ESTUDO E IMPLEMENTAÇÃO DE UM ALGORITMO HEURÍSTICO BASEADO EM GRASP PARA SOLUÇÃO DO PROBLEMA DO CONJUNTO INDEPENDENTE MÁXIMO

Leonardo Bravo Estácio[1], Selma Helena Marchiori Hashimoto, Irene Magalhães Craveiro
Faculdade de Ciências Exatas e Tecnologia, FACET, UFGD
79.805-000, Dourados, MS
leonardo.bravo.estacio@gmail.com, {selmahashimoto, irenecraveiro}@ufgd.edu.br

Resumo: O cojunto independente máximo em um grafo tem importantes aplicações e ainda não foram desenvolvidos algoritmos exatos que o resolvem satifatoriamente. A complexidade dos algoritmos exatos tende a ser uma função de crecimento exponencial enquanto a dos algoritmos aproximatimos tende a ser polinomial. Neste artigo apresentamos uma solução aproximativa que resolve o problema, algoritmos aproximativos não garantem uma solução ótima para o problema, mas em geral, geram uma solução satisfatória. Para a formulação do algoritmo foi proposta uma abordagem utilizando a metaheurística GRASP, composta por uma fase construtiva e uma fase de busca local. O algoritmo obteve de forma eficiente o conjunto independente máximo de vários grafos testados, porém é necessário efetuar uma melhor análise para quantizar o seu desempenho e sua complexidade.

Palavras-Chave: *Conjunto Independente Máximo, Grasp*, *Metaheurística*.

Introdução

Um grafo é uma estrutura formada por um conjunto finito de vértices e arestas, denotado por $G = (V, A)$ usada pela primeira vez pelo matemático James Joseph Sylvester (1814-1894) [9]. Onde G representa o nome do grafo, V representa o conjunto de todos os vértices que pertencem a esse grafo e A representa o conjunto de pares formados por exatamente dois vértices.

Uma vizinhança $U \subseteq V$ de um vértice v em um grafo G é denotada como $N_G(v)$, onde $N_G(v) = \{u \in U \mid (u, v) \in A\}$ [23]. Dois vértices em um grafo são considerados adjacentes se estão conectados por uma aresta, ou seja, $(u, v) \in A \Leftrightarrow u, v \in V \wedge v \in N_G(u)$.

Nesse trabalho trataremos apenas de grafos não orientados, onde uma aresta $(u, v) \in A \Leftrightarrow (v, u) \in A$, como neste caso as arestas são equivalentes somente uma delas estará inserida no conjunto A. Usaremos também a notação $|S|$ para representar a cardinalidade de um conjunto, dado $S = \{u, v, w\}$ sua cardinalidade será representada por $|S| = 3$. O grau de um vértice, denotado como $g_G(v)$, é definido como o número de arestas que incidem em um vértice, assim $|N_G(v)| = g_G(v)$.

Um conjunto independente, também chamado de conjunto estável [9], de um grafo

[1] Voluntário de Iniciação Científica, graduando em Matemática Licenciatura da Universidade Federal da Grande Dourados.

$G = (V, A)$ é qualquer conjunto $I \subseteq V$ onde os vértices pertencentes ao conjunto I são elementos não adjacentes uns aos outros, $(u, v) \notin A \ \forall u, v \in I$.

O conjunto independente máximo (acrônimo MIS – *Maximum Independent Subset*) é definido como um conjunto $I_m \subseteq V$ tal que não haja nenhum outro conjunto independente I onde $|I| > |I_m|$. Comumente na literatura $|I_m|$ é denotado como $\alpha(G)$ [24][14] [17][12][18].

De acordo com [6][2][11] dado um subconjunto de vértices I ⊆ V, pode-se verificar em tempo polinomial se esse subconjunto é ou não um conjunto indenpendente, porém não existe um algoritmo que encontre $\alpha(G)$ em tempo polinomial dado um grafo qualquer, sendo classificado como um problema NP-Hard. Entretanto, em alguns casos particulares de grafos, como em grafos perfeitos, o problema pode ser resolvido em tempo polinomial [15].

O estudo desse problema é importante porque possui aplicações em várias áreas como na teoria da codificação, tolerância a falhas, visão computacional, reconhecimento de padrões, química de proteínas, economia, entre outros [6][24][17].

Existem muitos trabalhos publicados, como visto em [24][14][12], visando encontrar algoritmos aceitáveis que solucionem o MIS. Para isso duas abordagens veem sendo exploradas: métodos exatos que resolvem o problema em tempo não polinomial [13][19] [22][10][21] e algoritmos aproximativos, que operam em tempo polinomial [20][3][4] [5][16], porém não garantem uma solução ótima. GRASP é uma metaheurística que pode ser usada no contexto de uma abordagem aproximativa.

GRASP foi proposta por [7], estudada afundo em [8], seu nome vem do inglês *Greedy Randomized Adaptive Search procedure*, que significa algoritmo heurístico construtivo guloso que usa uma componente aleatória e adaptativa sendo capaz de escapar de ótimos locais.

Qualquer algoritmo que seja proposto utilizando a metaheurística GRASP deverá possui duas fases básicas. A fase construtiva e a de busca local. A fase construtiva deverá apresentar os seguintes passos genéricos:

1. Escolhida uma solução inicial (que pode ser vazia), ela se transforma na solução corrente.
2. Para a solução corrente e usando um índice de desempenho ou indicador de sensibilidade, elaborar uma lista com as k componentes mais atrativas.
3. Escolhida uma componente da lista, essa é adicionada a solução corrente e o problema é atualizado com a exclusão da mesma.
4. Se a solução corrente é factível ou foi satisfeito o critério de parada (sem encontrar uma solução factível) terminar com a fase construtiva. Caso contrário voltar ao passo 2.

A fase construtiva termina com uma solução corrente factível e de boa qualidade,

no entanto, procura-se melhorá-la através da segunda fase do GRASP que é um algoritmo de otimização local, o qual consiste em fazer uma busca sistemática com o objetivo de encontrar uma solução factível de melhor qualidade a partir da solução corrente.

A fase de busca local em um algoritmo GRASP procura encontrar uma solução ótima local na vizinhança da solução encontrada na fase de busca construtiva. O processo consiste em definir uma vizinhança da solução encontrada pela fase construtiva e encontrar uma solução factível de melhor qualidade nessa vizinhança. Sempre que é encontrada uma solução factível de melhor qualidade a busca local deve ser reinicializada encontrando uma nova vizinhança da nova solução para reiniciar a busca local.

Este processo, geralmente, pode precisar de um esforço computacional relativamente elevado dependendo do tipo de problema. Em geral, a fase de busca local pode ser um processo muito simples ou muito complexo, na implementação e no esforço computacional.

Metodologia

No presente trabalho, a metodologia GRASP implementada para a resolução do MIS, possui a seguinte estrutura:

Fase construtiva:

1. Seja S o conjunto de vértices independentes que será encontrado na fase construtiva. Iniciar o processo com $S = \emptyset$. Seja S a solução corrente, fazer o contador de iterações $k = 1$.
2. A partir da solução corrente S montar um sub-grafo G_k após eliminar todos os vértices que estão em S ou que possuam uma aresta em comum com algum vértice em S. Seja V_k o conjunto de vértices de G_k, se $V_k = \emptyset$; então foi encontrada uma solução do problema na fase construtiva e terminar o processo. Caso contrario, ir ao passo 3.
3. Encontrar o grau mínimo D do subgrafo $G_k = (V_k, A_k)$ utilizando a Equação (5).

$$D = \min \left\{ g_{G_k}(v); v \in V_k \right\} \qquad (5)$$

4. Montar o conjunto reduzido de arestas RCL, que define os vizinhos atrativos da solução corrente, usando a relação dada pela Equação (6) em que α é um parâmetro fornecido pelo usuário que varia entre 0 e 1.

$$\text{RCL} = \{v \in V_k : D_k < (1+\alpha)D; \alpha > 0\} \qquad (6)$$

5. Selecionar uma aresta j de RCL aleatoriamente e atualizar o conjunto S, Equação (7).

$$S = S \cup \{j\} \qquad (7)$$

6. Atualizar o contador de iterações e voltar ao passo 2.

Fase de otimização local:

Na fase de busca local foi implementada uma busca na vizinhança da solução encontrada na fase construtiva com tamanho fixo t. Esse processo de busca pode ser resumido nos seguintes passos:

1. Seja S uma solução com m vértices encontrados na fase construtiva.
2. Encontrar todos os subconjuntos de S com cardinalidade t. Supor que existe p subconjuntos desse tipo: W_1, W_2, ..., W_p.
3. Para cada subconjunto W_p fazer seguinte:
   a. Identificar o conjunto S_i cujos elementos são elementos de S que não estão em W_p.
   b. Montar um grafo G_i formado pelos vértices do grafo que não estão em S_i nem são vizinhos de um vértice que está em S_i.
   c. Encontrar o conjunto máximo de tamanho N_i do grafo G_i por busca exaustiva.
   d. Se a cardinalidade de $S_i \cup N_j$ é maior que m então foi encontrada uma solução de melhor qualidade que S.

Resultados

Como resultado do estudo foi implementado um software, Figura 1, utilizando a linguagem de programação C++ em conjunto da IDE qt-creator.

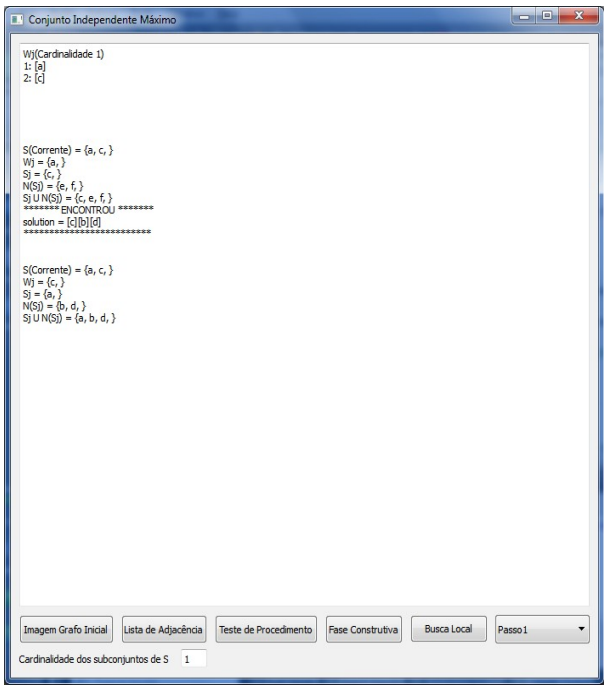

Figura 1 – Software Conjunto Independente Máximo

Seja o grafo dado pela Figura 2, mostraremos a seguir o passo a passo do algoritmo proposto. Adotando o valor do parâmetro α = 0,6, a fase construtiva do algoritmo segue os seguintes passos.

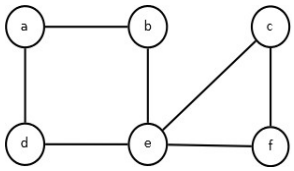

Figura 2 - Grafo $G_1 = (V, A)$, $|V|=6$ e $|A|=7$

Passo1: Iniciamos com $S = \varnothing$;
Passo2: G_1 é gerado com os vértices $V_1 = \{a, b, c, d, e, f\}$;
Passo3: D = min ($g_{G_1}(a)$, $g_{G_1}(b)$, $g_{G_1}(c)$, $g_{G_1}(d)$, $g_{G_1}(e)$, $g_{G_1}(f)$,) = min (2, 2, 2, 2, 4, 2) = 2;
Passo4: $(1 + \alpha).D = (1 + 0{,}6).2 = 3{,}2 \rightarrow$ RCL = $\{a, b, c, d, f\}$;
Passo5: Aleatoriamente é selecionado o vértice $a \rightarrow S = \{a\}$;
Passo6: k = 2.

Passo2: G_2 é gerado com os vértices $V_2 = \{c, e, f\}$. Conforme a Figura 3;

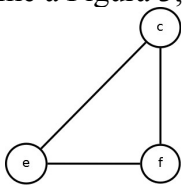

Figura 3 - Grafo $G_2 = (V, A)$, $|V|=3$ e $|A|=3$

Passo3: D = min ($g_{G_2}(c)$, $g_{G_2}(e)$, $g_{G_2}(f)$) = min (2, 2, 2) = 2;
Passo4: $(1 + \alpha).D = (1 + 0{,}6).2 = 3{,}2 \rightarrow$ RCL = $\{c, e, f\}$;
Passo5: Aleatoriamente é selecionado o vértice $c \rightarrow S = \{a, c\}$;
Passo6: k = 3.

Passo2: G_3 é gerado com os vértices V_3 =

$\mathcal{C}$.

Como $|V_3| = 0$, então foi encontrada a solução construtiva $S = \{a, c\}$. Para a busca local foram gerados os subconjuntos de S com cardinalidade $t = 1$.

Passo1: S = {*a*, *c*}.
Passo2: $\mathcal{W}_P = \{\{a\},\{c\}\}$.
Passo3: $\mathcal{W}_1 = \{a\}$.

a: $S_i = S - \mathcal{W}_1$, $S_i = \{c\}$.

b: $N_{G_1}(S_i) = \{e, f\}$, o grafo G_4, representando pela Figura 4, é gerado a partir de G_1 sem os elementos $N_{G_1}(S_i) \cup S_i = \{c, e, f\}$.

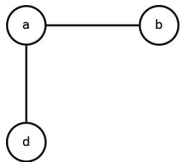

Figura 4 - Grafo G4 = (V, A), |V|=3 e |A|=3

c: Foi encontrado o $\alpha(G_4)$ através do algoritmo *Naive*, descrito em [1]. $\alpha(G_4) = \{b, d\}$.

d: como $|\alpha(G_4)| + |S_i| > |S|$, isto é, 2 + 1 > 2. Foi encontrada uma solução melhor que *S*. $S = \alpha(G_4) \cup S_i \rightarrow S = \{c, b, d\}$.

Ao final dessa iteração é encontrada a solução ótima $\alpha(G_1)$, então os demais laços serão apenas executados e nenhuma outra solução de melhor qualidade irá ser encontrada.

Também foram feitos testes no grafo apresentando pela Figura 5.

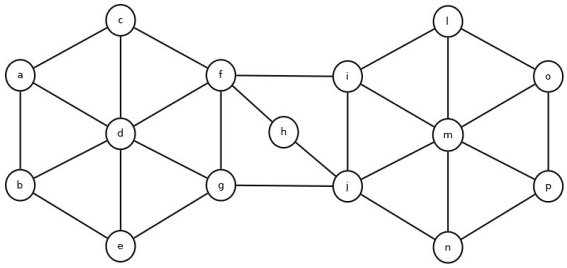

Figura 5 - Grafo G_5 = (V, A), $|V|$=15 e $|A|$=28

Sendo obtidos os seguintes resultados:

Iterações Utilizadas	Conjunto Independente encontrado	Tamanho dos Subconjuntos Testados
5	s = {p, h, c, e, l}	
6	S = {p, h, c, l, b, g}	*t*=1
8	S = {h, c, b, g, i, n, o}	*t*=1, 2
5	s = {l, n, h, e, c}	
6	s = {l, n, h, c, b, g}	*t*=1
7	s = {n, h, c, b, g, i, o}	*t*=1
6	s = {a, e, o, n, i, h}	
7	s = {a, e, o, n, i, h}	*t*=1
9	s = {o, n, i, h, b, c, g}	*t*=1, 2

Conclusão

O presente trabalho apresenta uma proposta de formulação e implementação de um algoritmo baseados na técnica de otimização GRASP (*Greed Randomized Search Procedure*), para solucionar o problema do conjunto independente máximo, que é um importante problema da literatura com várias aplicações em diferentes áreas de conhecimento.

O algoritmo obteve de forma eficiente o conjunto independente máximo de vários grafos testados, porém é necessário efetuar uma melhor análise para quantizar o seu desempenho e complexidade.

Esse corrente trabalho ainda está em andamento e os próximos passos previstos são sucessivas análises nos grafos propostos pela DIMACS (*Discrete Mathematics and Theoretical Computer Science*) e formulação

de uma análise de *benchmark* através de gráficos comparativos.

Referências

[1] ALVIM, M.S.F. Projeto e Análise de Algoritmos 2º Trabalho Prático, 2006, Disponível no endereço eletrônico: http://homepages.dcc.ufmg.br/~nivio/cursos/pa06/tp2/tp22/tp22.pdf, Data de Acesso: 16/09/2011.

[2] BAKER, B.S. Approximation algorithms for NP-complete problems on planar graphs. **Journal of the ACM**. New York, 01 Jan. 1994. Vol. 41, No. 1, p. 153-180.

[3] CHIBA, N.; NISHIZEKI, T.; SAITO, N. An approximation algorithm for maximum independent set in planar traingle-free graphs. **SIAM Journal on Computing**, 1982. Vol. 11, No. 4, p. 663-674.

[4] CHOUKHMANE, E.; FRANCO, J. An Approximation Algorithm for the Maximum Independent Set Problem in Cubic Planar Graphs. **Network Journal**, 11 Oct. 2006. Vol. 16, No. 4, p. 349-356.

[5] CRESCENZI, P.; FIORINI, C.; SILVESTRI, R. A Note on the Approximation of the MAX CLIQUE Problem. **Information Processing Letters**, Oct. 1991. Vol. 40, No. 1, p. 1-5.

[6] DU, Z.; PARDALOS, P.M. **Handbook of Combinatorial Optimization:** Supplement Volume A. 1.ed. Dordrecht: Kluwer Academic Publishers, 1999. 656p.

[7] FEO, T.A.; RESENDE, M.G.C. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, 1989. Vol. 8, No. 2, p. 67-71.

[8] FEO, T.A.; RESENDE, M.G.C. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, 1995. Vol. 6, No. 2, p. 109-133.

[9] FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. **Uma Introdução Sucinta à Teoria dos Grafos**. Ed. 2011. Salvador: SBM, 2004. 61p.

[10] FOMIN, F.V.; GRANDONI, F.; KARTSCH, D. Measure and Conquer: A Simple $O(2^{0.288n})$ Independent Set Algorithm. In: SODA '06 Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, 2006, New York. **proceedings…** New York: ACM, 2006. p. 18-25.

[11] GAREY, M.R.; JOHNSON, D.S. **Computers and Intractability:** A guide to the theory of NP-completeness, New York: Freeman, 1979. 340p.

[12] GOLUMBIC, M.C. **Algorithmic Graph Theory and Perfect Graphs**, New York: Academic Press, 1980. 284p.

[13] JIAN, T**.** An $O(2^{0.304n})$ Algorithm for Solving Maximum Independent Set Problem. **Journal IEEE Transactions on Computers**, Washington, September 1986. Vol. 35, No. 9, p. 847-851.

[14] JOU, M.J.; CHANG, G.J. The number of maximum independent sets in graphs, **Taiwanese Journal of Mathematics**. Taiwan, Dec. 2000. Vol. 4, No. 4, p. 685-695.

[15] LOZIN, V.; MILANIC, M. On the Maximum Independent Set Problem in Subclasses of Planar Graphs. **Journal of Graph Algorithms and Applications**, 2010. Vol. 14, No. 2, p. 269-286.

[16] MURPHY, W.J. Computing Independent Sets in Graphs with Large Girth. **Discrete Applied Mathematics**, 9 Jan. 1992. Vol. 35, No. 2, p. 167-170.

[17] PARDALOS, P.M.; XUE, J. The Maximum Clique Problem. **Journal of Global Optimization**, April 1994. Vol. 4, No. 3, p. 301-328.

[18] RÉGIN, J.C. Solving the Maximum Clique Problem with Constraint Programming. In**:** Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2003, Montreal. **proceedings…** Montreal: CA-AI-OR, 2003. p. 634-648.

[19] ROBSON, J.M. Algorithms for maximum independent sets. **Journal of Algorithms**, Sep. 1986. Vol. 7, No. 3, p. 425-440.

[20] ROPPANA, R.; HALLDÓRSSON, M.M. Approximating Maximum Independent Sets by Excluding Subgraphs, **BIT Numerical Mathematics**, 1992. Vol. 32, No. 2, p. 180-196.

[21] SHARIEH, A.; AI_RAWAGEPFEH, W.; MAHAFZAH, M.H.; DAHAMSHEH, A.A. An Algorithm for Finding Maximum Independent Set in a Graph. **European Journal of Scientific Research**, November 2008. Vol. 23, No. 4, p. 586-596.

[22] TARJAN, R.E.; TROJANOWSKI, A.E. Finding a Maximum Independent Set. **SIAM Journal on Computing**, 1977. p. 537-546.

[23] WARREN, J.S.; HICKS, I.V. Combinatorial branch-and-bound for the maximum weight independent set problem. 2007.

[24] WILSON, A.T. **Applying the Boundary Point Method to an SDP Relaxation of the Maximum Independent set problem for a branch and Bound Algorithm**. 2009. 71p. Dissertação (Master of Science in Mathematics) - New Mexico Institute of Mining and Technology. Socorro - Novo México, 2009.