

## Maximum Weight Cut

Hugo Veríssimo - 124348 - hugoverissimo@ua.pt

*Abstract* – abstrato em pt bla bla ns q in English  
in English in English in English in English in En-  
glish in English in English in English in English in  
English in English in English in English

*Resumo* – Este trabalho apresenta a implementação e comparação de dois métodos para resolver o problema *Max Weight Cut*: um algoritmo exaustivo e uma heurística gulosa. O problema do *Max Weight Cut* consiste em dividir um grafo em dois conjuntos complementares, de forma a que o peso das arestas cortadas seja maximizado.

## I. INTRODUÇÃO

Atualmente, os problemas em grafos são amplamente estudados, pelo facto de terem a capacidade de modelar diversas situações reais, desde as mais palpáveis, como redes de computadores (problema *Minimum Spanning Tree*) até às mais abstratas, como física estatística (problema *Maximum Weight Cut*) [1].

Este relatório visa explorar o problema *Maximum Weight Cut*, conhecido em português por Corte de Peso Máximo, que consiste na divisão de um grafo não direcionado,  $G(V, E)$ , onde  $|V| = n$  vértices e  $|E| = m$  arestas de peso  $w_{i,j} \geq 0 \ \forall (i, j) \in E$ , em dois subconjuntos complementares,  $S$  e  $T$ , de forma a maximizar a soma dos pesos das arestas que ligam os dois conjuntos [2], isto é

$$\max \sum_{i \in S, j \in T} w_{i,j}$$

$$\begin{cases} S \cup T = V \\ S \cap T = \emptyset \end{cases}$$

Apesar do problema oposto, conhecido como *Minimum Weight Cut*, ter um algoritmo de resolução em tempo polinomial, em certas condições, o problema *Maximum Weight Cut* não o possui, sendo um problema *NP-Hard*. Isto implica que à medida que o tamanho do grafo aumenta, encontrar soluções exatas para este problema tornam-se computacionalmente caras [3].

Ao longo deste relatório, serão abordadas duas estratégias para a resolução do problema: uma busca exaustiva e uma heurística gulosa.

## II. METODOLOGIA DA ANÁLISE

Com o intuito de analisar o problema em destaque, foi utilizada a linguagem de programação *Python*, conhecida pela sua simplicidade e pela vasta variedade de

bibliotecas, tais como *networkx*, *numpy* e *itertools*, que facilitaram a implementação das estratégias propostas.

A análise desenvolvida pode ser dividida em 2 ficheiros principais, sem desmerecer o uso de ficheiros auxiliares, sendo os primeiros:

```
$ python3 graphs.py
$ python3 benchmarks.py
```

O ficheiro *graphs.py* teve como propósito gerar vários grafos aleatórios, tendo em conta a semente 124348, com diferentes número de vértices, número de arestas e peso de arestas, de forma a avaliar o comportamento das estratégias aplicadas em diferentes cenários.

O ficheiro *benchmarks.py* foi o responsável por executar as estratégias de resolução do problema, nomeadamente a busca exaustiva e a heurística gulosa, para vários grafos gerados, guardando os resultados obtidos, tais como tempo de execução, número de operações básicas e precisão do resultado da heurística gulosa, para posterior análise.

### III. ALGORITMO DE PESQUISA EXAUSTIVA

Tendo em conta o algoritmo de pesquisa exaustiva, este visa gerar todas as combinações possíveis de subconjuntos de  $V$  e, para cada subconjunto, calcular o peso do corte e comparar com o melhor corte encontrado até ao momento, ou seja, o algoritmo tem duas fases: a geração de todos os subconjuntos possíveis e a avaliação de cada subconjunto. Esta estratégia garante a obtenção da solução ótima, no entanto, o seu custo computacional é exponencial, sendo impraticável para grafos de grande dimensão. Este algoritmo pode ser representado em pseudocódigo da seguinte maneira:

Pode-se verificar que as duas fases deste algoritmo apresentam complexidades  $O(2^n)$  e  $O(2^n \times n^2)$ , respectivamente. A primeira devido ao processo de geração de todos os subconjuntos possíveis ( $2^n$ ) e a segunda devido ao processo de percorrer cada subconjunto ( $2^n$ ) e para cada qual percorrer todas as combinações de arestas entre o próprio e o seu complementar (no pior caso,  $(n \div 2)^2 \rightarrow n^2$ ).

Assim, verifica-se que a complexidade deste algoritmo é exponencial com um fator polinomial:  $O(2^n \times n^2)$ , o que reforça a ideia de que este algoritmo é impraticável para grafos de grande dimensão, daí a necessidade de algoritmos alternativos, como o algoritmo de pesquisa gulosa.

#### IV. ALGORITMO DE PESQUISA GULOSA

Atendendo ao algoritmo de pesquisa gulosa com heurísticas, este segue uma abordagem diferente, uma

**Algorithm 1** Exhaustive Search**Input:**  $G$  matriz de adjacencia**Output:**  $s, t$ , weight cut value

---

```

1: input_set  $\leftarrow \{0, 1, \dots, \text{len}(G) - 1\}$ 
2: subsets  $\leftarrow$  EMPTY LIST
3:  $n \leftarrow \text{LENGTH OF input\_set}$ 
    $\triangleright$  Generate all subsets
4: for  $r$  from 0 to  $n$  do
5:   for each  $S$  in combinations(input_set,  $r$ ) do
6:     Add  $S$  to subsets
7:   end for
8: end for
9: best  $\leftarrow$  input_set
10: weight  $\leftarrow 0$ 
    $\triangleright$  Evaluate each subset
11: for each  $S$  in subsets do
12:   new_weight  $\leftarrow 0$ 
13:   for each  $i$  in  $S$  do
14:     for each  $j$  in input_set -  $S$  do
15:       new_weight  $\leftarrow$  new_weight +  $G[i, j]$ 
16:     end for
17:   end for
18:   if new_weight > weight then
19:     best  $\leftarrow S$ 
20:     weight  $\leftarrow$  new_weight
21:   end if
22: end for
23:  $S \leftarrow$  best
24:  $T \leftarrow$  input_set - best
25: return  $S, T$ , weight

```

---

vez que não garante a obtenção da solução ótima, mas sim uma solução aproximada, em tempo polinomial. Isto acontece devido à natureza do algoritmo, que em cada etapa faz escolhas localmente ótimas, sem considerar o impacto global da escolha, na esperança de alcançar um ótimo global.

Para o desenvolvimento este algoritmo, foi necessária uma análise a diversas regras heurísticas [4], com o objetivo de determinar a melhor estratégia a seguir. Desta forma, a estratégia escolhida pode ser examinada em detalhe no pseudocódigo apresentado a seguir.

Pode-se observar que o algoritmo tem duas etapas: uma de pré-processamento, onde são extraídas as arestas e os seus pesos e ordenados de forma decrescente, e outra de processamento, onde as arestas são processadas de acordo com as regras heurísticas definidas.

A primeira etapa deste algoritmo é a mais custosa em termos de complexidade, sendo  $O(n^2 + m \log m)$ . No pior caso, quando o grafo é denso, tem-se  $m = 0.5 \cdot n(n - 1)$ , o que faz com que o termo  $m \log m$  se torne dominante, e a complexidade da etapa passe a ser aproximadamente  $O(n^2 \log n)$ .

A segunda etapa, por sua vez, tem uma complexidade  $O(m)$ , dado que percorre todas as arestas do grafo, sendo que quando o grafo é denso, tem-se  $O(n^2)$ .

Portanto, pode-se concluir que esta abordagem ap-

**Algorithm 2** Greedy Heuristic**Input:**  $G$  matriz de adjacencia**Output:**  $s, t$ , weight cut value

---

```

1:  $n \leftarrow \text{len}(G)$ 
    $\triangleright$  Extract edges and their weights
2: edges  $\leftarrow$  EMPTY LIST
3: for  $i$  from 0 to  $n - 1$  do
4:   for  $j$  from  $i + 1$  to  $n - 1$  do
5:     weight  $\leftarrow G[i, j]$ 
6:     Add  $(i, j, \text{weight})$  to edges
7:   end for
8: end for
9: Sort edges in descending order by weight
    $\triangleright$  Process each edge
10: cut_weight  $\leftarrow 0$ 
11: seen,  $S, T \leftarrow$  EMPTY SETS
12: for each  $(u, v, \text{weight})$  in edges do
13:   if  $u$  not in seen and  $v$  not in seen then
14:     cut_weight  $\leftarrow$  cut_weight + weight
15:     Add  $u$  to  $S$ , add  $v$  to  $T$ 
16:     Update seen with  $\{u, v\}$ 
17:   else if  $u$  in  $S$  and  $v$  not in seen then
18:     cut_weight  $\leftarrow$  cut_weight + weight
19:     Add  $v$  to  $T$ , add  $v$  to seen
20:   else if  $u$  in  $T$  and  $v$  not in seen then
21:     cut_weight  $\leftarrow$  cut_weight + weight
22:     Add  $v$  to  $S$ , add  $v$  to seen
23:   else if  $v$  in  $S$  and  $u$  not in seen then
24:     cut_weight  $\leftarrow$  cut_weight + weight
25:     Add  $u$  to  $T$ , add  $u$  to seen
26:   else if  $v$  in  $T$  and  $u$  not in seen then
27:     cut_weight  $\leftarrow$  cut_weight + weight
28:     Add  $u$  to  $S$ , add  $u$  to seen
29:   else if  $u$  in  $S$  and  $v$  in  $T$  then
30:     cut_weight  $\leftarrow$  cut_weight + weight
31:   else if  $v$  in  $S$  and  $u$  in  $T$  then
32:     cut_weight  $\leftarrow$  cut_weight + weight
33:   else  $\triangleright$  Both  $u$  and  $v$  have been seen, skip
34:   end if
35: end for
36: return  $S, T$ , cut_weight

```

---

resenta uma complexidade polinomial com um fator logarítmico adicional:  $O(n^2 \log n)$ .

## V. ANÁLISE DOS RESULTADOS

Após a implementação e execução dos algoritmos de pesquisa exaustiva e heurística gulosa, através do ficheiro *benchmarks.py*, foi possível analisar os resultados obtidos, nomeadamente o número de operações básicas, o tempo de execução, a quantidade de diferentes soluções testadas e a precisão da solução da heurística gulosa.

## A. Análise do Número de Operações

Pelo facto da complexidade de um algoritmo ser uma medida fundamental para compreender a sua eficiência, e a primeira poder ser medida em termos do

número de operações básicas que ele realiza, em função do tamanho do parâmetro de entrada [5], esta análise visa validar as complexidades teóricas previamente discutidas nas secções referentes a cada algoritmo.

Para isso é criado um gráfico ...

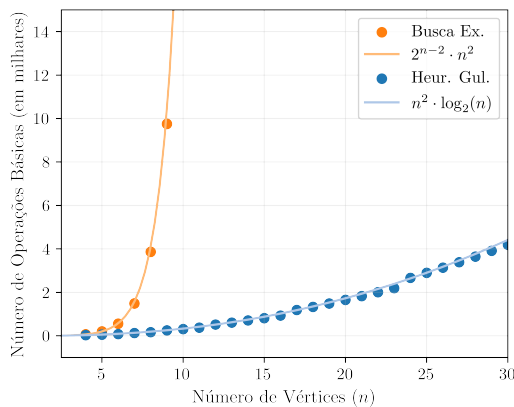


Fig. 1: Número de operações básicas realizadas pelos algoritmos de busca exaustiva e heurística gulosa em função do número de vértices, para diferentes grafos.

Este gráfico apresenta a quantidade de operações realizadas pelos algoritmos de busca exaustiva e heurística gulosa em função do número de vértices dos grafos analisados.

O gráfico do número de operações realizadas reflete a complexidade formal dos algoritmos, proporcionando uma análise prática que confirma a relação entre o crescimento do número de operações e a complexidade teórica previamente discutida.

A comparação entre essas duas abordagens permite não apenas visualizar a eficiência relativa

é de notar q n se nota diferencas de ops consoante o numero de arestas, isto pq o os algoritmos acabam por analisar todas as arestas, mesmo q tenham valor 0 fig 1

### B. análise de tempo de execução

para o calculo dos tempos de execucao, foram feitas pelo menos 3 execucoes para cada grafo tendo sido escolhido o tempo minimo entre elas para ter uma maior coerencia/precisao/whatever. para alem disso foi corrido num macbook ...

tendo em conta q varia com o ambiente ou assim ns q verifica se comportamento identeco ao numero de ops, o heuristico é mais rapido e tal 2

o exuastivo começa a crescer exponencialmente a partir dos ... vertices, tornando mt lenta a sua execucao ns q foram ajustadas regressoes para os tempos de execucao, de modo a prever o tempo de execucao para grafos de maior dimensao

heuristica, best fit para tempo

$$1.86 \times 10^{-8} \times n^2 \log n$$

NMAE (%) hugo: 8.376400389731693

and

exuastiva, best fit para tempo

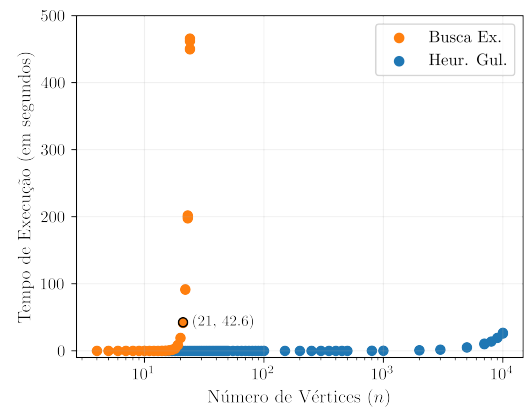


Fig. 2: Tempo de execução dos algoritmos de busca exaustiva e heurística gulosa em função do número de vértices, para diferentes grafos.

$$2^{(n-24.35)} \times n^2$$

NMAE (%) hugo: 2.484738758172613

ns q, boas medidas de erro

é de notas q os tempos de execucao nao sao uma grande medida para determinar a eficiencia dos algoritmos e complexidade e ns q, uma vez que depende de processador para processador. neste caso foram feitos num mac bla bla

### C. solucoes e precisao

ns q foram testadas  $2^n$  trivialmente, e 1 na heuristica, ns q, bla bla

usou se o conjunto de grafos do gset

[https://github.com/0816keisuke/](https://github.com/0816keisuke/max-cut-problem-benchmark)

max-cut-problem-benchmark e ns q e dps de fazer a precisao para os meus e para estes, tendo em conta a melhor conhecida .... e ns q mts nós e arestas comparativamente aos q criei e gset sao 69 grafos acho eu...

nível de confianca de 95% em torno da média [0.7631, 0.8120] para ter 95% de confiança que a média real está dentro do intervalo [0.7631, 0.8120].

## VI. CONCLUSÕES

dasdsad sn q disse

## REFERENCES

- [1] Rui-Sheng Wang and Li-Min Wang, "Maximum cut in fuzzy nature: Models and algorithms", *Journal of Computational and Applied Mathematics*, vol. 234, no. 1, pp. 240–252, 2010.
- [2] Noga Alon, Béla Bollobás, Michael Krivelevich, and Benny Sudakov, "Maximum cuts and judicious partitions in graphs without short cycles", *Journal of Combinatorial Theory, Series B*, vol. 88, no. 2, pp. 329–346, 2003.
- [3] Stefan Steinerberger, "Max-cut via kuramoto-type oscillators", *SIAM Journal on Applied Dynamical Systems*, vol. 22, no. 2, pp. 730–743, 2023.  
URL: <https://doi.org/10.1137/21M1432211>

- [4] Jianan Wang, Chuixiong Wu, and Fen Zuo, “More on greedy construction heuristics for the max-cut problem”, 2023.  
**URL:** <https://arxiv.org/abs/2312.10895>
- [5] J. Buhler and S. Wagon, “Basic algorithms in number theory”, *Algorithmic Number Theory*, vol. 44, 2008.  
**URL:** <https://pub.math.leidenuniv.nl/~stevenhagenp/ANTproc/02buhler.pdf>