# Algoritmos Avançados - Relatório do 1º Trabalho Prático

Gabriel Silva, 100451

## I. Introdução

Neste primeiro trabalho da cadeira de Algoritmos Avançados foi-nos proposto implementar a solução de um problema através de pequisa exaustiva ou de *dynamic programming* seguido de uma análise relacionada com o esforço computacional efetuado pelo algoritmo.

Da lista de problemas apresentados escolhi o problema de determinar a cobertura de vértices de cardinalidade mínima de um dado grafo não orientado. O algoritmo a desenvolver para este problema devia ser um algoritmo de pesquisa exaustiva onde consideraria todas as possibilidades.

## II. Problema

A cobertura de vértices de um grafo é um conjunto de vértices de maneira a que todas as arestas do grafo têm pelo menos um vértice a si associado, todos os vértices do grafo têm de estar ligados a pelo menos uma aresta. A cobertura de vértices de cardinalidade mínima é, então, o conjunto mínimo de vértices necessário de modo a que estes toquem em todas as arestas. Na figura 1 podemos ver exemplos de coberturas de vértices de um grafo, e na figura 2 podemos ver a cobertura de cardinalidade mínima deste mesmo grafos.

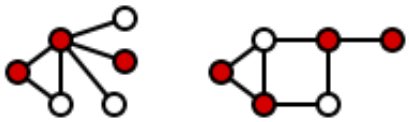

Fig. 1 - Exemplo de uma cobertura possível

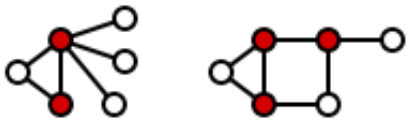

Fig. 2 - Exemplo de uma cobertura de cardinalidade mínima

Este é um problema conhecido como sendo *NP Complete*, isto é, não existe nenhuma solução em tempo polinomial para resolver este problema. Apesar disto existem algoritmos *greedy* que se conseguem aproximar-se de tempo polinomial para resolver este problema.

## III. Implementação

Todo o desenvolvimento do algoritmo para resolver este problema foi implementado com recurso à linguagem de programação *Python*. Os resultados para os tempos de execução foram obtidos num computador com um processador i5 8250u e 8GB de ram.

### A. Algoritmo de pesquisa exaustiva

O algoritmo de pesquisa exaustiva vai percorrer todas as possíveis soluções e no final vai dar a que teve menor tamanho.
O algoritmo desenvolvido funciona da seguinte maneira:

- Inicializar todas as possíveis soluções para um dado grafo, mesmo que não válidas
- Enquanto houver soluções, verificar a sua validade
- Para cada solução válida, contar o número de vértices necessário para cobrir o gráfico
- Após percorrer todas as soluções, devolver o menor número encontrado no passo anterior

Como exemplo, se tivermos um grafo com 2 vértices ligados, é criada uma lista de todas as possíveis soluções. Neste caso (0,0) (0,1) (1,0) (1,1) onde 0 representa que o vértice não está selecionado e 1 representa que o vértice está selecionado. Seguido disto é necessário verificar no grafo se estas soluções são válidas, por exemplo, a primeira solução não seria válida visto que não tenho nenhum vértice, logo, existem arestas que não são incidentes em nenhum vértice. Após estas verificações para todas as soluções o algoritmo devolve o número de "1" na solução encontrada, neste exemplo o algoritmo iria devolver 1 que vinha da solução (0,1) ou (1,0).

Visto que estamos a percorrer todo o espaço de possíveis soluções, este algoritmo garante sempre devolver o menor número de vértices necessários.

## IV. Respostas

Respostas às questões feitas no enunciado.

### A. Complexidade do Algoritmo

Este algoritmo de pesquisa exaustiva tem uma complexidade temporal exponencial. Na tabela I podemos ver os tempos (em minutos) obtidos para cada grafo

| Nº Vértices | Tempo de Execução | Crescimento |
|---|---|---|
| 4 | 0.00000387 | |
| 6 | 0.00000344 | 0.888888889 |
| 8 | 0.00001233 | 3.584302326 |
| 10 | 0.00005354 | 4.342254663 |
| 11 | 0.00011403 | 2.129809488 |
| 12 | 0.00018866 | 1.654476892 |
| 13 | 0.00037996 | 2.013993427 |
| 14 | 0.00068606 | 1.805611117 |

TABLE I
Tabela de tempos de execução e o seu respetivo crescimento para grafos de diferentes tamanhos

| Nº Vértices | Operações realizadas |
|---|---|
| 4 | 186 |
| 6 | 909 |
| 8 | 3998 |
| 10 | 17393 |
| 11 | 36662 |
| 12 | 75573 |
| 13 | 157219 |
| 14 | 328594 |

TABLE II
Tabela de número de operações para grafos de diferentes tamanhos

que foi testado. Na figura 3 podemos mais facilmente observar a relação entre o número de vértices do grafo e o seu tempo de execução.

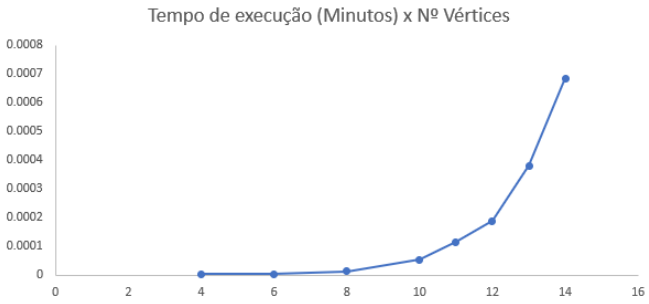


Fig. 3 - Crescimento do tempo de execução com o número de vértices do grafo

Tirando o outlier do grafo de 4 vértices, os tempos de execução vão sempre, aproximadamente, duplicando com a adição de um vértice pelo que é possível concluir que a complexidade temporal deste algoritmo é $O(2^n)$, ou seja tempo exponencial.

### *B. Testes Realizados*

#### *B.1 Número de Operações básicas*

Para contar o número de operações básicas foi criado um contador global, este contador conta as seguintes operações:

- Comparações - cada vez que uma comparação é feita é adicionado 1 ao contador
- Ciclos - cada passo do ciclo adiciona 1 ao contador

Ficando assim com o número de operações mostrado na tabela II. Podemos também ver a relação entre o número de vértices de um grafo com o número de operações na figura 4.

#### *B.2 Tempo de execução*

Os tempos de execução foram medidos para grafos com 4, 6, 8, 10, 11, 12, 13 e 14 vértices. Estes tempos já foram apresentados na tabela I.
Os tempos foram medidos com base em quando o algoritmo começava a correr e quando acabava. Como é possível observar o tempo de execução duplica com a adição de cada vértice.

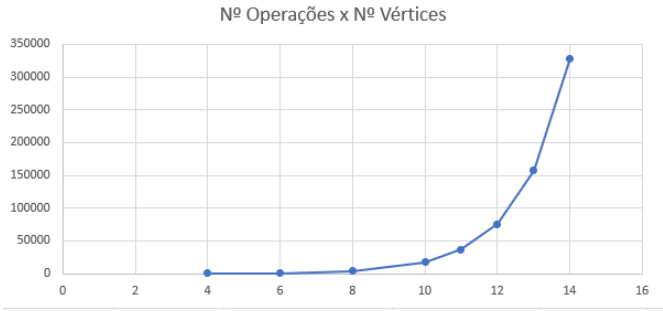


Fig. 4 - Crescimento do número de operações com o número de vértices do grafo

| Nº Vértices | Cardinalidade Mínima |
|---|---|
| 4 | 2 |
| 6 | 3 |
| 8 | 5 |
| 10 | 7 |
| 11 | 7 |
| 12 | 8 |
| 13 | 9 |
| 14 | 10 |

TABLE III
Cobertura de cardinalidade mínima encontrada para os grafos testados

#### *B.3 Número de soluções / configurações testadas*

Para cada um dos grafos com 4, 6, 8, 10, 11, 12, 13 e 14 vértices foi registada a cobertura de cardinalidade mínima encontrada pelo algoritmo implementado. Esta pode ser consultada na tabela III.
Para além disto, para cada grafo o algoritmo testa $2^n$ soluções onde n é o número de vértices que este tem, ou seja, para um grafo com 4 vértices o algoritmo vai testar $2^4 = 16$ soluções. Mesmo que o algoritmo encontre a ótima logo no ínicio este vai acabar por ter de testar todas as soluções para se certificar que é mesmo a ótima. Podemos encontrar o número de soluções testadas para cada grafo na tabela IV.

| Nº Vértices | Nº Soluções testadas |
|---|---|
| 4 | 16 |
| 6 | 64 |
| 8 | 256 |
| 10 | 1024 |
| 11 | 2048 |
| 12 | 4096 |
| 13 | 8192 |
| 14 | 16384 |

TABLE IV
NÚMERO DE SOLUÇÕES TESTADAS PELO ALGORITMO PARA CADA UM DOS GRAFOS USADOS

### C. Comparação dos Resultados Obtidos

Os resultados obtidos para a complexidade temporal do algoritmo não são surpreendentes visto que tanto o número de soluções a testar, que cresce a um ratio de $2^n$ onde n é o número de vértices do grafo, o tempo de execução do algoritmo, que duplica com cada vértice que é adicionado, e ainda aumentamos o número de operações efetuadas drasticamente, faz bastante sentido este algoritmo ter um tempo de execução de ordem $O(2^n)$.

Comparando o gráfico do tempo de execução (Fig. 3) com o gráfico de operações básicas realizadas (Fig. 4) é possível observar que estes são muito semelhantes, quase sobrepondo-se.

### D. Tempo de execução para instâncias de maior dimensão

Para calcular os tempos de execução para instâncias de maior dimensão, sabendo que este cresce exponencialmente, foi implementada, com ajuda da biblioteca scipy, uma função que faz fit de uma curva aos dados que temos. Depois de obter a curva podemos calcular estimativas para os tempos de execução que queremos.

Na figura 5 podemos observar os tempos previstos para grafos com 30, 32, 34, 36, 38 e 40 vértices. Sendo que para o grafo com 40 vértices o tempo de execução previsto é de 6169 minutos ou de aproximadamente 103 horas.
Para grafos com mais vértices é apresentada a tabela V, para mais fácil compreender a grandeza, onde podemos ver tempos de execução previstos para grafos com 35 a 60 vértices.

## V. CONCLUSÃO

A partir do relatório apresentado é possível concluir que o algoritmo de pesquisa exaustiva apresentado, apesar de devolver sempre o melhor resultado, para grafos com um elevado número de vértices é simplesmente impossível utilizar. Utilizando este tipo de algoritmos para um grafo com 50 vértices seriam necessárias aproximadamente 48496 horas que é cerca de 66 meses, o que não é aceitável.

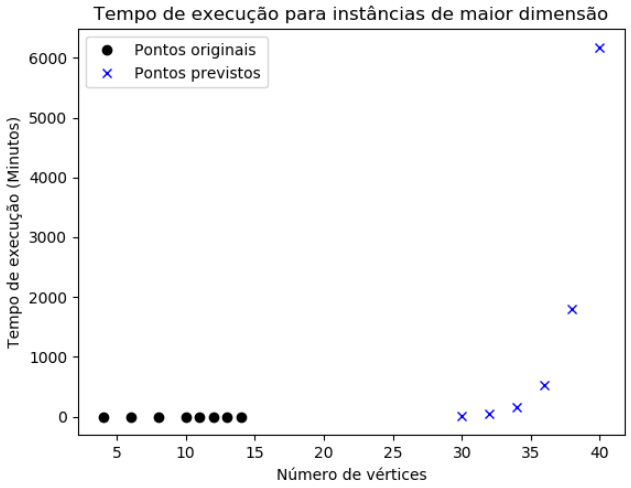


Fig. 5 - Tempo de execução previstos para grafos de maior dimensão

| Nº Vértices | Tempo de Execução(Horas) |
|---|---|
| 35 | 4.7 |
| 40 | 102.8 |
| 45 | 2233.1 |
| 50 | 48496.2 |
| 55 | 1053200 |
| 60 | 22872500 |

TABLE V
TEMPO DE EXECUÇÃO PREVISTO PARA GRAFOS DE GRANDE DIMENSÃO

Sendo que a utilização de algoritmos de pesquisa exaustiva não são viáveis de utilizar neste tipo de problemas, é necessário encontrar alternativas, nomeadamente o algoritmo *greedy* que foi brevemente mencionado neste relatório.
Apesar destes não garantirem sempre a melhor solução os seus tempos de execução aproximam-se a tempos polinomiais o que torna correr problemas para encontrar a cobertura de cardinalidade mínima de grafos com elevada dimensão de vértices uma tarefa mais exequível.