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 English 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. 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:

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

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
```

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
```

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 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

corrido o ficheiro benchmarks.py, foram obtidos varias medidas de resultados ns q bla bla

A. *análise de numero de operacoes*

grafico do numero de ops, corresponde a complexidade referida formalmente anteriormente

GRAFICO

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

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 ...

GRAFICO

ns q verifica se comportamento identeco ao numero de ops, o heuristico é mais rapido e tal

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

$a \times n^2 \log n [1.85901534e-08]$

NMAE (%) hugo: 8.376400389731693

and

exuastiva, best fit para tempo

$2^{(n-a)} \times n^2 [24.34775728]$

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 testasdas 2^n trivialmente, e 1 na heuristica, ns q, bla bla

usou se o conjunto de grafos do gset `https://github.com/0816keisuke/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...

nivel 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`