Algoritmo e Estrutura de Dados III

Trabalho Pratico 2

Isabela Meneguci de Souza Petrim

Novembro de 2017

1 Introdução

O problema do trabalho prático proposto apresenta uma cafeteria de nome fictício chamada UaiBucks. Essa cafeteria deseja instalar novas filiais de forma que não haja filiais vizinhas na mesma esquina e que seja escolhida a filial que possui a maior demanda de clientes. Esse problema pode ser modelado como um problema de decisão que se enquadra nas classes dos algoritmos Np-Completos. A motivação desse trabalho é extremamente aplicável a um problema real, como por exemplo empresas do ramo de Fast Food para instalar uma nova filial é necessário verificar se não existem outras lojas instaladas próximas para maximizar o lucro. Para a realização do trabalho prático eram necessários alguns conceitos principais. O mais importante era o conceito de grafos e conjuntos independentes.

A resolução do trabalho consistia em encontrar um conjunto independente de peso máximo. Em linhas gerais conjunto independente em teoria dos grafos é um subconjunto de vértices em que não existe nenhuma aresta entre qualquer par de elementos. Aplicando essa definição ao problema prático, consiste em encontrar um conjunto de filiais que não sejam vizinhas entre si e que possuam a maior demanda de clientes.

Para a resolução desse problema foi implementado um algoritmo exato que resolvia solução do problema em tempo exponencial e uma heurística que não era uma solução ótima, mas resolvia o problema em tempo polinomial. Para o algoritmo exato a solução é um algoritmo que utiliza a força bruta, basicamente todas as combinações possíveis de filiais são geradas, e para cada umas das combinações são testados se esse subconjunto é um conjunto independente. Após todos os subconjuntos terem sido testados, o maior deles e retornado como o conjunto independente máximo. Já a heurística toma uma decisão gulosa a cada passo, ou seja, ele toma decisões com base

nas informações disponíveis no momento, sem olhar as consequências que essas decisões terão no futuro. A ideia dessa solução é calcular uma primeira vez qual é a filial que possui o maior peso descartar todas as suas vizinhas. Para as filiais restantes ele seleciona qual é a que possui maior peso e retorna à solução.

2 Modelagem e Prova de Np-Completude

Nessa secção será realizada uma prova que o problema de encontrar um conjunto independente é classificado como Np-completo. Para isso é necessário mostrar como foi feita a modelagem do grafo. A modelagem do problema foi feita utilizando um grafo não direcionado, conectado e ponderado. As filiais representavam os vértices do grafo, as arestas do grafo foram as ligações entre duas filiais, ou seja, quando uma filial era vizinha de outra foi inserido uma aresta conectando esses dois vértices. O peso simbolizava a demanda de clientes para cada filial, sendo assim o grafo possui pesos nos vértices.

Para a confecção da prova de NP-Completude, foi utilizada a versão de decisão do problema. Ela possui o seguinte enunciado: Seja um grafo G(V,A), existe um sub-grafo deste que forme um conjunto independente o qual a soma de seus vértices é de no mínimo k? Essa prova passa por dois passos. Primeiro é necessário mostrar que existe um algoritmo polinomial que comprove que a resposta é um conjunto independente e é maior que um dado número k.

Algoritmo 1 Verifica

```
function VERIFICA A SOLUÇÃO (Grafo , Vértices, Arestas, CI, K )
    for each v1 ∈ CI do
        for each v2 ∈ CI do
            if aresta(v1, v2) & (v1 ≠ v2) then
                return false;
    S ← sum(CI)
    if S < k then
        return false;
    return true;
```

O segundo passo é comprovar a partir de um problema computacional conhecido, que é classificado como Np-Completo, e que é possível transforma-lo no Conjunto Independente que se esse problema é Np-Completa o Conjunto Independente também é Np-Completo. Esse problema é o problema da Click. Enunciado da Click: Dado um grafo (V,A) não-dirigido é um conjunto de vértices dois a dois adjacentes. Em outras palavras, um conjunto C de vértices

é uma Click se tiver a seguinte propriedade: para todo par v1,v2 de vértices distintos em um conjunto, existe uma aresta com pontas v e w. A prova de que a Click é um problema classificado como Np-Completo está enunciado em vários livros da computação. O primeiro a provar que a Np-Completude da Click foi Richard M. Karp em seu livro '"Reducibility among combinatorial problems."Complexity of computer computations. springer US, 1972. '

Sendo assim, é possível fazer uma transformação do Click para o conjunto independente e vice e versa: Dado um Grafo G(V,A) basta calcular seu grafo complementar G'(V,A) , ou seja, inserir arestas onde não existe e retirara arestas onde existe. Sendo assim se existe uma Click no grafo o restante seria o conjunto independente.

A baixo segue duas figuras para exemplificar, a primeira reprenta um grafo e um conjunto independente associado a ele. A segunda figura, representa o grafo complementar do primeiro e uma possível Click.

Figura 1: Conjunto independente

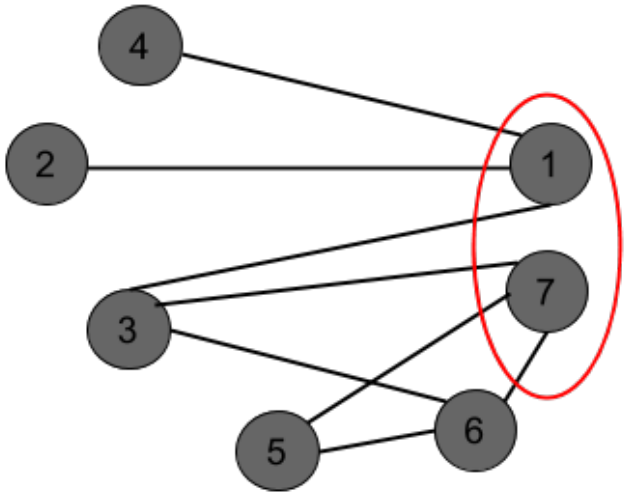
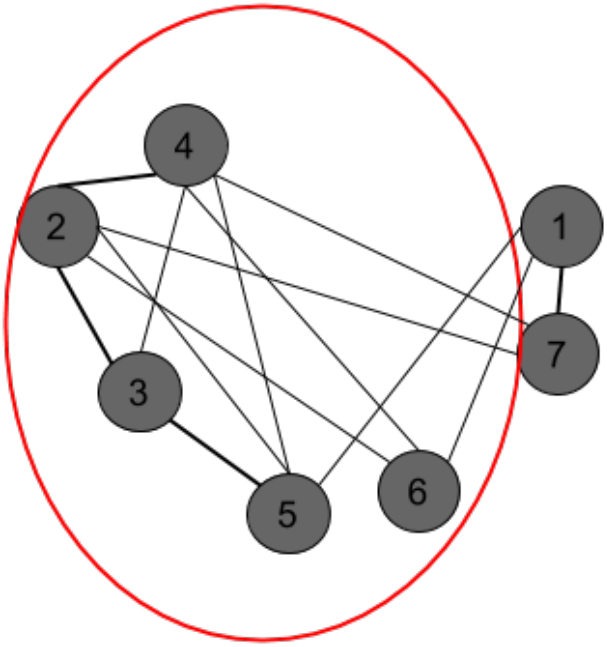


Figura 2: Click



3 Solução do Problema

Neste trabalho foram implementados dois algoritmos distintos para encontrar a solução para o problema. O primeiro algoritmo, o exato utiliza simplesmente força bruta para encontrar a solução já a heurística toma uma decisão gulosa a cada passo. O primeiro algoritmo o exato possui uma função chamada de 'comb' essa função calcula através da manipulação de bits todas as combinações possíveis para um vetor de entrada. Nela existe um loop inicial que controla o tamanho dos subconjuntos gerados, começando a partir de 2 até o número de vértices. E guarda cada uma das combinações no vetor '$\text{vet}_{c}ombina'$.

Em seguida para cada uma das combinações geradas a função chama outra função chamada 'conjIndependente' que percorre os vértices do subconjunto passado salva os vizinhos de cada um dos vértices em um vetor auxiliar e verifica se possui algum vizinho no vetor que foi passado. Ou seja, verifica se os vertices são vizinhos.

Se o conjunto for independente, ou seja, se a função 'conjIndependente' retorna 0, é chamado outra função que calcula o peso máximo do vetor passado, e compara se aquele é o maior peso já encontrado. A função 'comb'

Algoritmo 2 Conjunto Independente

```
function VERIFICA SE UM SUBCONJUNTO É UM CONJUNTO INDEPENDENTE (Grafo , Quandidade de Vertices , Vetor Entrada, Tamanho do Vetor )
    for Percorre todos os vizinhos do vertice do
        vet_vizinhos ← vizinhodovertice −

        for ( Percorre o subconjunto de entrada ) do
            for ( Percorre o vetor de vizinhos ) do
                if Elemneto do vetor entrada = vetor vizinhos then
                    return DEPENDENTE

                return INDEPENDENTE
```

retorna então o peso máximo encontado.

Algoritmo 3 Calcula Peso

```
function RETORNA O PESO MÁXIMO DO VETOR PASSADP (Grafo , Vetor, Tamanho do Vetor )
    for Percorre o vetor de Entrada do
        vertice = grafo para o indice de cada elemento do vetor
        Peso mais peso de cada vertice
        return peso;
=0
```

Para imprimir o resultado pedido na tela existe a função 'imprime' que possui a mesma estrutura da função 'comb'. Ela realiza o mesmo procedimento, mas quando encontra um peso igual ao peso máximo retornado pela função 'comb' imprime o resultado na tela. Para casos onde existe mais de umas combinações que retornem o peso máximo será impresso a que possuem maior quantidade de filiais, por isso a nesse caso os subconjuntos são gerados do maior possível para o menor.

A heurística proposta possui também uma função possui dois vetores de controle iniciais $\text{vetor}_r estante e \text{vetor}_i nicial, aprincpioosdoisvetorespossuemomesmovalorqueseri$

Na primeira iteração o vetor que será passado para a função será o vetor inicial de entrada. Em seguida a função pega o vértice que tiver maior peso e atribui no vetor, $\text{vetor}_f ilial_e xcluidatodososseusvizinhos. Af unoatualizaovetor_r estantescomosnd$

Essa heurística não foi a melhor que poderia ser aplicada, visto que seria necessário também comparar os vértices que possuem menor grau e escolher filiais a partir de uma mediana entre a quantidade de vértices e o número de ligações que esse vértice faz. Já que se uma filial que possui maior peso, por exemplo 100, e tiver arestas com outras três vizinhas com pesos 70, 80, 90 por exemplo claramente essa não foi a melhor escolha.

4 Análise de Complexidade

Nessa seção será apresentada a análise teórica de custo para os dois algoritmos o exato e a heurística.

A complexidade temporal do algoritmo que calcula o peso das arestas O(V), já que este possui somente um loop que é realizado até o tamanho do vetor de entrada. Já a complexidade do algoritmo que calcula o peso, por possuírem três loops em cadeia sua complexidade O(n). A complexidade da função comb, possui 5 loops sequenciais que são executados até a quantidade de vértices existentes e dentro dessa função ele chama as outras duas funções já analisadas a calcula peso e a conjunto independente. Sendo assim sua complexidade seria exponencial e calculada em $O(n^7)$.

Já a heurística possui a mesma complexidade para a função calcula peso, mas a função combinação possui somente dois loops encadeados possuindo complexidade O(n).

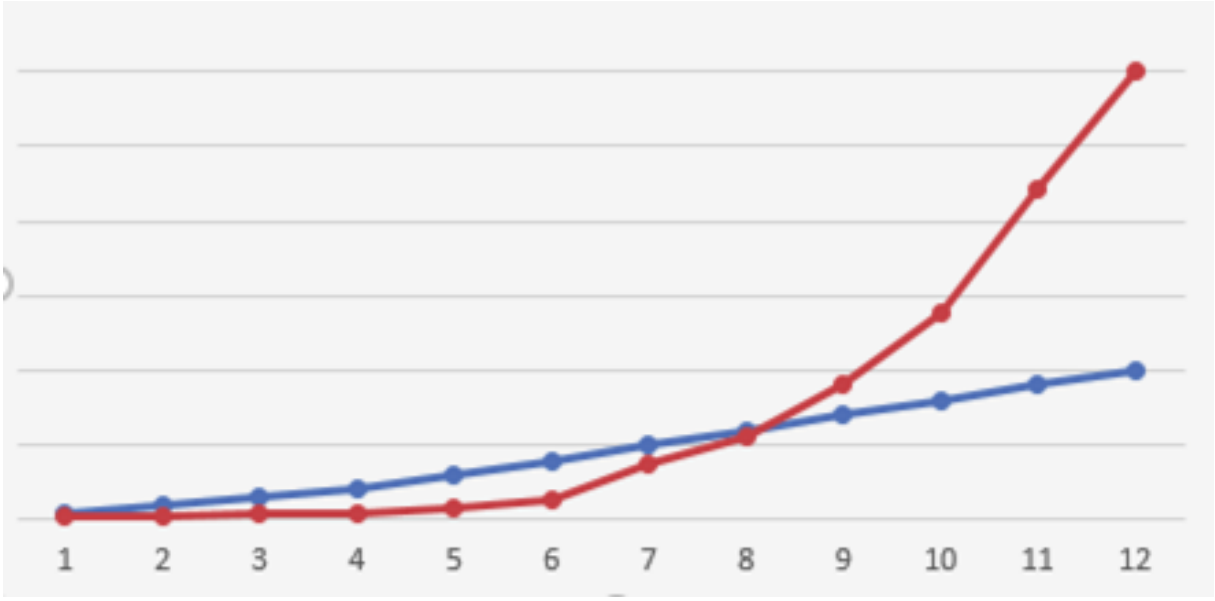
A complexidade espacial da heurística é maior que a do exato devido a necessidade de existir 4 vetores que são declarados de forma dinâmica, enquanto o exato só necessita de um vetor de forma dinâmica. Assim, a complexidade do extado seria O(vértices) e da heurística $O(vertices^4)$.

A complexidade temporal e espacial da main e do Grafo em si não foram inseridas por serem a mesma para os dois algoritmos não impactando no resultado final da análise de complexidade

5 Análise Experimental

Na seção a seguir é feita uma análise experimental . Será inserido os dados observados em um gráfico para melhor visualização. A análise experimental não pode ser realizada de forma mais precisa já que existe um erro de lógica para os dois algoritmos impedindo que eles sejam executados para valores maiores.

Figura 3: Gráfico



6 Conclusão

Através do trabalho prático foi possível entender a teoria dos grafos e np-completude aplicado a um problema que está muito próximo a problemas reais. Além disso, foi possível compreender que para se aplicar uma heurística a um algoritmo ele deve estar bem perto da solução esperado porque mesmo que este seja bem mais rápido do que o algoritmo exato se ele não trouxer uma boa resposta talvez o algoritmo exato seja uma melhor opção.