Heurísticas para Conjunto Independente Máximo

Bruno Monteiro, Bruno Nogueira

Novembro de 2019

1 Introdução

O problema do conjunto independente máximo consiste em achar um subconjunto dos vértices de um grafo de tamanho máximo tal que não exista aresta entre nenhum par de vértices. O problema é NP-Completo e por isso foram criadas diversas heurísticas para resolvê-lo.

O problema é equivalente ao de achar uma clique máxima no complemento. Por isso, ao procurar heurísticas para qualquer um dos problemas, podemos considerar resolver o outro. Além disso, existe uma relação com o problema de cobertura por vértices, isto é, achar o menor subconjunto S de vértices do grafo tal que todas aresta seja incidente a pelo menos um elemento de S. A solução desse problema é denominada $\beta(G)$. Note que $\alpha(G) + \beta(G) = n$. De novo, podemos resolver um desses problemas para resolver o outro.

2 Trabalhos relacionados

O problema do maior conjunto independente é antigo, foi um dos 21 problemas que Karp[1] mostrou ser NP-Completo em 1972. Depois disso, muitas pesquisas foram feitas para resolver o problema de maneira exata e exponencial, mas, o que nos interessa são as heurísticas polinomiais que lidam com o problema.

Ao realizar esse trabalho, optamos por pesquisar heurísticas específicas para o problema do conjunto independente máximo e heurísticas que resolvem o problema da clique máxima, já que são equivalentes.

Uma das referências mais citadas na área, Bomze, I. et. al.,[3] mostra diversas heurísticas para o problema em seu trabalho. Além disso, usando metaheurísticas e outras técnicas como algoritmos genéticos, diversas soluções são apresentadas.

Além disso, temos o trabalho de Johnson[4] que mostra construções gulosas sequenciais para resolver o problema, assim como Tomita.[5] Os dois artigos, ainda relevantes, serão os mais usados na primeira parte desse trabalho, visto que mostram heurísticas construtivas.

3 Formulação matemática

Dado um grafo G com conjunto de arestas E e conjunto de vértices V, o problema do conjunto independente máximo consiste em achar o maior conjunto S, tal que para todo $u, v \in S$, $uv \notin E$. O tamanho desse conjunto é conhecido como número de independência e denotado como $\alpha(G)$.

4 Prova de NP-Completude

Como um conjunto independente pode trivialmente verificado em tempo polinomial, o problema pertence a NP. Faremos uma redução do problema SATISFABILIDADE BOOLEANA (SAT). Uma instância de SAT consiste em uma expressão lógica E com n cláusulas, cada uma com a_i literais.

Construímos então o grafo $G = (V, E)$ da seguinte forma: criamos um vértice para cada literal de cada cláusula, de forma que $|V| = \sum_{i=1}^{n} a_i$. Temos assim que o tamanho do grafo é limitado polinomialmente pelo tamanho da entrada da instância SAT.

Fazemos com que os vértices correspondentes a literais que pertencem a mesma cláusula sejam adjacentes, ou seja, cada cláusula corresponderá a uma clique em G. Além disso, para cada par de vértices u em v cujos

literais correspondentes não pertences a mesma cláusula, adicionamos uma aresta entre u e v se e somente se seus literais correspondem à mesma variável, uma negada e a outra não negada.

Por exemplo, para seguinte instância SAT:

$$E = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_3} \vee x_4)$$

O grafo correspondente é:

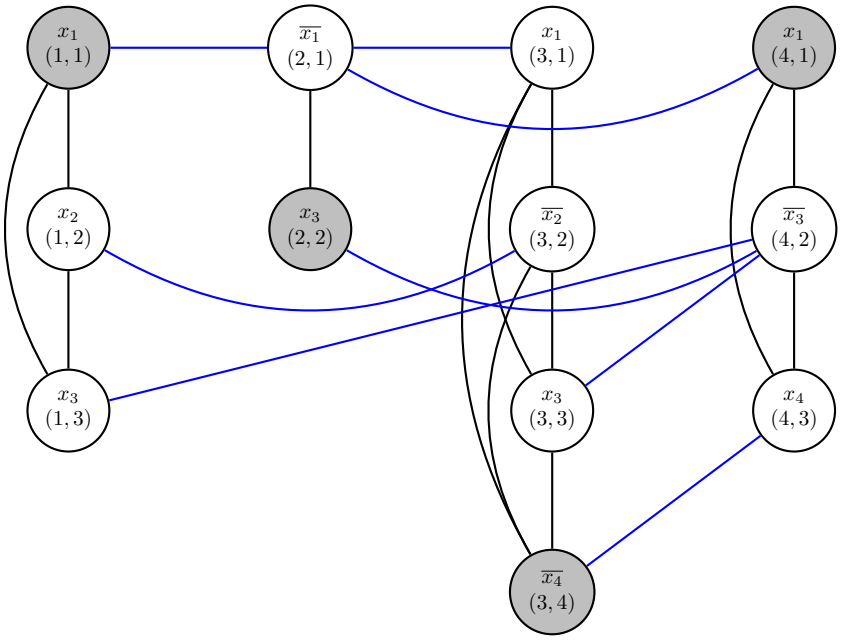

Figura 1: Grafo correspondente da instância exemplo.

Note que as arestas da clique estão desenhadas em preto e o restante das arestas em azul. Temos que a instância SAT é uma instância SIM se e somente se G possui um conjunto independente de tamanho n.

Teorema 1. *E é satisfazível se e somente se G possui um conjunto independente I de tamanho n.*

Demonstração. Se a fórmula é satisfazível, então existe pelo menos um literal verdadeiro por cláusula. Escolha qualquer literal verdadeiro de cada cláusula, e isso forma um conjunto independente em G, pois se existisse uma aresta entre algum par de vértices então teríamos que uma variável e sue negação são verdadeiras ao mesmo tempo, uma contradição.

Se existe um conjunto independente I de tamanho n em G, note que teremos exatamente um vértice de I está em cada clique que representa cada cláusula, uma vez que são cliques. Se atribuirmos o literal correspondente a cada vértice de I como verdadeiro, estaremos satisfazendo E, pois estaremos satisfazendo um literal por cláusula. Além disso, a atribuição é válida, pois se não fosse, haveria alguma aresta entre dois vértices de I, uma contradição.

□

5 Uso de Heurísticas

Por se tratar de um problema NP-Completo, só se conhece algoritmos exponenciais para resolver o problema, e não temos muitas esperanças de conseguir criar um algoritmo polinomial.

Entretanto, em muitas aplicações precisamos encontrar conjuntos independentes grandes para resolver problemas, então o uso de heurísticas pode ser uma solução para que consigamos encontrar um conjunto independente grande, mesmo que não seja o maior.

6 Heurísticas Construtivas da Literatura

Duas heurísticas já existentes foram criadas a fim de comparar resultados dos novos métodos. Aqui, mostramos um algoritmo chamado Ramsey e o clássico algoritmo guloso.

6.1 Ramsey

Esse algoritmo foi desenvolvido por Boppana e Halldórsson. Originalmente, ele retorna uma clique e um conjunto independente máximos, mas só nos preocupamos com a parte do conjunto estável.

Motivado pela Teoria de Ramsey, o algoritmo consiste num procedimento de exclusão de subgrafo e chamadas recursivas.

O caso base da recursão consiste num grafo vazio, que tem como resposta o conjunto vazio. Num caso normal, um vértice v é escolhido, independente do seu grau, e duas chamadas recursivas são feitas. Na primeira, v não faz parte da solução e somente seus vizinhos são considerados. Na segunda, v é inserido no conjunto que é resultado da chamada do algoritmo para a não-vizinhança de v. O maior desses dois conjuntos é retornado.

Algoritmo 1: Ramsey

Entrada: Um grafo simples $G = (V, E)$
Saída: Um conjunto independente I

```
se G = ∅ então
    Retorna ∅
Seja v um vértice de V
I_1 = Ramsey(N(v))
I_2 = {v} ∪ Ramsey(G \ N[v])
se |I_1| ≥ |I_2| então
    Retorna I_1
senão
    Retorna I_2
```

6.2 Guloso

O segundo algoritmo construtivo que mostraremos nesse trabalho é o mais simples e mais famoso. Johnson usa ele já em 1974, ou seja, ele também é um dos mais antigos.

Nele, escolhemos um vértice de grau mínimo e colocamos na solução. Removemos toda sua vizinhança fechada e chamamos recursivamente para o resto do grafo. Quando o grafo é vazio, retornamos um conjunto vazio.

Algoritmo 2: Guloso

Entrada: Um grafo simples $G = (V, E)$
Saída: Um conjunto independente I

```
se G = ∅ então
    Retorna ∅
v ← vértice de V de grau mínimo
retorna {v} ∪ Guloso(G \ N[v])
```

7 Novas Heurísticas Construtivas

Nessa seção vamos mostrar as duas heurísticas construtivas desenvolvidas.

7.1 Warshei um conjunto independente

A primeira heurística construtiva parte da ideia que se dois vértices estão distantes um do outro, então eles provavelmente podem ser colocados na solução.

Para achar esses nodos, começamos rodando o algoritmo de Floyd-Warshall no grafo. Como o grafo é simples, não há pesos nas arestas, então criamos um grafo auxiliar com peso infinito na aresta que liga dois vértices que não eram conectados originalmente e peso um se eles já eram vizinhos.

Depois disso, criamos uma lista ordenada de forma decrescente pela distância entre os pares de vértices e iteramos por ela. Se encontramos um vértice que não tem vizinhos na solução, adicionamos ele.

Algoritmo 3: Warshei um conjunto independente

Entrada: Um grafo simples $G = (V, E)$
Saída: Um conjunto independente I

$A \leftarrow$ `Floyd-Warshall`(G)
$L \leftarrow$ `Ordena`(`Lista`(A))
$I \leftarrow \emptyset$
para $(u, v) \in L$ **faça**
  se $N[u] \notin I$ **então**
    $I \leftarrow I \cup \{u\}$
  se $N[v] \notin I$ **então**
    $I \leftarrow I \cup \{v\}$
retorna I

7.2 Match na Solução

O último algoritmo que apresentamos aqui usa emparelhamento, ou matching. A ideia é que quando fazemos um matching máximo, os vértices que não estão na solução são um conjunto independente. Podemos usá-lo para criar um conjunto independente ainda maior.

O algoritmo começa rodando um algoritmo de emparelhamento máximo. Depois, é definido um conjunto independente I composto pelos vértices que não estão no matching. Removemos toda a vizinhança fechada de I do grafo e unimos esse conjunto com a solução da chamada recursiva para o grafo que restou

Algoritmo 4: Match na Solução

Entrada: Um grafo simples $G = (V, E)$
Saída: Um conjunto independente I

se $G = \emptyset$ **então**
  Retorna $\emptyset$
$S \leftarrow$ `EmparelhamentoMaximo(G)`
$I \leftarrow V \setminus S$
se $I = \emptyset$ **então**
  retorna `Guloso`(G)
retorna $I \cup$ `MatchNaSolução`$(G \setminus N[I])$

8 Metaheurísticas da Literatura

Aqui, vamos descrever dois algoritmos já existentes baseados em metaheurísticas.

8.1 Bruno Nogueira - literatura

Num artigo publicado em 2017, Bruno Nogueira[13] introduziu um algoritmo baseado em ILS e VND para resolver o problema do conjunto independente máximo. A seguir, o pseudo código.

O procedimento `PerturbNogueira`$(G,\ S,\ c_1)$ recebe um conjunto independente e retira c_1 elementos escolhidos aleatoriamente do conjunto S. Depois ele remove a vizinhança fechada de S e do grafo original, roda o procedimento guloso sem escolher um vértice de grau mínimo e une a resposta com S.

Algoritmo 5: Bruno Nogueira - literatura

Entrada: Um grafo simples $G = (V, E)$ e $maxIter$
Saída: Um conjunto independente I

```
1 S = Guloso'(G)
2 Best = S
3 para i = 1 ... maxIter faça
4 |   S = PerturbNogueira(G, S, c_1)
5 |   S = VND(G, S)
6 |   se |S| ≥ |Best| então
7 |   |   Best = S
8 retorna Best
```

O procedimento $\texttt{VND}(S)$ roda o algoritmo VND em duas vizinhanças. Na primeira, para cada vértice do conjunto S, ele é removido, e dois outros vértices são colocados em seu lugar, desde que isso seja válido. Na segunda, um vértice qualquer é colocado na solução enquanto sua vizinhança é removida dela. Depois, rodamos o algoritmo guloso escolhendo sempre um vértice escolhido aleatoriamente.

8.2 Feo

Em um artigo publicado em 1989,[14] Thomas Feo et. al. criaram uma heurística para Conjunto Independente baseada na metaheurística GRASP. Soluções iniciais aleatórias são geradas, e em seguida é aplicada a seguinte busca local: para um conjunto independente S, retira-se dois vértices e em seguida procura três vértices para inserir no conjunto.

Algoritmo 6: Feo

Entrada: Um grafo simples $G = (V, E)$ e $maxIter$
Saída: Um conjunto independente I

```
1 S* = ∅
2 para i = 1 ... maxIter faça
3 |   S = Init(G)
4 |   S = LocalFeo(G, S)
5 |   se |S| ≥ |S*| então
6 |   |   S* = S
7 retorna S*
```

Algoritmo 7: LocalFeo

Entrada: Um grafo simples $G = (V, E)$ e um conjunto independente S de G
Saída: Um conjunto independente I

```
1 para i ∈ S faça
2 |   para j ∈ S faça
3 |   |   S' = S − {i, j}
4 |   |   para u, v, w ∈ V − N[S'] faça
5 |   |   |   se (u, v) ∉ E e (v, w) ∉ E e (u, w) ∉ E então
6 |   |   |   |   retorna LocalFeo(G, S' ∪ {u, v, w})
7 retorna S
```

O processo de gerar a solução inicial aleatória é bastante complicado, e por isso não será descrito aqui.

9 Novas Algoritmos Baseados em Metaheurísticas

9.1 Bruno Nogueira - nosso

Aqui, descrevemos um novo algoritmo baseado naquele desenvolvido por Bruno Nogueira. De novo, vamos usar ILS e VND, mas mudamos a forma de construir a solução inicial e a vizinhança usada.

Algoritmo 8: Bruno Nogueira - nosso

Entrada: Um grafo simples $G = (V, E)$ e $maxIter$
Saída: Um conjunto independente I

```
1 S = Guloso(G)
2 Best = S
3 para i = 1 ... maxIter faça
4 |   S = PerturbBruno(G, S)
5 |   S = VND(G, S)
6 |   se |S| ≥ |Best| então
7 |   |   Best = S
8 retorna Best
```

O procedimento `PerturbBruno`(G, S) recebe um conjunto independente de tamanho m, sorteia um número aleatório n entre 1 e m e retira n elementos escolhidos aleatoriamente do conjunto S. Depois ele remove a vizinhança fechada de S e do grafo original, roda o procedimento guloso já descrito e une S com a saída do algoritmo guloso.

O procedimento `VND`(S) roda o algoritmo VND em duas vizinhanças. Na primeira, para cada vértice do conjunto S, ele é removido, e o algoritmo guloso é executado no grafo sem a vizinhança fechada do que restou de S. Na segunda, um vértice qualquer é colocado na solução enquanto sua vizinhança é removida dela. Depois, rodamos o algoritmo guloso.

9.2 Bonito

Baseado na heurística de Feo, utilizamos a metaheurística ILS-VND para encontrar o conjunto independente. Uma solução inicial é gerada usando o algoritmo guloso, e, a cada iteração, uma perturbação é feita removendo alguns vértices e tentando adicionar outros. Em seguida, executamos a mesma busca local usado pelo algoritmo Feo.

Algoritmo 9: Bonito

Entrada: Um grafo simples $G = (V, E)$ e $maxIter$
Saída: Um conjunto independente I

```
1 S* = ∅
2 S = Guloso(G)
3 para i = 1 ... maxIter faça
4 |   S = PerturbBonito(G, S)
5 |   S' = LocalFeo(G, S)
6 |   se |S'| ≥ |S*| então
7 |   |   S* = S'
8 retorna S*
```

10 Experimentos Computacionais

Nessa seção, vamos comparar o desempenho de todos os experimentos mostrados. Executamos 25 casos de teste gerados de forma que sabíamos o tamanho do conjunto independente máximo.

Usamos 5 instâncias com respectivos ótimos 30, 35, 40, 45 e 50. Para gerar os gráficos, foi feita uma média da saída dos algoritmos em cada um desses grupos.

10.1 Ambiente computacional

Pra executar os experimentos, foi usado uma máquina com sistema operacional Ubuntu 16.04 LTS 64-bit, com 16 GB de RAM e processador Core i7.

10.2 Resultados das Heurísticas Construtivas

Ao executar as heurísticas construtivas, vemos, na tabela 1 e na figura 2, que os resultados são, no geral, bem ruins. A heurística que utiliza matching máximo parece ser um pouco melhor que as outras, porém o gap de todas ficou em torno de 30%.

		Ramsey		**Guloso**		**Floyd**		**Matching**	
Instância	Ótimo	Solução	Tempo(ms)	Solução	Tempo(ms)	Solução	Tempo(ms)	Solução	Tempo(ms)
1	30	22	64	20	10	21	78	20	16
2	30	19	52	20	10	21	77	20	11
3	30	23	60	23	8	22	78	23	11
4	30	21	28	22	9	22	91	22	10
5	30	21	29	22	8	22	88	22	11
6	35	26	31	23	12	24	171	28	32
7	35	23	27	23	14	25	169	24	22
8	35	27	28	26	18	24	189	29	29
9	35	25	27	26	20	24	170	25	18
10	35	24	27	26	15	24	167	25	36
11	40	30	41	30	22	29	341	30	18
12	40	27	41	21	18	28	329	31	21
13	40	29	41	29	18	26	326	29	24
14	40	27	42	27	25	30	335	27	20
15	40	28	41	27	58	30	423	27	18
16	45	33	58	34	30	30	733	35	33
17	45	31	58	32	23	31	605	34	78
18	45	32	60	32	23	32	717	31	33
19	45	31	73	31	24	31	683	32	36
20	45	33	84	33	23	32	621	31	42
21	50	31	13	35	34	36	1200	35	34
22	50	35	85	35	35	37	1165	35	34
23	50	35	12	34	35	35	1180	34	33
24	50	35	17	38	35	34	1102	38	35
25	50	37	11	38	35	35	1169	38	34

Tabela 1: Resultados das Heurísticas Construtivas

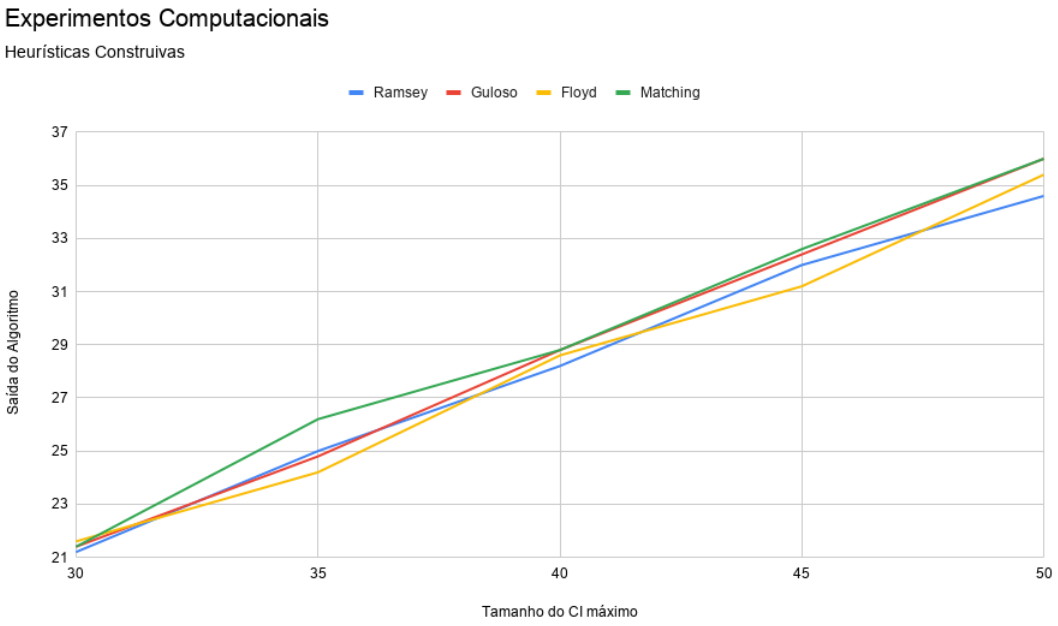

Figura 2: Resultados das Heurísticas Construtivas

10.3 Resultados das Metaheurísticas

Ao executar as metaheurísticas, vemos na tabela 2 e na figura 3 que o algoritmo de Feo[14] foi o que obteve os melhores resultados. Vemos que as metaheurísticas, no geral, são muito melhores que as heurísticas construtivas, porém elas demoram ordem de grandeza a mais para executar.

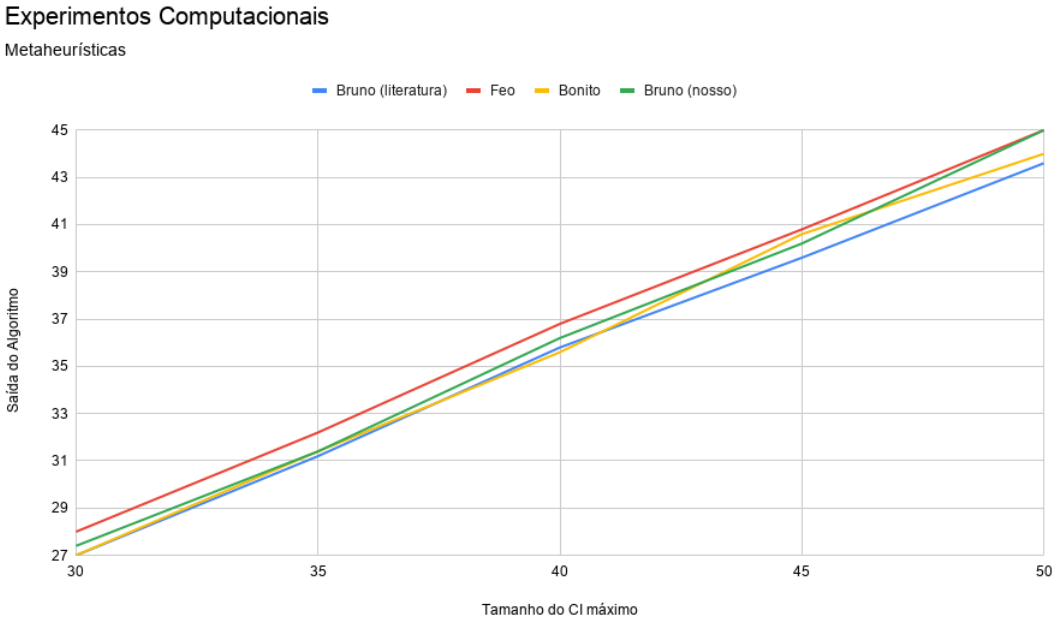

Figura 3: Resultados das Metaheurísticas

		Bruno (nosso)		**Bruno (literatura)**		**Bonito**		**Feo**	
Instância	Ótimo	Solução	Tempo(ms)	Solução	Tempo(ms)	Solução	Tempo(ms)	Solução	Tempo(ms)
1	30	27	2535	27	2590	27	1311	28	8851
2	30	28	2504	27	2606	26	1121	28	8954
3	30	27	2376	26	2565	27	1375	27	8288
4	30	27	2525	27	2602	27	1447	29	8744
5	30	28	2529	28	2939	28	1283	28	8542
6	35	31	4695	31	5541	31	2595	31	17588
7	35	32	5037	31	5188	32	2530	32	16858
8	35	32	4685	31	5168	32	2906	33	18326
9	35	31	5114	31	5455	31	2691	32	17479
10	35	31	5093	32	5462	31	2584	33	17716
11	40	35	8641	35	9496	35	5526	37	34049
12	40	36	9200	36	9515	36	5286	37	33849
13	40	37	9474	35	9442	35	4783	36	33502
14	40	37	8953	36	9550	37	5386	37	33079
15	40	36	10293	37	13041	35	4806	37	32340
16	45	41	15818	39	18201	42	8445	41	55966
17	45	41	16225	39	29361	40	8530	40	56813
18	45	39	17168	40	16428	40	8556	41	55018
19	45	40	17825	40	18564	41	9437	41	59351
20	45	40	15355	40	17096	40	9956	41	558780
21	50	44	25384	44	31167	44	18844	46	10332
22	50	44	30118	44	30308	44	16862	45	980184
23	50	46	29259	43	29955	44	18032	45	10612
24	50	45	31773	44	30763	44	17182	44	980675
25	50	46	28478	43	30760	44	21037	45	10046

Tabela 2: Resultados das Metaheurísticas

11 Conclusão

Foi possível ver com clareza a deficiência das heurísticas construtivas em aproximar o problema do Conjunto Independente. Enquanto isso, as metaheurísticas tiveram êxito em encontrar conjuntos independentes grandes em relação ao máximo da instância.

Encontramos dificuldades principalmente envolvendo a leitura de artigos de uma área nunca antes por nós estudada. Com uma certa dose de perseverança, foi possível compreender a intenção das autoras das artigas e, com isso, implementar as heurísticas e metaheurísticas.

Por fim, pela primeira vez tivemos que lidar com a construção de experimentos computacionais. Depois de estudar a melhor maneira de realizá-los, concluímos com sucesso o trabalho.

Referências

[1] Richard M. Karp (1972). Reducibility Among Combinatorial Problems. Complexity of Computer Computations. Plenum: Pages 85–103

[2] Boppana, R.; Halldórsson, M. M. (1992). Approximating maximum independent sets by excluding subgraphs. BIT Numerical Mathematics, Volume 32: Pages 180–196

[3] Bomze, I. M.; Budinich, M.; Pardalos, P. M.; Pelillo, M (1999). The Maximum Clique Problem. Handbook of Combinatorial Optimization

[4] Approximation algorithms for combinatorial problems. Johnson, D. S. (1974). Comput. Sys. Sci, Volume 9: Pages 256-278.

[5] Tomita, E.; Mitsuma, S; Takahashi, H. (1988). Two algorithms for finding a near-maximum clique. Tech. Rep UEC-TR-C1.

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

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

[8] Balas, E.; Yu, C. S. (1986), Finding a maximum clique in an arbitrary graph. SIAM Journal on Computing, Volume 15: Pages 1054–1068

[9] Battiti, R.; Protasi, M. (2001). Reactive local search for the maximum clique problem. Algorithmica, Volume 29: Pages 610–637

[10] Katayama, K.; Hamamoto, A.; Narihisa, H. (2005). An effective local search for the maximum clique problem. Information Processing Letters. Volume 95: Pages 503-511

[11] Grosso, A.; Locatelli, M.; Croce, F.D. Combining Swaps and Node Weights in an Adaptive Greedy Approach for the Maximum Clique Problem. Journal of Heuristics, Volume 10: Pages 100-106

[12] NP-Completeness of Independent Set, <`https://www.nitt.edu/home/academics/departments/cse/faculty/kvi/NPC%20INDEPENDENT%20SET-CLIQUE-VERTEX%20COVER.pdf`>. Acesso em 10/10/2019.

[13] Nogueira, Bruno & Pinheiro, Rian & Subramanian, Anand. (2017). A hybrid iterated local search heuristic for the maximum weight independent set problem. Optimization Letters. 10.1007/s11590-017-1128-7.

[14] Feo, Thomas A., Mauricio G. C. Resende, and Stuart H. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. Operations Research 42, no. 5 (1994): 860-78. `www.jstor.org/stable/171545`.