Algoritmos Avançados – 1º Projeto
Pesquisa exaustiva –Maximum Weight Cut

Mariana Rosa

Resumo **- Este artigo tem como propósito demonstrar o trabalho desenvolvido para o 1º projeto da disciplina Algoritmos Avançados, que consistiu em desenhar e testar um algoritmo de pesquisa exaustiva para resolver um problema de grafos: o Maximium Weight Cut. Para isso, duas abordagens foram feitas: uma pesquisa exaustiva e uma pesquisa gulosa.**

I. INTRODUÇÃO

O problema de grafos Maximum Weight Cut, em português, Peso Máximo de Corte [1], consiste em, dado um grafo não-direcionado *G=(V, E)* com o conjunto de vértices V = [1,2, … n] e um conjunto de arestas E com um peso $w_{i,j} \geq 0$ para cada $(i,j) \in E$, encontrar o peso máximo dado por um corte em arestas. Formam-se dois conjuntos de vértices, uma bipartição do conjunto V em S e T, tal que $S \cap T = 0$ e $S \cup T = V$. As arestas com os vértices em S, são cortadas e a soma do peso das suas arestas é o resultado do corte. Os vértices do conjunto S é a soma das arestas formados pelos vértices do conjunto S em T: $corte(S, T) = \sum_{i \in S, j \in T} w_{ij}$. O objetivo é encontrar o melhor conjunto S e T para se obter o maior peso possível para o corte. Este problema na teoria não é difícil de explicar, contudo quando vamos tentar resolver já não é bem assim, sendo classificado como um dos NP-hard problem. Matematicamente falando, este problema escreve-se com a seguinte fórmula[2]:

$pesoMáximo = max \sum_{i,j=1, i<j}^{n} w_{ij} * y_{ij}$. Onde:

- $y\,ij - x\,i - x\,j \leq 0,\ i, j = 1, 2, \ldots, n,\ i < j;$
- $y\,ij + x\,i + x\,j \leq 2,\ i, j = 1, 2, \ldots, n,\ i < j;$
- $x\,i \in \{0, 1\},\ i = 1, 2, \ldots, n.$

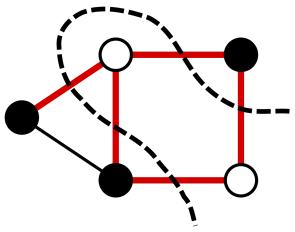


Fig. 1 - [3] Exemplo de demonstração do corte das arestas. Os vértices que estão a preto simbolizam o conjunto S, ou seja, todas as arestas que a estes se ligam, serão cortadas. Sendo o peso do corte igual ao peso (ou custo) de todas as arestas pelo qual o corte passa.

Para estudar este algoritmo, foi necessário realizar uma pesquisa exaustiva[4] e uma pesquisa gulosa[5]. A técnica de pesquisa exaustiva, também denominada de força bruta, tal como o nome diz resolve o problema através da exaustão, testando todas as possibilidades até encontrar a melhor solução. Podem ser simples em teoria, contudo esta técnica geralmente é menos eficiente em termos de tempo de execução. Uma vez que o algoritmo tem de calcular todas as possibilidades, a sua complexidade computacional aumenta, fazendo com que o tempo de execução para encontrar uma solução aumente também.
O algoritmo de pesquisa gulosa consiste em estabelecer regras (heurísticas). Ao definir uma heurística, o algoritmo tem uma certa regra a seguir, fazendo com que o número de possibilidades reduza, diminuindo o tempo de execução. É de notar que o algoritmo de pesquisa gulosa talvez não retorne a melhor solução, o objetivo deste algoritmo é encontrar a melhor opção disponível naquele momento, sendo por isso muito mais rápido. De notar que se o grafo a analisar não poderá ter vértices isolados, se não nenhum dos algoritmos irá funcionar.

II. ESTRUTURA DO CÓDIGO DESENVOLVIDO

O código para este problema está dividido em três ficheiros escritos em *Python* e podem ser corridos da seguinte maneira:

```
· $ python3 graph_generator.py
· $ python3 main.py
· $ python3 plot_results.py
```

No primeiro *script* são gerados 57 grafos aleatoriamente com a *seed* "98390", que são guardados nas respectivas pastas "*Percentage_X*" onde X pode ter o valor de 12.5, 25, 50 ou 75. Estes números correspondem à percentagem de arestas criadas. Os vértices são letras do alfabeto representados pela letra e pela sua posição, por exemplo: *[["A", [7, 6]]*. As arestas são representadas da seguinte forma: *[["C", "D"], ["A", "B"]]*.

No ficheiro *main.py* é onde está toda a elaboração dos algoritmos (tanto o da pesquisa bruta como o da pesquisa gulosa). No final do ficheiro existem funções comentadas, caso o utilizador queira realizar apenas um exemplo de cada algoritmo ou experimentar. As funções desenvolvidas são:

- *adjacency_list(edges)* : retorna a lista de adjacências das arestas;

· *check_if_there_are_isolated_vertex()* : retorna um valor booleano: verdadeiro se existir realmente algum vértice isolado; e falso se o contrário. Isto pois, não podemos resolver o problema caso haja algum vértice isolado;

· *calc_weight_edges(vertices, edges)* : retorna um dicionário com as chaves sendo arestas e o valor o peso entre as mesmas. Exemplo: *{('A', 'B'): 1, ('C', 'D'): 5};*

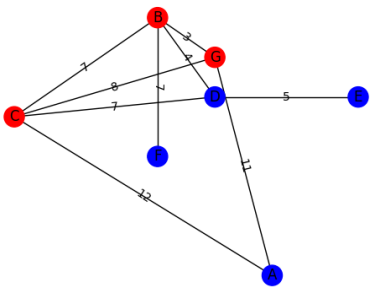
·*calculate_weight_cut(subsetS,subsetT, weight_list)* : retorna a soma total do corte do gráfico. Segundo o nosso problema, este valor obtém-se com a soma do peso das arestas formados pelos vértices de *S* adjacentes aos vértices de *T*, já explicado anteriormente;

·*find_max_cut_brute_force(vertex, edges)* : a função do algoritmo de força bruta, retorna uma lista composta pelo valor máximo do corte, o conjunto de vértices em *S*, o conjunto de vértice em *T* e o número de iterações realizadas na função;

·*find_max_cut_greedy(vertex, edges)* : a função do algoritmo de pesquisa gulosa, que retorna tal como a função anterior, uma lista composta pelo valor máximo do corte, o conjunto de vértices em *S*, o conjunto de vértice em *T* e o número de iterações realizadas na função;

·*plot_cut(vertices,edges, cuts,filepath,algorithm):* função que permite obter uma representação gráfica do grafo que analisamos que desenha o grafo com vértices em cores diferentes. Os vértices a vermelho pertencem ao conjunto S e os azuis ao conjunto T. Todas as representações visuais encontram-se na pasta *solutions*. Por exemplo, na pasta *Percentage_50* encontrámos o grafo da Fig. 2.

Fig. 2 - Grafo composto por 7 vértices, 9 arestas. S = {B, C, G} e T = {A, F, D, E }.



· *save_solution(vertex, edges, filepath):* esta função escreve todos os resultados do grafo, tanto através da força bruta, como também da pesquisa gulosa. Mais uma vez, todos os resultados encontram-se na pasta *solutions*, dentro das respectivas pastas de percentagem adequada. Recolhe dados sobre o número de vértices, arestas, o número do maior peso de corte, que vértices estão presentes no conjunto *S*, no conjunto *T*, o número de iterações realizadas e o tempo de execução de cada função.

· *plot_solutions_all():* função que percorre todos os grafos gerados pelo ficheiro *plot_results.py* e elabora a análise dos mesmos.

Por fim, o ficheiro *plot_results.py* é composto pelas funções responsáveis por extrair os resultados e representá-los graficamente. Todos os gráficos estão representados na pasta *graficos* e vão ser analisados posteriormente.

II. ALGORITMO DE FORÇA BRUTA

Este algoritmo foca-se em testar todas as possibilidades de dois subconjuntos dos vértices existentes em *V* para que se possa encontrar a melhor solução possível.

Para fazer as combinações de vértices possíveis, foi utilizado a livraria *itertools* que contém um método de combinações, gerando automaticamente todas as combinações possíveis para o conjunto *S* (*possible_cuts)*. De acordo com o problema, todos os vértices que estão em *S*, não estão em *T*. Assim, o conjunto *T* é formado pelos restantes vértices. Dado estes dois subconjuntos, é possível para cada um deles calcular o peso do corte para cada possibilidade com a função *calculate_weight_cut*. Por fim, um dicionário é criado com todos os pesos encontrados e retornado o maior valor encontrado, sendo esta a nossa melhor solução. Como foi dito anteriormente, esta pesquisa exaustiva é simples, contudo muito dispendiosa em termos de complexidade computacional, sendo esta uma função quadrática de complexidade de tempo $O(n^2)$. Pode ser descrito em pseudocódigo da seguinte forma:

Algoritmo 1: Algoritmo de Pesquisa Exaustiva

Input: *(V, E),* sendo que *V* e *E* formam um Grafo *G,* com pesos (custo) w_{ij}, $\forall i, j \in V, i \neq j$
Output: Uma lista formada pelo *custo* do corte *(S,T),* os conjuntos *S* e *T* e o número de iterações pelos ciclos *for*.
De notar que, $S \cap T = 0$ e $S \cup T = V$
adj_list ←Lista das adjacências do grafo *G*

weights ←Lista do custo (w_{ij}) de todas as arestas, sendo, w_{ij}, $\forall i, j \in V, i \neq j$
possible_cuts ←{}
dic_cut_weight ←{}, dicionário onde as chaves são o corte efetuado e o valor o maior corte possível no grafo
max_cut_vertice ←{}, lista das chaves (vértices) do *dic_cut_weight*
max_cut_value ← {}
iterations ← 0
for *i=1 ... range(V/2)* **do**
    possible cuts ← Todas as combinações possíveis de vértices
    for *cut in possible_cuts* **do**
    iterations ← iterations + 1
    subset_s ← *cut*
    subset_t ← V \ {subset_s}
    dic cut weight ← calcular o custo do corte com o conjunto S e T e guardar para cada um

max_cut_vertices ← *max{* chaves do dicionário *dic_cut_weight }*
max_cut_value ← valour da chave *max_cut_vertices*
return *max_cut_value*, *{S}*, *{T}*, iterations

III. ALGORITMO DE PESQUISA GULOSA

Para desenvolver o algoritmo de pesquisa gulosa, foi preciso idealizar um conjunto de heurísticas para tentar chegar a uma solução aceitável que responda ao problema. Após estudo de vários artigos, mais especificamente o artigo *On greedy construction heuristics for the MAX-CUT problem ref [2],* um conjunto de regras foram desenvolvidas. Nomeadamente, como ponto de partida para a elaboração do algoritmo, obtive algumas ideias dos algoritmos SG1 e SG2:

Algoritmo 2: Algoritmo de Pesquisa Gulosa

Input: (V, E), sendo que V e E foram um Grafo G, com pesos (custo) w_{ij}, $\forall i, j \in V, i \neq j$
Output: Uma lista formada pelo *custo* do corte *(S,T)*, os conjuntos S e T e o número de iterações. De notar que, $S \cap T = 0$ e $S \cup T = V$
weights ←Lista ordenada do custo (w_{ij}) de todas as arestas, sendo, w_{ij}, $\forall i, j \in V, i \neq j$
(*x,y)* ← Escolher a aresta mais pesada (*x,y), x Λ y* ∈ *V*
S ← {*x}*
T← {*y}*
custo ←0
if len(weights) == 1 **then**
    end
{w,z} = max{(*weights) \ (x,y)}* =
if *{w}* == *{x}* **then**
    T ← *{y,z}*
if *{w}* == *{y}* **then**
    S ← *{x,z}*
if *{z}* == *{x}* **then**
    T ← *{y,w}*
if *{z}* == *{y}* **then**
    S ← *{x,w}*
iterations ← 0
for {i,j} {(*weights) \ (x,y), {w,z}* **do**
    iterations ← iterations + 1
    Adicionar todos os restantes vértices a *T*
end
custo ← Calcular o custo de {*S*} e {*T*}
return custo, {*S*}, {*T*}, iterations

Como se pode observar a heurística estabelecida foi formar um conjunto *S* composto por dois vértices (tendo *G* mais de dois vértices) que possuem as arestas mais pesadas (com maior custo), com restrições para que o seu vértice adjacente pudesse estar no conjunto oposto. Assim, será feito um corte entre eles, entrando o seu custo para o custo total do corte. O conjunto T é formado por todos os restantes vértices que não estão em *S*.

Este algoritmo é claramente muito mais eficiente em termos de complexidade computacional, tendo *O(n)*. Contudo, não é o ideal uma vez que não encontra a melhor solução possível. Isto pois ao só escolher dois vértices limita muito o conjunto S, mas para encontrar uma das soluções é aceitável.

IV. RESULTADOS

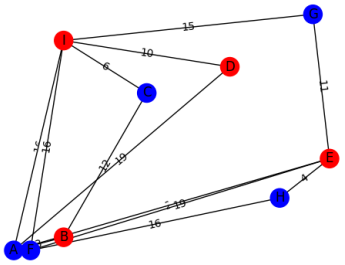
Para fazer uma análise de todos os resultados, foram recolhidos dados de 56 grafos, com vértices de n=2 a n=24, com quantidade de arestas diferentes (12.5%, 25%, 50% ou 75%) . Iremos dividir a análise por partes:

· Análise *Grafo resolvida por pesquisa exaustiva* VS *Grafo resolvida por pesquisa gulosa;*
· Avaliar o número de iterações;
· Avaliar o tempo de execução;
· Avaliar os resultados obtidos do corte máximo;
· Avaliar a aplicação dos algoritmos para problemas muitos maiores.

A. *Análise Grafo resolvida por pesquisa exaustiva VS Grafo resolvida por pesquisa gulosa*

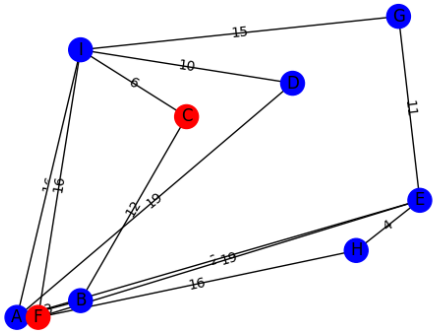
Aplicando um exemplo para o algoritmo de pesquisa bruta, o grafo da Figura 3 foi obtido através do mesmo. O corte máximo possível deste grafo foi obtido em 255 iterações durante 0.0026 segundos e tem um valor de 153.

Fig. 3 - Grafo composto por 9 vértices, 14 arestas. S = {B, D, E, I} e T = {A, C, F, G, H } resolvido por força bruta



Podemos observar na Figura 4 o mesmo grafo da Figura 3, mas neste caso resolvido pela pesquisa gulosa. O custo total é cerca de 83 tendo feito apenas 12 iterações. Encontrou uma solução muito mais rapidamente do que o algoritmo anterior, mais especificamente em 4.4 e-5 segundos.

Fig. 4 - Grafo composto por 9 vértices, 14 arestas. S = {F,C} e T = {A,B, D, E, G, H } resolvido pela pesquisa gulosa



B. Análise do número de iterações

Tal como discutido na seção anterior, o algoritmo de força bruta faz muitas iterações quando comparado com o algoritmo de pesquisa gulosa (Fig. 4). Uma vez que a sua complexidade computacional é $O(n^2)$ e a complexidade computacional do de pesquisa gulosa é $O(n)$[6].

Analisando individualmente cada um, o de força bruta (Fig. 5) cresce exponencialmente, chegando mesmo a fazer mais de 600.000 iterações para um grafo com 20 vértices, quando para um grafo com apenas 4 vértices faz apenas 10 iterações quando analisado pela pesquisa gulosa (Fig. 6).

Apesar do algoritmo de pesquisa gulosa ter sido desenvolvida para uma pesquisa mais rápida, com a minha implementação é de notar que à medida que o número de vértices do grafo aumentava, o número de iterações rapidamente cresce, quase como se tornasse exponencial, mas ainda assim cresce a um nível muito mais lento que o de força bruta. Isto é um aspecto positivo, pois beneficia o tempo de execução. Enquanto para 20 vértices o máximo de iterações foi 126 com este algoritmo e 616.665 iterações quando aplicado o de força bruta.

Fig. 5 -Gráfico de análise do número de iterações por força bruta

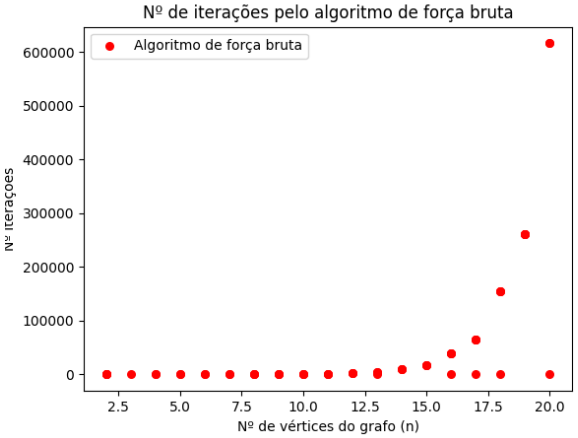


Fig. 6 -Gráfico de análise do número de iterações por pesquisa gulosa

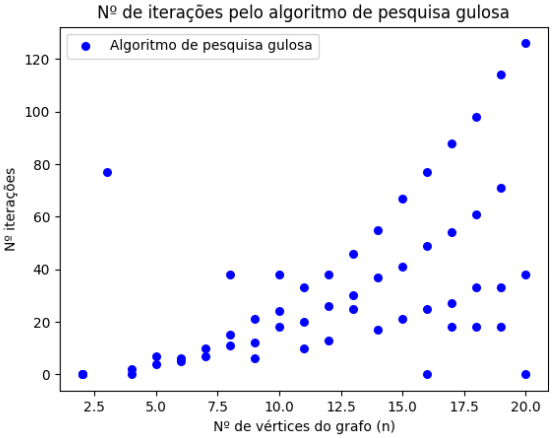
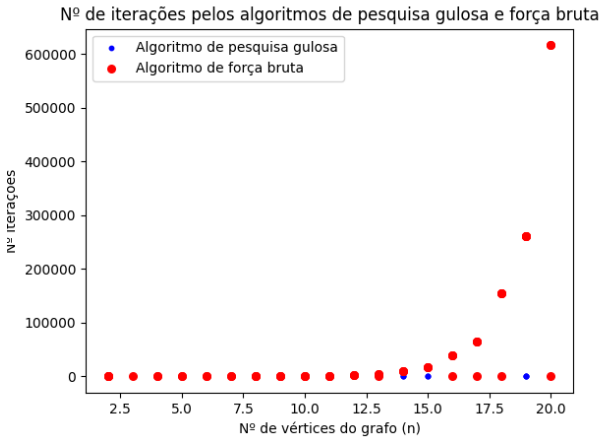


Fig. 7 -Gráfico a sobrepor as duas análises de iterações dos dois algoritmos



C. Análise dos tempos de execução

Ao comparar os dois algoritmos, não há dúvida que o algoritmo de pesquisa gulosa é muito mais rápido que o algoritmo de pesquisa exaustiva (Fig.8). Obtendo resultados completamente distintos: o grafo mais complexo destes dois conjuntos, exemplo o grafo da Figura 9 mais rápidos no algoritmo de pesquisa gulosa que contém 20 vértices e 128 arestas, demorou 200 segundos (3.3 minutos) por força bruta e 0.005 segundos por pesquisa gulosa, sendo 40.000 vezes mais rápido.

Podemos analisar graficamente pelas Figuras 9 e 10, que mais uma vez o tempo de execução da pesquisa exaustiva tem um crescimento exponencial, tendo uma gama de valores entre 0 e 200 segundos para o mesmo conjunto de vértices do algoritmo de pesquisa gulosa. Este último apenas varia de 0.0001 a 0.0005 segundos.

Fig. 8 - Gráfico a sobrepor as duas análises dos tempos de execução dos dois algoritmos

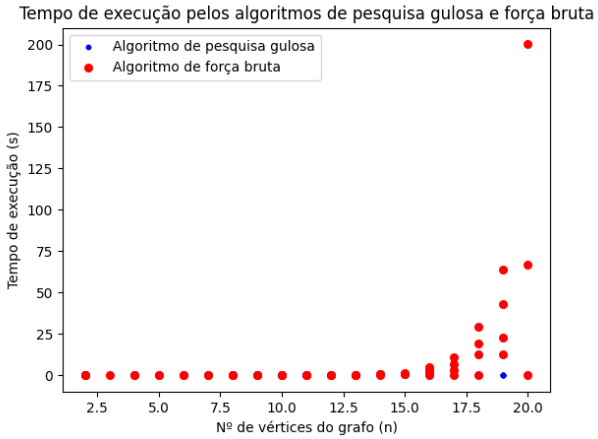


Fig. 9 -Maior grafo que pertence aos dados analisados, composto por 20 vértices e 128 arestas.

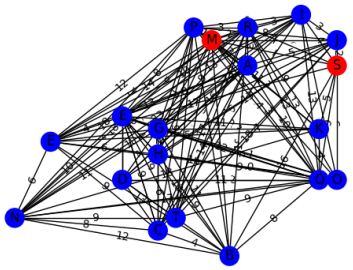


Fig. 10 - Gráfico a avaliar o tempo de execução do algoritmo de força bruta

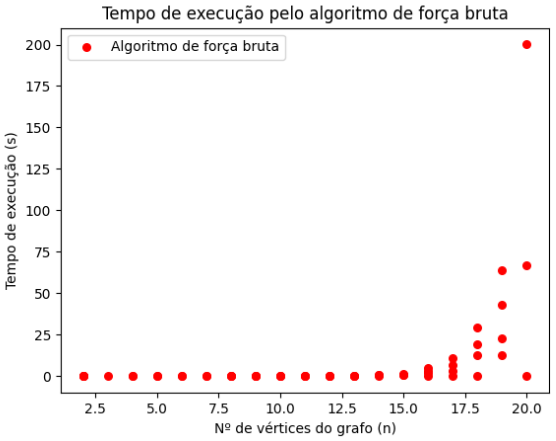
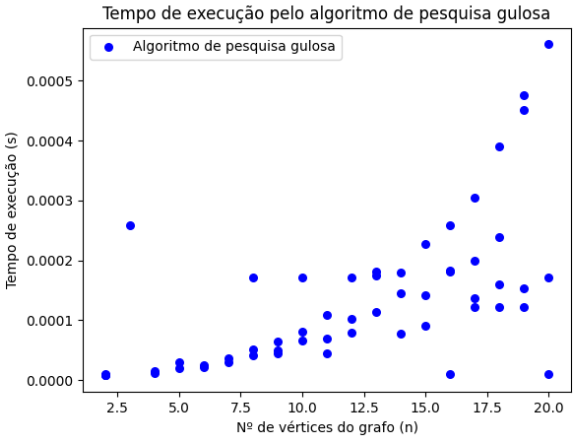


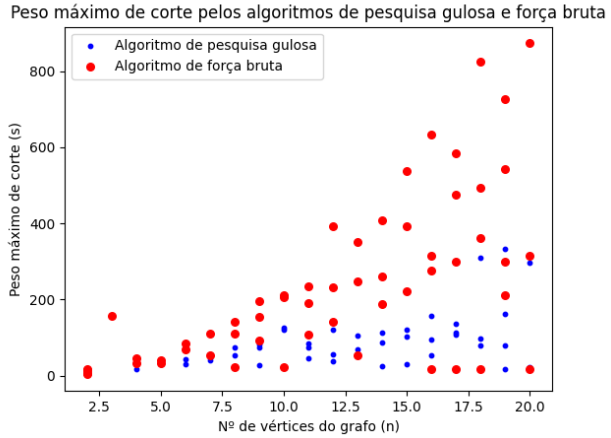
Fig. 11 - Gráfico a avaliar o tempo de execução do algoritmo de pesquisa gulosa



D. Análise dos resultados obtidos do corte máximo

Quando sobrepomos os dois gráficos (Fig. 12) que analisam o peso máximo obtido para os grafos, concluímos que o algoritmo de força bruta conseguiu resultados muito mais elevados em quase todos os grafos, mas também já era de esperar uma vez que o propósito deste algoritmo é mesmo ir buscar a melhor solução possível.

Fig. 12- Gráfico a sobrepor os resultados do peso máximo obtido pelos dois algoritmos



O maior peso conseguido através do algoritmo de pesquisa gulosa foi cerca de 297 para o grafo da figura 9, enquanto que na realidade o maior peso possível de corte é cerca de 873 (número obtido a partir da pesquisa exaustiva). Como se pode observar, para os grafos com um *n* mais baixo, os valores não diferem assim tanto, contudo quando o *n* aumenta aí sim já se distanciam mais. Tornando a gama de valores do eixo do *Y* dos dois algoritmos bastante díspares(Fig. 13 e 14).

Fig. 13- Gráfico dos resultados do peso máximo obtido pelo algoritmo de força bruta

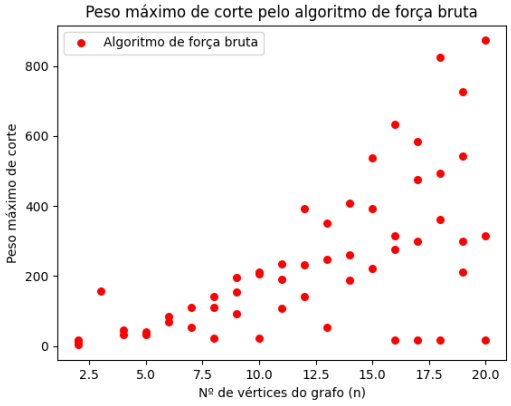
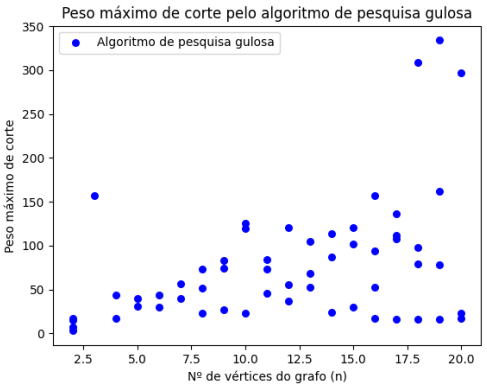


Fig. 14- Gráfico dos resultados do peso máximo obtido pelo algoritmo de pesquisa gulosa



E. Avaliar a aplicação dos algoritmos para problemas muitos maiores

Na geração dos grafos para análise, numa primeira tentativa estava a gerar os grafos de n=2 vértices até n=25. Contudo, notei que a partir de n=20 vértices o programa já começava a ser extremamente lento para o algoritmo de pesquisa bruta. Só para um teste, decidi analisar um grafo que continha 25 vértices e 207 arestas. Pela pesquisa de força bruta, conseguiu encontrar uma solução para o peso máximo (1534) contudo foi necessário quase 17 milhões de iterações, demorando 4 horas e meia. Pela pesquisa gulosa a solução encontrada foi muito menor, um peso de cerca de 459. Já demorou muito menos tempo (0.0014) segundos ao fazer 205 iterações.

Ou seja, para problemas muito maiores, o algoritmo de pesquisa exaustiva não seria de todo uma boa opção. Porém, o tempo de execução do algoritmo de pesquisa gulosa não se compara ao anterior, sendo assim possível aplicá-lo a uma maior escala, sabendo que não irá encontrar a melhor solução, mas irá certamente encontrar a melhor possível.

V. CONCLUSÃO

Por toda a análise realizada, os resultados sobre o problema em si definitivamente foram melhores com o algoritmo de força bruta. Porém, a complexidade computacional necessária para executar esse algoritmo é muito dispendiosa, sendo o tempo de execução muito maior. Através do algoritmo de pesquisa gulosa, foi demonstrado ser possível chegar a soluções relativamente próximas às melhores soluções encontradas no outro algoritmo para um grafo com n vértices < 12. Existem inúmeros outros algoritmos de pesquisa gulosa para solucionar este problema, muito provavelmente com melhor desempenho do que este. Pois a maioria dos valores não ficaram muito semelhantes, contudo ao menos consegue ser eficiente ao ter um tempo de execução baixo.

References

[1] P. Festa, P.M. Pardalos, M.G.C. Resende & C.C. Ribeiro (2002) Randomized heuristics for the Max-Cut problem, Optimization Methods and Software, 17:6, 1033-1058, DOI: 10.1080/1055678021000090033

[2] Sera Kahruman, Elif Kolotoglu, Sergiy Butenko and Illya V. Hicks (2002) On greedy construction heuristics for the MAX-CUT problem

[3] Corte Máximo - Wikipédia

[4] Brute Force Algorithms | Codecademy

[5] Greedy Algorithm

[6] Understanding time complexity with Python examples | by Kelvin Salton do Prado | Towards Data Science