Heurísticas e Metaheurísticas para Conjunto Independente

Breno Matos, Isadora Salles, Rennan Cordeiro

Novembro 2019

1 Introdução

Neste trabalho, exploramos o problema de encontrar Conjuntos Independentes em grafos. Este problema é conhecidamente NP-Completo, logo, a utilização de heurísticas e meta-heurísticas é crucial para encontrar conjuntos independentes em grafos com muitos vértices e arestas. Neste trabalho, propomos 6 heurísticas e 6 meta-heurísticas. Dentre cada grupo, metade são algoritmos da literatura e metade são algoritmos propostos neste trabalho.

1.1 Definição do Problema

Dado um grafo G formado por V vértices e E arestas, um conjunto U é independente quando $\forall u, v \in U : u \neq v \Rightarrow (u, v) \notin E$. Ou seja, não existe arestas entre os vértices de U. De modo equivalente, podemos descrever o conjunto independente U de forma que todas as arestas do grafo G se conectam no máximo com um vértice de U.

O conjunto independente máximo é um conjunto independente cuja cardinalidade é maior ou igual a de qualquer outro conjunto independente do mesmo grafo. Considerando isso, vale lembrar que são possíveis diferentes conjuntos independentes máximos em um mesmo grafo.

1.2 Motivação

Esse problema é semelhante a diversos outros problemas, como clique máxima, cobertura por vértice mínima e emparelhamento máximo, o que torna seu estudo mais relevante. Em termos práticos, um exemplo de situação que pode ser modelada como um problema de conjunto independente máximo seria: numa competição, cada jogo, em um mesmo horário, seria representado como um vértice, e as arestas são os jogos em que competidores estão em ambos os jogos. O objetivo é conseguir o maior número de jogos sem que um competidor esteja em ambos, assim seria possível alocá-los simultaneamente.

O problema de encontrar um conjunto independente máximo é um problema de otimização NP-difícil, o que significa não haver, atualmente, nenhum algoritmo que o resolve em tempo polinomial. Nesse caso, o estudo de heurísticas e meta-heurísticas torna-se necessário para encontrar possível soluções sub-ótimas, mas que, de alguma maneira, já possuem utilidade prática.

2 Prova de NP-Completude

Nesta seção, iremos provar que o problema de encontrar um conjunto independente de tamanho k em um grafo $G(V, E)$ é NP-Completo, utilizando o fato de que o problema do 3SAT é NP-completo. Como toda prova de NP-completude há duas partes:

1. Provar que o problema do conjunto independente está em NP:

   Isso significa que é possível apresentar um certificado que comprova quando uma dada instância tem conjunto independente. Neste caso, o certificado consiste dos k vértices pertencentes a V', sendo este o subconjunto de V em que nenhum par de vértices é adjacente. É possível verificar, em tempo polinomial, que não há aresta ligando os vértices u e v para cada par de vértices $u, v \in V'$

2. Provar que o problema do conjunto independente é NP-Difícil:

   Devemos mostrar que um problema NP-Completo (neste caso, 3-SAT) é redutível em tempo polinomial ao problema do conjunto independente. Seja F uma expressão booleana na forma 3-CNF[1]. Queremos encontrar uma função f , que pode ser computada em tempo polinomial, para mapear F para uma entrada para o problema do conjunto independente, um graph G e um inteiro k. Ou seja, $f(F) = (G, k)$, tal que F é satisfeita se e somente se G tem um conjunto independente de tamanho k. Isso implica que, se conseguimos resolver o conjunto independente para G e k em tempo polinomial, então seria possível resolver o 3SAT em tempo polinomial.

[1] Exemplo da forma 3-CNF: $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3})$

3 Trabalhos Relacionados

Nesta seção serão apresentadas algumas heurísticas presentes na literatura já propostas para o problema do conjunto independente. Em [9] é proposta uma heurística para o conjunto independente máximo que utiliza os resultados clássicos para o problema de otimização da função quadrática sobre uma esfera.

No artigo [13] a heurística da otimização de colônias de formigas é estendida para resolver o problema do máximo conjunto independente. Essa heurística é uma técnica para resolver problemas que podem ser reduzidos a encontrar bons caminhos num grafo. Como no problema que estamos tratando não existe o conceito de caminho ou ordem, a heurística é modificada de forma que deve ser projetado a construção da solução complementar.

Já no artigo [5] é apresentado um algoritmo eficiente chamado Vertex Support (VSA) para encontrar o conjunto independente máximo de um grafo. Esse algoritmo foi testado em um número grande de grafos randômicos com mais de 5000 vértices e os resultados mostram que o mesmo supera outros algoritmos existentes na literatura para resolver o problema em questão. O VSA é baseado no problema de encontrar o mínimo vertex cover de um grafo, adicionando à cobertura os vértices que possuem maior grau. No final, o conjunto independente retornado será composto por todos os vértices que não estão na cobertura.

Em [8] são apresentadas duas heurísticas para aproximação do conjunto independente máximo o algoritmo de Ramsey que calcula o maior conjunto independente juntamente a clique máxima do grafo, fazendo o uso de chamadas recursivas. O segundo algoritmo é o Clique Removal que faz chamadas ao algoritmo de Ramsey para encontrar cliques e removê-las até o grafo estar vazio, e então retorna o maior conjunto independente encontrado pelas chamadas ao Ramsey.

Em [1], [3] e [11] é proposta uma heurística gulosa baseada no grau mínimo de um vértice, adicionando ao conjunto os vértices com menor grau e excluindo do grafo os vértices pertencentes ao conjunto e todos os seus vizinhos. Em [3] também é proposta uma abordagem de busca local para resolver este problema. E nos artigos [4], [7] e [12] também são apresentadas heurísticas de busca local para o problema do conjunto independente.

E finalmente, nos artigos [10] e [6] temos heurísticas para encontrar o clique máximo em um grafo, e sabemos que um clique num grafo G é um conjunto independente no grafo complementar de G e vice versa, sendo assim solucionando o problema do clique máximo conseguimos encontrar o conjunto independente máximo.

4 Heurísticas Construtivas

Nessa seção serão apresentadas as três heurísticas da literatura que implementamos, e seus respectivos pseudocódigos, e além disso nossas contribuições, que são três novas heurísticas construtivas para o problema do conjunto independente máximo.

4.1 Heurísticas da Literatura

4.1.1 Heurística Gulosa

O algoritmo inclui no conjunto a cada passo o vértice de menor grau e exclui do grafo os todos os vizinhos desse vértice no grafo, sendo assim, como desejamos o maior conjunto independente, é fácil perceber que se incluirmos os vértices com menor grau no conjunto teremos uma boa aproximação do máximo conjunto independente. Essa heurística pode ser encontrada em [1], a seguir temos o pseudocódigo dessa heurística:

Algorithm 1: Greedy

Result: independent set
$S \leftarrow 0$;
while *G is not empty do* **do**
  Let v be the node with minimum degree in G;
  $S \leftarrow S \cup \{v\}$;
  Remove v and its neighbors from G;
end

4.1.2 Ramsey

A heurística Ramsey é implementada recursivamente com o seguinte raciocínio: para todo o vértice em v, em uma chamada recursiva é calculado o conjunto independente dentro da vizinhança de V sem incluir esse vértice e em outra chamada inclui-se o grafo excetuando o vértice v e seus vizinhos, mas esse vértice v é incluído na solução. Ao final, retorna-se o independente de tamanho maior. A heurística esta em [8]

Algorithm 2: Ramsey (G)

Result: independent set
if *G is empty* **then**
  return $\emptyset$ $\emptyset$;
end
Let v be any node in G;
$(C_1, I_1) \leftarrow Ramsey(N(V))$;
$(C_2, I_2) \leftarrow Ramsey(\overline{N(V)})$;
return *(larger of$(C_1 \cup v, C_2)$,larger of$(I_1, v \cup I_2)$)*

4.1.3 CliqueRemoval

O CliqueRemoval faz chamadas da função Ramsey a cada passo enquanto o grafo não estiver vazio,

em cada uma dessas iterações, ele remove a clique encontrada, o que diminui o tamanho da instância e remove vértices que não podem entrar na solução. Em seguida, retorna o maior dos conjuntos independentes encontrados junto com a sequência de cliques encontradas. Como esse conjunto é uma partição dos vértices do grafo em cliques, ele forma uma aproximação do problema de cobertura por cliques. A heurística se encontra em [8].

Algorithm 3: CliqueRemoval(G)

```
Result: independent set
i ← 1;
(C_i, I_i) ← Ramsey(G);
while G is not empty do
    G ← G − C_i;
    i ← i + i;
    (C_i, I_i) ← Ramsey(G);
end
return (larger of all(C_i),larger of all(I_i))
```

4.2 Contribuições

Nesta seção apresentamos as heurísticas criadas pelos integrantes do grupo.

4.2.1 Heurística utilizando Floyd–Warshall

Primeiramente, é executado o algoritmo de Floyd-Warshall para calcular a distância entre cada par de vértices. Inicializa-se a solução como conjunto vazio. Em seguida, percorre-se todos os vértices do grafo (iniciando sempre do vértice 0), computando da seguinte forma: para cada vértice $v \in V$, busca-se o vértice d mais distante de v. Em seguida, confere-se se d forma um conjunto independente com os demais vértices da solução. Caso forme, adiciona-se ao conjunto solução. Caso contrário, repete-se o procedimento para os demais vizinhos de v, ordenados por distância de forma decrescente.

Algorithm 4: Heurística Floyd-Warshall

```
Result: independent set
S ← 0;
v_i ← 0;
Dist = FloydWarshall(G)
while Every vertice of G isn't visited do
    S ← the most distant vertice d_j
    if d_j forms independent set with S then
        S ← S ∪ {d_j}
    end
    else
        d_j = d_{j+1}
    end
end
```

4.2.2 Heurística baseada no Vertex Cover

Uma das heurísticas é baseada na heurística gulosa 2-aproximativa para o problema de cobertura por vértices mínima. Nessa heurística, percorre-se por todas as arestas do grafo, a cada aresta adicionamos os dois vértices ligados a ela na cobertura e deletamos todas as arestas incidentes a esses dois vértices. Sabemos que dado uma cobertura por conjuntos, os vértices que não pertencem a ela formam um conjunto independente, porém como a heurística para o vertex cover é 2-aproximativa, pegar apenas os vértices que não estão no conjunto independente nos dará uma aproximação ruim para o conjunto independente máximo.

Então, o que foi feito foi computar a cobertura por vértices, retirar os vértices que não fazem parte da cobertura e adicionar ao conjunto independente, depois computar a cobertura para os vértices que não foram adicionados ao conjunto e repetir esse processo até que nenhum vértice possa ser adicionado ao conjunto independente. Depois rodamos a heurística gulosa para o conjunto independente nos vértices que ainda não foram adicionados ao conjunto. Assim, garantimos que formaremos um conjunto válido e temos uma aproximação digna para o conjunto independente máximo. O pseudocódigo para a heurística citada é dado a seguir:

Algorithm 5: Vertex Cover Heuristic

```
Result: vertex cover
C ← 0;
while Edges is not empty do
    Pick any {v, u} in Edges;
    C ← S ∪ {v, u};
    Remove all edges incident to either v or u;
end
```

Algorithm 6: Based on Vertex Cover

```
Result: independent set
S ← 0;
improvement = 1;
while improvement = 1 do
    C ← vertex cover heuristic(G);
    if C < G.nodes then
        S ← C̄;
        for each v in S remove v and its
          neighbors from G;
    else
        improvement = 0;
    end
end
S ← greedy heuristic(G);
```

4.2.3 Heurística do conjunto independente maximal

O algoritmo inclui no conjunto de soluções, a cada passo, algum vértice escolhido aleatoriamente e em seguida exclui todos seus vizinhos. Esse processo é repetido até que o grafo não contenha mais nenhum vértice. O conjunto independente encontrado é maximal, caso contrário, haveria algum vértice no grafo que poderia ser incluído e que não é vizinho de nenhum outro vértice encontrado, o que seria uma contradição. Em outras palavras, esse não é um subconjunto de nenhum outro conjunto independente. A seguir temos o pseudocódigo dessa heurística:

Algorithm 7: Maximal

```
Result: independent set
S ← 0;
while G is not empty do do
    Let v be any node in G;
    S ← S ∪ {v};
    Remove v and its neighbors from G;
end
```

5 Meta-heurísticas

Nessa seção serão apresentadas três meta-heurísticas da literatura, e seus respectivos pseudocódigos, e além disso nossas contribuições, que são três novas meta-heurísticas para o problema do conjunto independente máximo.

5.1 Meta-heurísticas da Literatura

5.1.1 Iterated Local Search com (1,2)-swap

Nessa heurística, a solução inicial é dada pela heurística gulosa, e a busca local é feita com (1,2)-swap, na qual escolhemos o vértice de maior grau pertencente ao nosso conjunto independente atual e tentamos trocar esse vértice por outros 2 vértices da sua vizinhança de forma que o novo conjunto formado deve ser um conjunto independente válido. O mecanismo ILS cria uma diversificação global, pois nesse caso iremos gerar uma pertubação na solução atual, isso ajuda a evitar ciclagem, que pode ocorrer quando o algoritmo fica preso a visitar apenas soluções que já foram visitadas.

A perturbação é dada por adicionar até 4 novos vértices aleatórios no nosso conjunto, e retirar os vértices que são vizinhos dos que foram adicionados. Ou seja, retiramos os vértices que possuem arestas que incidem que algum vértice adicionado. O critério de aceitação de uma solução será avaliar se ela é melhor ou igual a melhor solução até então. Esse processo se repete por um número máximo de iterações que varia de acordo com a instância ou até que exceda 120 segundos. Essa meta-heurística pode ser encontrada em [4]. O pseudocódigo para o algoritmo está a seguir:

Algorithm 8: Pseudo-code for the ILS with (j,k)-swap

```
Result: Independent Set
S = Greedy(G);
S ← two_improvement(G, S);
iter = 1;
duration = 0;
best_size = 0;
while iter ≤ maxIter and
 duration ≤ 120 seconds do
    S' = Perturb(S, G);
    S' = two_improvement(S', G);
    if best_size < size(S) then
        best_size = size(S);
        S = S'
    end
    iter = iter + 1;
end
return (best_size)
```

5.1.2 Hybrid Iterated Local Search Heuristic

Este algoritmo apresenta uma heurística baseada em ILS hibridizada com VND para resolver o problema do Conjunto Independente de Peso Máximo. Foram sugeridas duas novas estruturas de vizinhança e também foi desenvolvido um mecanismo de perturbação reativa. Nas instâncias de testes utilizadas neste trabalho, os vértices não são ponderados, portanto, inicializamos todos com peso 1. A ideia é que, com pesos iguais, o algoritmo irá encontrar o maior conjunto independente possível. Proposto em [14], o algoritmo funciona da seguinte maneira:

Algorithm 9: Pseudo-code for the ILS for the MWIS

```
Result: Independent Set
S_0 = Initialize(G);
S = LocalSearch(S_0, G);
S* = S;
local_best_w = Weight(S);
iter = 1;
i = 1;
while iter ≤ maxIter do
    S' = Perturb(c_1, S, G);
    S' = LocalSearch(S', G);
    (Si, S*, i, local_best_w) =
      Accept(S, S*, S', i, local_best_w, G);
    iter = iter + 1;
end
return (S*)
```

5.1.3 Multi-neighborhood Tabu Search

Proposto em [15], este algoritmo apresenta uma heurística de *tabu search* para o problema de clique de peso máximo com base em uma vizinhança combinada, induzida por três tipos de movimentos. No nosso caso, os vértices não possuem peso, por isso inicializamos todos com peso 1. O algoritmo explora todos esses movimentos a cada iteração e seleciona a melhor solução admissível (que não é tabu ou que melhora globalmente) que produz o maior ganho de peso, ou seja, o maior conjunto de vértices. O mecanismo tabu cria uma diversificação local eficaz e uma estratégia de múltiplas etapas é empregada para criar uma diversificação global. O pseudocódigo para o algoritmo pode ser visualizado abaixo.

Algorithm 10: Pseudo-code for the multi-neighborhood tabu search for the maximum clique problem

```
Result: Independent Set
Iter = 0;
C* = ∅;
while iter ≤ maxIter do
    C = Initilize();
    Initiatetabu_list;
    NI = 0;
    C_local_best = C;
    while NI < L do
        Construct neighborhoods N_1, N_2, N_3
         from C;
        C = C';
        NI = NI + 1;
        Iter = Iter + 1;
        Update tabu_list;
        if W(C) > W(C_local_best) then
            NI = 0;
            C_local_best = C
        end
        NI = NI + 1;
        Iter = Iter + 1;
    end
end
if W(C_local_best) > W(C*) then
    C* = C_local_best
end
return (Clique C*)
```

5.2 Contribuições

Nesta seção apresentamos as meta-heurísticas criadas pelos integrantes do grupo.

5.2.1 GRASP

Uma das meta-heurísticas criadas foi baseada em GRASP com a vizinhaça 2-improvement já citada anteriormente. Nesse mecanismo ao chegar em um ótimo local começamos novamente a partir de uma heurística gulosa. As soluções diferentes são dadas por uma aleatoriedade na solução. A heurística aleatorizada criada foi baseada na heurística gulosa citada anteriormente, mas ao invés de incluir o vértice com menor grau a cada etapa, incluímos o primeiro ou segundo vértice de menor grau de forma aleatória. Após encontrar a melhor solução localmente, utilizando vizinhança 2-improvement, geramos uma nova solução inicial aleatorizada e repetimos esse processo durante 10 segundos. A solução final será o maior conjunto independente encontrado.

Algorithm 11: GRASP Metaheuristic

```
Result: Independent Set
duration = 0;
while duration ≤ 10 seconds do
    S = GreedyAleatorized(G);
    S ← two_improvement(G, S);
end
return largest independent set found ;
```

5.2.2 Tabu Search

Nesta meta-heurística, implementamos um *tabu search* que utiliza as vizinhanças *(1,k)-swap* e *(2,k)-swap*, e lista tabu que armazena 57 soluções. O pseudocódigo para o algoritmo pode ser visualizado abaixo.

Algorithm 12: Tabu Search Metaheuristic

```
Result: Independent Set
S = Greedy(G);
Iter = 0;
best_size = 0;
tabu_list = ∅;
for Iter < tabu_list_size do
    S* = S;
    S ← n_improvement(G, S);
    if best_size < size(S) then
        best_size = size(S)
    end
    S ← 2k_swap(G, S);
    if best_size < size(S) then
        best_size = size(S)
    end
    if S is in tabu_list then
        break
    end
    if tabu_list is full then
        remove_first(tabu_list)
    end
    tabu_list ← S;
    Iter + +;
end
return largest independent set of tabu_list ;
```

5.2.3 VND

Nesta solução, implementamos uma metaheurística do tipo VND que utiliza as vizinhanças *two improvement* e *three improvement*, iniciando de uma solução obtida utilizando a heurística gulosa. O pseudocódigo para o algoritmo pode ser visualizado abaixo.

Algorithm 13: VND Metaheuristic

Result: Independent Set
$S = Greedy(G)$;
$two\_improvement(G, S)$;
$three\_improvement(G, S)$;
return $size(S)$

6 Experimentação

As heurísticas implementadas foram avaliadas usando as instâncias do DIMACS [2], que foram escolhidas porque na maioria dos artigos citados na sessão de Trabalhos Relacionados são usadas essas instâncias para avaliar heurísticas de problemas de conjunto independente máximo ou clique máximo. Porém, esse *dataset* foi feito para problemas de clique máximo. Para poder comparar com a solução ótima encontrada temos que computar as heurísticas para conjunto independente no grafo complementar ao grafo formado pelas instâncias. Isso é possível pois se existe um clique máximo de tamanho k em G, existe um conjunto independente de tamanho k em G complementar.

6.1 Ambiente computacional

A implementação das heurísticas foi feita utilizando-se a linguagem C++11 e o tempo de execução de cada heurística foi medido com a função clock() da biblioteca ctime do C++. Os experimentos foram executados em um computador de 16Gb de RAM, intel core i7 com 8 núcleos num sistema operacional Ubuntu 16.04LTS.

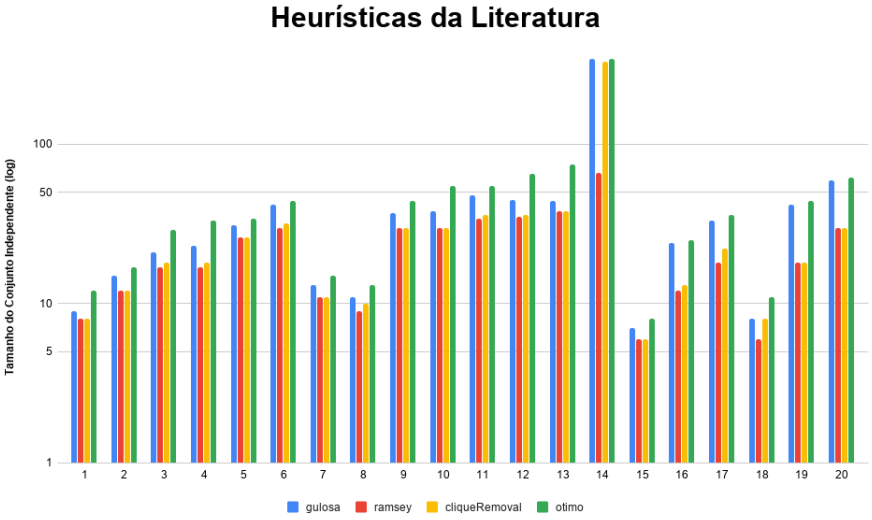
6.2 Resultados

Serão comparados os resultados obtidos pelas heurísticas propostas com os das heurísticas da literatura. Serão apresentadas as tabelas contendo os resultados dos experimentos e os respectivos gráficos. Inicialmente, vamos avaliar a qualidade da heurística gulosa e dos algoritmos de Ramsey e Clique Removal.

		Gulosa			**Ramsey**			**CliqueRemoval**		
Instância	Ótimo	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)
brock200_2	12	9	2,26E-03	25,0	8	2,08E-01	33,3	8	2,12E-01	33,3
brock200_4	17	15	3,13E-03	11,8	12	2,04E-01	29,4	12	2,04E-01	29,4
brock400_2	29	21	1,88E-02	27,6	17	1,43E+00	41,4	18	1,46E+00	37,9
brock400_4	33	23	1,81E-02	30,3	17	1,42E+00	48,5	18	1,32E+00	45,5
C125.9	34	31	4,07E-03	8,8	26	5,28E-02	23,5	26	5,70E-02	23,5
C250.9	44	42	1,48E-02	4,5	30	3,86E-01	31,8	32	3,96E-01	27,3
DSJC1000_5	15	13	3,78E-02	13,3	11	1,95E+01	26,7	11	1,50E+01	26,7
DSJC500_5	13	11	1,20E-02	15,4	9	2,72E+00	30,8	10	1,92E+00	23,1
gen200_p0.9_44	44	37	9,51E-03	15,9	30	2,06E-01	31,8	30	1,44E-01	31,8
gen200_p0.9_55	55	38	9,53E-03	30,9	30	1,85E-01	45,5	30	1,45E-01	45,5
gen400_p0.9_55	55	48	3,48E-02	12,7	34	1,44E+00	38,2	36	1,04E+00	34,5
gen400_p0.9_65	65	45	3,31E-02	30,8	35	1,42E+00	46,2	36	1,05E+00	44,6
gen400_p0.9_75	75	44	3,38E-02	41,3	38	1,02E+00	49,3	38	1,07E+00	49,3
MANN_a45	345	342	1,71E+00	0,9	66	1,66E+01	80,9	331	1,61E+01	4,1
p_hat300-1	8	7	3,49E-03	12,5	6	5,73E-01	25,0	6	4,85E-01	25,0
p_hat300-2	25	24	1,05E-02	4,0	12	5,64E-01	52,0	13	6,87E-01	48,0
p_hat300-3	36	33	1,46E-02	8,3	18	5,63E-01	50,0	22	6,50E-01	38,9

É possível perceber que a heurística gulosa em alguns casos, principalmente quando o número de vértices é pequeno, pode ser uma boa aproximação para o conjunto independente mesmo tratando-se de uma heurística simples, em alguns casos, ela retorna o resultado ótimo, como pode ser observado nas instâncias *keller4* e *hamming8-4*. Usando essa heurística para gerar a solução inicial de uma metaheurística, como GRASP, podemos ter uma melhoria significativa.

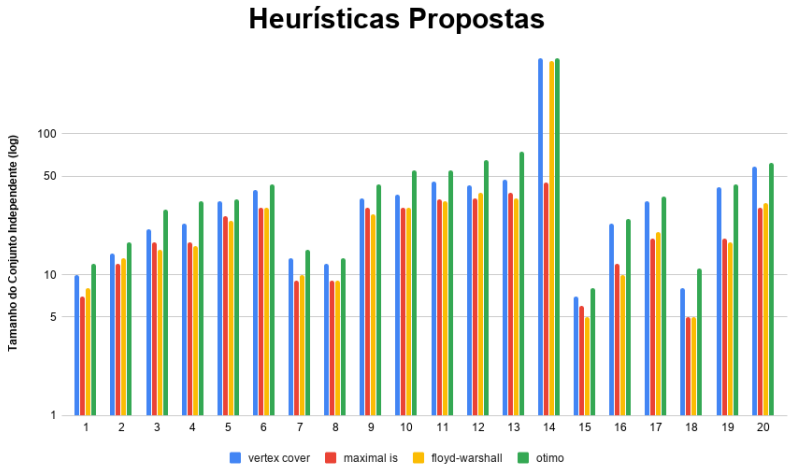
O gráfico a seguir permite visualizar o comportamento de cada algoritmos apresentado na tabela nessas instâncias



Na tabela a seguir temos os resultados encontrados apenas para as heurísticas propostas por nós, e em alguns casos conseguimos um resultado melhor do que a heurística gulosa, esses casos foram marcados de negrito na tabela.

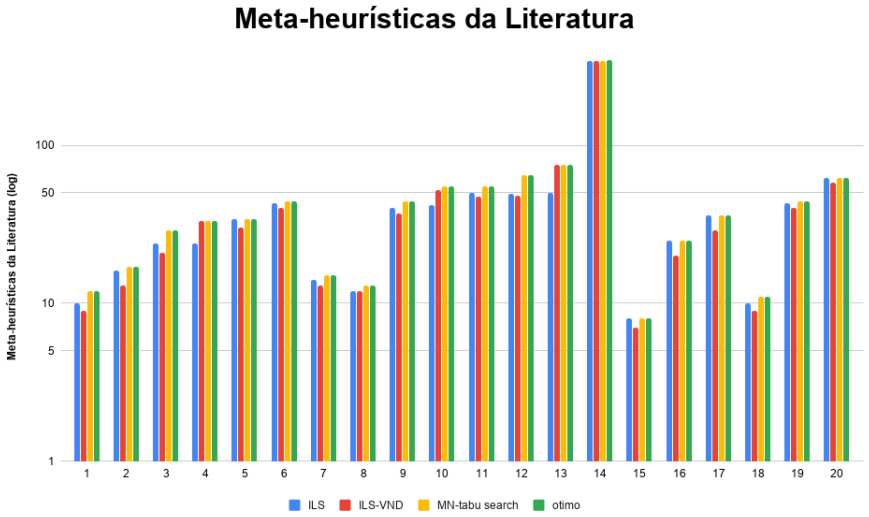
		Vertex Cover			**Maximal**			**Floyd-Warshall**		
Instância	Ótimo	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)
brock200_2	12	**10**	4,13E-02	16,7	7	5,50E+01	41,7	8	5,59E+03	33,3
brock200_4	17	14	3,36E-02	17,6	12	8,27E+02	29,4	13	5,18E+16	23,5
brock400_2	29	21	2,09E-01	27,6	17	4,75E+03	41,4	15	3,88E+05	48,3
brock400_4	33	23	6,41E-02	30,3	17	4,21E+03	48,5	16	3,92E+16	51,5
C125.9	34	**33**	8,36E-03	2,9	26	7,62E+02	23,5	24	1,21E+16	29,4
C250.9	44	40	3,78E-02	9,1	30	3,28E+03	31,8	30	9,53E+04	31,8
DSJC1000_5	15	13	7,51E-01	13,3	9	1,47E+04	40	10	9,34E+05	33,3
DSJC500_5	13	**12**	9,92E-01	7,7	9	3,68E+02	45,5	9	8,98E+05	30,8
gen200_p0.9_44	44	35	1,54E-02	20,5	30	1,96E+02	31,8	27	4,76E+04	38,6
gen200_p0.9_55	55	37	1,50E-02	32,7	30	1,77E+03	45,5	30	4,61E+04	45,5
gen400_p0.9_55	55	46	1,36E-01	16,4	34	8,10E+03	38,2	33	3,77E+05	40,0
gen400_p0.9_65	65	43	1,31E-01	33,8	35	7,95E+03	46,2	38	3,78E+16	41,5
gen400_p0.9_75	75	47	1,51E-01	37,3	38	8,62E+03	49,3	35	3,86E+05	53,3
MANN_a45	345	342	3,68E+00	0,9	45	7,26E+04	87,0	331	8,31E+15	4,1
p_hat300-1	8	7	1,78E-02	12,5	6	9,95E+02	25,0	5	1,91E+05	37,5
p_hat300-2	25	23	1,36E-01	8,0	12	1,81E+03	52,0	10	1,81E+05	60,0
p_hat300-3	36	33	4,54E-02	8,3	18	2,89E+03	50,0	20	1,69E+05	44,4

A heurística baseada no vertex cover foi a que obteve melhores resultados dentre as propostas, aproximando-se bastante da solução dada pela heurística gulosa ou até mesmo ultrapassando-a. A heurística maximal é uma heurística simples que sempre irá retornar um conjunto independente maximal, porém dado que a escolha dos vértices é a aleatória não levamos em conta algumas características que poderiam ampliar o tamanho do conjunto. Por exemplo, em alguns casos pode ocorrer de adicionarmos ao conjunto o vértice com maior grau, isso faz com que menos vértices possam ser adicionados ao conjunto.



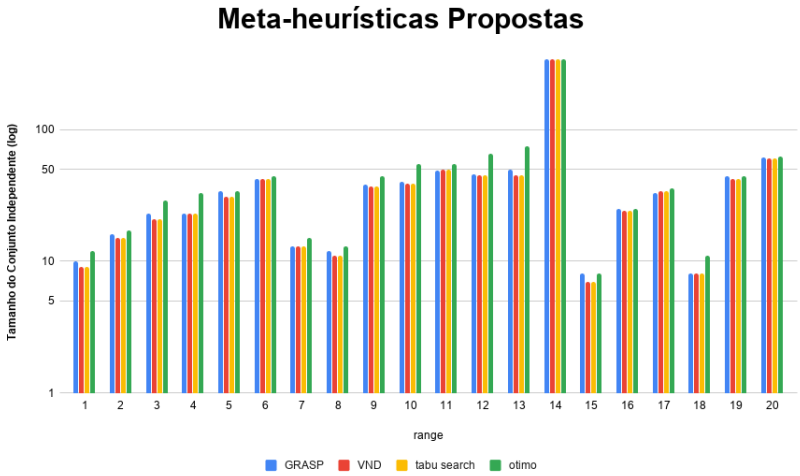
		ILS			ILS-VND			Multi-neighborhood tabu search		
Instância	Ótimo	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)
brock200_2	12	10	1,09E+00	16,7	9	1,81E+01	25,0	12	4,34E+00	0,0
brock200_4	17	16	1,86E+00	5,9	13	1,65E+01	23,5	17	6,74E+00	0,0
brock400_2	29	24	1,26E+01	17,2	21	2,91E+01	27,6	29	1,83E+02	0,0
brock400_4	33	24	1,28E+01	27,3	33	2,29E+01	0,0	33	2,36E+01	0,0
C125.9	34	34	2,23E+00	0,0	30	1,29E+01	11,8	34	1,16E-02	0,0
C250.9	44	43	1,04E+01	2,3	40	2,46E+01	9,1	44	2,82E-01	0,0
DSJC1000_5	15	14	3,61E+01	6,7	13	1,54E+02	13,3	15	3,30E+01	0,0
DSJC500_5	13	12	7,71E+00	7,7	12	8,03E+01	7,7	13	5,98E-01	0,0
gen200_p0.9_44	44	40	6,51E+00	9,1	37	1,67E+01	15,9	44	1,76E-01	0,0
gen200_p0.9_55	55	42	7,52E+00	23,6	52	1,98E+01	5,5	55	4,27E-02	0,0
gen400_p0.9_55	55	50	3,14E+01	9,1	47	3,52E+01	14,5	55	2,57E+02	0,0
gen400_p0.9_65	65	49	3,15E+01	24,6	48	3,51E+01	26,2	65	8,92E+00	0,0
gen400_p0.9_75	75	50	3,02E+01	33,3	75	4,14E+01	0,0	75	6,05E+00	0,0
MANN_a45	345	342	1,30E+02	0,9	343	1,27E+02	0,6	341	5,76E+03	1,2
p_hat300-1	8	8	1,56E+00	0,0	7	9,00E+01	12,5	8	3,70E-02	0,0
p_hat300-2	25	25	7,91E+00	0,0	20	9,13E+01	20,0	25	2,13E-02	0,0
p_hat300-3	36	36	1,17E+01	0,0	29	5,03E+01	19,4	36	1,27E-01	0,0

Como pode ser observado na tabela acima, a metaheurística *Multi-neighborhood tabu search* teve um desempenho ótimo em todas as instâncias, exceto em *MANNa45*, em compensação foi a que demorou mais tempo em quase todas elas. No gráfico a seguir podemos visualizar o comportamento dos algoritmos apresentados para essas instâncias.



		GRASP			VND			Tabu Search		
Instância	Ótimo	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)	Solução	Tempo(s)	gap(%)
brock200_2	12	10	1,02E+01	16,7	9	3,95E-01	25,0	9	1,14E+01	25,0
brock200_4	17	16	1,00E+01	5,9	15	7,09E-01	11,8	15	3,01E+01	11,8
brock400_2	29	23	1,00E+01	20,7	21	3,36E-01	27,6	21	2,07E+02	27,6
brock400_4	33	23	1,00E+01	30,3	23	4,05E-01	30,3	23	2,43E+02	30,3
C125.9	34	34	1,02E+01	0,0	31	6,65E-01	8,8	31	7,22E+01	8,8
C250.9	44	42	1,00E+01	4,5	42	3,36E-01	4,5	42	4,84E+02	4,5
DSJC1000_5	15	13	1,00E+01	13,3	13	1,34E-01	13,3	13	8,02E+02	13,3
DSJC500_5	13	12	1,00E+01	7,7	11	2,83E-01	15,4	11	1,46E+02	15,4
gen200_p0.9_44	44	38	1,00E+01	13,6	37	1,82E-01	15,9	37	2,68E+02	15,9
gen200_p0.9_55	55	40	1,00E+01	27,3	39	2,43E-01	29,1	39	2,75E+02	29,1
gen400_p0.9_55	55	49	1,01E+01	10,9	50	1,12E-01	9,1	50	2,18E+03	9,1
gen400_p0.9_65	65	46	1,00E+01	29,2	45	8,43E-01	30,8	45	1,58E+03	30,8
gen400_p0.9_75	75	50	1,01E+01	33,3	45	1,08E-01	40,0	45	1,08E-01	40,0
MANN_a45	345	342	1,05E+01	0,9	342	6,04E+00	0,9	342	8,31E+15	0,9
p_hat300-1	8	8	1,00E+01	0,0	7	8,86E-01	12,5	7	2,90E+01	12,5
p_hat300-2	25	25	1,00E+01	0,0	24	3,24E-01	4,0	24	3,48E+02	4,0
p_hat300-3	36	33	1,00E+01	8,3	34	5,96E-01	5,6	34	6,70E+02	5,6

A tabela acima revela um desempenho semelhante do *tabu search* e do *VND*, porém o primeiro demorou tempo muito superior ao segundo, o que é uma desvantagem desse algoritmo desenvolvido. O *GRASP* por sua vez mostrou um bom equilíbrio entre tempo e desempenho, com resultados melhores que os outros algoritmos na maioria das instâncias, e portanto, um gap de otimalidade inferior.



7 Conclusão

O uso de heurísticas é importante na solução de problemas NP-Completos. O trabalho apresentou diferentes soluções heurísticas e meta-heurísticas para um problema específico, o de conjunto independente máximo. Para compará-las, foram realizados diversos experimentos que revelaram o desempenho de cada uma delas em diferentes instâncias conhecidas de grafos.

Referências

[1] Approximation Algorithms. `https://courses.engr.illinois.edu/cs598csc/sp2011/Lectures/lecture_7.pdf`. (Acesso em 10/10/2019.).

[2] Dimacs Benchmark Set. `http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark#detC250.9`. (Acesso em 06/11/2019.).

[3] Heuristics for Maximum Clique and Independent Set. `https://www.dsi.unive.it/~pelillo/papers/EoO-heuristics.pdf`. (Acesso em 09/10/2019.).

[4] Andrade, D. V., Resende, M. G., and Werneck, R. F. Fast local search for the maximum independent set problem.

[5] Balaji, S., Swaminathan, V., and Kannan, K. A simple algorithm to optimize maximum independent set. *Advanced Modeling and Optimization 12*, 1 (2010).

[6] Balas, E., and Yu, C. S. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing 15* (1986), 1054–1068.

[7] Battiti, R.and Protasi, M. Reactive local search for the maximum clique problem. *Algorithmica 29* (2001), 610–637.

[8] Boppana, R. B., and Halldórsson, M. M. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics 32* (2006).

[9] Busygin, S., Butenko, S., and Pardalos, P. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. *Journal of Combinatorial Optimization 6*, 3 (2002), 287–297.

[10] Grosso, A., Locatelli, M., and Croce, F. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics 10* (2004), 100–106.

[11] Halldórsson, M. M., and Radhakrishnan, J. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica 18* (1997), 145–163.

[12] Katayama, K., Hamamoto, A., and Narihisa, H. An effective local search for the maximum clique problem. *Information Processing Letters 95* (2005), 503–511.

[13] Li, Y., and Xul, Z. An ant colony optimization heuristic for solving maximum independent set problems. *ICCIMA* (2003).

[14] Nogueira, B., Pinheiro, R. G., and Subramanian, A. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters 12*, 3 (2018), 567–583.

[15] Wu, Q., Hao, J.-K., and Glover, F. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research 196*, 1 (2012), 611–634.