LI Simpósio Brasileiro de Pesquisa Operacional
Limeira–SP, 02 a 06 de setembro de 2019

Um Algoritmo Exato para o Problema da Biclique Induzida Balanceada Máxima com Partições em Cliques e Bonecas Russas

Marcus Vinicius Martins Melo
Universidade Federal do Ceará - Campus de Crateús
Crateús, Ceará, CEP 63700-000
viniips@alu.ufc.br

Rennan Ferreira Dantas, Luiz Alberto do Carmo Viana
Universidade Federal do Ceará - Campus de Crateús
Crateús, Ceará, CEP 63700-000
rennan@ufc.br, luizalberto@crateus.ufc.br

RESUMO

Neste trabalho apresentamos um algoritmo com Bonecas Russas e Partições em Cliques para o Problema da Biclique Induzida Balanceada Máxima (PBIBM). A estratégia do algoritmo é utilizar Partições em Cliques para podar cada subproblema gerado por cada boneca. Além disso adicionamos um procedimento de *Upper Bound Propagation* (UBP) desenvolvido por [Zhou et al., 2018]. Por fim, uma comparação com os algoritmos proposto por [Silva, 2016] e [McCreesh e Prosser, 2014] utilizando as instâncias presentes em DIMACS e KONECT foram realizadas. O algoritmo desenvolvido se tornou mais eficiente que a solução proposta por [Silva, 2016] e superou o algoritmo de [McCreesh e Prosser, 2014] em algumas instâncias, principalmente para grafos bipartido.

PALAVRAS CHAVE. Biclique Induzida Balanceada, Bonecas Russas, Partições em Cliques.

Tópicos: Otimização Combinatória, Teoria e Algoritmos em Grafos.

ABSTRACT

In the present paper, we introduce an algorithm based on Russian Dolls and Clique Partition for the Maximum Balanced Induced Biclique Problem (PBIBM). The algorithm consists of using Clique Partition for prunning each doll related subproblem. Besides, we add a procedure of the *Upper Bound Propagation* (UBP) that was proposed in [Zhou et al., 2018]. At last, we present a comparison with the algorithms proposed in [Silva, 2016] and [McCreesh e Prosser, 2014], using instance sets DIMACS and KONECT. Our proposed algorithm turns out to be more efficient than [Silva, 2016], and surpassed [McCreesh e Prosser, 2014] in some instances, most of them being bipartite graphs.

KEYWORDS. Balanced Induced Biclique, Russian Dolls, Clique Partition.

Topics: Combinatorial Optimization, Theory and Algorithms in Graphs.

1. Introdução

Seja $G = (V, E)$ um grafo simples e não-direcionado formado por um conjunto finito e não-vazio de vértices $V = \{v_1, \ldots, v_n\}$ e um conjunto finito de arestas $E \subseteq \{\{u, v\} : u, v \in V \text{ e } u \neq v\}$. A vizinhança de um vértice v é denotada por $N_G(v) = \{u \in V : \{u, v\} \in E\}$. O grau de um vértice v é denotado por $deg(v) = |N_G(v)|$.

Uma clique é um grafo onde todos os seus vértices são adjacentes entre si. Denotamos por K_n, uma clique de tamanho n. Uma partição de um conjunto de vértices V' é uma coleção de subconjuntos não-vazios $V'_1, V'_2, \ldots, V'_n$, tal que, $\bigcup_{i=1}^{n} V'_i = V'$ e $V'_i \cap V'_j = \emptyset$, para todo $i \neq j$. Uma partição em cliques é uma partição dos vértices do grafo em que cada subconjunto é uma clique.

Uma biclique é um par de subconjuntos de vértices $\{A, B\}$ tal que existe uma aresta $\{u, v\}$, para todo $u \in A$ e $v \in B$. A biclique é dita induzida se A e B são conjuntos independentes. A biclique é dita balanceada se $|A| = |B|$. Várias aplicações podem ser modeladas utilizando bicliques em diversas áreas, algumas podem ser encontradas em [Al-Yamani et al., 2007], [Cheng e Church, 2000] e [Ravi e Lloyd, 1988].

O Problema da Biclique Induzida Balanceada Máxima (PBIBM) é encontrar uma biclique induzida balanceada de tamanho máximo em um grafo arbitrário. Este problema pertence a classe de complexidade NP-Difícil [Gary e Johnson, 1979]. Uma forma de resolver esse tipo de problema seria encontrando propriedades na estrutura de grafo utilizada que possam ser úteis para encontrar um algoritmo eficiente para resolvê-lo. Uma outra forma seria decompor o problema em subproblemas que podem ser resolvidos eficientemente. Vários algoritmos são propostos na literatura utilizando estas estratégias, alguns exatos e outros heurísticos.

Em [McCreesh e Prosser, 2014] uma solução exata para o PBIBM é proposta utilizando *Branch and Bound* e Partições em Cliques. Uma ordem inicial estática dos vértices é definida para particionar as cliques, esta ordem é fixa e evita a reordenação dos vértices. Uma estratégia para o problema da simetria na expansão dos vértices é desenvolvida. Dada a biclique $\{A, B\}$, são consideradas todas as possibilidades de um vértice pertencer ao conjunto A no topo da busca, para evitar que em algum momento ele seja adicionado ao conjunto B.

Já em [Silva, 2016] um algoritmo exato para o PBIBM é desenvolvido utilizando *Branch and Bound* e Bonecas Russas. Esse algoritmo se baseia em [McCreesh e Prosser, 2014], embora não utilize Partições em Cliques para podar os subproblemas e sim cortes via método das Bonecas Russas.

2. Algoritmo de Partições em Cliques

[McCreesh e Prosser, 2014] propõe um algoritmo *Branch and Bound* para o PBIBM. Neste algoritmo cada subproblema é definido por uma tupla $(A, B, P_a, P_b, A_{max}, B_{max})$, onde A e B representam a biclique que está sendo construída, P_a e P_b são os possíveis vértices candidatos ao conjunto A e B respectivamente e A_{max} e B_{max} a maior biclique encontrada.

Um critério de poda simples para esse algoritmo seria $|A|+|P_a| < |A_{max}|$ ou $|B|+|P_b| < |B_{max}|$. Assim um subproblema pode ser podado quando a biclique que está sendo construída mais os possíveis candidatos não conseguem superar a melhor solução encontrada. Entretanto este critério considera o tamanho absoluto do conjunto de candidatos, e como A e B devem ser conjuntos independentes, nem todos os candidatos são aptos. A técnica de Partições em Cliques então é aplicada visando aperfeiçoar esse limite, levando em consideração a seguinte Proposição:

Proposição 1. *[McCreesh e Prosser, 2014] Seja $G = (V, E)$ um grafo arbitrário, se G pode ser particionado em k cliques, então k é um Upper Bound para um conjunto independente máximo em G.*

Demonstração. *Suponha por contradição que exista um conjunto independente de tamanho $k+1$. Como G é particionado em k cliques, isso necessariamente ocasionaria de dois vértices do conjunto independente estarem contidos em uma mesma clique, e serem obviamente vértices adjacentes. Absurdo.*

A partir da proposição 1, o Algoritmo 1 é elaborado abaixo:

Algoritmo 1 Algoritmo de Partição em Cliques

```
1:  função CLIQUESORT(G, P)
2:      bounds ← vetor de inteiros
3:      order ← vetor de inteiros
4:      P′ ← P
5:      k ← 1
6:      i ← 1
7:      enquanto P′ ≠ ∅ faça
8:          Q ← P′
9:          enquanto Q ≠ ∅ faça
10:             v ← primeiro elemento de Q
11:             P′ ← P′ \ {v}
12:             Q ← Q ∩ N_G(v)
13:             bounds[i] ← k
14:             order[i] ← v
15:             i ← i + 1
16:         fim enquanto
17:         k ← k + 1
18:     fim enquanto
19:     devolve (bounds, order)
20: fim função
```

A função *CLIQUESORT* no Algoritmo 1, produz dois arrays, um array de $bounds$ e um de $order$. O array $order$ contém os vértices de P em ordem arbitrária. O array de $bounds$ contém limites para o tamanho do maior conjunto independente, isto é, o subgrafo induzido pelos vértices $v_1, v_2, \ldots, v_n$ de $order$ não pode ter um conjunto independente maior que $bounds[n]$.

Os arrays são criados da seguinte forma: a variável P' contém inicialmente todos os vértices que podem formar uma clique, ou seja, todos os vértices do conjunto P. Enquanto P' não for vazio, podemos construir uma nova clique, então Q guarda os vértices candidatos a nova clique corrente. Inicialmente considera-se que todos os vértices de P' formam uma clique. Na linha 10, um vértice $v \in Q$ é selecionado e então todos os vértices em Q que não são adjacentes a v são filtrados. Note que depois que um vértice é selecionado para pertencer a uma clique, ele deve ser removido

de P' para que não seja utilizado em outra clique posteriormente. O procedimento continua até que Q seja vazio. Enquanto houver vértices não selecionados, uma nova clique é construída.

A partir da função *CLIQUESORT* obtemos um limite superior aperfeiçoado. Ao selecionar um vértice de P_a, ao invés de utilizar $|P_a|$ como limite, utiliza-se o array $bounds$ para tal finalidade.

3. Algoritmo de Bonecas Russas

O método das Bonecas Russas foi proposto originalmente por [Verfaillie et al., 1996] para um problema de satisfação booleana. O método das Bonecas Russas consiste em subdividir o problema em problemas menores, teoricamente mais fáceis, e através dos menores resolver os maiores, possivelmente mais difíceis, até chegar no problema completo. Em geral, a estrutura do método das Bonecas Russas se divide em duas etapas: a primeira consiste na técnica de enumeração de bonecas e a segunda na estratégia de resolução das bonecas.

A técnica de Bonecas Russas é comumente utilizada em problemas de otimização discreta, como pode ser visto em [Vaskelainen et al., 2010]. Segundo [Vaskelainen et al., 2010], a facilidade de implementação deste método aumenta na chance de obter resultados satisfatórios. Além disso, também é uma técnica comumente utilizada em problemas que envolvem grafos, como pode ser visto em [Corrêa et al., 2014] e [Araujo Tavares, 2016] onde a utilização desta técnica proporcionou bons resultados.

O algoritmo proposto por [Silva, 2016] utiliza a mesma técnica de *Branch and Bound* que [McCreesh e Prosser, 2014]. A única diferença é não utilizar Partições em Cliques como *bound* em cada subproblema, mas com o incremento de poda via método de Bonecas Russas.

A técnica de Bonecas Russas têm características em comum com Programação Dinâmica, pois ambas armazenam resultado de subproblemas já calculados, mas diferem na maneira em que eles utilizam esses resultados. Enquanto Bonecas Russas utiliza o resultado somente como $bound$ durante a busca, Programação Dinâmica combina os resultados para obter o resultado final [Verfaillie et al., 1996].

No Algoritmo 2 temos a implementação dessa técnica.

Algoritmo 2 Algoritmo de Bonecas Russas

1: **função** BONECASRUSSAS(G, A, B)
2: ORDENACAO(G) ▷ ordenação inicial dos vértices
3: R ← vetor de inteiros
4: **para** i ← 1 ate $|V_G|$ **faça**
5: $v \leftarrow V_G[i]$
6: PA ← $V_G \setminus \overline{N_G(v)}$
7: PB ← $V_G \setminus N_G(v)$
8: $A' \leftarrow v$
9: $B' \leftarrow \emptyset$
10: NOVOEXPAND($G, B', A', PB, PA, B, A, R$)
11: R[v] ← $|A|$
12: **fim para**
13: **fim função**

O algoritmo funciona da seguinte forma, primeiramente o conjunto de vértices é ordenado e um conjunto R para armazenar o tamanho das melhores soluções para cada subproblema é criado. Para cada subproblema criam-se seus conjuntos candidatos e considera-se que o vértice v faz parte da solução. Após a chamada da função *NOVOEXPAND*, $|A|$ contém o tamanho da melhor solução encontrada para o subproblema e esse valor pode ser armazenado em R.

Assim o *upper bound* é utilizado através de R, da seguinte forma, se $R[v]$ é a melhor solução possível com o vértice v, então ao selecionar v para uma nova solução, existe a possibilidade de adicionar no máximo $R[v]$ vértices a solução. Da mesma forma, se a melhor solução possível não ultrapassa o valor da melhor solução alcançada, o subproblema é podado.

4. Algoritmo Proposto

Nesta seção iremos mostrar o algoritmo *Branch and Bound* proposto neste trabalho. Utilizaremos da técnica de Partições em Clique e Bonecas Russas, conforme os Algoritmos 1 e 2 respectivamente. Além de adicionar uma solução para fraqueza de Partições em Cliques para grafos bipartidos.

Grafos bipartidos são livres de triângulo e consequentemente não possuem cliques de tamanho maior que 2, K_2. Para esta classe de grafo o Algoritmo 1 não seria efetivo, pois P_a e P_b seriam conjuntos independentes e suas próprias cardinalidades poderiam ser utilizadas como *bound*. Segundo [McCreesh e Prosser, 2014], encontrar um limite que possa ser utilizado neste caso não seria barato e nem trariam benefícios sem impactar no tempo do algoritmo.

Uma alternativa seria elaborar um algoritmo que substituísse o *CLIQUESORT* e que não afetaria no tempo computacional. Em [Zhou et al., 2018] é proposta uma solução para esse problema, onde um procedimento de *Upper Bound Propagation* (UBP) envolvendo cada vértice é produzido. Este procedimento se beneficia do fato de que o grafo de entrada é bipartido e utiliza alguns limites específicos para essa classe de grafos. Este procedimento é calculado uma única vez no pré-processamento do grafo.

5. Resultados Computacionais

Comparamos os resultados do algoritmo proposto com o algoritmo de [Silva, 2016], ambos utilizam Bonecas Russas. Avaliamos o desempenho do algoritmo utilizando as instâncias presentes em DIMACS, grupo de instâncias consolidadas na literatura. Comparamos também com o algoritmo desenvolvido por [McCreesh e Prosser, 2014], utilizando instâncias de grafos bipartidos presentes em KONECT e de grafos livre de triângulo presentes em DIMACS.

Os testes foram realizados utilizando um computador com 16GB de memoria RAM DDR4 2400Mhz, processador Intel(R) Pentium(R) CPU G4560 @ 3.50GHz, sistema operacional Linux com 64 bits, distro Linux Mint 19. O algoritmo foi implementado na linguagem C++ utilizando do recurso de paralelismo de bits.

A Tabela 1 representa a comparação com o algoritmo de [Silva, 2016]. A primeira coluna representa o nome da instância, para ambos algoritmos são representados o tempo de execução em segundos e a quantidade de subproblemas necessários para resolvê-lo. Para cada instância foi destacado o algoritmo que obteve melhor resultado em relação a tempo de execução.

Analisando as instâncias na Tabela 1, temos que o algoritmo implementado foi melhor que o algoritmo de [Silva, 2016] em 25 delas, nas demais os resultados foram muito próximos.

Tabela 1: Resultados para instâncias DIMACS de grafos aleatórios.

	Algoritmo de SILVA		Algoritmo Proposto	
instância	**tempo(s)**	**#subproblemas**	**tempo(s)**	**#subproblemas**
C250.9	0.064	229737	**0.012**	42020
C500.9	0.726	1329995	**0.231**	434024
DSJC500_5	380.475	744285913	**124.395**	470704593
MANN_a45	0.998	1000157	**0.107**	284687
MANN_a81	32.182	10598405	**3.201**	2985362
brock200_1	0.240	1067562	**0.061**	281049
brock200_4	0.339	1501065	**0.107**	504173
c-fat200-5	**0.003**	13031	0.004	94008
c-fat500-10	**0.025**	64553	0.060	1046263
c-fat500-2	0.005	9111	**0.003**	54465
c-fat500-5	**0.009**	25338	0.016	280974
gen200_p0.9_44	**0.002**	9067	0.003	11402
gen200_p0.9_55	0.046	195825	**0.005**	19253
gen400_p0.9_55	0.696	1689192	**0.022**	33304
gen400_p0.9_65	0.770	1748479	**0.025**	38948
hamming10-2	36.394	25881989	**0.028**	1035
hamming6-2	0.003	28821	**0.000**	71
hamming6-4	0.001	14569	**0.001**	10170
hamming8-2	0.326	977253	**0.001**	265
hamming8-4	878.184	2794267681	**0.081**	152638
johnson16-2-4	**0.538**	3847179	1.002	15524907
keller4	2.079	11523829	**0.133**	564270
p_hat1000-3	3023.770	5880195360	**1925.381**	3615283080
p_hat300-3	1.155	3073777	**0.359**	1219481
p_hat500-3	38.351	62208669	**11.660**	36776745
p_hat700-3	154.138	170859513	**95.594**	198647061
san1000	**0.174**	659897	0.398	143938
san200_0.9_2	0.043	201574	**0.004**	13962
san400_0.7_1	5.232	12163775	**0.088**	113194
sanr200_0.9	0.052	226344	**0.005**	20797
sanr400_0.7	27.491	61393975	**4.222**	11855063

O algoritmo implementado é mais rápido que o algoritmo de comparação para grande maioria das instâncias, sendo até dez vezes mais rápido na instância **san200_0.9_2**.

A Tabela 2, representa uma comparação do algoritmo proposto por [McCreesh e Prosser, 2014] com o algoritmo proposto neste trabalho. A técnica de Partições em Cliques foi substituída

pela técnica de *Upper Bound Propagation*(UBP) proposta em [Zhou et al., 2018], tal técnica foi utilizada buscando fortalecer o algoritmo para grafos bipartidos.

Observando os resultados da Tabela 2, o algoritmo proposto foi superior a o algoritmo de [McCreesh e Prosser, 2014] na maioria das instâncias. Temos que embora o algoritmo proposto tenha quantidade de subproblemas superiores, ele consegue resolvê-los em tempo inferior a [McCreesh e Prosser, 2014]. Isso é consequência de que no algoritmo de [McCreesh e Prosser, 2014] a função *CLIQUESORT* é invocada em cada ramo, enquanto no algoritmo implementado utilizando procedimento UBP, é invocado uma única vez. Embora utilizar o *CLIQUESORT* em cada ramo aumente a precisão do *bound* e reduza a quantidade de subproblemas, o seu custo influência bastante no tempo computacional.

Tabela 2: Resultados para instâncias adaptadas de KONECT de grafos bipartidos e DIMACS.

	Algoritmo de MCCREESH		**Algoritmo Proposto + UBP**	
instância	**tempo(s)**	**#subproblemas**	**tempo(s)**	**#subproblemas**
edit-frwikinews	2.763	27094	**0.877**	414941
escorts	0.490	42213	**0.186**	173322
moreno_crime_crime	0.003	1031	**0.001**	3482
movielens-10m_ti	2.452	43745	**0.336**	74401
movielens-10m_ui	**0.913**	186258	1.839	2122300
movielens-10m_ut	1.873	73521	**1.000**	577917
myciel3	0.000	19	0.000	87
myciel4	0.000	192	0.000	1118
myciel5	**0.000**	841	0.002	3.812
myciel6	**0.001**	8963	0.003	19405
myciel7	0.006	56321	**0.001**	57230
opsahl-collaboration	**0.006**	56321	0.722	47005
opsahl-ucforum	**0.006**	18875	0.011	193266
unicodelang	0.003	799	**0.001**	4454
youtube-groupmemberships	1.943	66376	**0.575**	106913
dbpedia-writer	1.385	19403	**0.735**	26228

6. Conclusões e Trabalhos Futuros

Apresentamos um algoritmo *Branch and Bound* combinando e aperfeiçoando várias técnicas já aplicadas com sucesso na literatura. Os resultados mostram que as técnicas utilizadas no algoritmo provocam redução no número de subproblemas e consequentemente no tempo, sendo mais efetivo que o algoritmo de [Silva, 2016]. Os resultados mostram também que o algoritmo implementado é mais efetivo em relação a tempo computacional de que [McCreesh e Prosser, 2014] para grafos bipartidos.

Algumas pesquisas não realizadas neste trabalho assim como melhorias e passos podem ser acrescentados ao algoritmo. Alguns fatores que podem ser estudados são: Estudar técnicas de

ordenação de vértices baseada em coloração ou em outros critérios, tais técnicas podem resultar na obtenção de Partições de Cliques mais exatas e diminuir o número de bonecas; Desenvolver um algoritmo para biclique induzida desbalanceada, não necessariamente utilizando as mesmas técnicas que para a balanceada, embora esta esteja contida nesta classe; Estudar técnicas de paralelismo para adicionar ao algoritmo, além de paralelismo de bits.

Referências

Al-Yamani, A. A., Ramsundar, S., e Pradhan, D. K. (2007). A defect tolerance scheme for nanotechnology circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(11): 2402–2409.

Araujo Tavares, W. (2016). Algoritmos exatos para problema da clique maxima ponderada. *UniversitÃ© D'Avignon*.

Cheng, Y. e Church, G. (2000). Biclstering of expression data, in proceedings of the eighth international conference on intelligent systems for molecular biology (ismb).

Corrêa, R. C., Michelon, P., Cun, B. L., Mautor, T., e Donne, D. D. (2014). A bit-parallel russian dolls search for a maximum cardinality clique in a graph. *arXiv preprint arXiv:1407.1209*.

Gary, M. R. e Johnson, D. S. (1979). Computers and intractability: A guide to the theory of np-completeness.

McCreesh, C. e Prosser, P. (2014). An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, p. 226–234. Springer.

Ravi, S. e Lloyd, E. L. (1988). The complexity of near-optimal programmable logic array folding. *SIAM Journal on Computing*, 17(4):696–710.

Silva, A. M. (2016). Um algoritmo exato para o problema da biclique induzida balanceada máxima. *Monografia (Graduação em Sistemas de Informação), Campus Quixadá, Universidade Federal do Ceará*.

Vaskelainen, V. et al. (2010). Russian doll search algorithms for discrete optimization problems. *Aalto-yliopiston teknillinen korkeakoulu*.

Verfaillie, G., Lemaître, M., e Schiex, T. (1996). Russian doll search for solving constraint optimization problems. In *AAAI/IAAI, Vol. 1*, p. 181–187.

Zhou, Y., Rossi, A., e Hao, J.-K. (2018). Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. *European Journal of Operational Research*, 269(3): 834–843.