Maior conjunto de vértices independentes em grafos
Projeto 2

Vasco Vieira Costa - 97746

***Abstract* –Continuation on the search for the maximum independent set, taking a randomized approach instead of an exhaustive one as it was done in the previous project. By generating random combinations and analising different ways to choose how many of each size should be tested. In addition to a modification to the heuristic method developed before. Some of the results obtained are satisfactory and reflect that randomized methods can be used in practice when the error can be limited.**

***Resumo* –Continuação da procura pelo *maximum independent set* tomando uma abordagem aleatória face a uma técnica exaustiva como no projeto anterior. Gerando combinações de forma aleatória e analisando diferentes formas de ver quantas de cada tamanho se concretizam. Incluindo ainda uma modificação à função heurística desenvolvida anteriormente. Os resultados obtidos apresentam-se alguns como sendo satisfatórios e refletem que métodos aleatórios poderão ser utilizados em algumas situações na prática em que o erro cometido seja possível de limitar.**

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

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

I. Introdução

Com este projeto pretende-se efetuar a pesquisa do *maximum independent set*[1], o maior subconjunto de vértices de um grafo em que nenhum par de dois vértices está ligado por uma aresta [1]. Ao contrário do projeto anterior em que se pretendia analisar de uma forma exaustiva desta vez a abordagem de utilizar métodos aleatórios é tomada.

Um algoritmo aleatório é aquele que utiliza números aleatórios para efetuar escolhas no seu decorrer, não garantido que os mesmos dados de entrada levem sempre a um resultado igual. Estes métodos ganharam popularidade devido a normalmente serem mais rápidos e apresentarem uma maior simplicidade face ás suas alternativas [2]. Um dos métodos mais utilizados é o de Monte Carlo em que se geram objetos aleatórios para processar e observar o seu comportamento, estimando quantidades numéricas através de amostragem repetida ou para resolver problemas de otimização através de algoritmos com métodos aleatórios [3].

Na área do desenvolvimento de redes de informação, para a criação de ambientes de simulação, métodos aleatórios em combinação com grafos mostram-se como sendo de alta relevância [4].

[1]Problema 12.

II. Algoritmia

Pelo enunciado é sugerida a criação das combinações de forma aleatória e sem que ocorram repetições. Poder-se-ia assim efetuar o mesmo que se efetuou no projeto anterior de ir armazenando uma lista de todas as combinações de diferentes tamanhos, (algo que se verifica agora que foi desnecessário e feito de forma menos correta), e escolher ao acaso as que seriam para se concretizar. Contudo, esta abordagem leva a um uso de memória bastante grande e desnecessário.

Assim a alternativa passa por selecionar ao acaso vértices, não os repetindo na mesma combinação até ao tamanho desejado do conjunto a querer analisar e verificando se esta já fora analisada, guardando-se esta em memória pois foi analisada tendo em atenção que a ordem não importa.

A. Método aleatório de criação de combinações

Um método aleatório desenvolvido é o apresentado na função `random_method_generate_combination`. Começando por verificar as combinações de maior para as de menor tamanho por se garantir que ao encontrar uma de maior tamanho se pode parar aí, como se concluiu do trabalho anterior.

Dado que para cada número de n vértices é possível saber quantas combinações de k vértices existem, que serão $\frac{n!}{k!(n-k)!}$, correspondendo a cada elemento da n-ésima linha do triângulo de Pascal. Podendo-se assim analisar uma percentagem de cada número de combinações e esperar que se encontre a solução certa.

Analisar uma percentagem, p, poderá ser preferível face a um número fixo pois consoante o valor de k o número de combinações difere. Obrigando a que se fizessem mais repetições do que o número de combinações existente ou então seriam pouco representativas para o número possível de combinações. Favorecendo assim o número de combinações com mais e menos elementos face aos centrais, como se pode verificar analisando uma das linhas da Fig. 1.

Supondo que se pretendia analisar um grafo com 7 vértices, a última linha da figura, escolhendo um número fixo de 8 tentativas para cada k, por exemplo faria com que se repetissem os conjuntos de 0, 1, 6 e 7 elementos mais vezes que as combinações existentes. Efetuando-se uma menor percentagem dos valores possíveis para os restantes, com um menor número de tentativas para k igual a 3 que a 2, favorecendo-se assim certos k.

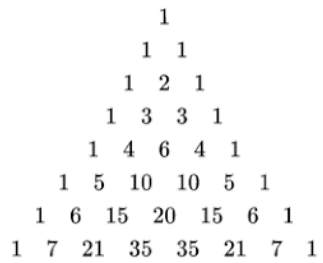


Fig. 1: Primeiras 7 linhas do Triângulo de Pascal, imagem de [5]

Para criar a combinação pode-se tomar a abordagem mencionada anteriormente da seleção de vértices até se obter o tamanho procurado, k, sendo necessário fazer isto para os diferentes tamanhos, de n até 1 só vértice. Gerando uma combinação diferente caso essa já tenha sido analisada.

Sendo uma combinação possível e ainda não vista, uma solução candidata, efetua-se a verificação como no projeto anterior de se este é um conjunto independente. Verificando se existe um qualquer par de vizinhos, i e j que esteja ligado por uma aresta. Caso não exista nenhum par não é necessário prosseguir para mais qualquer análise pois os candidatos seguintes serão apenas do mesmo tamanho ou mais pequenos, não indo no caminho de uma solução mais correta.

Na Equação 1 pode-se verificar como o processo é concretizado sem contar com o processo de gerar a combinação que poderá necessitar de bastantes tentativas consoante o valor de p, uma maior percentagem de valores a analisar leva a serem necessários guardar mais conjuntos e assim ao aumento da probabilidade de gerar uma combinação já vista. Assim a complexidade relativa ao número de iterações pode ser considerada como sendo $\mathcal{O}(2^{n-3}n^2p)$, p pode colocar-se a depender de n, sendo assim uma complexidade exponencial de $\mathcal{O}(2^n n^2 p)$.

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

A.1 Variações a 'p' e os efeitos no número de iterações

Utilizando um p entre 0 e 1 irá-se obter uma complexidade exponencial igual ao método apresentado no projeto anterior, isto por p, neste caso, ser escondido pela notação $\mathcal{O}$.

Tendo isto em conta pode-se então tomar outra abordagem e limitar o número de combinações, representando-se o número de iterações pela Equação 2, estabelecendo-se assim um *limit* em que se efetua ou o número de combinações possíveis caso seja menor que o valor limite ou então efetua-se esse número, função `random_method_generate_combination_limit`.
Também seria possível combinar esta lógica com o método percentual mas tal não foi analisado.

$$\sum_{k=1}^{n}\sum_{l=1}^{\min\left(\binom{n}{k},limit\right)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} 1 \tag{2}$$

Transformando p em $\frac{1}{2^k}$, Equação 3, obtém-se uma complexidade inferior, o resultado pode ser desenvolvido para $2^{-n}2^{-1}3^n3^{-2}(n-1)n$ que retrata-se numa complexidade de $\mathcal{O}(\frac{3}{2}^n n^2)$. Sendo assim uma complexidade inferior ao método exaustivo apresentado no projeto anterior e até ao apresentado anteriormente. No entanto, é à mesma exponencial, função `random_method_generate_combination_2k`.

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

Alterando p para $\frac{1}{2^n}$, Equação 4, chega-se a uma complexidade de $\mathcal{O}(n^2)$. Sendo assim um método polinomial para chegar a um resultado aproximado que será bastante mais rápido que os vistos até agora. Contudo não seriam feitas quaisquer análises pois o número de combinações de cada tamanho a analisar seria 0, tomando o *ceiling*, por vezes poderá-se obter uma unidade mas tal não é retratado na equação.

$$\sum_{k=1}^{n}\sum_{l=1}^{\frac{1}{2^n}\binom{n}{k}}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} 1 = \frac{1}{8}(n-1)n \tag{4}$$

Assim a forma mais aproximada e útil será a de se realizar só uma combinação de cada tamanho, retratado na equação 5, através da função `random_method_generate_combination_2n` que será apresentada como sendo $p = \frac{1}{2^n}$ para o restante do trabalho. Sendo assim uma complexidade computacional de $\mathcal{O}(n^3)$, ou seja, polinomial.

$$\sum_{k=1}^{n}\sum_{l=1}^{1}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} 1 = \frac{1}{6}(n-1)n(n+1) \tag{5}$$

Na Tabela I é apresentado um exemplo do número de combinações diferentes a serem analisadas para um caso de um grafo com 13 vértices. Podendo-se daqui verificar que o método mais rápido, de ordem cúbica, irá efetuar apenas uma combinação de cada tamanho pelo que as hipóteses de obter a solução correta é bastante baixa. Verificando-se que ao seguir-se a probabilidade se vai de encontro ao que foi dito anteriormente assim como quando limitado, verificando-se que quando $p = \frac{1}{2^k}$ o número de combinações testadas com menos elementos aumenta.

TABELA I: Número de combinações analisadas de tamanho k para 13 vértices, $n = 13$, para os diferentes valores de p. Não sendo todos os valores apresentados devido a ter-se encontrado a solução.

k	Número de Combinações	$p = 0.1$	limit = 1000	$p = \frac{1}{2^k}$	$p = \frac{1}{2^n}$
13	1	1	1	1	1
12	13	2	13	1	1
11	78	8	78	1	1
10	286	29	286	1	1
9	715	72	715	2	1
8	1287	129	1000	6	1
7	1716	172	1000	14	1
6	1716	172	-	27	1
5	1287	-	-	41	1
...	...	...	...	...	...
1	13	-	-	-	1

B. Random Greedy

Partindo do método *Greedy* desenvolvido no **Projeto 1** mas não tomando o seu princípio de remoção do vértice com menor número de vizinhos mas sim o da repetição de seleção de *maximal independent sets* várias vezes. Um *maximal independent set* é um conjunto de vértices ou arestas, neste caso, vértices independentes, que não permite a adição de qualquer outro vértice. Note-se que um *maximal set* não implica que seja um *maximum set* mas o oposto verifica-se sempre [6].

Escolhendo primeiramente à sorte um dos i dos n vértices do grafo para incluir na lista de vértices e removendo os seus vizinhos, v, e a si mesmo repetindo-se até não existirem mais vértices. Analisando a cada tentativa se o tamanho obtido é superior, caso o seja então será a melhor solução até agora.

O número de vezes que se repete este processo, p, para configurações diferentes pode-se fazer variar da forma que se entender. Quantas mais vezes se efetuar o processo mais conjuntos se verificam e menor acaba por ser a probabilidade de não encontrar a solução pretendida à custa de um maior tempo computacional.

Assim os passos para este método poderão ser representado por $p \cdot f(n)$, Equação 6, função `greedy_random`. A sua complexidade não se torna possível de desenvolver devido ao número variado de vizinhos que cada vértice apresenta sendo que as mesmas combinações de vértices podem ser vistas diversas vezes. Apesar da ordem de escolha dos vértices poder ser diferente esta não se torna relevante, visto que a ordem não importa. Inclusive, não são guardadas as combinações já visualizadas por não fazer sentido uma vez que ao chegar-se ao fim já se obteve a resposta. O número de combinações possíveis também irá diferir consoante o quão completo o grafo se encontre pelo que aguardar que um certo número de combinações diferentes seja visto poderá não ser o mais viável.

Assim sendo, uma possível alternativa será atribuir um tempo limite para a repetição do processo, sendo que se irá sempre obter um resultado, podendo não ser o certo. Levando a que o número de combinações analisadas vá depender do computador onde é executado, sendo possível que ao não existirem mais combinações possíveis se desperdice mais tempo que o que seria necessário por outros métodos. Concluí-se assim que este algoritmo é transformado numa complexidade de $\mathcal{O}(1)$.

$$f(n) = \begin{cases} 0 & , n = 0 \\ 1 + v + f(n - 1 - v) & , n \neq 0 \end{cases} \quad (6)$$

Foi decidido para a análise que se segue que o tempo limite seria de 3 segundos, o valor *limit* seria de 1000 e que p quando considerado como um número entre 0 e 1 que seria de 0.15.

III. Iterações

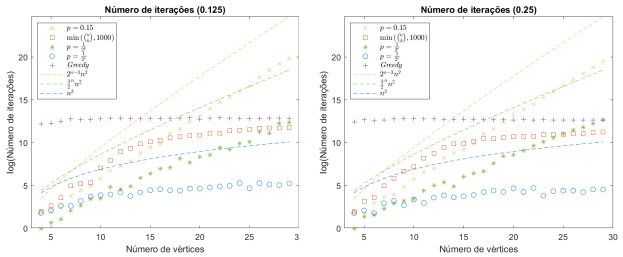
Na Fig. 2 encontram-se representados o número de iterações que foram necessárias para encontrar uma solução. Verifica-se que todos os r levam sempre ás mesmas tendências à exceção do caso em que $r = 0$ em que aqui todos os métodos seguem aproximadamente a linha de n^2. Para a função *Greedy* visualiza-se que no geral existe uma diminuição do número de iterações em função do aumento do número de vértices, o que seria de esperar uma vez que ao ser um grafo maior, mais complexo, o tempo de computação é maior pelo que menos tentativas serão efetuadas.

Os restantes métodos seguem todos uma tendência crescente como era proposto pelas complexidades computacionais apresentadas anteriormente. Sendo claro para o método em que $p = 0.15$ que este cresce exponencialmente, que o método em que $p = \frac{1}{2^k}$ segue a mesma curva que $\frac{3}{2}^n n^2$ mas bastante afastado verticalmente. O que se apresenta limitado começa inicialmente por requerer mais iterações que o que apresenta $p = 0.15$ mas existe sempre um número de vértices em que estes trocam e o método limitado torna-se quase constante mas sempre um pouco crescente devido ao aumento de n, resultante em mais tamanhos de combinações para analisar, número de itens na linha do triângulo de Pascal.

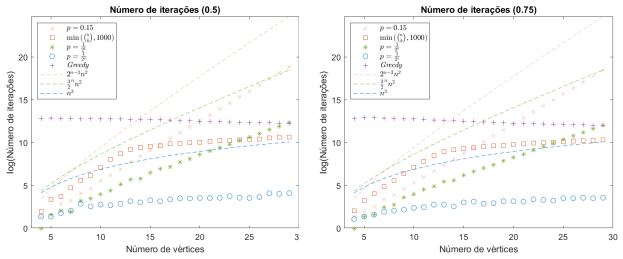
A. Contabilização mais detalhada

Verificar apenas o número de iterações que o algoritmo requer poderá não ser suficiente visto que gerar as próprias combinações apresenta um custo. Na Fig. 3 é apresentada a soma do número de iterações com o número de tentativas necessárias a gerar as combinações, efetuando-se assim uma maior aproximação ao número de operações básicas concretizadas.

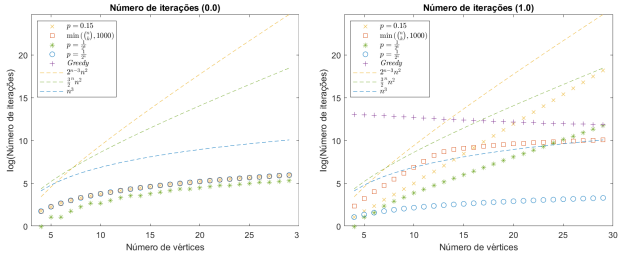
As tendências aqui verificadas seguem a mesma tendência que o número de iterações, analisado na secção anterior. Sendo mais notória a diminuição do número de operações para a função *Greedy* com o aumento do número de vértices. Assim como a variação para quase constante com a função limitada. Apresentando mais passos até ser limitada devido à maior dificuldade que se torna em gerar combinações não vistas por terem que se fazer todas as possíveis até ser limitada. A situação de $r = 0$ apresenta-se como sendo



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

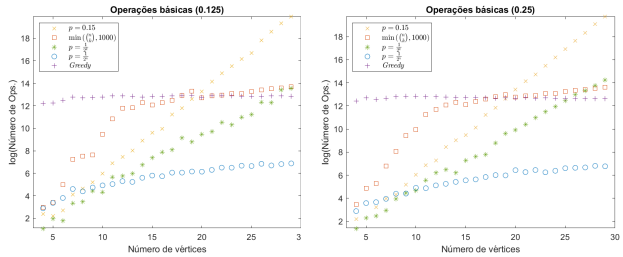


(c) $r = 0.5$ (d) $r = 0.75$

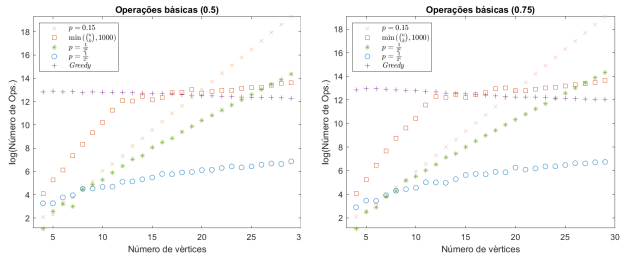


(e) $r = 0.0$ (f) $r = 1.0$

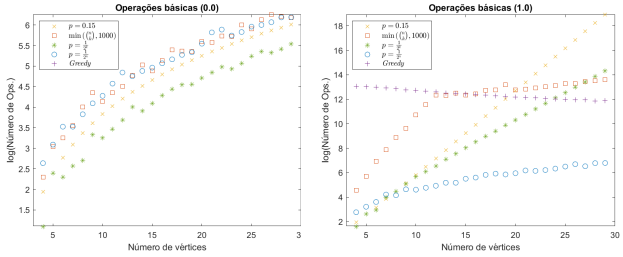
Fig. 2: Figura com os gráficos do número de iterações necessárias. Para diferentes r de probabilidade de ser um grafo completo em diferentes valores de p, com o método limitado e com a função *Greedy*.



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



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



(e) $r = 0.0$ (f) $r = 1.0$

Fig. 3: Figura com os gráficos do número de iterações necessárias com o número de vezes que foi necessário selecionar um vértice para criar as combinações a analisar. Para diferentes r de probabilidade de ser um grafo completo em diferentes valores de p, com o método limitado e com a função *Greedy*.

diferente de todos os restantes e com uma escala vertical mais reduzida.

IV. Tempo de Computação

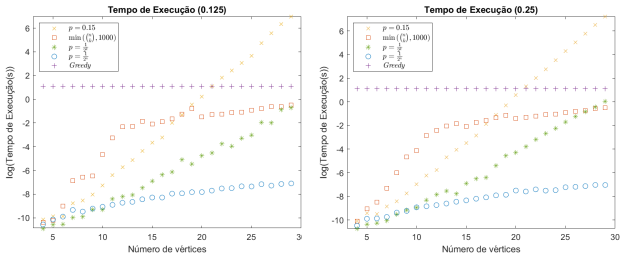
Estes dados foram obtidos num processador AMD Ryzen™ 7 3700X de frequência base de 3.60 GHz com *boost* ativo com 16 GB de memória RAM a 3333 MHz. Na versão 22H2 do Microsoft Windows 11 Pro na versão 3.10.5 de *Python*.

Da Fig. 4 pode-se verificar o tempo de execução que os diferentes métodos necessitaram. Verificando-se que aqui para a função *Greedy* o tempo é sempre constante como se esperaria. Para $r = 0$ constata-se que este método é desvantajoso em termos computacionais pois outros métodos apresentaram um tempo de execução muito menor. Não sendo estes nada indicativos da exatidão da resposta obtida.

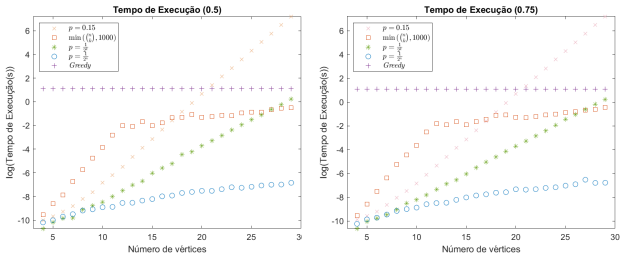
Apura-se também que existem alguns picos para o método limitado que podem ser explicados por diferentes motivos, desde alguns não controláveis como operações do sistema operativo ou devido a fatores aleatórios afetos ao próprio algoritmo. Com os restantes a transparecerem a mesma informação possível de retirar dos gráficos do número de iterações, Fig. 2.

Isto leva a que se possa efetuar a estimativa do tempo que seria necessário para grafos muito maiores com o que aqui foi desenvolvido e assim, efetuando as médias dos tempos para cada um dos valores de $r = 0.125$, $r = 0.25$, $r = 0.5$, $r = 0.75$ e $r = 1.0$ para cada número de vértices, obtém-se a Fig. 5. O valor de $r = 0$ não foi incluído devido a este ser o melhor caso e não ser significativo para um uso real. Mais situações não foram testadas devido a já se verificar um tempo de computação elevado.

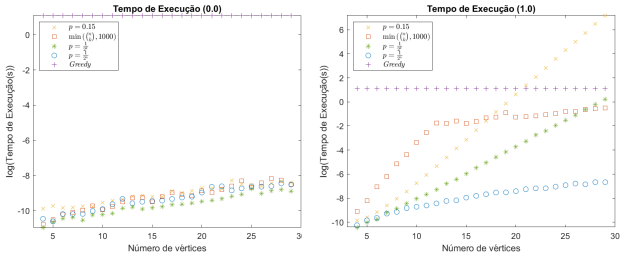
Assim, efetuando as estimativas pode-se primeiramente afirmar que a função *Greedy* irá sempre fazer uso o tempo limite utilizado, podendo ser no entanto mais tempo requerido para efetuar a leitura do grafo para a memória que não é contabilizado. Na situação limitada verifica-se que ao chegar-se a um número de vértices igual a 12 este segue outra tendência e por isso para estimar os valores apenas se pode esperar que se



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



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



(e) $r = 0.0$ (f) $r = 1.0$

Fig. 4: Figura com os gráficos do tempo de computação. Para diferentes r de probabilidade de ser um grafo completo em diferentes valores de p, com o método limitado e com a função *Greedy*.

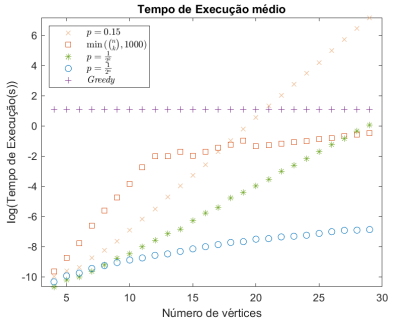


Fig. 5: Figura com a média dos gráficos do tempo em cada um dos vértices para os valores de $r = 0.125$, $r = 0.25$, $r = 0.5$, $r = 0.75$ e $r = 1.0$.

siga esta nova tendência. Fazendo-se a aproximação a $a \cdot e^{bn}$ com a e b constantes a determinar.

Para a situação de $p = 0.15$, fazendo a aproximação a $a \cdot 2^n n^2 + b$ em que a e b são parâmetros livres. Para $p = \frac{1}{2^k}$ fazendo-se a $a \cdot \frac{3}{2}^n n^2 + b$ com a e b parâmetros livres e para $p = \frac{1}{2^n}$ a $a \cdot n^2 + b$, com a e b parâmetros livres[2]. Chegando-se aos valores apresentados na Tabela II com o erro quadrado ajustado, R^2, através do método de Levenberg-Marquardt.

TABELA II: Tabela com os coeficientes obtidos e com o valor ajustado de R^2, indicativo de um bom ajuste, utilizando o algoritmo de Levenberg-Marquardt.

	$p = 0.15$	$\min\left(\binom{n}{k}, 1000\right)$	$p = \frac{1}{2^k}$	$p = \frac{1}{2^n}$
a	2.91e-09	0.05081	1.01e-08	1.272e-06
b	0.5664	0.08506	0.005418	1.128e-05
R^2	0.9992	0.9335	0.998	0.9966

Efetuando a extrapolação para outros grafos com n vértices obtém-se os valores obtidos na Tabela III. Verificando-se que ainda assim os métodos aqui desenvolvidos apresentam um tempo de computação bastante grande.

TABELA III: Tabela com os tempos extrapolados que seriam necessários para estudar grafos com um dado número de vértices.

Grafo	$p = 0.15$(s)	$\min\left(\binom{n}{k}, 1000\right)$(s)	$p = \frac{1}{2^k}$(s)	$p = \frac{1}{2^n}$(s)	*Greedy* (s)
50	8,19E+09	3,57E+00	1,61E+04	3,19E-03	3
100	3,69E+25	2,51E+02	4,11E+13	1,27E-02	3
500	2,38E+147	1,50E+17	2,80E+85	3,18E-01	3
1000	3,12E+298	4,44E+35	1,25E+174	1,27E+00	3

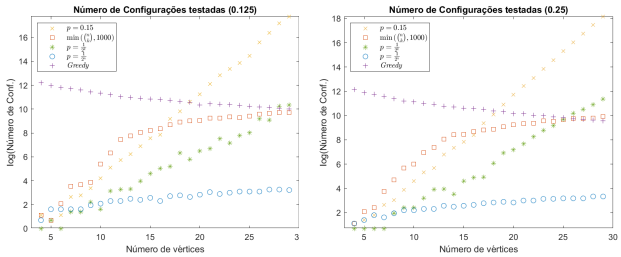
V. Resultados obtidos

Primeiramente é possível verificar o número de configurações testadas, ou seja, de soluções candidatas que são verificadas como é apresentado na Fig. 6. Em que no método *Greedy* são contabilizadas todas as tentativas pois esta foi computada face aos restantes métodos que apenas apresentam as configurações diferentes, descartando as geradas que podem ter sido repetidas, não tendo as mesmas sido analisadas.

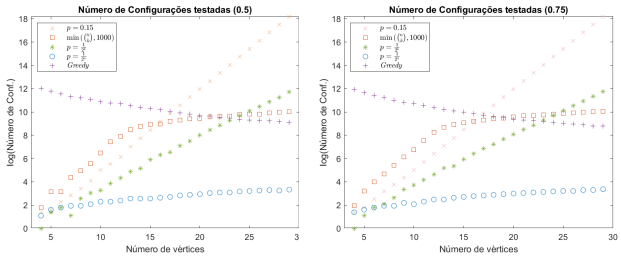
Sendo mais uma vez tendência que se verificou com o número de iterações a ser seguida com um decréscimo acentuado na função *Greedy* para o aumento do número de vértices. Com uma menor dispersão face à tendência visível para grafos mais completos, r superior.

Na Fig. 7 são apresentados o número de elementos encontrados por cada um dos métodos aqui testados. Verificando-se que o método de $p = \frac{1}{2^n}$ apresenta por norma um resultado bastante mau. Com todos os outros métodos apresentando sempre falhas, alguns na maioria das vezes uma diferença de um elemento que pode ser considerado como uma boa aproximação, que é o caso do método $p = 0.15$ ou do limitado. Para a alternativa *Greedy* esta conseguiu sempre encontrar a solução correta uma vez que o tempo dado foi grande o suficiente.

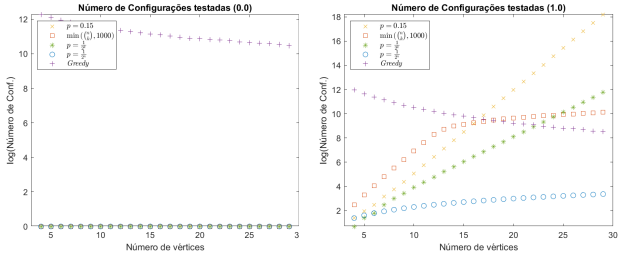
[2]Seria de esperar que a ordem cúbica fosse ser melhor mas na prática, fazendo $a \cdot n^3 + b$ resultou num R^2 de 0.6781. Pelo que se utilizou o modelo quadrado como se tinha verificado pelo número de iterações que seria preferível.



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



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



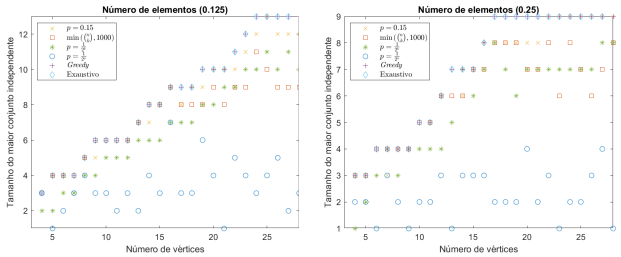
(e) $r = 0.0$ (f) $r = 1.0$

Fig. 6: Figura com os gráficos do número de soluções candidatas testadas. Para diferentes r de probabilidade de ser um grafo completo em diferentes valores de p, com o método limitado e com a função *Greedy*.

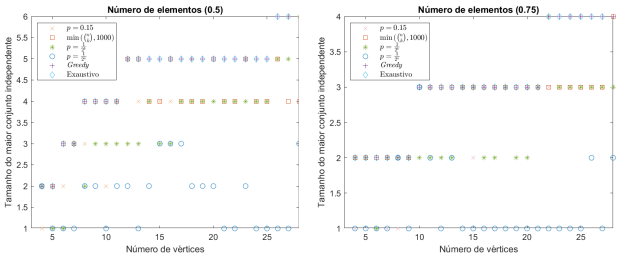
Para $r = 1$ como a solução exata foi sempre encontrada os mesmos não são apresentados. Sendo que apenas são apresentados os resultados exatos para comparação até aos 27 vértices, a partir daí apenas se estimam os resultados obtidos pelo que não são apresentados. Na pasta também se encontra a figura para $r = 0$ em que o método $p = \frac{1}{2^k}$ nunca encontrou a solução correta pelo que poderá ser um defeito do algoritmo em si que ao ter p desta maneira não permite que este seja o máximo. Dado ser um caso muito raro e não significativo para o uso real não foi incluído para não dar ponderação a algo que se torna meramente académico.

Uma forma de explorar em maior detalhe é através da informação ilustrada na Tabela IV. Onde se apresenta o número de vezes em que as funções levaram a um resultado correto, ou seja, igual ao que foi obtido exaustivamente.

Verificando-se mais claramente que o método *Greedy* funcionou em todas as situações e que dos restantes o limitado é o que apresenta um melhor desempenho. Com $p = \frac{1}{2^n}$ a ser o pior método seguindo-se $\frac{1}{2^k}$ e de $p = 0.15$. A tendência decrescente para este método,



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



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

Fig. 7: Figura com os gráficos do número máximo de elementos. Para diferentes r de probabilidade de ser um grafo completo em diferentes valores de p, com o método limitado e com a função *Greedy*. Os valores exaustivos foram apenas obtidos até ao valor de 27 vértices.

$p = 0.15$, com o aumento de r pode ser explicado pelo pouco número de combinações de tamanho certo que são testadas uma vez que o tamanho do *maximum independent set* segue uma tendência decrescente com o aumento de r, para os grafos aqui testados.

TABELA IV: Tabela com a contabilização do número de vezes que nos 24 testes para cada valor de r a função obteve um valor igual ao obtido pelo método exaustivo do projeto anterior assim como a sua percentagem.

	$r = 0.125$		$r = 0.25$		$r = 0.5$		$r = 0.75$		Geral	
$p = 0.15$	12	50%	12	50%	8	33.33%	7	29,12%	39	40.63%
$\min\left(\binom{n}{k}, 1000\right)$	14	58.33%	12	50%	14	58.33%	19	79,12%	59	61.46%
$p = \frac{1}{2^k}$	0	0%	2	8.33%	3	12.5%	6	25%	11	11.46%
$p = \frac{1}{2^n}$	2	8.33%	2	8.33%	0	0%	1	4.17%	5	5.21%
Greedy	24	100%	24	100%	24	100%	24	100%	96	100%

Dado que por vezes se verifica que o resultado obtido é próximo, podendo ser bom o suficiente para a sua utilização de forma prática. Pode-se analisar a média do erro relativo, efetuando $\frac{100}{24}\sum_{i=1}^{24}\frac{n_{exaustiva}-n_{metodo}}{n_{exaustiva}}$ para cada um dos diferentes métodos nos diferentes r, obtendo-se assim os valores presentes na Tabela V.

Daqui verifica-se que o método *Greedy* como obteve sempre a resposta certa o seu erro é de 0% seguindo-se a tendência apresentada anterior em termos de pior escolha de p, chegando-se a um erro médio de abaixo de 10% para o método limitado. Contudo, esta forma de medir o erro apresentado poderá não ser visto como sendo a desejada. Dependendo do caso em que se pretende utilizar uma vez que a diferença de um vértice

TABELA V: Tabela com os valores médios do erro relativo entre os diferentes métodos e a solução exaustiva para diferentes valores de r.

	$r = 0.125$	$r = 0.25$	$r = 0.5$	$r = 0.75$	Média
$p = 0.15$	6.48%	11.90%	16.11%	26.04%	15.13%
$\min\left(\binom{n}{k}, 1000\right)$	7.37%	9.45%	8.75%	5.21%	7.70%
$p = \frac{1}{2^k}$	20.78%	23.15%	24.72%	25.69%	23.59%
$p = \frac{1}{2^n}$	54.64%	56.61%	65.83%	58.68%	59.69%
Greedy	0.00%	0.00%	0.00%	0.00%	0.00%

no resultado obtido apresenta um peso diferente consoante o tamanho da solução correta.

VI. *Benchmarks*

Para analisar melhor alguns dos métodos aqui desenvolvidos uma forma de o fazer é através de *benchmarks*, ou seja, com grafos modelo que podem conter algumas particularidades como a presença de laços. Para isso foram analisados os grafos disponibilizados no eLearning e alguns dos presentes em [7], nomeadamente os relativos a dados sobre a rede social `Facebook` e do serviço de *streaming* `Deezer` em `benchmarks.py`. Dos presentes no eLearning não foi possível testar o maior devido ás limitações verificadas na leitura do grafo para memória e mesmo com a introdução dos dados na estrutura de dados de `networkx` [8], não tendo sido possível analisar-se o mesmo.

Os resultados destas experiências encontram-se registados na Tabela VI. Aqui alguns dos métodos não conseguiram concretizar uma solução em tempo útil pelo que não se encontram registados. Analisando os resultados apenas se pode especular em relação a se estes estão corretos uma vez que não são apresentados métodos exatos com os quais seria possível comparar. Ainda assim, é possível afirmar que quanto maior o número mais perto este estará da resposta certa. Verificando-se que nesta análise o método *Greedy* do projeto anterior apresenta melhores resultados seguindo-se da *Greedy* aleatória e com os restantes a seguirem a tendência já verificada.

TABELA VI: Tabela com o número de elementos do *maximum independent set* e o tempo de execução necessário para diferentes métodos em que a *Random Greedy* foi dado o tempo de execução de 10 segundos e o *limit* foi decidido como 50.

		Greedy Projeto 1	*Random Greedy*	*generate combinations* $p = \frac{1}{2^n}$	*generate combinations* $p = \frac{1}{2^k}$	*generate combinations limit*
SWtinyG	Número de elementos	7	7	2	6	6
	Tempo(s)	1.27e-4	10.000	2.44e-4	5.47e-3	5.26e-3
SWmediumG	Número de elementos	50	48	6	14	13
	Tempo(s)	2.34e-2	10.003	8.27e-2	4.01	4.02
facebook_combined	Número de elementos	1018	850	11	-	-
	Tempo(s)	2.97	10.143	172.92	-	-
RO_edges	Número de elementos	19607	16683	-	-	-
	Tempo(s)	392.15	10.598	-	-	-

Uma maior ponderação nos métodos aqui desenvolvidos levou a constatar-se que estes em si não foram desenhados para estes casos de maior extensividade e que poder-se-ia seguir uma forma de *Decrease and Conquer* em vez de uma linear.

Primeiramente poderia-se testar a combinação com o tamanho máximo e caso se encontre então devolvia-se, caso não exista então tentar-se-ia algumas das combinações com 50% do número de vértices. Verificando-se a existência de algum então testar-se-ia com 75%, caso contrário passar-se-ia para 25%. Na situação de se testarem os 75% e de este existir então efetuar-se-ia para 87.5% mas não existindo experimentar-se-ia com 62.5%. Tomando-se esta metodologia repetidamente enquanto o número de vértices se alterar. Desta forma pode-se esperar que se convirja para o *maximum independent set* de uma forma logarítmica. Podendo-se efetuar diferentes formas de estudo para um dado tamanho, utilizando um dos diferentes métodos, p. Sendo ainda possível efetuar a paralelização para se efetuar o teste das diversas combinações, sendo agora no entanto necessário haver mais elementos a comunicar, nomeadamente os conjuntos já testados que precisam de ser escritos e lidos e o tamanho do resultado final.

Além do mais também se poderiam utilizar formas probabilísticas, através de distribuições estatísticas para o número de combinações a serem testadas, por exemplo normais ou até de qui-quadrado. Sendo que já iriam requerer mais ponderações para o seu estudo e implementação devido à necessidade de utilizar modelos truncados.

VII. Conclusão

Com este projeto foi possível verificar que de facto métodos aleatórios apresentam vantagens e que existem situações em que são a única alternativa concretizável. Havendo situações em que seja concebível haver um limite ao erro cometido estes métodos apresentam ainda uma maior segurança para o seu uso. Nesta situação apenas se limita o erro através da porção de combinações analisadas com o número de combinações totais.

Nos métodos aqui implementados verificou-se que o limitado acabou por apresentar um erro inferior e acabando por ser uma melhor abordagem face ao da concretização de uma dada percentagem das combinações totais. Isto poderá dever-se ao facto de que a resposta certa se encontra mais vezes nos extremos de uma das linhas do triângulo de Pascal em que ao fazerem-se as combinações todas se obtém a resposta certa mais vezes visto estas serem menos que o valor limite. Alterando os valores tanto de *limit* como de p poder-se-iam ter obtido conclusões completamente diferentes.

Ainda assim, os métodos que fazem uso da geração de combinações apresentaram-se inferiores ao *Greedy* que aqui foi desenvolvido. Apresentando no entanto este erros quando foi utilizado nos *benchmarks*, por comparação ao método heurístico desenvolvido no projeto anterior não se sabendo ao certo qual seria a resposta certa.

BIBLIOGRAFIA

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

[2] Motwani, R. & Raghavan, P. Randomized algorithms. *ACM Computing Surveys*. **28**, 33-37 (1996,3), https://doi.org/10.1145/234313.234327

[3] Kroese, D. & Rubinstein, R. Monte Carlo methods. *WIREs Computational Statistics*. **4**, 48-58 (2012), https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.194

[4] Jardosh, A., Belding-Royer, E., Almeroth, K. & Suri, S. Real-world environment models for mobile network evaluation. *IEEE Journal On Selected Areas In Communications*. **23**, 622-632 (2005)

[5] Wikipedia Pascal's triangle. *Wikipedia*. (2022,10), https://en.wikipedia.org/wiki/Pascal%27s_triangle

[6] Wolfram Maximal Independent Vertex Set – from Wolfram MathWorld — mathworld.wolfram.com. https://mathworld.wolfram.com/MaximalIndependentVertexSet.html, [Accessed 10-Dec-2022]

[7] Rozemberczki, B., Davies, R., Sarkar, R. & Sutton, C. GEMSEC: Graph Embedding with Self Clustering. *Proceedings Of The 2019 IEEE/ACM International Conference On Advances In Social Networks Analysis And Mining 2019*. pp. 65-72 (2019)

[8] 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