UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE CRATEÚS

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCUS VINICIUS MARTINS MELO

ALGORITMOS EXATOS PARA O PROBLEMA DA BICLIQUE INDUZIDA BALANCEADA MÁXIMA

CRATEÚS

2019

MARCUS VINICIUS MARTINS MELO

ALGORITMOS EXATOS PARA O PROBLEMA DA BICLIQUE INDUZIDA BALANCEADA MÁXIMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de graduação em Ciência da Computação.

Orientador: Prof. Dr. Rennan Ferreira Dantas

Coorientador: Prof. Msc. Luiz Alberto do Carmo Viana

CRATEÚS

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M486a Melo, Marcus Vinicius Martins.
Algoritmos exatos para o problema da biclique induzida balanceada máxima / Marcus Vinicius Martins Melo. – 2019.
50 f.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Ciência da Computação, Crateús, 2019.
Orientação: Prof. Dr. Rennan Ferreira Dantas.
Coorientação: Prof. Me. Luiz Alberto do Carmo Viana.

1. Biclique Induzida Balanceada. 2. Bonecas Russas. 3. Partições em Cliques. 4. Branch and Bound. I. Título.

CDD 004

MARCUS VINICIUS MARTINS MELO

ALGORITMOS EXATOS PARA O PROBLEMA DA BICLIQUE INDUZIDA BALANCEADA MÁXIMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de graduação em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Rennan Ferreira Dantas (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Msc. Luiz Alberto do Carmo Viana (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Pablo Luiz Braga Soares
Universidade Federal do Ceará (UFC)

Msc. Mardson Da Silva Ferreira
Universidade Federal do Ceará (UFC)

Prof. Msc. Jefferson Lourenço Gurguri
Instituto Federal do Ceará (IFCE)

A minha família que não mediu esforços, mesmo com dificuldades para que eu chegasse a esta etapa da minha vida.

AGRADECIMENTOS

Primeiramente quero agradecer a Deus por ter me dado forças para seguir em frente nos momentos mais difíceis durante esta jornada.

Ao Prof. Msc. Rennan Ferreira Dantas por me orientar e por ter paciência em me ensinar, pela dedicação e disponibilidade. Professor excepcional, a quem devo grande parte de minha evolução acadêmica e conquistas, sem ele este trabalho não seria possível.

Ao Prof. Msc. Luiz Alberto do Carmo Viana por me coorientar, pela excelente pessoa, professor excepcional. Pelas reuniões primorosas e contribuições fundamentais para o desenvolvimento deste trabalho.

Aos membros participantes da banca por terem aceito o convite e pelas suas contribuições e sugestões fundamentais para qualidade do trabalho.

A Profa. Msc. Lisieux Marie Marinho dos Santos Andrade pelo acompanhamento e sugestões durante as aulas de PPCT, foram auxílios valiosos e que enriqueceram mais a qualidade do trabalho.

Aos amigos e colegas de turma por me proporcionarem grande aprendizado e que durante a graduação dividiram comigo muitas noites em claro, muitos risos, dias de lutas e de sofrimento. Quero destacar em especial: Wermeson Rocha, Davi Barros, Mardônio Vieira, Ícaro Mourão, Tiago Rocha e principalmente João Paulo de Araújo.

Por fim, a todos que participaram de forma direta ou indireta da minha formação durante a graduação. Seja de forma positiva ou negativa, mas que me ajudaram a ser a pessoa que sou hoje.

"... são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar."

(Alan Turing)

RESUMO

O presente trabalho apresenta dois algoritmos Branch and Bound (B&B) para o Problema da Biclique Induzida Balanceada Máxima (PBIBM). Ambos os algoritmos combinam de forma eficiente técnicas já empregadas com sucesso na literatura para resolver o problema. O primeiro algoritmo, denominado ALGBRPC, combina as técnicas de Partições em Cliques e Bonecas Russas. A estratégia do algoritmo é utilizar Partições em Cliques para podar os subproblemas gerados por cada boneca. O segundo algoritmo, denominado ALGBRUBP, combina a técnica de Bonecas Russas com um procedimento de *Upper Bound Propagation* (UBP). Este algoritmo foi desenvolvido especificamente para a classe de grafos bipartidos, visto que Partições em Cliques é ineficaz para esta classe. Por fim, testes computacionais foram realizados comparando os dois algoritmos propostos com os algoritmos base, utilizando conjuntos de instâncias disponíveis na literatura e de grafos gerados de forma aleatória. O primeiro algoritmo supera o algoritmo base na grande maioria das instâncias, tanto no número de subproblemas quanto no tempo computacional demandado. O segundo algoritmo supera o algoritmo base em algumas instâncias de grafos bipartidos.

Palavras-chave: Biclique Balanceada Máxima. Bonecas Russas. Partições em Cliques. Branch and Bound

ABSTRACT

The present work introduce two Branch and Bound (B&B) algorithms for the Maximum Balanced Induced Biclique Problem (PBIBM). Both algorithms efficiently combine techniques already successfully used in the literature to solve the problem. The first algorithm, called ALGBRPC, combines the techniques of Clique Cover and Russian Dolls. The strategy of the algorithm is to use Clique Cover to prune subproblems generated by each doll. The second algorithm, called ALGBRUBP, combines the tecniques of Russian Dolls and Upper Bound Propagation (UBP). This algorithm was developed specifically for the bipartite class of graphs, since Clique Cover is ineffective for this class. Finally, computational tests were performed comparing the two proposed algorithms with the base algorithms, using sets of instances available in the literature and randomly generated graphs. The first algorithm outperforms the base algorithm in the vast majority of instances, both in terms of the number of subproblems or in terms of computational time demanded. The second algorithm outperforms the base algorithm in some instances of bipartite graphs.

Keywords: Balanced Induced Biclique. Russian Dolls. Clique Cover. Branch and Bound

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALGBRPC	Algoritmo de Bonecas Russas e Partições em Cliques
ALGBRUBP	Algoritmo de Bonecas Russas e *Upper Bound Propagation* (UBP)
B&B	Branch and Bound
PBIBM	Problema da Biclique Induzida Balanceada Máxima
PD	Programação Dinâmica
PL	Programação Linear
PLI	Programação Linear Inteira
UBP	*Upper Bound Propagation*

SUMÁRIO

1 INTRODUÇÃO

A Teoria dos Grafos é uma área da matemática e da ciência da computação que proporcionou a elaboração de vários problemas, alguns destes vêm sendo estudados há bastante tempo na literatura. Um desses é o problema de encontrar uma Biclique Induzida Balanceada Máxima em um grafo arbitrário. Este problema pertence a classe de complexidade NP-Difícil, como provado em Gary e Johnson (1979), e portanto ainda não se conhece na literatura um algoritmo polinomial para resolvê-lo.

Uma biclique é uma estrutura formada por um par de subconjuntos disjuntos $\{A, B\}$, onde existe uma aresta ligando todo vértice de A a todo vértice de B. A biclique é induzida se não existem arestas entre vértices do mesmo conjunto e a biclique é balanceada se A e B tem a mesma quantidade de vértices. O Problema da Biclique Induzida Balanceada Máxima (PBIBM) é encontrar uma biclique induzida balanceada de tamanho máximo em um grafo arbitrário.

Várias aplicações podem ser modeladas utilizando bicliques em diversas áreas, como: projeto de sistemas nanoelétricos (AL-YAMANI *et al.*, 2007), bi-clusterização de dados de expressão gênica em biologia computacional (CHENG; CHURCH, 2000) e PLA-*folding* na teoria VLSI (RAVI; LLOYD, 1988). Um outro exemplo de aplicação seria sistemas de recomendação, onde pode-se estabelecer uma relação entre um conjunto de usuários e um conjunto de produtos. Pode-se representar essa relação por uma biclique, o conjunto de usuários representado por A e o conjunto de produtos por B, assim existe uma aresta entre um usuário e um produto se o usuário avaliou positivamente o produto. Poderia-se extrair a partir desta relação informações para prever os próximos produtos que o usuário pode gostar.

Por se tratar de um problema NP-Difícil, o PBIBM é intratável do ponto de vista computacional. Segundo Silva *et al.* (2007), 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; 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 baseia-se em (MCCREESH; PROSSER, 2014), embora não utilize Partições em Cliques para podar os subproblemas e sim cortes via método das Bonecas Russas. Este será o algoritmo base para este trabalho.

Em (ZHOU *et al.*, 2018), um algoritmo para o PBIBM é proposto especialmente para a classe de grafos bipartidos. Este algoritmo utiliza um procedimento de propagação de *upper bound* para encontrar um limite para cada vértice do grafo. Este trabalho baseia-se no algoritmo apresentado em (MCCREESH; PROSSER, 2014), embora não utilize Partições em Cliques pela sua ineficiência para grafos bipartidos.

Neste trabalho, apresentamos dois algoritmos exatos para o PBIBM. No primeiro algoritmo, denominado ALGBRPC, foram combinadas as técnicas de Bonecas Russas e Partições em Cliques. No segundo, denominado ALGBRUBP, foi aplicada a técnica de Bonecas Russas e acrescentado o procedimento proposto por (ZHOU *et al.*, 2018) para grafos bipartidos. Foram apresentadas comparações dos algoritmos desenvolvidos com os algoritmos base, o primeiro comparado com (SILVA, 2016) e o segundo com (MCCREESH; PROSSER, 2014) para grafos bipartidos. Este trabalho buscou contribuir com a comunidade acadêmica por meio da expansão de conhecimento acerca do PBIBM e formas de resolução do mesmo. Além de demonstrar o funcionamento de tais técnicas e como utilizá-las, o presente trabalho explana brevemente como elas podem ser aplicadas para outros problemas.

O presente trabalho está dividido em cinco capítulos, além desta introdução. No Capítulo 2, será apresentado as notações e definições necessárias para o entendimento do trabalho. Nele serão abordados alguns conceitos de Teoria dos Grafos, NP-Completude e Otimização Combinatória. No Capítulo 3, será apresentado uma revisão sobre os principais trabalhos presentes na literatura relacionados ao presente estudo. No Capítulo 4, será apresentado a estrutura geral dos algoritmos propostos neste trabalho e os métodos utilizados. No Capítulo 5, será apresentado o ambiente de testes e os resultados de comparação com os algoritmos bases. No Capítulo 6, serão apresentados as principais conclusões desse trabalho e alguns aprimoramentos que podem ser seguidos em trabalhos futuros.

1.1 Objetivos

Nesta seção serão mostrados os objetivos deste trabalho, que serão divididos em Objetivo geral e Objetivos específicos.

1.1.1 Objetivo geral

Aplicar a técnica de Bonecas Russas combinada com a técnica de Partições em Cliques para elaborar um algoritmo exato para o Problema da Biclique Induzida Balanceada Máxima (PBIBM).

1.1.2 Objetivos específicos

- Demonstrar a utilização das técnicas que foram aplicadas no PBIBM, como Bonecas Russas, Branch and Bound e Partições em Cliques.
- Identificar novas técnicas e métodos que possam contribuir para a resolução do PBIBM. Como uma nova ordenação inicial de vértices que possa facilitar na obtenção de novas cliques.
- Analisar os resultados obtidos e comparar com o algoritmo base deste trabalho.

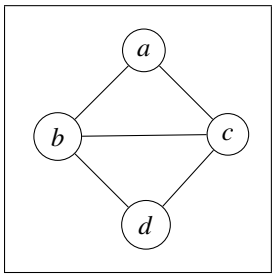
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão definidos alguns conceitos e notações importantes para o entendimento deste trabalho. Na Seção 2.1, serão apresentados alguns conceitos de grafos, a estrutura principal estudada nesta monografia. Na Seção 2.2, será apresentado resumidamente NP-Completude e, na Seção 2.3, alguns conceitos de Otimização Combinatória, incluindo Programação Linear (PL) e Programação Linear Inteira (PLI). Na Seção 2.3, também serão apresentadas algumas técnicas como *Branch and Bound* (B&B) e Bonecas Russas, que serão cruciais para o desenvolvimento deste trabalho. Na Seção 2.4, será abordado sobre a técnica de paralelismo de bits.

2.1 Grafos

Um grafo simples $G = (V,E)$ é uma estrutura formada por um conjunto finito não vazio de vértices $V = \{v_1, \ldots, v_n\}$ e um conjunto finito de arestas $E \subseteq \{\{u,v\} : u,v \in V \text{ e } u \neq v\}$. Vértices podem ser chamados de pontos ou nós e arestas de linhas ou curvas. Denotam-se por $E(G)$ e $V(G)$ o conjunto de arestas e vértices do grafo G, respectivamente. Na Figura 1, tem-se que V(G)=$\{a,b,c,d\}$ e E(G)=$\{\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{c,d\}\}$. Neste trabalho será utilizado grafos simples e não direcionados, ou seja, a aresta $\{a,b\}$ é igual a aresta $\{b,a\}$.

Figura 1 – Grafo G



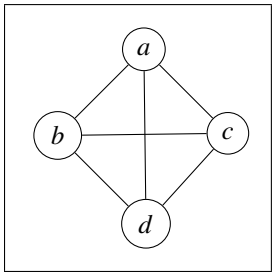
Fonte: Próprio autor.

Para cada aresta $\{u,v\} \in E(G)$, dizemos que u e v são vértices adjacentes e são as extremidades dessa aresta. Na Figura 1, os vértices a e b são adjacentes. A vizinhança de um vértice v é denotada por $N_G(v) = \{u \in V : \{u,v\} \in E\}$, assim denota-se por $\overline{N_G(v)} = V(G) \setminus N_G(v)$ os conjunto de vértices não-vizinhos a v. O grau de um vértice v de G é denotado por $deg(v) = |N_G(v)|$. A cardinalidade de um grafo G, denotada por $|G|$, é a quantidade de vértices deste grafo.

Uma clique é um grafo onde todos os seus vértices são adjacentes entre si. 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 induz uma clique. O grafo G da Figura 2 é uma clique.

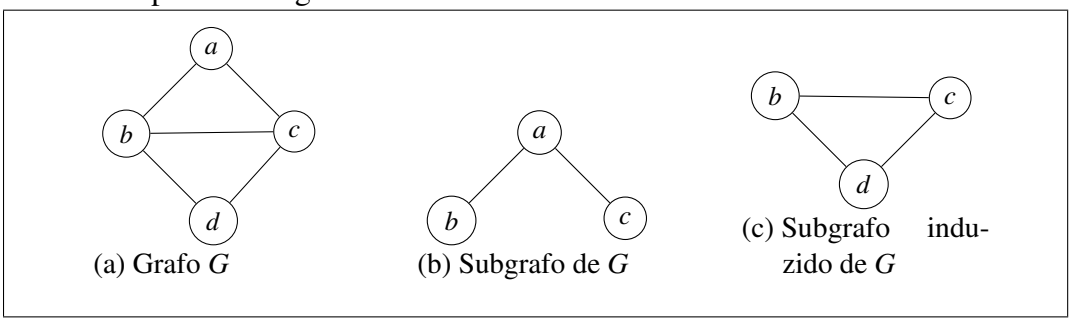
Figura 2 – Um grafo G de cardinalidade 4



Fonte: Próprio autor.

Um grafo S é dito subgrafo de G se $V(S) \subseteq V(G)$ e $E(S) \subseteq E(G)$. Um subgrafo $H = (U,T)$ de G é dito subgrafo induzido de G, denotado por $G[H]$, se $U \subseteq V(G)$ e $T = \{e \in E(G) : |e \cap U| = 2\}$, ou seja, H tem todas as arestas de G com ambos os extremos em U. Na Figura 3c, temos um subgrafo induzido de G. Note que na Figura 3b temos um subgrafo de G, embora não seja induzido, pois não possui a aresta $e = \{b,c\}$.

Figura 3 – Exemplos de subgrafos



(a) Grafo G (b) Subgrafo de G (c) Subgrafo induzido de G

Fonte: Próprio autor.

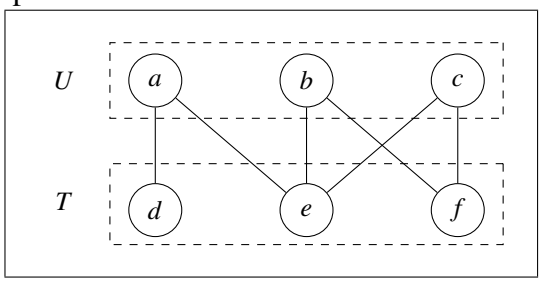
Um subconjunto de vértices S de um grafo G é chamado de conjunto independente, se para todo par de vértices $u, v \in S$ não existe a aresta $\{u,v\} \in E(G)$. Na Figura 3a, $S = \{a,d\}$ é um subconjunto de vértices de G que induz um conjunto independente.

Um grafo G é dito triângulo, se $V(G) = \{a,b,c\}$ e $E(G) = \{\{a,b\},\{b,c\},\{c,a\}\}$. Na Figura 3a, temos um triângulo induzido pelo subconjunto de vértices $\{a,b,c\}$. Um grafo triângulo também é uma clique de tamanho 3. Um grafo G é chamado de livre de triângulo se para qualquer subconjunto $S \subseteq V(G)$ com $|S| = 3$, $G[S]$ não é um grafo triângulo. Na Figura 3b,

temos um grafo livre de triângulo.

Um grafo $G = (U \cup T, E)$ é dito bipartido se toda aresta tem uma extremidade em U e outra em T, e portanto, U e T são dois conjuntos independentes. Na Figura 4, temos um grafo bipartido com $U = \{a,b,c\}$ e $T = \{d,e,f\}$. Um grafo $G = (V,E)$ é dito k-partido se V pode ser particionado em k conjuntos independentes. Se k = 2 então ele pode ser chamado de grafo 2-partido ou bipartido.

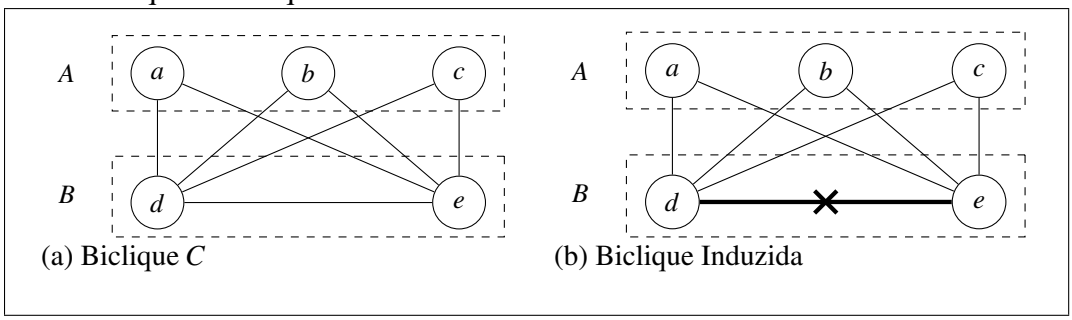
Figura 4 – Grafo Bipartido G



Fonte: Próprio autor.

Seja G um grafo qualquer. Dizemos que um par de subconjuntos disjuntos A e B de $V(G)$ é chamada biclique C de G, se existe uma aresta $\{u,v\} \in E(G)$, para todo $u \in A$ e $v \in B$ (SILVA, 2016). Na Figura 5a, temos uma biclique com $A = \{a,b,c\}$ e $B = \{d,e\}$. Se A e B são conjuntos independentes, a biclique é dita induzida. Na Figura 5a, tem-se uma biclique não induzida, pois a aresta $\{d,e\} \in B$. Na Figura 5b, tem-se uma biclique induzida com a remoção da aresta $\{d,e\}$.

Figura 5 – Biclique e Biclique Induzida

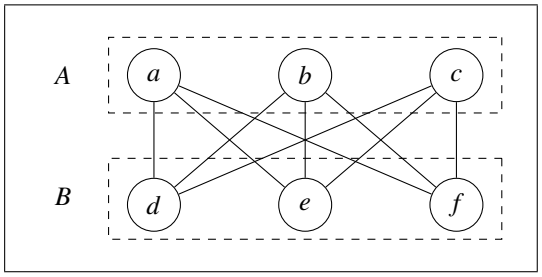


(a) Biclique C (b) Biclique Induzida

Fonte: Próprio autor.

Uma biclique é dita balanceada se $|A| = |B|$. Na Figura 6, temos uma biclique induzida balanceada, pois tanto $A = \{a,b,c\}$ quanto $B = \{d,e,f\}$ têm a mesma cardinalidade e são conjuntos independentes. Uma biclique C de G é dita balanceada máxima se não existe uma biclique balanceada $C^{'}$ em G, tal que $|C^{'}| > |C|$.

Figura 6 – Biclique Induzida Balanceada



Fonte: Próprio autor.

Diversos problemas que envolvem grafos são difíceis de resolver computacionalmente (CHEESEMAN *et al.*, 1991). Como mencionado anteriormente, o problema de encontrar uma Biclique Induzida Balanceada Máxima em um grafo arbitrário é um problema que pertence a classe de complexidade NP-Difícil. Na próxima subseção, esta e outras classes de complexidade serão abordadas.

2.2 NP-Completude

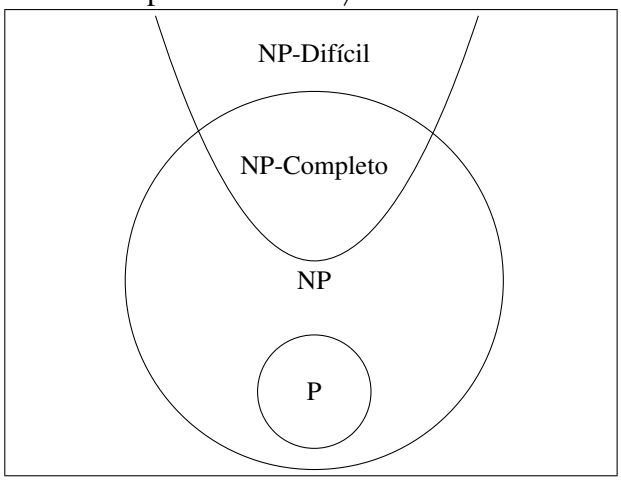
Uma classe de complexidade pode ser definida como um conjunto de problemas que podem ser resolvidos dentro de um determinado limite de recurso computacional, seja ele tempo ou espaço (PAPADIMITRIOU, 2003). Um conceito muito importante com relação as classes de complexidade é o conceito de redução. Um problema A se reduz polinomialmente a um problema B, denotado por $A \leq_p B$, se existe uma função computada em tempo polinomial que transforma qualquer entrada do problema A em alguma entrada do problema B (CORMEN *et al.*, 2009). Um outro conceito importante é o de certificado, um certificado é um candidato a solução de uma instância de um problema (CORMEN *et al.*, 2009).

A classe de complexidade P engloba o conjunto de problemas de decisão que podem ser resolvidos em tempo polinomial, isto é, o consumo de tempo para resolver o problema é limitado por um polinômio (PAPADIMITRIOU, 2003). A classe de complexidade P é conhecida por ser a classe dos problemas que podem ser resolvidos de forma eficiente. A classe de complexidade NP engloba todos os problemas de decisão que podem ser verificados em tempo polinomial, isto é, para um determinado certificado o tempo para verificar se ele é válido é polinomial (CORMEN *et al.*, 2009).

A classe de complexidade NP-Difícil engloba o conjunto de problemas que são tão difíceis de resolver quanto qualquer outro problema em NP (PAPADIMITRIOU, 2003). Dado um problema B, $B \in$ NP-Difícil se e somente se $A \leq_p B$, para todo $A \in$ NP. Um problema é

NP-Completo, se ele é NP-Difícil e está em NP. Uma relação entre as classes de complexidade P, NP, NP-Difícil e NP-Completo é apresentada na Figura 7.

Figura 7 – Classes de Complexidade se $P \neq NP$.



Fonte: Próprio autor.

Além de ser um problema NP-Difícil, o PBIBM é um problema de otimização, pois o objetivo principal é encontrar a maior biclique induzida balanceada em um grafo arbitrário. Na próxima seção serão mostrados e exemplificados alguns conceitos e ramos da Otimização Combinatória, bem como algumas técnicas de resolução de problemas dessa área.

2.3 Otimização Combinatória

Em geral, os problemas de otimização dividem-se em duas categorias distintas: os problemas com variáveis contínuas e os problemas com variáveis discretas (PAPADIMITRIOU; STEIGLITZ, 1998). São dois tipos de problemas diferentes, enquanto os problemas contínuos buscam por um conjunto de números reais, os discretos buscam por um inteiro e/ou conjunto enumerável (PAPADIMITRIOU; STEIGLITZ, 1998). Na próxima seção serão abordadas algumas definições importantes para se iniciar o estudo desses problemas.

2.3.1 Definições e notações

Um problema linear pode ser definido como um conjunto de coeficientes $c_1, c_2, \ldots, c_n$, um conjunto de variáveis $x_1, x_2, \ldots, x_n$, uma função objetivo $c_1x_1 + c_2x_2 + \cdots + c_nx_n$ e um conjunto de restrições de igualdades ou desigualdades, como $a_{11}x_1 + ... + a_{1n}x_n \leq b_1$. As restrições impõem às variáveis limites viáveis a serem atingidos (BAZARAA *et al.*, 2011). A função objetivo é a finalidade principal do problema, ela pode ser de maximização ou minimização. Na

forma canônica do problema linear a função objetivo é sempre de maximização e as restrições são de desigualdade da forma "$\leq$". Abaixo temos um problema linear com uma função objetivo de maximização:

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + ... + c_nx_n \\ \text{s.a.} \quad & a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2 \\ & \dots \\ & a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m \\ & x_1, x_2, \dots, x_n \geq 0 \end{aligned} \tag{2.1}$$

A inequação $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, representa a i-ésima restrição do problema. Os coeficientes a_{ij} formam a matriz de restrições $\boldsymbol{A}$. Os elementos b_i representam os requisitos máximos do problema e as restrições $x_1, x_2, \dots, x_n \geq 0$ representam as restrições de não negatividade do problema. O conjunto de valores $x_1, \dots, x_n$ satisfazendo todas as restrições representam os pontos viáveis ou soluções viáveis.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Um problema linear pode ser representado utilizando matrizes. Se considerarmos $(c_1, c_2, \dots, c_n)^T$ como $c \in \mathbb{R}^{n\times 1}$, $(x_1, x_2, \dots, x_n)^T$ como $x \in \mathbb{R}^{n\times 1}$, $(b_1, b_2, \dots, b_m)^T$ como $b \in \mathbb{R}^{m\times 1}$ e $A \in \mathbb{R}^{m\times n}$. O problema pode ser representado da seguinte forma:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.a.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{2.2}$$

Um poliedro é a interseção de um número finito de semi-espaços. Os semi-espaços podem ser representados por uma inequação $a_i x \leq b_i$, cada semi-espaço representa uma restrição. Desta forma, o poliedro pode ser representado por um sistema $a_i x \leq b_i$ para $i = 1, 2, \dots, m$ que

é equivalente a representar o poliedro como $P = \{x \mid Ax \leq b\}$ (BAZARAA *et al.*, 2011). O poliedro representa a região com todas as soluções possíveis para o problema que respeitam as restrições. Para um problema de maximização, dizemos que x' é uma solução ótima se, $x' \in P$ e $c^T x' \geq c^T x'', \forall x'' \in P$. Para um problema de minimização, basta inverter a desigualdade para $c^T x' \leq c^T x''$.

Um conjunto $X \subseteq \mathbb{R}^n$ é chamado de convexo se para todo $x, y \in X$ e $\lambda \in [0,1]$, $\lambda x + (1-\lambda)y \in X$ (BERNHARD; VYGEN, 2008). O fecho convexo de um conjunto X, denotado por $conv(X)$, é definido como o conjunto de todas as combinações convexas de pontos em X (BERNHARD; VYGEN, 2008). Como o poliedro é convexo então um ótimo local também é ótimo global e como a função objetivo é linear então todo ponto ótimo está nas faces do poliedro.

2.3.2 Programação Linear

A Programação Linear (PL) foi elaborada pelo matemático George B. Dantzig em 1947. A PL lida com problemas de otimização, sejam eles de minimizar ou maximizar uma função, satisfazendo um conjunto de restrições impostas sobre as variáveis da função (BAZARAA *et al.*, 2011).

Seja um problema linear de maximização como $max\{c^T x : Ax \leq b\}$. Uma solução viável para esse problema seria um vetor x satisfazendo $Ax \leq b$. Se essa solução fosse a máxima, ela seria uma solução ótima do problema (BERNHARD; VYGEN, 2008). Quando um problema em PL não tem solução, podem-se ter duas possibilidades. A primeira delas, quando o problema é inviável, ou seja, o poliedro é vazio, isto é, não existem soluções que satisfaçam as restrições. A segunda é quando o problema é ilimitado, ou seja, (para $\alpha \in \mathbb{R} : x \in P, c^T x > \alpha$).

Podem-se formular muitos problemas de otimização com PL. Dantzig elaborou em 1949, um método que viria a ser muito utilizado, servindo de base para alguns outros métodos de resolução de programação linear, chamado Método Simplex, um dos algoritmos mais conhecidos e mais antigos na Programação Linear (BERNHARD; VYGEN, 2008).

2.3.2.1 Método Simplex

O Método Simplex pode ser utilizado para resolver problemas de programação linear diversos. Se existe uma solução ótima para o problema, então existe um ponto de extremo no poliedro também e esses pontos são soluções viáveis para o problema (BAZARAA *et al.*, 2011).

O Método Simplex consegue melhorar as soluções até que um ponto ótimo seja alcançado, ou até concluir que o problema é ilimitado (BAZARAA *et al.*, 2011). Tome o problema a seguir:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.a} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.3}$$

O Método Simplex permite encontrar uma solução ideal para um problema de programação linear em um número finito de passos. O Simplex recebe como entrada uma matriz $A \in \mathbb{R}^{m \times n}$, vetores coluna $b \in \mathbb{R}^{m \times 1}$ e $c \in \mathbb{R}^{n \times 1}$, um ponto inicial viável $\bar{x}$ de $P = \{\bar{x} \in \mathbb{R}^{n \times 1} : A\bar{x} \leq b\}$ e tem como saída um ponto x' de P satisfazendo $\max\{c^T x' : x' \in P\}$ como solução final (PAPADIMITRIOU; STEIGLITZ, 1998).

Em cada passo o Simplex se desloca de um ponto atual até um ponto extremo vizinho que não tenha um valor objetivo menor que o atual. O Simplex para sua execução quando ele encontra um ponto que não possua um vizinho com valor objetivo maior, este ponto é um ótimo local e como o poliedro é convexo, um ótimo global.

2.3.3 Programação Linear Inteira

A Programação Linear Inteira (PLI) foi criada partindo do fato de que para determinados problemas os resultados fracionários obtidos com a aplicação de PL não eram ideais. PLI utiliza variáveis inteiras enquanto PL utiliza variáveis fracionárias. Um problema com PLI seria que não há garantia que exista um ponto ótimo inteiro no poliedro, mesmo que haja um ponto ótimo fracionário (PAPADIMITRIOU; STEIGLITZ, 1998).

Todos os problemas de otimização combinatória podem ser formulados com PLI. Em geral, PLI é muito mais difícil de resolver computacionalmente do que PL. Um conjunto de soluções viáveis pode ser definido como $\{x \in \mathbb{Z}^n : Ax \leq b\}$ para qualquer matriz A e vetor b (BERNHARD; VYGEN, 2008).

Seja um poliedro $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ e seja $P_I = \{x \in P, x \in \mathbb{Z}^n\}$, denotamos por $conv(P_I)$ como o fecho convexo dos pontos inteiros em P(BERNHARD; VYGEN, 2008). Não é difícil de perceber que $conv(P_I) \subseteq P$.

Geralmente, algoritmos para resolver PLI são divididos em duas categorias. A primeira é a categoria dos algoritmos de planos de corte, que utilizam do método simplex. A

segunda é a dos algoritmos enumerativos, que são baseados na enumeração inteligente de todos as soluções possíveis, como será mostrado a seguir.

2.3.3.1 Branch and Bound

Branch and Bound(B&B) é um método baseado na ideia de enumerar todas as soluções candidatas à solução ótima inteira do problema (PAPADIMITRIOU; STEIGLITZ, 1998). B&B é o método de resolução mais comum a ser utilizada quando se trata de problemas da classe computacional NP-Difícil. O termo Branch se refere a dividir o problema em ramos/partições, e Bound se refere ao limite que é calculado sobre os problemas ao longo da enumeração (PAPADIMITRIOU; STEIGLITZ, 1998). Com a utilização dos *bounds*, espera-se que apenas uma pequena parte das soluções seja examinada.

O B&B pode ser utilizado para resolver problemas de programação linear inteira como, por exemplo, problemas quadráticos binários (NEMHAUSER; WOLSEY, 1988). Assim, cada ramificação se limita a valorar uma variável com 1 ou 0 (BERNHARD; VYGEN, 2008). No pior caso o B&B enumera todas as soluções possíveis. A eficiência do algoritmo não está apenas em escolher o Branch e o Bound, algumas estratégias podem ser utilizadas em conjunto, como por exemplo, heurísticas (BERNHARD; VYGEN, 2008). Segundo (BLUM *et al.*, 2008, p. 2-4) as heurísticas são procedimentos projetados para solucionar problemas sem a necessidade de utilizar grande quantidade de recurso computacional, porém sem garantir a otimalidade de seus resultados, mas buscando soluções próximas a solução ótima.

Algumas utilizações desta técnica podem ser vistas em Balas e Xue (1996), que utiliza B&B juntamente com um procedimento para definir o conjunto de ramificação de vértices para resolver o problema da clique máxima. Temos também alguns objetivos distintos em relação a utilização desta técnica. Enquanto em Carraghan e Pardalos (1990) o objetivo principal está em reduzir o conjunto de ramificações gerado, em Babel (1994), o objetivo principal é reduzir o limite superior e por consequência aumentar a capacidade de poda (TAVARES, 2016). Segundo (TAVARES, 2016, p. 61), a maior parte dos algoritmos falha em não conciliar esses dois objetivos. Na próxima seção será mostrado um método que pode ser utilizado em conjunto com B&B para resolver problemas inteiros.

2.3.3.2 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.

Segundo Tavares (2016), a eficiência desse método provém da utilização da resolução das bonecas menores, com o objetivo de diminuir a árvore de busca das bonecas maiores. A partir da utilização do método das Bonecas Russas, pode-se definir uma relação de dominação entre subproblemas. "Um subproblema S_1 domina um outro subproblema S_2 se as soluções de S_1 são sempre melhores que as soluções de S_2"(TAVARES, 2016).

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. Em Östergård (1999) é proposto uma implementação dessas estratégias, para resolver o problema da clique ponderada máxima. A enumeração das bonecas baseia-se em uma ordem inicial dos vértices, e uma boneca corresponde ao problema da clique ponderada máxima no subgrafo que inclui um vértice a mais que a boneca anterior. A estratégia utilizada para resolver cada boneca é um algoritmo de B&B.

A técnica de bonecas russas é similar a outra técnica muito utilizada em problemas que podem ser decompostos em subproblemas. A Programação Dinâmica (PD) é um método que pode ser utilizado para resolver problemas que não tem subproblemas independentes, isto é, quando os subproblemas compartilham subsubproblemas (CORMEN *et al.*, 2009). Utilizando PD pode-se resolver cada subproblema uma única vez e guardar o resultado em uma tabela, para evitar que toda vez que for preciso da resposta do subproblema, esta seja recalculada (CORMEN *et al.*, 2009).

Os métodos de PD e Bonecas Russas têm características em comum, mas diferem na maneira em que eles utilizam os resultados dos problemas já calculados. Bonecas Russas utiliza o resultado somente como *bound* durante a busca, enquanto PD combina os resultados para obter o resultado final (VERFAILLIE *et al.*, 1996).

2.4 Paralelismo de bits

O paralelismo de bits é uma técnica muita utilizada em algoritmos de grafos, como podem ser vistos em (TAVARES, 2016), (CORRÊA *et al.*, 2014) e (SEGUNDO *et al.*, 2007). Nestes trabalhos a aplicação desta técnica proporcionou bons resultados. Segundo McCreesh (2017) com *hardwares* cada vez mais modernos, explorar o paralelismo de bits é cada vez mais beneficial aos algoritmos.

Através desta técnica podemos utilizar de vetores de bits para representar os conjuntos de vértices. Assim as operações entre conjuntos se tornam mais baratas computacionalmente, como união e interseção, que se beneficiam de operações a nível de bit.

Também tem-se ganho com a redução do espaço de memória utilizado para representar os vértices. Ao utilizar um vetor de inteiros para representar um conjunto de N vértices, utilizando um inteiro para representar cada vértice, teríamos $64N$ *bits*, assumindo que um inteiro ocupa 64 *bits*. Ao utilizar um vetor de bits teríamos apenas N *bits*, um *bit* representando cada vértice.

Considere o conjunto $U = \{0,1,2,3,4,5,6,7\}$ e uma representação de palavra de 8 *bits*. Considere $A = \{0,2,3,4,6\}$ e $B = \{1,2,4,5,7\}$ como subconjuntos de U. Os subconjuntos podem ser representados a nível de bit por $A' = 10111010$ e $B' = 01101101$. Assim dado $S \subseteq U$ e S' a representação de S a nível de bit, $S'[i] = 1 \Leftrightarrow i \in S$. Operações de conjuntos podem ser utilizadas através de A' e B', como a interseção e a união.

Interseção entre os conjuntos $A \cap B = A' \& B'$:

A'	10111010	$\{0,2,3,4,6\}$
B'	01101101	$\{1,2,4,5,7\}$
	00101000	$\{2,4\}$

União entre os conjuntos $A \cup B = A' | B'$:

A'	10111010	$\{0,2,3,4,6\}$
B'	01101101	$\{1,2,4,5,7\}$
	11111111	$\{0,1,2,3,4,5,6,7\}$

Além dessas, também podemos ter operações de complemento e diferença. $\overline{A} = \neg A' = 01000101 = \{1,5,7\}$ e $A \setminus B = A' \& (\neg B') = 10010010 = \{0,3,6\}$.

3 TRABALHOS RELACIONADOS

Neste capítulo, serão mostrados alguns trabalhos que foram utilizados como base e referência para este trabalho, sendo abordado as técnicas utilizadas e os principais resultados. Na seção 3.1, são mostrados alguns trabalhos que utilizam da técnica de Branch and Bound (B&B), já na seção 3.2, são mostrados alguns trabalhos que utilizam Bonecas Russas e por fim na seção 3.3, será abordado um trabalho que utiliza de um procedimento de *Upper Bound Propagation* (UBP).

3.1 Algoritmos de Branch and Bound

Em Tomita e Seki (2003) um algoritmo baseado no método B&B é desenvolvido para encontrar uma clique máxima em um grafo arbitrário. O principal foco deste algoritmo está em reduzir o espaço de busca com baixa sobrecarga de problemas. Uma técnica de ordenação de vértice baseada em coloração aproximada é utilizada. Dada uma partição de vértices R, se é necessário k cores para colorir R, então a clique máxima de R deve ter tamanho menor ou igual a k. Embora não exista um algoritmo eficiente para coloração ótima, um algoritmo guloso é utilizado.

Como resultados o algoritmo desenvolvido em Tomita e Seki (2003) se mostrou superior aos algoritmos desenvolvidos anteriormente para grafos arbitrários, principalmente na redução do espaço de busca proporcionado pela técnica de coloração aproximada. Esta técnica mostrou-se muito eficiente quando usada em conjunto com B&B.

3.1.1 Algoritmo proposto em McCreesh e Prosser (2014)

Em McCreesh e Prosser (2014) é desenvolvido um algoritmo exato para o PBIBM. Este algoritmo utiliza-se da técnica de B&B e de um procedimento de quebra de simetria na expansão dos vértices. 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.

Uma ordem estática dos vértices é definida para particionar as cliques, esta ordem é fixa e evita a reordenação dos vértices. Um limite superior é usado baseado em Partições de Cliques e utiliza a estratégia vista em Tomita e Seki (2003). O Teorema 1 demonstra o limite.

Teorema 1. *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.*

Prova. *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. Contradição.*

Um algoritmo simples para o PBIBM utilizando Branch and Bound pode ser visto no Algoritmo 1. Este algoritmo alterna na construção de dois conjuntos independentes A e B que formam a biclique $\{A,B\}$. Neste algoritmo P_a representa todos os vértices candidatos a solução do conjunto A e P_b os candidatos ao conjunto B, inicialmente ambos são inicializados com todos os vértices do grafo (todos os candidatos possíveis).

Algoritmo 1 Algoritmo para o PBIBM

```
1:  função SIMPLEBICLIQUE(G = (V,E))
2:      (A_max, B_max) ← (∅, ∅)
3:      EXPAND(G, ∅, ∅, V(G), V(G), A_max, B_max)
4:      devolve (A_max, B_max)
5:  fim função
6:
7:  função EXPAND(G, A, B, P_a, P_b, A_max, B_max)
8:      para v ∈ P_a faça
9:          se |A| + |P_a| > |A_max| e |B| + |P_b| > |B_max| então
10:             A ← A ∪ {v}
11:             P_a ← P_a \ {v}
12:             P'_a ← P_a ∩ N_G(v)
13:             P'_b ← P_b ∩ N_G(v)
14:             se |A| = |B| e |A| > |A_max| então
15:                 (A_max, B_max) ← (A, B)
16:             fim se
17:             se P'_b ≠ ∅ então
18:                 EXPAND(G, B, A, P'_b, P'_a, B_max, A_max)
19:             fim se
20:             A ← A \ {v}
21:         fim se
22:     fim para
23: fim função
```

Note que na chamada recursiva da função *EXPAND*, um vértice v é escolhido de P_a e adicionado a A. Isso gera algumas implicações, um novo P'_a é construído removendo os elementos que são adjacentes a v em P_a, isto é necessário, já que A deve ser um conjunto independente.

Um novo P'_b também é construindo, porém desta vez removendo os vértices não adjacentes a v em P_b, já que todos vértices de A devem ser adjacente a todos de B. Se P'_b não for vazio, então podemos adicionar novos elementos a B, assim nós chamamos a função recursiva e alternamos entre os conjuntos A e B para adicionar os vértices nos dois lados em ordem alternada. Depois de considerar a situação em que o vértice v pertence ao conjunto A, o algoritmo considera a situação em que ele não pertence. Nesse caso o algoritmo continua o *loop* e seleciona outro vértice candidato.

O algoritmo mencionado acima é uma implementação bem simples, porém não é eficiente, uma vez que ele explora um espaço de busca muito grande. Isso é causado pelo *bound* que leva em consideração o tamanho absoluto dos conjuntos. Note também, que ao considerar todas as possíveis soluções com $v \in A$ e depois considerar com $v \notin A$, não impede que em algum momento, ao considerar um $v' \in A$ podemos por consequência considerar $v \in B$, isto poderia causar simetria na solução. No Algoritmo 2, também desenvolvido por McCreesh e Prosser (2014), podemos observar uma implementação que utiliza um *bound* aperfeiçoado, baseado em Partições de Cliques, além de uma estratégia para eliminar a simetria e uma ordenação inicial de vértices.

Primeiramente os vértices são ordenados baseado no grau, a fim de tentar melhorar a obtenção de cliques, na linha 3 o grafo é permutado . A chance de encontrar uma Partição de Cliques ótima aumenta se vértices de maior grau forem usados primeiro.

A função *CLIQUESORT* no Algoritmo 3, 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. Vale ressaltar que o *CLIQUESORT* é guloso.

Algoritmo 2 Algoritmo melhorado para o PBIBM

```
1: função IMPROVEDBICLIQUE(G = (V,E))
2:     (A_max, B_max) ← (∅,∅)
3:     ordena de forma decrescente os vértices de G
4:     EXPAND(G,∅,∅,V(G),V(G),A_max,B_max)
5:     devolve (A_max, B_max)
6: fim função
7:
8: função EXPAND(G,A,B,P_a,P_b,A_max,B_max)
9:     (bounds,order) ← CLIQUESORT(G,P_a)
10:    para i ← |P_a| até 1 faça
11:        se bounds[i] + |A| > A_max e |B| + |P_b| > B_max então
12:            v ← order[i]
13:            A ← A ∪ {v}
14:            P_a ← P_a \ {v}
15:            P'_a ← P_a ∩ N_G(v)
16:            P'_b ← P_b ∩ N_G(v)
17:            se |A| = |B| e |A| > |A_max| então
18:                (A_max, B_max) ← (A,B)
19:            fim se
20:            se P'_b ≠ ∅ então
21:                EXPAND(G,B,A,P'_b,P'_a,B_max,A_max)
22:            fim se
23:            A ← A \ {v}
24:            se B ≠ ∅ então
25:                P_b ← P_b \ {v}
26:            fim se
27:        fim se
28:    fim para
29: fim função
```

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. Além disso, o array *order* é acessado da direita pra esquerda.

Note que ainda assim, este algoritmo tem uma pequena desvantagem. No caso em que P_a é um conjunto independente, a função *CLIQUESORT* seria desnecessária, pois um bound eficiente naturalmente aconteceria. Segundo McCreesh e Prosser (2014), na literatura tentativas de resolução desse problema não tiveram resultados satisfatórios.

O trabalho de McCreesh e Prosser (2014) contribuiu com um algoritmo eficaz para encontrar um conjunto independente em um grafo arbitrário. Além do ganho com a exclusão da simetria, há também um bom desempenho para grafos densos com grandes quantidades de vértices.

Algoritmo 3 Algoritmo CLIQUESORT

```
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
```

3.2 Algoritmos de Bonecas Russas

Em Corrêa *et al.* (2014), um algoritmo baseado no método das Bonecas Russas é desenvolvido para encontrar uma clique de cardinalidade máxima em um grafo arbitrário. Quando comparado com o método Branch and Bound, a principal diferença do método das Bonecas Russas é que os nós de sua árvore de busca correspondem aos subproblemas de decisão, em vez dos subproblemas de otimização do método Branch and Bound.

Em relação com implementações anteriores deste método na literatura, várias melhorias são apresentadas. Algumas delas são adaptações de técnicas já empregadas com sucesso em algoritmos Branch e Bound, como o uso de coloração aproximada com o intuito de podar os subproblemas.

Os resultados obtidos com o algoritmo superou a maioria dos algoritmos da literatura para grafos densos. Em alguns casos tendo desempenho duas vezes mais rápido. Na próxima seção será mostrado um algoritmo baseado no método das Bonecas Russas que servirá como base para o desenvolvimento do algoritmo do presente trabalho.

3.2.1 Algoritmo proposto por Silva (2016)

Em Silva (2016) é desenvolvido um algoritmo exato baseado nas técnicas de Branch and Bound e Bonecas Russas para resolver o PBIBM. Este algoritmo se baseia no algoritmo proposto por McCreesh e Prosser (2014) citado na seção 3.1. Algumas adaptações foram aperfeiçoadas, como a não utilização do algoritmo de $CLIQUESORT$, o que implica em não utilização de Partições em Cliques. Outra mudança em relação ao algoritmo de McCreesh e Prosser (2014) é a utilização de podas na exploração da árvore de busca via método das Bonecas Russas.

O primeiro tipo de poda ocorre quando não é possível com o tamanho da solução alcançada, somado a quantidade de candidatos, ultrapassar o valor máximo atual. O segundo tipo de poda ocorre quando, ao selecionar um determinado vértice do conjunto de candidatos, se o tamanho da melhor solução que pode ser encontrada com aquele vértice, somada com o número de candidatos, não consegue ser melhor que o da solução já encontrada. Neste caso, o subproblema é podado.

O algoritmo proposto por Silva (2016) utiliza a técnica de Branch and Bound utilizada em McCreesh e Prosser (2014), como mostrado na seção 3.1, com o incremento da técnica de Bonecas Russas, baseado na técnica desenvolvida em Trukhanov *et al.* (2013). Conforme mostrado no Algoritmo 4 desenvolvido por Silva (2016).

Algoritmo 4 Algoritmo das Bonecas Russas desenvolvido em Silva (2016)

```
1:  função BONECASRUSSAS(G, A, B)
2:      π ← ORDENACAO(G)
3:      RECONSTRUCAOADJACENCIA(G, π)
4:      R ← vetor de inteiros
5:      para i ← 1 ate |V_G| faça
6:          v ← V_G[i]
7:          PA ← V_G \ N_G(v)‾
8:          PB ← V_G \ N_G(v)
9:          A' ← v
10:         B' ← ∅
11:         NOVOEXPAND(G, B', A', PB, PA, B, A, R)
12:         R[v] ← |A|
13:     fim para
14: fim função
```

O algoritmo funciona da seguinte forma, primeiramente o conjunto de vértices é ordenado e um vetor R para armazenar onde 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 $EXPAND$, $|A|$ contém o tamanho da melhor solução encontrada para o subproblema e esse valor pode ser armazenado em R. Cada boneca gerada pelo o Algoritmo 4 é resolvida pelo Algoritmo 5.

Algoritmo 5 Algoritmo NOVOEXPAND desenvolvido em Silva (2016)

```
1:  função NOVOEXPAND(G, A, B, PA, PB, A_max, B_max, R)
2:      se |A| = |B| e |A| > |A_max| então
3:          (A_max, B_max) ← (A, B)
4:      fim se
5:      enquanto |A| + |PA| > A_max e |B| + |PB| > B_max faça
6:          v ← last(PA)
7:          se max(A,B) + R[v] ≤ |A_max| então
8:              devolve
9:          fim se
10:         A' ← A ∪ {v}
11:         PA ← PA \ {v}
12:         PA' ← PA \ N_G(v)
13:         PB' ← PB \ N_G(v)
14:         NOVOEXPAND(G, B', A', PB', PA', B_max, A_max, R)
15:     fim enquanto
16: fim função
```

O algoritmo *NOVOEXPAND* é baseado no algoritmo *EXPAND* desenvolvido em McCreesh e Prosser (2014). As únicas diferenças são: Não se utiliza o algoritmo de ordenação de cliques, o $CLIQUESORT$, ao invés disso, utiliza-se de podas através das Bonecas Russas. Não utiliza ordem estática dos vértices.

Em McCreesh e Prosser (2014) o *Bound* provinha-se da função $CLIQUESORT$, visando limitar superiormente um conjunto independente máximo que continha um vértice v, com o objetivo de não visitar todo o espaço de busca. Em Silva (2016) o *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. Assim a melhor solução possível é o máximo entre A e B somado com $R[v]$. Da mesma forma, se a melhor solução possível não ultrapassa o valor da melhor solução alcançada, não adianta continuar, pois não teria resultados melhores.

Como resultados, foi obtido um algoritmo exato para resolver o PBIBM, embora quando comparado com McCreesh e Prosser (2014) não tenham tão bons tempos de execução, superando o próprio em apenas um caso de instância. Este caso de instância reduzindo o tempo

de 94 segundos no algoritmo de McCreesh e Prosser (2014) para menos de 1 segundo. Foram usadas instâncias de grafos obtidas a partir de DIMACS (2016).

3.3 Algoritmo proposto por Zhou *et al.* (2018)

Em Zhou *et al.* (2018) é proposta uma solução exata para o PBIBM para a classe de grafos bipartidos. É desenvolvido um procedimento de *Upper Bound Propagation* (UBP) envolvendo cada vértice do grafo. 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.

Tome $G = (A \cup B, E)$ um grafo bipartido e $H = (U \cup V, E')$ uma biclique induzida balanceada de G. Denotaremos por $cpv(i)$ uma função que retorna o conjunto ao qual pertence o vértice i em H, $cpv(i) = U$ se $i \in U$. Assumindo que $w_{ij} = |N_H(i) \cap N_H(j)|$ para $i, j \in U$. Seja também ub um vetor de inteiros onde serão armazenados os *bounds* para cada vértice, onde ub_i representa o *bound* armazenado para o vértice i. A cardinalidade de $cpv(i)$ não pode ser maior que ub_i. O UBP é baseado conforme as seguintes proposições elaboradas por Zhou *et al.* (2018):

Proposição 1. *Para qualquer vértice $i \in U \cup V$, $|cpv(i)| \leq deg(i)$.*

Prova. *Suponha por contradição que $|cpv(i)| > deg(i)$. Isto implica que existe $v \in \overline{cpv(i)}$ tal que $v \notin N_H(i)$. Neste caso H não seria uma biclique. Absurdo.*

Tome o grafo bipartido G da Figura 8 como exemplo. Conforme a Proposição 1 temos que $ub_1 = ub_5 = 2$, $ub_2 = ub_3 = ub_4 = 3$. Embora o grau seja um limite razoável, existem limites mais aperfeiçoados, como serão apresentados nas Proposições 2 e 3.

Ainda na Figura 8, tome como exemplo o vértice 1, $deg(1) = 2$. Logo pela Proposição 1 temos que $|cpv(1)| \leq 2$. Mas para que este limite fosse efetivo, um outro vértice $v \in cpv(1)$ deveria compartilhar de ao menos dois vizinhos em comum com o vértice 1. Assim temos que $|N_G(1) \cap N_G(v)| \geq 2$. Caso contrário, embora $deg(1)$ pudesse ser utilizado como *bound*, este não seria o melhor *bound* para o vértice 1. A Proposição 2 é então elaborada visando melhorar este *bound*.

Proposição 2. *Seja um vértice $i \in U$ e seja y_i o maior inteiro tal que existem pelo menos y_i valores em $\{w_{ij} : j \in U\}$ maiores ou iguais a y_i. Então $|cpv(i)| \leq y_i$.*

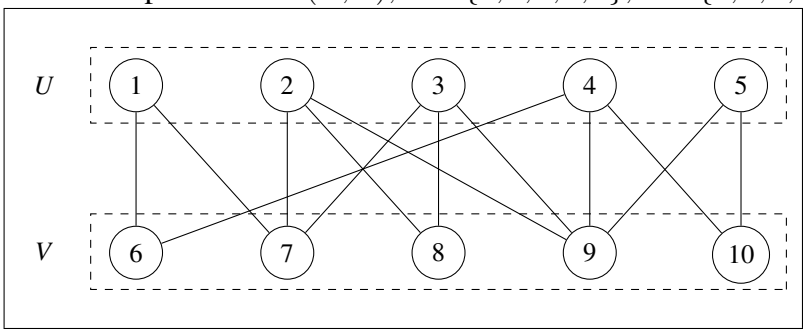
Prova. *Para quaisquer dois vértices $i, j \in U$ temos que $V \subseteq N_H(i) \cap N_H(j)$. Assim se no máximo y_i vértices em U compartilham de y_i vértices com i, y_i é um upper bound para o tamanho do conjunto da biclique induzida balanceada contendo i.*

Ainda na Figura 8, conforme a Proposição 2 temos que $ub_1 = 1$, pois $w_{12} = w_{13} = w_{14} = 1 (y_1 = 1)$. Similarmente $ub_2 = ub_3 = 2, ub_4 = ub_5 = 1$. Embora possa-se utilizar a Proposição 2 também para V, esta não é aplicada, pois segundo Zhou *et al.* (2018) este procedimento requer grande quantidade de memória e tempo. Por outro lado a Proposição 3 melhora os limites tanto de U quanto de V.

Proposição 3. *Seja um vértice $i \in U \cup V$ e seja t_i o maior inteiro tal que existem t_i vértices em $N_H(i)$ com upper bound pelo menos t_i. Então $|cpv(i)| \leq t_i$.*

Prova. *Suponha por contradição que t_i não seja um upper bound para o tamanho do conjunto da biclique induzida balanceada contendo i, então existe uma biclique induzida balanceada $\{U', V'\}$ envolvendo $i \in U'$ de tamanho de um lado t_i', com $t_i' > t_i$. Isso implica que t_i' vértices em V' tem upper bound pelo menos t_i'. Isso contradiz a condição de que t_i é o máximo inteiro tal que existe em $N_H(i)$ t_i vértices com upper bound pelo menos t_i.*

Figura 8 – Grafo Bipartido $G = (U,V), U = \{1,2,3,4,5\}, V = \{6,7,8,9,10\}$.



Fonte: Zhou *et al.* (2018)

A partir dessas proposições o Algoritmo 6 é produzido para calcular um *upper bound* de cada vértice. Na linha 2 o vetor *ub* é inicializado de acordo com a Proposição 1, com o grau de cada vértice. Entre as linhas 5 a 9 a matriz w_{ij} é inicializada. Nas linhas 11 a 16 o vetor *ub* é atualizado de acordo com a Proposição 2, apenas para U. Entre as linhas 18 e 27 então é aplicada a Proposição 3.

O algoritmo proposto por Zhou *et al.* (2018) se mostrou bastante eficiente para classe de grafos bipartidos. Quando comparado ao algoritmo de McCreesh e Prosser (2014) se mostrou superior em algumas instâncias bipartidas.

Algoritmo 6 Procedimento de UBP desenvolvido em Zhou *et al.* (2018)

função UBP($G = (U, V, E)$)
    $\forall i \in U \cup V$, $ub_i \leftarrow deg(i)$
    $\forall (i, j) \in U \times U$, $w_{ij} \leftarrow 0$

    para $k \in V$ **faça**
        para $(i, j) \in N_G(k) \times N_G(k)$ **faça**
            $w_{ij} \leftarrow w_{ij} + 1$
        fim para
    fim para

    para $i \in U$ **faça**
        O maior inteiro y_i tal que: $|\{j \in U : w_{ij} \geq y_i\}| \geq y_i$
        se $y_i < ub_i$ **então**
            $ub_i \leftarrow y_i$
        fim se
    fim para

    $stable \leftarrow false$
    enquanto $stable \neq true$ **faça**
        $stable \leftarrow true$
        para $i \in U \cup V$ **faça**
            O maior inteiro t_i tal que: $|\{j \in N_G(i) : ub_j \geq t_i\}| \geq t_i$
            se $t_i < ub_i$ **então**
                $ub_i \leftarrow t_i$
            fim se
        fim para
    fim enquanto
    devolve ub
fim função

4 ALGORITMOS PROPOSTOS

Nesta seção serão apresentados os dois algoritmos *Branch and Bound* propostos neste trabalho. O primeiro, utilizando da técnica de Partições em Clique e Bonecas Russas, conforme os Algoritmos 2 e 5 respectivamente. O segundo, utilizando da mesma técnica de Bonecas Russas com um procedimento de UBP, uma solução para fraqueza de Partições em Cliques para grafos bipartidos, conforme Algoritmo 6.

4.1 Algoritmos base

McCreesh e Prosser (2014) propõe um algoritmo B&B 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.

O critério de poda utilizado neste algoritmo é $|A| + bounds[v] < |A_{max}|$ 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.

Silva (2016) também propõe um algoritmo B&B para o PBIBM. Onde cada subproblema pode ser definido por uma tupla $(A, B, P_a, P_b, A_{max}, B_{max}, R)$, mesmos parâmetros de que McCreesh e Prosser (2014), acrescentando R, que representa um vetor com os resultados obtidos por cada boneca.

Um critério para podar cada subproblema consiste em $max(|A|, |B|) + R[v] < |A_{max}|$. 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.

Em Zhou *et al.* (2018) também propõe um algoritmo B&B, onde cada subproblema é podado baseado em um limite gerado por um procedimento de UBP no pré processamento do grafo. Tal procedimento é bastante útil quando aplicado a grafos da classe bipartido. Cada subproblema é definido por uma tupla $(A, B, P_a, P_b, A_{max}, B_{max}, ub)$, mesmo parâmetros que McCreesh e Prosser (2014), acrescentando o vetor ub onde estão armazenados os limites calculados para cada vértice.

O critério utilizado para podar cada subproblema consiste em $|A| + ub_v < |A_{max}|$. Se

ub_v armazena a melhor solução possível com o vértice v então ao selecionar v para uma nova solução, existe a possibilidade de adicionar no máximo ub_v vértices a solução. Assim se a melhor solução possível não ultrapassa o valor da melhor solução alcançada, o subproblema é podado.

4.2 Ordem inicial dos vértices

A ordenação inicial dos vértices é realizada de forma estática, evitando reordenação a cada subproblema gerado. É baseada no grau do vértice e dependendo da ordem causa um impacto sob a quantidade de subproblemas gerados.

Em McCreesh e Prosser (2014) é adotada ordem decrescente de grau do vértice. Segundo McCreesh e Prosser (2014) ao selecionar vértices de maior grau primeiro têm-se mais chances de obter Partições em Cliques mais próxima da exata. Em Silva (2016) é adotada ordem crescente na seleção dos vértices, pois quanto menor o grau do vértice menor a boneca e menos subproblemas são gerados.

A ordenação inicial de vértices adotada neste trabalho é a mesma que (MCCREESH; PROSSER, 2014), utilizando a ordem inversa para selecionar vértices de cada boneca, como em (SILVA, 2016). Para o procedimento de UBP é utilizada ordem crescente no grau do vértice.

4.3 ALGBRPC: Algoritmo de Bonecas Russas e Partições em Cliques

O primeiro algoritmo deste trabalho se fundamenta na utilização de Bonecas Russas e de Partições em Cliques para encontrar bons limites superiores para cada subproblema gerado, cada boneca. Utilizando algoritmo de B&B de (MCCREESH; PROSSER, 2014) para resolver cada boneca, cada subproblema do primeiro algoritmo proposto pode ser representado pela tupla $(A, B, P_a, P_b, A_{max}, B_{max}, R)$, onde R representa o vetor onde serão armazenados os resultados de cada boneca.

Assim o critério de poda consiste em $max(|A|, |B|) + R[v] < |A_{max}|$ e $|A| + bounds[v] < |A_{max}|$. Utilizando os limites de Partições em Cliques através do vetor *bounds* e de Bonecas Russas através do vetor R.

4.4 ALGBRUBP: Algoritmo de Bonecas Russas e *Upper Bound Propagation* (UBP)

Grafos bipartidos são livres de triângulo conforme Proposição 4 e não possuem cliques de tamanho maior que 2 conforme Proposição 5. Para grafos da classe bipartida o Algo-

ritmo 2 de Partições em Cliques não seria efetivo, pois P_a e P_b seriam conjuntos independentes e suas próprias cardinalidades poderiam ser utilizadas como *bound*.

Proposição 4. *Grafos bipartidos são livres de triângulo.*

Prova. *Suponha por contradição que, dado um grafo bipartido $G = (A \cup B, E)$, G possua um subgrafo que induza um triângulo. Isso ocasionaria que existe $a,b,c \in A \cup B$ tal que, $\{\{a,b\},\{b,c\},\{c,a\}\} \subseteq E$. Como $\{a,b\} \in E$, então a e b pertencem a conjuntos diferentes. Note também que $\{\{c,a\},\{c,b\}\} \subseteq E$, assim se $c \in A(B)$, $A(B)$ não seria um conjunto independente e este grafo não seria bipartido. Absurdo.*

Proposição 5. *Grafos bipartidos não possuem cliques de tamanho maior que 2.*

Prova. *Esta é uma prova simples, pois todo grafo triângulo induz uma clique de tamanho 3 e como grafos bipartidos são livres de triângulos, consequentemente são livres de cliques de tamanho 3. Se um grafo não possui nenhum subgrafo que induz uma clique de tamanho 3, então consequentemente não possuem cliques de tamanho $N(N > 3)$, pois estas teriam subgrafos que induzem cliques de tamanho 3.*

Para solucionar este problema é criado um segundo algoritmo, este específico para classe de grafos bipartidos. Neste algoritmo também é utilizado da técnica de Bonecas Russas, mas é utilizado o procedimento de UBP proposto em Zhou *et al.* (2018) para podar cada subproblema, substituindo o *CLIQUESORT*.

Utilizando algoritmo B&B de (MCCREESH; PROSSER, 2014), cada subproblema do segundo algoritmo proposto pode ser representado pela tupla $(A, B, P_a, P_b, A_{max}, B_{max}, ub, R)$, onde *ub* e *R* representam, respectivamente, o vetor onde serão armazenados os *bounds* para cada vértice e o vetor onde serão armazenados os resultados de cada boneca.

Assim o critério de poda para cada subproblema consiste em $max(|A|,|B|) + R[v] < |A_{max}|$ e $|A| + ub_v < |A_{max}|$. Utilizando como limites para cada vértice tanto do vetor *ub* provindos do procedimento de UBP, quanto do vetor *R* com um *bound* para cada vértice, cada boneca.

5 RESULTADOS

Comparamos os resultados do primeiro algoritmo proposto, denominado ALGBRPC, com o algoritmo desenvolvido por Silva (2016). Avaliamos o desempenho do algoritmo utilizando as instâncias presentes em DIMACS, grupo de instâncias consolidadas na literatura. Comparamos também o segundo algoritmo proposto, denominado ALGBRPC, com o algoritmo desenvolvido por McCreesh e Prosser (2014), utilizando instâncias adaptadas de grafos bipartidos presentes em KONECT e de grafos bipartidos gerados de forma aleatória. A descrição das instâncias utilizadas estão no Apêndice A.

Os testes foram realizados utilizando um computador com as seguintes configurações: 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 o recurso de paralelismo de bits.

A Tabela 1 representa a comparação do ALGBRPC com o algoritmo proposto de (SILVA, 2016). A primeira coluna representa o nome da instância e 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. Na ultima linha da tabela foi adicionado um campo com a média geral dos algoritmos, comparando numero de subproblemas e no tempo computacional.

Analisando a Tabela 1, tem-se que o ALGBRPC foi melhor que o algoritmo de (SILVA, 2016) em 25 de 31 instâncias, nas demais os resultados de ambos foram muito próximos. Sendo assim o ALGBRPC é mais rápido que o algoritmo de comparação em aproximadamente 80% das instâncias, sendo até dez vezes mais rápido na instância **san200_0.9_2**. Analisando minuciosamente a média geral na Tabela 1, tem-se que o ALGBRPC mostra-se melhor tanto no número de subproblemas quanto no tempo computacional.

A Tabela 2, representa uma comparação do algoritmo proposto por (MCCREESH; PROSSER, 2014) com o ALGBRUBP 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, tem-se que o ALGBRUBP foi superior ao algoritmo de (MCCREESH; PROSSER, 2014) em 15 das 25 instâncias. Sendo assim o ALGBRUBP proposto é mais rápido que o algoritmo de comparação em aproximadamente 60% das instâncias, sendo até sete vezes mais rápido na instância **movielens-10m_ti**.

Tabela 1 – Resultados para instâncias de grafos DIMACS.

	Algoritmo de (SILVA, 2016)		**ALGBRPC**	
Instância	**tempo(s)**	**nº subproblemas**	**tempo(s)**	**nº 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
Média Geral	147.88	315848117	**69.90**	**140552942**

Fonte: Próprio autor.

Embora o ALGBRUBP tenha quantidade de subproblemas superior, ele consegue resolvê-los em tempo inferior a (MCCREESH; PROSSER, 2014), como pode ser evidenciado conforme a média geral na Tabela 2. Isso é consequência de que no algoritmo de (MCCREESH; PROSSER, 2014) a função *CLIQUESORT* é invocada em cada ramo, enquanto no ALGBRUBP utilizando 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 e instâncias geradas de forma aleatória de grafos bipartidos.

	Algoritmo de *MCCREESH*		**ALGBRUBP**	
Instância	**tempo(s)**	**nº subproblemas**	**tempo(s)**	**nº subproblemas**
moreno_crime_crime	0.003	1031	**0.001**	3482
movielens-10m_ui	**0.913**	186258	1.839	2122300
mv300_0.40	1.122	9015605	**0.354**	12532169
movielens-10m_ti	2.452	43745	**0.336**	74401
movielens-10m_ut	1.873	73521	**1.000**	577917
mv191_0.90	0.006	56321	**0.001**	57230
mv300_0.50	**28.490**	212284340	182.552	6646097451
unicodelang	0.003	799	**0.001**	4454
dbpedia-writer	1.385	19403	**0.735**	26228
youtube-groupmember.	1.943	66376	**0.575**	106913
mv250_0.40	**0.111**	1110219	0.287	14996116
mv250_0.50	3.073	27521363	**1.684**	92773996
mv400_0.40	**9.560**	72893969	42.329	1502152959
mv250_0.60	85.828	576589463	**33.482**	1788037197
opsahl-collaboration	**0.006**	56321	0.722	47005
mv100_0.80	**0.311**	2154321	0.453	23627494
mv200_0.50	0.158	1430326	**0.053**	2767743
mv200_0.60	**9.328**	65167403	59.286	3159248411
edit-frwikinews	2.763	27094	**0.877**	414941
escorts	0.490	42213	**0.186**	173322
mv600_0.40	239.267	1461964718	**30.979**	587745188
mv350_0.40	**1.050**	8834208	1.141	42193057
mv500_0.50	3161.274	19679981005	**2255.293**	83727482748
mv450_0.40	**38.160**	284580336	167.345	5982637305
mv500_0.40	**45.298**	353179187	198.379	7149175012
Média Geral	145,39	**910291181**	**119.19**	4429403001

Fonte: Próprio autor.

6 CONCLUSÕES E TRABALHOS FUTUROS

Apresentamos dois algoritmos *Branch and Bound* para o PBIBM combinando e aperfeiçoando várias técnicas já aplicadas com sucesso na literatura. Os resultados mostram que as técnicas utilizadas no primeiro algoritmo, denominado ALGBRPC, provocaram redução no número de subproblemas e consequentemente no tempo, sendo mais efetivo que o algoritmo de (SILVA, 2016). Os resultados mostram também que o segundo algoritmo, denominado ALGBRUBP, se tornou mais efetivo em relação a tempo computacional de que (MCCREESH; PROSSER, 2014) para grafos bipartidos.

Algumas pesquisas e melhorias não realizadas neste trabalho podem ser desenvolvidas. Alguns pontos 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 próxima da exata 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 *multi-threading* para adicionar ao algoritmo, neste caso deve ser pensado uma outra forma de construção das bonecas.
- Desenvolver uma modelagem matemática específica para o problema, a maioria das modelagens existentes utilizam do fato de encontrar dois conjuntos independentes máximos no complemento do grafo.
- Realizar testes com conjuntos de instâncias diferentes, a fim de observar o comportamento dos algoritmos para essas instâncias.
- Desenvolver um particionamento específico para bicliques, um *BICLIQUESORT*, alternativa ao *CLIQUESORT*.
- Desenvolver um algoritmo híbrido que combine Partição em Cliques com UBP. Uma sugestão seria identificar quando o grafo de entrada é bipartido e a partir deste ramo substituir Partições em Cliques pelo procedimento de UBP.

REFERÊNCIAS

AL-YAMANI, A. A.; RAMSUNDAR, S.; PRADHAN, D. K. A defect tolerance scheme for nanotechnology circuits. **IEEE Transactions on Circuits and Systems I: Regular Papers**, IEEE, v. 54, n. 11, p. 2402–2409, 2007.

BABEL, L. A fast algorithm for the maximum weight clique problem. **Computing**, Springer, v. 52, n. 1, p. 31–38, 1994.

BALAS, E.; XUE, J. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. **Algorithmica**, Springer, v. 15, n. 5, p. 397–412, 1996.

BAZARAA, M. S.; JARVIS, J. J.; SHERALI, H. D. **Linear programming and network flows**. [S.l.]: John Wiley & Sons, 2011.

BERNHARD, K.; VYGEN, J. Combinatorial optimization: Theory and algorithms. **Springer, Third Edition, 2005.**, 2008.

BLUM, C.; ROLI, A.; SAMPELS, M. **Hybrid metaheuristics: an emerging approach to optimization**. [S.l.]: Springer, 2008. v. 114.

CARRAGHAN, R.; PARDALOS, P. M. **A parallel algorithm for the maximum weight clique problem**. [S.l.]: Pennsylvania State University, Department of Computer Science, 1990.

CHEESEMAN, P. C.; KANEFSKY, B.; TAYLOR, W. M. Where the really hard problems are. In: **IJCAI**. [S.l.: s.n.], 1991. v. 91, p. 331–340.

CHENG, Y.; CHURCH, G. Biclstering of expression data, in proceedings of the eighth international conference on intelligent systems for molecular biology (ismb). AAAI Press Menlo Park, California, 2000: 93–103, 2000.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

CORRÊA, R. C.; MICHELON, P.; CUN, B. L.; MAUTOR, T.; DONNE, D. D. A bit-parallel russian dolls search for a maximum cardinality clique in a graph. **arXiv preprint arXiv:1407.1209**, 2014.

DIMACS. **DIMACS BENCHMARK**. 2016. <http://dimacs.rutgers.edu>. "[Online: acessado em 29/10/2018]".

GARY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-completeness**. [S.l.]: WH Freeman and Company, New York, 1979.

MCCREESH, C. **Solving hard subgraph problems in parallel**. Tese (Doutorado) — University of Glasgow, 2017.

MCCREESH, C.; PROSSER, P. An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem. In: SPRINGER. **International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems**. [S.l.], 2014. p. 226–234.

NEMHAUSER, G. L.; WOLSEY, L. A. Integer programming and combinatorial optimization. **Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin**, Springer, v. 20, p. 8–12, 1988.

ÖSTERGÅRD, P. R. A new algorithm for the maximum-weight clique problem. **Electronic Notes in Discrete Mathematics**, Elsevier, v. 3, p. 153–156, 1999.

PAPADIMITRIOU, C. H. **Computational complexity**. [S.l.]: John Wiley and Sons Ltd., 2003.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. [S.l.]: Courier Corporation, 1998.

RAVI, S.; LLOYD, E. L. The complexity of near-optimal programmable logic array folding. **SIAM Journal on Computing**, SIAM, v. 17, n. 4, p. 696–710, 1988.

SEGUNDO, P. S.; RODRIGUEZ-LOSADA, D.; GALAN, R.; MATIA, F.; JIMENEZ, A. Exploiting cpu bit parallel operations to improve efficiency in search. In: IEEE. **19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)**. [S.l.], 2007. v. 1, p. 53–59.

SILVA, A.; SILVA, A.; SALES, C. L. Largura em arvore de grafos planares livres de ciclos pares induzidos. **Anais do 39o. Simpósio Brasileiro de Pesquisa Operacional**, 2007.

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

TAVARES, W. A. Algoritmos exatos para problema da clique maxima ponderada. **Université D'Avignon**, Avignon, 2016.

TOMITA, E.; SEKI, T. An efficient branch-and-bound algorithm for finding a maximum clique. In: **Discrete mathematics and theoretical computer science**. [S.l.]: Springer, 2003. p. 278–289.

TRUKHANOV, S.; BALASUBRAMANIAM, C.; BALASUNDARAM, B.; BUTENKO, S. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. **Computational Optimization and Applications**, Springer, v. 56, n. 1, p. 113–130, 2013.

VERFAILLIE, G.; LEMAÎTRE, M.; SCHIEX, T. Russian doll search for solving constraint optimization problems. In: **AAAI/IAAI, Vol. 1**. [S.l.: s.n.], 1996. p. 181–187.

ZHOU, Y.; ROSSI, A.; HAO, J.-K. Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. **European Journal of Operational Research**, Elsevier, v. 269, n. 3, p. 834–843, 2018.

APÊNDICE A – DADOS DAS INSTÂNCIAS

Tabela 3 – Detalhes das instâncias utilizadas nos testes.

Instância	$\|V\|$	$\|E\|$	Densidade
C250.9	250	27984	0.89
C500.9	500	112332	0.90
DSJC500_5	500	125248	0.59
MANN_a45	1035	533115	0.99
MANN_a81	3321	5506380	0.99
brock200_1	200	14834	0.74
brock200_4	200	13089	0.65
c-fat200-5	200	8473	0.42
c-fat500-10	500	46627	0.37
c-fat500-2	500	9139	0.07
c-fat500-5	500	23191	0.18
gen200_p0.9_44	200	17910	0.90
gen200_p0.9_55	200	17910	0.90
gen400_p0.9_55	400	71820	0.90
gen400_p0.9_65	400	71820	0.90
hamming10-2	1024	518656	0.99
hamming6-2	64	1824	0.90
hamming6-4	64	704	0.34
hamming8-2	256	31616	0.96
hamming8-4	256	20864	0.63
johnson16-2-4	120	5460	0.76
keller4	171	9435	0.64
p_hat1000-3	1000	371746	0.74
p_hat300-3	300	33390	0.74
p_hat500-3	500	93800	0.75
p_hat700-3	700	183010	0.74
san1000	1000	250500	0.50
san200_0.9_2	200	17210	0.90
san400_0.7_1	400	55860	0.70
sanr200_0.9	200	17863	0.89
sanr400_0.7	400	55869	0.70

Fonte: Próprio autor.

Tabela 4 – Detalhes das instâncias bipartidas geradas aleatoriamente.

Instância	$\|A\|$	$\|B\|$	$\|E\|$	Densidade
mv300_0.40	128	172	8806	0.40
mv300_0.50	127	173	10985	0.50
mv250_0.40	68	182	4950	0.40
mv250_0.50	82	168	6888	0.50
mv400_0.40	160	240	15360	0.40
mv250_0.60	10	144	9158	0.60
mv100_0.80	34	66	1792	0.80
mv200_0.50	50	150	3750	0.50
mv200_0.60	98	102	5997	0.60
mv427_0.55	300	127	68675	0.55
mv600_0.40	182	418	30430	0.40
mv350_0.40	88	262	9222	0.40
mv500_0.50	214	286	30602	0.50
mv450_0.40	205	245	20090	0.40
mv500_0.40	160	340	21760	0.40

Fonte: Próprio autor.