Propagação da Gripe Espanhola em Redes de Contato

Uma abordagem computacional utilizando grafos

LUCAS SARAIVA FERREIRA, Universidade Federal de Minas Gerais, MG

Palavras-chave: Grafos, Gripe Espanhola, Influenza, Complexidade

1 INTRODUÇÃO

A transmissão de doenças infecciosas ocorre de diversas formas, por exemplo o contato direto (Secreções) e indireto (Objetos) com pessoas infectadas. Um exemplo de doença transmitida por contato direto é a Gripe Espanhola. A gripe espanhola era transmitida através de gotículas de secreção de uma pessoa já infectada. No total, a gripe espanhola matou cerca de 50 milhões de pessoas entre 1918 e 1920 [Taubenberger and Morens 2006].

A dinâmica da transmissão de doenças e os fatores que influenciam na transmissão são de grande importância para entender a doença. Além disso, partindo destes fatores é possível criar medidas de prevenção, fazer previsões de contágios em massa, traçar zonas de quarentenas entre outros.

No caso da gripe espanhola, o contato entre pessoas foi crucial para a propagação da doença em massa. Quando uma nova doença é inserida na sociedade, o contagio é quase iminente ao entrar em contato com uma pessoa infectada. Levando em conta este cenário (contagio iminente) é possível calcular o número mínimo (Ou próximo do mínimo) de pessoas necessárias para contagiar toda um bairro, cidade, país ou continente.

Nesse contexto, o referido trabalho se propõe a encontrar o conjunto mínimo (ou próximo) de pessoas necessárias para causar uma pandemia ou contagio em massa de dada doença. Será utilizada a Gripe Espanhola como doença a ser propagada na rede, por ser uma doença conhecida pelo seu elevado grau de contagio e mortalidade.

Para encontrarmos esse número é necessário criar uma rede de contato, modelando a conexão entre as pessoas. Apos isso, podemos agregar outros fatores da doença (Tempo de incubação, mortalidade entre outros) a análise e realizar previsões. Por exemplo: Considere uma rede de contato com 20 pessoas e suas conexões, suponhamos que basta 3 pessoas se contagiarem para que toda a rede pegue a doença. Sabendo que a doença demora 1 semana para manifestar e se tornar contagiosa, e que o número mínimo de pessoas para causar uma pandemia na rede é 3, as autoridades precisam agir de forma rápida, para conter o vírus o quanto antes o possível.

O trabalho se divide da seguinte forma: Apresentação do problema de forma formal e sua dificuldade, modelagem em grafo do problema, logica e complexidade do algoritmo baseline, logica e complexidade do algoritmo heurístico, experimentação utilizando diferentes grafos e conclusão.

2 DEFINIÇÃO DO PROBLEMA E MODELAGEM

A definição formal do problema a ser tratado neste trabalho é:

> Dado um conjunto de pessoas interconectadas pelos seguintes critérios:
> (1) A pessoa X tem contato com a pessoa Y.
> (2) A pessoa X ou a pessoa Y é propensa a contrair a doença.
> Desejamos descobrir qual o número mínimo de pessoas necessárias para contagiar toda a rede. Considerando que dado uma pessoa i e uma pessoa j conectadas por uma aresta $e(i, j)$, a pessoa j contrai a doença ou é a infectada com a doença. Ao final, teremos dois conjuntos, o conjunto dos Infectados e o conjunto dos Suscetíveis. O conjunto dos infectados consiste na resposta, o número mínimo de pessoas para atingir todos suscetíveis.

Este problema pode ser considerado um problema difícil de ser resolvido, onde sua solução exata é custosa, como mostrado na seção 2.2. Para resolver o problema podemos utilizar estratégias existentes para encontra o "Conjunto Dominante" ou "Número de dominância" em um grafo. O problema do conjunto dominante consiste em encontrar um grupo de vértices D de tal modo que cada vértice fora de D é adjacente a pelo menos um vértice em D. O número de dominação do grafo é o número de vértices no conjunto D, que por sua vez pode ser lido como a quantidade de pessoas necessárias para "dominar" a rede, ou seja, contagiar todas as pessoas. É importante notar que, estamos trabalhando sobre o problema de Dominância de Vértices, mas o objetivo é voltado a propagação de doenças.

2.1 Modelagem em grafo

O problema a ser abordado pode ser modelado como um grafo. A definição formal deste grafo é:

- Seja G um grafo não direcionado, não ponderado, sem vértices com ciclos em si mesmo e conexo;
- Seja V o conjunto de vértices que compõem G, cada $v \in V$ corresponde a uma pessoa;
- Seja E o conjunto de arestas que componem G, cada $e \in E$ corresponde a uma relação entre dois vértices $e(i, j)$, ou seja, pessoas que tem contato.

O grafo modela as pessoas de alguma região, que tem contato entre si e que são potenciais "vetores" da doença sendo analisada, a Gripe Espanhola. Uma vez que será considerado um cenário onde ninguém tem imunidade a doença, não é necessário colocar pesos nas arestas, que poderiam eventualmente representar uma chance de contagio.

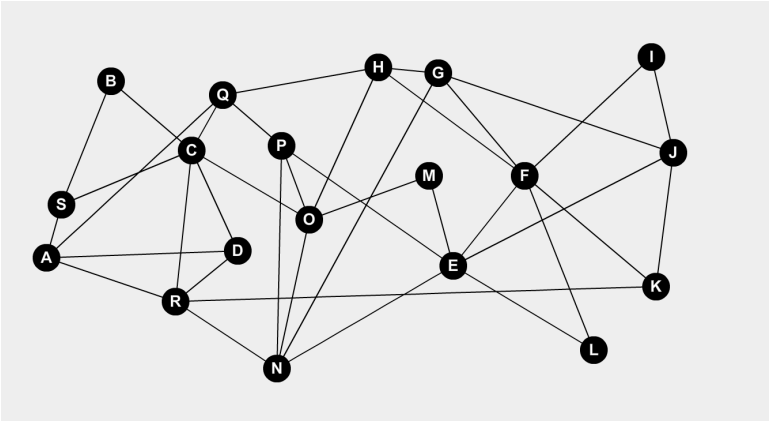

Fig. 1. Exemplo de grafo G de entrada

A figura 1 mostra um exemplo de grafo de entrada para o problema, nesta rede, o conjunto mínimo de pessoas necessárias é composto pelos vértices: $D = \{A, C, F, E\}$. Logo, o número de dominância deste grafo é 4.

Se estas pessoas contraírem a doença, elas serão suficientes para propagar a doença para todos outros envolvidos na rede. Causando assim um estado de pandemia. Esse tipo de comportamento pode ser encontrado na história quando um novo patógeno afeta a sociedade. A peste negra é um exemplo deste acontecimento. Bastou que algumas pessoas se infectassem com a doença para causar um alastramento em massa [Byrne 2004].

Se a doença da rede acima tivesse fosse letal em poucos dias após a manifestação dos sintomas, seria necessária uma ação rápida das autoridades de saúde para impedir que ela se alastrasse.

Obviamente, quanto mais conexões, menos pessoas são necessárias para causar um problema gigante.

Esta solução foi calculada utilizando o algoritmo *baseline* apresentado no próximo capítulo.

2.2 Dificuldade do problema

O problema do conjunto dominante em sua versão de decisão é um problema NP-Completo. A versão de decisão do problema do conjunto dominante pode ser descrita da seguinte forma:

Dado um grafo ***G*** e um inteiro ***k***, existe um conjunto dominante de tamanho no máximo k?

Segue abaixo uma prova que dominância de vértices é Np-Completo. Redução de cobertura de vértices para dominância de vértices.

- **Provar que Dominância de vértices é NP:**
  Dado um conjunto de vértices ***D***, um conjunto de vértices ***V'*** e um grafo ***G*** é possível criar um algoritmo que em tempo polinomial checa se esta solução é correta. Seja ***D*** o conjunto solução, os vértices dominante e ***V'*** os vértices não inseridos na solução ***D***. Percorra ***V'*** checando se para cada **v** pertencente a ***V'*** existe um vizinho de **v** em ***D***, para tal é necessário percorrer todos vértices de ***D***. Caso todos v_i vértices de ***V'*** tenham um vizinho em ***D***, a resposta está correta, do contrário, a resposta está incorreta.
  Este algoritmo roda em tempo $O(V'D)$, sabendo que a união de ***V'*** e ***D*** deve resultar em ***V***, podemos afirmar que o algoritmo roda em $O(V)$, sendo ***V*** o conjunto inicial de vértices do grafo ***G***.
- Redução:

(1) **Entrada:**
Dado um grafo conectado ***G*** transformamos todas arestas em ***G*** em um novo vértice conectado aos vértices pertencentes a esta aresta, geramos assim ***G'*** em tempo polinomial. Dado um valor ***K*** para o problema da cobertura de vértice, igualamos o valor de ***K'*** ao de ***K***.

(2) **Saída da cobertura de vértices para conjunto dominante:**
Se existir uma cobertura de vértices em ***G*** de tamanho ***K***, logo existe uma dominância de vértice de tamanho K em ***G'***, já que os vértices de ***G*** existem em ***G'***.

(3) **Saída do conjunto dominante para cobertura de vértices:**
Se em ***G'*** existe uma dominância de vértices de tamanho ***K'***, podemos afirmar que ***G*** tem uma dominância de tamanho ***K***. Para garantir isto, basta testar se o vértice escolhido no conjunto dominante é um dos vértices artificiais (Vértices que se tornaram arestas), caso sim, substitua ele por um de seus adjacentes, que consequentemente pertencem ao conjunto ***V***.

A versão de otimização deste problema, a qual é tratada neste trabalho, consiste em encontrar o menor número de pessoas necessárias para dominar todo o grafo. Esta versão do problema exige que sejam feitas combinações de soluções ou que decisões sejam revisadas, uma vez que não existe uma escolha gulosa/exata que leve a resposta ótima em tempo polinomial. Encontrar a resposta ótima ou aproximada do conjunto dominante mínimo não é trivial e demanda bastante esforço [Laskar et al. 1984]. Diversas tentativas de melhorias para classes de grafos específicas e/ou arbitrarias vem sendo publicadas, podemos citar alguns artigos: [Fomin and Thilikos 2006], [Bourgeois et al. 2013], [Van Rooij and Bodlaender 2011] entre outros. Notamos que encontrar o número mínimo de pessoas para causar a infecção na rede pode ser mapeado ao problema da Dominância de vértices, podemos assumir dessa forma que este é um problema difícil de ser resolvido.

3 BASELINE

Foi decidido utilizar o algoritmo exato criado por Fomin [Fomin et al. 2005] como *baseline*. Em seu artigo, Fomim explica que este é o primeiro algoritmo a quebrar a barreira do "2^n" para o problema do conjunto dominante mínimo. Seu algoritmo realiza algumas reduções e lemas para ganhar eficiência. Tendo o conjunto dominante mínimo, temos o número de pessoas necessárias para propagação da doença em todo o grafo.

O algoritmo tem como pilar o seguinte lemma: "Dado um grafo G com N vértices de grau no mínimo 3, existe um conjunto dominante de tamanho 3n/8" [Reed 1996].

O algoritmo reduz o grafo realizando a remoção de vértices com grau 1 e 2, até encontrar um grafo com vértices de grau zero ou grau pelo menos 3. Tendo um grafo com vértices de grau maior igual a 3, utilizamos o lema de Reed, partindo deste ponto o algoritmo força bruta é executado recursivamente tentando encontrar a partição de tamanho máximo 3n/8 que corresponda a um conjunto dominante mínimo.

Dado um grafo *G = (V, E)*, um subconjunto de vértices *X*, um conjunto *D* contendo uma solução parcial e um vértice v escolhido para análise. Para gerar os subcasos, quatro testes distintos, para cada v , são realizados. Uma vez encontrado qual caso se encaixa e realizada certas operações, uma chamada recursiva é realizada sobre o novo grafo gerado. Os testes são:

- **Caso A**: O vértice v tem grau 1 e $v \in V$. Nesse caso não é necessário dominar o vértice v. Removemos v de G e X. Realizamos uma chamada recursiva. Este caso tem a seguinte equação de recorrência: $T(n) = T(n-1)$
- **Caso B**: O vértice v tem grau 1 e $v \in X$. Seja w o único vizinho de v. Nesse caso removemos v e w de G e removemos de X todos vizinhos de w. Este caso tem a seguinte equação de recorrência: $T(n) = T(n-1)$
- **Caso C**: O vértice v tem grau 2 e $v \in V$. Seja *w1* e *w2* os vizinhos de v, temos 3 possíveis decisões:
  - Se $w1 \in D$ e $v \notin V$. Removemos *w1* e v de G, removemos *w1* de X, realizamos a recursão.
  - Se $v \in D$ e $w1, w2 \notin D$. Removemos *w1, v* e *w2* de G, removemos *w1* e *w2* de X, realizamos a recursão.
  - Se $w1 \notin D \wedge v \notin V$. Removemos v de G e realizamos a recursão.

  Este caso tem a seguinte equação de recorrência: $T(n) = T(n-1) + T(n-2) + T(n-3)$
- **Caso D**: O vértice v tem grau 2 e $v \in X$, realizamos os mesmos testes do caso C. Somente o terceiro teste tem resultado diferente: removemos v e *w2* de G, removemos os vizinhos de *w2* de X e, por fim, realizamos a recursão. Este caso tem a seguinte equação de recorrência: $T(n) = 2 * T(n-2) + T(n-3)$

Como o autor afirma em seu artigo, este é um algoritmo exato (exponencial) e que se aproveita da força bruta para instâncias reduzidas do problema. Sendo esta redução não trivial ou simplista.

3.1 Complexidade do *baseline*

A complexidade deste algoritmo é dada pelo autor. Através de diversas análises Fomin chega à conclusão que a tarefa mais custosa é a de checar todos subconjuntos de tamanho $3t/8$. Sendo esta parte realizada em $O(1.93782^n)$. As inúmeras recursões que o algoritmo realiza nos quatro casos demonstrados acima, tem uma complexidade total de: $T(n) = O(1.8393^n)$.

Logo, existe um algoritmo exato(Exponencial) para encontra o conjunto dominante mínimo de um grafo qualquer em $O(1.93782^n)$.

Todos cálculos de complexidade, que são de certa forma complexos, e a descrição formal do algoritmo aqui citada, foram retiradas do artigo "Exact (exponential) algorithms for the dominating set problem" - Fedor V. Fomin et Al. [Fomin et al. 2005]

3.2 Testando o *baseline*

O algoritmo *baseline* foi implementado em Java 8. Diversos grafos pequenos foram testados para se verificar o funcionamento do programa. Grafos com um grande número de vizinhos levam tempo considerável para rodar, pois não é possível realizar muitas reduções na entrada inicial de qualquer grafo arbitrário. Mesmo sendo um algoritmo exato otimizado, ele continua sendo exponencial, logo, mesmo entradas consideradas triviais para algoritmos polinomiais tomam um tempo absurdo para serem executadas.

Quanto mais redutível a entrada (Mais vértices de grau 1 ou 2) melhor o desempenho do algoritmo. Um grafo completo neste algoritmo ira rodar sem nenhuma redução, entradas deste tipo com mais de 20 vértices podem levar um tempo muito grande para serem executadas.

A saída do programa consiste no valor mínimo do número de dominação do grafo. Ou seja, o número mínimo de pessoas necessárias para infectar toda a rede. A figura 2 mostra um exemplo da saída do algoritmo exato para um grafo pequeno onde os vértices têm poucos vizinhos.

```
Analyzing /home/saraiva/eclipse-workspace/DominatingSet/src//Ex1.txt ...
The undirected graph has 19 nodes.
The Undirected graph has 35 edges.

The program found a minimum dominating set in 0.1700000762939453 seconds.
The minimum dominating set found contains 5 nodes.
The minimum dominating set is: [1, 3, 6, 10, 15]
```

Fig. 2. Exemplo de saída do programa implementado

Apesar de ser um algoritmo exponencial, ainda assim diversos casos de testes conseguem obter uma resposta de forma rápida, devido as reduções feitas pelo algoritmo. Estas analises podem ser encontradas na seção 5 deste trabalho.

4 HEURÍSTICA DESENVOLVIDA

A heurística aplicada visa encontrar uma resposta mais próxima da ótima, para resolver o problema será criado um algoritmo que aplicara a seguinte heurística:

Dado um grafo ***G = (V,E)***, um conjunto ***D*** contendo a resposta atual, ordenamos os vértices de ***G*** pelo grau. Para cada vértice ***i*** de ***V*** precisamos decidir se incluímos ***i*** no conjunto ***D***. Seja ***N[i]*** o conjunto de vizinhos de ***i***.

Para criar essa heurística, foi decidido explorar uma propriedade de grafos que pode ajudar a encontrar uma resposta a este problema. Uma pessoa com mais conexões, infecta mais pessoas. Baseado nisto, tomamos decisões de inclusão ou não destas pessoas na resposta, o mero fato de ela ser um vértice de grau alto não torna ela uma resposta viável, é necessário verificar algumas restrições aqui criadas:

(1) **Caso 1:** Se $i \notin D$ e *N[i]* tenha mais de dois membros que não pertencem a D, adicione i a D;
(2) **Caso 2:** Se $i \notin D$ e exatamente 1 vizinho de i não está em D, adicione o vizinho que não está em D ao conjunto D;
(3) **Caso 3:** Se $i \notin D \wedge N[i] \subseteq D$ vá para o próximo vértice.
(4) **Caso 4:** Se $i \in D$ vá para o próximo vértice.

Exemplo: Suponha que um vértice ***i*** foi selecionado, este vértice tem 3 adjacentes *(x,y,z)*. Dos 3 adjacentes, dois já foram dominados *(x,y)*. Este é o **Caso 2**, neste caso o algoritmo deve escolher o vértice adjacente ainda não dominado: ***(z)***. Com esta escolha ele dominará o vértice ***i***, ele mesmo ***(z)*** e seus adjacentes, caso existam.

Uma vez realizada estas operações, uma resposta é gerada. Esta resposta então é revisitada e duas verificações são feitas, no intuito de melhorar a solução:

- **Revisitando a resposta:** Dado um vértice *i*, se este for dominado por dois vértices ele é um candidato a ser o verdadeiro dominante, faça a seguinte checagem: Se seus vizinhos que te dominam, dominam vértices que são dominados por outros, logo eles são descartáveis e *i* deve ser dominado no lugar.
- **Finalizando a resposta:** Dado dois vértices ***(i,j)*** do conjunto solução ***D***, se ***i*** for adjacente a ***j*** e ***i*** ou ***j*** não tem outros vértices adjacentes, remova o sem outros adjacentes.

Em resumo, a heurística se aproveita dos vértices de maior grau, mas a decisão de adicionar este vértice no conjunto solução não parte somente deste parâmetro. A ideia de ordenar os vértices pelo grau é uma tentativa de dominar mais vértices, com menos escolhas.

4.1 Complexidade da heurística

A complexidade do algoritmo heurístico foi calculada baseada na implementação realizada. O codigo foi dividido em 4 blocos, cada bloco realiza uma quantidade de funções. Abaixo é apresentado a complexidade de cada bloco:

(1) Bloco 1 - Processamento da entrada: Os vértices do grafo são ordenados em ordem decrescente em uma lista por ordem de grau. A ordenação aqui utilizada foi a padrão da biblioteca Collections do Java8, em sua documentação consta a complexidade. Esse bloco tem complexidade $O(n \lg_n)$;
(2) Bloco 2 - Escolha gulosa de vértices: Para cada vértice na lista ordenada, é verificado se ele será incluso na solução ou não, baseado nos casos apresentados na seção 4. Para cada vértice que é incluído na solução, precisamos marcar seus vizinhos como infectados. No pior caso, esse bloco acessa todos vértices e eles são vizinhos de todos os outros vértices, logo todas arestas são acessadas V vezes. Concluímos então $O(VE)$;
(3) Bloco 3 - Revisitando a resposta: Este bloco faz a primeira melhoria da resposta, revisando a solução calculada e tentando trocar alguns vértices. No pior caso, todos V vértices, serão dominados por todos V vértices que por sua vez dominam V vértices (Imagine um grafo completo, todo mundo domina todo mundo). Nesse caso, a complexidade máxima atingível aqui é $O(V^3)$;
(4) Bloco 4 - Finalizando a resposta: Uma última limpeza é realizada, sendo está feita somente nos vórtices escolhidos como resposta. Este conjunto tem no máximo tamanho V. Por exemplo, caso todos vértices sejam isolados, todos estarão no conjunto resposta. É feita uma iteração sobre os vizinhos dos dominantes. É possível dizer que este bloco tem uma complexidade de aproximadamente $O(V^2)$;

Levando em conta a parte mais custosa do algoritmo, sendo está o **Bloco 3**, podemos estimar que o algoritmo criado tem complexidade $O(V^3)$. Sendo este um algoritmo polinomial que encontra o numero aproximadamente mínimo de pessoas para propagar a doença.

5 EXPERIMENTAÇÃO

Aconteceram 3 etapas de experimentação. O ambiente utilizado nos testes foi um notebook Lenovo Z50, processador I7, 8GB de RAM e sistema Linux distro Debian. Para todos testes listados abaixo ambos algoritmos foram executados com as mesmas entradas e no mesmo ambiente. Salvo os testes da subseção 6.2 que foram realizados somente com o algoritmo heurístico.

5.1 Testes de Tempo e Espaço

Os testes aqui realizados visam comparar o algoritmo exato com o algoritmo heurístico. Foram testados grafos gerados de forma aleatória e de tamanho pequeno (Por volta de 30 vértices). Os grafos foram testados e em todos casos ambos algoritmos obtiveram a resposta ótima. Cada grafo

conta com a variação de denso e esparso, os resultados obtidos nos grafos densos são apresentados nas tabelas desta seção. Cada grafo foi executado 15 vezes e uma media dos tempos obtidos foi calculada. Na seção 6.2 é apresentado um teste com grafos maiores para o algoritmo heurístico onde é possível ver sua curva de crescimento

A tabela 1 mostra o tempo, em segundos, obtido por ambos algoritmos na execução. Para o algoritmo heurístico o tempo foi menor que 1 segundo para todas entradas, uma vez que são entradas extremamente pequenas para um algoritmo polinomial. O mesmo não se nota no algoritmo exato, onde para o grafo denso de 30 vértices, levou aproximadamente 35 minutos para terminar. Em nível de comparação, o algoritmo exato levou 932.120 segundos para rodar o grafo esparso 30 vértices. Foi possível notar que o aumento do tempo de execução se deu devido ao aumento de vértices e arestas, já que o exato não consegue realizar muitas reduções a medida que o grafo cresce principalmente em arestas. A última coluna da tabela mostra o qual melhor foi o algoritmo heurístico em relação ao tempo do exato.

Table 1. Tempo do Algoritmo Heurístico vs Exato para Grafo Denso em segundos

Num Vértices	Tempo Alg. Exato	Tempo Alg. Heurístico	Melhoria
20	1.135s	0.0032s	99.762%
25	51.167s	0.0041s	99.991%
30	2214.0069s	0.0058s	99.999%

Para instâncias pequenas, o algoritmo tem um bom tempo. Mas é importante notar que nem sempre a resposta será a ótima, nos casos aqui apresentados foi garantido que o heurístico calculava a resposta certa, gerando grafos até encontrar algum do tamanho necessário(20, 25, 30) que ele encontrava a resposta.

A tabela 2 apresenta a análise de espaço (Memória da JVM) gasta por cada algoritmo durante a computação. Como esperado, o algoritmo exato toma muito mais espaço que o heurístico por ser um algoritmo recursivo e manter na memória diversos "casos" que estão sendo calculados. O algoritmo heurístico realiza computações sobre algumas listas, logo seu uso de memória é pequeno e cresce de forma sutil para grafos pequenos. Obviamente, grafos muito grande podem aumentar o tamanho das listas e causar um uso de espaço muito maior que os aqui apresentados.

Table 2. Resultados Algoritmo Heurístico vs Exato para Grafo Denso em Memória (MB)

Num Vértices	Memória Total	Memória Usada Exato	Memoria Usada Heurístico
20	212.860	30.09788	2.600512
25	212.860	62.661784	2.600528
30	212.860	98.529512	2.600541

5.2 Teste de Contagio

Esta bateria de testes foi executada exclusivamente sobre o algoritmo heurístico. A premissa inicial deste trabalho era encontrar o número mínimo de pessoas para propagação da gripe espanhola em uma rede. Esta seção demonstra os testes realizados com este intuito.

Foram geradas redes simulando o grafo descrito na seção 2.1. Grafos contendo 1000, 2000, 3000, 4000, 5000 e 6000 vértice foram gerados. Foi gerada uma instância esparsa e uma instância densa ($E = \frac{|V^2|}{2}$) para cada grafo.

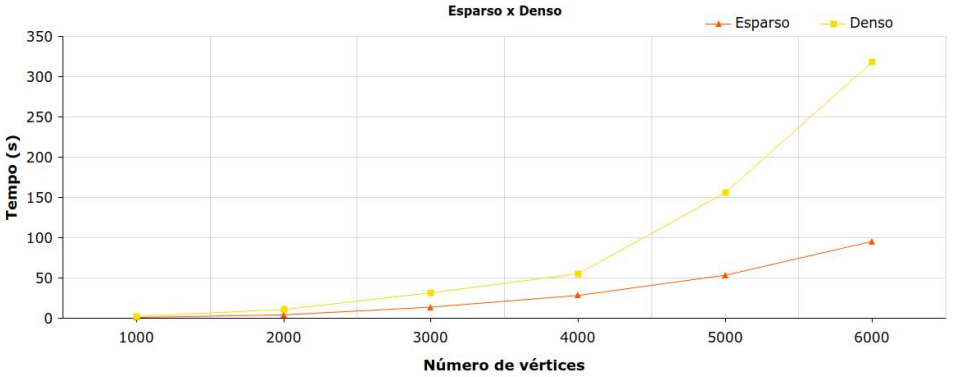

Fig. 3. Comparação de tempo entre grafos Denso e Esparso

A figura 3 mostra a curva do tempo de execução em detrimento do tamanho da entrada. Uma curva representa os grafos esparsos e a outra os grafos densos. Neste caso não é possível dizer se a resposta encontrada foi a ótima, podemos afirmar que no pior dos casos foi aproximada.

Para o grafo de tamanho 1000, o algoritmo calculou que seriam necessárias 13 pessoas para infectar toda a rede e para o grafo de 4000 foi encontrado um valor de 8. Esses são números pequenos e que tecnicamente não fogem da realidade. Ao se estudar as epidemias podemos notar que não se faz necessário um alto número de pessoas infectadas para realizar a propagação da doença em massa. Além disso, era esperado que em grafos densos este número fosse menor se comparado ao esparso, já que cada pessoa (vértice) tem contato com mais pessoas. Por exemplo, para o grafo denso de 1000 vértices, o algoritmo calculou que seriam necessárias 6 pessoas para a infecção de toda a rede.

O crescimento apresentado na curva era esperado, onde grafos mais densos levariam um tempo maior de processamento. Para cada vértice escolhido é necessário atualizar e checar uma grande quantidade de vizinhos. Podemos notar que o crescimento é polinomial e que o algoritmo heurístico consegue processar grafos 100x maiores que os do exato (Vide seção 6.1) em tempo aceitável.

5.3 Testes por Tipo

Entradas representando os seguintes tipos de grafo foram montadas e testadas em ambos algoritmos: Grafo Nulo, Arvore, Completo, Estrela, Caminho (Basicamente uma reta), Bipartido, Cubo e Peterson.

Para cada tipo de grafo, foram realizadas 15 rodadas de testes. Foi retirada a média dos tempos de execução e esta é a que consta na tabela 3. O número de vértices de cada tipo foi decidido de forma a agilizar o processo de testes, visando execuções rápidas em ambos algoritmos. O foco aqui não foi o tempo e sim se era dada a resposta ótima ou não. Os tempos são um adicional neste teste. Podemos observar na tabela 3 que para 3 tipos de grafos o algoritmo heurístico não conseguiu obter a resposta ótima. Uma pequena análise da distância da resposta dada pelo heurístico, em relação a ótima, pode ser encontrada na tabela 4.

A tabela 4 mostra o qual longe estão as respostas da Heurística para o grafo do tipo Bipartido. É notável que o algoritmo heurístico encontrou respostas em alguns casos $2x$ maiores que o mínimo. Durante todos testes, este foi o pior caso encontrado para a heurística. A figura 3 ilustra o grafo de 10 vértices bipartido. Todos outros aqui testados tem configurações parecidas, tendo somente um acréscimo de vértices e arestas.

No grafo da figura 4 a resposta correta seria 3, sendo o conjunto $D = \{6, 3, 7\}$ uma possível solução. A heurística encontra uma resposta contendo 4 vértices $D = \{6, 9, 8, 4\}$.

Table 3. Resultados por Tipo

Tipo	Num Vértices	Tempo Alg. Exato	Tempo Alg. Heurístico	Mínimo
Nulo	100	0.0140s	0.0060s	Sim
Arvore	40	131.5759s	0.0079s	Sim
Completo	30	6.6802s	0.0050s	Sim
Estrela	100	140.2710s	0.0050s	Sim
Caminho	30	218.8299s	0.0035s	**Não**
Cubo	20	0.1090s	0.0049s	**Não**
Peterson	20	0.0069s	0.0020s	Sim
Bipartido	20	0.3729	0.00600	**Não**

|V|	Minimo	Valor Encontrado
10	3	4
20	2	6
50	3	9
100	2	13

Table 4. Dominancia minima grafos bipartidos, em vértices

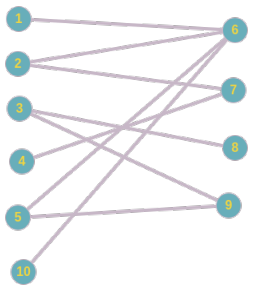

Fig. 4. Grafo bipartido de 10 vértices

Quanto maior o número de vértices (Tabela 4), pior fica a resposta dado pelo algoritmo heurístico. Enquanto o algoritmo exato é limitado em função do tempo de execução, a heurística aqui criada para grafos grandes de certas classes gera respostas distantes da mínima.

5.3.1 Analise pior e melhor caso. A heurística apresentou bons resultados para diversos grafos. O pior caso do algoritmo ocorreu no grafo do tipo Bipartido. A resposta mínima do grafo bipartido para o problema da infecção pode ser construída com os vértices da menor partição, a heurística pega os vértices pela ordem de grau e acaba não conseguindo criar uma resposta tão próxima do ótimo neste grafo. A ordem de escolha aqui acaba sendo importante, devido a natureza do grafo. O melhor caso do algoritmo é, sem dúvida, o grafo estrela, onde ele consegue encontrar a resposta com uma única interação, já que a pessoa central no grafo domina todos os outros e consequentemente é o de maior grau. Outro grafo que a heurística se saiu bem foi o grafo nulo.

Se tratando da análise temporal feita na seção 5.2. O pior caso foi com os grafos densos, onde se faz necessário um processamento de mais arestas, levando a atualização de diversos vértices a cada mudança da resposta.

6 CONCLUSÃO

Este trabalho tem como objetivo tratar a questão do espalhamento de doenças em uma rede de contato, dada a inserção de um novo patógeno em algum local. O problema de transmissão de doenças infecto-contagiosas por contato é algo que assola a humanidade desde os primórdios. Um dos fatores cruciais nessa transmissão são as pessoas já infectadas, que propagam a doença em escalas gigantes.

Para encontrar estes resultados foi decidido utilizar as mesmas abordagens adotadas para resolução do "Conjunto Dominante Mínimo". Um algoritmo baseline, que pode ser encontrado na literatura, foi implementado e testado para diversos casos de redes. Além disso, uma heurística foi desenvolvida, implementada e testada.

Foi possível ver que em uma rede esparsa de 1000 pessoas, somente 13 pessoas foram necessárias para que toda a rede fosse "dominada", ou seja, contaminada. Já na rede densa, onde as pessoas têm mais conexões, para 1000 pessoas, somente 6 pessoas são necessárias.

O algoritmo *baseline* é um algoritmo exato que roda em tempo exponencial, tornando ele completamente inviável para entradas grandes. A heurística foi criada se inspirando no conceito de grau dos vértices e visando infectar a rede com o menor numero de escolhas possíveis. Além disso, após gerada uma resposta era feito um aperfeiçoamento da resposta baseado em algumas intuições.

A heurística se mostrou satisfatório em questão de tempo de execução e conseguiu encontrar a resposta ótima para diversos casos testados. Vale ressaltar que não se existe até o momento qualquer garantia que a heurística aqui criada consegue para todos casos de uma X classe de grafo encontrar a resposta ótima.

Como trabalho futuro, o autor deseja calcular, se possível, um "fator de aproximação" para o algoritmo heurístico para grafos arbitrários. Além disso, é desejável estudar um pouco mais a natureza do problema e evoluir a heurística em uma segunda versão, mais otimizada. O algoritmo criado se encontra disponível no Github de forma Open Source sobre a licença MIT, no seguinte link: "https://github.com/saraiva3/DominatingSet".

REFERENCES

N. Bourgeois, F. Della Croce, B. Escoffier, and V.Th. Paschos. 2013. Fast algorithms for min independent dominating set. *Discrete Applied Mathematics* 161, 4 (2013), 558 – 572. https://doi.org/10.1016/j.dam.2012.01.003 Seventh International Conference on Graphs and Optimization 2010.

J.P. Byrne. 2004. *The Black Death.* Greenwood Press. https://books.google.com.br/books?id=yw3HmjRvVQMC

Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. 2005. *Exact (Exponential) Algorithms for the Dominating Set Problem.* Springer Berlin Heidelberg, Berlin, Heidelberg, 245–256. https://doi.org/10.1007/978-3-540-30559-0_21

Fedor V. Fomin and Dimitrios M. Thilikos. 2006. Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up. *SIAM J. Comput.* 36, 2 (2006), 281–309. https://doi.org/10.1137/S0097539702419649 arXiv:https://doi.org/10.1137/S0097539702419649

Renu Laskar, John Pfaff, S. M. Hedetniemi, and S. T. Hedetniemi. 1984. On the Algorithmic Complexity of Total Domination. *SIAM Journal on Algebraic Discrete Methods* 5, 3 (1984), 420–425. https://doi.org/10.1137/0605040 arXiv:https://doi.org/10.1137/0605040

Bruce Reed. 1996. Paths, Stars and the Number Three. *Combinatorics, Probability and Computing* 5, 3 (1996), 277–295. https://doi.org/10.1017/S0963548300002042

J.K. Taubenberger and D.M. Morens. 2006. 1918 Influenza: the mother of all pandemics. *Rev Biomed* 17 (2006), 69–79.

Johan M. M. Van Rooij and Hans L. Bodlaender. 2011. Exact Algorithms for Dominating Set. *Discrete Appl. Math.* 159, 17 (Oct. 2011), 2147–2164. https://doi.org/10.1016/j.dam.2011.07.001