Maximum Weight Cut Problem

Hugo Veríssimo - 124348 - hugoverissimo@ua.pt

***Abstract* – ... abstrato em ingles**

***Resumo* – Este relatório apresenta a implementação e comparação de dois métodos para resolver o problema *Maximum Weight Cut*: uma pesquisa exaustiva e uma heurística gulosa. O problema *Maximum Weight Cut* con ESTE É O ANTIGO FAZER NOVO**

I. Introdução

O problema *Maximum Weight Cut* é um problema de otimização, que tem como objetivo encontrar o corte mais pesado num grafo não direcionado $G(V, E)$, onde $|V| = n$ vértices e $|E| = m$ arestas. Este corte envolve dividir os vértices do grafo em dois subconjuntos disjuntos S e T, sendo que o corte é a soma dos pesos das arestas que ligam os vértices de S aos vértices de T: $|E(S, T)|$ [1].

No passado relatório foram analisados algoritmos determinísticos para resolver o problema *Maximum Weight Cut*, nomeadamente a pesquisa exaustiva e a heurística gulosa. Neste relatório, serão analisados novos algoritmos com um certo grau de estocasticidade, com o objetivo de encontrar um algortimo que otimize o equilíbrio entre a complexidade computacional e a qualidade da solução obtida.

para alem disso os resultados são comparados aos obtidos anteriormente

serao entao implexmentados 3 algoritmos, nomeadamente: ... e ...

II. Metodologia da Análise

Com o intuito de analisar o problema em destaque, implementar os algoritmos referidos e comparar os resultados obtidos, foi utilizada a linguagem de programação *Python*, devido à vasta variedade de bibliotecas que contém, facilitando a implementação eficiente e simplificada dos algoritmos necessários.

Sem desmerecer o uso de ficheiros auxiliares, a análise desenvolvida pode ser dividida em 2 ficheiros principais, sendo estes:

```
$ python3 benchmarks.py
```

Para a realização da análise dos algoritmos criados, foram utilizados grafos gerados aleatoriamente, com a semente 124348, com diferentes números de vértices e densidade de arestas, e os grafos da coleção *Gset*, disponibilizada por Yinyu Ye [2].

III. Algoritmo de Particionamento Aleatório

O primeiro algoritmo a ser implementado é um algoritmo de particionamento aleatório, que consiste em gerar várias soluções aleatórias e comparar as mesmas, escolhendo a melhor solução [1].

Este será um algortimo computacionalmente leve, pela sua simplicidade, mas não garante a obtenção da solução ótima, devido à sua natureza aleatória, sendo que a probabilidade de encontrar a mesma, assumindo que é única, é dada por

$$1 - \left(1 - \frac{1}{2^{n-1}}\right)^{solutions}$$

onde n é o número de vértices e *solutions* é o número de soluções a gerar. Pode-se facilmente verificar que, para grafos de grandes dimensões, esta probabilidade decresce exponencialmente, tornando o algoritmo cada vez menos preciso.

Pelo facto do algoritmo gerar muitas soluções aleatórias, é importante garantir que não existem soluções repetidas a ser testadas, para evitar o cálculo do peso do corte, uma operação computacionalmente cara. Para isso é criado um *set* onde serão guardadas as soluções já testadas, e cada vez que uma solução for gerada, a mesma só será testada depois de ser verificado que não é uma repetição.

Atendendo à paragem do algoritmo, este tem dois critério de paragem, parando assim que um deles é verificado. O primeiro, e mais provável em grafos de grandes dimensões, é quando o número de soluções geradas atinge o limite, definido pelo utilizador. O segundo critério, é verificado quando todas as soluções possíveis foram testadas, ou seja, quando o *set* que acompanha as soluções testadas contém 2^n elementos.

Este algoritmo pode ser então traduzido para o seguinte pseudocódigo:

Algoritmo 1 Particionamento Aleatório

Entrada:
- lista de arestas e respetivos pesos (*edges*)
- número de vértices (*n_nodes*)
- número de soluções a gerar (*solutions*)

Saída: subconjuntos S e T, peso do corte (*weight*)

```
best_solution ← None
weight ← 0
seen_solutions ← empty set
for i ← 1 to solutions do
    partition ← random partition of the nodes
    if length(seen_solutions) = 2^n_nodes then
        break
    end if
    partition_hash ← hash the partition
    if partition_hash ∈ seen_solutions then
        continue
    end if
    Add partition_hash to seen_solutions
    new_cut_weight ← compute the cut weight
    if new_cut_weight > weight then
        weight ← new_cut_weight
        best_solution ← copy of partition
    end if
end for
S ← set of nodes assigned to 0 in best_solution
T ← set of nodes assigned to 1 in best_solution
return S, T, weight
```

quando a complexidade, este algortimo, a parte mais cara é o loop que corre no maximo `solutions` vezes, e dentro dele, a complexidade é O(n + m) por gerar uma particao aleatoria e calcular o peso do corte, logo a complexidade final é $O((m+m) \times \texttt{solutions})$, tendendo para $O(n^2)$ para grafos densos e um n grande
meter grafico e verificar complexidade

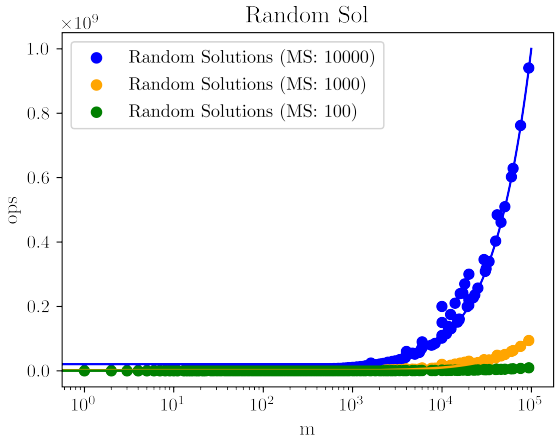

Fig. 1: camptionsdasid8

IV. ALGORITMO DE 2

o segundo algortimo a ser implementado é o Simulated Annealing, que é um algoritmo de otimizacao global, que procura a melhor solucao possivel, e que é baseado no processo de arrefecimento de metais, que consiste em arrefecer um metal a uma taxa controlada, para que os atomos se organizem de forma a minimizar a energia do sistema. ns q, o algortimo simulated Annealing consite em ... e é heuristico e ns q e random (referencias)

este algortimo já foi implementado no problema max cut por exemplo [3]

neste caso tem como componente aleatorio a selecao de uma solucao inicial e a aceitacao de solucoes piores, com uma probabilidade que decresce com o tempo, mas sendo esta aceite smp que a solucao for melhor que a anterior (determinisico)

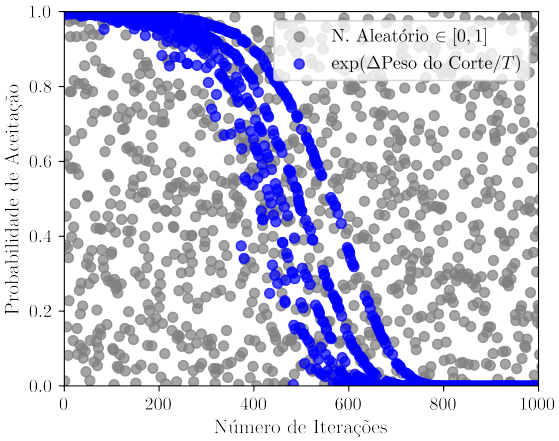

Fig. 2: ver que a aceitacao diminui com as iteracaoes por causa do arrefcimento e bla bla, G59 usado e seed do SA 124348

por ser sensivel a solucao inicial, é interessante testar diferentes solucoes iniciais, e por isso o algoritmo deve ser corrido um numero de vezes

neste algoritmo não ha garantia de que cada sol so é testada uma vez, por opção propria. visto que a probabilidade de testar a mesma solucao é dada por:

...

para grafos de maiores dimensões, esta prob torna-se muito baixa, e por isso a probabilidade de testar a mesma solucao é baixa, fazendo com que o processo de comparar solucoes ja testadas possa prejudicar a eficiencia do algoritmo

o algortimo para quando a temperatura é menor que 10^{-3}, pelo que o numero de iteracoes é variavel e depende do valor da temperatura inicial e da taxa de arrefecimento

Algoritmo 2 *Simulated Annealing*

Entrada:
- lista de arestas e respetivos pesos (*edges*)
- temperatura (*Temp*)
- taxa de arrefecimento (*cooling_rate*)

Saída: subconjuntos S e T, peso do corte (*best_cut*)

```
partition ← random partition of the nodes
best_partition ← partition
current_cut ← compute the cut weight
best_cut ← current_cut
while Temp > 10^-3 do
    node ← randomly select a node
    Flip the partition of node in partition
    new_cut ← compute the new cut weight
    cost_diff ← new_cut − current_cut
    if cost_diff > 0 or random number ∈ [0, 1] < e^(cost_diff/Temp) then   ▷ Accept the move
        current_cut ← new_cut
        if new_cut > best_cut then
            best_cut ← new_cut
            best_partition ← partition
        end if
    else                                                                   ▷ Reject the move
        Revert the partition of node in partition
    end if
    Temp ← Temp × cooling_rate
end while
S ← set of nodes assigned to 0 in best_partition
T ← set of nodes assigned to 1 in best_partition
return S, T, best_cut
```

- complexidade on4 ?

tendo em conta a complexidade do algortimo, o gerar uma particao inicial envolve o(n) pq vai vertice a evrtice atribuir determinado subt. depois o valvulo do current cut envolve o(m) visto q corre a lista de todos os vertices

depois corre o loop K vezes

esolhe um nó e muda a sua partição o(1), recalcula o novo peso do corte o(m) e depois compara o novo corte com o anterior o(1) e se for melhor atualiza o corte e a particao o(1)

ou seja a complexidade é O(m) * K

K é dado por:

$$
\begin{aligned}
& T_0 \cdot (\text{cooling\_rate})^k \leq 10^{-3} \\
\Leftrightarrow\ & k \geq \frac{\log\left(\frac{10^{-3}}{T_0}\right)}{\log(\text{cooling\_rate})} \\
\Leftrightarrow\ & k = \left\lceil \frac{\log\left(\frac{10^{-3}}{T_0}\right)}{\log(\text{cooling\_rate})} \right\rceil
\end{aligned}
$$

logo a complexidade final é dada por $O(m) \approx O(n^2)$ para grafos densos , pq k é uma constante que não depende nem de m nem de n

V. Algoritmo de 3

este algoritmo consiste numa heuristica gulosa, que consiste em iterar por todos os vertices e trocar a sua particao, verificando se a solucao é melhor, e se for, atualiza a solucao, e quanto iterar por todos e nao melhorar em nenhum para.

contudo isto tornava o algoritmo muito lento, apesar dos bons repertidos, para isso foi adicionado um fator de ajuste do maximo de iteracoes (*itLim*), que é o numero de arestas vezes o fator de ajuste, evitando assim que o algoritmo corra indefinidamente

como o algoritmo é guloso, a solucao final depende da solucao inicial, gerada aleatoriamente, e por isso o algoritmo deve ser corrido varias vezes, para garantir uma maior probabilidade a melhor solucao é encontrada

neste algoritmo, como a unica componente aleatoria é a particao inciial e como todas as alteracoes sao feitas em diracao a melhor solucao, o algoritmo nunca ira testar a mesma solucao duas vezes, pelo que nao é necessario guardar as solucoes ja testadas

Algoritmo 3 NOME DO ALGORTIMO

Entrada:
- lista de arestas e respetivos pesos (*edges*)
- número de vértices (*n_nodes*)
- fator de ajuste do máximo de iterações (*itLim*)

Saída: subconjuntos S e T, peso do corte (*weight*)

```
partition ← random partition of the nodes
cut_weight ← compute the cut weight
improved ← True
it_limit ← len(edges) × itLim
while improved and it_limit > 0 do
    it_limit ← it_limit −1
    improved ← False
    for node in range(n_nodes) do
        Flip the partition of node in partition
        new_cut_weight ← compute the cut weight
        if new_cut_weight > cut_weight then
            cut_weight ← new_cut_weight
            improved ← True
            break      ▷ Stop iteration for this node
        end if
        Revert the partition of node in partition
    end for
end while
S ← Set of nodes assigned to 0 in partition
T ← Set of nodes assigned to 1 in partition
return S, T, cut_weight
```

quanto a compelxidade, gerar a particao inicial e calcular o seu peso é O(n + m), pq corre a lista de vertices e a lista de arestas

depois com o ciclo, ira correr no maximo O(itLim x m) e dentro dele a compelxidade é O(n) por correr os nós todos x O(m) por calcular o peso a cada vertice q passa

logo a complixidade final é $O(m \times \texttt{itLim} \times n \times m)$ que tende para $O(n^5)$ para grafos densos

VI. Análise dos Resultados

Compare the results of the experimental and the formal analysis.
todos os grafos devem ser corridos pelo menos 5 vezes, e a media dos resultados deve ser calculada e mediana do tempo , por causa dos tempos e da aleatoriedade dos resultados
Graphs for the Computational Experiments: mine and elearnig and gset
asdasds

A. (1) the number of basic operations carried out

dsadasds

B. 2 the execution time

- Determine the largest graph that you can process on your computer, without taking too much time.
- Estimate the execution time that would be required by much larger problem instances.
dsadasd

C. solution

asdad

C.1 (3) the number of solutions / configurations tested

sadsad

C.2 precision

asdasd

Bibliografia

[1] Anupam Gupta, "15-854: Approximations algorithms", 2014, https://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec02.pdf. Accessed: 2024-11-28.

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

[3] Tor G. J. Myklebust, "Solving maximum cut problems by simulated annealing", 2015, https://arxiv.org/abs/1505.03068. Accessed: 2024-11-28.