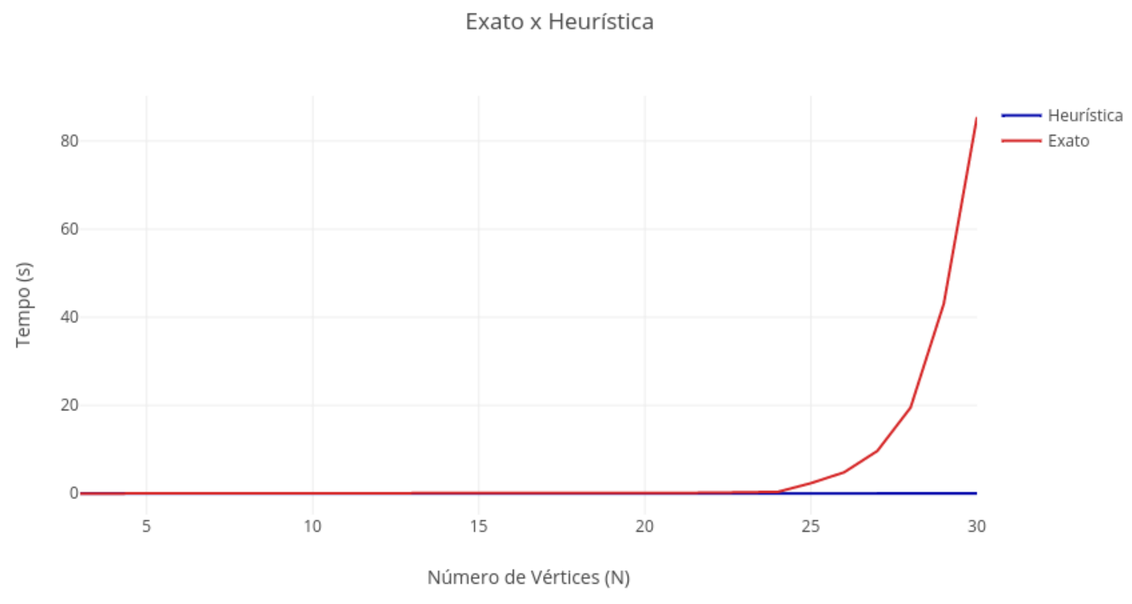

Figura 6 – Análise experimental do algoritmo de acordo com o número de esquinas ou vértices (n) crescente.

Com a finalidade de verificar a aproximação feita um outro gráfico foi criado listando para um mesmo número de casos de teste a soma do conjunto independente encontrada tanto pelo algoritmo exato quanto pela heurística.

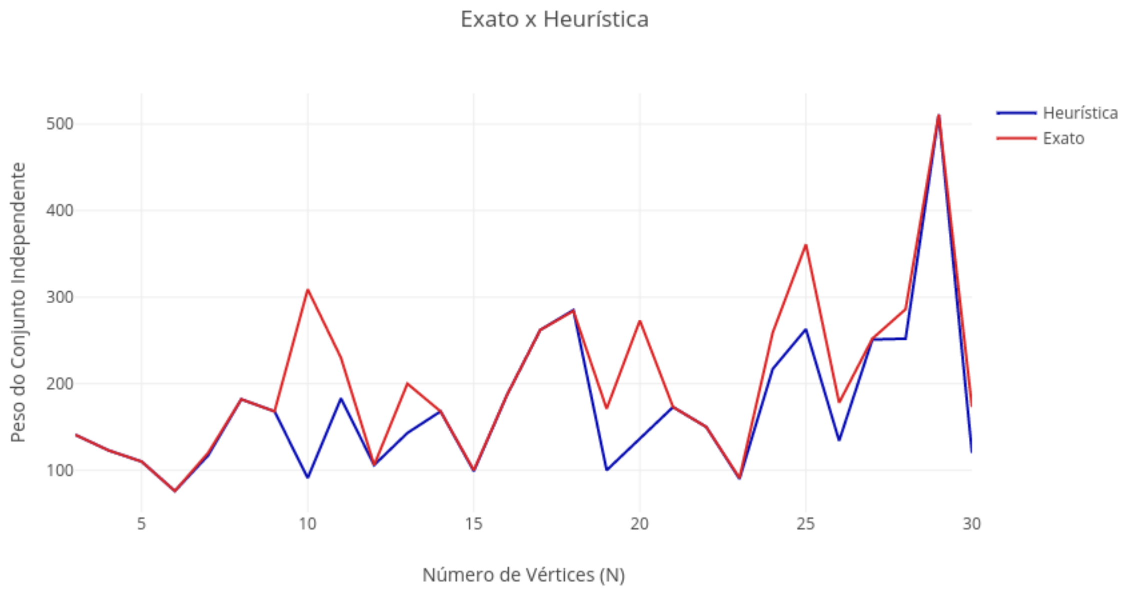

Figura 7 – Soma dos pesos do conjunto independente encontrado pelo algoritmo exato e pela heurística.

5 Conclusão

Esse trabalho apresentou a solução exata, porém exponencial, e a solução aproximada, porém polinomial, de encontrar um conjunto independente com peso máximo. Além disso, provou sua NP-completude mostrando que ele faz parte de NP e mostrando um outro problema NP-completo, transformando-o no problema em questão polinomialmente.

A estratégia utilizada na heurística apresenta uma complexidade temporal interessante e resultados satisfatórios. Ela fornece uma soluçao mais próxima da exata para grafos densos com pesos desequilibrados. Porém, pode ocorrer de um vértice com peso alto ter um grau elevado, enquanto o conjunto independente com maior demanda seria um que utilizasse vértices de peso menor mas com poucos vizinhos.

6 Referências Bibliográficas

Cormen, Thomas H. Introduction to Algorithms. Cambridge, Mass: MIT Press, 2001.

ZIVIANI, Nivio. Projeto de Algoritmos com implementações em PASCAL e C, 3 ed. Cengage Learning, 2011.

Proving a Problem is NP-Complete. Computer Science Department Cornell University. Acesso em: 16 nov 2017. Disponível em: <http://www.cs.cornell.edu/courses/cs482/2005su/handouts/>.

Graph Theory. Keijo Ruohonen. Acesso em: 16 nov 2017. Disponível em: <http://math.tut.fi/ ruohonen/GT$_E$*nglish.pdf* >