Maior conjunto de vértices independentes em grafos

Vasco Vieira Costa - 97746

***Abstract* –The search for the maximum independent set in graphs is usually done in an exhaustive way, with a very high computational complexity, resulting in a unfeasible computational time. As an alternative heuristic methods exist that produce an approximated result or even the correct one in much less time. With this work both exhaustive and heuristic approaches to the task are analyzed with their advantages and disadvantages.**

***Resumo* –A procura do *maximum independent set* em grafos é tipicamente feita de uma forma exaustiva apresentando uma complexidade computacional muito elevada que resulta num tempo computacional inviável. Como alternativa existem métodos heurísticos que conseguem chegar a um resultado aproximado ou até o correto muito mais depressa. Com este trabalho comparam-se tanto a abordagem exaustiva como a heurística analisando as suas vantagens e desvantagens.**

***Keywords* –Maximum Independent Set, Computational Complexity, NP-hard, Exaustive Research, Greedy Heuristic**

***Palavras chave* –Conjunto Independente Máximo, Complexidade Computacional, NP-difícil, Pesquisa Exaustiva, Heurística Gananciosa**

I. Introdução

Este projeto parte com o objetivo de estudar diferentes formas de otimização em problemas, especificamente em grafos, na procura do maior conjunto de vértices independentes de um grafo, *maximum independent set*[1]. Este é um subconjunto de vértices do grafo em que nenhum par de dois vértices está ligado por uma aresta [1].

Esta modelação pode ser aplicada no mundo real em diversos tópicos como na representação de pessoas e dos contactos entre estas. Com utilizações nas mais diversas áreas como a de *marketing* nas redes sociais ou mesmo para a recomendação de pessoas a conhecer.

II. Algoritmia

O problema na sua generalidade é um problema considerado *strong np-hard*. Isto, por até à data não se ter encontrado um algoritmo de tempo polinomial que o consiga resolver [2].

Para a formulação dos grafos, `G`, fez-se uso do *package* `networkx` [3] para a sua criação, `fast_gnp_random_graph()` [4], e uso de instruções básicas em grafos, como a verificação quais os vizinhos de um vértice. Existindo também funções neste módulo para encontrar o *maximal* e uma aproximação ao *maximum* conjuntos independentes de vértices, `maximal_independent_set(G)` [5] e `maximum_independent_set(G)` [6], respetivamente, não sendo o primeiro elemento de estudo.

Da Fig. 1 é possível visualizar-se a diferença entre estes dois conjuntos. Procurando-se encontrar nesta resolução[2] apenas o *maximum*, que, ao não ser único apenas a sua dimensão importa para a verificação.

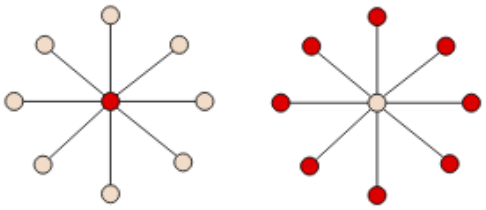


Fig. 1: Dois *maximal sets*, representados a vermelho, independentes para o mesmo grafo sendo o da direita o *maximum*. Figura de [7]

No código incluído existe a função `plot_graphs(G,result)` que permite visualizar o grafo original, `G`, com o subconjunto `result` em sobreposição para averiguar visualmente o funcionamento dos algoritmos, para um reduzido número de vértices. Na Fig. 2 é apresentado um *maximum independent set* com os vértices a vermelho, sendo que o grupo de vértices 2, 5, 6 e 7 também é um *maximum independent set* do grafo.

A. Exaustiva

[3] A tarefa apresentada resolve-se de forma tradicional num método de pesquisa exaustiva verificando de todas as combinações de n vértices do grafo quais satisfazem a condição. Não importando a ordem, o problema apresenta logo à partida uma complexidade de $\Omega(2^n)$, a cada vértice existem duas possibilidades, estar presente no conjunto ou não. Isto sem contar com o custo da verificação de se é uma combinação possível.

É então necessário verificar as combinações de todos os i tamanhos, de 1 até n. A cada combinação j são necessários verificar se os vértices são vizinhos. Efetuando-se isto com dois ciclos, um, k, começando no primeiro vértice e indo até ao penúltimo, $i-1$, e o

[1] Respondendo ao Problema 12

[2] Código disponível em `code.py`

[3] Método realizado pela função `exhaustive(G)`

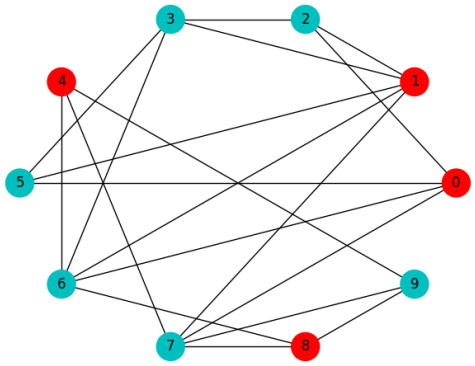


Fig. 2: Representação de um grafo de 10 vértices com percentagem de ser completo, k, de 0.4. A vermelho são apresentados os vértices pertencentes a um *maximum independent set*.

outro ciclo começando em $k+1$ e indo até ao último elemento, i. Podendo-se representar este número de operações como está na Equação 1. Verificando-se assim que este algoritmo apresenta uma complexidade de $\mathcal{O}(n^2 2^{n-3})$

$$\sum_{i=1}^{n}\sum_{j=1}^{\binom{n}{i}}\sum_{k=1}^{i-1}\sum_{l=k+1}^{i} 1 = 2^{n-3}(n-1)n = 2^{n-3}(n^2-n) \quad (1)$$

Sendo esta a contabilização de iterações necessárias ao algoritmo, não olhando a outras operações como comparações, por exemplo, ou à criação das listas das combinações dos vértices.

A.1 Exaustiva melhorada

[4] Uma vez que o processo anteriormente apresentado é bastante penoso, apesar de não se conseguir melhorar o pior caso, consegue-se melhorar o caso médio, Θ, e o melhor, Ω.

Ao verificarem-se primeiro as combinações que apresentam mais vértices e só depois os que apresentam menos vértices. Em grafos mais esparsos consegue-se assim primeiro chegar ao maior conjunto independente. Tendo em conta que apenas se procura o maior conjunto é escusado verificar as combinações do mesmo tamanho ou menores que o melhor caso já guardado. Conseguindo-se ainda melhorar mais o código, como é feito em `exhaustive_the_fastest(G)` introduzindo a comparação antes de gerar as combinações para memória, reduzindo assim o tempo computacional, função que não foi incluída para a análise geral.

B. Greedy Heuristic

[5] Devido à grande complexidade que este tipo de problemas apresenta existem situações em que é preferível obter uma resposta perto da correta, *heuristic*, ou até a correta mas sem a garantia de o ser, num tempo muito menor.

O vértice ideal e certo a incluir no *maximum independent set* é um isolado, sem vizinhos, isto por não ter qualquer aresta a ligá-lo a outros vértices. Partindo deste princípio pode-se esperar que os vértices com menor número de vizinhos sejam os que fazem parte do conjunto [8]. Tomando-se assim uma regra que se segue sempre com a esperança, *greedy*, que esta forneça o resultado correto, nunca regressando atrás para alterar ou verificar outra hipótese.

Deste modo, efetua-se uma iteração ao longo de todos os n vértices para verificar qual apresenta um menor número de vizinhos, escolhendo-se esse mesmo para incluir no *maximum independent set* e retira-se assim como os seus v vértices vizinhos, caso existam, do grafo. Repetindo-se este processo até já não existirem mais vértices no grafo. Em termos de complexidade computacional este não pode ser analisado da mesma forma como o algoritmo anteriormente apresentado pois o número de vizinhos difere de caso para caso. Se o grafo não apresentar vértices o algoritmo termina.

Na Equação 2 é possível formular este algoritmo, percorrendo os n vértices, escolhendo-o e removendo-o do grafo, 1, e depois removendo os v vizinhos do vértice escolhido. Efetuando o mesmo para o restante grafo que terá $n-1-v$ vértices, até se chegar a um grafo vazio.

$$f(n) = \begin{cases} 0 & , n = 0 \\ n+1+v+f(n-1-v) & , n \neq 0 \end{cases} \quad (2)$$

Como o grafo diminui de tamanho consoante o número de vizinhos do vértice escolhido, verifica-se que esta é uma técnica de *Decrease and Conquer*, vai-se reduzindo o tamanho do problema mas não por um fator constante em todas as situações. Nos casos de um grafo completo ou sem arestas é possível no entanto efetuar a formulação. Isto por se saber o número de vizinhos de cada vértice.

Caso não tenha arestas, esta pode-se representar pela situação apresentada na Equação 3. Até se chegar ao grafo vazio, percorrem-se os n vértices, remove-se um, 1 e chama-se outra vez a função para o restante grafo, como não existem vértices a remover $v = 0$.

$$f(n) = \begin{cases} 0 & , n = 0 \\ n+1+f(n-1) & , n \neq 0 \end{cases} \quad (3)$$

Efetuando-se a expansão para $(n+1)+(n)+f(n-2)$ e depois para $(n+1)+(n)+(n-1)+f(n-3)$ e generalizando para $(n+1)+(n)+(n-1)+...+(n+2-k)+f(n-k)$ terminando quando $k = n$. Chegando-se a $\sum_{k=1}^{n}(n+2-k) = \frac{1}{2}n(n+3)$. Refletindo-se assim numa complexidade computacional de $\mathcal{O}(n^2)$. Confirmando-se a complexidade estimada anteriormente, sendo este o pior caso.

Quando o grafo é completo primeiramente percorrem-se todos os n vértices, removendo-se o vértice escolhido, 1, e os seus vizinhos, $n-1$, e chamando-se a

[4] Método implementado pela função `exhaustive_faster(G)`
[5] Método implementado pela função `greedy(G)`

função outra vez que ao não ter mais vértices termina. Podendo-se representar pela Equação 4 que é igual a $2n$, sendo assim uma complexidade computacional de $\mathcal{O}(n)$ para este caso, o melhor.

$$f(n) = \begin{cases} 0 & , n = 0 \\ n + 1 + (n-1) + f(0) & , n \neq 0 \end{cases} \quad (4)$$

Podendo-se assim afirmar que a complexidade para este algoritmo será de $\mathcal{O}(n^2)$.

III. Iterações

De forma a efetuar um estudo prático das complexidades destes algoritmos aqui encontrados foram conduzidas experiências para diferentes dimensões de grafos, diferente número de n vértices e para diferentes números de arestas, representado como k que reflete a percentagem que este tem de ser um grafo completo, $K - complete$.

Na Figs. 3, 4, 5 e 6 é possível verificar o número de iterações que foram utilizadas por cada uma das funções, a exaustiva, a exaustiva melhorada e pela *Greedy* para diferentes valores de k, apresentados com as linhas de $n^2 2^{n-3}$, 2^n, n^2 e $n\log(n)$ para se terem referências a comparar com as complexidades analisadas formalmente anteriormente.

Na Fig. 3 verifica-se que nenhuma das funções exaustivas apresenta a pior complexidade mencionada anteriormente de $n^2 2^{n-3}$ estando a função exaustiva melhorada quase perto de 2^n. A função *Greedy* por comparação apresenta menos iterações, não chegando a uma complexidade de n^2. Contudo, nesta situação verificou-se que este método não funcionou plenamente, analisando o número máximo de elementos encontrados para o caso de 25, 26 e 27 vértices ficou um vértice aquém do obtido exaustivamente, visível na Fig. 12.

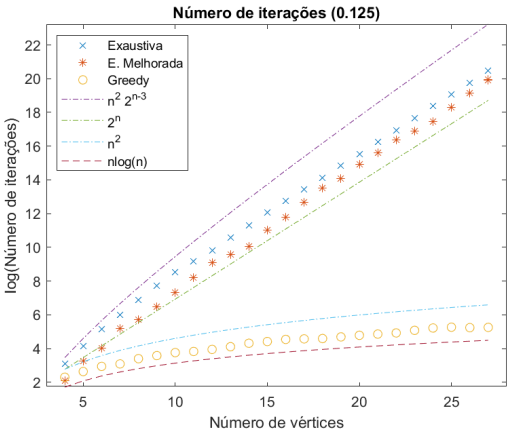


Fig. 3: Número de iterações quando $k = 0.125$ para as funções exaustiva, exaustiva melhorada e *Greedy*.

Para $k = 0.25$, Fig. 4, as conclusões a retirar são as mesmas que para quando $k = 0.125$ verificando-se apenas uma maior proximidade das duas funções exaustivas apresentadas, mostrando que esta não apresenta ser vantajosa para grafos mais completos. No que toca ao número de elementos obtidos pela função *Greedy*, neste caso, por sorte, coincidiram.

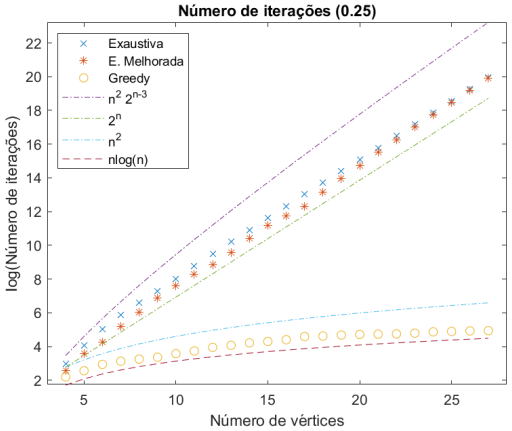


Fig. 4: Número de iterações quando $k = 0.25$ para as funções exaustiva, exaustiva melhorada e *Greedy*.

Quando $k = 0.5$, como se pode ver na Fig. 5, verifica-se que o número de iterações das duas funções exaustivas são quase coincidentes e muito próximas da linha de 2^n com a função *Greedy* seguindo quase exatamente a linha de $n\log(n)$. No entanto, nesta ocasião verificou-se mais uma vez que esta nem sempre apresenta o resultado correto, Fig. 13, em que para quando o grafo apresentava 17 e 18 vértices este conjunto apresentou menos um vértice.

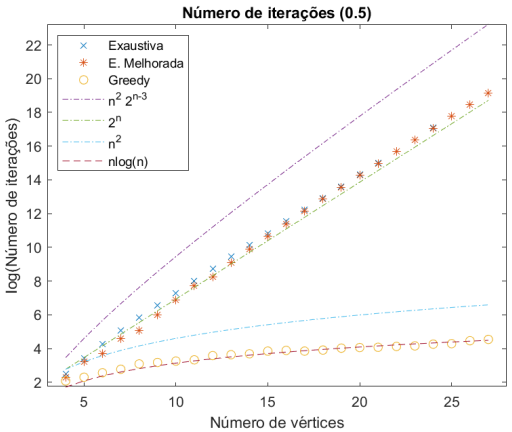


Fig. 5: Número de iterações quando $k = 0.5$ para as funções exaustiva, exaustiva melhorada e *Greedy*.

Na situação de $k = 0.75$, Fig. 6, verifica-se que as funções exaustivas seguem ambas aproximadamente a complexidade de 2^n e que a função *Greedy* aproxima-se a $nlog(n)$.

Atentando agora a Fig. 7, nesta figura são apresentados os casos especiais em que o grafo não tem arestas, $k = 0$, e completo, $k = 1$. Confirmando-se o pior caso para a função exaustiva quando $k = 0$, seguindo praticamente a linha de $n^2 2^{n-3}$. Confirmando-se a vantagem que a função exaustiva melhorada pode apresentar

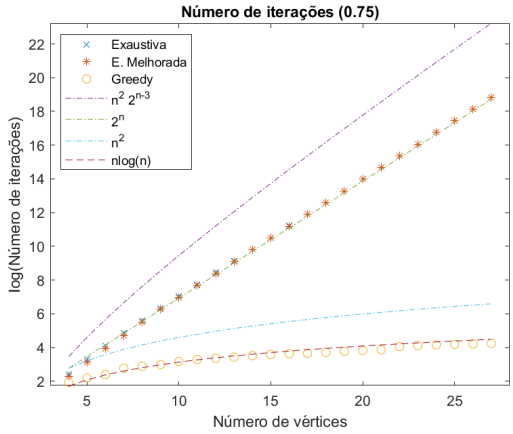


Fig. 6: Número de iterações quando $k = 0.75$ para as funções exaustiva, exaustiva melhorada e *Greedy*.

num grafo sem arestas. Apesar de rara a ocasião onde esta possa ser benéfica, não se verifica desvantagem no que toca ao número de iterações quando o grafo é completo ou com um maior número de arestas, pelo que se verificou nestes casos.

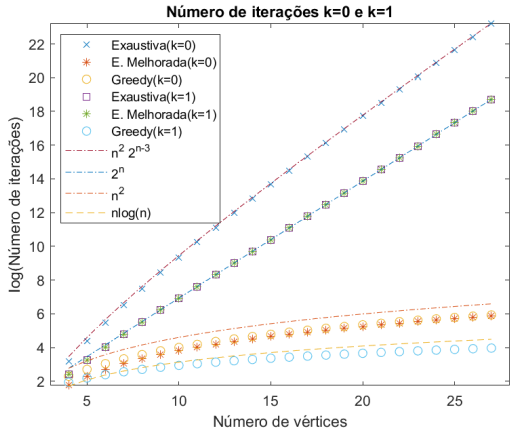


Fig. 7: Número de iterações quando $k = 0$ e quando $k = 1$ para as funções exaustiva, exaustiva melhorada e *Greedy*.

Dos testes aqui realizados verifica-se que as complexidades formalmente apresentadas raramente foram visualizadas. Isto devido à programação utilizada, nomeadamente, ao verificar que um conjunto de combinações não era possível este este era acabado de estudar aí não levando ao aumento do número de iterações que iria levantar a curva a se aproximar à linha de $n^2 2^{n-3}$.

Com `exhaustive_no_lock(G)` foi efetuado o estudo caso não fossem efetuadas essas alterações e nota-se que a linha não é ultrapassada, Fig. 8, onde o número de iterações é sempre o mesmo para cada número de vértices independentemente do valor de k, percentagem de arestas para ser um grafo completo.

Ao longo de toda esta análise só foram consideradas como operações básicas o número de iterações, não contabilizando as comparações e outras instruções de mais alto nível.

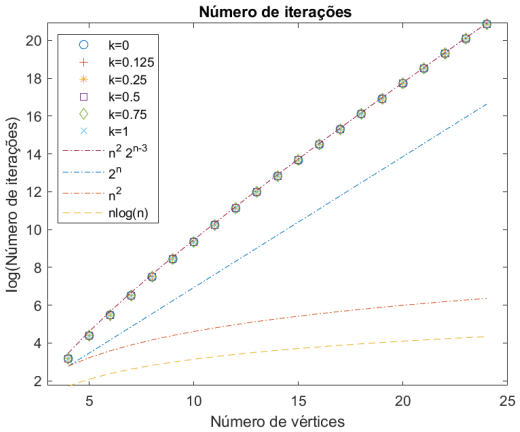


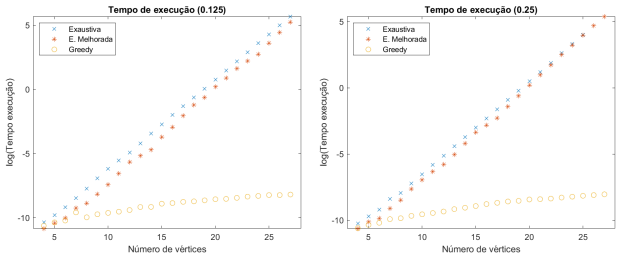
Fig. 8: Número de iterações para o método que efetua exaustivamente todas as combinações mesmo após verificar que esta não seria possível, para diferentes números de k.

IV. Tempo de execução

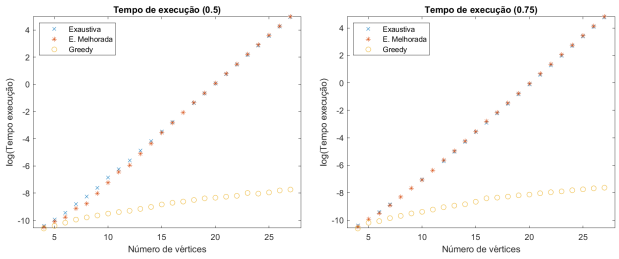
Pode-se também ver a escalabilidade do problema apresentado através do seu tempo de computação. Para isto foi temporizado o tempo de execução de cada um dos métodos. Estes dados foram obtidos num processador AMD Ryzen™ 7 3700X de frequência base de 3.60 GHz com *boost* ativo com 16 GB de memória RAM a 3333 MHz. Na versão 21H2 do Microsoft Windows 11 Pro na versão 3.10.5 de *Python*.

Na Fig. 9 é possível visualizar o tempo que cada uma das funções necessitou, verificando-se que a quanto mais completo o grafo fosse mais as funções exaustivas se aproximavam. Assemelhando-se ás tendências apresentadas pelo número de iterações. Notando-se que por vezes a função melhorada apresenta um tempo superior devido a um maior número de comparações a efetuar. Sendo explicito o grande crescimento temporal que os métodos exaustivos apresentam relativamente ao método heurístico.

Na Fig. 10 é possível comparar os tempos para as situações de grafos sem arestas, $k = 0$, e completos, $k = 1$, verificando-se o que se esperava pelo número de iterações. Seguindo o tempo de execução a mesma tendência que o número de iterações, Fig. 8, à exceção do método exaustivo melhorado. Como este continua a ter que gerar as combinações o processo demora bastante tempo, algo que não se verificaria com `exhaustive_the_fastest(G)`. As imagens `iteracoes_the_fastest.png` e `time_the_fastest.png` incluídas na pasta do trabalho contém as iterações e os tempos, respetivamente, da substituição da função `exhaustive_faster(G)` por `exhaustive_the_fastest(G)` como se esta fosse a melhorada. Mostrando que `exhaustive_the_fastest(G)` é uma melhor alternativa a `exhaustive_faster(G)`



(a) $k = 0.125$ (b) $k = 0.25$



(c) $k = 0.5$ (d) $k = 0.75$

Fig. 9: Figura com os gráficos dos tempos de execução para os k de 0.125, 0.25, 0.5 e 0.75 para as funções exaustiva, exaustiva melhorada e *Greedy*.

mesmo com o aumento de número de comparações que acaba por ser mitigado pela necessidade de gerar combinações desnecessariamente.

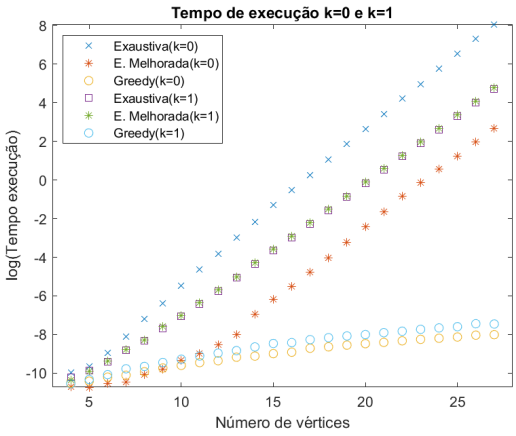


Fig. 10: Tempo de execução quando $k = 0$ e quando $k = 1$ para as funções exaustiva, exaustiva melhorada e *Greedy*.

Analisando o tempo de execução para a função exaustiva sem qualquer condição de paragem, obtém-se o gráfico presente na Fig. 11 que se reflete sempre como o pior caso para a função exaustiva. Seguindo a mesma tendência que o número de iterações para esta função.

Das Tabelas I e II é possível verificar os coeficientes que melhor ajustam as equações $an^2 2^{n-3}+b$ e an^2+b ás funções exaustivas e à função *Greedy*, respetivamente. Sendo possível extrapolar o tempo que seria necessário para grafos muito maiores.

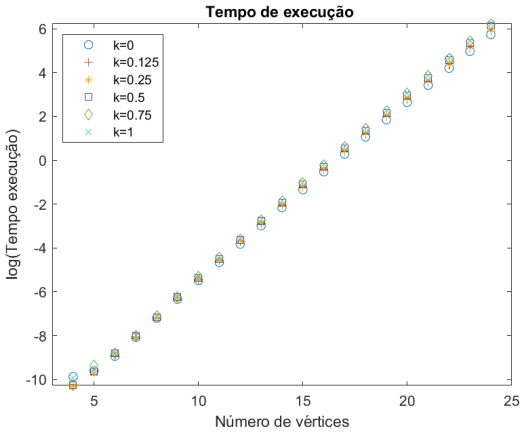


Fig. 11: Tempo de execução para o método que efetua exaustivamente todas as combinações mesmo após verificar que esta não seria possível, para diferentes números de k.

A título de exemplo, para um grafo com $k = 0.125$, para 36 vértices, a função exaustiva iria necessitar de 34 dias a melhorada 2 dias, e a completamente exaustiva 38 dias, não demorando um segundo obter a resposta com a função *Greedy*. Aumentando o número de vértices numa unidade, 37 vértices os tempos passariam a ser 70, 4 e 81 dias para as mesmas funções. Apesar dos valores serem extrapolados, efetuando previsões incorretas para previsões muito afastadas. Para situações maiores torna-se evidente que é impossível usar métodos exaustivos pois o seu tempo de resposta não é comportável, tornando-se a utilização de métodos aproximados a única atual possibilidade.

V. Número de elementos encontrados e de soluções possíveis

Uma vez que o *maximum set* não é único a forma exclusiva para averiguar se este foi encontrado é através do tamanho do maior conjunto de elementos a que se chegou.

Na Fig. 12 e na Fig. 13 é possível verificar que o método *Greedy* apresentou falhas, encontrando menos um elemento que as suas restantes alternativas, não tendo o mesmo se verificado para os outros k testados, não sendo apresentados devido a não apresentarem informações mais relevantes.

É no entanto possível verificar o número de soluções ao problema em causa através da Fig. 14. Verificando em todas as combinações que são um *independent set* quantas existem com o maior tamanho, implementado em `exhaustive_no_lock(G)`. Podendo-se verificar que em diversas situações existe mais que uma solução possível. Verifica-se como esperado que para o grafo sem arestas, $k = 0$, existe apenas sempre uma só solução, todos os vértices. E que para o grafo completo, $k = 1$, existem tantas como o número de vértices, que será sempre de só um elemento.

Adicionalmente, na Fig. 14 verifica-se que nos casos

		a	b	r^2
$k = 0$	Exaustiva	2.603e-07	0.1812	1
	E. Melhorada	1.216e-09	0.05583	0.9988
	Exaustiva sem paragem	2.567e-07	0.2003	1
$k = 0.125$	Exaustiva	2.437e-08	1.226	0.998
	E. Melhorada	1.517e-08	0.2311	0.999
	Exaustiva sem paragem	2.996e-07	0.09841	1
$k = 0.25$	Exaustiva	1.845e-08	0.4991	0.9976
	E. Melhorada	1.821e-08	0.3189	0.9988
	Exaustiva sem paragem	3.314e-07	-0.01823	1
$k = 0.5$	Exaustiva	1.15e-08	0.2689	0.9978
	E. Melhorada	1.221e-08	0.3782	0.998
	Exaustiva sem paragem	3.785e-07	-0.2089	1
$k = 0.75$	Exaustiva	9.773e-09	0.5262	0.9979
	E. Melhorada	1.049e-08	0.5479	0.9979
	Exaustiva sem paragem	4.108e-07	-0.1867	1
$k = 1$	Exaustiva	9.213e-09	0.1734	0.9977
	E. Melhorada	9.812e-09	0.6848	0.9979
	Exaustiva sem paragem	4.284e-07	-0.2141	1

TABELA I: Tabela com os coeficientes obtidos efetuando o *fit* do tempo demorado pelas funções exaustivas ao número de vértices do método exaustivo, exaustivo melhorada e do método que efetua exaustivamente todas as combinações mesmo após verificar que estas não seriam possíveis, para cada valor de k utilizando o método de *Levenberg-Marquardt*. Ajustando à equação $an^2 2^{n-3} + b$ com n a representar o número de vértices, e a e b como valores a encontrar. Sendo r^2 o valor ajustado de $R - Square$.

	a	b	r^2
$k = 0$	4.437e-07	2.431e-05	0.9962
$k = 0.125$	3.639e-07	3.488e-05	0.9793
$k = 0.25$	4.208e-07	3.166e-05	0.988
$k = 0.5$	5.658e-07	1.672e-05	0.9946
$k = 0.75$	6.572e-07	2.461e-05	0.994
$k = 0.1$	7.999e-07	1.624e-05	0.997

TABELA II: Tabela com os coeficientes obtidos efetuando o *fit* do tempo das funções ao número de vértices do método *Greedy* para cada valor de k utilizando o método de de *Levenberg-Marquardt*. Ajustando à equação $an^2 + b$ com n a representar o número de vértices, e a e b como valores a encontrar. Sendo r^2 o valor ajustado de $R - square$, indicativo da qualidade do ajuste.

em que a função *greedy* apresentou falhas, Fig. 12 e 13 o número de soluções nesse caso era 1.

VI. Conclusão

A procura do *maximum independent set* só se consegue efetuar, até à data, através de um método exaustivo. Método que apresenta uma complexidade computacional, formulada matematicamente e visualizada, de carácter exponencial o que se torna impraticável para grafos muito grandes. Apesar da possibilidade de paralelização poder fazer acelerar este processo, com

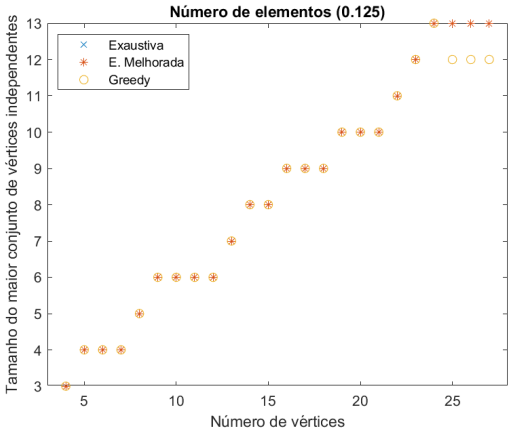


Fig. 12: Número de elementos do *maximum independent set* encontrados para cada método para diferentes números de vértices com $k = 0.125$

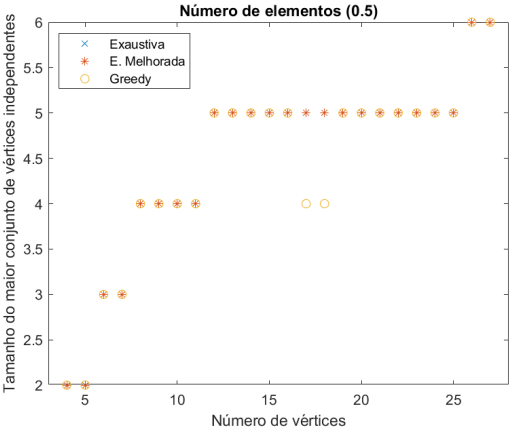


Fig. 13: Número de elementos do *maximum independent set* encontrados para cada método para diferentes números de vértices com $k = 0.5$

poucos elementos a necessitarem de ser partilhados, este continuará a ter uma complexidade exponencial. Mesmo métodos que guardem logo combinações que à partida serão impossíveis por já terem sido verificados, evitando analisar combinações maiores, irão requerer muita memória e não conseguiram facilitar o processo.

Não havendo assim melhor alternativa a uma redução da complexidade computacional pelo que será sempre necessário recorrer a técnicas que forneçam um resultado mesmo que aproximado mas rapidamente como foi visto ao longo deste trabalho que a resposta aproximada acertou na maioria das vezes. Podendo-se até utilizar diferentes métodos com abordagens distintas para aumentar as chances de ser a resposta certa.

Ainda assim existem casos cruciais que precisam de uma resposta garantida certa pelo que cada método será preferível consoante a aplicação. Efetuando a escolha entre rapidez e a garantia da exatidão da resposta obtida.

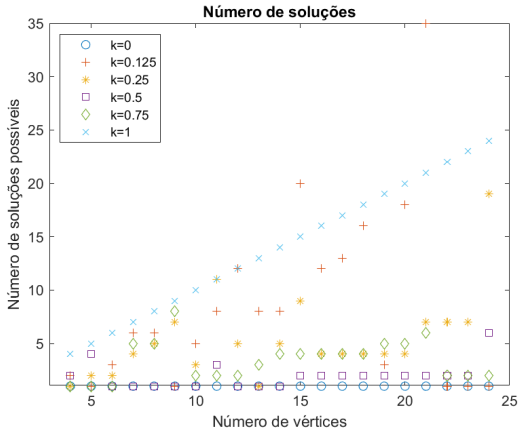


Fig. 14: Número de soluções possíveis para cada número de vértices para diferentes valores de k.

Bibliografia

[1] Maximum Independent Vertex Set, `https://mathworld.wolfram.com/MaximumIndependentVertexSet.html`, 19-10-2022

[2] Garey, M. & Johnson, D. " Strong " NP-Completeness Results: Motivation, Examples, and Implications. *J. ACM.* **25**, 499-508 (1978,7), https://doi.org/10.1145/322077.322090

[3] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

[4] Batagelj, V. & Brandes, U. FAST_GNP_RANDOM_GRAPH#. *Fast_gnp_random_graph - NetworkX 2.8.8 Documentation.*, `https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.fast_gnp_random_graph.html`

[5] Networkx Maximal_independent_set. *Maximal_independent_set - NetworkX 2.8.7 Documentation.*, `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.mis.maximal_independent_set.html#networkx.algorithms.mis.maximal_independent_set`

[6] Networkx Maximum_independent_set. *Maximum_independent_set - NetworkX 2.8.7 Documentation.*, `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.clique.maximum_independent_set.html`

[7] Wikiwand Maximal independent set. *Wikiwand.*, `https://www.wikiwand.com/en/Maximal_independent_set`

[8] Ballard-Myer, J. Ballard Myer - georgia college & state university. *Gcsu.edu.* (2019,12), `https://www.gcsu.edu/sites/files/page-assets/node-808/attachments/ballardmyer.pdf`