Encontrar o mínimo conjunto dominante de um grafo usando um algoritmo *Randomized*

Gonçalo Freitas
Mestrado em Engenharia Computacional

***Abstract* –Given a undirected graph G(V, E), use an *Randomized* algorithm to find, if possible, the minimum dominating set of vertices. An analysis about the computational complexity, time execution and number of basic operations of the algorithm will also be made, that allows us to conclude that this algorithm has a complexity of order $\mathcal{O}(n)$, which allows us to study graphs of larger dimensions, when comparing with the algorithms of the first project, however, this algorithm does not always return the optimal solution.**

***Resumo* –Dado um grafo não direcionado G(V, E), é usado um algoritmo *Randomized* para descobrir, se possível, o mínimo conjunto dominante de vértices de um grafo. Também será feita uma análise sobre a complexidade computacional, o tempo de execução e o número de operações básicas do algoritmo, que permite concluir que este algoritmo tem uma complexidade de ordem $\mathcal{O}(n)$, o que permite estudar grafos de dimensões maiores, quando comparando com os algoritmos do projeto 1, no entanto, este algoritmo nem sempre retorna a solução ótima.**

Palavras chave* –Grafo, Conjunto Dominante de vértices, Complexidade Computacional, Algoritmo *Randomized

I. Introdução

Foi, após uma conversa com o professor, proposto alterar o tema do projeto de um problema de decisão para um problema de minimização. Posto isto, ao contrário do que foi feito no projeto 1, onde apenas se procurava encontrar, se possível, um conjunto dominante de vértices para um grafo, agora o principal objetivo do algoritmo é encontrar, se possível, o mínimo conjunto dominante, tomando como mínimo conjunto dominante o conjunto dominante com menos vértices.

A. Conjunto Dominante

Na teoria dos grafos, um conjunto dominante de um grafo G, com V vértices e E arestas, $G = (V, E)$, é um subconjunto D de V de tal forma que cada vértice que não está em D, é adjacente a pelo menos um membro de D. [1]

Analisando o grafo da Fig.1 podemos afirmar que um conjunto dominante de vértices deste grafo poderá ser $D = \{a, g, e\}$.

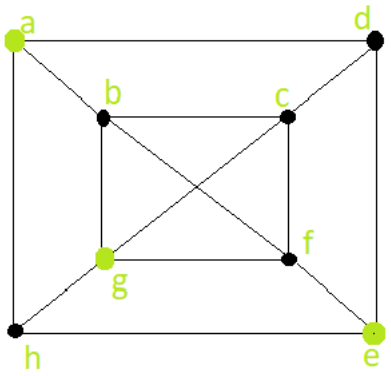

Fig. 1: Exemplo de um grafo com 8 vértices com conjunto dominante. [2]

II. Algoritmia

A. Algoritmo Randomized

Um algoritmo *Randomized* é um algoritmo que usa uma fonte de aleatoriedade como parte da sua lógica, sendo normalmente usado para reduzir o tempo de execução ou a complexidade computacional do problema. O algoritmo funciona gerando um número aleatório, r, dentro de um intervalo especificado de números, e tomando decisões com base no valor de r. [3]
Este tipo de algoritmo ajuda a acelerar um processo de força bruta por amostragem aleatória do input, a fim de obter uma solução que pode, no entanto, não ser totalmente ótima.
Neste caso, aquilo que é gerado aleatoriamente é a combinação de V vértices, com K elementos, que será analisada para verificar se é conjunto dominante. Assim, dado um grafo com V vértices vamos incrementando o valor de K de 1 até $V - 1$, analisando N combinações para cada valor de K, até encontrar um conjunto que verifique as condições de conjunto dominante. Caso encontre um conjunto dominante, o código termina e não analisa valores de K superiores, pois como apenas procuramos o mínimo e começamos a analisar os valores de K em 1 o primeiro conjunto dominante encontrado traduz-se no mínimo conjunto encontrado pelo algoritmo.

Para o valor de N, que representa o número máximo de combinações que irão será analisadas, para cada valor de K, foi assumido o valor retornado pela função $min(N_{comb}, 1000)$, em que N_{comb} representa o número de combinações possíveis de V vértices com K elemen-

tos, e pode ser calculado pela seguinte fórmula:

$$N_{comb} = \frac{V!}{K!(V-K)!}$$

Foi optado usar a função min pois, para valores de V e K elevados, o número de combinações possíveis, N_{comb}, seria muito elevado então, no pior dos casos, apenas analisamos 1000 combinações, de forma a diminuir a complexidade computacional do algoritmo.

Assim, o código desenvolvido para este algoritmo poderá ser descrito, resumidamente, da seguinte forma, onde V representa o tamanho do grafo e usr_conf o valor para a função $min(N_{comb}, usr\_conf)$, no nosso caso este valor foi de 1000.

Algorithm 1 Resumo do código para o algoritmo *Randomized*

```
function CHECK_DOMINATING_SET(V, usr_conf)
    for K = 1, ..., V − 1 do
        N ← V!/(K! × (V − K)!)
        Max ← min(N, usr_conf)
        for i = 1, ..., Max do
            Gerar uma combinação de V vértices de tamanho K, que ainda não tenha sido analisada
            if conjunto dominante then
                Parar o código e devolver a combinação e o valor de K correspondente
            else
                Continuar até encontrar um conjunto dominante ou até i = Max
            end if
        end for
    end for
end function
```

De referir que caso não seja encontrada uma solução para um certo valor de V depois de varrer todos os valores de K possíveis, o código retorna um conjunto vazio. Importante referir também que é mantido, em memória, uma lista com todas as combinações já analisadas para cada valor de K, de forma a nunca analisar a mesma combinação duas vezes.
De maneira a verificar se o conjunto obtido a partir deste algoritmo se traduz no verdadeiro conjunto dominante mínimo, iremos comparar os resultados obtidos por este algoritmo com aqueles obtidos a partir de um algoritmo Exaustivo, adaptado do projeto 1. Um algoritmo Heurístico, adaptado do projeto 1, também será usado para comparar os seus resultados com aqueles do algoritmo *Randomized*, deste projeto.

III. Análise Formal

Dado uma combinação de n vértices com k elementos, de forma a verificar se se traduz num conjunto dominante, é necessário verificar todos os vértices que não estão na combinação e todos que se encontram na combinação. No total, temos $n-k$ vértices que não pertencem à combinação, o que se traduz num somatório de $i = 1$ até $n-k$. Por outro lado, temos k vértices que pertencem a combinação, o que leva a um somatório de $j = 1$ até k. Assim sendo, esta verificação é traduzida em dois somatórios como os de seguida:

$$\sum_{i=1}^{n-k}\sum_{j=1}^{k} 1$$

No entanto, ao contrário do que foi feito no projeto 1, agora não analisamos todas as combinações possíveis, mas sim um número predefinido, C. Posto isto, a complexidade computacional pode ser calculada a partir dos somatórios seguintes:

$$C \times \sum_{i=1}^{n-k}\sum_{j=1}^{k} 1 = C \times k(n-k)$$

Assim, podemos concluir, que este problema possui uma complexidade computacional de ordem n, $\mathcal{O}(n)$, onde $\mathcal{O}$ representa a notação de *Big-O*.
Podemos usar os nossos dados para verificar a complexidade computacional do algoritmo. Realizando então um fit linear aos nossos dados relativos ao número (total) de operações realizadas pelo algoritmo, Fig.2, podemos concluir que o algoritmo possui realmente uma complexidade computacional de $\mathcal{O}(n)$. Para a obtenção deste gráfico considerou-se $P = 0.125$, $V \in [10, ..., 30]$ e $C = 5$, isto é, para cada $K \in [1, ..., V-1]$, o algoritmo apenas analisa (aleatoriamente) 5 combinações.

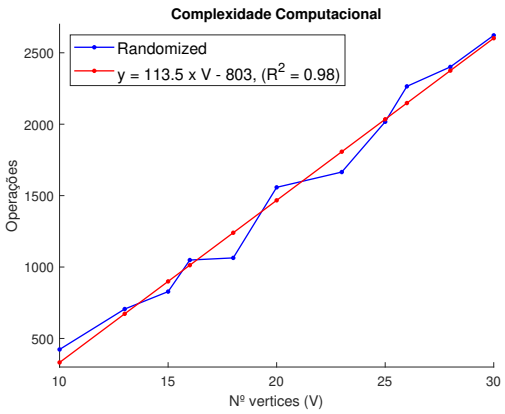

Fig. 2: Verificação da complexidade computacional do algoritmo - $C = 5$ - $P = 0.125$

IV. Resultados

Como este tipo de algoritmos possuem uma grande aleatoriedade associada, foi fixada a *seed* da função *random*, [4], de forma a que cada vez que se corra o código os resultados sejam sempre iguais. Para o valor da *seed*, foi optado usar o número mecanográfico do aluno, neste caso, 98012.

A. *Exemplos*

Podemos adaptar os códigos da pesquisa Exaustiva e do algoritmo Heurístico, do projeto 1, para este problema, de forma a poder comparar com os resultados do algoritmo *Randomized*. Para isso basta, em vez de considerar K como uma percentagem do número total de vértices, considerar K como um valor inteiro pertencente ao intervalo $[1, ..., V-1]$ que se vai incrementando de 1 em 1 até encontrar uma solução ou chegar ao fim do intervalo.

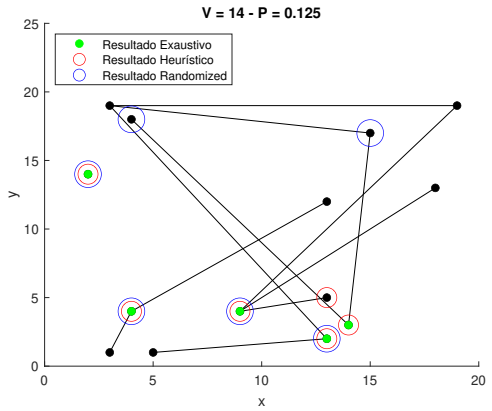

Fig. 3: Exemplo de um grafo com 14 vértices onde os algoritmos Heurístico e *Randomized* não retornaram o verdadeiro mínimo conjunto dominante de vértices.

Na Fig.3 podemos visualizar um exemplo onde os algoritmos Heurístico e *Randomized* retornaram um conjunto de 6 vértices enquanto que o algoritmo Exaustivo retornou um conjunto com 5 vértices. Pelo que concluímos que, apesar de ambos os conjuntos retornados pelos algoritmos Heurístico e *Randomized* serem conjuntos dominantes, estes não traduzem no mínimo conjunto dominante de vértices.

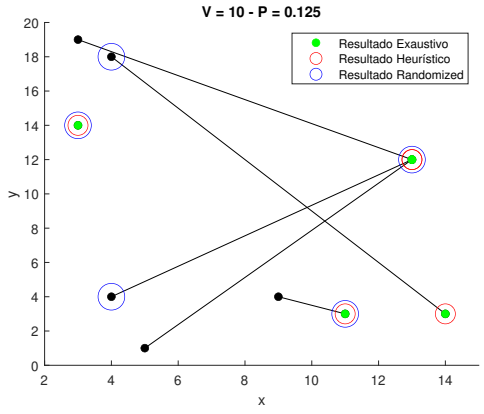

Fig. 4: Exemplo de um grafo com 10 vértices onde os algoritmos Heurístico e *Randomized* retornaram o verdadeiro mínimo conjunto dominante de vértices.

Já na Fig.4 podemos visualizar um exemplo onde todos os algoritmos retornaram conjuntos com o mesmo tamanho, pelo que, neste exemplo podemos afirmar que os algoritmos Heurístico e *Randomized* funcionaram corretamente. De notar também que o conjunto dominante retornado pelo algoritmo *Randomized* é diferente daquele retornado pelos dois outros algoritmos. Isto pode acontecer pois pode haver mais que uma possibilidade para o mínimo conjunto dominante de vértices.

B. *Tempo de Execução e Número de Operações Realizadas*

Para os outros valores de P os gráficos representativos do número de operações de cada algoritmo e dos respetivos tempos de execução encontram-se no Appendix A, Fig.10 e 11, respetivamente. Estes não foram alvo de análise pois esta seria igual a que será feita para um valor de $P = 0.125, 0.25$.

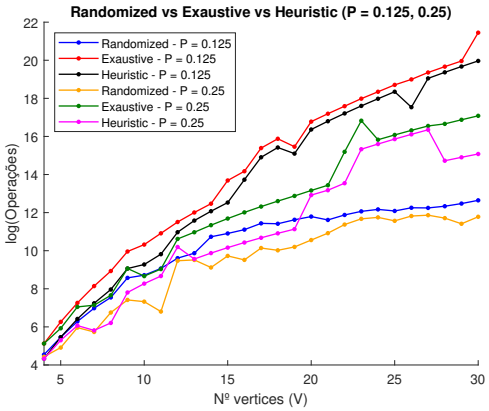

Fig. 5: Logaritmo do número de operações realizadas pelos algoritmos em função do número de vértices - $P = 0.125, 0.25$

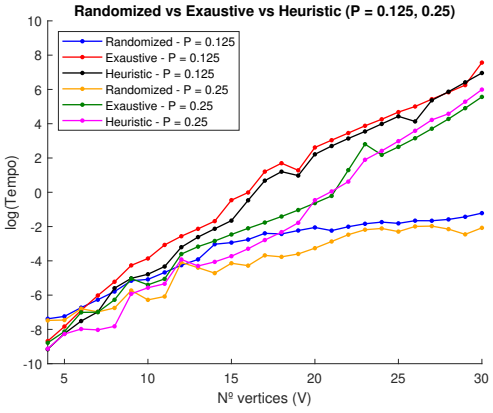

Fig. 6: Logaritmo do tempo de execução dos algoritmos em função do número de vértices - $P = 0.125, 0.25$

Como podemos ver pelos gráficos das Fig.5 e 6, quando comparamos o resultados do algoritmo *Randomized* com os resultados dos outros algoritmos, fixando o valor de P, reparamos que este algoritmo é aquele que realiza menos operações bem como é aquele que

menos tempo de execução requer para encontrar (ou não) uma solução. Assim, podemos concluir que o algoritmo *Randomized* é aquele menos exigente a nível computacional, sendo o algoritmo Exaustivo o mais exigente. Isto leva-nos a concluir que, para grafos de grandes dimensões, o melhor procedimento seria usar o algoritmo *Randomized*, devido à sua inferior complexidade computacional.

C. Número de Configurações

Também poderíamos estudar o número de configurações testadas, isto é, o número de combinações analisadas até encontrar uma solução ou atingir o número máximo de combinações para análise. Contudo, este número seria proporcional ao número de operações realizadas, já analisado anteriormente, Fig.5, e que podemos confirmar pela Fig.7 que realmente o é.

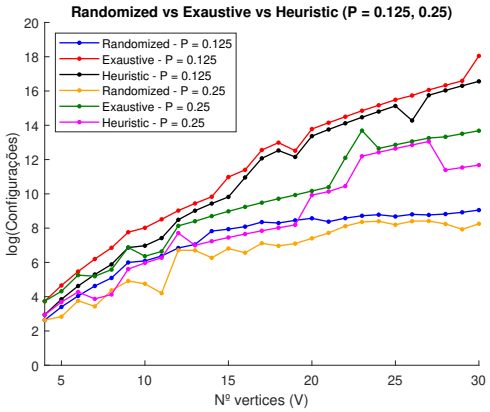

Fig. 7: Logaritmo do número de configurações analisadas pelos algoritmos em função do número de vértices - $P = 0.125, 0.25$

Para os outros valores de P o gráfico representativo ao número de configurações analisadas de cada algoritmo encontra-se no Appendix A, Fig.12.

D. Soluções

Olhando para as soluções obtidas pelos algoritmos Exaustivo, Heurístico e *Randomized*, podemos analisar o número de vezes que estes dois últimos falham, isto é, o número de vezes para o qual o valor de K associado ao tamanho do conjunto dominante mínimo retornado pelos algoritmos Heurístico e *Randomized* é maior que o valor de K retornado pelo algoritmo Exaustivo. Posto isto, obtemos a seguinte Tabela I, considerando $V \in [4, ..., 30]$ e um número máximo de combinações para análise para o algoritmo *Randomized* de $min(N_{comb}, 1000)$.

Como podemos ver pela Tabela I o algoritmo *Randomized* retorna um falso conjunto mínimo menos vezes que o algoritmo Heurístico, o que nos leva a concluir que o algoritmo *Randomized* é uma melhor abordagem a este problema. Sabendo que existem 27 testes para cada valor de P podemos calcular a média com que os algoritmos erram, tendo-se obtido 13.5 vezes para o algoritmo Heurístico, o que se traduz em, aproximadamente, 50.0% das vezes, e 6 vezes para o algoritmo *Randomized*, traduzindo-se em, aproximadamente, 22.2%.

TABLE I: Número de ocasiões que os algoritmos Heurístico e *Randomized* não retornaram o verdadeiro conjunto dominante mínimo de vértices - $min(N_{comb}, 1000)$

Algoritmo	$P = 0.125$	$P = 0.25$	$P = 0.50$	$P = 0.75$
Heurístico	16	10	16	12
Randomized	12	10	2	0

Como é de esperar, caso se desse a opção ao algoritmo *Randomized* de analisar um menor número de combinações, o número de ocasiões que o algoritmo não retorna o verdadeiro conjunto dominante mínimo aumentaria, como podemos analisar pela Tabela II, onde comparamos o número de vezes que este algoritmo falha quando consideramos o número de combinações máximo para análise como $min(N_{comb}, 1000)$ com o número de vezes quando consideramos este valor como $min(N_{comb}, 100)$ e $min(N_{comb}, 10)$.

TABLE II: Número de ocasiões que o algoritmo *Randomized* não retorna o verdadeiro conjunto dominante mínimo em função do número máximo de combinações para análise

$min(N_{comb}, x)$	$P = 0.125$	$P = 0.25$	$P = 0.50$	$P = 0.75$	$Time(s)$
1000	12	10	2	0	4.12
100	18	15	7	0	0.93
10	22	20	14	1	0.45

Assim, da mesma forma que o número de soluções corretas esta relacionada com o número máximo de combinações para análise, a exigência computacional do código também o está, ou seja, quanto maior for o número máximo de combinações que permitimos ao algoritmo de verificar, maior será a precisão e a exigência computacional do algoritmo, o que leva, por sua vez, a um aumento do tempo de execução deste, como se pode ver na coluna '$Time(s)$' na Tabela II, que representa o tempo de execução total, ou seja, $T_{total} = T_{P=0.125} + T_{P=0.25} + T_{P=0.50} + T_{P=0.75}$, para o algoritmo em função do número máximo de combinações para análise.

V. Grafos de dimensões maiores

Visto este algoritmo ter uma grande aleatoriedade associada pois, como analisa combinações aleatoriamente tanto pode encontrar a solução na primeira ou na última combinação analisada levando a que o tempo de execução possa variar bastante entre tamanhos de grafos. No entanto podemos sempre estimar, a partir dos nossos dados, relativos ao tempo de execução, qual seria, aproximadamente, o tempo de execução do nosso algoritmo Randomized para um grafo de tamanho N, tal como realizado no projeto anterior, de forma a ter uma ideia de quanto tempo poderia demorar. Para isso basta, realizar o fit (exponencial) aos nossos dados para

obter uma equação aproximada da curva formada por estes dados. Aqui vamos apenas considerar $P = 0.125$ mas, para outros valores de P poderia-se proceder da mesma forma.

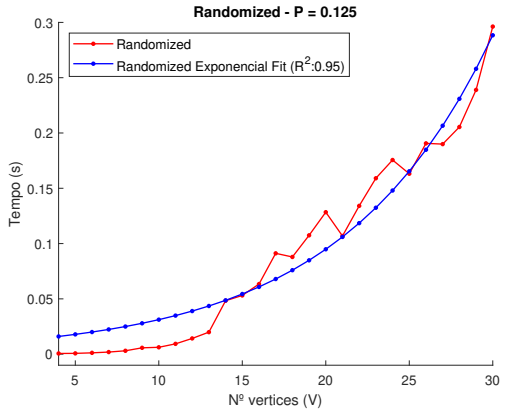

Fig. 8: Tempo de execução do algoritmo Randomized em função do número de vértices + Exponential Fit - $P = 0.125$

Na Fig.8 esta representado o fit realizado ao tempo de execução do algoritmo *Randomized*, para $P = 0.125$, cuja equação é a seguinte:

$$f_R(x) = 0.01026 \times e^{0.1112x}$$

Os gráficos representativos dos nossos dados juntamente com os respetivos fits para os algoritmos Exaustivo e Heurístico estão representados no Appendix B, Fig.13 e 14, respetivamente, cujas equação são as seguintes:

$$f_E(x) = (7.25 \times 10^{-11}) \times e^{1.03x}$$

$$f_H(x) = (6.88 \times 10^{-5}) \times e^{0.5515x}$$

TABLE III: Estimativas do tempo de execução dos algoritmos para grafos de tamanho N - $P = 0.125$

(N, Tempos (s))	Exaustivo	Heurístico	Randomized
$N = 35$	3.29×10^5	1.66×10^4	0.5028
$N = 40$	5.66×10^7	2.62×10^5	0.8768
$N = 45$	9.77×10^9	4.13×10^6	1.5288
$N = 50$	1.68×10^{12}	6.51×10^7	2.6658
$N = 55$	2.90×10^{14}	1.03×10^9	4.6483
$N = 60$	5.00×10^{16}	1.62×10^{10}	8.1051

Analisando a Tabela III reparamos que os tempos de execução dos algoritmos do projeto 1 são bastantes superiores ao tempo de execução do algoritmo *Randomized*. Isto permite-nos concluir, novamente, que, para grafos de grandes dimensões, o algoritmo *Randomized* seria a melhor opção.

Embora se tenha optado por uma abordagem diferente, outra forma de estimativa do tempo de execução do algoritmo poderia ser estimar o pior dos casos, ou seja, analisar sempre o valor máximo de combinações para análise definido, sem parar até chegar a este valor, mesmo que encontre uma solução, de forma assim a ter uma estimativa do tempo máximo que o algoritmo iria demorar para esse tamanho.

Foi realizado também um estudo prático de forma a tentar descobrir qual o maior grafo que se pode estudar com o algoritmo *Randomized*, sem este ter um tempo de execução demasiado elevado. Experimentou-se um grafo com 1000 vértices, no entanto, este teve um tempo total de execução, isto é, $T_{total} = T_{P=0.125} + T_{P=0.25} + T_{P=0.50} + T_{P=0.75}$, de apenas 638 segundos, aproximadamente. Como este tempo de execução ainda é relativamente pequeno, foi feito um estudo para um grafo com 2000 vértices o que elevou o tempo total de execução para, aproximadamente, 2845 segundos, que se traduz em 47 minutos e 30 segundos. Por fim, realizou um estudo para um grafo com 3000 vértices obtendo-se um tempo total de execução de, aproximadamente, 8157 segundos, que se traduz em 2 horas e 16 minutos, pelo que se decidiu parar o estudo neste grafo, assumindo-se então, o grafo com 3000 vértices o grafo de maior dimensão que se pode estudar, em tempo útil.
De notar que estes tempos dependem da rapidez do computador, entre muitos outros fatores, pelo que estes valores podem variar caso se execute o algoritmo noutro computador.

VI. Grafos da Web

Como visto anteriormente, a complexidade computacional e o tempo de execução do algoritmo *Randomized* são inferiores quando comparando com os algoritmos do projeto 1. Isto permite-nos estudar este problema para grafos de maiores dimensões, como alguns daqueles fornecidos pelo professor na pasta zipada *SW_ALGUNS_GRAFOS.zip*, que iremos estudar de seguida.

A. SWtinyG

Este grafo possui 13 vértices e 13 arestas e, como ainda se trata de um grafo de pequena dimensão foi possível implementar o algoritmo Exaustivo a este, tendo o algoritmo retornado um conjunto dominante com 4 vértices, $\{0, 9, 3, 7\}$. Este foi também o conjunto dominante retornado pelo algoritmo *Randomized* pelo que podemos concluir que o algoritmo funcionou corretamente. Em relação ao tempo de execução e ao número (total) de operações realizadas foram obtidos valores de 0.12 segundos e 56862 operações para o algoritmo Exaustivo e valores de 0.03 segundos e 6293 operações para o algoritmo *Randomized*. Estes valores servem para confirmar que o algoritmo *Randomized* possui uma complexidade computacional inferior, tal como visto anteriormente.

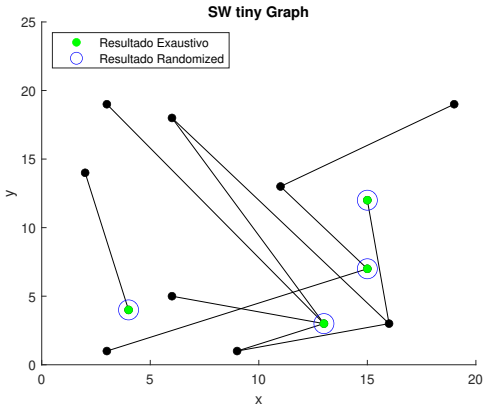

Fig. 9: Exemplo de um grafo seguindo as arestas do grafo *SWtinyG* com os conjuntos retornados pelos algoritmos devidamente assinalados

Na Fig.9 temos um exemplo de um grafo seguindo as arestas do grafo *SWtinyG*, com os conjuntos retornados por cada algoritmo devidamente assinalados.

B. *SWmediumG*

Este grafo possui 250 vértices e 1273 arestas e, como já é um grafo bastante grande não foi possível implementar o algoritmo Exaustivo, devido ao tempo de execução que seria necessário. Contudo, usando a função *dominating_set*, [5], da package NewtorkX foi possível obter um conjunto dominante com 39 vértices. Já o algoritmo *Randomized* retornou um conjunto dominante com 69 vértices pelo que podemos concluir que o algoritmo não funcionou corretamente. Em relação ao tempo de execução e ao número (total) de operações realizadas foram obtidos valores de 35.2 segundos e 19327875 operações.

C. *Grafos direcionados*

Até aqui apenas analisamos grafos não direcionados contudo, este problema também pode ser aplicado a grafos direcionados. Vamos analisar dois exemplos da web, que também podem ser encontrados na pasta zipada *SW_ALGUNS_GRAFOS.zip*, sendo estes os grafos *SWmediumEWD*, um grafo orientado com custos/pesos associados às arestas, e *SWmediumDG*, um grafo orientado sem custos/pesos associados.

TABLE IV: Resultados do grafos direcionados

Grafo	nX - K	K	Operaçoes	Tempo(s)
$mediumEWD$	39	69	19327875	33.38
$mediumDG$	21	37	1433998	1.34

Como podemos ver pela Tabela IV, em ambos os casos, o algoritmo não retornou o verdadeiro conjunto dominante mínimo, isto pois, a função *dominating_set*, [5], da package NewtorkX retornou conjuntos dominantes com valores de K mais pequenos, cujo valor esta representado na Tabela IV na coluna 'nX - K', do que o valor retornado pelo algoritmo *Randomized*, representado na coluna 'K'. Já nas colunas 'Operações' e 'Tempo(s)' temos representado o número total de operações realizadas pelo algoritmo e o tempo de execução deste, respetivamente.

Não foi feito o estudo para outros grafos, de maior dimensão, como, por exemplo, o *SWlargeG*, pois não seria possível obter um resultado para estes em tempo útil.

VII. Aplicações na vida real

Grafos são objeto de vários estudos e têm inúmeras aplicações na vida real como, redes sociais, redes biológicas, recomendações de produtos, mapas/gps entre muitas outras, [6]. Para este problema em específico, uma boa aplicação seria, por exemplo, junto da polícia para descobrir os chefes por detrás de uma rede criminosa, isto pois, os chefes contactam com vários trabalhadores que, em principio, têm pouco contacto entre sim, ou seja, dado N suspeitos (vértices) e as suas relações (arestas) o mínimo conjunto dominante de suspeitos pode traduzir-se no conjunto das principais pessoas por detrás da rede criminosa.

VIII. Conclusão e Trabalho futuro

Este trabalho permitiu por em prática vários tópicos, desde os mais teóricos, como a análise formal para descobrir a complexidade computacional, aos mais práticos, nomeadamente a escrita de código.

Foi possível também reparar nas vantagens e desvantagens dos algoritmos *Randomized*, em que vimos que estes algoritmos possuem uma complexidade computacional baixa, que nos leva a conseguir estudar grafos de maiores dimensões quando comparando com os algoritmos do projeto 1 mas, no entanto, a solução retornada pelo algoritmo pode não se traduzir na solução ótima.

Para trabalho futuro, este passaria por otimizar o código, visto que os grafos utilizados para problemas reais possuem uma grande quantidade de vértices e arestas, como pode ser encontrado em [7], bem como a melhor definição do número de combinações máximo para análise de forma a que este não seja constante, tal como aconteceu neste projeto para grafos em que $N_{comb} \geq 1000$, mas sim uma valor que varie consoante o tamanho do grafo.

References

[1] E. Sampathkumar and L. Latha, "Set domination in graphs", *Journal of Graph Theory*, vol. 18, pp. 489 – 495, 08 1994.

[2] GeeksforGeeks, "Dominant set of a graph", `https://www.geeksforgeeks.org/dominant-set-of-a-graph/`, Aug 2022.

[3] Joaquim Madeira, *Slides Teóricos - Algoritmos Avançados (40751)*, Universidade de Aveiro, 2022-23.

[4] Python documentation, "Random - generate pseudo-random numbers", `https://docs.python.org/3/library/random.html`.

[5] "Networkx documentation - dominating_set", https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.dominating.dominating_set.html#networkx.algorithms.dominating.dominating_set.

[6] "Applications of graphs in computer science", https://unacademy.com/content/nta-ugc/study-material/computer-science/applications-of-graphs-in-computer-science/, Jun 2022.

[7] Jure Leskovec and Andrej Krevl, "SNAP Datasets: Stanford large network dataset collection", http://snap.stanford.edu/data, June 2014.

Appendices

I. Resultados

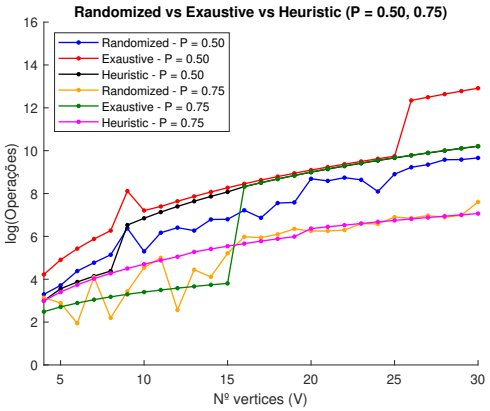

Fig. 10: Logaritmo do número de operações realizadas pelos algoritmos em função do número de vértices - $P = 0.50, 0.75$

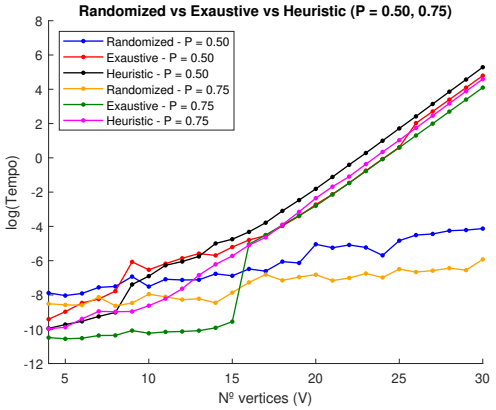

Fig. 11: Logaritmo do tempo de execução dos algoritmos em função do número de vértices - $P = 0.50, 0.75$

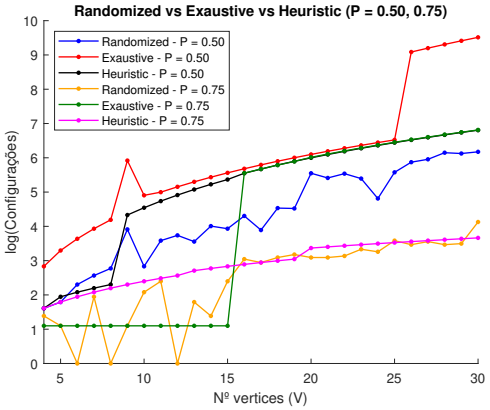

Fig. 12: Logaritmo do número de configurações analisadas pelos algoritmos em função do número de vértices - $P = 0.50, 0.75$

II. Grafos de dimensões maiores

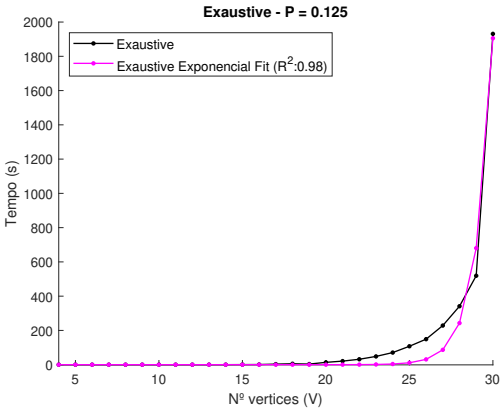

Fig. 13: Tempo de execução do algoritmo Exaustivo em função do número de vértices + Exponential Fit - $P = 0.125$

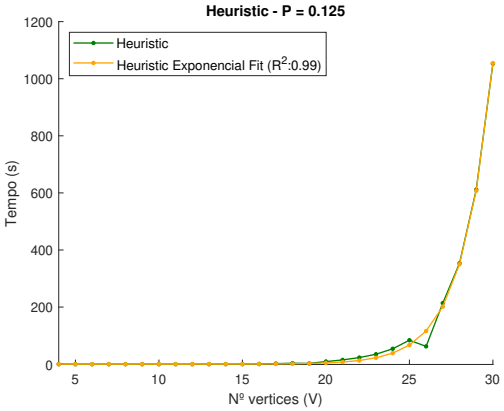

Fig. 14: Tempo de execução do algoritmo Heurístico em função do número de vértices + Exponential Fit - $P = 0.125$