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 ...

DIZER ALGURES Q O NUMERO DE OPERACOES SERA A METRICA USADA PARA CONFIRMAR A COMPLEXIDADE DOS ALGORTIMO

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 Corte Aleatório

O primeiro algoritmo a ser implementado é um algoritmo de corte 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 Corte 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
```

Pode-se observar que a parte computacionalmente mais custosa deste algoritmo é o ciclo, que é responsável por gerar cortes aleatórios e calcular o peso dos mesmos. A complexidade deste ciclo é dado por $O(n+m)$, por percorrer todos os vértices atribuindo-as a um dos subconjuntos, e por calcular o peso do corte, percorrendo todas as arestas. Assim, a complexidade final do algoritmo é dada por $O((n+m)\times \texttt{solutions})$, visto que o ciclo corre no máximo `solutions` vezes. Esta complexidade pode ser simplifcada para $O(m)$, visto que $m \gg n$ para grafos densos e um n grande, e que `solutions` é uma constante.

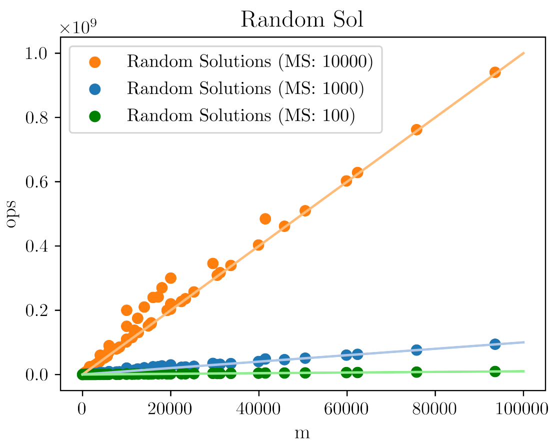

Fig. 1: camptionsdasid8

A complexidade referida é confirmada pela análise experimental apresentada na figura 1, onde se pode observar que o número de de operações básicas realizadas é linear em relação ao número de arestas do grafo, para diferentes números de soluções geradas.

IV. Algoritmo de Simulated Annealing

O segundo algoritmo a ser implementado é o algoritmo de *Simulated Annealing* (SA), uma heurística de pesquisa aleatória, que consiste em encontrar soluções aproximadas para problemas de otimização combinatória [3]. Este método consite em gerar uma solução inicial aleatória, e a partir desta solução, gerar soluções vizinhas, que são soluções obtidas a partir da solução atual, e comparar as soluções, aceitando as soluções melhores e algumas piores, com uma probabilidade que decresce ao longo das iterações, até que a temperatura (parâmetro do algoritmo), que vai arrefecendo ao longo das iterações a uma determinada taxa de arrefecimento (paraâmetro do algoritmo), seja menor que um determinado valor, por exemplo 10^{-3}, sendo este o critério de paragem do algoritmo [4].

Assim, é possível verificar que este algortimo tem como componente aleatória a seleção de uma solução inicial e a aceitação de soluções piores, e como parte determinística a diminuição da probabilidade de aceitação de soluções piores ao longo das iterações e a garantia de aceitação de soluções melhores.

Na figura 2, pode-se observar o comportamento decrescente da probabilidade de aceitação de soluções piores ao longo das iterações do algoritmo SA. Os pontos cinzentos representam números aleatórios gerados dentro do intervalo $[0, 1]$ a cada iteração. A proposta de uma nova solução é aceite se esta for melhor que a solução atual ou se o valor aleatório (ponto cinzento) estiver abaixo da "curva"definida pelos pontos azuis. Esta curva reflete a probabilidade de aceitação, que diminui à medida que a temperatura decresce, limitando cada vez mais a aceitação de soluções subótimas.

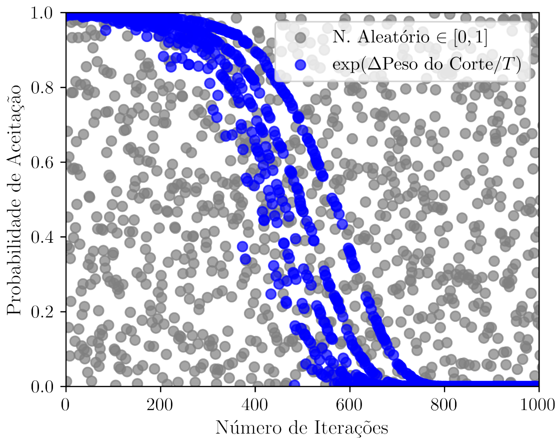

Fig. 2: Probabilidade de aceitação de uma solução subótima, em função do número de iterações, quando o SA é aplicado com a semente 124348, ao grafo G59.

É também importante referir que o algoritmo não garante a não repetição de soluções previamente testadas, e não foi implementado um mecanismo para tal, visto que a probabilidade de testar a mesma solução é baixa para grafos de grandes dimensões, e o processo de comparação de soluções já testadas poderia prejudicar a eficiência do algoritmo.

O algoritmo de *Simulated Annealing* pode ser examinado em detalhe no seguinte pseudocódigo:

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
```

Tal como foi referido na descrição do algoritmo, pode-se verificar que este é sensível à solução inicial, pelo que será interessante executar o algoritmo várias vezes, cobrindo uma maior área do espaço de soluções.

Para além disso, através do pseudocódigo, é possível analisar a complexidade do algoritmo em questão. As operações computacionalmente mais custosas encontram-se dentro do ciclo, que só termina após a temperatura ser inferior a 10^{-3}. Assim, torna-se importante calcular o total de iterações (k) que o ciclo irá realizar, o que pode ser feito através da seguinte equação:

$$\begin{aligned}
&\texttt{Temp}_0 \cdot (\texttt{cooling_rate})^k \leq 10^{-3} \\
\Leftrightarrow\ & k \geq \frac{\log\left(\frac{10^{-3}}{\texttt{Temp}_0}\right)}{\log(\texttt{cooling_rate})} \\
\Leftrightarrow\ & k = \left\lceil \frac{\log\left(\frac{10^{-3}}{\texttt{Temp}_0}\right)}{\log(\texttt{cooling_rate})} \right\rceil
\end{aligned}$$

Quanto à complexidade dentro do ciclo, a mesma é dada por $O(m)$, visto que a operação mais custosa é o cálculo do peso do corte, que percorre todas as arestas do grafo. Assim, a complexidade final do algoritmo é dada por $O(m \times k)$, e pelo facto de k depender apenas da temperatura inicial e da taxa de arrefecimento, a complexidade final pode ser aproximada por $O(m)$.

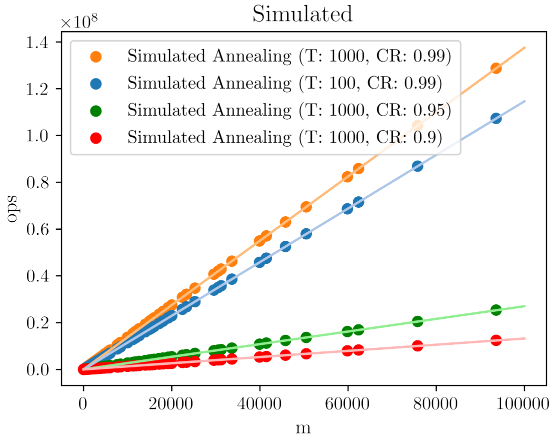

Fig. 3: camptionsdasid8

Através da figura 3 que representa o número de operações básicas realizadas em função do número de arestas do grafo, para diferentes temperaturas iniciais e taxas de arrefecimento, pode-se verificar que a complexidade do algoritmo é linear em relação ao número de arestas do grafo, para diferentes temperaturas iniciais e taxas de arrefecimento.

V. ALGORITMO GULOSO ALEATÓRIO

Por fim, o terceiro algoritmo a ser implementado é um algoritmo guloso aleatório, que segue uma heurística baseada numa abordagem gulosa, não garantido encontrar a solução ótima. Este algoritmo itera sobre todos os vértices do grafo e, para cada vértice, troca a sua partição, verificando se a nova configuração melhora a solução atual. Caso o peso do corte com o vértice na partição oposta seja maior que o atual, a solução é atualizada. O processo continua até que uma iteração completa seja realizada sem encontrar melhorias, momento em que o algoritmo termina.

Devido ao critério de paragem do algoritmo, este pode correr indefinidamente, devido à natureza aleatória da solução inicial, que pode estar a um grande número de iterações da solução estável que o algoritmo procura. Por isso, é adicionado um fator de ajuste do máximo de iterações (*itLim*) ao algoritmo, tornando o número máximo de iterações do mesmo $m \times itLim$.

É também importante referir que, pelo factor na única componente aleatória deste algoritmo ser a criação de uma solução inicial, e como todas as iterações realizam alterações em direção à melhor solução, o algoritmo nunca irá testar a mesma solução mais que uma vez, pelo que manter um registo sobre as soluções já testadas não é necessário.

Este algoritmo pode ser representado em pseudocódigo da seguinte forma:

Algoritmo 3 Guloso Aleatório

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
```

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

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.

[4] Galen Hajime Sasaki, "Optimization by simulated annealing: A time-complexity analysis.", Defense Technical Information Center, 1987, https://apps.dtic.mil/sti/pdfs/ADA185547.pdf. Accessed: 2024-11-28.