Análise de métodos computacionais para identificação de genes significativos para o câncer

Bolsista: Lucas Alves Roris

NUSP: 11913771

Orientador: Adenilso da Silva Simão

São Carlos — Março/2022

Resumo

Câncer é um doença resultante de diversas mutações durante o ciclo de vida das células. Essas mutações podem surgir de fontes internas através da má duplicação, ou de fontes externas como a radiação solar, e podem ser divididas em *drivers*, mutações que alteram o comportamento natural da célula, e *passengers*, mutações que não ameaçam o comportamento da célula. De todas as mutações que um indivíduo desenvolve na vida, em sua maioria são *passengers*, e apenas algumas são *drivers*, porém como as *drivers* são as principais responsáveis pelo surgimento do câncer, é essencial conseguirmos diferenciar as duas. Através de avanços no sequenciamento de DNA/RNA, o custo de gerar os dados necessários para analisar mutações caiu muito, com isso, contribuíram para o surgimento de complexos bancos de dados que disponibilizam materiais para a sociedade científica analisar e interpretar os dados obtidos. Existem diversos meios de se analisar esses dados, porém os métodos computacionais são muito úteis por conseguir trabalhar com uma grande quantidade de dados rapidamente. Dentre os métodos computacionais, existe o nCOP, que utiliza uma rede biológica junto com os dados de entrada de mutações para gerar *paths* de mutações e rankeamento de genes. Esse relatório explica sobre como o nCOP funciona e apresenta propostas e implementações de melhorias e abordagens capazes de melhorar o desempenho do algoritmo.

Palavras-chave: Método computacional; câncer; genes; mutação

Sumário

1 Introdução

Câncer é uma doença causada por alterações e mutações de diversas fontes nas células. Essas mudanças ocorrem naturalmente durante a vida do indivíduo, porém algumas delas podem acabar mudando o comportamento da célula, gerando uma resposta anormal e originando o início de um câncer.

Essas mutações podem ser classificadas em *drivers*, que são mutações que alteram o comportamento da célula e *passenger*, que não interferem na função da célula. As mutações podem vir tanto de fontes internas, como má duplicação celular, ou fontes externas, como a radiação solar.

O desafio dessa área é identificar e diferenciar genes que são mais prováveis de desencadearem mutações *drivers*. Para isso, tecnologias de sequenciamento de DNA/RNA conseguem analisar as alterações genéticas nos genes e disponibilizar em um banco de dados genômicos para analise.

Através do avanço na tecnologia de sequenciamento de genes, a quantidade de dados disponíveis para serem analisados aumentou muito, e com isso, passaram a existir métodos computacionais que utilizam os dados de sequenciamento e através de alguma abordagem, tentam classificar os genes como *drivers* ou *passengers*. O método analisado neste relatório é o nCOP, um método que, através de arquivos de entrada, prioriza os genes com base em uma rede biológica.

O nCOP é um algoritmo bem documentado [2], e por conta disso é de interesse tentar melhorá-lo, tanto em desempenho, quanto em qualidade de resultados, o que é o objetivo desse relatório.

2 nCOP

O nCOP é um método computacional que busca rankear os genes com base na sua frequência em sub-redes geradas a partir de uma rede biológica já conhecida. Seu funcionamento, arquivos de entrada e arquivos de saída serão explicados a seguir.

2.1 Dados de entrada

Os dados de entrada se resumem em principalmente 3 arquivos e alguns parâmetros.

Primeiro temos que entregar uma rede biológia que simbolizará o grafo de genes, onde é um arquivo que diz dois genes que teriam uma relação. Geralmente tiramos esse arquivo de alguma base de dados conhecida, como o Reactome, mas o nCOP já disponibiliza um rede HPRD como base para ser utilizada. Por exemplo, para

representar a interação entre 4 genes, $Gene_1$, $Gene_2$, $Gene_3$ e $Gene_4$, o arquivo de entrada pode ser descrito como a seguir:

- $Gene_1$ $Gene_2$
- $Gene_1$ $Gene_4$
- $Gene_2$ $Gene_3$

Depois temos o arquivo de pacientes mutados em um gene específico. Ele é montado a partir de um arquivo *.maf* que contém mutações de diversos pacientes e em diversos genes. Porém o nCOP só utiliza a parte de mutações em genes em um paciente específico, então o arquivo se resume no nome de um gene seguido de todos os pacientes que apareceram com esse gene mutado.

- $Gene_1$ P1 P3 P4
- $Gene_2$ P1 P4
- $Gene_3$ P2 P3
- $Gene_4$ P3

Por último temos um arquivo opcional de pesos para se aplicar nos genes. É um arquivo com o nome do gene e um valor de peso, que é representado pelo número de mutações nele dividido pelo seu tamanho, então caso ele seja muito grande, é mais esperado que ele tenha uma mutação o peso dele cairá com relação aos outros. Esses pesos ajudam a diferenciar falsos positivos, já que genes grandes tendem a aparecer mutados mais frequentemente, porém ele não necessariamente é um gene *driver*.

- $Gene_1$ 0.3
- $Gene_2$ 1.1
- $Gene_3$ 3.4
- $Gene_4$ 0.04

Ademais temos alguns parâmetros possíveis de se colocar, como mudar o nome padrão do arquivo de saída e pular a parte de escolher o melhor alpha, dando um alpha específico para iniciar.

2.2 Funcionamento do algoritmo

O objetivo primário do algoritmo é encontrar um sub-grafo (G') conectado a uma rede biológica (G) que minimize a função abaixo:

$$f = \alpha \cdot X + (1 - \alpha) \cdot Size(G')$$

Onde X é a porcentagem de pacientes não cobertos pelo sub-grafo, e $Size(G')$ é o tamanho de G', e α sendo um parâmetro (de 0 a 1) de trade-off entre deixar a rede pequena e cobrir mais pacientes.

Colocando essa expressão em palavras, ao encontrar um sub-grafo G', podemos determinar se a quantidade de pacientes que ele cobre com relação ao seu tamanho vale a pena. Por exemplo, se G' conseguiu cobrir 30% dos pacientes, mas teve que incluir muitos genes para isso, a função objetivo demonstraria um valor muito alto.

Com isso dito, o que o algoritmo deve fazer é encontrar um sub-grafo G' que ao colocar na função objetivo, nos dê o menor valor possível (esse valor dependendo do α.

2.3 Expansão do sub-grafo

Para o nCOP encontrar o melhor G', ele começa com um dos cinco genes com maior número de pacientes mutados, com probabilidade proporcional ao número de pacientes mutados. A partir dele, o algoritmo analisa todos os vizinhos de todos os genes presentes em G' (atualmente apenas o primeiro), e verifica qual deles diminui mais a função objetivo. No caso de empate, é escolhido aleatoriamente pelo método de roleta, com base no número de arestas conectadas em G' que o gene em questão possúi.

Quando chegar no ponto em que não é possível uma expansão, no caso de todas as possibilidades piorarem a função objetivo, o algoritmo busca com uma distância 2 de cada vértice de G', e se caso encontre uma melhora, o algoritmo volta a tentar expandir com distância 1.

Após não conseguir expandir com distância 1 ou 2, essa parte do algoritmo retorna os vértices de G' para que o nCOP realize as devidas analises.

2.4 Constante de normalização

Analisando a função objetivo, podemos ver que na soma, o lado esquerda é uma multiplicação de um número de 0 a 1 (α) e uma váriavel de 0 a 1 (porcentagem de pacientes não cobertos), porém no lado direito é a multiplicação de um número de 0

a 1 $(1-\alpha)$ e o tamanho do sub-grafo, que é uma váriavel sem restrições de tamanho.

Dessa forma para resolver esse problema, delimita-se qual o maior valor que o sub-grafo pode alcançar que consiga cobrir todos os pacientes. Uma vez que existem pacientes mutados em vários genes, não existe apenas uma resposta exata para essa pergunta, portanto para aproximarmos o resultado, é executado o passo a passo abaixo.

Primeiro é executado o algoritmo de achar o sub-grafo com o parâmetro $\alpha = 1$, para assim, ser eliminado a importância do tamanho do sub-grafo na função objetivo, o que fará com que o algoritmo tente expandir para qualquer direção que tiver pacientes, e mesmo que não cubra 100% dos pacientes, será um valor bom o suficiente para garantir que para qualquer $\alpha < 1$, o tamanho do sub-grafo gerado seja menor que o para $\alpha = 1$. Esse passo será executado 10 vezes, já que dependendo do vértice inicial escolhido, os caminhos podem divergir.

A constante de normalização é a média das 10 iterações do tamanho do sub-grafo. Tendo a constante em mão, é adicionado na função objetivo, e considerando que esse valor é o limite para o tamanho dos sub-grafos, dividiremos o tamanho do sub-grafo em cada tentativa de expansão pela constante de normalização, assim a função objetivo estará pronta para ser executada e pode ser vista abaixo.

$$f = \alpha \cdot X + (1 - \alpha) \cdot \frac{Size(G')}{cte}$$

2.5 Parâmetro α

Pode-se notar que na função objetivo tem um parâmetro α, que simplificando, controla quantos pacientes são necessários para que valha a pena adicionar um gene no sub-grafo.

Por exemplo, caso tenhamos o $\alpha = 0.1$, o peso de se adicionar um gene fica muito alto, ou seja, o sub-grafo final tenderá a ser pequeno. Já caso o $\alpha = 0.9$, o sub-grafo tenderá a ser grande, já que o critério para adicionar um gene é muito baixo.

Esse parâmetro é muito importante para que o resultado do algoritmo não seja tendencioso, pois para cada tipo de câncer, ou tamanho da entrada, o α se moldará ao sub-grafo e reproduzirá o melhor resultado possível.

2.6 Descobrir melhor α

Explicado o funcionamento do algoritmo, para coloca-lo em prática, devemos escolher um alpha que proporcione o melhor resultado do conjunto de dados disponibilizado.

Dependendo do modo em que as mutações dos pacientes aparessem, o sub-grafo pode ficar tendencioso para otimizar a solução para apenas um grupo de pacientes. Por isso para encontrar o melhor alpha que cubra a maioria, em cada iteração, separamos os pacientes em teste (10%), validação (18%) e treinamento (72%).

É utilizado o conjunto de treinamento para chegar em um sub-grafo. Após a execução do algoritmo, é comparado a porcentagem de cobertura do conjunto de treinamento com relação a porcentagem de cobertura do conjunto de validação, eliminando assim a possibilidade do sub-grafo ser uma boa solução apenas para um grupo de dados. O conjunto de teste serve apenas para imbuir aleatoriedade no processo.

Agora para descobrir o melhor α que maximiza o resultado e minimiza o tempo de execução, rodaremos 100 vezes a iteração descrita acima para α no intervalo de 0 a 1 (por exemplo 0.1, 0.15, 0.20 ... 0.90, 0.95).

Para definir quais valores de α, seguiremos duas regras: menor α que tiver uma diferença de porcentagem de cobertura de no máximo 10% da maior porcentagem de cobertura encontrada para qualquer α. A diferença da cobertura do conjunto de treinamento e validação deve exceder 5%, para caso a aleatoriedade dividir os pacientes de maneira tendenciosa.

O menor α que cumprir essas duas regras, é selecionado para as a parte final do nCOP.

Para demonstrar como a escolha do alpha é importante para o algoritmo, abaixo há um gráfico que mostra quantos dos genes priorizados são encontrados no banco de dados de genes *drivers* do Intogen para cada alpha.

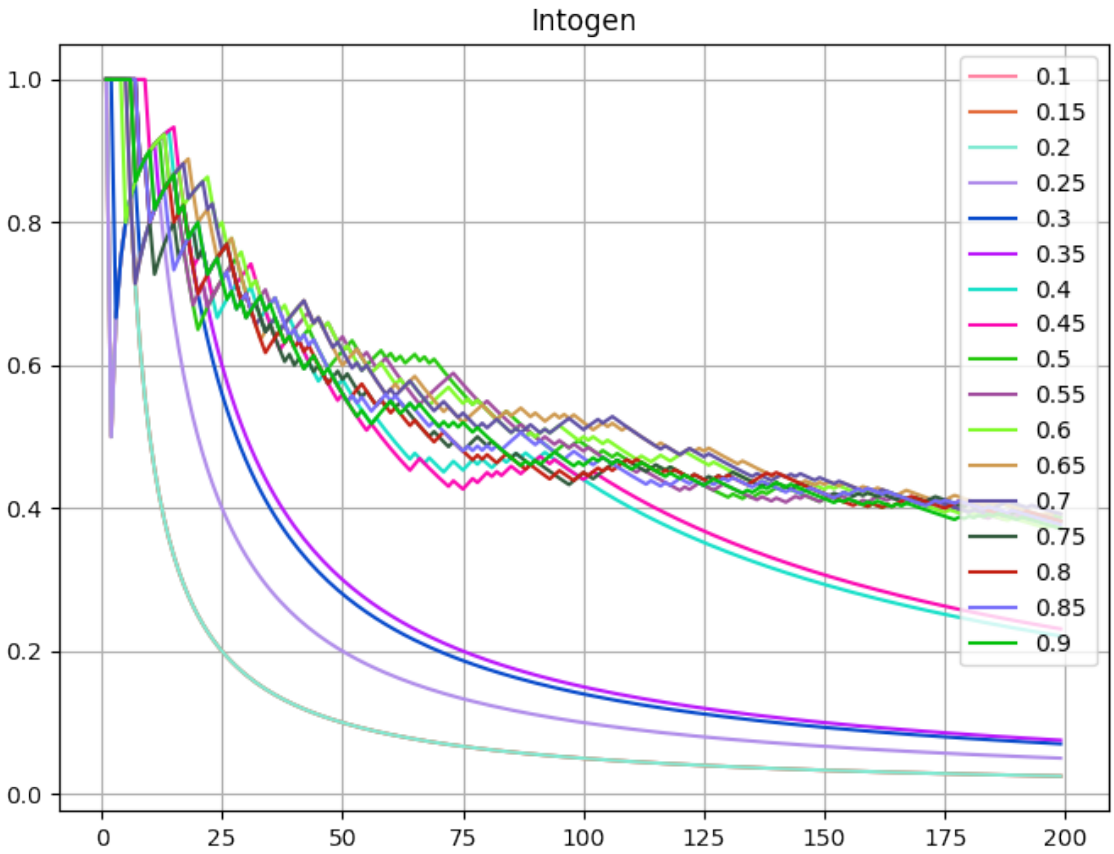

Figura 1: Interssecção de genes dos conjuntos dos alphas no Intogen

Esse gráfico foi montado verificando a porcentagem de genes priorizados que aparecem no banco de dados de genes *drivers* do Intogen. Isso nos ajuda a verificar se a priorização está coerente com a realidade.

Como podemos ver, o gráfico para alphas muito pequenos acabam decaindo muito rapidamente, já que seus sub-grafos são muito pequenos e acabam tendo pouco mais de 10 genes. Já os alphas mais promissores, ao final dos 200 genes analisados se mantiveram estáveis em 40%.

De acordo com o algoritmo nesse conjunto de dados, o melhor alpha seria o 0.65, onde realmente podemos perceber que ele se mantém em sua maior parte, acima dos outros

2.7 Iterações Finais

Com o melhor alpha em mãos, a última coisa a se fazer é descobrir quais genes mais aparecem nos *paths* gerados pelo algoritmo.

Para isso retemos 15% dos pacientes para imbutir uma aleatoriedade no processo e é rodado o algoritmo de achar o sub-grafo 1000 vezes, guardando a frequência que cada gene aparece.

Caso o parâmetro α seja passado como dado de entrada, o algoritmo pula diretamente para esse passo, mesmo que não seja considerado um bom α.

2.8 Arquivo de saída

O arquivo que o nCOP gera representa a frequência que os genes foram encontrados pelo algoritmos, ou seja, os genes que sempre aparecem nos *paths*, ficam acima, e os genes que só aparecem de vez em quando por causa de um grupo específico de pacientes, ficam abaixo.

3 Exemplo de execução

Para melhor a compreensão do algoritmo, abaixo será descrito passo a passo uma execução do algoritmo, achando a constante de normalização e fazendo a iteração da parte final, considerando o melhor alpha = 0.6 . Para facilitar o exemplo, não reteremos nenhum paciente.

Vamos supor que nossa rede biológica se resume aos genes mostrados abaixo:

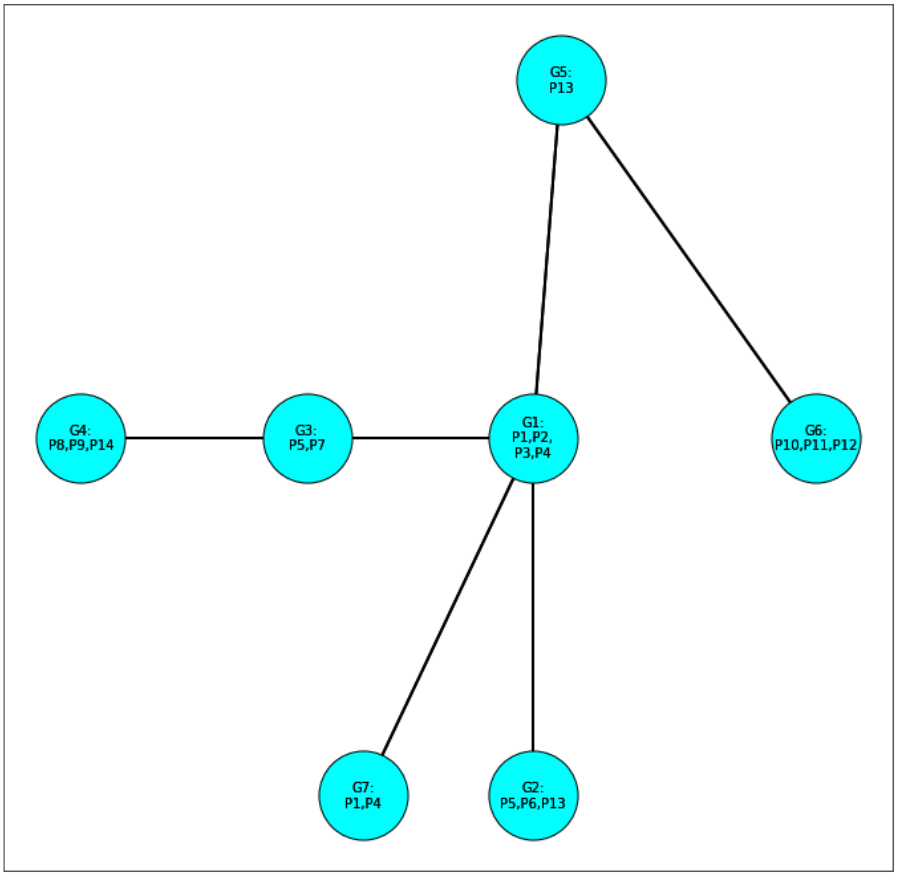

Figura 2: Rede biológica artificial

Começando com a constante de normalização, podemos começar sorteando entre os 5 vértices com maior número de pacientes, que são esses: G1, G2, G4, G6 e G3. Vamos supor que o sub-grafo comece com G1.

Como no passo a passo da constante de normalização o α escolhido como igual a 1, ao invés de verificar a função objetivo, podemos verificar apenas qual gene tem mais pacientes. Portanto a sequência do algoritmo fica:

- Olhando para o sub-grafo atual (G1), temos os vizinhos G2, G3, G5 e G7. Como o G2 é o que tem mais pacientes, ele é o que minimiza mais a função objetivo, portanto, o escolhido é o G2.

- Agora o sub-grafo é (G1, G2). Como o G2 não tem vizinhos, os nós a serem analisados são os mesmos de antes. O G7 não contribui para a função objetivo, já que seus pacientes já foram cobertos por G1. O G5 também não, já que seu paciente aparece em G2. Sobrou o G3 que contribui com 1 paciente novo.

- Agora o sub-grafo é (G1, G2, G3). Como o G5 e o G7 não podem contribuir, só sobra o G4, que possui 3 pacientes novos.

- Agora o sub-grafo é (G1, G2, G3, G4). Porém não existem mais vizinhos do sub-grafo que podem melhorar a função objetivo, portanto tentaremos verifica a uma distância de dois genes do sub-grafo, o que adiciona o par G5 e G6 na análise. Considerando o par G5/G6 o sub-grafo consegue cobrir mais 3 pacientes novos.

- Agora o sub-grafo é (G1, G2, G3, G4, G5, G6). Por fim, nem um outro vértice consegue melhorar a função objetivo, que por coincidência, conseguiu cobrir todos os pacientes, apesar de não precisar obrigatoriamente acontecer, portanto o sub-grafo ficou G1, G2, G3, G4, G5, G6. Por isso, a constante de normalização seria igual a 6.

Normalmente executamos 10 vezes o passo a passo acima para garantir uma boa cobertura de toda a rede biológica, mas para fins práticos consideraremos a média da constante de normalização igual a 6 .

Agora considerando que o melhor α seja igual a 0.6, vamos simular uma iteração do algoritmo.

- Começaremos o sub-grafo com o vértice G1 por ter o maior número de pacientes mutados.

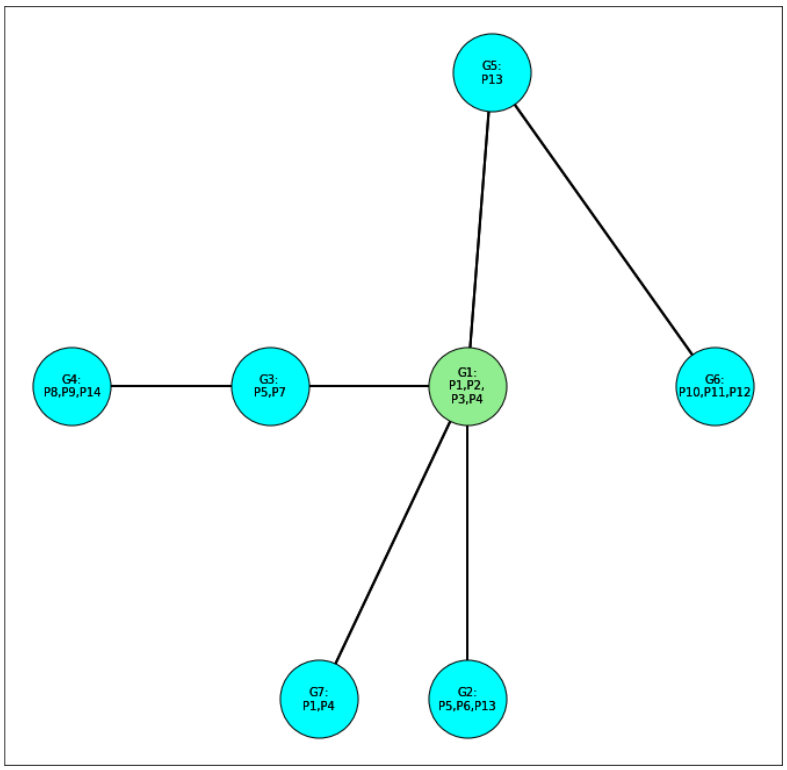

Figura 3: Rede biológica artificial expandindo

- Com o total de pacientes mutados e a constante de normalização, podemos transformar a função objetivo na fórmula abaixo:

$$f = 0.6 \cdot (1 - \frac{P}{14}) + 0.4 \cdot \frac{S}{6}$$

  Onde P é a quantidade de pacientes cobertos até agora e S o número de vértices do sub-grafo.

- Verificando todos os vizinhos do sub-grafo (G1) e aplicando a fórmula, resulta em:

  - G2: 7 pacientes diferentes cobertos com 2 vértices. Nota: 0.4333 ;
  - G3: 6 pacientes diferentes cobertos com 2 vértices. Nota: 0.4762 ;
  - G5: 5 pacientes diferentes cobertos com 2 vértices. Nota: 0.5190 ;
  - G7: 4 pacientes diferentes cobertos com 2 vértices. Nota: 0.5190 .

- É possível ver que adicionando o G2 no sub-grafo resulta na melhor minimização da função objetivo. Portanto será adicionado o G2 no sub-grafo.

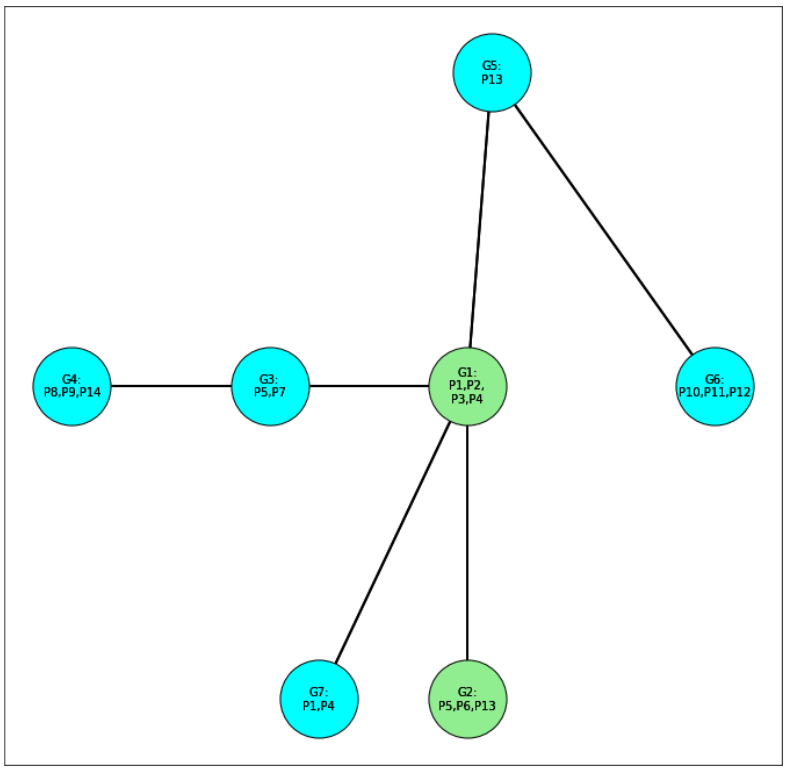

Figura 4: Rede biológica artificial expandindo

- Agora o sub-grafo é (G1,G2). Como G2 não possui vizinhos além de G1, os vértices analizados são os mesmos acima:
  - G3: 8 pacientes diferentes cobertos com 3 vértices. Nota: 0.4571 ;
  - G5: 7 pacientes diferentes cobertos com 3 vértices. Nota: 0.5 ;
  - G7: 7 pacientes diferentes cobertos com 3 vértices. Nota: 0.5 .
- Como podemos ver, não foi possível diminuir mais a função objetivo, já que a nota atual do sub-grado é 0.4333, por isso agora será verificado em uma distância de dois genes do sub-grafo.
  - (G3, G4): 11 pacientes diferentes cobertos com 4 vértices. Nota: 0.3952 ;
  - (G5, G6): 10 pacientes diferentes cobertos com 4 vértices. Nota: 0.4381 .

  Com isso pode-se adicionar o par (G3, G4), já que eles conseguiram minimizar mais do que o sub-grafo atual.

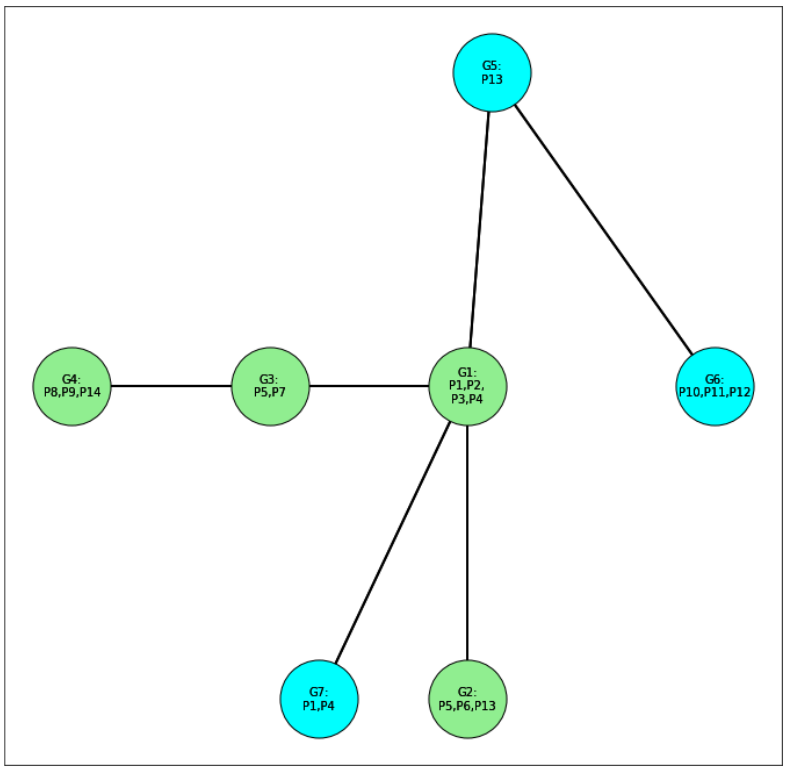

Figura 5: Rede biológica artificial final

- Agora a analisar volta a verificar numa distância 1 do sub-grafo:
  - G5: 11 pacientes diferentes cobertos com 5 vértices. Nota: 0.4619 ;
  - G7: 11 pacientes diferentes cobertos com 5 vértices. Nota: 0.4619 .
- Não foi possível diminuir mais a função objetivo, por isso agora é feito a análise em uma distância de dois genes do sub-grafo.
  - (G5, G6): 14 pacientes diferentes cobertos com 6 vértices. Nota: 0.4 .
- Agora percebe-se que mesmo olhando numa distância 2 não foi possível expandir o sub-grafo, portanto o sub-grafo final é (G1, G2, G3, G4).

Esse caso acima acontece se começarmos com qualquer um dos genes G1, G2 e G4. Porém caso o primeiro gene escolhido fosse o G6, realizando a mesma analise feita acima, o sub-grafo final seria (G6, G5, G1, G2, G3, G4). O que significa os genes G1 e G2 aparecem independente do gene inicial.

Portanto, se fossemos rodar 1000 vezes, considerando que cada um dos genes iniciais aparecessem 200 vezes, o arquivo de saída seria esse:

- G1: 100%
- G2: 100%
- G3: 80%
- G4: 80%
- G5: 20%
- G6: 20%

G1 e G2 aparecem com 100% pelos motivos acima. G3 e G4 com 80% por não aparecerem nas 200 vezes onde o vértice inicial é o G6. E o G5 e G6 aparecem com 20% por aparecem apenas nas 200 vezes que o vértice inicial é G6.

4 Modificações

Foi realizado adaptações do código fonte disponibilizado pelo artigo com intuito de melhor tanto o desempenho, quanto os resultados.

4.1 Mudança de linguagem

O código fonte original é feito em uma linguagem de programação chamada Ruby. Essa é uma linguagem matemática, dinâmica com foco na simplicidade da sintaxe.

Na adaptação, foi utilizado uma linguagem de programação chamada Python [5]. Uma linguagem simples, porém muito otimizada, que juntando as bibliotecas mais famosa como Numpy [1], MatPlotLib [3], Pandas [4], etc, é possível conseguir ótimos resultados.

A adaptação foi feito principalmente com base no artigo, onde o autor explicou como o algoritmo deveria funcionar, e ao longo da implementação, houve diversas consultas ao código fonte para compreender melhor o passo a passo.

Apenas essa modificação na linguagem, o código já foi capaz de resolver a priorização mais rapidamente produzindo os mesmos resultados. Para comprovar isso, rodamos o algoritmo com um alpha de 0.6 e, tirando todas as partes aleatórias, o sub-grafo gerado era sempre o mesmo.

Para realizar o teste do tempo de execução, foi utilizado a rede biológica HPRD e a base de pacientes mutados KIRC, ambos são disponibilizados junto dos arquivos do nCOP. Com os mesmos dados de entrada, verificamos que para o nCOP levou cerca de 76 minutos, enquanto o código em python levou cerca de 50 minutos, uma redução no tempo de aproximadamente 35%. Esse teste foi realizado após todas as abordagens de aceleração.

4.2 Acelerando abordagem de união de pacientes

Quando o método vai avaliar um sub-grafo, ele verifica quantos pacientes foram cobertos e coloca na fórmula, porém como existem pacientes que aparecem em mais de um gene, deve ser feito um filtro, para acabar não duplicando os pacientes e dar uma nota falsa.

Normalmente essa operação é bem custosa, já que cada paciente a ser contado deve passar por todos os outros, afim de verificar se ele já apareceu alguma vez. Em Python existem métodos para resolver esse problema de maneiras otimizadas.

Um dos item feitos nessa mudança é usar o conceito de verificar que a união de dois conjuntos é a soma do seus tamanhos menos a sua intersecção. Descobrir o tamanho de dois conjuntos de pacientes é trivial, e a intersecção o Python possui um método que acha usando processamento paralelo, o que deixa muito mais rápido. Com essa operação resolvida, o algoritmo performa as iterações muito mais rápido.

4.3 Apagar pacientes já cobertos

Após a otimização de contar os pacientes cobertos, foi descoberto que apagar do grafo todos os pacientes cobertos em cada iteração, deixava o código mais rápido, já que ao verificar novos genes, só teriam pacientes não cobertos, portanto não precisaria verificar a intersecção, apenas o tamanho da lista é o suficiente.

Com essa modificação, dependendo do conjunto a ser testado, a melhora pode ser de 15% a menos no tempo de execução comparando os mesmos dados de entrada com e sem essa abordagem de otimização.

4.4 Permitir arquivo maf de entrada

O nCOP, como explicado em seções anteriores, necessita de um arquivo de pacientes mutados em um gene específico, que normalmente é gerado a partir de um arquivo .maf, disponível em portais de compartilhamento de dados de mutações.

Porém normalmente deve-se transformar o arquivo maf em um arquivo texto apenas com as informações que o nCOP usa. Para o usuário sem experiência pode ser algo complicado, custoso e demorado caso existam muitos arquivos para analisar.

Para que o usuário não precise transformar o .maf em um arquivo só para a execução do nCOP, foi implementada a opção de inserir o arquivo maf diretamente e o próprio programa processa as informações que ele precisa para ser executado, gerando o arquivo texto para caso seja executado mais de uma vez.

4.5 Relatório pós execução

O Arquivo de saída padrão do nCOP é um .txt com a porcentagem da frequência de cada gene ao gerar os sub-grafos, o que representa o ranqueamento dos genes. Porém para uma analise mais profunda, podem ser necessárias outras informações que o algoritmo gera e que não são disponibilizadas.

Com isso em mente, outra mudança foi a opção de gerar um relatório de execução do nCOP, que consta tempos de cada etapa, gráficos, decisões, etc.

5 Conclusão

O câncer é uma doença perigosa que pode surgir do nada, e dado os métodos de tratamento atuais, conseguir distinguir o começo de um câncer é imprescindível para a eficácia do tratamento. Portanto as mutações *drivers* e *passengers* são muito dificeis de distinguir em seu estágio inicial, por isso contamos com a ajuda de métodos computacionais para a detecção dessas mutações.

O nCOP é um algoritmo utilizado nessas analises, permitindo um rankeamento dos genes mais prováveis de carregarem uma mutação *driver*. Seu código fonte e sua especificação estão bem documentados [2], portanto foi de interesse analisar e recriar o algoritmo, procurando ações que pudessem aperfeiçoar o método.

Através do artigo explicando o funcionamento do algoritmo, foi possível refazê-lo mais otimizado, performático e prático, com o auxílio de diversas técnicas da computação, permitindo uma maior facilidade a usuários da ferramenta para fins de pesquisa.

Testes mostraram uma redução de até 35% do tempo pela mudança na linguagem de programação e abordagem de análise. Melhorando também a praticidade de permitir a entrada de um arquivo maf e gerando um relatório final com informações possivelmente úteis.

O código juntamente com esse relatório estão disponíveis na plataforma GitHub no link: `https://github.com/LucasRorisCube/nCOPPython.git`. O repositório conta com um arquivo *README.md* que explica os detalhes de execução do algoritmo.

Referências

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van

Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[2] Singh M. Hristov BH. *Network-Based Coverage of Mutational Profiles Reveals Cancer Genes.* Cell Syst., 2017.

[3] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[4] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[5] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.