Perfect Awareness Problem: modelo exato e heurísticas baseadas em GRASP

Felipe de Carvalho Pereira
Instituto de Computação, UNICAMP, Campinas, Brasil

Resumo. O projeto computacional relatado neste documento visou a utilização da meta-heurística GRASP e de um modelo exato para solucionar o problema de otimização combinatória *perfect awareness* (PA), um problema NP-difícil. No PA, considera-se uma rede social na qual uma informação é propagada entre indivíduos conectados. O objetivo é selecionar um conjunto inicial de disseminadores de menor tamanho possível, de modo que ao fim do processo de propagação todos os indivíduos sejam conhecedores da informação. Neste trabalho, foram propostas três heurísticas baseadas em GRASP para o PA, além de um modelo de programação linear inteira. As heurísticas propostas diferem entre si em relação à estratégia utilizada para a fase de construção do GRASP. Foram conduzidos um conjunto de experimentos com 17 instâncias já utilizadas na literatura do problema. A análise dos resultados mostrou pouca discrepância em relação ao desempenho das heurísticas, exceto para instâncias muito grandes. Para ao modelo exato, alcançou-se o valor ótimo de uma instância entre duas testadas. Além disso, dentre as três heurísticas propostas, duas apresentaram resultados superiores a um algoritmo existente na literatura, considerando o conjunto de instâncias testado.

Palavras-chave. perfect awareness problem, active spreading, social networks, GRASP, integer programming.

1. Introdução

Este documento consiste em um relatório do projeto computacional desenvolvido na disciplina "Tópicos de Otimização Combinatória" (MO824), ministrada pelo Prof. Dr. Fábio Luiz Usberti. O projeto visou resolver um problema de otimização combinatória NP-difícil chamado *perfect awareness* (PA) através da meta-heurística GRASP e de um modelo de programação linear inteira (PLI). O PA pertence a um conjunto de problemas que envolve a propagação de informações em redes sociais.

O estudo de redes sociais tem sido realizado por décadas desde o século passado. Recentemente, com o advento dos sites e aplicativos de redes sociais *online*, o interesse por essa área cresceu de forma extraordinária. Tal crescimento ocorreu principalmente em razão da disponibilidade de grandes quantidades de dados proporcionadas por tais redes [1].

Existem diversos modelos de propagação de informações em redes sociais na literatura. Para o PA considera-se um modelo no qual, em um dado instante de

tempo, cada indivíduo pode ser considerado *ignorant* (desconhecedor), *aware* (conhecedor) ou *spreader* (disseminador) em relação a um meme [2]. Um *meme* pode ser compreendido como uma unidade de informação cultural, ou mesmo como um padrão cognitivo ou comportamental, o qual pode ser transmitido de um indivíduo para outro [3].

No modelo aqui considerado, assumimos que inicialmente todos os indivíduos são desconhecedores de uma informação. A partir daí um conjunto de disseminadores é selecionado. Quando um disseminador informa um indivíduo v desconhecedor, v passa a ser um conhecedor. Além disso, assim que v é informado por um número de disseminadores maior ou igual a um *threshold* $t(v)$, v passa a ser um disseminador [2].

1.1. Perfect Awareness

Formalmente, o modelo utilizado em [2] para o PA consiste em uma rede social representada por um grafo simples não direcionado $G = (V, E)$, onde V corresponde ao conjunto de indivíduos e o conjunto de arestas E indica o relacionamento entre os membros da rede. Ou seja, $\{u, v\} \in E$ se os indivíduos (representados por) u e v podem se comunicar diretamente. Denota-se por $N_G(v)$ a vizinhança de v em G, i.e., $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$.

Neste modelo, também é considerada uma função de *threshold* $t : V \to \mathbb{N}_0 = \{0, 1, 2, \dots\}$. Conhecida esta função, pode-se descrever um processo de disseminação que começa com $S \subseteq V$ e consiste de uma sequência de subconjuntos de V, $\text{Spreader}_G[S, \tau]$, para $\tau = 0, 1, \dots$, tais que:

- $\text{Spreader}_G[S, 0] = S$,
- $\text{Spreader}_G[S, \tau] = \text{Spreader}_G[S, \tau - 1] \cup \{u : |N_G(u) \cap \text{Spreader}_G[S, \tau - 1]| \geq t(u)\}$, para $\tau \geq 1$.

Observe que a cada incremento no valor de τ, o conjunto de disseminadores cresce com todos os nós u para os quais o número de vizinhos que são disseminadores é pelo menos o *threshold* $t(u)$. O fim da propagação ocorre quando $\text{Spreader}_G[S, \rho] = \text{Spreader}_G[S, \rho - 1]$ para algum $\rho \geq 1$. Nesse caso, os conjuntos finais de disseminadores e conhecedores são respectivamente:

- $\text{Spreader}_G[S] = \text{Spreader}_G[S, \rho]$,
- $\text{Aware}[S] = \text{Spreader}_G[S] \cup \{u : N_G(u) \cap \text{Spreader}_G[S] \neq \emptyset\}$.

Ademais, um conjunto $S \subset V$ é chamado de *perfect seed set* se $\text{Aware}[S] = V$. Além disso, os nós em S são chamados de *seeds*.

Finalmente, dados um grafo $G = (V, E)$ e a função *threshold* $t : V \to \mathbb{N}_0$, o problema de encontrar um *perfect seed set* de tamanho mínimo é chamado de *perfect awareness* [2].

A figura 1 consiste em um exemplo de propagação. Em $\tau = 0$, os vértices em verde representam o *seed set*, ou seja, são os *seeders*. Quando $\tau = 1$, um dos vértices na vizinhança dos *seeders* se torna disseminador e passa a ser representado na cor

verde e outros dois se tornam conhecedores e tomam a cor amarela. A propagação continua até que todos os vértices sejam conhecedores ou disseminadores.

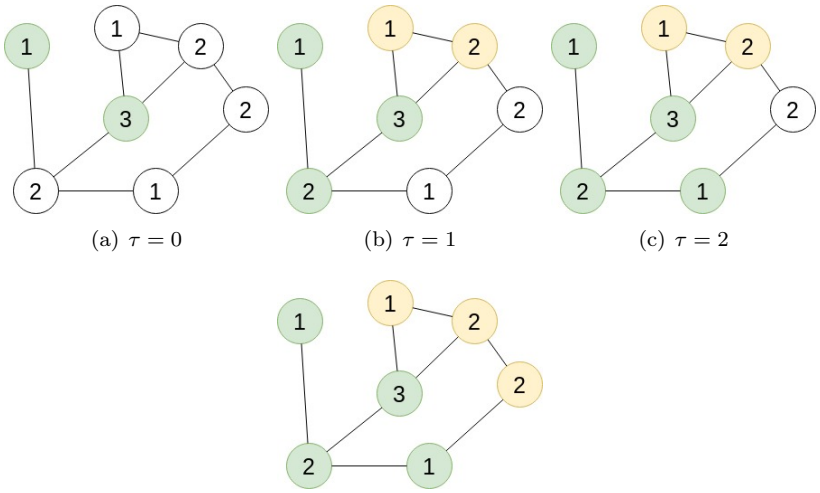

Figura 1: Exemplo de propagação. Os vértices brancos são desconhecedores, verdes são disseminadores e amarelos são conhecedores. Os números dentro das circunferências representam seus *thresholds*.

1.2. Estado da arte

Embora tenha sido lançado recentemente na literatura, já existem resultados e algoritmos computacionais propostos para o PA. O primeiro resultado mostra que o problema não pode ser aproximado por um fator de $O(2^{\log^{1-\epsilon} n})$ para qualquer $\epsilon > 0$, a menos que NP $\subseteq$ DTIME($n^{polylog(n)}$) [2]. A prova é baseada em uma redução a partir do problema *minimum target set selection* (TSS) para o PA, a fim de estender o resultado que fora originalmente provado para o TSS em [4].

O TSS utiliza um modelo de propagação semelhante ao PA, de modo que o objetivo é encontrar um $S \in V$ de tamanho mínimo tal que $\text{Spreader}_G[S] = V$. Ou seja, ao fim da propagação, todos os indivíduos devem ser disseminadores. Outros resultados computacionais para o PA para grafos densos e para grafos de Ore foram demonstrados em [2]. Note que um grafo G é dito um grafo de Ore se a soma dos graus de quaisquer vértices independentes é pelo menos igual a V, isto é, $|N_G(u)| + |N_G(v)| \geq |V|$, para todo $u, v \in V$, tal que $(v, u) \in E$.

Dois algoritmos foram propostos em [2] para o PA. O primeiro deles é o algoritmo exato TREE-PA(T, t) que devolve um *perfect seed set* mínimo para uma instância qualquer do PA, desde que o grafo T dado seja uma árvore. A ideia geral do algoritmo é obter o *seed set* enquanto se realizam visitas (na ordem inversa de uma busca em largura) nos vértices da árvore T, tentando adiar ao máximo a adição de

nós ao *seed set*. O TREE-PA(T, t) tem complexidade de tempo linear no tamanho da entrada.

O segundo algoritmo, o PA(G, t), itera de maneira gulosa sobre os vértices do grafo G fornecido, realizando depreciações nos vértices visitados, de forma que quando uma determinada condição ocorre, um certo nó é adicionado ao *seed set*. O algoritmo devolve um *perfect seed set* para um grafo G arbitrário, ou seja, trata-se de uma heurística para o PA. A complexidade de tempo do PA(G, t) é $O(|E| \log |V|)$.

Experimentos foram conduzidos e apresentados em [2] para 17 redes sociais com utilização do algoritmo PA(G, t) e de outras duas heurísticas presentes na literatura chamadas MTS e DOM. Estas foram propostas respectivamente para os problemas TSS e *dominating set*. Vale ressaltar que os três algoritmos possuem a mesma complexidade de tempo. Os resultados mostraram que os tamanhos dos *seed sets* obtidos via PA(G, t), para os casos de testes executados, são, em média, menores que aqueles obtidos pelas demais estratégias.

1.3. Problemas relacionados

Existem três problemas de propagação em redes sociais muito próximos ao PA. O primeiro deles é o já mencionado TSS, para o qual já foram propostos na literatura diversas variações. Um modelo de programação linear inteiro foi proposto em [5] para uma versão do TSS onde o grafo G fornecido é um dígrafo. Entre outros resultados, os autores também apresentam limitantes superiores e inferiores para o TSS.

O segundo problema, lançado em [6], é chamado de *perfect evangelizing set* (PES) e consiste basicamente em uma generalização do PA em que cada vértice $v \in G$ também está associado a um outro *threshold* $t_A(v)$. Este *threshold* determina quantos vizinhos são necessários para que v se torne conhecedor. Ou seja, o PA é um caso especial do PES em que $t_A(v) = 1$, para todo $v \in G$.

O terceiro problema, também proposto em [6], é o *maximally evangelizing set* (MES), que pode ser entendido como a versão de maximização do PES. Isto é, no MES, o objetivo é obter a maior quantidade possível de conhecedores em G, sob a restrição de que $|S| \leq \beta$, para um dado inteiro β. Além de resultados teóricos e algoritmos parametrizados para ambos os problemas, o trabalho propõe algoritmos exatos para o MES, para casos em que G é um grafo completo ou uma árvore.

2. Metodologia

Este projeto objetivou a construção de heurísticas baseadas na meta-heurística GRASP para solucionar instâncias do PA. A principal justificativa da aplicação de uma heurística para este problema provém da sua complexidade, uma vez algoritmos exatos tendem a se tornar ineficientes para instâncias grandes de problemas NP-difíceis. Entretanto, também foi elaborado um modelo PLI para PA, afim de verificar a viabilidade de uma abordagem exata.

2.1. GRASP

O *Greedy Randomized Adaptive Search Procedure* (GRASP) é uma meta-heurística iterativa, na qual cada iteração consiste de duas fases: construção e busca local. A fase de construção cria uma solução através de uma estratégia que pode combinar características gulosas e randômicas. Se a solução construída não for viável, então é necessário aplicar um procedimento de reparo para alcançar viabilidade. Uma vez obtida uma solução viável, inicia-se a fase de busca local, onde sua vizinhança no espaço de soluções é investigada até que um mínimo local seja encontrado. A melhor solução global é mantida como resultado [7].

Na fase de construção, uma solução parcial é tomada, sendo inicialmente vazia. Além disso, uma lista de elementos chamada *candidate list* (CL) é mantida, contendo todos os elementos que podem ser inseridos em uma solução parcial. Os elementos da CL são associados a um custo calculado por alguma função de avaliação e podem ser inseridos na *restricted candidate list* (RCL) de acordo com algum critério. Os elementos da RCL devem conter os elementos cuja incorporação à solução parcial atual resulta nos melhores custos incrementais. Segundo um critério randômico, algum elemento da RCL deve ser adicionado a solução parcial. Esse procedimento é repetido até que um critério de parada pré-definido seja satisfeito [7].

A fase de busca local considera cada solução produzida na fase de construção e realiza buscas em sua vizinhança. Observe que é necessário definir, para o problema a ser tratado, quais são as soluções que representam uma vizinhança. Em geral, são definidas operações que podem ser realizadas sobre uma solução, modificando-a. Essa modificação resulta em uma solução diferente considerada vizinha. A movimentação durante a busca local segue, em geral, uma destas duas estratégias: *best improvement*, na qual seleciona-se sempre o melhor dentre todos os vizinhos ou *first improvement*, na qual escolhe-se o primeiro vizinho com valor melhor do que a solução atual. O critério de parada da busca local em geral é definido como a obtenção de um mínimo local [7].

2.2. Modelagem GRASP para o PA

Para construir uma heurística baseada em GRASP para um problema de otimização combinatória é necessário definir inicialmente o que é uma solução factível. Para o PA, uma solução qualquer (factível ou não) é simplesmente um *seed set* e uma solução só é factível se for também um *perfect seed set*, caso contrário, é infactível. Uma vez que o objetivo é encontrar um *perfect seed set* de tamanho mínimo, a fase de construção do GRASP para o PA pode ser finalizada quando a solução parcial S for factível.

2.2.1. Fase de construção

Neste trabalho, três diferentes estratégias para a fase de construção são consideradas. Em todas elas a CL é composta por qualquer vértice v, desde que v não pertença a S, ou seja, na CL encontram-se todos os vértices que não fazem parte da solução parcial. As estratégias diferem quanto à criação da RCL e quanto ao

critério de inserção de um vértice v na solução parcial. Nesta seção, definiremos uma estratégia de construção padrão e as estratégias alternativas chamadas *random plus greedy* e *sampled greedy*. Nas três estratégias um mesmo cálculo de custo de inserção de elementos na solução parcial é utilizado.

Durante a fase de construção, uma maneira intuitiva de calcular o custo da inserção de um vértice v à solução parcial seria calcular $|\text{Aware}_G(S \cup \{v\}) - \text{Aware}_G(S)|$. Isso significa que o custo $c(v)$ de v seria igual a quantidade "extra" de vértices que se tornariam conhecedores caso v fosse inserido no *seed set*. Nesse caso, um critério de escolha gulosa consideraria os vértices com maiores custos. Entretanto, adotar esse custo na etapa de construção torna o procedimento caro, uma vez que o processo de propagação teria que ser realizado para cada elemento da CL antes de cada uma das inserções para a obtenção de $|\text{Aware}_G(S \cup \{v\})|$. Observe que o processo de propagação tem complexidade da ordem de $O(|E| + |V|)$.

Uma forma mais barata de calcular o custo da inserção de um vértice v na solução parcial pode ser definida como calcular $|N_G(v) \setminus Aware_G(S)|$. Ou seja, o custo $c(v)$ é igual à quantidade de vizinhos de v que se tornarão conhecedores caso este vértice seja inserido em S. Note que neste caso, para calcular $c(v)$ basta percorrer a vizinhança de v e checar quantos vértices são desconhecedores. Ou seja, não é necessário realizar propagações para cada elemento da CL, apenas a verificação de suas respectivas vizinhanças. Se a quantidade de vizinhos desconhecedores de um vértice for atualizada a cada inserção, então o cálculo de $c(v)$ pode ser realizado em tempo constante. Neste trabalho, consideraremos este custo para as três estratégias para fase de construção;

Quanto à criação da RCL, uma maneira padrão de se fazê-la é utilizar-se um parâmetro $\alpha \in [0, 1]$ de modo que um vértice $v \in$ CL será inserido na RCL se, e somente se, $c(v) \in [c_{\max} - \alpha(c_{\max} - c_{\min}), c_{\max}]$, onde $c_{\min}$ é o menor custo dentre todos os elementos da CL e $c_{\max}$, o maior. Observe que o parâmetro α controla o quão guloso (menores valores de α) ou randomizado (maiores α's) é o processo de construção. Após a construção da RCL, um vértice v pertencente à RCL é escolhido randomicamente para ser inserido na solução parcial [7]. Chamaremos de estratégia de construção padrão do GRASP, uma vez que ela utiliza as formas padrões de criação da RCL e de incremento da solução parcial.

Como mencionado anteriormente, outras duas estratégias para a fase de construção foram consideradas. A primeira é chamada de *random plus greedy* e consiste em inserir randomicamente elementos na solução parcial durante os p primeiros passos da construção. Após estes passos, os elementos passam a ser inseridos de maneira puramente gulosa [7]. Observe que isso seria o equivalente a ajustarmos $\alpha = 1$ para os p passos iniciais e em seguida reajustarmos o valor de α para 0 após os p passos.

A segunda estratégia tem o nome de *sampled greedy* e combina a randomização e o critério guloso de uma maneira diferente. Neste tipo de construção, uma amostra randômica da CL é escolhida para a formação da RCL, isto é, a RCL é formada pro um subconjunto aleatório da CL. Em seguida, um elemento da CL é escolhido de maneira puramente gulosa para ser acrescentado à solução parcial. O tamanho da amostra é definido por $\min\{p, |CL|\}$, tal que p é um parâmetro previamente fixado [7].

Na proposta inicial deste trabalho foi proposto um algoritmo alternativo para a fase de construção baseado em heurísticas já existentes na literatura. Descrevemos esse algoritmo a seguir. Dada uma instância (G, t) do PA, o primeiro passo deste algoritmo é obter uma árvore geradora aleatória T de G. Em T, ajustam-se os *thresholds* de cada vértice para o mínimo entre o *threshold* original e o grau daquele vértice em T. No segundo passo, executa-se o algoritmo TREE-PA(T, t') (seção 1.2), obtendo uma solução ótima. Verifica-se então a factibilidade dessa solução em relação à instância original. Se for factível, então será a solução final da fase de construção. Caso contrário, inicia-se o terceiro passo.

No terceiro passo, considera-se um grafo $G' = T$ e adicionam-se arestas de G em G' até que o grau de cada vértice em G' seja igual ao máximo entre seu grau em G e seu *threshold* original. Essas adições devem ser feitas de maneira gulosa, ou seja, escolhendo primeiro arestas que contribuam para vértices que mais necessitam de aumento de grau. Finalmente, no quarto passo, executamos o algoritmo PA(G', t) (seção 1.2) e obtemos uma solução que é factível para a instância original. Observe que a etapa de geração de T e a etapa de adição de arestas em G' correspondem justamente às características randômica e gulosa da fase de construção do GRASP.

Este algoritmo alternativo não foi implementado e nem considerado para os experimentos conduzidos neste trabalho. Decidiu-se pela implementação dessa estratégia em trabalhos futuros, a ser realizada após uma análise mais aprofundada dos algoritmos TREE-PA e PA. Nesse sentido, priorizou-se neste trabalho estudar abordagens que melhorassem a eficiência da fase de construção além da exploração de técnicas alternativas estudadas na disciplina, como a *random plus greedy* e a *sampled greedy*.

2.2.2. Fase de busca local

Quanto à fase de busca local, o valor de um *perfect seed set* S pode ser definido como $|S|$. Ou seja, o valor da solução gerada pela fase de construção é igual à quantidade de vértices que pertencem ao *seed set*. Assim, definimos também duas operações básicas para obtenção da vizinhança de S: a remoção de um vértice em S, e a troca de um vértice em S por outro fora de S. Observe que uma operação só é válida se a solução resultante também for factível. Isso implica em um custo alto de se adotar a estratégia de *best improvement*, uma vez que a factibilidade de toda a vizinhança precisaria ser testada.

Observe também que para um conjunto de soluções vizinhas e factíveis, aquelas que são obtidas por remoção sempre melhoram o valor da solução, enquanto que aquelas obtidas via troca não alteram a cardinalidade do conjunto S. Isso sugere que remoções sejam feitas até que vizinhos factíveis possam ser obtidos via remoção. Nesse caso, trocas são realizadas até que alguma remoção possa ser realizada novamente. Para evitar ciclagem durante uma sequência de trocas, um parâmetro inteiro k de limite de trocas pode ser definido.

Neste trabalho, apenas a operação de remoção foi considerada na condução de experimentos, uma vez que, por meio de testes preliminares com as instâncias adotadas, notou-se um alto custo computacional com utilização apenas da operação

de remoção. Assim, foi estabelecida uma busca local apenas com a operação de remoção. Ou seja, vizinhos distintos da solução atual representam essa mesma solução com um vértice distinto removido. Através de política de *first improvement*, assim que um vizinho factível é encontrado, o movimento é realizado e este vizinho passa a ser a solução atual, de modo que a busca segue a partir dele. A busca deve parar quando não há mais vizinhança factível.

2.3. Processo de propagação na fase de construção

Considerando o que foi proposto na seção 2.2.1, após cada inserção de vértice no *seed set* durante a fase de construção, uma nova propagação deve ser realizada, a partir da rodada 0, para a obtenção do novo valor da solução parcial. Com o objetivo de melhorar a eficiência da fase de construção, conjecturou-se neste trabalho a ideia de que, após a inserção de um vértice na solução parcial, a propagação ocorra a partir da rodada em que a propagação anterior à inserção foi finalizada. Ou seja, nesse caso, o estado da propagação é mantido entre as inserções, e ao invés de simular uma propagação a partir da rodada inicial, dá-se continuidade à propagação que ocorreu antes da inserção a partir do novo nó inserido na solução parcial. A seguir formalizamos essa ideia.

Considere uma instância do PA composta por um grafo $G = (V, E)$ e a função *threshold* $t : V \to \mathbb{N}_0$. Seja S um *seed set* desta instância. Denotamos por S^x a expansão do conjunto S após a ocorrência de todas as rodadas de propagação. Ou seja, S^x contém, além dos disseminadores iniciais, todos os indivíduos que se tornaram disseminadores até o fim da propagação. Analogamente, temos que $(S \cup \{u\})^x$ é o conjunto expandido de $(S \cup \{u\})$. Observe que $S^x \subseteq (S \cup \{u\})^x$.

Conjectura 1. *Se $S \subseteq V$ e $u \in V$, então $(S \cup \{u\})^x = (S^x \cup \{u\})^x$.*
Ideia de prova. Mostrar que $(S \cup \{u\})^x \subseteq (S^x \cup \{u\})^x$, pois $S \subseteq S^x$, e mostrar que $(S^x \cup \{u\})^x \subseteq (S \cup \{u\})^x$.

Observe então que a conjectura mostra que a propagação a partir de um *seed set* $S \cup u$ resulta num estado idêntico à uma propagação do *seed set* S em que o vértice u é inserido em S em alguma rodada durante a propagação. Uma vez que seja provada, a conjectura acima permite que a ideia de melhoria mencionada anteriormente seja válida.

Considere agora um algoritmo de construção básico em que S é inicialmente vazio. A construção é conduzida por sucessivas inserções de vértices em S. A cada nova inserção, o processo de propagação deve ser realizado para verificar se S é um *perfect seed set* e, portanto, uma solução válida.

Suponha que d_1 seja o grau do primeiro vértice que foi inserido em S, d_2 o grau do segundo e assim sucessivamente. Suponha também que $n = |V|$. Assumindo um grafo conexo, o pior caso de construção, uma solução factível S é obtida quando $|S| = n - 1$, isto é, após $n - 1$ inserções. O custo do algoritmo de construção pode ser calculado por:

$$(n-1)d_1 + (n-2)d_2 + \cdots + d_{n-1} = \sum_{i=1}^{n-1}(n-i)d_i = n\sum_{i=1}^{n-1} d_i - \sum_{i=1}^{n-1} id_i$$

Observe que $n\sum_{i=1}^{n-1} d_i$ é da ordem de $O(n \cdot m)$, onde m é o número de arestas. Além disso $O(n \cdot m) > n^2$. Por outro lado, $\sum_{i=1}^{n-1} id_i$ é da ordem de $O(\triangle n)$ e, portanto, o algoritmo tem custo $O(n\dot{m} - \triangle n)$.

Podemos melhorar o algoritmo fazendo com que apenas uma propagação seja conduzida ao longo de todas as inserções. Ou seja, a cada nova inserção, a propagação deve continuar a partir do estado em que a propagação anterior foi finalizada.

Se S^x é o conjunto expandido de disseminadores resultante de uma propagação finalizada na rodada k e u é o próximo vértice a ser inserido na solução parcial, então a conjectura 1 nos diz que a expansão de $S^x \cup \{u\}$ resulta num conjunto igual a expansão de $S \cup \{u\}$. Ou sejam, o estado final da propagação de $S \cup \{u\}$ a partir da rodada 0 é igual ao estado final da propagação de $S^x \cup \{u\}$ a partir da rodada k.

O custo do pior caso do algoritmo melhorado pode ser calculado por $\sum_{i=1}^{n-1} d_i$ e é da ordem de $O(n + m)$. Assim, a adoção do algoritmo melhorado pode resultar em um desempenho superior.

Nas implementações e experimentos conduzidos neste trabalho, considerou-se a melhoria descrita para a fase de construção, assumindo-se a conjectura 1 como verdadeira. Entretanto, para evitar a produção de solução infactíveis na fase de construção, uma propagação a partir da rodada 0 é realizada após a fase de construção afim de certificar a factibilidade da solução. A prova do enunciado será elaborada futuramente como parte da pesquisa do aluno durante o curso de mestrado.

2.4. Programa linear inteiro para o PA

Nesta seção, descrevemos um modelo de programação linear inteira (PLI) elaborado para o PA. Considere uma instância do PA composta por um grafo $G = (V, E)$ e a função *threshold* $t : V \to \mathbb{N}_0$. No modelo proposto, há apenas um conjunto de variáveis binárias. Cada variável $s_{v,\tau}$ é associada ao vértice $v \in V$ e à rodada de propagação τ, tal que $0 \leq \tau \leq n = |V|$. Se $s_{v,\tau} = 1$, então o vértice v é um disseminador no instante τ, caso contrário $s_{v,\tau} = 0$. Temos então o modelo PLI:

minimize:

$$\sum_{v \in V} s_{v,0} \tag{2.1}$$

sujeito a:

$$s_{v,\tau} - s_{v,\tau-1} \geq 0 \qquad \forall\, v \in V \text{ e } \tau \in [1,n] \tag{2.2}$$

$$\sum_{u \in N_G(v)} (s_{u,\tau-1}) + t(v) \cdot s_{v,0} - t(v) \cdot s_{v,\tau} \geq 0 \qquad \forall\, v \in V \text{ e } \tau \in [1,n] \tag{2.3}$$

$$\sum_{u \in N_G(v)} (s_{u,\tau-1}) - (d(v) - t(v) + 1) \cdot s_{v,\tau} \leq t(v) - 1 \qquad \forall\, v \in V \tag{2.4}$$

$$s_{v,n} + \sum_{u \in N_G(v)} (s_{u,n-1}) \geq 1 \qquad \forall\, v \in V \text{ e } \tau \in [1,n] \tag{2.5}$$

$$s_{v,\tau} \in \{0,1\} \qquad \forall\, v \in V \text{ e } \tau \in [0,n] \tag{2.6}$$

A função objetivo (2.1) minimiza a quantidade de disseminadores na rodada 0, ou seja, o tamanho do *seed set*. A restrição (2.2) garante que um vértice v que tenha se tornado disseminador permaneça nesse estado até o fim da propagação. A restrição (2.3) garante que um vértice v não se torne disseminador caso a quantidade de vizinhos disseminadores seja inferior ao seu *threshold*, com exceção dos vértices que eram *seeds* desde a rodada 0, isto é, vértices que pertencem ao *seed set*.

De maneira complementar à (2.3), a restrição (2.4) força um vértice v a se tornar disseminador quando a quantidade de vizinhos disseminadores atinge seu *threshold*. Ademais, a restrição (2.5) garante que a solução só é válida se todos os vértices são conhecedores ao fim da propagação. Por fim, a restrição (2.6) garante que as variáveis $s_{v,t}$ sejam binárias. Note que $N_G(v)$ refere-se ao conjunto de vértices que são vizinhos de v; $d(v)$ denota o grau de v; e $t(v)$ denota o *threshold* de v.

A prova de corretude deste modelo será elaborada futuramente como parte da pesquisa do aluno durante o curso de mestrado. Entretanto, como evidenciado na seção 3, o modelo apresentado foi utilizado na condução de experimentos.

3. Experimentos

Nesta seção são descritos os experimentos computacionais realizados. As implementações e resultados dos experimentos estão disponíveis em um repositório no GitHub acessível por `https://github.com/fcpereira97/grasp_and_pli_for_perfect_awareness_problem.git`.

3.1. Instâncias

As instâncias utilizadas no experimento consistem em 17 redes sociais que foram adotadas nos experimentos conduzidos em [2] e estão disponíveis em [8, 9, 10]. A tabela 1 apresenta os principais dados que descrevem as instâncias, tais como o nome, o número de vértices, o número de arestas e densidade do grafo. O cálculo da densidade D de um grafo não direcionado pode ser descrito por $D = \frac{2 \cdot |E|}{|V| \cdot (|V| - 1)}$, onde V é o conjunto de vértices e E é o conjunto de arestas.

Tabela 1: Instâncias

Nome	**Nº de vértices**	**Nº de arestas**	**Densidade**
Amazon0302 [8]	262111	899792	$2,62 \cdot 10^{-5}$
BlogCatalog3 [10]	10312	333983	$6,28 \cdot 10^{-3}$
BuzzNet [10]	101168	2763066	$5,40 \cdot 10^{-4}$
Ca-AstroPh [8]	18772	198110	$1,12 \cdot 10^{-3}$
Ca-CondMath [8]	23133	93497	$3,49 \cdot 10^{-4}$
Ca-GrQc [8]	5242	14496	$1,06 \cdot 10^{-3}$
Ca-HepPh [8]	12008	118521	$1,64 \cdot 10^{-3}$
Ca-HepTh [8]	9877	25998	$5,33 \cdot 10^{-4}$
Cit-HepTh [8]	27770	352324	$9,14 \cdot 10^{-4}$
Delicious [8]	536408	1366663	$9,50 \cdot 10^{-6}$
Douban [10]	154908	327162	$2,73 \cdot 10^{-5}$
Facebook [8]	4039	88234	$1,08 \cdot 10^{-2}$
Jazz [9]	198	2742	$1,41 \cdot 10^{-1}$
Karate [9]	34	78	$1,39 \cdot 10^{-1}$
Last.fm [10]	1191812	4519349	$6,36 \cdot 10^{-6}$
Power grid [9]	4941	6594	$5,40 \cdot 10^{-4}$
Youtube2 [10]	1138499	2990443	$4,61 \cdot 10^{-6}$

Para todas as instâncias foram desconsideradas arestas múltiplas existentes no grafo original, uma vez que no modelo de propagação considerado neste trabalho, as arestas representam o relacionamento entre dois indivíduos representados por vértices. Portanto, o número de arestas apresentados na tabela 1 corresponde à quantidade de arestas já desconsideradas as multiplicidades.

Originalmente as instâncias não possuem *thresholds* associados aos vértices e portanto é necessário definir estes valores. Assim como nos experimentos conduzidos por [2], foram adotados neste trabalho os chamados *majority thresholds*. O *majority threshold* $t(v)$ de um vértice v é calculado através de seu grau $d(v)$, onde $t(v) = 0,5 \cdot d(v)$. Observe que no caso de uma divisão não exata, o teto da divisão é considerado.

3.2. Casos de teste

Para avaliar as heurísticas propostas e o modelo PLI apresentado, foram consideradas as quatro configurações de execução:

- P: heurística com estratégia de construção padrão e busca local *first improvement*;
- RG: heurística com estratégia de construção *random plus greedy* e busca local *first improvement*;
- SG: heurística com estratégia de construção *sampled greedy* e busca local *first improvement*;
- PLI: modelo PLI proposto para o PA.

Cada configuração foi executada para cada uma das instâncias, ou seja, cada caso de teste corresponde a uma configuração executada sobre uma instância.

Após testes preliminares sobre as instâncias menores, os parâmetros das estratégias de construção foram ajustados. Para a construção padrão (configuração P), o parâmetro α foi configurado em $0,15$. Já o parâmetro p da estratégia *random plus greedy* (configuração RG) foi ajustado para $0,01 \cdot |V|$, isto é, 1% do número de nós do grafo. Por fim, o parâmetro p da estratégia *sampled greedy* (configuração SG) foi configurado em $0,1 \cdot |V|$, isto é 10% do número de nós do grafo.

O modelo PLI foi experimentado através do resolvedor comercial Gurobi [11], com as configurações *default* da aplicação.

O critério de parada de todos os casos de teste foi baseado em tempo. Em testes preliminares, verificou-se que para as instâncias com maior entrada ($|V|+|E|$) e com menor densidade (mais esparsos) as heurísticas realizavam uma menor quantidade de iterações. Assim, ficaram estabelecidos os seguintes critérios de parada para os casos de teste:

- Instâncias com densidade menor que $1 \cdot 10^{-4}$: 60 minutos;
- Instâncias com densidade entre $1 \cdot 10^{-4}$ e $1 \cdot 10^{-1}$, inclusive: 30 minutos;
- Instâncias com densidade maior que $1 \cdot 10^{-1}$: 10 minutos.

Para as instâncias com densidade menor que $1 \cdot 10^{-4}$, alguns ajustes nas heurísticas foram necessários. Estes ajustes foram realizados de maneira igual para as três heurísticas, de a cordo com as suas respectivas estratégias de construção. Cada passo da fase de construção realizou a inserção de, no máximo, 100 vértices, ao invés de apenas 1 vértice. Para essas mesmas instâncias, na fase de busca local, o número de iterações da busca foi limitado para 5% do tamanho do *seedset* produzido na fase de construção. Esses ajustes foram necessários para que as heurísticas desenvolvidas fossem capazez de executar ao menos algumas iterações para estas instâncias, as quais tem o número de vértices da ordem de grandeza 10^5 e 10^6.

Todas as implementações, tanto das heurísticas quanto do modelo exato foram implementadas na linguagem C++.

As execuções foram realizadas em uma máquina com processador POWERPC POWER9 com 8 núcleos de processamento e 10 GB de memória RAM. Cada núcleo possui *clock* de $2,25$ Ghz. O sistema operacional utilizado foi o Ubuntu vesão 19.04. O acesso a esta máquina foi possibilitado pela Minicloud, a nuvem disponibilizada pela OpenPower para alunos da Unicamp. Mais informações sobre a Minicloud podem ser acessadas em openpower.ic.unicamp.br/minicloud.

3.3. Métricas

Para os casos de teste com uso de heurísticas, foram coletadas as seguintes informações:

- O valor da melhor solução obtida;
- O valor médio das soluções obtidas a cada iteração;
- O número total de iterações;
- O número da iteração em que a melhor solução obtida foi encontrada;

Além disso, para cada execução, foi gerado um *log* contendo, além dos dados acima, a descrição da melhor solução obtida e as iterações em que um novo *incumbent* (nova melhor solução) foi encontrado.

Para os casos de teste com uso de PLI, foram coletadas as seguintes informações:

- Melhor limitante primal;
- Melhor limitante dual;

Além disso, foram exportados os *logs* das execuções através da captura do console, obtendo as informações de console padrão do Gurobi. Foram exportados também os arquivos padrão do Gurobi para os modelo PLI de cada instância e suas respectivas melhores soluções primais.

4. Resultados e discussão

Nesta seção são descritos e discutidos os resultados do experimento.

4.1. Modelo PLI

Os resultados dos experimentos conduzidos sobre o modelo PLI produzido para o Gurobi podem ser observados na tabela 2. As colunas representam respectivamente, a instância testada, o melhor limitante primal obtido e o melhor limitante dual obtido. Para a instância Karate, o resolvedor foi capaz de encontrar a solução ótima. Já para a instância Jazz, não foi obtido nenhum limitante dual. Para as

demais instâncias, houve *overflow* de memória RAM, devido à grade quantidade de variáveis e desigualdades.

Tabela 2: Resultados do modelo PLI

Instância	**Melhor limitante primal**	**Melhor limitante dual**
Jazz	48	-
Karate	3	3

Sendo assim, verificou-se que a viabilidade do uso do modelo proposto está restrita à grafos da ordem de 100 vértices, considerando o conjunto de instâncias testado. Como será exposto a seguir, as três heurísticas desenvolvidas no trabalho alcançaram o valor ótimo da instância Karate e superaram o limitante primal obtido pelo modelo exato para a instância Jazz.

O resultado obtido com o modelo PLI proposto é esperado e satisfatório, tendo em vista que se trata de um modelo inicial sobre o qual ainda não foi realizado qualquer tipo de aprofundamento.

4.2. Heurísticas

Os resultados dos experimentos conduzidos sobre as heurísticas desenvolvidas podem ser observadas na tabela 3. Nas colunas temos representadas, respectivamente: a instância testada; a configuração de execução aplicada; o valor da melhor solução (MS) obtida pela configuração; a média dos valores das soluções obtidas em cada iteração do GRASP; a quantidade total de iterações realizadas; e o número da iteração (ordinal) em que a melhor solução foi obtida.

Analisaremos primeiro a quantidade de iterações realizadas pelas heurísticas em relação ao tamanho das instâncias. É fácil perceber que quanto maior a entrada da instância e quanto mais esparso é o grafo, menor foi a quantidade de iterações que as heurísticas foram capazes de realizar. Esse comportamento foi observado em testes preliminares e já eram esperados para os experimentos finais.

Observe que mesmo ajustando o critério de parada de acordo com as características das instâncias, para instâncias muito pequenas como Karate, Jazz e Facebook, as heurísticas foram capazes de realizar uma quantidade elevada de iterações. É possível observar também que em muitos casos a melhor solução foi encontrada nas primeiras iterações, como no caso das instâncias Karate e Facebook.

Note que se dispuséssemos do valor ótimo das soluções, um critério de parada mais adequado poderia ser estabelecido. Além disso, para instâncias menores sugerimos que para os próximos trabalhos sejam adicionados outros critérios de parada, como por exemplo uma quantidade limite de iterações sem melhoria da solução.

Tabela 3: Resultados das heurísticas

Instância	**Config.**	**MS**	**Média sol.**	**Nº iter.**	**Iter. MS**
Amazon0302	P	30039	30167	9	5
	RG	53767	54048.5	4	1
	SG	29763	29918.33	9	1
BlogCatalog3	P	206	210.43	413	238
	RG	207	211.37	266	6
	SG	205	210.32	411	165
BuzzNet	P	129	133.30	151	4
	RG	135	139	13	10
	SG	130	133.28	145	64
Ca-AstroPh	P	1726	1894.45	42	13
	RG	1698	1860.56	37	23
	SG	1707	1896.31	41	26
Ca-CondMath	P	2140	2178.21	46	38
	RG	2113	2148.48	41	6
	SG	2146	2171.97	43	42
Ca-GrQc	P	756	766.10	802	257
	RG	752	764.61	758	242
	SG	752	765.83	742	70
Ca-HepPh	P	1221	1243.02	132	23
	RG	1210	1239.66	122	100
	SG	1213	1243.75	125	15
Ca-HepTh	P	1120	1139.99	248	97
	RG	1113	1134.26	226	183
	SG	1117	1141.50	226	140
Cit-HepTh	P	2430	2528.96	30	19
	RG	2461	2523.73	26	18
	SG	2424	2514.44	27	1
Delicious	P	13380	13440.18	11	6
	RG	51187	51529.50	2	2
	SG	13214	13276.41	12	9
Douban	P	3998	4036.88	170	123
	RG	7872	8102.14	78	61
	SG	3984	4054.27	174	4
Facebook	P	10	10	82334	1
	RG	10	10	15227	1
	SG	10	10	50853	1
Jazz	P	13	14.06	394520	23917
	RG	13	14.11	367324	856
	SG	13	15.06	286802	25
Karate	P	3	3	5836098	1
	RG	3	3	5921564	1
	SG	3	3.41	4190706	1
Last.fm	P	5164	5373.50	8	6
	RG	-	-	0	-
	SG	5175	5317.12	8	8
Power grid	P	577	588.75	1069	695
	RG	578	592.95	1005	567
	SG	577	588.65	970	183
Youtube2	P	39715	39715	1	1
	RG	-	-	0	-
	SG	39585	39585	1	1

Para algumas instâncias maiores, como por exemplo Ca-CondMath, as heurísticas obtiveram a melhor solução encontrada nas últimas iterações realizadas. Para estes casos, nota-se uma necessidade de ajuste nos algoritmos para que mais iterações sejam realizadas.

Em relação às instâncias mais difíceis, como Amazon0302, Delicious, Douban, Last.fm e Youtube2, nota-se que o número de iterações foi bastante inferior às demais instâncias, mesmo considerando o tempo limite maior estabelecido para a execução e os ajustes nas heurísticas. Nos casos extremos, a configuração RG não foi capaz de completar uma única iteração para as instâncias Youtube2 e Last.fm.

Isso sugere que, para essas instâncias, a estratégia *random plus greedy* pode ter inserido uma quantidade excessiva de nós randômicos no *seed set*, tornando mais longo o procedimento de construção. Além disso, *seed sets* maiores tendem a ocasionar *overhead* na busca local, pois quantidade de vértices vizinhanças verificadas na busca local é definida em função do tamanho do *seed set* obtido pela construção.

Para as demais instâncias, consideramos que a quantidade de iterações foi satisfatória, uma vez que os valores das melhores soluções obtidas estão muito próximos da média.

Agora, partiremos para a análise do desempenho das heurísticas em termos dos valores das soluções obtidos. Com exceção de algumas das instâncias mais difíceis, as heurísticas alcançaram resultados próximos entre si tanto em relação ao melhor valor da solução como também em relação à média.

Para as instâncias Amazon0302, Delicious e Douban, a configuração RG obteve desempenho discrepante em relação às outras configurações. Isso evidencia que, para instâncias grandes, o parâmetro p da estratégia *random plus greedy* pode ser melhor ajustado.

Em quantidades absolutas, a configuração P alcançou o melhor resultado (dentre as três) para 6 diferentes instâncias. A configuração RG alcançou esse feito para 8 instâncias e a configuração SG obteve os melhores valores para 11 instâncias. Note que em muitos casos, as melhores soluções obtidas por pelo menos duas das heurísticas têm valores idênticos. Estes casos correspondem às execuções sobre as instâncias Power grid, Facebook, Karate, Jazz e Ca-GrQc.

A maior discrepância (absoluta) entre o desempenho das duas melhores heurísticas para uma única instância ocorreu sobre a instância Amazon0302. Neste caso, a diferença no valor da solução foi de 276, onde o valor obtido pela configuração SG foi $0,92\%$ menor que o da configuração P. Já a maior discrepância proporcional ocorreu sobre a instância Delicious, onde a configuração SG obteve uma solução com valor $1,24\%$ menor que a configuração P.

De um modo geral, a configuração SG obteve melhores resultados, embora a diferença entre os valores não sejam tão significativos. Já a configuração RG obteve o pior desempenho para instâncias grandes, mas um desempenho pouco superior à P para instâncias médias e pequenas.

4.3. Comparativo com resultados da literatura

Os experimentos conduzidos com o algoritmo PA em [2] foram realizados sobre o mesmo conjunto de instâncias considerados neste trabalho e com o mesmos *thresholds*. Entretanto, o relato dos experimentos carece de algumas informações importantes como os critérios de parada, a quantidade de execuções do algoritmo e as especificações da máquina utilizada.

Sobre os experimentos em [2] também não são relatados os tratamentos feitos em relação às arestas múltiplas presente nas instâncias originais. Na tabela 1 em [2], são apresentadas à quantidade de nós e arestas de cada instância. Para as instâncias CA-AstroPh, Ca-CondMat, CA-GrQc, CA-HepPh, CA-HepTh e Karate, os autores apresentam a quantidade de arestas sem multiplicidade, isto é, a mesma quantidade apresentada neste trabalho.

Já para instâncias Amazon0302, BuzzNet, Cit-HepTh, Delicious e Last.fm, a quantidade de arestas apontadas pelos autores corresponde à quantidade original com multiplicidade. As demais instâncias não contém arestas múltiplas. Assim, como existem instâncias para as quais os autores aparentemtene desconsideraram as arestas múltiplas, vamos assumir que o mesmo foi feito para as demais e que descrição de parte das instâncias em [2] estavam incorretas.

Outra questão detectada neste trabalho está relacionada às instâncias Last.fm e Delicious. Em ambos os casos, os arquivos que descrevem as características das instâncias (arquivos do tipo README) apontam um número de vértices inferior ao que realmente existem nas instâncias. Neste trabalho, foram considerados o números reais de vértices existentes nas instâncias. Em [2], os autores aparentemente detectaram essa questão para a instância Last.fm e apresentaram o valor correto de vértices na tabela 1. Entretanto, o mesmo não foi feito para a instância Delicious. Mais uma vez, não há qualquer relato sobre essas questões em [2].

A tabela 4 apresenta um comparativo dos valores das melhores soluções obtidas pelas heurísticas produzidas neste trabalho e as melhores soluções obtidas pelo algoritmo PA em [2].

Analisando a tabela 4, observamos que a heurística com estratégia *sampled greedy* superou o algoritmo PA em 12 instâncias, e obteve o valor da melhor solução igual ao PA para 2 instâncias. Apenas para as instâncias BlogCatalog3 e Youtube2 a configuração SG foi superada pelo algoritmo PA.

A maior discrepância de valor absoluto entre SG e PA ocorreu sobre a instância Amazon0304 em que o valor da melhor solução obtida pela configuração SG difere em 6990 da instância obtida pelo PA, ou seja, a solução obtida por SG é $19,02\%$ menor.

Já a maior discrepância proporcional entre SG e PA ocorreu sobre a instância Last.fm, onde a configuração SG alcançou um solução com valor $81,11\%$ menor que o algoritmo PA.

Podemos afirmar então que o algoritmo PA foi superado de maneira significativa pela heurística GRASP com estratégia de construção *sampled greedy* para a maioria das instâncias testadas.

Ademais, verificamos que a configuração P também superou o algoritmo PA em

12 instâncias e com diferenças significativas nos valores das soluções. O mesmo não pode se dizer da configuração RG, uma vez que ela é superada em 6 instâncias pelo algoritmo PA e com diferença significativa dos valores das soluções para as instâncias maiores.

Tabela 4: Comparativo com resultados da literatura

	Melhor solução			
Instância	**P**	**RG**	**SG**	**PA**
Amazon0302	30039	53767	29763	36753
BlogCatalog3	206	207	205	99
BuzzNet	129	135	130	141
Ca-AstroPh	1726	1698	1707	2174
Ca-CondMath	2140	2113	2146	2901
Ca-GrQc	756	752	752	897
Ca-HepPh	1221	1210	1213	1610
Ca-HepTh	1120	1113	1117	1531
Cit-HepTh	2430	2461	2424	2996
Delicious	13380	51187	13214	19568
Douban	3998	7872	3984	4832
Facebook	10	10	10	10
Jazz	13	13	13	15
Karate	3	3	3	3
Last.fm	5164	-	5175	27403
Power grid	577	578	577	1367
Youtube2	39715	-	39585	33046

Uma maneira de analisar os resultados apresentados na tabela 4 é através de um gráfico de *performance profiles*. Um gráfico *performance profiles* exibe a probabilidade de um dos algoritmos testados ter fornecido a melhor solução, dentre todos os outros algoritmos, para as instâncias testadas. Quanto mais próximo do eixo y a linha correspondente ao algoritmo estiver, maior é a probabilidade de que ele tenha atingido o valor da melhor solução encontrada. Observe então que quanto maior a proximidade horizontal do ponto no gráfico em relação ao eixo y, menor foi o desvio do valor da solução produzida pelo algoritmo em relação à melhor solução encontrada.

A figura 2 exibe o *performance profile* dos resultados das heurísticas produzidas e do algoritmo PA em relação às 17 instâncias. Observe que o eixo y representa a probabilidade do algoritmo ter encontrado a melhor solução. Já o eixo x representa o desvio em relação à melhor solução. Segundo a figura, os algoritmos com melhor performance geral correspondem às configurações SG e P, com leve vantagem para a configuração SG. Como o algoritmo PA se saiu melhor que a configuração SG em apenas duas instâncias, ele coincide com o SG apenas na parte inferior do gráfico.

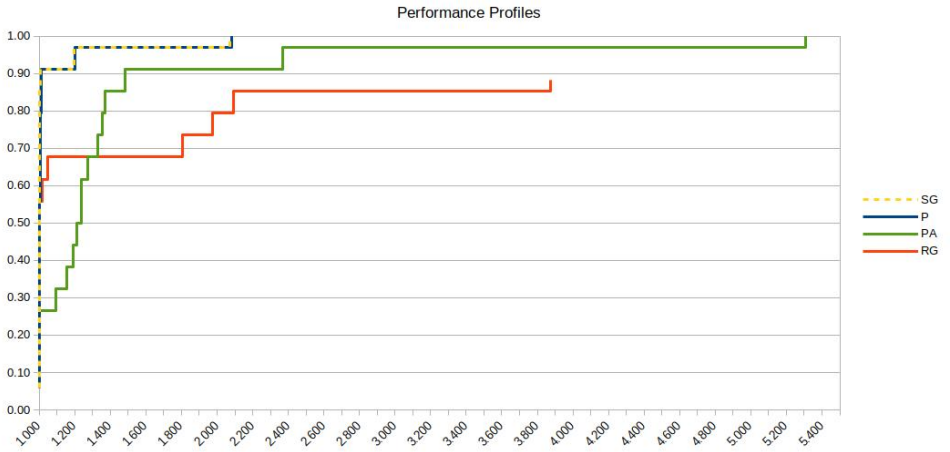

Figura 2: Gráfico *performance profiles* das configurações P, RG, SG e do algoritmo PA em relação às instâncias testadas.

5. Conclusões

As atividades que foram realizadas neste trabalho correspondem aos objetivos expostos na proposta inicial, salvas as mudanças de objetivo realizadas no decorrer do projeto e que foram justificadas neste documento.

Dentre as principais atividades, foi realizado um estudo sobre a meta-heurística GRASP e sobre o o estado da arte do problema PA. Além disso, foram elaboradas e implementadas 3 heurísticas baseadas em GRASP e um modelo PLI para o PA. Os algoritmos e modelo foram submetidos a experimentos computacionais com instâncias utilizadas na literatura. Os resultados desses experimentos apontaram um limite de usabilidade do modelo exato e evidenciaram o desempenho superior de duas heurísticas em relação ao algoritmo PA existente na literatura.

Tanto o modelo quanto as heurísticas foram descritas de maneira que possam ser reproduzidas em outros trabalhos, assim como os experimentos propriamente ditos. Cuidados especiais foram tomados em relação às instâncias, uma vez que o relato experimental existente na literatura em [2] não considerou aspectos importante das instâncias e do procedimento experimental.

Os resultados apresentados neste relatório são de grande relevância para o estudo do *perfect awareness* e servirão como ponto de partida para o tratamento desse problema na pesquisa que está sendo conduzida no curso mestrado. Os próximos passos envolvem o estudo do modelo exato visando melhorias, além do estudo da topologia das instâncias tratadas, afim de obter informações relevantes para a o aperfeiçoamento das heurísticas produzidas.

A continuidade do trabalho também envolverá o estudo dos algoritmos já existentes na literatura, além da implementação da fase de construção alternativa que

fora proposta inicialmente e que é baseada nestes algoritmos.

O aluno considera que a execução do trabalho foi bastante satisfatória, no sentido de que foi possível aplicar diretamente os conhecimentos adquiridos na disciplina. Reserva-se também aqui um espaço para agradecer aos professores Fábio Usberti, Pedro de Rezende e Cid de Souza pela contribuições proporcionada ao longo do desenvolvimento deste trabalho.

Referências

[1] W. Chen, L. V. Lakshmanan, and C. Castillo, *Information and Influence Propagation in Social Networks*, vol. 5, pp. 1–177. 2013.

[2] G. Cordasco, L. Gargano, and A. A. Rescigno, *Active influence spreading in social networks*, vol. 764, pp. 15 – 29. 2019. Selected papers of ICTCS 2016 (The Italian Conference on Theoretical Computer Science (ICTCS).

[3] R. Dawkins, *The Selfish Gene*. Oxford paperbacks, Oxford University Press, 1989.

[4] N. Chen, *On the Approximability of Influence in Social Networks*, vol. 23, pp. 1400–1415. 2009.

[5] E. Ackerman, O. Ben-Zwi, and G. Wolfovitz, *Combinatorial model and bounds for target set selection*, vol. 411, pp. 4017 – 4022. 2010.

[6] G. Cordasco, L. Gargano, A. A. Rescigno, and U. Vaccaro, *Evangelism in social networks: Algorithms and complexity*, vol. 71, pp. 346–357. 2018.

[7] M. G. Resende and C. C. Ribeiro, *Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications*, pp. 283–319. Boston, MA: Springer US, 2010.

[8] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June 2014. [Online; acessado em 29 de abril de 2019].

[9] M. Newman, "Network data." `http://www-personal.umich.edu/~mejn/netdata/`. [Online; acessado em 29 de abril de 2019].

[10] R. Zafarani and H. Liu, "Social computing data repository at ASU." `http://socialcomputing.asu.edu`, 2009. [Online; acessado em 29 de abril de 2019].

[11] G. Optimization, "Gurobi optimizer. versão 8.1," 2019.