RENATO SILVA DE MELO

MAXIMIZAÇÃO DE INFLUÊNCIA EM GRAFOS LEI DE POTÊNCIA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Renato José da Silva Carmo
Coorientador: Prof. Dr. André Luís Vignatti

CURITIBA

2016

RENATO SILVA DE MELO

MAXIMIZAÇÃO DE INFLUÊNCIA EM GRAFOS LEI DE POTÊNCIA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Renato José da Silva Carmo
Coorientador: Prof. Dr. André Luís Vignatti

CURITIBA

2016

Melo, Renato Silva de
Maximização de influência em grafos lei de potência / Renato Silva de Melo. - Curitiba, 2016
65 f. : il.; tabs.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Renato José da Silva Carmo
Coorientador: André Luís Vignatti
Bibliografia: p.63-65

1. Algoritmos. 2. Teoria dos grafos. 3. Redes sociais.
I. Carmo, Renato José da Silva. II. Vignatti, André Luís. III. Título

CDD 511.5

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós Graduação em INFORMÁTICA
Código CAPES: 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **RENATO SILVA DE MELO**, intitulada: "**Maximização de Influência em Grafos Lei de Potência**", após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO.

Curitiba, 20 de Maio de 2016.

Prof RENATO JOSÉ DA SILVA CARMO
Presidente da Banca Examinadora (UFPR)

Prof ANDRÉ LUIS VIGNATTI
Coorientador - Avaliador Interno (UFPR)

Prof EDUARDO TODT
Avaliador Interno (UFPR)

Prof JAIME COHEN
Avaliador Externo (UEPG)

AGRADECIMENTOS

O agradecimento principal e mais especial de todos é em memória da minha amada mãe, dona Maria dos Santos, que tanto se esforçou para que eu me tornasse um homem de verdade. Os ensinamentos e o amor que em vida me deu, bem como seus valores e princípios, fizeram de mim a pessoa que sou hoje. Se eu pudesse voltar a estar com ela, repetiria infinitamente o quanto eu a amo, mas como não posso, busco agradecer não apenas com palavras, mas tentando diariamente ser a melhor pessoa que ela lutou para que eu fosse um dia. Assim tento honrar a sua memória.

À minhas duas "irmãzinhas", Sângela e Sônia, que foi o que me restou de família. A vontade de lhes educar e poder mudar nossas vidas, certamente, foi o que me motivou a não ter medo de evoluir e melhorar nossa existência. Com certeza, vocês fazem parte de mais esta conquista, que vai muito além do título de mestre.

Quero agradecer a todos os meus amigos que de alguma forma fizeram parte desta trajetória. Pelas noites que vagávamos pelo centro do Curitiba, buscando conhecer a cidade, novos bares e novas pessoas. Pelas comemorações com churrasco e karaokê que sempre fizemos. Não poderia esquecer as animadas rodas de violão e tereré que fazíamos depois do almoço no pátio do Dinf. Assim como as reuniões do café da tarde, sempre regadas de muitas gargalhadas. O que aprendemos em momentos como estes não foi escrito em textos científicos e nem avaliado em termos formais, mas com toda certeza terá significado permanente em nossas relações. Tentei listar (em ordem alfabética) os nomes daqueles que fizeram parte desta fase, como segue. Alane Marie, Alisson Puska, Benevid felix, Carlos Zava, Camile Frazão, Clariane Menezes, Daniel Rezende, Danilo Faria, Edgar Cabral, Elaine Rocha, Elisabete Ferreira, Elisângela Rocha, Ítalo Brilhante, Ivan Pires, Maurício Barros, Otto Júlio, Patrícia Ternus, Santiago Viertel, Válber Zacarkim, Vanessa Fernandes (e família).

Ao professor André Vignatti, que além de orientador foi um amigo, sou imensamente grato pela oportunidade e pelo voto de confiança para que eu pudesse aprender tanto o quanto aprendi neste mestrado.

Graças a essas pessoas, que estiveram com sinceridade torcendo para que este desafio fosse vencido, eu vivo um momento tão significativo.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O problema de maximização de influência em redes sociais, procura pelos vértices que permitam espalhar uma informação para o maior número possível de membros da rede. Um algoritmo guloso proposto por Kempe *et al.* [19], que escolhe iterativamente os vértices de maior influência, encontra um conjunto resposta cujo alcance da influência é pelo menos $1 - 1/e$ do ótimo. Mas esta abordagem possui alguns agravantes que comprometem o tempo de execução do algoritmo. Nesta dissertação propomos uma melhoria para o algoritmo guloso de Kempe *et al.* [19] com foco no tempo de execução deste sobre grafos de lei de potência. A melhoria consiste em fazer uma seleção prévia dos vértices mais promissores. Verificamos por meio de análise experimental que esta pré seleção reduz expressivamente o tempo de execução, além de manter a qualidade da solução compatível com a do algoritmo guloso de Kempe *et al.* [19]. A otimização por pré seleção utiliza propriedades presentes em grafos de lei de potência, explorando a relação de influência social com a distribuição de graus. Esta abordagem reduz em até 58% o tempo de execução do algoritmo CELF [22], que é uma das otimizações mais conhecidas do procedimento de Kempe *et al.* [19].

ABSTRACT

The influence maximization problem in social networks, asks for the nodes that allow to spread a information for the highest number of members. A greedy algorithm proposed by Kempe *et al.* [19], that choose iteratively the members of greatest influence, find a solution set whose the reach of influence is at least $1 - 1/e$ of the optimum. But some aggravating have affecting the runtime of this approach. In this work we propose improvements for the algorithm of Kempe *et al.* [19], with focus on the runtime in power law graphs. The improvement consists in make a previous selection of most promising nodes. We have verified by experimental analysis that this pre selection reduces significantly the runtime, in addition maintaining a quality compatible with the greedy algorithm of Kempe *et al.* [19]. The optimization by pre selection uses properties present in power law graphs, by exploring the relationship between social influence and the degree distribution. Besides, this approach reduces the runtime of CELF [22] algorithm by up to 58%, which is one of the most known optimizations of the algorithm of Kempe *et al.* [19].

CAPÍTULO 1

INTRODUÇÃO

O fenômeno da difusão de ideias, comportamentos e inovações em redes sociais, possui a propriedade de sempre ter início com um pequeno grupo de *adeptos iniciais* [21]. A partir deste grupo, cada vez mais pessoas adotam o mesmo comportamento ao observar que seus amigos, vizinhos ou colegas já o fizeram, assim uma informação se espalha como uma epidemia. O estudo do processo de difusão surgiu nas ciências sociais [31, 30], ganhou espaço com modelos matemáticos [16, 17] e atualmente existem vários estudos que seguem uma abordagem algorítmica deste problema [9, 19, 22, 23, 5, 15].

Um problema decorrente da investigação dessa influência social é a *maximização de influência*, que surge no contexto de adoção em cadeia de novos comportamentos [9] . Uma das motivações iniciais para estudos de maximização de influência foi o marketing viral [9], onde a ideia é fornecer algum produto para um grupo de indivíduos, que através da disseminação "boca a boca" possa resultar em um grande número de adoções do produto [19]. Outras aplicações são em detecção de surtos [22], redução da violência de gangues [32], previsão de novas tendências [2], entre outros. Informalmente, o problema da maximização de influência tem como objetivo encontrar um conjunto S de tamanho fixo, tal que a influência do conjunto S seja a maior possível.

Partindo do pressuposto que características estruturais influenciam na difusão de informação em redes sociais, investigamos este problema em redes complexas cuja distribuição de graus dos vértices siga uma lei de potência. Assim, com base nessas características, desenvolvemos um algoritmo que escolhe os vértices mais influentes de maneira eficiente, com foco principal no tempo de execução e em qualidade de otimização. Parte da dissertação se concentra em descrever esta abordagem com mais detalhes.

1.1 Metodologia

Optamos por projetar um algoritmo baseado em simulações Monte Carlo para solucionar o problema, pois esta abordagem proporciona uma boa precisão nas estimativas realizadas no algoritmo. Dentre os modelos de difusão estudados, escolhemos o de cascata independente. Este modelo de difusão foi escolhido por que se apresentou mais apropriado para a proposta, nos aspectos de implementação e simulação, como é descrito no Capítulo 2. Além disso, comparamos experimentalmente o desempenho do algoritmo resultante com outros algoritmos para identificação de indivíduos influentes, assim, mostramos que nossa solução promove bons resultados.

Utilizamos nos experimentos grafos lei de potência reais e gerados artificialmente. Com

grafos gerados artificialmente podemos ajustar os experimentos e, desta forma, verificar os resultados em diferentes tamanhos e escalas de densidade. Com grafos reais buscamos resultados mais concretos. Optamos por implementar nossos próprios algoritmos para gerar grafos de lei de potência e, para isso, usamos modelos já estabelecidos na área [1, 7, 34].

1.2 Contribuições

Como nos empenhamos para contribuir com a pesquisa da área, e conseguir melhorias para as atuais soluções do problema, ficamos satisfeitos em listar a seguir algumas destas contribuições.

- Heurística eficiente para seleção prévia: apresentamos um método de escolha dos vértices mais influentes que reduz o tamanho da entrada. Além do mais, fornecemos uma análise teórica sobre a qualidade da pré seleção e o tempo de execução desta heurística.

- Otimização para o algoritmo guloso de Kempe *et al.* [19] em grafos lei de potência: nos experimentos realizados com grafos lei de potência, a heurística de pré seleção em conjunto com a otimização CELF reduz de 20 a 58% o tempo de execução do próprio CELF, que é uma das melhorias mais expressivas do algoritmo guloso. Consequentemente, isso aumenta a viabilidade de utilização do algoritmo guloso de Kempe *et al.* [19] em grafos desse tipo.

- Algoritmos para gerar grafos lei de potência: propomos um algoritmo para geração de grafos aleatórios com distribuição de lei de potência, fundamentado no modelo de geração de Aiello *et al.* [1], aliado ao modelo de geração de grafo aleatório generalizado de Chung *et al.* [7].

- Documentação e implementação: além da proposta do algoritmo, buscamos proporcionar neste documento um "passo-a-passo" para o estudo da difusão de informações, fornecendo conceitos importantes sobre os modelos matemáticos de difusão e técnicas algorítmicas utilizadas nos trabalhos da literatura, tais técnicas e modelos formam os fundamentos para pesquisas mais profundas. O código fonte[1] do ambiente implementado também contribui neste sentido, uma vez que todo um *framework* foi desenvolvido para que se pudesse simular a difusão de informação em redes sociais.

[1]Disponível em: gitlab.c3sl.ufpr.br/rsmelo/influence_maximization

1.3 Organização

A dissertação foi dividida em duas partes, nas quais, a primeira parte reúne explicação do problema, trabalhos relacionados e os requisitos para entendimento da solução que está sendo proposta, que são os Capítulos 2 - 5. A segunda parte, abrange os Capítulos 6, 7 e 8, onde é descrita a proposta e sua análise, assim como o modelo de grafos aleatórios implementado, seguido pelas avaliações e resultados obtidos.

No Capítulo 2 são apresentados os conceitos necessários para o entendimento do problema, incluindo os passos para modelar a difusão de informação. O Capítulo 3, contém a definição do problema de maximização de influência e o algoritmo guloso proposto por Kempe *et al.* [19]. No Capítulo 4 são descritos os trabalhos relacionados considerados mais relevantes dentre os levantados nesta dissertação, em seguida são apresentados os modelos dos grafos usados neste trabalho, no Capítulo 5. No Capítulo 6 propomos a otimização por pré seleção. Os algoritmos para geração dos grafos artificiais são apresentados no Capítulo 7. No Capítulo 8 descrevemos como foram feitos os experimentos e avaliações do algoritmo proposto em comparação com outros algoritmos.

Parte I

Modelos Teóricos, Maximização de Influência e Lei de Potência

CAPÍTULO 2

MODELOS DE DIFUSÃO PROBABILÍSTICA

Neste capítulo é feita uma introdução aos princípios fundamentais que deram origem aos modelos matemáticos para representar uma difusão de informações, uma vez que a maximização de influência depende destes modelos teóricos para a formulação do problema. Inicialmente, utilizamos conceitos de teoria dos jogos e redes sociais até chegar nos modelos básicos de difusão que usam princípios probabilísticos para simular a propagação de informações em uma rede.

2.1 Difusão de Informação

O entendimento de como funciona o processo de difusão foi construído ao longo de um trabalho empírico da sociologia, intitulado *Diffusion of Innovations* [30]. Este trabalho reuniu alguns princípios comuns que se aplicam a diferentes domínios, além de um conjunto de motivos que podem causar a falha de uma inovação ao se espalhar por uma população. Para Rogers [30], difusão é o processo pelo qual uma determinada "inovação" é transmitida através de certos canais ao longo do tempo entre membros de um sistema social, no qual os participantes criam e compartilham informações uns com os outros a fim de chegar a um entendimento mútuo. Podemos considerar que uma inovação é um novo comportamento, ideia, opinião e tecnologia que se espalha em uma rede social. Assim, no decorrer da dissertação usamos o termo informação para fazer referência uma determinada inovação.

Em redes sociais, a análise da difusão é feita por meio de grafos, para observar como alguns membros são influenciados por outros, explorando as escolhas individuais que levam estes a tomarem decisões semelhantes aos seus vizinhos. Uma das razões pelas quais imitar o comportamento dos outros pode ser bom é o *benefício direto* [11, 18], em que existe uma recompensa direta em copiar as decisões dos outros, por exemplo, vantagens ao utilizar tecnologias compatíveis em vez de incompatíveis. Assim, ao explorar os princípios que levam as pessoas a imitar o comportamentos dos seus vizinhos, é possível modelar o processo da tomada de decisão das pessoas em uma rede social.

2.2 Princípios dos Modelos de Difusão

Como indivíduos tomam decisões baseados nas escolhas dos seus vizinhos, um padrão de comportamento pode começar a se espalhar através das arestas de uma rede [11]. Assim, os modelos baseados em efeito de benefício direto consideram que as vantagens de adotar

um comportamento novo aumentam a medida que os vizinhos de um indivíduo também o adotam. Neste caso, o simples interesse próprio do indivíduo vai fazer ele adotar o novo comportamento, já que uma fração suficiente dos seus vizinhos também o fizeram. Easley e Kleinberg [11] capturam esta ideia em um *jogo*, da teoria dos jogos, como descrito na Seção 2.2.1

2.2.1 Tomada de Decisão Individual

As regras que definem a tomada de decisão de cada membro da rede são estabelecidas da seguinte maneira. A rede social é modelada por um grafo $G = (V, E)$, onde os vértices são os membros da rede e as arestas indicam quando existe algum relacionamento entre estes membros. Cada vértice tem uma escolha entre dois comportamentos, A ou B. Se $\{v, w\} \in E$, então supõe-se que existe incentivo para que v e w tenham um comportamento condizente. Tal comportamento é representado usando um jogo, onde v e w são jogadores e A e B são as estratégias. Neste trabalho, o termo *payoff* expressa a recompensa que um jogador obtém por adotar uma determinada estratégia no jogo. Para os dois jogadores, se ambos adotam o comportamento A, cada um recebe *payoff* de $a > 0$, se ambos adotam B, o *payoff* de cada um é $b > 0$, se adotam comportamentos opostos o *payoff* é 0 para os dois, onde termo *payoff* representa a recompensa recebida por escolher determinada estratégia. A Tabela 2.1 mostra a matriz de *payoffs* para este jogo.

		w	
		A	B
v	A	a, a	0, 0
	B	0, 0	b, b

Tabela 2.1: Matriz de payoffs dos vértices v e w.

Todo vértice v executa uma cópia deste jogo em cada uma de suas arestas, o *payoff* total de v é a soma dos resultados do jogo com todo w adjacente a v. Assim, as preferências de v serão baseadas nas escolhas feitas por seus vizinhos. Saber o que v deve fazer para maximizar seu *payoff* depende do número de vizinhos adotando cada uma das estratégias e a relação entre os valores de *payoff* a e b.

Considerando que alguns dos vizinhos de v estão adotando um comportamento A e outros utilizando B, v se encontra em situação de escolher qual comportamento adotar, a fim de maximizar seu *payoff*. Desta forma, a regra de decisão para v é como segue. Seja p a fração dos vizinhos de v que adotou o comportamento A e seja $(1 - p)$ a fração que adotou o comportamento B. Assim, se v tem d vizinhos, então pd adotam A e $(1 - p)d$ adotam B, como é mostrado na Figura 2.1. Se v escolhe A, obtém *payoff* de pda e se

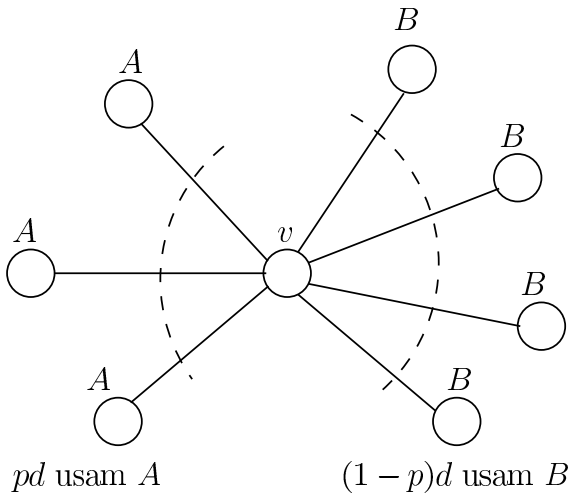


Figura 2.1: O vértice v tem que escolher entre os comportamentos A e B, baseado nas escolhas de todos seus d vizinhos, onde p é fração dos vizinhos de v que usam A e $1-p$ os que adotaram B [11].

escolhe B, obtém $(1-p)db$. Assim, A é a melhor estratégia se

$$pda \geq (1-p)db$$

ou, rearranjando os termos,

$$p \geq \frac{b}{a+b}.$$

Seja q a expressão da direita, a inequação diz que se pelo menos uma fração $q = \frac{b}{a+b}$ dos vizinhos de v segue o comportamento A, então ele deve fazer isso também. Quando q é pequeno, A é o comportamento mais tentador, pois precisa apenas de uma pequena fração dos vizinhos engajados para que v também use A. Analogamente, se q é grande acontece o oposto, B é o mais atrativo e é preciso muitos vizinhos engajados em A para que v use A. Neste sentido, q funciona como um limitante para a mudança de comportamento.

O exemplo a seguir apresentado por Kleinberg *et al.* [11] ilustra como funciona a regra de decisão. Considere a rede social da Figura 2.2, supondo que no jogo entre A e B os payoffs são de $a = 3$ e $b = 2$. Pela regra de decisão, isso nos dá um *limitante* $q = 2/(3+2) = 2/5$. Além disso, os vértices 7 e 8 são os *adeptos iniciais* do comportamento A, sendo que os outros usam B. No primeiro passo da execução do processo, os vértices 5 e 10 mudarão para A, porque $2/3 > 2/5$ dos seus vizinhos estão usando A. Da mesma forma, no passo seguinte 4 e 9 mudarão, e por último o 6 também adota A. A partir deste ponto, nenhum outro vértice da rede estará disposto a mudar, chegando assim no resultado da Figura 2.3.

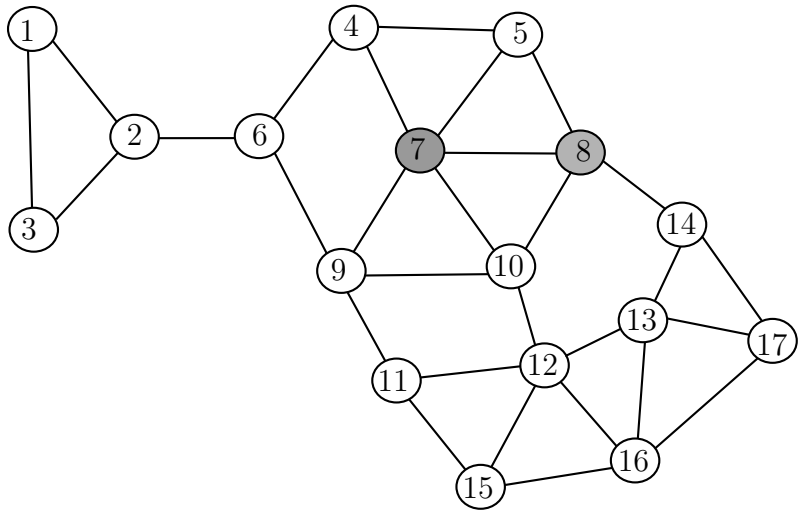


Figura 2.2: Os vértices 7 e 8 são os adeptos iniciais do comportamento A [11].

2.2.2 Equilíbrios do Jogo

Ainda em termos de teoria dos jogos, existem dois equilíbrios de Nash[1] para o jogo da Tabela 2.1, um é em que todos adotam A e o outro em que todos assumem B [11]. Mas para que uma *cascata completa* aconteça na rede, todos os vértices que inicialmente usam B tem que passar a usar A. Além disso, outros equilíbrios intermediários podem surgir, isto é, no final do processo podem existir vértices com comportamentos diferentes.

Considere que todos os vértices da rede estão usando B como comportamento padrão. Então, por alguma razão externa ao jogo, um subconjunto de vértices decide usar A. Todos os outros continuam avaliando seus *payoffs* usando a matriz de *payoffs*. Como exemplificado nas Figuras 2.2 e 2.3, o processo de adoção é uma reação em cadeia, ou seja, dado o fato de que existem vértices usando A, alguns dos seus vizinhos podem decidir trocar para A também. Logo, alguns vizinhos dos vizinhos também podem e assim por diante, gerando um possível cascateamento. Por isso, é importante lembrar que existe a possibilidade da adoção de A parar depois de um tempo. A reação de mudanças para A é uma cascata de adoções e distingue-se entre duas possibilidades [11]:

i) executa por um tempo, mas pára enquanto ainda existem vértices usando B, ou

ii) uma cascata completa em que todos os vértices da rede mudam para A

As duas possibilidades dependem da estrutura da rede, da escolha dos adeptos iniciais e do valor do limitante q que os vértices usam para decidir se mudam para A.

A homofilia[2] pode muitas vezes servir como barreira para a difusão, uma vez que

[1]Equilíbrio de Nash representa uma situação na qual nenhum jogador tem incentivo para mudar de estratégia [11]

[2]Homofilia é a tendência que as pessoas têm de interagir mais com outras que são semelhantes a elas [11].

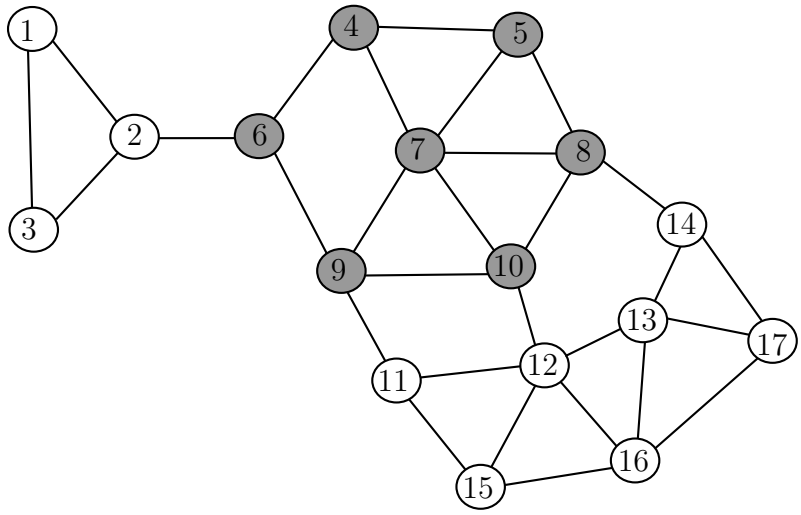


Figura 2.3: A cascata tem início, mas pára [11]. Os vértices marcados são aqueles que usam A.

uma das possíveis consequências da homofilia é o surgimento de agrupamentos, que informalmente, são comunidades fortemente conectadas [18]. Easley e Kleinberg [11] demonstram formalmente que estes agrupamentos não apenas podem dificultar a chegada de informações de fora das comunidades, como também são os únicos obstáculos para que ocorra uma cascata completa.

Até agora mantemos o modelo de comportamento individual o mais simples possível, ou seja, todos os vértices tem o mesmo *payoff* e a mesma intensidade de interação com seus vizinhos. Mas podemos fazer suposições mais ricas preservando os mesmo pontos básicos, adicionando sutilezas como pesos nas arestas, limitantes heterogêneos, *payoffs* individuais, etc.

2.3 Modelos Básicos

Existem na literatura vários modelos teóricos para a difusão informações em redes sociais. Neste documento, mantemos o foco nos dois modelos utilizados por Kempe *et al.* [19] na proposta do problema de maximização de influência, que são *limitante linear* e *cascata independente*.

Para descrever os modelos vamos considerar que uma rede social é definida como um grafo $G = (V, E)$, onde V é o conjunto de indivíduos na rede e E é o conjunto de relacionamentos entre estes indivíduos. Assim como é feito em [19, 21], os comportamentos modelados aqui são *progressivos*, ou seja, cada vértice pode assumir um entre dois estados, *ativo* ou *inativo* e pode mudar de inativo para o ativo, mas não de ativo para inativo. Esta característica progressiva pode ser entendida com mais clareza na descrição do processo de difusão, nas Seções 2.3.1 e 2.3.2.

Dizemos que um vértice está ativo se ele segue o comportamento A e inativo se segue B. No tempo $t = 0$, um subconjunto S de V é escolhido como um conjunto de adeptos iniciais. Quando um vértice v se torna ativo por causa de S, dizemos que S tem influência sobre v. De agora em diante consideramos apenas grafos *direcionados*, tal que para dois vértices v e w a influência de v para w seja diferente da influência de w para v.

Por completude as definições dos dois modelos de influência serão dadas, no entanto, manteremos o foco apenas nas propriedades do modelo de cascata independente, pois é este que utilizamos na análise teórica da melhoria que está sendo proposta e nos experimentos.

2.3.1 Limitante Linear

Abreviado como LT do inglês *Linear Threshold*, neste modelo, assume-se um limitante distinto em cada vértice. Lembrando que o limitante é a fração de vizinhos necessária para um vértice adotar um novo comportamento. Assim, no modelo de limitante linear, se destacam dois ingredientes [21]:

- Um peso não negativo $b_{v,w}$ em cada aresta (v, w), indicando a influência que v exerce sobre w, satisfazendo a condição

$$\sum_{v \in N(w)} b_{v,w} \leq 1,$$

  onde $N(w)$ denota o conjunto de vizinhos de w.

- Cada vértice w tem um limitante q_w escolhido aleatoriamente do intervalo $[0, 1]$. O limitante q_w indica a fração dos vizinhos de w que devem adotar o comportamento antes que w o faça.

A dinâmica da difusão opera como já explicado na Seção 2.3. Os vértices em S começam o processo como ativos e todos os outros como inativos. O tempo passa em passos discretos $t = 1, 2, 3, ...$ Em um dado momento t, um vértice inativo w se torna ativo caso sua fração de vizinhos ativos, denotada por $N^a(w)$, exceda o limitante:

$$\sum_{v \in N^a(w)} b_{v,w} \geq q_w.$$

O exemplo da Figura 2.4 pode ilustrar como esse processo funciona. Supondo que w tem um limitante de $q_w = 0,5$, uma vez que $\sum_{x \in N(w)} b_{x,w} = 0,3 + 0,4 \geq q_w$, w é ativado no próximo passo. Note que diferentes limitantes para os vértices indicam distintas predisposições à se tornar ativo. Para um vértice w um pequeno q_w indica uma abordagem mais liberal para adoções, enquanto um grande q_w espera uma fração maior dos seus vizinhos que já estejam ativados.

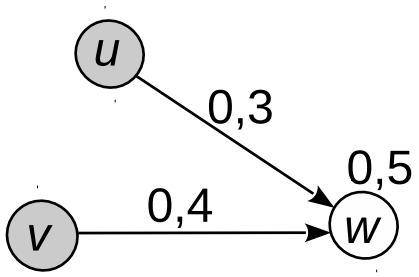


Figura 2.4: Ativação do vértice w no modelo LT.

2.3.2 Cascata Independente

Nas Seções 2.2 e 2.3.1 descrevemos modelos para difusão de informação estritamente em termos de limitantes de vértices. Mas, além destes, Kempe *et al.* [19] consideram modelos de cascata baseados em sistemas de teoria da probabilidade. Assim, em arestas (u, v) tal que u é ativo e v não, a ativação não depende apenas de u, mas também dos outros vizinhos de v que já foram ativados. Se u tem sucesso ao tentar ativar v, então v agora tenta ativar algum dos seus vizinhos de saída, mas se u falha, então u entra para o conjunto de vértices que falharam ao tentar ativar v.

Um modelo conceitualmente simples deste tipo é o modelo de Cascata Independente, muitas vezes abreviado como IC (*Independent Cascade*). Grosseiramente falando, no modelo IC cada aresta tem uma probabilidade de ativação e a influência é propagada por vértices ativos, ativando de forma independente os seus vizinhos inativos, baseado na probabilidade de ativação das arestas [5]. Assim como no modelo LT, o processo de adoção se inicia através de um conjunto S de vértices ativos, e se desenrola em passos discretos de tempo. Quando o vértice v se torna ativo no passo t, é lhe dada uma chance de ativar cada vizinho w inativo, com probabilidade $p_{v,w}$ de sucesso (parâmetro de entrada do modelo). Se v tem sucesso, então w será ativo no passo $t+1$, mas se v não tem sucesso, não pode fazer mais nenhuma tentativa de ativar w nas rodadas subsequentes [19].

Por exemplo, na Figura 2.5 w é ativado por u e v com probabilidades $p_{u,w} = 0,3$ e $p_{v,w} = 0,4$, respectivamente. Supondo que u e v foram ativados no tempo t. Assim, no tempo $t+1$, u e v podem ativar w de maneira independente, e portanto, w é ativado com probabilidade $p_{u,w} + p_{v,w} - p_{u,w} \cdot p_{v,w}$, ou seja, $0,3 + 0,4 - (0,3 \cdot 0,4) = 0,58$.

Vale observar que os modelos de propagação Cascata Independente e Limitante Linear são probabilísticos. No modelo IC, sorteios aleatórios decidem se um vértice ativo tem sucesso ao ativar outros ou não. No modelo LT, os limitantes dos vértices são escolhidos aleatoriamente de maneira uniforme, tal como o peso de influência dos vizinhos ativos. Assim, em ambos os modelos podemos pensar na propagação como um resultado de um processo aleatório.

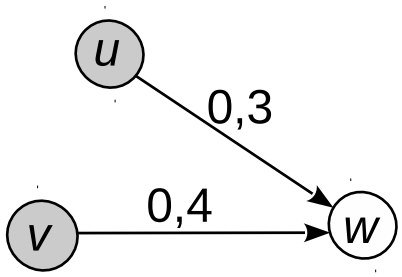


Figura 2.5: Ativação do vértice w no modelo IC.

2.4 Sobre a Escolha do Modelo

Com base nas características listadas a seguir, escolhemos neste estudo o modelo de cascata independente para representar o processo de difusão.

Entre os principais pontos levados em consideração estão:

i) Preparação das instâncias: Não existem muitas restrições quanto à distribuição dos pesos nas arestas, os grafos para as simulações podem ser produzidos atribuindo pesos de influência aleatoriamente. Sendo que no modelo LT não é tão simples assim, uma vez que os pesos atribuídos para toda aresta (v, w) tem que respeitar a restrição $\sum_{v \in N(w)} b_{v,w} \leq 1$ para cada vértice w, além disso, o limitante de cada vértice também precisa ser definido.

ii) Implementação e simulação: O processo de difusão se dá por um sorteio em cada aresta que decide se um vértice influencia outro, e essa verificação acontece apenas uma vez pra cada aresta. Por outro lado no modelo LT é necessário que a soma dos pesos das arestas que chegam de vértices ativos, seja maior do que o limitante de cada vértice w, isso dificulta a implementação da difusão.

iii) Tempo discreto: Todas as etapas do processo de difusão acontecem em tempo discreto, e cada vértice ativo tenta ativar apenas uma vez cada um dos seus vizinhos de saída.

iv) Heterogeneidade de preferências: Neste ponto o modelo LT tem vantagem, uma vez que cada vértice avalia sua decisão com base não apenas na quantidade de vizinhos ativos, mas também com base na sua credulidade (limitante individual) no que está prestes a adotar. Enquanto no modelo IC a probabilidade de um vértice v se tornar ativo aumenta conforme mais vizinhos de v são ativados, mas como não tem limitantes, não tem heterogeneidade de preferências.

v) Análise teórica: Como o modelo IC é conceitualmente simples, a análise teórica do processo aleatório de difusão é mais fácil devido a independência das probabilidades na ativação das arestas.

Finalmente, a discussão mostra como o modelo básico de difusão baseado em um simples jogo de coordenação apresentado na seção 2.1 é passível de extensões que capturam recursos adicionais de situações reais onde a difusão pode ter lugar. Mesmo pequenas extensões como as consideradas até aqui, podem apresentar novas fontes significativas de complexidade, e o desenvolvimento de extensões ainda mais ricas é uma área aberta de pesquisa.

CAPÍTULO 3

O PROBLEMA DE MAXIMIZAÇÃO DE INFLUÊNCIA

Em um dos trabalhos mais conhecidos sobre maximização de influência, Kempe *et al.* [19] sintetizam uma questão inicialmente levantada no artigo de Domingos e Richardson [9], da seguinte forma: *"Para convencer indivíduos de uma rede social a adotar uma nova ideia, quais indivíduos deveriam ser o alvo principal da propaganda, para que se inicie uma cascata de adoções?"*. Isso deu origem a uma série de trabalhos que buscam tratar esse problema de maneira eficiente e sistemática.

Depois que Domingos e Richardson [9] abordaram o problema de forma algorítmica, em 2011, Kempe *et al.* [19] o formularam como um problema de otimização, onde o objetivo é basicamente encontrar o melhor conjunto de adeptos iniciais em uma rede social, seja no modelo de limitante linear ou cascata independente. Este conjunto tem tamanho fixo e deve ser o que proporciona maior alcance das adoções de uma informação sobre toda a rede.

No decorrer do capítulo apresentamos a definição formal do problema, suas características intrínsecas e o algoritmo guloso padrão de Kempe *et al.* [19], que tem razão de aproximação $1 - \frac{1}{e}$.

3.1 Definição do Problema

Seja m um modelo de propagação, tal como o modelo de cascata independente ou limitante linear. Para qualquer instância do modelo m, existe uma *função de influência* σ_m, conforme formalizado na Definição 1 [13, 19, 21]:

Definição 1 (Função de Influência)**.** *A função de influência σ_m é uma função tal que $\sigma_m : 2^V \to \mathbb{R}$, onde dado $S \subseteq V$ o conjunto de adeptos iniciais, $\sigma_m(S)$ denota o número esperado de vértices ativos no final do processo de ativação partindo de S.*

O Problema 1 define formalmente o problema da maximização de influência, que é o foco deste trabalho.

Problema 1 (Maximização de Influência)**.** *São dados um grafo direcionado $G = (V, E)$, um modelo de propagação m e um inteiro $1 \leq k \leq |V|$. O objetivo é encontrar um subconjunto $S^* \subseteq V$ de tamanho k, tal que $\sigma_m(S^*) = \max\{\sigma_m(S)$ tal que $|S| = k, S \subseteq V\}$.*

Em ambos os modelos de propagação, IC e LT, este problema é NP-difícil [19] e Kempe *et al.* [19] apresentam um algoritmo guloso que garante uma aproximação com um fator $1 - \dfrac{1}{e}$.

3.2 Propriedades da Função de Influência

Seja U um conjunto e seja f uma função arbitrária definida como $f : 2^U \to R^+$. Dizemos que f é *submodular* se possui a propriedade de diminuição de ganho, isto é, se o ganho obtido ao adicionar um determinado elemento a um conjunto S é pelo menos tão alto quanto adicionar o mesmo elemento a um superconjunto de S [19], como mostra a Definição 2. A mesma função f também é *monotônica*, quando o ganho marginal da função é não decrescente [19], como mostra a Definição 3.

Definição 2. *Sejam S, T e U conjuntos tais que $S \subseteq T \subseteq U$ e $f : 2^U \to R^+$. Dizemos que f é* submodular *se $f(S \cup \{w\}) - f(S) \geq f(T \cup \{w\}) - f(T)$ para todo $w \in U \setminus T$.*

Definição 3. *Sejam S, T e U conjuntos tais que $S \subseteq T \subseteq U$ e $f : 2^U \to R^+$. Dizemos que f é* monotônica *(não decrescente) se $f(S) \leq f(T)$.*

Para obter a garantia de aproximação, Kempe *et al.* [19] provam que a função de influência σ é submodular e monotônica para os modelos IC e LT. Em termos da função σ de propagação esperada, seja S o conjunto de adeptos iniciais e $\sigma(S \cup \{w\}) - \sigma(S)$ o *ganho marginal* de adicionar um novo vértice w ao conjunto S. Submodularidade diz que o ganho marginal de um novo vértice encolhe a medida que o conjunto cresce [14]. No mesmo sentido, monotonicidade diz que a medida que mais vizinhos de algum vértice u se tornam ativos, a probabilidade de u ser ativado aumenta.

3.3 Algoritmo Guloso $(1 - 1/e)$-aproximado

Por causa das propriedades descritas na Seção 3.2, um algoritmo que escolhe iterativamente o vértice com maior ganho marginal em cada iteração, pode ser suficientemente bom.

Algoritmo 1: Algoritmo Guloso

Entrada: $G = (V, E, b), k \in \mathbb{N}, \sigma_m$
Saída: Conjunto semente S

início
  $S = \emptyset$
  enquanto $|S| \leq k$ **faça**
    seleciona $u = \arg\max_{w \in V \setminus S}\{\sigma_m(S \cup \{w\}) - \sigma_m(S)\}$
    $S \leftarrow S \cup \{u\}$.
  fim
fim
retorna S

O Algoritmo 1 mostra um algoritmo guloso para qualquer função σ_m que seja submodular e monotônica. Os parâmetros de entrada são um grafo G, um inteiro k e uma

função σ_m, e a saída é um conjunto S de k vértices. O único laço do algoritmo executa k iterações, a operação mais complexa está na linha 4, onde em cada iteração i é selecionado o vértice que fornece o maior ganho marginal com relação a propagação esperada do conjunto S_i.

Seja S a saída do Algoritmo 1 e S^* o conjunto que tem maior influência dentre todos os conjuntos possíveis de tamanho k em V. Este algoritmo garante que $\sigma_m(S) \geq (1 - 1/e) \cdot \sigma_m(S^*)$ [19, 27]. Essa taxa de aproximação é considerada relativamente boa, por se tratar de um algoritmo simples. No entanto, existem duas principais fontes de ineficiência neste algoritmo. Uma é que o tempo de processamento da função $\sigma_m(S)$ é alto demais por ser um problema difícil de resolver nos modelos LT e IC [19, 4, 6], e a outra é que o algoritmo faz muitas chamadas ao cálculo desta função. Tais agravantes são discutidos com mais detalhes na Seção 3.6.

3.4 Simulação da Propagação no Modelo IC

Seja $G = (V, E, b)$ uma rede no modelo cascata independente e S o conjunto de adeptos iniciais. Para simular a propagação de influência no modelo de IC, realizamos a travessia do grafo por meio de uma busca em largura a partir de S. Nesta busca, cada aresta (u, v) é atravessada com probabilidade $p_{u,v}$.

Algoritmo 2: CASCATA

Entrada: $G = (V, E, b), S \subseteq V$
Saída: Conjunto A de vértices ativos

```
1  início
2      A ← ∅
3      para cada u ∈ S faça
4          Enfileira u em Q
5      fim
6      enquanto Q ≠ ∅ faça
7          Desenfileira u de Q
8          A ← A ∪ {u}
9          para cada v ∈ N^s(u) \ A faça
10             Enfileira v em Q com probabilidade p_{u,v}
11         fim
12     fim
13 fim
14 retorna A
```

Como o nome sugere, o Algoritmo 2 faz uma cascata de ativações, isto é, simula o processo de propagação de informação. A entrada consiste de um grafo $G = (V, E, b)$ e um subconjunto $S \subseteq V$, a saída é o conjunto A de vértices ativos ao final do processo, note que $A \supseteq S$. Inicialmente todos os vértices de S são inseridos na fila Q, nas linhas

3-4. Depois, cada vértice u retirado de Q é ativado e tenta ativar todos os seus vizinhos de saída no próximo passo, onde $N^s(u) \setminus A$ denota todos os vizinhos de saída de u que ainda não estão ativos. Quando a fila fica vazia o processo termina e o tamanho do conjunto A é o total de vértices que foram ativados por S. Observe que este algoritmo atravessa no máximo todas as arestas do grafo, onde o pior caso ocorreria se houvesse ativação de todos os vértices. Portanto, termina em $O(|E|)$ passos.

3.5 Estimando a Propagação no Modelo IC

Como calcular o valor exato de $\sigma_{IC}(S)$ é computacionalmente difícil [4, 6], é preciso fazer uma estimativa deste valor. Uma forma de estimar esse valor é obtendo uma média do número de vértices ativados em diversas execuções do Algoritmo 2.

Supondo que utilizamos o Algoritmo 2 para calcular a propagação de um conjunto S no grafo G, a cada execução de CASCATA(G, S) ele pode retornar saídas A diferentes, pois as ativações dependem de probabilidades, logo, $|A|$ é uma variável aleatória cujo valor esperado é $\sigma_{IC}(S)$. Assim, podemos ter mais precisão sobre o valor de $\sigma_{IC}(S)$ se forem feitas várias execuções do Algoritmo 2 e obter uma média de todas as execuções. Dessa forma, a estimativa da propagação de um conjunto é obtida pela soma

$$\frac{1}{r}\sum_{i=1}^{r}|A_i|,$$

onde r o número de execuções do Algoritmo 2 e A_i é o resultado do processo na i-ésima execução.

Esta técnica de estimar um valor por meio de simulações é referenciada na literatura de análise probabilística como *Método Monte Carlo*. Mitzenmacher e Upfal [25] definem este método como uma coleção de ferramentas para estimar valores através de amostragem e simulação. Observe que no Algoritmo 1, em cada iteração i do laço, é escolhido o vértice com maior ganho marginal em relação ao conjunto S_i, onde S_i é o conjunto S na iteração i. Para isso é preciso calcular o ganho marginal de todos os vértices que não estejam em S_i. Esse valor é alcançado fazendo a estimativa da propagação esperada, que repete r chamadas à função que simula a difusão (Algoritmo 2, no nosso caso).

3.6 Deficiências do Algoritmo Guloso

Na primeira deficiência do algoritmo, o cálculo exato de $\sigma_m(S)$ foi deixado como um problema em aberto em [19] e provada mais tarde por Chen *et al.* [4, 6] que é #P-difícil nos modelos Cascata Independente [4] e Limitante Linear [6]. Por isso, é necessário obter uma estimativa do valor de $\sigma_m(S)$. O algoritmo de Kempe et al. [19] faz uso de uma técnica conhecida como *simulações Monte Carlo*. Nesta técnica, um número muito grande

(normalmente 10.000 [5, 22, 14]) de repetições do processo de ativação é realizado até que se obtenha uma precisão razoável do valor de $\sigma_m(S)$, tanto para o modelo IC como para o LT.

A segunda deficiência do Algoritmo 1 é consequência das muitas chamadas ao procedimento que estima σ_m, no qual, o cálculo é realizado para todos os $|V \setminus S_i|$ vértices em cada iteração, onde S_i é o conjunto semente no i-ésimo passo do laço. Como neste algoritmo a estimativa de σ_m é feita por meio do método Monte Carlo, o custo de fazer tantas simulações se torna elevado demais para redes de grande escala.

A proposta deste trabalho é atacar a segunda deficiência, ou seja, buscamos reduzir o tempo de execução do algoritmo diminuindo o número de chamadas ao procedimento que faz a estimativa da função $\sigma_m(S)$. Alguns algoritmos como LDAG [6] e SIMPATH [15] tratam a primeira deficiência e são discutidos no Capítulo 4. O Capítulo 6, fornece detalhes sobre a nossa proposta de solução.

CAPÍTULO 4

TRABALHOS RELACIONADOS

Para superar a ineficiência do Algoritmo 1, alguns estudos posteriores foram desenvolvidos com o objetivo de aprimorá-lo e reduzir seu custo computacional. Conforme observado por Goyal *et al.* [14] muitos deles estudam o problema da maximização de influência a partir de duas direções complementares. A primeira é a melhoria do algoritmo guloso original, que busca reduzir o seu tempo de execução, onde dois algoritmos (CELF e CELF++) se destacam por fornecer bons resultados, estes são descritos na Seção 4.1. A segunda direção é propor novas heurísticas que aperfeiçoam o cálculo da propagação de influência, mais detalhes sobre esta abordagem são fornecidos na Seção 4.2.

4.1 Baseados em Simulações Monte Carlo Para Propagação Esperada

As soluções apresentadas nesta seção aplicam técnicas algorítmicas mais elaboradas ao Algoritmo 1, com o objetivo de reduzir o número de chamadas ao procedimento que estima σ usando simulações Monte Carlo.

4.1.1 Algoritmo Celf

Nesta abordagem, a característica de submodularidade da função de influência foi explorada novamente, desta vez para se chegar a uma solução de maneira mais eficiente por meio da seleção gulosa. Este algoritmo utiliza uma otimização chamada de *Lazy-Forward* para seleção dos vértices mais influentes. Foi proposto originalmente para resolver o problema em redes cujos vértices possuem custos atrelados a sua escolha, neste caso, devem ser escolhidos vértices de melhor custo benefício. Por isso é chamado de *Cost-effective Lazy Forward* (CELF). De acordo com Leskovec *et al.* [22], no caso em que os vértices não possuem custos ou em que todos os custos são iguais, é mantido o fator de aproximação do Algoritmo 1.

A ideia principal é que o ganho marginal de um vértice em uma determinada iteração, não pode ser maior do que ganho do mesmo vértice nas iterações anteriores (submodularidade) [26]. O algoritmo mantém uma lista de vértices ordenada de maneira não crescente pelo ganho marginal dos vértices. Assim, seja S o conjunto semente em alguma iteração da busca. O valor de $\sigma(S)$ é reavaliado apenas para o vértice do inicio da fila em cada passo, esta fila é reordenada somente quando necessário. Se um vértice continua no topo

da lista, ele será selecionado como o próximo vértice semente. Normalmente, uma *heap* Q é empregada como fila de prioridade para manter a lista ordenada.

Algoritmo 3: CELF

```
   Entrada: G = (V, E, b), k, σ_m
   Saída: Conjunto semente S
1  início
2      S ← ∅, Q ← ∅
3      para cada u ∈ V faça
4          u.mg ← σ({u}); u.it ← 0
5          Adiciona u à Q em ordem não crescente de mg
6      fim
7      enquanto |S| < k faça
8          Desenfileira u de Q
9          se u.it = |S| então
10             S ← S ∪ {u}
11         fim
12         senão
13             u.mg ← σ_m(S ∪ {u}) − σ_m(S)
14             u.it ← |S|
15             Reinsere u em Q e reordena
16         fim
17     fim
18 fim
19 retorna S
```

O Algoritmo 3 apresenta o esqueleto da otimização com CELF. Neste, o elemento de Q que corresponde a um vértice u, armazena uma dupla da forma $\langle u.mg, u.it\rangle$, onde $u.mg = \sigma(S \cup \{u\}) - \sigma(S)$ representa o ganho marginal de u com relação ao conjunto S, enquanto $u.it$ marca qual a iteração que $u.mg$ foi atualizado pela última vez. No início do algoritmo, linhas 3-7, o ganho marginal de cada vértice u é calculado e adicionado à Q em ordem não crescente de ganho marginal. Depois, em cada uma das k iterações, u é removido da fila e é verificado se já teve seu ganho marginal calculado na iteração atual, usando o atributo *it*. Se sim, devido à submodularidade, u é o vértice de maior ganho marginal na iteração atual, assim, ele será selecionado como o próximo vértice semente (linhas 10-13). Caso contrário, nas linha 14-18, é recalculado o ganho marginal de u e ele é reinserido em Q de modo que a ordem seja mantida.

Esta alteração evita o cálculo repetitivo do ganho marginal de todos os vértices em cada iteração, com exceção da primeira. Portanto, o CELF é melhor do que o Algoritmo 1 em tempo de execução, porque reduz o número de chamadas ao procedimento que estima $\sigma(S)$, por isso reduz o tempo do algoritmo guloso em até 700 vezes [22].

4.1.2 Algoritmo Celf++

Goyal *et al.* [14] propõe novos ajustes ao CELF. As configurações são similares ao Algoritmo 3, uma *heap* Q guarda os vértices. Mas ao invés de uma dupla, o elemento de Q que corresponde ao vértice u, armazena uma tupla da forma $\langle u.mg1, u.prev\_best, u.mg2, u.it\rangle$[14]. Deste modo, seja $\delta_u(S)$ o ganho marginal de u com relação ao conjunto S. Então $u.mg1 = \delta_u(S)$ e $u.prev\_best$ é o vértice que tem máximo ganho marginal entre todos os vértices examinados na iteração atual, antes de u. A variável $u.mg2$ representa $\delta_u(S \cup \{prev\_best\})$ e $u.it$ guarda a iteração em que $u.mg1$ foi atualizado pela última vez.

Algoritmo 4: OTIMIZAÇÃO COM CELF++

Entrada: $G = (V, E, b), k, \sigma_m$
Saída: Conjunto semente S

início
  $S \leftarrow \emptyset, Q \leftarrow \emptyset$, $last\_seed \leftarrow NULL, cur\_best \leftarrow NULL$
  para cada $u \in V$ **faça**
    $u.mg1 \leftarrow \sigma_m(\{u\})$; $u.prev\_best \leftarrow cur\_best$
    $u.mg2 \leftarrow \delta_u(cur\_best)$; $u.it \leftarrow 0$
    Adiciona u a Q
    Atualiza cur_best baseado em $u.mg1$
  fim
  enquanto $|S| < k$ **faça**
    Desenfileira u de Q;
    se $u.it = |S|$ **então**
      $S \leftarrow S \cup \{u\}$
      $last\_seed \leftarrow u$
    fim
    senão se $u.prev\_best = last\_seed$ **então**
      $u.mg1 \leftarrow u.mg2$
    fim
    senão
      $u.mg1 \leftarrow \delta_u(S)$; $u.prev\_best \leftarrow cur\_best$
      $u.mg2 \leftarrow \delta_u(S \cup \{cur\_best\})$
    fim
    $u.it \leftarrow |S|$
    Atualiza cur_best
    Reinsere u em Q e reordena.
  fim
fim
retorna S

A ideia central é que se o vértice selecionado na última iteração ainda é a raiz do *heap* Q, não é preciso recalcular os ganhos marginais. Fazendo isso, é possível economizar algumas chamadas a função σ, e consequentemente, reduzir o tempo de processamento [14]. O algoritmo é implementado de modo que $\delta_u(S)$ e $\delta_u(S \cup \{prev\_best\})$ sejam avaliados

simultaneamente em uma única iteração do procedimento de simulação Monte Carlo. Os resultados dos experimentos mostram uma melhoria de 17-61% sobre o CELF [14].

No Algoritmo 4 a variável *last_seed* armazena o índice do último vértice escolhido, *cur_best* guarda o vértice com maior ganho marginal entre todos os examinados naquela iteração. Então este continua selecionando vértices sementes até chegar a k. O ajuste proposto está nas linhas 15-17, onde se atualiza $u.mg1$ sem recalcular o ganho marginal. Isso pode ser feito desde que $u.mg2$ já tenha sido calculado com relação ao ultimo vértice selecionado. Se nenhum dos casos acima são aplicados, ele então recalcula o ganho marginal de u, assim como é feito no CELF.

As melhorias no Algoritmo de Kempe *et al.* [19] obtidas tanto com o CELF++ como CELF, apesar de obterem um ganho de desempenho, ainda não são escaláveis, conforme é relatado em [6]. As heurísticas apresentadas na Seção 4.2, buscam superar esta deficiência.

4.2 Baseados em Heurísticas para a Função de Influência

As heurísticas descritas nesta seção são mais eficientes do que as implementações mais rápidas do Algoritmo 1 (CELF e CELF++, por exemplo), mantendo um nível competitivo de propagação de influência [6, 15]. Estas, buscam estimar de maneira mais eficiente a função de propagação de influência σ, ao invés de focar na diminuição do número de chamadas à função, como era o objetivo dos algoritmos 3 e 4. Neste caso, algumas propriedades do modelo de influência Limitante Linear foram fundamentais para a criação destas melhorias.

4.2.1 Algoritmo LDAG

Proposto por Chen et al. [6], este algoritmo usa uma estrutura local para tornar tratável a computação da propagação de influência σ no modelo LT. O ganho em eficiência com esta heurística se deve por limitar a computação às regiões locais dos vértices. Isto garante mais escalabilidade na solução do problema. A ideia principal se estabelece em um algoritmo que faz, em tempo linear, o cálculo exato da propagação de influência em grafos acíclicos direcionados (DAG - *Directed Aciclic Graphs*). Mas o uso deste algoritmo requer manipulações nas vizinhanças locais dos vértices para se chegar em resultados aproximados da propagação nestas regiões.

A estimativa de σ_{LT} em DAGs é a base para a solução do problema de maximização. Considere $S \subseteq V$ um conjunto semente em D, onde $D = (V, E, b)$ é um DAG com pesos $b \in [0, 1]$ nas arestas. Seja $ap(v)$ a probabilidade de v ser ativado em D no modelo de limitante linear dado S, para todo $v \in V$. Chen *et al.* [6] demonstra o Lema 1 e o utiliza como base para construir o algoritmo que calcula $ap(v)$ em tempo linear.

Lema 1. *Para todo $v \in V \setminus S$, temos*

$$ap(v) = \sum_{u \in V \setminus \{v\}} ap(u) \cdot b(u, v). \tag{4.1}$$

De maneira resumida, o algoritmo para DAGs ordena topologicamente todos os vértices que são alcançáveis por S em uma sequência ρ. Depois calcula $ap(v)$ para todo $v \in \rho \setminus S$ usando a Equação 4.1. A ordenação topológica garante que quando v é calculado, todos os vértices que apontam para v já foram calculados antes. Por isso, este algoritmo é rápido [6]. No entanto, como um grafo nem sempre é um DAG, logo, não é interessante aplicar o Lema 1 diretamente em qualquer rede.

Para se obter uma estimativa da influência de maneira eficiente em grafos gerais, Chen *et al.* [6] sugere determinar um DAG local para cada vértice v do grafo. Dessa forma, a influência de v é obtida por meio da DAG local. A ideia intuitiva para justificar isso é que, primeiro, a computação de σ_{LT} em DAGs é fácil, segundo, normalmente a cascata de influência no modelo LT acontece na vizinhança, devido à probabilidade decrescente da propagação de influência [6]. Dessa forma, Chen *et al.* [4] chegam a uma estrutura chamada *modelo de influência LDAG*, onde cada vértice v é associado com uma DAG local, denotada por $LDAG(v)$, que é um DAG local com raiz em v. Dado S, assume-se que a influência de S para v se propaga dentro da $LDAG(v)$ em conformidade com o modelo LT. Seja σ_D a propagação de influência de S no modelo LDAG, o cálculo de σ_D é dado por

$$\sigma_D(S) = \sum_{v \in V} ap(v, S).$$

Dado G e LDAGs com raiz em todos os vértices, maximizar $\sigma_D(S)$ ainda é NP-Difícil [4]. Dado S, obter $\sigma_D(S)$ leva tempo polinomial porque todos os cálculos são feitos em DAGs. Por fim, conforme Chen *et al.* [4], o algoritmo resultante para selecionar k sementes possui tempo total de $O(nt + knm \log n)$, onde $n = |V|$, m é o número máximo de arestas de $LDAG(v)$ para todo $v \in V$, e t é a média de tempo para processar uma $LDAG(v)$ entre todo $v \in V$. Pelos resultados empíricos em [6], o LDAG é um algoritmo bom para o problema de maximização de influência no modelo LT, porque possui um bom desempenho e é escalável para grafos de milhões de vértices e arestas.

4.2.2 Algoritmo Simpath

De acordo com Goyal *et al.* [15], o SIMPATH supera o LDAG em desempenho, consumo de memória e propagação de influência alcançada. O algoritmo combina o uso do CELF (Algoritmo 3) com heurísticas para o cálculo da propagação, sem fazer uso de simulações Monte Carlo.

Seja G um grafo direcionado. Goyal *et al.* [15] mostra que a propagação esperada de

um conjunto S pode ser obtida pela soma da propagação de todo $u \in S$ em subgrafos induzidos por $(V \setminus S) \cup \{u\}$. Com base nesta afirmação, o procedimento de cálculo da estimativa σ_{LT} do SIMPATH utiliza a ideia de que a propagação de influência pode ser calculada pela enumeração dos caminhos simples que saem dos vértices em S.

Uma vez que o problema de enumerar caminhos simples em um grafo é #P-Difícil [15], o SIMPATH também faz uso da ideia de que a maior parte da influência de um vértice se concentra em uma pequena vizinhança e pode ser capturada pela enumeração dos caminhos nesta vizinhança. Assim uma "poda" é realizada nos caminhos, isto é, eles são desprezados sempre que o tamanho atingido é inferior a um limitante de poda. Este limitante é basicamente o erro que pode ser tolerado, isso representa um meio termo entre a precisão da propagação estimada e o tempo de execução do algoritmo [15].

Com o objetivo de reduzir o número de chamadas ao procedimento de estimativa de propagação, Goyal *et al.* [15] propõe a otimização por cobertura de vértices. Esta melhoria usa o fato de que, quando se possui os valores de propagação de todos os vizinhos de saída no subgrafo induzido por $V \setminus \{v\}$, para qualquer v, então é possível calcular diretamente a propagação de v. Dessa forma, no início da execução do algoritmo, o conjunto V é dividido em dois subconjuntos C e $V \setminus C$, onde C é a cobertura de vértices de um grafo não direcionado G', obtido ignorando as direções das arestas de G e removendo as arestas duplicadas. Assim, é possível calcular simultaneamente, para todo vértice $u \in C$, a propagação tanto em G, como no subgrafo induzido por $V \setminus \{v\}$, para cada vizinho de saída v de u que estiver presente em $V \setminus C$.

O algoritmo final faz uso das três melhorias descritas. A primeira otimização é enumerar os caminhos simples para estimar $\sigma_{LT}(S)$, mas para isso é preciso usar a segunda otimização, que é fazer a poda no tamanho dos caminhos simples, isso melhora a eficiência da enumeração. Por último, o número de chamadas à estimativa de $\sigma_{LT}(S)$ é reduzido por meio da otimização por cobertura de vértices.

O pseudocódigo final tanto do SIMPATH como do LDAG não foram mostrados neste estudo por serem ambos muito extensos. Assim, mais detalhes sobre o código, por hora, sairiam do escopo do trabalho. Por isso, optamos apenas por ilustrar a ideia principal por trás destes.

O diferencial destas duas heurísticas é que ambas evitam simulações Monte Carlo e empregam técnicas algorítmicas sofisticadas, baseadas principalmente nas propriedades probabilísticas do modelo LT. Embora as duas tenham apresentado evolução na eficiência, a qualidade destes resultados são observáveis apenas empiricamente, pois não possuem garantia de aproximação. Principalmente por esse motivo, propomos no Capítulo 6 uma solução baseada em simulações Monte Carlo. Esta solução combina as melhorias conquistadas pelo CELF com uma proposta de *pré seleção*, e dessa forma, o algoritmo guloso pode ficar ainda mais eficiente de modo que a seu fator de aproximação de $1 - \frac{1}{e}$ seja mantido.

Vale lembrar que nos experimentos realizados no final deste trabalho, dentre os algoritmos descritos neste capítulo, comparamos nossa solução apenas com o CELF, uma vez que nossa solução usa o método baseado em simulações Monte Carlo no modelo IC, enquanto SIMPATH e LDAG são algoritmos para o modelo LT.

CAPÍTULO 5

GRAFOS LEI DE POTÊNCIA ALEATÓRIOS

Neste capítulo finalizamos a parte do referencial teórico, no qual apresentamos as definições do modelo de rede estudado no trabalho. Dado que o escopo da proposta se concentra em resolver o problema da maximização de influência em grafos cuja distribuição de graus segue uma lei de potência.

5.1 Grafos Aleatórios

Neste estudo consideramos grafos em que as arestas são geradas por meio de mecanismos aleatórios. O modelo mais simples de grafos aleatórios é o modelo de Erdös-Rényi, que foi apresentado em uma série de estudos entre os anos de 1950 e 1960 [28]. A ideia consiste em dois modelos muito parecidos que são denotados por $G_{n,m}$ e o $G_{n,p}$. No primeiro deles, cada possível grafo com n vértices e m arestas é escolhido com a mesma probabilidade. O segundo modelo, $G_{n,p}$, é obtido ao adicionar de maneira independente cada uma das $\binom{n}{2}$ arestas possíveis do grafo com probabilidade p.

Resultados recentes mostram que diversas redes do mundo real não podem ser modeladas através do modelo de Erdos-Rényi [12]. Em consequência, novos modelos matemáticos foram propostos para geração de grafos aleatórios de redes reais. Um dos resultados mais interessantes obtidos em análises estruturais de redes de grande escala, é a presença de *lei de potência* na distribuição de graus. Uma vez que modelos como $G_{n,p}$ não tem essas propriedades, alguns estudos desencadearam pesquisas em modelos teóricos para explicá-las [3, 34]. Neste capítulo destacamos dois modelos para geração de grafos aleatórios cuja distribuição de graus é dada. Primeiro, o *modelo de configuração* que é baseado na ideia de emparelhamento aleatório de arestas. Segundo, o *modelo geral de grafo aleatório*, cuja probabilidade de cada aresta existir depende de pesos atribuídos aos vértices.

5.2 Grafos Lei de Potência

Seja X uma variável aleatória que assume valores no intervalo $[0, \infty]$. Matematicamente, X obedece uma *lei de potência* se ela tem distribuição de probabilidade $\Pr(X = x) = c \cdot x^{-\beta}$, onde c é uma constante normalizadora (para que $\sum_x Pr(X = x) = 1$) e β é um parâmetro constante da distribuição conhecido como o expoente ou parâmetro de escala [8]. Em teoria dos grafos, quando os graus dos vértices de um grafo seguem essa distribuição, chamamos de *grafos lei de potência*. Para a maioria dos grafos do mundo

real de grande escala, o parâmetro de escala se encontra no intervalo $2 < \beta < 3$, embora existam exceções [8, 7].

O modelo de grafo aleatório de lei de potência utilizado neste estudo se baseia principalmente no modelo de Aiello *et al.* [1], denotado por $P(\alpha, \beta)$, no qual considera-se um grafo aleatório em que a distribuição de graus depende de dois valores dados α e β, estes valores representam basicamente o tamanho e a densidade do grafo. Seja y o número de vértices de grau x, o modelo $P(\alpha, \beta)$ atribui uma probabilidade uniforme para todos os grafos com $y = \frac{e^\alpha}{x^\beta}$, ou seja,

$$|\{v : \delta(v) = x\}| = y = \frac{e^\alpha}{x^\beta}. \tag{5.1}$$

Note que o lado mais à direita da equação pode gerar números reais. Por isso, como o grau de um vértice deve ser um valor inteiro, na prática arredondamos para $\lfloor \frac{e^\alpha}{x^\beta} \rfloor$. Aiello *et al.* [1] mostra que tais arredondamentos geram erros que são negligíveis.

Alguns fatos importantes deste modelo são que o grau máximo do grafo é $e^{\frac{\alpha}{\beta}}$ e o número de vértices pode ser calculado somando todas as quantidades de vértices de um dado grau para o grau variando de 1 até $e^{\frac{\alpha}{\beta}}$. Como segue a equação

$$n = \sum_{x=1}^{e^{\frac{\alpha}{\beta}}} \frac{e^\alpha}{x^\beta} \approx \begin{cases} \zeta(\beta)e^\alpha & \text{se } \beta > 1 \\ \alpha e^\alpha & \text{se } \beta = 1 \\ \frac{e^{\frac{\alpha}{\beta}}}{1-\beta} & \text{se } 0 < \beta < 1 \end{cases} \tag{5.2}$$

onde $\zeta(\beta) = \sum_{i=1}^{\infty} \frac{1}{i^\beta}$ é a função Zeta de Riemann.

Utilizamos estes fatos para gerar um grafo cuja sequência de graus seja com distribuição de lei de potência no Capítulo 7, onde são apresentados os algoritmos para geração de grafos aleatórios.

5.3 Modelos Geradores

Os modelos de geração de grafos discutidos nesta seção servem para gerar grafos cuja sequência de graus siga uma distribuição específica, em particular, a distribuição lei de potência.

5.3.1 Modelo de Configuração

Seja $\delta(v)$ um valor inteiro que representa o grau de v, para todo vértice v de um grafo G. Uma sequência $(\delta(v_1), \delta(v_2), ..., \delta(v_n))$, onde n é o número de vértices de G, é dita *gráfica* se corresponde à sequência de graus de um grafo simples. Mais detalhes sobre a validade de uma sequência gráfica podem ser vistos em Hofstad [34]. Neste modelo, cada vértice

tem seu grau definido de acordo com uma *sequência gráfica* dada como entrada. Assim, todas as arestas de cada vértice são "meias arestas", e são emparelhadas aleatoriamente para que o grafo siga a distribuição de graus desejada [34]. O modelo de Aiello *et al.* [1] discutido na Seção 5.2, segue o modelo de configuração para construir grafos lei de potência.

A geração de grafos neste modelo funciona da seguinte forma. Dada uma sequência gráfica, o modelo é definido como segue. Para cada $v \in V$, são geradas $\delta(v)$ pontas (meias arestas) para v. Então todas as pontas são unidas aleatoriamente para formar arestas entre os vértices. Mais precisamente, escolhe-se duas pontas aleatórias entre todas as pontas do grafo e então emparelha-se as duas. Em seguida, escolhe-se outras duas pontas dentre as $\sum_{v \in V} \delta(v) - 2$ restantes para pareá-las novamente, e assim por diante até todas as pontas se unirem [3, 29].

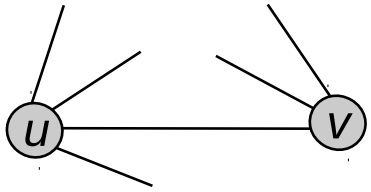


Figura 5.1: Emparelhamento entre vértices u e v.

A Figura 5.3.1 ilustra o emparelhamento entre os vértices u e v, onde o grau deles é 4 e 3 respectivamente, dos quais apenas uma ponta de cada foi emparelhada e as demais ainda são apenas pontas.

Como explica Britton *et al.* [3], ao seguir esta metodologia de geração de grafos, alguns problemas podem ocorrer quando o objetivo é gerar grafos simples. Neste caso as soluções sugeridas em [3] são remover os laços que surgirem e juntar as arestas paralelas sempre que aparecerem, ou reexecutar o algoritmo até que por acaso um grafo simples seja gerado. Um dos motivos por não usarmos neste estudo o próprio modelo de geração descrito por Aiello *et al.* [1], é justamente porque este gera multigrafos, uma vez que estamos interessados em grafos sem laços e arestas paralelas.

5.3.2 Modelo de Grafo Aleatório Generalizado

No modelo de grafo aleatório generalizado é criado um grafo vazio onde cada vértice recebe um peso que vai influenciar no sorteio das arestas. Diferentemente do $G_{n,p}$, neste modelo as probabilidades para a presença das aresta no grafo são diferentes, pois são moderadas em conformidade com os pesos atribuídos aos vértices [34]. Sejam v_i e v_j vértices de um grafo, a probabilidade de existir uma aresta entre v_i e v_j, tal que $v_i \neq v_j$ é dada por

$$p_{ij} = \frac{w_i w_j}{\sum_{i \in V} w_i + w_i w_j}, \tag{5.3}$$

onde w_i é o peso de cada vértice v_i.

A topologia dos grafos depende dos pesos escolhidos, assim, vértices com pesos maiores tem mais probabilidade de ter vizinhos do que vértices com pesos menores [34]. Dessa forma os pesos podem ser manipulados de modo que se tenha uma sequência gráfica esperada que obedeça uma certa distribuição de graus escolhida. Logo, grafos produzidos a partir do modelo generalizado de grafos aleatórios tem uma distribuição de graus que segue uma lei de potência quando a distribuição dos pesos também segue.

Os grafos gerados para este estudo combinam a topologia proposta no modelo $P(\alpha, \beta)$ com o modelo de grafo aleatório generalizado para criar grafos lei de potência com expoente de escala ajustável, como explicado no Capítulo 7.

Parte II

Algoritmos, Implementações e Resultados

CAPÍTULO 6

ALGORITMO PROPOSTO

Mesmo utilizando otimizações como o CELF, no algoritmo guloso de Kempe *et al.* [19] (Algoritmo 1), a estimativa da propagação σ precisa ser feita para quase todos os vértices do grafo em cada uma das k iterações. Isto faz com que o processo de seleção dos vértices sementes seja lento, uma vez que σ é obtido com simulações Monte Carlo. Trabalhamos na hipótese de que não são necessários tantos vértices examinados para se obter a mesma precisão.

Neste capítulo propomos uma otimização que reduz significativamente o número de chamadas à estimativa σ e obtém resultados equiparáveis aos obtidos com o Algoritmo 1. Além disso, comprovamos analiticamente os resultados sobre tempo de execução (Teoremas 5 e 7) e qualidade da solução (Lemas 2, 3 e 4).

6.1 Ideia Principal

Ao invés de percorrer todos os vértices calculando o ganho marginal para selecionar o maior de cada iteração, percorremos só um subconjunto dos vértices. Chamaremos esse subconjunto de *conjunto dos candidatos*. Este conjunto é escolhido previamente por uma heurística (Algoritmo 5) que explora as propriedades de diminuição de ganho marginal da função $\sigma(\cdot)$.

A melhoria que está sendo proposta se sustenta na suposição de que reduzindo o número de vértices a serem avaliados pelo Algoritmo 1, o tempo de execução pode diminuir. Com o objetivo de reduzir o número de chamadas à estimativa de propagação σ, podemos descartar os vértices que possam ter pouco ganho marginal antes de processá-los de fato. Desta forma, calculamos o ganho marginal apenas nos vértices considerados mais promissores, assim, escolhendo corretamente tais vértices, evitamos várias chamadas à estimativa de σ.

A abordagem para o cálculo da estimativa de propagação escolhida foi o método baseado em simulações Monte Carlo. A principal vantagem desta escolha é que conforme Kempe *et al.* [19], podemos obter uma boa precisão do valor de σ com o uso destas simulações. As alternativas existentes ao uso de repetidas simulações é a estimativa da função σ por meio de heurísticas, mas desta forma, não há garantia de precisão no valor estimado.

6.2 Seleção dos Candidatos

Para chegar a uma concepção razoável sobre quais vértices escolher na pré seleção, foi preciso levar em consideração alguns fatores que grafos de redes sociais comumente apresentam, o principal deles foi a distribuição de graus. Em tais redes existe uma minoria de vértices com um número muito grande de vizinhos de saída [11, 24, 7]. Assim, o cálculo da propagação de influência para estes vértices consome muito tempo, pois é necessário coletar informações sobre a propagação de influência de todos os seus vizinhos de saída. Consequentemente, isso compromete bastante o tempo de processamento dos grafos.

É documentado por Xiaodong *et al.* [24], que o cálculo da influência dos vértices de grau maior, chamados comumente de *hubs* [11, 24, 29], leva de 50 à 80% de todo o tempo necessário para processar o grafo inteiro. Além disso, apenas alguns poucos filhos (vizinhos de saída) dos *hubs* possuem grande influência, ao passo que muitos dos filhos são vértices que contribuem muito pouco para a influência dos *hubs*. Isso nos leva a crer que o alcance de uma informação que se propaga pela rede tem relação com a distribuição de graus.

Em resumo essas informações levantam quatro hipóteses importantes:

i) o tempo de processamento dos hubs é alto;

ii) poucos filhos dos hubs possuem grande influência;

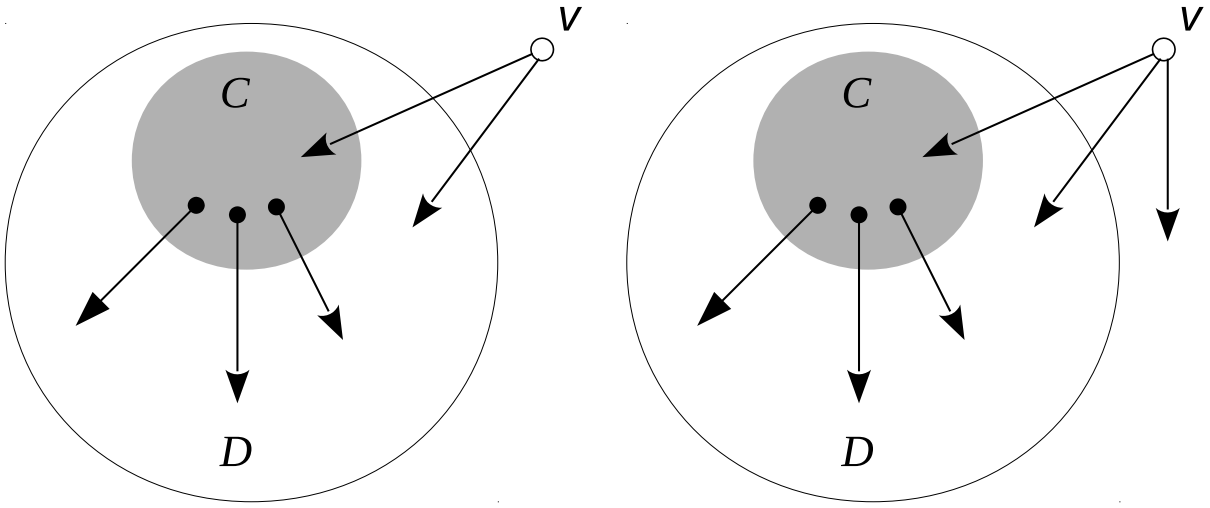
iii) muitos filhos dos hubs possuem pouca influência;

iv) possível relação da propagação de influência com a distribuição de graus.

A nossa estratégia se baseia fundamentalmente nesses quatro fatos para definir os critérios de escolha dos candidatos. Sabendo destas características, podemos supor que um vértice não terá alto ganho marginal se seus vizinhos de saída já tem a possibilidade de ser influenciados por vértices de grau maior. Então, a coleta de candidatos escolhe vértices que possam cobrir o maior número de vizinhos não cobertos. Um vértice é considerado coberto quando ele é vizinho de saída de outro vértice que já foi escolhido no processo iterativo. O procedimento pára quando não existem mais vértices descobertos.

6.2.1 O Conjunto de Candidatos

Antes de apresentarmos formalmente o procedimento de pré seleção, descrevemos a notação utilizada. Assim, definimos como $C \subseteq V$ o conjunto de candidatos escolhido pelo Algoritmo 5. Inicialmente, $C = \emptyset$, e os vértices são adicionados iterativamente durante a execução do algoritmo. Para todo $v \in V$, denotamos como $N^s(v) = \{w \in V \text{ tal que } (v, w) \in E\}$ o conjunto dos vizinhos de saída de v. A vizinhança de saída do conjunto C é denotada como $N^s(C) = \{(u, v) \in E \text{ tal que } u \in C \text{ e } v \notin C\}$.



(a) O vértice v não é selecionado (b) O vértice v é selecionado

Figura 6.1: Conjunto dos vértices cobertos por C.

O Algoritmo 5, considera os vértices $v_1, v_2, ..., v_{|V|}$ em V ordenados de maneira não crescente pelo grau de saída. Depois adiciona ao conjunto C todo vértice v_i tal que $N^s(v_i) \nsubseteq C \cup N^s(C)$, isto é, v_i é escolhido quando tem pelo menos um vizinho que ainda não está coberto. Para simplificar o pseudocódigo denotamos por $D = C \cup N^s(C)$ o conjunto de vértices coberto por C, como é ilustrado na Figura 6.1.

Algoritmo 5: PRÉ SELEÇÃO

Entrada: $G = (V, E)$
Saída: Conjunto C de candidatos

```
1  início
2  |  Vértices v_1, v_2, ..., v_|V| em ordem não crescente de grau
3  |  C ← ∅
4  |  para i ← 1 até |V| faça
5  |  |  se N^s(v_i) ⊈ D e N^s(v_i) ≠ ∅ então
6  |  |  |  C ← C ∪ {v_i}
7  |  |  |  Atualiza D
8  |  |  fim
9  |  fim
10 fim
11 retorna C
```

A Figura 6.1 exemplifica um passo do Algoritmo 5 em duas situações distintas. Na situação da Figura (a), o vértice v já tem seus vizinhos contidos em D, portanto não é escolhido. Na Figura (b), v possui um vizinho de saída que ainda não foi coberto, neste caso, v entraria para o conjunto C. A ideia disso é que, se v_i já tem todos os seus vizinhos de saída em D, então adicionar v_i a C não traria um aumento significativo da influência do conjunto C exercida sobre seus vizinhos. Esta ideia é verificada analiticamente nos

argumentos que seguem.

6.2.2 Análise da Pré Seleção

Para a análise do Algoritmo 5 definimos dois critérios, qualidade da pré seleção e tempo de execução. Por motivo de simplificação, na análise, assumimos que no modelo de cascata independente a probabilidade de ativação é igual em toda aresta, digamos, p. Além disso, ao invés de usar a função de influência σ nos cálculos, consideramos uma função mais simples que chamaremos de *influência direta*. Estas simplificações tornam os contas mais fáceis. Não fazer tais simplificações significa calcular a influência exata de cada vértice, o que não é o objetivo desta análise.

Definição 4 (Influência direta de um vértice)**.** *Seja v um vértice ativo do grafo, denotamos $inf(v)$ a influência direta de v, definida como o número de vizinhos de v que são ativados pelo próprio v no processo de ativação.*

Da mesma forma, a Definição 5 se aplica a um conjunto de vértices.

Definição 5 (Influência direta de um conjunto)**.** *Seja C um conjunto de vértices ativos no grafo, denotamos $inf(C)$ a influência direta de C, definida como o número de vértices em $N^s(C)$ que são ativados por C no processo de ativação.*

Observe que para todo v, $\inf(v)$ é uma variável aleatória, pois cada um dos vizinhos de v é ativado com probabilidade p. Logo, cada execução do processo de ativação, $\inf(v)$ pode possuir um valor diferente entre zero e $|N^s(v)|$. Equivalentemente, o mesmo acontece com $\inf(C)$. Por isso, nos resultados que seguem trabalhamos com valor esperado.

Para a análise da qualidade da pré seleção, note que, durante a execução do Algoritmo 5, um vértice v sendo avaliado, **não** é adicionado a C se ele não atende a condição da linha 5, ou seja, se $N^s(v) \subseteq D$. Isso pode acontecer de duas formas: (i) se $N^s(v) \subseteq C$; ou (ii) se $N^s(v) \nsubseteq C$. Os Lemas 2 e 4 tratam, respectivamente, dos casos (i) e (ii).

Lema 2. *Seja v_i um vértice avaliado na iteração i do Algoritmo 5. Se $N^s(v_i) \subseteq C$, então* $\mathrm{E}[inf(C \cup \{v_i\})] - \mathrm{E}[inf(C)] \leq 0$*, ou seja, a influência adicional de v_i é nula ou negativa.*

Prova. Para verificarmos a desigualdade do lema é preciso obter o valor de $\inf(C)$. Como cada $w \in N^s(C)$ pode receber mais de uma aresta de C, então o raciocínio é como segue.

Para saber o número de vértices que são ativados diretamente por C, vamos considerar uma variável aleatória Y_w, definida como

$$Y_w = \begin{cases} 1, \text{ se } w \text{ é ativado por um vértice em } C \\ 0, \text{ caso contrário.} \end{cases}$$

Deste modo,

$$\text{inf}(C) = \sum_{w \in N^s(C)} Y_w$$

Pela linearidade da esperança e por Y_w ser binária, o valor esperado de inf(C) é

$$\begin{aligned} \text{E}[\text{inf}(C)] &= \sum_{w \in N^s(C)} \text{E}[Y_w] \\ &= \sum_{w \in N^s(C)} \Pr(Y_w = 1). \end{aligned} \tag{6.1}$$

Agora, vamos supor que adicionamos v_i a C com o objetivo de aumentar inf(C). Isto gera dois resultados possíveis, um em que $v_i \notin N^s(C)$ e outro em que $v_i \in N^s(C)$.

No primeiro resultado, é simples de visualizar que incluir v_i em C não diminui o valor de inf(C), uma vez que v_i não tem vizinho fora de C, nenhuma aresta será adicionada na soma das probabilidades da Equação 6.1, ou seja,

$$\text{E}[\text{inf}(C \cup \{v_i\})] = \text{E}[\text{inf}(C)].$$

Assim, considerando um processo de ativação em que C é um conjunto de vértices ativos, todos os vizinhos que v_i ativaria, já estariam ativos.

O segundo e pior, é quando v_i é um elemento do conjunto $N^s(C)$, neste caso, ao adicionar v_i à C estamos reduzindo um elemento de $N^s(C)$, com isso o valor de inf(C) passa a ser no máximo $|N^s(C)| - 1$. Desta forma, pelo menos uma aresta é retirada na soma da Equação 6.1, ou seja,

$$\begin{aligned} \text{E}[\text{inf}(C \cup \{v_i\})] &= \sum_{w \in N^s(C) \setminus \{v_i\}} \Pr(Y_w = 1) \\ &< \text{E}[\text{inf}(C)]. \end{aligned}$$

Considerando as duas possibilidades, temos que

$$\text{E}[\text{inf}(C \cup \{v_i\})] \leq \text{E}[\text{inf}(C)].$$

□

Estendendo o raciocínio usado na demostração do Lema 2 para o caso em que o vértice avaliado na iteração atual possui vizinhos tanto em C como em $N^s(C)$, chegamos ao resultado do Teorema 3. Neste Teorema é definido um limitante superior para a influência direta adicional que este tipo de vértice pode proporcionar ao conjunto dos candidatos.

Teorema 3. *Seja v_i um vértice avaliado na iteração i do Algoritmo 5. Se $N^s(v_i) \nsubseteq C$ mas $N^s(v_i) \subseteq D$, então* $\mathrm{E}[\inf(C \cup \{v_i\})] - \mathrm{E}[\inf(C)] \leq \frac{1}{4}|N^s(v_i) \setminus C|$.

Prova. Pela hipótese, $N^s(v_i)$ não está inteiramente contido em C, mas pertence ao conjunto D. Por isso, pelo Algoritmo 5, é um vértice que deve permanecer em $V \setminus C$. No entanto, se consideramos adicionar v_i ao conjunto C, afim de aumentar o valor de $\inf(C)$, temos que levar em conta os dois casos possíveis em que v_i pode se encontrar, que são: (i) $v_i \in N^s(C)$ e (ii) $v_i \notin N^s(C)$.

Diferentemente do que foi visto no Lema 2, agora, tanto no caso (i) como no caso (ii) existe a possibilidade de aumentar a influência direta esperada de C. Isso porque aumentando o número de arestas incidentes em $N^s(C)$, aumentamos também a probabilidade destes vértices serem influenciados. Queremos encontrar um limitante superior para o aumento da probabilidade de ativação.

Desta forma, considere o vértice $w \in N^s(v_i) \setminus C$. Se não adicionamos v_i em C, então w sofre influência apenas de C. Podemos calcular a probabilidade de C ativar w diretamente da seguinte maneira. Seja A o evento em que w é ativado por C. Considerando que p é a probabilidade de ativação no modelo de cascata independente, então $\Pr(A) \geq p$, pois $w \in N^s(C)$, isto é, existe pelo menos uma aresta que sai de C e chega em w.

Agora, considere que adicionamos v_i à C, então w sofre influência de C e de uma aresta que vem de v_i. Assim, seja B o evento em que w é ativado por v_i, logo, $\Pr(B) = p$ e a probabilidade do conjunto $C \cup \{v_i\}$ ativar w pode ser obtida como

$$\begin{aligned}\Pr(A \cup B) &= \Pr(A) + \Pr(B) - \Pr(A \cap B) \\ &= \Pr(A) + \Pr(B) - \Pr(A) \cdot \Pr(B) \\ &= \Pr(A) + p - \Pr(A) \cdot p\end{aligned}$$

Lembrando que aplicamos o princípio da inclusão e exclusão na soma das probabilidades de ativação, onde a segunda igualdade ocorre devido à independência dos eventos A e B.

Para encontrar o aumento proporcionado por v_i na probabilidade de ativar w, queremos saber a diferença entre as probabilidades $\Pr(A \cup B)$ e $\Pr(A)$. Observe que $\Pr(A)$ é equivalente ao valor de $\Pr(Y_w = 1)$ na Equação 6.1. Dessa forma, seja $\mathrm{E}[\inf_w(C)]$ a influência esperada de C sobre w, pela Equação 6.1 temos

$$\mathrm{E}[\inf\nolimits_w(C)] = \Pr(Y_w = 1) = \Pr(A),$$

da mesma forma

$$\mathrm{E}[\inf\nolimits_w(C \cup \{v_i\})] = \Pr(A \cup B).$$

Então, o aumento é

$$\begin{aligned}\mathrm{E}[\inf_w(C \cup \{v_i\})] - \mathrm{E}[\inf_w(C)] &= \Pr(A \cup B) - \Pr(A) \\ &= \Pr(A) + p - \Pr(A) \cdot p - \Pr(A) \\ &= p - \Pr(A) \cdot p \\ &= p(1 - \Pr(A)) \\ &\leq p(1-p),\end{aligned}$$

onde a desigualdade é decorrente do fato de que $\Pr(A) \geq p$. Isso vale para qualquer $w \in N^s(v_i) \backslash C$, consequentemente, é preciso realizar a mesma diferença das probabilidades para todos os vértices de $N^s(v_i) \setminus C$.

Perceba que o aumento resultou em uma função quadrática e seu gráfico é uma parábola com concavidade para baixo, portanto, podemos encontrar o ponto máximo absoluto desta função. Em vista disso, temos que o maior valor que a função $(-p^2 + p)$ atingirá é $\frac{1}{4}$, quando $p = \frac{1}{2}$.

Desta forma, temos

$$\begin{aligned}\mathrm{E}[\inf(C \cup \{v_i\})] - \mathrm{E}[\inf(C)] &= \prod_{w \in N^s(v_i) \backslash C} (\mathrm{E}[\inf_w(C \cup \{v_i\})] - \mathrm{E}[\inf_w(C)]) \\ &\leq \prod_{w \in N^s(v_i) \backslash C} p(1-p) \\ &\leq \prod_{w \in N^s(v_i) \backslash C} \frac{1}{4} \\ &= \frac{1}{4} \cdot |N^s(v_i) \setminus C|.\end{aligned}$$

□

O resultado do Teorema 3 mostra que selecionar como candidato um vértice v_i tal que $N^s(v_i) \nsubseteq C$ e $N^s(v_i) \subseteq D$, aumenta muito pouco o valor esperado para $\inf(C)$. Além disso, o Lema 4 mostra que no final do processo de pré seleção, qualquer vértice que foi descartado exerce menos influência sobre seus vizinhos do que o conjunto C.

Lema 4. *Ao final da pré seleção, para cada $v \in V \setminus C$, $\mathrm{E}[inf(v)] \leq \mathrm{E}[inf_{N^s(v)}(C)]$, onde $inf_{N^s(v)}(C)$ representa a influência direta de C sobre $N^s(v)$.*

Prova. Se $v \in V \setminus C$ então, pelo Algoritmo 5, todo $w \in N^s(v)$ deve estar em D, ou seja, w está em C ou em $N^s(C)$. Mas, como queremos comparar a influência de v com a de C sobre $N^s(v)$, podemos levar em consideração apenas os vértices em $N^s(v) \setminus C$. Assim, quando $w \in N^s(C)$, além de receber uma aresta de v, w também recebe pelo menos uma aresta de C, ou seja, para cada vértice em $N^s(v) \setminus C$, existe pelo menos um vértice $u \in C$, tal que $(u, w) \in E$, caso contrário $N^s(v) \nsubseteq D$.

Desta forma, podemos calcular o valor de $\inf(v)$ como segue. Seja $X_{v,w}$ uma variável aleatória binária tal que,

$$X_{v,w} = \begin{cases} 1, \text{ se } v \text{ ativa } w \\ 0, \text{ caso contrário.} \end{cases}$$

Então,

$$\inf(v) = \sum_{w \in N^s(v) \setminus C} X_{v,w}.$$

Usando a linearidade da esperança e o fato de cada $X_{v,w}$ ser uma variável aleatória binária, temos

$$\begin{aligned} \mathrm{E}[\inf(v)] &= \mathrm{E}\left[\sum_{w \in N^s(v) \setminus C} X_{v,w}\right] \\ &= \sum_{w \in N^s(v) \setminus C} \mathrm{E}[X_{v,w}] \\ &= \sum_{w \in N^s(v) \setminus C} \Pr(X_{v,w} = 1) \\ &= \sum_{w \in N^s(v) \setminus C} p \end{aligned} \tag{6.2}$$

Para chegar ao valor de $\inf_{N^s(v)}(C)$ o raciocínio é parecido com o utilizado para obter a Equação 6.1, uma vez que existe um conjunto de vértices em C com arestas apontando para $N^s(v)$, além disso, cada $w \in N^s(v)$ pode receber mais de uma aresta de C.

Sendo assim, seja $B_w = \{u \in C \text{ tal que } (u, w) \in E\}$, para todo $w \in N^s(v) \setminus C$ o conjunto dos vértices em C que tem arestas para w. Neste contexto, $|B_w| \geq 1$, porque senão w não estaria coberto por C. Agora, considere os eventos $E_1, E_2, ..., E_{|B_w|}$ tal que E_i é o evento em que w é ativado por $u_i \in B_w$.

Uma vez que definimos que toda aresta tem probabilidade p de ativação, então $\Pr(E_1) = \Pr(E_2) = ... = \Pr(E_{|B_w|}) = p$. Logo, para cada $w \in N^s(v) \setminus C$, temos

$$\begin{aligned} \Pr(X_{v,w} = 1) &= p \\ &\leq \Pr(E_1 \cup E_2 \cup ... \cup E_{|B_w|}) \\ &= \Pr(B_w \text{ ativar } w). \end{aligned} \tag{6.3}$$

Para saber o número de vértices em $N^s(v)$ que são ativados por C, vamos considerar novamente uma variável aleatória Y_w, definida como

$$Y_w = \begin{cases} 1, \text{ se } w \text{ é ativado por um vértice em } C \\ 0, \text{ caso nao contrário.} \end{cases}$$

Deste modo,

$$\inf_{N^s(v)}(C) = \sum_{w \in N^s(v) \setminus C} Y_w$$

Novamente, pela linearidade da esperança,

$$\begin{aligned}
\mathrm{E}[\inf_{N^s(v)}(C)] &= \sum_{w \in N^s(v) \setminus C} \mathrm{E}[Y_w] \\
&= \sum_{w \in N^s(v) \setminus C} \Pr(Y_w = 1) \\
&= \sum_{w \in N^s(v) \setminus C} \Pr(B_w \text{ ativar } w) \\
&\geq \sum_{w \in N^s(v) \setminus C} p \\
&= \mathrm{E}[\inf(v)],
\end{aligned}$$

onde a desigualdade segue da Equação 6.3 e a última igualdade segue da Equação 6.2. □

Apesar do fato de fazermos os cálculos considerando a influência direta ao invés do ganho marginal, acreditamos que o vértices desprezados na seleção dos candidatos não fazem falta em um modelo mais complexo com probabilidades de propagação distintas em cada aresta, como é verificado nos experimentos do Capítulo 8.

A seguir, o Teorema 5 determina um limitante superior para o tempo de execução do Algoritmo 5.

Teorema 5. *Seja $G = (V, E)$ um grafo direcionado com $|V| = n$ e $|E| = m$. A* Pré Seleção *termina em $O(n + m)$ passos.*

Prova. Como no início do algoritmo os vértices devem ser ordenados pelo grau de saída, se usarmos um algoritmo eficiente como o *counting sort*, esta tarefa é feita com $O(n)$ operações. Podemos usar este algoritmo devido ao fato dos graus de saída dos vértices estarem entre os valores de 1 até $n - 1$.

No laço das linhas 4-9 a operação mais complicada é a condição da linha 5, que depende de saber se o conjunto $N^s(v_i)$ está contido em D. Podemos usar uma *tabela hash* para armazenar os elementos do conjunto D, dessa forma, é possível verificar se cada $w \in N^s(v_i)$ pertence a D em tempo $O(1)$.

Assim, seja $\delta^+(v_i)$ o grau de saída de v_i, em no máximo $\delta^+(v_i) \cdot O(1)$ passos é feita a verificação de v_i. Logo, pelo teorema de "*handshake*" de teoria dos grafos, no laço são executados até

$$\sum_{v_i \in V} \delta^+(v_i) = O(m)$$

comparações.

Por fim, o tempo total para a pré seleção é $O(n+m)$. □

Corolário 6. *Seja $G = (V, E)$ um grafo direcionado com $|V| = n$ e $|E| = m$. Em grafos lei de potência com $\beta > 2$, a* Pré Seleção *termina em tempo $O(n)$.*

Prova. Para chegar a este resultado usamos o fato de que em grafos lei de potência com $\beta > 2$ temos que $m = O(n)$. Isto pode ser observado no modelo $P(\alpha, \beta)$ de Aiello *et al.* [1], onde é pontuado que nos grafos deste modelo $n \approx e^{\alpha}\zeta(\beta)$ e $m \approx e^{\alpha}\zeta(\beta - 1)\frac{1}{2}$.

Usando estes resultados, queremos provar que existe uma constante $c > 0$ tal que $m = c \cdot n$. Substituindo estes valores, temos

$$e^{\alpha}\zeta(\beta - 1)\frac{1}{2} \leq c \cdot e^{\alpha}\zeta(\beta),$$

que rearranjando os termos, obtemos

$$\frac{\zeta(\beta - 1)}{\zeta(\beta)} \leq 2c. \tag{6.4}$$

Como a função $\zeta(z)$ de Riemann é definida para valores z cuja parte real ($\Re(z)$) é maior ou igual a 1, então, $\zeta(z)$ converge para todo $z \in \mathbb{C}$ tal que $z > 1$ [10]. Assim, temos que para $\beta > 2$, $\zeta(\beta)$ e $\zeta(\beta - 1)$ convergem. Logo, o lado esquerdo da Equação 6.4 converge, portanto, existe a constante $c' = 2c$. Desta forma, temos a desigualdade $m \leq c'n$, e consequentemente, $O(n + m) = O(n)$. □

Levando em consideração a informação de que normalmente grafos lei de potência do mundo real possuem o valor de β entre 2 e 3, obtemos o resultado do Corolário 6 como consequência do Teorema 5 no contexto deste tipo de grafo. Isso mostra que a heurística que escolhe os vértices pelo grau é uma maneira rápida de pré seleção e os benefícios de escolher tais vértices, são maiores do que os custos com processamento.

É importante lembrar que o processo de seleção analisado nesta seção, não é um procedimento pra escolher os adeptos iniciais, o objetivo deste algoritmo é fazer um filtro para agilizar na escolha dos vértices sementes, que é feita na Seção 6.3.

6.3 Otimização por Pré Seleção

Nesta seção propomos o algoritmo que escolhe os adeptos inicias de uma rede utilizando a pré seleção. A ideia combina seleção prévia com um algoritmo guloso mais um esquema de atualização "*Lazy Forward*". Denotamos por PrevalentSeed o algoritmo que usa a otimização por pré seleção.

Usando o Algoritmo 5 como sub rotina do Algoritmo 6, primeiro dividimos o conjunto de vértices em dois subconjuntos disjuntos, C e $V \backslash C$, tal que C é o conjunto de candidatos

considerados mais promissores de G (na linha 3). Ao invés do algoritmo guloso original, optamos por utilizar o CELF como sub rotina do nosso algoritmo, uma vez que este é mais rápido. Portanto, o trecho de código entre as linhas 4-18 é uma modificação do Algoritmo 3, na qual, a diferença se encontra no laço das linhas 4-7, onde inserimos na fila Q apenas os vértices escolhidos para o conjunto C, a partir deste instante, o ganho marginal dos vértices em $V \setminus C$ não é calculado. O ganho marginal de cada $v \in C$ é calculado e adicionado a Q em ordem não crescente de ganho marginal. Depois, a busca segue a mesma ideia do CELF, nas linhas 8-18, fazemos uma busca gulosa para selecionar os k vértices de maior ganho marginal dentre os vértice pertencentes ao conjunto C, ao passo que se mantém a ordenação da fila de prioridade dos ganhos marginais.

Algoritmo 6: PREVALENTSEED

```
   Entrada: G, k, σ
   Saída: Conjunto semente S
 1 início
 2 |  S ← ∅, Q ← ∅
 3 |  C ← PRÉ SELEÇÃO(G)
 4 |  para cada u ∈ C faça
 5 |  |  δ_u ← σ({u}); u.it ← 0
 6 |  |  Adiciona u à Q em ordem decrescente de δ_u
 7 |  fim
 8 |  enquanto |S| ≤ k faça
 9 |  |  Desenfileira u de Q
10 |  |  se u.it = |S| então
11 |  |  |  S ← S ∪ {u}
12 |  |  fim
13 |  |  senão
14 |  |  |  δ_u ← σ(S ∪ {u}) − σ(S)
15 |  |  |  u.it ← |S|
16 |  |  |  Reinsere u em Q e reordena
17 |  |  fim
18 |  fim
19 fim
20 retorna S
```

Assim como no Algoritmo 3, o elemento de Q que corresponde a v, armazena uma dupla da forma $\langle \delta_v, v.it \rangle$, onde $\delta_v = \sigma(S \cup \{v\}) - \sigma(S)$ representa o ganho marginal de v com relação ao conjunto S, enquanto $v.it$ marca qual a iteração que δ_v foi atualizado pela última vez. Em cada uma das k iterações do laço 'enquanto', v é removido da fila e é verificado se já teve seu ganho marginal calculado na iteração atual, usando o atributo *it*. Se sim, v é o vértice de maior ganho marginal na iteração atual, assim, ele será selecionado como semente (linhas 9-12). Caso contrário, nas linha 13-17, é recalculado o ganho marginal de v e ele é reinserido em Q de modo que a ordem seja mantida.

O Teorema 7 determina o tempo de execução do PrevalentSeed.

Teorema 7. *Seja $G = (V, E)$ um grafo direcionado com $|V| = n$ e $|E| = m$. O algoritmo* PrevalentSeed *tem tempo de execução $O(knrm)$.*

Prova. A complexidade do PrevalentSeed é dada da seguinte maneira.

i) Pelo Teorema 5, o Algoritmo 5 usa $O(n + m)$ passos para encontrar o conjunto dos candidatos, na linha 3.

ii) São realizadas $O(|C|rm)$ operações no laço das linhas 4-7.

- Este laço calcula o valor de $\sigma(v)$ para todo $v \in C$. O valor de $\sigma(v)$ é estimado com $r = 10.000$ simulações do processo de propagação (método Monte Carlo). Cada simulação é uma chamada ao Algoritmo 2, que leva tempo $O(m)$. Dessa maneira, toda chamada a $\sigma(v)$ custa tempo $O(rm)$.
- Além disso, cada inserção na fila Q é feita em $O(1)$ passos. O que resulta em $O(|C|rm)$ operações neste laço.

iii) Para selecionar k vértices sementes são necessários $O(knrm)$ passos, no laço 'enquanto'.

- A linha 9 custa tempo $O(\log n)$ para extrair o vértice de maior ganho marginal. Dado que Q é uma *Heap de Fibonacci*.
- Depois, a próxima operação fundamental é o cálculo do ganho marginal (δ_v), na linha 14. Esta linha faz duas chamadas a função σ, assim como na linha 5, cada chamada dessas gasta $O(rm)$ passos.
- No pior caso, em cada iteração do laço é preciso calcular δ_v de todos os vértices da fila Q. Cada vez que um vértice é escolhido, 1 elemento é retirado de Q. Logo, são necessários $|C| - |S| = O(|C|)$ cálculos de ganho marginal.
- Ao final de k iterações, serão realizadas $O(k|C|)$ chamadas à σ. Então, o laço demanda o total de $O(\log n) + O(k|C|) \cdot O(rm)$ operações. Arrendondando tanto $\log n$ como $|C|$ para n, chegamos em $O(knrm)$ passos neste laço.

Concluímos que o tempo total é dado pela soma dos itens i), ii) e iii), como segue. $O(n + m) + O(|C|rm) + O(knrm) = O(knrm)$. □

Note que pelo Teorema 7, assintoticamente, o tempo de execução do Algoritmo 6 é igual ao do Celf (Algoritmo 3), mesmo fazendo a pré seleção. Mas na prática, o primeiro se comporta bem melhor, como é discutido no Capítulo 8, onde são descritas as avaliações deste algoritmo.

6.4 Complexidade de Tempo dos Algoritmos

Para efeitos de comparação, a Tabela 6.1 contém o tempo de execução dos algoritmos utilizados neste trabalho que fazem uso de simulações Monte Carlo. De acordo com Leskovec *et al.* [22], assintoticamente, o tempo de execução do CELF permanece o mesmo do algoritmo guloso original. Da mesma forma, o CELF++ mantém essa complexidade.

Tabela 6.1: Complexidade de tempo dos algoritmos descritos, onde n é o número de vértices, m o número de arestas e r o número de simulações Monte Carlo.

Algoritmo	Complexidade de Tempo
Algoritmo 1: GULOSO	$O(knrm)$
Algoritmo 2: CASCATA	$O(m)$
Algoritmo 3: CELF	$O(knrm)$
Algoritmo 4: CELF++	$O(knrm)$
Algoritmo 5: PRÉ SELEÇÃO	$O(n+m)$
Algoritmo 6: PREVALENTSEED	$O(knrm))$

6.5 Questões em Aberto

Separamos duas questões que vêm a ser interessantes sobre as vantagens obtidas com o processo de pré seleção do Algoritmo 5 e a utilização deste no Algoritmo 6. A primeira é o tamanho do conjunto C, que acreditamos ser possível limitar considerando-se grafos de lei de potência. A segunda é chegar à uma taxa de aproximação para o conjunto semente final. Para esta questão, acreditamos que o Algoritmo 6 preserva o fator $(1 - \frac{1}{e})$ do Algoritmo 1. Ambas as indagações necessitam de uma análise rigorosa do algoritmo e dos grafos estudados. Deixamos estas questões em aberto, mas acreditamos resolvê-las em trabalhos futuros. A seguir estão alguns indícios e a intuição que observamos sobre o assunto.

6.5.1 Limitantes de Tamanho para o Conjunto dos Candidatos

Embora o Algoritmo 5 seja determinístico, a entrada pode ser um grafo aleatório, neste caso, é importante lembrar que o conjunto resultante da PRÉ SELEÇÃO depende da distribuição de graus. Por isso, uma vez que o algoritmo foi elaborado para grafos com distribuição de lei de potência, seria interessante haver limitantes inferiores e superiores para $|C|$ com base no expoente β da lei de potência.

Um limitante inferior é importante porque a busca por k adeptos iniciais ocorre dentro do conjunto C, então, espera-se que $k < |C|$ para que a busca termine corretamente. Além do mais, encontrar também um limitante superior para $|C|$ ajuda no sentido de que, sabendo-se o tamanho máximo e mínimo do conjunto de candidatos, é possível estimar se

vale a pena ou não empregar uma pré seleção, pois, caso $|C|$ seja próximo de n, a redução em tempo de execução não seria tão notável.

Como neste estudo trabalhamos com $\beta > 2$ e $k < |V|$, todos os experimentos ocorreram sem problemas com relação ao tamanho de C.

6.5.2 Fator de Aproximação

Seja S^* o subconjunto que maximiza a propagação esperada (σ) entre todos os subconjuntos de vértices de tamanho k. O Algoritmo 1 é uma $(1 - 1/e)$-aproximação do ótimo [20]. Logo, o conjunto S encontrado pelo Algoritmo 1 satisfaz a desigualdade

$$\sigma(S) \geq (1 - 1/e) \cdot \sigma(S^*).$$

Neste raciocínio, seja S' um conjunto encontrado pelo algoritmo PrevalentSeed. Na intenção de encontrar um fator de aproximação para S', devemos provar que existe uma constante c tal que

$$\sigma(S') \geq \frac{1}{c} \cdot \sigma(S^*).$$

Mas, como acreditamos na hipótese de que o Algoritmo 6 preserva o mesmo fator do Algoritmo 1, então para garantir a aproximação desejada, seria suficiente provar que

$$\sigma(S') \geq \sigma(S),$$

onde S é o conjunto obtido pelo Algoritmo 1.

Prosseguindo com as argumentações da hipótese, seja $C \subseteq V$ o subconjunto resultante de uma pré seleção feita em um grafo G usando o Algoritmo 5. Considere A^*_C o subconjunto que maximiza σ entre todos os subconjuntos de tamanho k em C. Agora, observe que ao selecionar um conjunto A' de k vértices em C com o Algoritmo 1, pelas propriedades da função σ, o A' obtido é $\sigma(A') \geq (1 - 1/e) \cdot \sigma(A^*_C)$ dentre os resultados possíveis no conjunto C. Considere também o conjunto semente A escolhido com o mesmo algoritmo guloso do A', mas dessa vez avaliando o ganho marginal de todos os vértices em V. Note que este conjunto pode conter tanto elementos de $V \setminus C$ como de C.

Desta forma, supomos que a maioria dos elementos em $V \setminus C$ é composta de vértices cujo ganho marginal é sempre baixo e que seriam descartados de qualquer forma. Dado isto, parte do conjunto A deve conter elementos de C, então supomos que podemos selecionar todos os elementos de A apenas dentro de C obtendo a mesma qualidade, assim, não precisamos examinar todos os vértices do grafo na busca gulosa. Acreditamos que isso é verdade pelos seguintes aspectos dos conjuntos C e $C \setminus V$:

- Devido aos critérios para a escolha de C, todos os vértices que não possuem vizinhos de saída estão em $V \setminus C$ e a maior parte dos *hubs* estão em C. Consequentemente,

os vértices em C tem mais chances de pertencer a S do que a maioria dos vértices em $V \setminus C$.

- Como os grafos considerados são de lei de potência e direcionados, a maioria dos vértices tem grau 1, dos quais, boa parte não possuem arestas de saída, assim, já podemos supor vários vértices nesta condição.

- Além destes, existem ainda os vértices de grau maior que 1 mas que também não têm aresta de saída, por isso, pertencem a $V \setminus C$.

- A maioria dos vértices em $V \setminus C$ são vértice de baixa propagação estimada, isto é, não ativariam mais ninguém além deles mesmos porque não possuem arestas de saída.

- Para todo $v \in V \setminus C$ cujo grau de saída é maior que 1, v está sujeito aos Lemas 2 e 4, ou seja, os vértices em C são suficientes para alcançar os vizinhos de saída de v.

Apesar de não fornecermos uma prova desta hipótese no trabalho, estas observações nos levam a crer que calculando o ganho marginal apenas dos vértices em C, podemos obter um conjunto S' tão bom quanto S, mas com a diferença de que são feitas menos chamadas à função σ, e consequentemente, menos tempo de execução é necessário.

CAPÍTULO 7

GERAÇÃO DE GRAFOS ALEATÓRIOS

Uma vez fornecidos os devidos parâmetros para o modelo $P(\alpha, \beta)$, é possível obter um grafo lei de potência, conforme a descrição do modelo no Capítulo 5. Nas seções que seguem descrevemos como combinamos o modelo $P(\alpha, \beta)$ com o modelo de grafo aleatório generalizado (GRG - *Generalized Random Graphs*) para gerar grafos lei de potência.

7.1 Gerando a sequência de pesos

Dado o número n de vértices e o expoente de escala β de um grafo, podemos manusear a Equação 5.2 (Capítulo 5) para obter a quantidade de vértices de cada grau, respeitando a distribuição de lei de potência. Desta forma, seja $s = (s_1, s_2, ..., s_n)$ uma sequência onde s_i é número de vértices de grau i. Conhecendo os valores n e β da Equação 5.2, obtemos que

$$e^{\alpha} \approx \begin{cases} \frac{n}{\zeta(\beta)}, & \text{se } \beta > 1 \\ \frac{n}{\log n}, & \text{se } \beta = 1 \\ (n - n\beta)^{\beta}, & \text{se } 0 < \beta < 1. \end{cases} \tag{7.1}$$

Agora, podemos usar o valor e^{α} obtido para atribuir um número para cada s_i diretamente pela Equação 5.1, ou seja,

$$s_i = \frac{e^{\alpha}}{i^{\beta}}. \tag{7.2}$$

Deste modo, usamos a distribuição de s obtida com a Equação 7.2 para gerar uma sequência gráfica que segue uma lei de potência. Assim, ao invés dos parâmetros α e β, os parâmetros são n e β, como mostra o Algoritmo 7.

Na situação ideal, $\sum_{i=1}^{n-1} s_i = n$, mas o número de vértices resultantes é possivelmente menor que n devido a função de piso na linha 13. A quantidade de arestas muda para os valores diferentes de β por que este é o parâmetro que define a densidade do grafo. Outra observação sobre a distribuição gerada é que quanto maior o valor de β, menor é o grau máximo e muitos mais vértices de grau 1 aparecem na sequência.

Algoritmo 7: Distribuição de Graus

Entrada: n, β
Saída: Uma sequência $s = (s_1, s_2, ..., s_{n-1})$

```
1  início
2      se β > 1 então
3          e^α ← n / ζ(β)
4      fim
5      se β = 1 então
6          e^α ← n / log n
7      fim
8      se 0 < β < 1 então
9          e^α ← (n − nβ)^β
10     fim
11     para i = 1 até n faça
12         s_i ← ⌊e^α / i^β⌋
13     fim
14 fim
15 retorna s
```

7.2 Algoritmo Para Gerar Grafos

Sabendo que o modelo de grafo generalizado recebe como entrada uma sequência $w = (w_1, w_2, ..., w_n)$ de pesos que determina a topologia do grafo resultante, a saída do Algoritmo 7 é usada para obter w.

Algoritmo 8: Grafo Aleatório Generalizado

Entrada: Sequência de pesos $w = (w_1, w_2, ..., w_n)$
Saída: Um grafo com distribuição de graus esperada

```
1  início
2      para i ← 1 até |w| faça
3          Adiciona i vértices com peso w_i ao grafo G
4      fim
5      para cada v_i ∈ V faça
6          para cada v_j ∈ V \ {v_i} faça
7              p_ij = w_i w_j / (Σ_{i∈V} w_i + w_i w_j)
8              Adiciona a aresta {v_i, v_j} em G com probabilidade p_ij
9          fim
10     fim
11 fim
12 retorna G
```

Baseado no artigo de Chung e Lu [7], construímos o Algoritmo 8 para geração de grafos aleatórios generalizados, no qual, é dada uma sequência w de pesos para n vértices. Na linha 8, uma aresta entre os vértices v_i e v_j é adicionada com probabilidade p_{ij} definida na

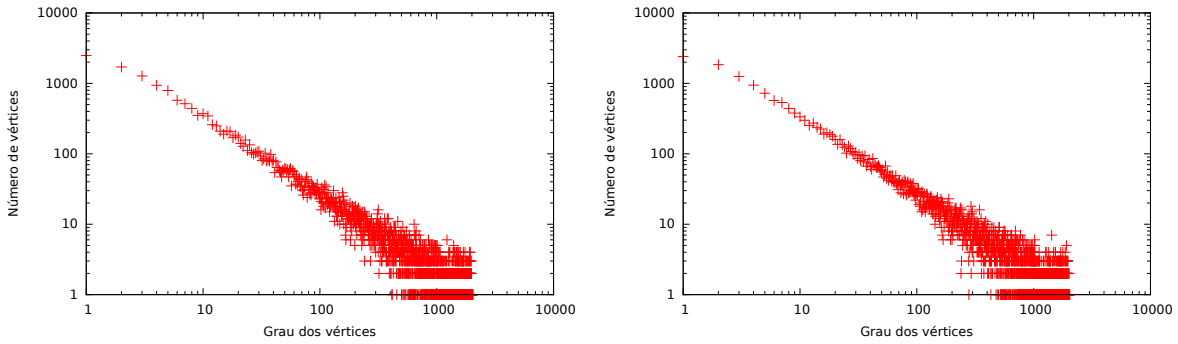
Equação 5.3. A saída é um grafo aleatório generalizado não direcionado com distribuição de graus esperada. Note que o grafo G resultante é um grafo simples e direcionado, mas o algoritmo pode ser facilmente alterado para gerar grafos não direcionados. Concentramos o interesse no estudo com grafos direcionados para este trabalho.

7.2.1 Conectividade

Os grafos gerados usando o Algoritmo 8 possuem distribuição dos graus do modelo $P(\alpha, \beta)$. Com isso, temos a vantagem de poder estimar algumas características estruturais da rede como, por exemplo, o número de componentes conexos do grafo. Experimentalmente, constatamos vários destes resultados antecipados por Aiello *et al.* [1], alguns destes já foram observados na Seção 7.1 sobre os resultados do algoritmo Distribuição de Graus.

Além disso, Aiello *et al.* [1] mostra que no modelo $P(\alpha, \beta)$ para $\beta < 3,47875...$ existe com quase certeza um componente gigante, isto é, um componente de tamanho $\Theta(n)$ no grafo. Quando $0 < \beta < 1$ o grafo quase sempre é conexo, o que significa que quanto menor é o β mais conexo é o grafo. Em concordância com o modelo, nos experimentos realizados, além da presença de um componente gigante, notamos que para $\beta < 3$ o componente gigante extraído de cada grafo gerado também segue uma lei de potência, de modo que o gráfico da distribuição de graus plotado em escala log-log mantém a reta característica do modelo, ao passo que na escala normal a curva tem decaimento de cauda longa. Da mesma forma, o tamanho do componente diminui quando β aumenta.

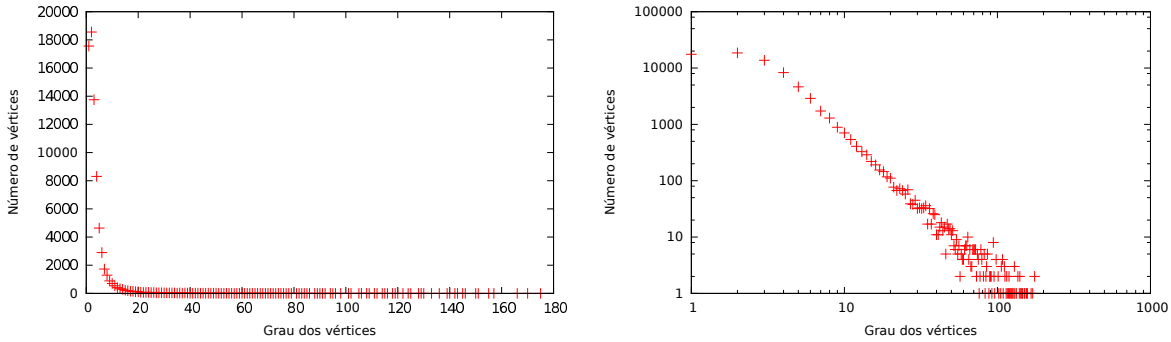
Com base nisso, optamos por realizar os experimentos no subgrafo induzido pelo componente gigante, ignorando os componentes menores. Visto que podemos trabalhar com o componente gigante sem perda de generalidade pois, continuamos com um grafo lei de potência, mas sabendo que agora ele é conexo, o que pode ser interessante para o estudo do problema.



(a) Grafo Aleatório (b) Componente Gigante

Figura 7.1: Distribuição de graus em escala logarítmica de um grafo com $\beta = 1,1$, $n = 22.335$ e $m = 1.833.964$.

A Figura 7.1 ilustra o exemplo de um grafo lei de potência com $\beta = 1,1$ gerado pelo Algoritmo 8, no qual o número de vértices é 22.335 e o número de arestas é 1.833.964. A entrada do Algoritmo 7 foi $n = 25.000$ e $\beta = 1,1$, a diferença no número de vértices do grafo resultante acontece por causa do arredondamento realizado na linha 12 deste algoritmo. O valor de β foi escolhido com o objetivo de destacar que existe componente gigante. O gráfico (a) mostra a distribuição de graus do grafo inteiro. O componente gigante, no gráfico (b), possui 22.331 vértices. Note que praticamente não há diferença visual no gráfico pois o grafo inteiro e o componente gigante são praticamente do mesmo tamanho.



(a) Escala normal (b) Escala log-log

Figura 7.2: Distribuição de graus em escala normal e logarítmica de um grafo com $\beta = 2,5$, $n = 74.037$ e $m = 145.766$.

Para fins de exemplo, temos na Figura 7.2 os gráficos que mostram a distribuição de graus em escala normal e na escala logarítmica de um grafo direcionado com $n = 74.037$ vértices e $m = 145.766$ arestas, onde o parâmetro de escala é $\beta = 2,5$. Este valor para β está nos moldes dos grafos gerados para os experimentos do Capítulo 8. Aqui os parâmetros de entrada do Algoritmo 7 foram $n = 100.000$ e $\beta = 2,5$, onde o arredondamento da linha 12 apresenta uma redução maior do número de vértices por que o valor β é maior do que no exemplo anterior. No primeiro gráfico (a), é possível reparar o decaimento de cauda longa, e no segundo (b) uma linha reta, como era de se esperar. Perceba que em comparação com o componente gigante da Figura 7.1 na escala logarítmica, a linha do componente gigante é muito mais cheia. Esta diferença existe devido o valor de β em cada um deles, pois o grafo da Figura 7.1 possui muito mais arestas do que o da Figura 7.2, porque quanto menor é o β mais arestas o grafo possui.

7.2.2 Sobre a Validação

Embora seja uma prática comum identificar distribuições de lei de potência pelo comportamento aproximado de uma linha reta do histograma no plot logarítmico, temos consciência que só este fato pode não ser suficiente. Como aponta Clauset *et al.* [8], tal

comportamento de linha reta é uma condição necessária, mas não significa que é o único requisito para determinar que uma distribuição segue o comportamento de lei de potência.

Os principais motivos para não considerarmos análises mais rigorosas sobre a validação da lei de potência dos grafos gerados pelo Algoritmo 8 são diversos. Primeiramente, porque isto não é uma tarefa trivial, como mostrado no trabalho de Clauset *et al.* [8], onde se apresenta um conjunto de técnicas com princípios estatísticos que permitem a validação e a quantificação de lei de potência. Segundo, porque nos baseamos em modelos já estabelecidos na literatura de geração de grafos, para gerar redes com estas características. Portanto, além da linha reta presente na escala logarítmica, usamos a garantia dos modelos discutidos em Chung e Lu [7] e Aiello *et al.* [1] para considerar os grafos gerados aqui como grafos aleatórios lei de potência.

CAPÍTULO 8

AVALIAÇÃO

Para compreender o comportamento da otimização por pré seleção na prática, conduzimos experimentos tanto em redes do mundo real quanto em redes artificiais. Para fins de avaliação de desempenho, comparamos com algoritmos de aproximação e heurísticas da literatura. A comparação foi feita levando em conta dois aspectos principais: tamanho do conjunto de vértices alcançados pela propagação de influência (isto é, qualidade do conjunto semente) e tempo de execução. Nos experimentos observamos que o algoritmo proposto nesta dissertação obteve ganhos expressivos de desempenho sobre uma das otimizações mais conhecidas do algoritmo guloso, o CELF [22], preservando a propagação esperada num nível competitivo.

O código foi escrito em Java usando a biblioteca JGraphT (jgrapht.org) e todos os experimentos foram executados em uma máquina com GNU/Linux (LinuxMint 17) cujas configurações de hardware são:

- Processador: Intel(R) Xeon(R) 3.00GHz com 25 MB de cache e 16 núcleos.
- Memória RAM: DDR3 com 8 GB.

8.1 Conjunto de Dados

Buscamos utilizar redes que apresentem características estruturais de redes sociais de grande escala, assim como distribuição de graus que seja de lei de potência. Na metodologia de testes adotada, os testes foram realizados em duas das redes sociais mais utilizadas nos trabalhos da área, como por exemplo, em [22, 15, 14, 19], às quais chamamos de *NetHEPT* e *NetPHY*. Mais detalhes sobre cada uma delas estão na Sessão 8.1.1.

8.1.1 Grafos Reais

Ambos os grafos, NetHEPT e NetPHY, são redes de coautoria de artigos, cujas informações foram retiradas do banco de dados do site arxiv.org. A rede NetHEPT (*High Energy Physics - Theory*) foi retirada da seção de física teórica de alta energia do arXiv. A segunda, é a rede NetPHY (*Physics*), que foi retirada da sessão de física.

Os dados representam informações dos anos de 1991 à 2003 e foram modelados da seguinte maneira. Um vértice é criado para cada pesquisador que tem pelo menos um artigo em conjunto com outros autores, artigos com um único autor são ignorados. Para cada artigo com dois ou mais autores, uma aresta é inserida entre todo par de autores

[14, 4, 19]. Note que vão existir arestas paralelas quando dois pesquisadores tem coautoria em artigos diferentes, mas neste trabalho arestas paralelas são descartadas. Deste modo, os grafos utilizados são grafos simples, no sentido que não possuem arestas paralelas nem laços. Alguns dados desses grafos são resumidos na Tabela 8.1.

Tabela 8.1: Dados dos grafos NetHEPT e NetPHY

Rede Social	$\|V\|$	$\|E\|$	$\|C\|$	Expoente (β)
NetHEPT	15.233	32.213	4.208	2,651
NetPHY	37.154	180.826	8.428	2,843

Os símbolos $|V|, |E|$ e $|C|$ denotam número de vértices, número de arestas e tamanho do conjunto dos candidatos, respectivamente. Lembrando que o conjunto dos candidatos ao qual se refere a tabela, foi obtido por meio do Algoritmo 5. Os valores de expoente β foram retirados do trabalho de Liu *et al.* [24].

8.1.2 Grafos Artificiais

Os grafos utilizados nessa fase da avaliação são aleatórios, conexos, direcionados e seguem distribuição lei de potência nos graus, construídos a partir do modelo de geração de grafo generalizado descrito no Capítulo 7. Foram gerados grafos com os tamanhos de 4 mil, 8 mil e 16 mil vértices. Para cada tamanho, foram gerados 10 grafos e o tempo de execução e propagação resultante é a média entre a simulação nestes 10 grafos.

No Capítulo 7 vimos que o valor do expoente β tem relação com a conectividade do grafo, e que podemos estimar o tamanho do componente gigante dos grafos aleatórios de acordo com o β escolhido. Definimos $\beta = 2,5$ para todos os grafos gerados. Deste modo, espera-se que o grafo resultante do Algoritmo 8 contenha um componente gigante. Isso nos permite usar grafos conexos, por questões de simplicidade. Portanto, nos experimentos, são usados os grafos induzidos pelo componente gigante dos grafos gerados artificialmente. Isto é, seja G o grafo resultante do Algoritmo 8, e seja G' o grafo no qual os algoritmos são avaliados. Obtemos G' como $G' \leftarrow G[CG]$, onde CG é o componente gigante do grafo G.

8.2 Modelo de Influência e Probabilidades

Todos os testes foram realizados no modelo de cascata independente. Para atribuir o peso de influência de cada aresta, optamos por um método simplista em que as probabilidades são distribuídas em três configurações distintas. A primeira configuração se mantém fiel às definições do modelo em questão, na qual, o peso de influência em cada aresta é um valor escolhido aleatoriamente de maneira uniforme no intervalo [0,1]. A segunda e

a terceira são consideradas casos especiais da cascata independente. Na segunda configuração, atribui-se uma probabilidade p fixa para todas as arestas, neste caso escolhemos $p = 0,025$. A terceira configuração do modelo tenta combinar o valor probabilístico da primeira configuração conservando uma probabilidade baixa de propagação. Deste modo, os pesos são escolhidos aleatoriamente no intervalo $[0, 1/4]$.

O motivo do uso de tais configurações segue. Na primeira, a propagação é muito grande, uma vez que a probabilidade média de 0,5 é uma probabilidade relativamente alta, fazendo com que mais vértices sejam atingidos na propagação e consequentemente simulações mais demoradas, fator que inviabiliza a realização de experimentos para grafos grandes, este fato é apresentado com mais detalhes na Seção 8.4. Na segunda configuração, é resolvido o problema da alta probabilidade diminuindo a escala de propagação, e assim, a simulação se torna mais rápida. Esse método é o mais utilizado na literatura da área [33, 5, 4], porém, implica que os vértices de maior grau têm mais probabilidade de influenciar muitos vértices do que vértices de grau menor, característica verificada na Seção 8.4. A terceira aborda o problema de ambas as anteriores, mantendo uma probabilidade que não é muito alta, porém não fixa.

8.3 Algoritmos

Além da otimização por pré seleção apresentada no Capítulo 6, outras três soluções de mesma finalidade foram avaliadas: a otimização CELF 3, uma heurística baseada no grau dos vértices e a escolha de vértices aleatórios como conjunto semente. Apesar de alguns experimentos terem sido realizados com algoritmo guloso original (sem otimização), os resultados não foram relatados porque a propagação da influência é a mesma obtida com a otimização CELF enquanto o tempo de execução é muito maior, como foi mostrado por Leskovec *et al.* [22]. Segue uma lista com mais detalhes sobre os algoritmos implementados.

- **HighDegree:** Heurística baseada na noção de *centralidade de grau*. É comumente utilizada em trabalhos da sociologia, considerando os vértices de maior grau como mais influentes [19, 4]. Desta forma, nesta heurística, escolhemos os k vértices com maior grau de saída, ignorando o grau de entrada dos vértices, uma vez que o grau de entrada só está relacionado com as chances do vértice ser influenciado, mas não de influenciar.

- **RandomSeed**: Escolhe k vértices aleatoriamente, sem reposição, de maneira uniforme.

- **Celf:** É o algoritmo guloso $(1 - 1/e)$-aproximado de Kempe *et al.* [19], mas com a otimização *Lazy Forward* [22] apresentada no Capítulo 4. Implementado em confor-

midade com a literatura, com 10.000 simulações Monte Carlo para estimar o valor de σ.

- **PrevalentSeed:** Otimização por pré seleção, proposto no Capítulo 6 e descrito no Algoritmo 6.

Tanto para escolher os vértices com algoritmos gulosos como para avaliar a qualidade dos vértices escolhidos, é necessário calcular o valor da propagação esperada $\sigma(S)$. Como foi discutido no Capítulo 3, ainda é uma questão aberta, onde boas estimativas podem ser obtidas através de simulações do processo aleatório. Neste caso, simulamos 10.000 vezes o processo para cada conjunto escolhido. O número de simulações escolhido segue os trabalhos da literatura, onde [19] indica que com 10.000 iterações, obtêm-se um resultado semelhante ao obtido com 30.000 iterações ou mais.

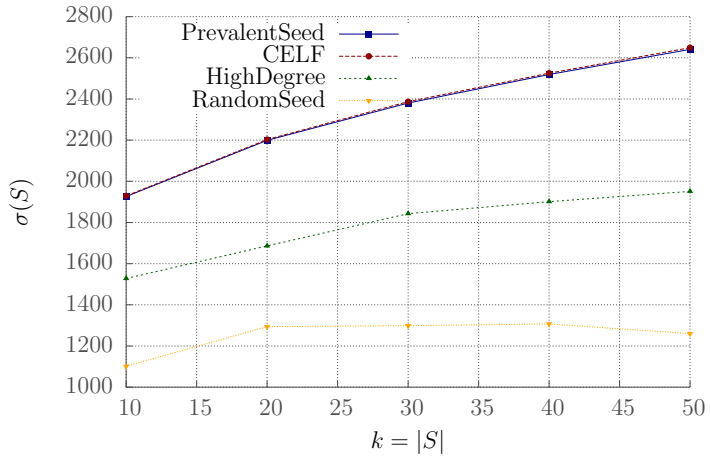
Executamos todos os algoritmos com diversos tamanhos k de adeptos iniciais, e comparamos o desempenho para todo $k \in \{10, 20, ..., 50\}$.

8.4 Resultados

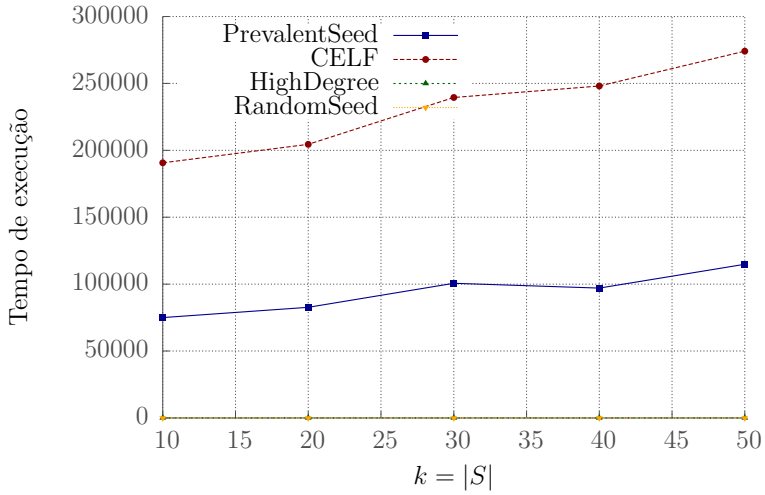
Comparamos o desempenho dos quatro algoritmos em duas métricas: qualidade do conjunto semente e tempo de execução. A qualidade dos conjuntos escolhidos pelos algoritmos é avaliada com base no número de vértices ativados a partir dos adeptos inicias. Quanto maior a propagação, melhor é a qualidade. Para que as comparações sejam justas, executamos 10.000 simulações Monte Carlo para obter a propagação de todos os algoritmos. No tempo de execução, os algoritmos HighDegree e RandomSeed terminaram quase que instantaneamente em todos os cenários, usando menos de 1 segundo de processamento. Por esse motivo, não são visíveis na escala de tempo de execução em que os gráficos foram gerados.

A Figura 8.1 mostra o desempenho dos algoritmos na rede NetHEPT com probabilidades do intervalo [0,1] o PrevalentSeed obteve uma vantagem sobre o Celf de até 58,08% do tempo de processamento (veja no gráfico (b)), ao passo que manteve a qualidade da propagação alcançada equiparável ao Celf, como pode ser observado no gráfico (a). A Figura 8.2 ilustra melhor a diferença de tempo em um gráfico de barras. Outras redes não foram avaliadas da mesma forma devido à inviabilidade de tempo, uma vez que para estes resultados foram necessárias mais de 76 horas de processamento para o Celf encontrar 50 vértices e 31 horas para o PrevalentSeed, o que torna proibitivo a realização de simulações com grafos muito grandes.

Os gráficos das Figuras 8.3 e 8.4 mostram testes em que todas as probabilidades de influência são iguais ($p = 0,025$). Note que existe diferença na escala do gráfico da propagação (gráfico (a)) de ambas as figuras em comparação com a Figura 8.1. Esta estratégia é convencionalmente utilizada na literatura [19, 4, 5] porque torna mais viável realizar as



(a) Propagação esperada



(b) Tempo de execução

Figura 8.1: Grafo NetHEPT com probabilidades entre 0 e 1: (a) propagação esperada; (b) tempo de execução em segundos

avaliações. Na Figura 8.3, os resultados também são positivos para PrevalentSeed, que encontrou um conjunto semente de 50 vértices com 69,6% do tempo utilizado pelo Celf. Mas, na Figura 8.4 temos um resultado inesperado no grafo NetHEPT. Neste experimento, a heurística HighDegree teve maior propagação esperada do que todos os outros algoritmos, além de ser extremamente rápida.

Ao investigar as possíveis razões pelas quais a heurística HighDegree se comportou tão bem, notamos que no modelo de cascata independente com probabilidade p fixa nas arestas, os vértices com grau maior têm mais chance de ativar muitos outros do que os vértice de grau menor, o que explica o resultado do gráfico (a) da Figura 8.4. Este fato já havia sido mencionado em [19]. Por esse motivo Kempe *et al.* [19] usa como alternativa outro caso especial do modelo de cascata independente, chamado "*Weighted Cascade*",

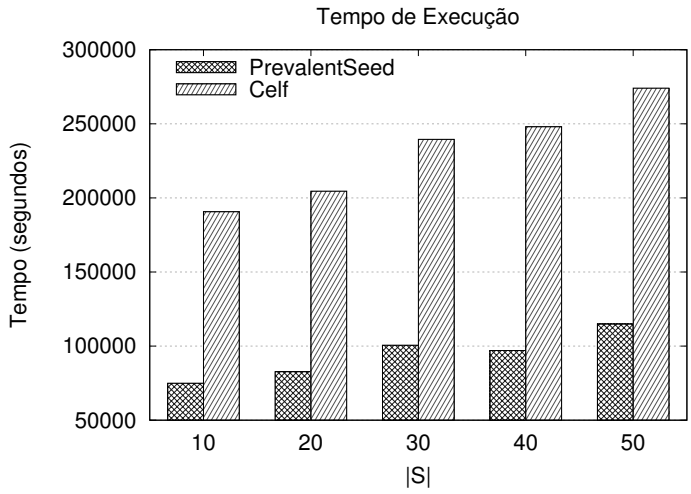


Figura 8.2: Tempo de execução dos algoritmos PrevalentSeed e Celf na rede NetHEPT.

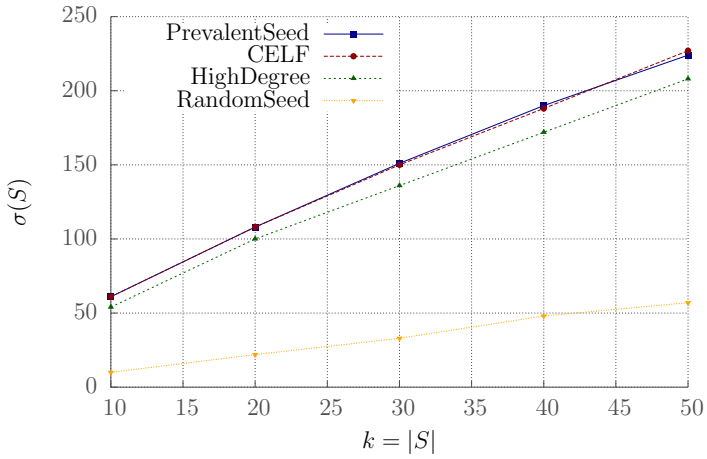
em que cada aresta de u para v recebe uma probabilidade $1/\delta(v)$ de ativar v. O modelo *Weighted Cascade* não é estudado neste trabalho.

Os gráficos da Figura 8.5 exibem os resultados dos experimentos que avaliam os algoritmos nos grafos aleatórios gerados artificialmente. Como já foi descrito na Seção 8.1.2, os gráficos ilustram a média de desempenho entre 10 grafos de cada tamanho e os pesos de influência são sorteados no intervalo $[0, 1/4]$. Note que quanto menor o grafo, mais parecida é a propagação de influência dos três primeiros algoritmos. No gráfico (a) o PrevalentSeed apresentou um resultado levemente melhor que os outros dois seguintes, Celf e HighDegree, com tempo melhor que o do Celf (gráfico (b)). Os gráficos (c) e (d), mostram que, à medida que o grafo cresce, a heurística de grau maior perde em qualidade, enquanto o ganho de tempo do PrevalentSeed torna-se maior que o Celf, como pode ser visto nos gráficos (e) e (f).

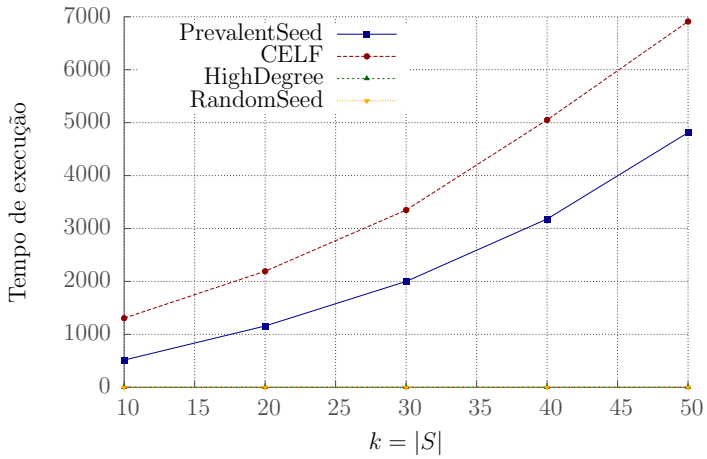
A Tabela 8.2 resume a efetividade do PrevalentSeed em relação ao tempo do Celf quando o tamanho do conjunto semente é de 50 vértices. Por simplicidade foram mantidas apenas duas casas decimais de precisão. A última coluna exibe o quanto o PrevalentSeed foi mais rápido do que o Celf. Neste caso, a diminuição de tempo se manteve entre 20,56% e 36,36%. Tais resultados sugerem que quanto maior a rede em que se aplica os algoritmos, melhor é o ganho de tempo.

Tabela 8.2: Tempo de execução (em segundos) no grafos aleatórios 4 mil, 8 mil e 16 mil vértices.

Nº de vértices	PrevalentSeed	Celf	Ganho obtido
4k	719,19	905,43	20,56%
8k	2436,94	3829,81	36,36%
16k	6401,26	9094,71	29,61%



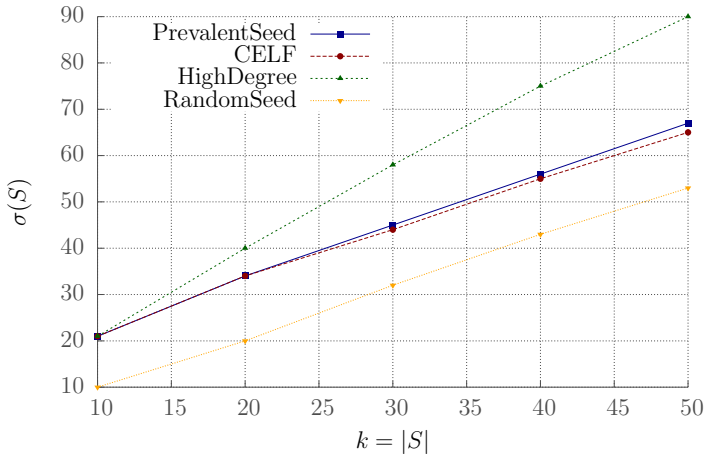
(a) Propagação esperada



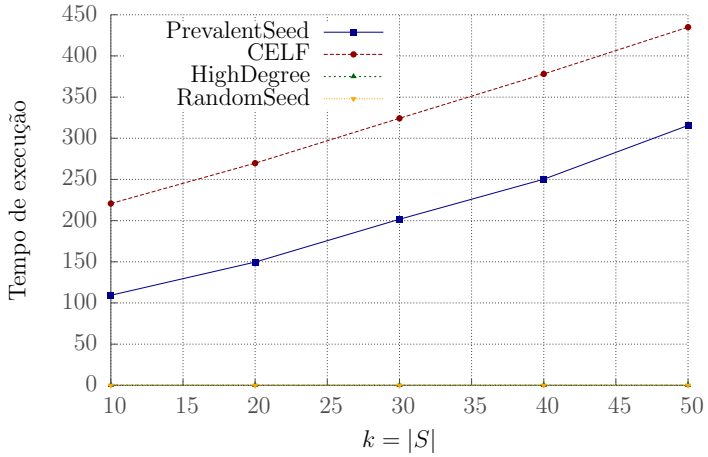
(b) Tempo de execução

Figura 8.3: Grafo NetPHY com probabilidade $p = 0,025$: (a) propagação esperada; (b) tempo de execução em segundos

Na primeira parte dos experimentos, os algoritmos foram avaliados apenas no grafo NetHEPT porque as avaliações com probabilidades no intervalo [0,1] são mais demoradas do que no restante do experimento. Isso acontece porque esta distribuição de probabilidades permite um número maior de ativações, ou seja, como cada aresta tem um peso $p \in [0, 1]$, temos que cada vértice tem uma probabilidade esperada igual a 0,5 de ativar seus vizinhos de saída. Já para a segunda parte dos experimentos, usamos uma probabilidade fixa, assim, os experimentos puderam ser realizados em tempo menor. O motivo é que uma vez que fixamos $p = 0,025$, cada vértice ativo tem uma probabilidade esperada de ativar seus vizinhos igual a p, que é menor do que no caso anterior. Consequentemente, com um p fixo, as ativações terminam antes do que o caso com p não fixo, pois isso reduz o número de vértices ativados em cada simulação. Desta forma, todos os algoritmos demoram mais para terminar na primeira parte do que na segunda, mas é possível notar



(a) Propagação Esperada



(b) Tempo de Execução

Figura 8.4: Grafo NetHEPT com probabilidade $p = 0,025$: (a) propagação esperada; (b) tempo de execução em segundos

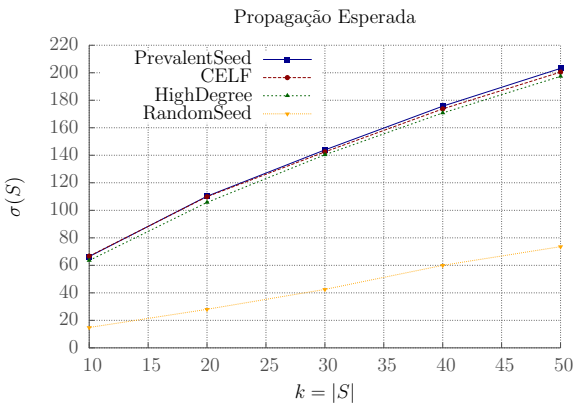
pelo gráfico (b) da Figura 8.1 que a diferença entre tempo de execução do Celf e do PrevalentSeed é mais evidente do que nos demais gráficos. Por fim, nos experimentos com grafos gerados artificialmente, definimos que os pesos são escolhidos aleatoriamente de maneira uniforme, tal que $p \in [0, \frac{1}{4}]$. Assim, foi possível comparar os resultados em grafos de vários tamanhos e a diferença de tempo continua perceptível.

8.4.1 Discussão a respeito dos resultados

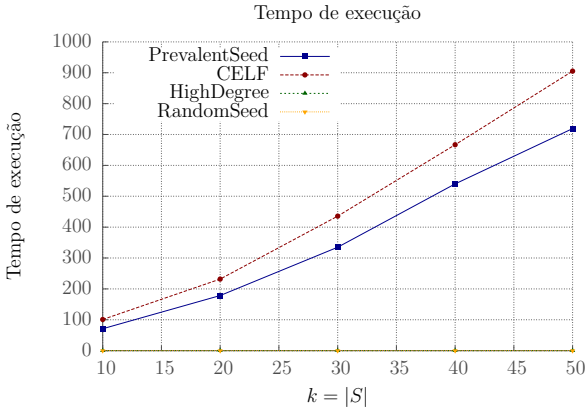
Os resultados mostram que o algoritmo PrevalentSeed é razoavelmente mais rápido do que o Celf. Isto se deve principalmente ao fato de que o número médio de cálculos de propagação esperada por iteração Monte Carlo é menor, uma vez que o número de vértices examinados é o tamanho do conjunto de candidatos do grafo. Nas redes reais NetHEPT e

NetPHY (Figuras 8.1 e 8.3) a redução no tempo de processamento foi muito mais expressiva. Além disso, os conjuntos de vértices influenciados resultantes do PrevalentSeed são muito competitivos com os encontrados pelo Celf em termos de qualidade.

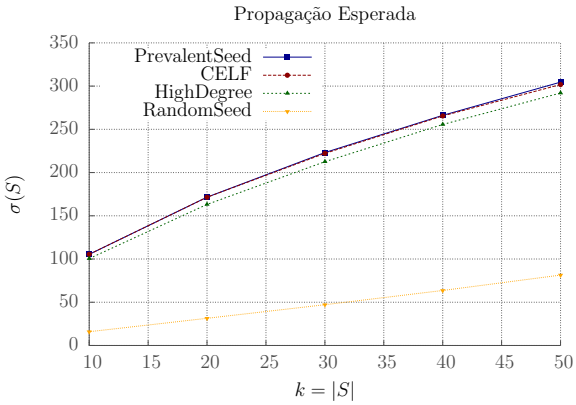
Algumas vantagens do nosso algoritmo, o PrevalentSeed, podem ser consideradas a partir dos resultados mostrados nas seções anteriores. Primeira, além de manter a qualidade da propagação de influência equiparável à do algoritmo guloso $(1 - 1/e)$-aproximado, ainda reduzimos o tempo de execução do algoritmo otimizado, tal redução se manteve entre 20% e 58% nos cenários avaliados. Segunda, a nossa otimização melhora a escalabilidade em relação ao Celf, permitindo que este seja aplicável em redes maiores, uma vez que quanto maior o grafo, mais aumenta a diferença no tempo de execução, como foi visto nos experimentos feitos com grafos gerados artificialmente.



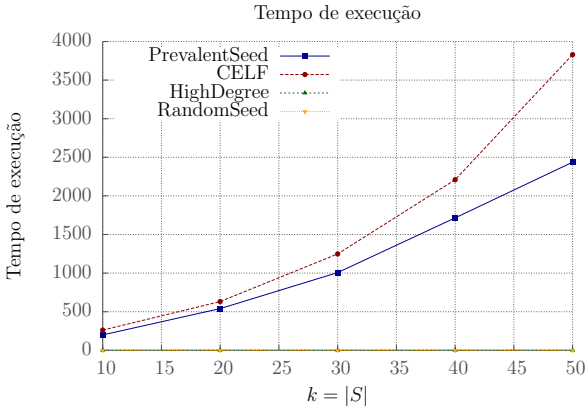
(a) 4 mil vértices



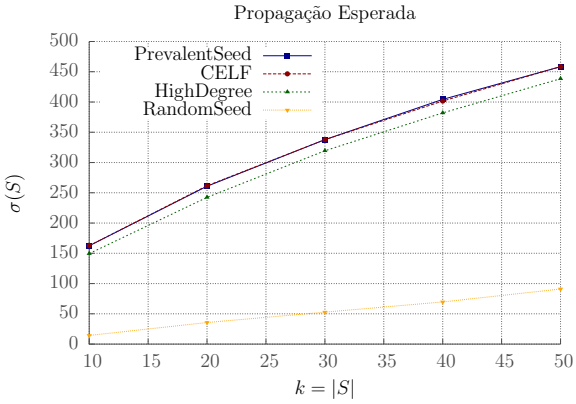
(b) 4 mil vértices



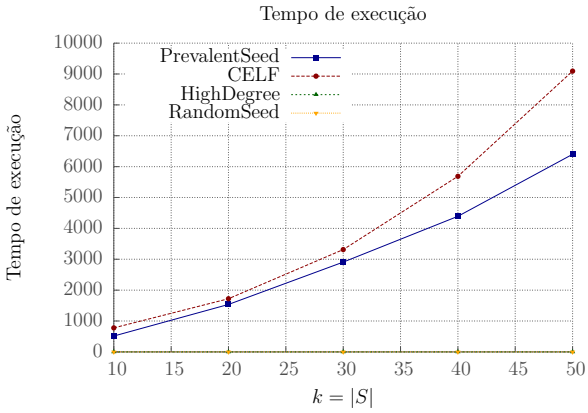
(c) 8 mil vértices



(d) 8 mil vértices



(e) 16 mil vértices



(f) 16 mil vértices

Figura 8.5: Grafos aleatórios de 4 mil, 8 mil e 16 mil vértices e probabilidades aleatórias no intervalo $[0, 1/4]$.

CAPÍTULO 9

CONCLUSÃO

Neste trabalho mostramos empiricamente que nossa metodologia de pré seleção de vértices influentes combinada ao CELF, a qual chamamos de PREVALENTSEED (Algoritmo 6), reduz significativamente o tempo de execução do algoritmo guloso proposto por Kempe *et al.* [19], de modo que os resultados referentes a qualidade da solução se mantiveram semelhantes aos que são obtidos com o próprio CELF. Ainda sobre a qualidade da propagação, a análise teórica fornece evidências que vão de acordo com os resultados da análise empírica.

Entre as vantagens obtidas com a solução que foi proposta nesta dissertação, estão:

- Redução do tempo de execução do algoritmo guloso de Kempe *et al.* [19] em grafos lei de potência.
- Durante o trabalho, observamos que existe uma relação entre a propagação de influência e a distribuição de graus de uma rede social. A pré seleção usa esta relação para descartar alguns vértices menos importantes na busca.
- A pré seleção busca evitar processamento desnecessário. Assim, realiza a busca apenas na parte considerada mais promissora dos vértices.
- A redução de tempo melhora a escalabilidade do algoritmo guloso e permite o uso desta estratégia gulosa em grafos maiores.
- Na prática, a natureza linear do algoritmo de pré seleção não compromete o tempo de execução do algoritmo final. Isso acontece porque o tempo de processamento que é economizado fazendo menos chamadas à função σ, supera o custo da pré seleção.

Em relação ao fato de utilizarmos simulações Monte Carlo, temos consciência de que mesmo com a redução na quantidade de simulações como foi feita, este método ainda é demasiado custoso. Mesmo assim, esta ainda é uma estratégia confiável, devido ao nível de precisão obtido na estimativa de propagação. Porém, não descartamos o uso dos mesmo princípios obtidos neste trabalho, para propostas futuras de calcular a estimativa de propagação sem o uso de simulações Monte Carlo.

9.1 Trabalhos Futuros

Existem muitas direções em que este problema pode ser explorado. O que propomos aqui, foi investigar uma estrutura de rede específica e utilizá-la para facilitar a maximização

de influência nestas redes. Trabalhos futuros podem usar este mesmo caminho empregando técnicas mais avançadas. Entre as possíveis direções que a pesquisa realizada neste trabalho pode seguir, estão:

- Encontrar limitantes teóricos para a pré seleção: fazer uma análise teórica mais precisa sobre o tamanho esperado do conjunto resultante da pré seleção ou provar limitantes para este resultado, uma vez que os grafos gerados possuem distribuição de graus conhecida.

- Calcular a estimativa de propagação: estender a pesquisa para calcular a estimativa de propagação sem o uso de simulações Monte Carlo em grafos aleatórios de lei de potência, utilizando informações como os parâmetros α e β dos grafos lei de potência.

- Extrair pesos de influência em redes reais: usar dados de interações reais para extrair os pesos de influência que uns usuários exercem sobre os demais, servindo-se de métodos computacionais que permitam detectar padrões de comportamento para reconhecer tais influencias entre usuários de uma rede social.

- Provar um fator de aproximação: buscar resultados teóricos com base nas evidências levantadas sobre um fator de aproximação para o algoritmo PrevalentSeed ou provar que este é $(1 - \frac{1}{e})$-aproximado.

Quanto a este último ponto, no Capítulo 6 defendemos que há evidências de que o algoritmo PrevalentSeed é $(1 - \frac{1}{e})$-aproximado do ótimo. Além disso, a semelhança observada nos experimentos, no que se refere à qualidade do conjunto semente fortalece esta suposição. Caso esta tese seja comprovada, este é um resultado importante não só para o problema de maximização de influência em si, mas também para funções de maximização submodulares e monotônicas em geral. Desde que estas funções possam ser resolvidas em grafos lei de potência direcionados.

Portanto, além de tentar reduzir o tempo de execução, estamos interessados em encontrar heurísticas mais efetivas para obter a estimativa de propagação, uma vez que formular e resolver estes problemas com modelos mais práticos e encontrar algoritmos mais eficientes é um desafio fascinante e com muito potencial.

BIBLIOGRAFIA

[1] William Aiello, Fan Chung, e Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

[2] Yaniv Altshuler, Wei Pan, e Alex Sandy Pentland. Trends prediction using social diffusion models. *Social computing, behavioral-cultural modeling and prediction*, páginas 97–104. Springer, 2012.

[3] Tom Britton, Maria Deijfen, e Anders Martin-Löf. Generating simple random graphs with prescribed degree distribution. *Journal of Statistical Physics*, 124(6):1377–1397, 2006.

[4] Wei Chen, Chi Wang, e Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 1029–1038. ACM, 2010.

[5] Wei Chen, Yajun Wang, e Siyu Yang. Efficient influence maximization in social networks. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 199–208. ACM, 2009.

[6] Wei Chen, Yifei Yuan, e Li Zhang. Scalable influence maximization in social networks under the linear threshold model. *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, páginas 88–97. IEEE, 2010.

[7] Fan Chung e Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

[8] Aaron Clauset, Cosma Rohilla Shalizi, e Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[9] Pedro Domingos e Matt Richardson. Mining the network value of customers. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 57–66. ACM, 2001.

[10] Roman J Dwilewicz e Ján Minác. Values of the riemann zeta function at integers. *Materials matemàtics*, páginas 0001–26, 2009.

[11] David Easley e Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[12] Stephen Eubank, Anil Kumar, Madhav Marathe, Aravind Srinivasan, e Nan Wang. Structural and algorithmic aspects of massive social networks. *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, páginas 718–727. Society for Industrial and Applied Mathematics, 2004.

[13] Amit Goyal, Francesco Bonchi, e Laks VS Lakshmanan. A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment*, 5(1):73–84, 2011.

[14] Amit Goyal, Wei Lu, e Laks VS Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. *Proceedings of the 20th international conference companion on World wide web*, páginas 47–48. ACM, 2011.

[15] Amit Goyal, Wei Lu, e Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, páginas 211–220. IEEE, 2011.

[16] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, páginas 1420–1443, 1978.

[17] Peter Hedström e Peter Bearman. *The Oxford handbook of analytical sociology*. OUP Oxford, 2009.

[18] Matthew O Jackson. *Social and economic networks*. Princeton University Press, 2010.

[19] David Kempe, Jon Kleinberg, e Éva Tardos. Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 137–146. ACM, 2003.

[20] David Kempe, Jon Kleinberg, e Éva Tardos. Influential nodes in a diffusion model for social networks. *Automata, languages and programming*, páginas 1127–1138. Springer, 2005.

[21] Jon Kleinberg. Cascading behavior in networks: Algorithmic and economic issues. *Algorithmic game theory*, 24:613–632, 2007.

[22] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, e Natalie Glance. Cost-effective outbreak detection in networks. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 420–429. ACM, 2007.

[23] Yishi Lin e John Lui. Algorithmic design for competitive influence maximization problems. *arXiv preprint arXiv:1410.8664*, 2014.

[24] Xiaodong Liu, Shanshan Li, Xiangke Liao, Shaoliang Peng, Lei Wang, e Zhiyin Kong. Know by a handful the whole sack: efficient sampling for top-k influential user identification in large graphs. *World Wide Web*, 17(4):627, 2014.

[25] Michael Mitzenmacher e Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[26] Seth A Myers, Chenguang Zhu, e Jure Leskovec. Information diffusion and external influence in networks. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 33–41. ACM, 2012.

[27] George L Nemhauser, Laurence A Wolsey, e Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.

[28] Mark Newman. Random graphs as models of networks. *Handbook of Graphs and Networks: From the Genome to the Internet*, 2006.

[29] Mark Newman. *Networks: an introduction*. Oxford University Press, 2010.

[30] Everett M Rogers. *Diffusion of innovations*. Simon and Schuster, 2010.

[31] Bryce Ryan e Neal Crasilneck Gross. *Acceptance and diffusion of hybrid corn seed in two Iowa communities*, volume 372. Agricultural Experiment Station, Iowa State College of Agriculture and Mechanic Arts, 1950.

[32] Paulo Shakarian, Joseph Salmento, William Pulleyblank, e John Bertetto. Reducing gang violence through network influence based targeting of social programs. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 1829–1836. ACM, 2014.

[33] Xinfei Shi, Hongzhi Wang, Jianzhong Li, e Hong Gao. Influence spread maximization in social network. *International Journal of Information & Education Technology*, 3(6), 2013.

[34] Remco Van Der Hofstad. Random graphs and complex networks. *Disponível em http://www. win. tue. nl/rhofstad/NotesRGCN. pdf*, 2009.