Universidade Federal de Minas Gerais

Bacharel em Sistemas de Informação
Algoritmos e Estruturas de Dados 3

Trabalho Prático 2
Novembro 2017

Gabriel Silva Bastos
Matrícula: 2016058204

1 Introdução

A startup de Nubby foi contratada pela Uaibucks para calcular a melhor forma de implantar filiais em determinada cidade. A Uaibucks deseja implantar suas filiais em esquinas, mas com a restrição de não implantar filiais em esquinas vizinhas para evitar auto-concorrência. Além disso, a Uaibucks possui dados da demanda esperada de cada esquina candidata, e deseja maximizar a demanda total no plano de implantação.

2 Visão Geral da Solução

O plano de implantação das filiais pode ser modelado como um problema de grafos. No grafo, cada esquina é modelada como um vértice, e cada vértice possui como peso a demanda associada. Há uma aresta entre dois vértices caso as esquinas correspondentes sejam vizinhas. O grafo não é direcionado, pois a relação de vizinhança é comutativa. As arestas não possuem peso. Portanto, o grafo é simples.[1]

No grafo, como cada vértice representa uma esquina, desejamos obter um conjunto de vértices não vizinhos que maximize a demanda total. Para tal, exploramos o conceito de conjunto máximo independente.[2] Pela definição, um conjunto máximo independente é um conjunto que não é subconjunto de nenhum outro conjunto independente. Em outras palavras, não há vértice fora do conjunto que, ao ser adicionado, mantenha a propriedade de Independência[3] do conjunto. Considerando que a demanda mínima para uma esquina é zero, um conjunto máximo independente possui demanda total sempre maior ou igual à de seus subconjuntos.

Portanto, para maximizar a demanda, selecionamos dentre todos os conjuntos máximos independentes o conjunto que gera a maior demanda total.

2.1 NP-Completo

O problema do conjunto máximo independente pode ser definido como um problema de decisão:

$$\text{O grafo } G \text{ possui algum conjunto máximo independente de tamanho } \geq k?$$

Partindo do fato que o problema da clique máxima é NP-Completo[4], podemos provar que o problema do conjunto máximo independente é NP-Completo.

$$\text{Se } S \text{ é um conjunto máximo independente em um grafo } G\text{, então } S \text{ é uma clique máxima no grafo } G'.^5 \tag{1}$$

Tal relação permite uma transformação polinomial de uma clique máxima para um conjunto máximo independente, pois o grafo complemento pode ser obtido em tempo polinomial.

$$\text{Conjunto máximo independente} \propto \text{Clique máxima}$$

Portanto, o conjunto máximo independente é pelo menos tão difícil quanto a clique máxima.
Sendo a clique máxima NP-Completo, o conjunto máximo independente é NP-Completo.

2.2 Entrada

A entrada consiste de um número n de vértices, e um número m de arestas. Em seguida, os pesos de cada vértice e, logo após, os pares que formam cada aresta.

[1] `http://mathworld.wolfram.com/SimpleGraph.html`
[2] `https://en.wikipedia.org/wiki/Maximal_independent_set`
[3] `https://en.wikipedia.org/wiki/Independent_set_(graph_theory)`
[4] `https://en.wikipedia.org/wiki/Clique_problem#NP-completeness`
[5] `https://en.wikipedia.org/wiki/Maximal_independent_set#Related_vertex_sets`

2.3 Algoritmo exato

Há um algoritmo bem conhecido para listar cliques máximas em um grafo: algoritmo de Bron-Kerbosch. [1] Tal algoritmo foi utilizado em conjunto com a propriedade (1) para listar os conjuntos máximos independentes, e destes selecionar o que gera a maior soma dos pesos dos vértices. A versão com pivoteamento descrita por Tomita *et al.* [3] foi implementada para melhor performance:

Algoritmo 1 Bron-Kerbosch com pivoteamento

procedure BRON-KERBOSCH(G)
    max $\leftarrow \varnothing$
    RECURSION($\varnothing$, $V(G)$, $\varnothing$) ▷ $V(G)$ is the vertex set of G.
    return max

procedure RECURSION(R, P, X)
    if $P \cup X = \varnothing$ **then** ▷ R is a maximal clique.
        if weight(R) $>$ weight(max) **then**
            max $\leftarrow R$
    $u \leftarrow$ choose one vertex in $P \cup X$ maximizing $|P \cap \Gamma(u)|$ ▷ $\Gamma(x)$ is the neighbor set of x.
    for each vertex v **in** $P \setminus \Gamma(u)$ **do**
        RECURSION($R \cup \{v\}$, $P \cap \Gamma(v)$, $X \cap \Gamma(v)$)
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$

Para representar os conjuntos de vértices, considerando que 30 foi a ordem máxima especificada para o grafo, um inteiro de largura fixa de 32 *bits* seria suficiente. Cada *bit* corresponde à presença do vértice no grupo. Tal estratégia torna rápidas as operações necessárias para o algortimo, como união, interseção e subtração de conjuntos.

$$0\ldots0\,00011010_2 \equiv \{1,3,4\}$$
$$0\ldots1\,00100101_2 \equiv \{0,2,5,8\}$$
$$0\ldots0\,11111111_2 \equiv \{0,1,2,3,4,5,6,7\}$$

Porém, após realizar a análise experimental, foi constatado que o programa executava em um centésimo de segundo para entradas de tamanho 30. Portanto, para permitir experimentos mais conclusivos, inteiros com largura de 64 *bits* foram utilizados, permitindo entradas com até 64 vértices.

Devido à necessidade recorrente de se obter o conjunto de vizinhos de um vértice no algoritmo, o grafo foi representado de forma que essa operação tenha complexidade $\mathcal{O}(1)$. Para cada vértice, ao invés de uma lista de adjacentes, é armazenado um único inteiro que representa o conjunto dos adjacentes. Tal estratégia requer uma implementação mais elaborada para ler a entrada, mas apresentou uma grande melhora na performance do algoritmo.

2.4 Heurística

Para a heurística, a entrada foi interpretada como uma construção incremental do grafo. Desta forma, antes da leitura das arestas, o grafo é considerado vazio e o conjunto independente possui todos os vértices. Cada aresta lida da entrada é adicionada ao grafo, e possivelmente resulta na remoção de um vértice do conjunto independente. Assim, sempre quando dois vértices no conjunto independente são conectados por uma aresta, um destes é removido do conjunto para manter a propriedade de independência. Ao final da leitura de todas as arestas, o conjunto é independente no grafo.

Este algoritmo é muito simples, mas oferece a vantagem de ser muito rápido por operar com custo baixo logo na leitura de cada aresta, que já é necessária devido ao formato da entrada.

Não há nenhuma garantia de que o conjunto independente gerado será próximo máximo, nem que será próximo do conjunto que gera a maior soma dos pesos. Outro detalhe a ser notado é que o algoritmo é sensível à ordem das arestas na entrada. Portanto, diferentes entradas descrevendo o mesmo grafo podem gerar saídas muito diferentes.

3 Análise de Complexidade

3.1 Algoritmo exato

3.1.1 Espacial

Para cada vértice, um inteiro representando o peso e um inteiro de 64 *bits* representando o conjunto de seus adjacentes são armazenados. Portanto, a complexidade espacial é $\Theta(|V| + |V|) = \Theta(|V|)$.

3.1.2 Temporal

De acordo com Tomita *et al.*[3], a complexidade do algoritmo de Bron-Kerbosch com a técnica de pivoteamento possui complexidade $\mathcal{O}(3^{|V|/3})$ no pior caso. Esta complexidade é ótima, pois segundo Moon e Moser[2], um grafo com n vértices possui no máximo $3^{n/3}$ cliques máximas.

3.2 Heurística

3.2.1 Espacial

Para cada vértice, são armazenados um inteiro representando o peso e um *byte* indicando a presença do vértice no conjunto independente. Portanto, a complexidade espacial é $\Theta(|V| + |V|) = \Theta(|V|)$.

3.2.2 Temporal

Devido à construção incremental do grafo, inicialmente este não possui arestas, então o conjunto máximo independente possui todos os vértices. Antes da leitura das arestas, é calculada a soma dos pesos de todos os vértices, com complexidade $\Theta(|V|)$.

Para cada aresta na entrada, uma aresta pode ser removida do conjunto independente. Tal remoção apresenta complexidade $\Theta(1)$, devido à representação do conjunto independente como um vetor de *bytes*. Portanto, a complexidade desta operação é $\Theta(|A|)$.

Complexidade final: $\Theta(|V| + |A|)$.

4 Análise Experimental

A análise experimental das implementações é mostrada pelas figuras. Para realizar os experimentos, diversas entradas foram construídas. Para medir o tempo de execução do código, foi utilizada a informação de uso de recursos fornecida pelo sistema operacional linux.[6] A razão inicial é verificar, de forma geral, o comportamento do algoritmo para casos genéricos.

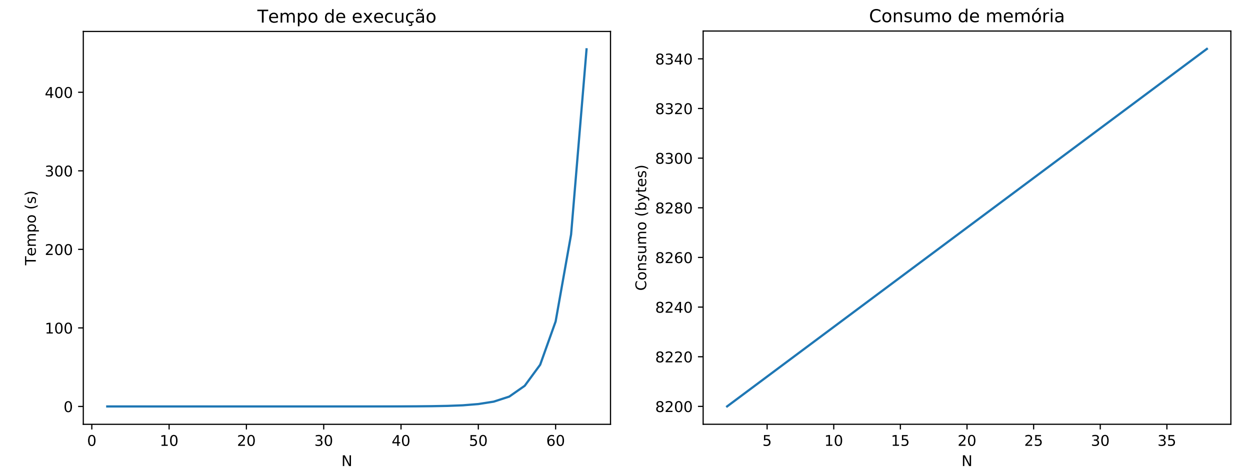

Figura 1: Ensaio do algoritmo exato.

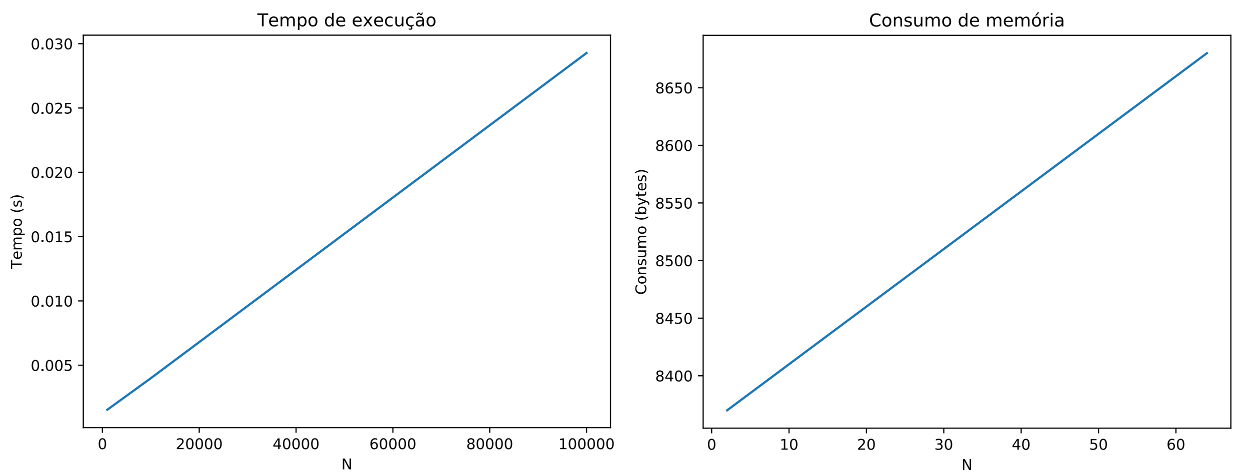

Figura 2: Ensaio da heurística.

É interessante observar na Figura 1 que o comportamento exponencial do algoritmo exato é notável apenas a partir das entradas de tamanho 50. Isso se deve às estratégias utilizadas para tornar rápidas as operações intrínsecas do algoritmo.

Também é notável que a heurística se mostra extremamente rápida, mesmo para entradas muito grandes.

[6] `https://en.wikipedia.org/wiki/Procfs`

4.1 Comparação das soluções

Para avaliar a qualidade da heurística, uma comparação dos resultados produzidos a partir das entradas *toys* fornecidas pelos monitores foi realizada.

	Demanda total		
Toy	Algoritmo exato	Heurística	Degradação
3	170	170	0 %
7	30	30	0 %
6	139	125	10 %
5	304	250	18 %
8	265	199	25 %
9	312	207	34 %
10	198	124	37 %
1	118	57	52 %
4	10	4	60 %
2	168	57	66 %

Tabela 1: Comparação entre a otimalidade das soluções

É notável que metade dos resultados apresentou degradação inferior a 30%. Tal resultado indica uma boa aproximação produzida pela heurística.

5 Conclusão

Grafos são uma forma elaborada de modelar vários problemas, e esta estrutura de dados apresenta toda uma classe de algoritmos. É importante saber como modelar os problemas em grafos, e quais algoritmos são necessários para extrair cada tipo de informação necessária para o problema.

Problemas NP-Completo estão presentes na nossa realidade, e precisamos de ferramentas eficientes para resolvê-los. Além da formalidade necessária para provar que um problema é NP-Completo, desenvolvemos diferentes formas de abordar o problema.

Algoritmos exatos produzem respostas ideais, e são úteis não só para obter tais respostas, mas também para fornecer uma base de comparação para outras soluções. Porém, estes algoritmos são viáveis apenas para entradas pequenas, devido à sua alta complexidade.

Algoritmos aproximados são interessantes quando existe a necessidade de um certo nível de correção da resposta, porém pode ser complexo o desenvolvimento do algortimo e da prova da sua razão de aproximação.

Heurísticas são especialmente interessantes quando precisamos de uma resposta para o problema, não necessariamente próxima da ótima, mas de forma muito rápida. É interessante como heurísticas simples podem apresentar bons resultados em complexidade muito inferior à do algoritmo exato.

Referências

[1] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[2] J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1):23–28, Mar 1965.

[3] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28 – 42, 2006. Computing and Combinatorics.