# Maximum Weight Cut

Hugo Veríssimo - 124348 - hugoverissimo@ua.pt

***Abstract*** – **This report presents the implementation and comparison of two methods for solving the Max Weight Cut problem: an exhaustive search and a greedy heuristic. The Max Weight Cut problem consists of dividing a graph into two complementary subsets in order to maximize the sum of the weights of the edges that connect the two subsets. While exhaustive search guarantees an optimal solution, its computational complexity makes it impractical for large graphs. On the other hand, the greedy heuristic provides fast solutions, although suboptimal ones. The experimental results demonstrate the trade-off between accuracy and performance, highlighting the efficiency of the heuristic for graphs with a larger number of vertices.**

***Resumo*** – **Este relatório apresenta a implementação e comparação de dois métodos para resolver o problema *Max Weight Cut*: uma busca exaustiva e uma heurística gulosa. O problema *Max Weight Cut* consiste em dividir um grafo em dois subconjuntos complementares, de forma a maximizar a soma dos pesos das arestas que ligam os dois subconjuntos. Enquanto a pesquisa exaustiva garante uma solução ótima, a sua complexidade computacional torna-a impraticável para grafos de grande dimensão. Por outro lado, a heurística gulosa fornece soluções rápidas, embora subótimas. Os resultados experimentais demonstram o compromisso entre precisão e desempenho, destacando a eficiência da heurística para grafos com maior número de vértices.**

## 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. Este 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:

**Algorithm 1** Exhaustive Search

**Input:** $G$ matriz de adjacencia
**Output:** s, t, weight cut value

```
input_set ← {0, 1, . . . , len(G) - 1}
subsets ← EMPTY LIST
n ← LENGTH OF input_set
                                        ▷ Generate all subsets
for r from 0 to n do
    for each S in combinations(input_set, r) do
        Add S to subsets
    end for
end for
best ← input_set
weight ← 0
                                        ▷ Evaluate each subset
for each S in subsets do
    new_weight ← 0
    for each i in S do
        for each j in input_set - S do
            new_weight ← new_weight + G[i, j]
        end for
    end for
    if new_weight > weight then
        best ← S
        weight ← new_weight
    end if
end for
S ← best
T ← input_set - best
return S, T, weight
```

Pode-se verificar que as duas fases deste algoritmo apresentam complexidades $O(2^n)$ e $O(2^n \times n^2)$, respetivamente. 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 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.

**Algorithm 2** Greedy Heuristic

**Input:** $G$ matriz de adjacencia
**Output:** s, t, weight cut value

```
n ← len(G)
                              ▷ Extract edges and their weights
edges ← EMPTY LIST
for i from 0 to n - 1 do
    for j from i + 1 to n - 1 do
        weight ← G[i, j]
        Add (i, j, weight) to edges
    end for
end for
Sort edges in descending order by weight
                                          ▷ Process each edge
cut_weight ← 0
seen, S, T ← EMPTY SETS
for each (u, v, weight) in edges do
    if u not in seen and v not in seen then
        cut_weight ← cut_weight + weight
        Add u to S, add v to T
        Update seen with {u, v}
    else if u in S and v not in seen then
        cut_weight ← cut_weight + weight
        Add v to T, add v to seen
    else if u in T and v not in seen then
        cut_weight ← cut_weight + weight
        Add v to S, add v to seen
    else if v in S and u not in seen then
        cut_weight ← cut_weight + weight
        Add u to T, add u to seen
    else if v in T and u not in seen then
        cut_weight ← cut_weight + weight
        Add u to S, add u to seen
    else if u in S and v in T then
        cut_weight ← cut_weight + weight
    else if v in S and u in T then
        cut_weight ← cut_weight + weight
    else            ▷ Both u and v have been seen, skip
    end if
end for
return S, T, cut_weight
```

Pode-se observar que o algortimo 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 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 apresenta 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 que ilustra o número total de operações básicas executadas, por ambos os algoritmos, para grafos com diferentes números de arestas e de vértices.

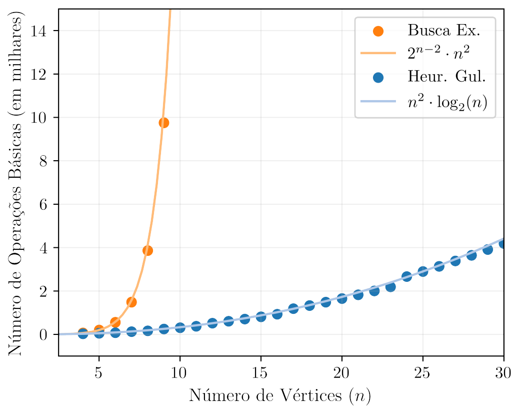


Fig. 1: Número de operações básicas realizadas pelos algoritmos de busca exaustiva e heurística gulosa em função do número de vértices, para diferentes grafos.

Pode-se observar no gráfico, a laranja, a informação relativa ao algoritmo de busca exaustiva, especificamente, os resultados para diferentes grafos, representados por pontos, e uma linha que representa a função $e(n) = 2^{n-2} \times n^2$, que modela de forma eficaz o comportamento do número de operações realizadas por este algoritmo. Analogamente, a azul, encontra-se a informação relativa ao algoritmo de heurística gulosa, cujos resultados são modelados pela função $f(n) = n^2 \log n$.

Assim, é possível verificar que o número de operações básicas realizadas pelos algoritmos é consistente com as complexidades teóricas previamente referidas: $O(2^n \times n^2)$, para o algortimo de pesquisa exaustiva, e $O(n^2 \log n)$, para o algoritmo de heurística gulosa, reforçando a ideia de que a heurística gulosa é mais eficiente que a busca exaustiva, em termos de complexidade.

Para além disso, é importante destacar que não há diferenças significativas no número de operações realizadas pelos algoritmos em relação ao número de arestas, o que é evidenciado no gráfico pelo facto de grafos com o mesmo número de vértices, mas com diferentes quantidades de arestas, estarem sobrepostos. Isto está relacionado com o facto de ambos os algoritmos analisarem todas as arestas possíveis, ao invés de apenas as arestas com peso diferente de zero, de forma a evitar possíveis problemas relacionados a nós soltos.

### B. *Análise de Tempo de Execução*

Outra métrica relevante na avaliação de algoritmos é o tempo de execução, uma vez que permite compreender a eficiência das operações e do algoritmo em relação ao tamanho do parâmetro de entrada. Ademais, esta análise possibilita comparações objetivas entre diferentes abordagens, facilitando a escolha do algoritmo mais adequado de acordo com cada situação.

Contudo esta métrica pode ser facilmente influenciada por diversos fatores, tais como o ambiente em que o algoritmo é executado, nomeadamente as especificações do *hardware* utilizado, e as características específicas dos grafos. Para minimizar a risco do enviesamento destes resultados, foram realizadas pelo menos três medições para cada grafo, tendo sido escolhido o tempo mínimo entre elas, para ter uma maior coerência nos dados obtidos. Para além disso, todas as execuções foram realizadas num *MacBook Air*, com um processador *Apple M1* e $16GB$ de memória RAM, garantindo a manutenção de um *hardware* constante para a análise.

Assim, para a realização desta análise, foi criado um gráfico que ilustra o tempo de execução de cada algoritmo, para diferentes grafos, em função do número de vértices.

A partir do gráfico, é possível verificar os tempos de execução, em função do número de vértices para o algoritmo de busca exaustiva, a laranja, e para o algoritmo de heurística gulosa, a azul.

Atendendo ao algoritmo de busca exaustiva, é possível reparar que a partir de $n = 21$, o tempo de execução explode, sendo mais notório o comportamento exponencial do algoritmo. Por outro lado, como seria de esperar, o algoritmo de heurística gulosa apenas apresenta o seu crescimento quadrático, de forma mais notória, a partir de $n \approx 10^4$, revelando-se significativamente mais eficiente que o algoritmo de busca exaustiva em grafos de maiores dimensões.

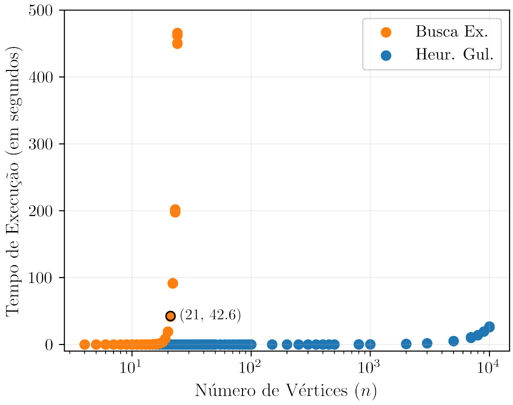


Fig. 2: Tempo de execução dos algoritmos de busca exaustiva e heurística gulosa em função do número de vértices, para diferentes grafos.

De modo a prever o tempo de execução para grafos de maior dimensão, foram ajustadas regressões para os tempos de execução de cada algoritmo, de modo a obter funções que modelassem o comportamento dos mesmos. O critério usado para a modelação foi a minimização da medida de erro *Normalized Mean Absolute Error* (NMAE).

Assim, obteve-se que o tempo de execução do algoritmo de busca exaustiva pode ser modelado pela função $g(n) = 2^{(n-24.35)} \times n^2$, com um NMAE de 2.48%, e o tempo de execução do algoritmo de heurística gulosa pela função $h(n) = 1.86 \times 10^{-8} \times n^2 \log n$, com um NMAE de 8.38%.

Por fim, nota-se que esta métrica não depende do número de arestas, como seria de esperar, uma vez que o tempo de execução é influenciado pelo número de operações básicas realizadas, que por sua vez apenas dependem do número de vértices, e é também importante referir que as regressões obtidas estarão fortemente correlacionadas com o *hardware* utilizado na obtenção dos tempos, pelo que os resultados obtidos podem não ser generalizáveis para outros ambientes.

### C. Análise de Solucões e Precisão

Em última análise, foi possível comparar a quantidade de diferentes soluções testadas pelos algoritmos, bem como a precisão da solução da heurística gulosa, para diferentes grafos.

#### C.1 Quatidade de Soluções Testadas

Atendendo ao algoritmo de pesquisa exaustiva, sabendo que esta testa todas as combinações de subconjuntos possíveis, trivialmente, o número de soluções testadas é dado por $2^n$, o que é possível verificar tanto analiticamente, como através da análise dos resultados obtidos.

Por outro lado, o algoritmo de heurística gulosa, apenas testa uma solução, a que é obtida pela heurística.

Assim, é possível verificar que o número de soluções testadas pelo algoritmo de busca exaustiva é exponencial, enquanto que o número de soluções testadas pelo algoritmo de heurística gulosa é constante, o que reforça a ideia de que a heurística gulosa é mais eficiente, contudo, apenas o primeiro algoritmo garante a solução ótima, ou seja, uma precisão constante de 100%.

#### C.2 Precisão da Heurística Gulosa

Quanto à precisão da heurística gulosa, esta varia de acordo com o grafo em análise, uma vez que não é garantida a solução ótima. Esta pode variar entre 1 (100%), quando a solução obtida é idêntica à solução ótima, e aproximadamente 0 (0%), quando as escolhas locais da heurística são muito desfavoráveis para o corte global.

De forma a criar um intervalo de confiança para a precisão deste algoritmo, foram comparados os pesos obtidos por este algoritmo com os pesos obtidos pelo algoritmo de busca exaustiva, para grafos até 24 vértices, com 4 diferentes densidades cada, e com os melhores pesos conhecidos para os grafos pertencentes ao conjunto *Gset* [6], [7], para testar grafos de maiores dimensões, em particular entre 800 e 10000 vértices.

Assim, após medir a precisão do algoritmo de heurística gulosa para 153 grafos distintos, obteve-se um intervalo de confiança de 95% em torno da média, permitindo concluir, com 95% de confiança, que a precisão média real do algoritmo está contida no intervalo $[0.7631, 0.8120]$.

PODE SE FAZER UM GRAFICO 3D OU ASSIM PARA VER SE DEPENDE DAS ARESTAS (% e avsoluto) OU DA QNT DE NÓS OU ASSIM

## VI. CONCLUSÃO

exaustivo garante sol otima mas e mt complexo, demora mt tempo, ineficiente ns q para grandes

dai haver necessidade de novos algoritmo

heuristica e bom, mas n garante otimo, tem boa complexidade

ns q fazer ponte para random ?

## 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`

[6] Yinyu Y. and S. Karisch, "Gset: A collection of graphs for benchmarking", Stanford University, n.d., Accessed: 2024-11-02.
**URL:** `https://web.stanford.edu/~yyye/yyye/Gset/`

[7] Y. Matsuda, "Benchmarking the max-cut problem on the simulated bifurcation machine", *Toshiba SBM, Medium*, September 2019, https://medium.com/toshiba-sbm/benchmarking-the-max-cut-problem-on-the-simulated-bifurcation-machine-e26e1127c0b0.