# Métodos Exatos e Heurísticos para Biclusterização em Grafos

**Rian Gabriel S. Pinheiro, Ivan César Martins, Fábio Protti,**
**Luiz Satoru Ochi, Luidi Gelabert Simonetti**
Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Patria, 156 - Bloco E - 3ºandar, São Domingos, Niterói - RJ
{rgpinheiro,imartins,fabio,satoru,luidi}@ic.uff.br

**RESUMO**

Neste trabalho apresentamos um método *Branch and Cut* e uma meta-heurística GRASP para o *Bicluster Graph Editing Problem* (BGEP). O BGEP é um problema de biclusterização semelhante ao *Cluster Graph Editing* mas é menos explorado pela literatura. Propomos um teorema que servirá como um pré-processamento em que para todo par de vértice, podemos saber à priori se eles estarão em biclusters diferentes. No GRASP propomos uma busca local e um método construtivo baseado em árvore geradora mínima. Com o resultado vimos que tanto o GRASP quanto o método exato são satisfatórios para as instâncias propostas.
**PALAVRAS CHAVE**. Biclusterização. Meta-heurísticas. Branch and Cut.
**Área principal** OC - Otimização Combinatória.

**ABSTRACT**

We present a Branch and Cut method and a GRASP metaheuristic for the Bicluster Graph Editing Problem (BGEP). The BGEP is similar to the Cluster Graph Editing but less explored in the literature. We propose a theorem that will serve as a preprocessor that for every pair of vertex, we know if they are in different biclusters. In GRASP we propose a local search and constructive method based on minimum spanning tree. As a result we have seen that both, GRASP and exact method, are satisfactory for instances proposed.
**KEYWORDS**. Biclustering. Metaheuristics. Branch and Cut.
**Main area** OC - Combinatorial Optimization.

## 1 Introdução

O *Bicluster Graph Editing Problem* (BGEP) é um problema NP-completo (Amit, 2004) em que dado um grafo bipartido $G = (V, U, E)$ e um inteiro $k \geq 0$, deseja-se remover ou adicionar no máximo $k$ arestas de forma a tornar $G$ uma união de subgrafos bipartidos completos (bicliques).

Um problema semelhante é o *Cluster Graph Editing Problem*, primeiramente estudado por Gupta & Palit (1979) que consiste em tornar $G$ uma partição de subgrafos completos (cliques) em que $G$ é não necessariamente bipartido. Ambos são casos de problemas de partição em grafos.

Vários tipos de algoritmos de particionamento são descritos em Pothen (1997) aplicado em diversas áreas da ciência da computação. O trabalho de Alpert & Kahng (1995) mostra como explorar o espaço de soluções viáveis através de algoritmos gulosos, buscas locais, *simulated annealing* e algoritmos evolucionários. No entanto, poucos trabalhos na literatura abordam o BGEP.

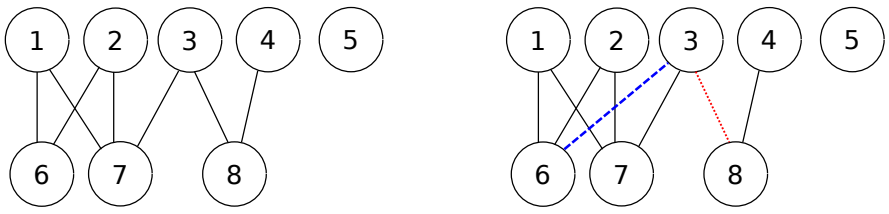
O conceito de biclusterização foi introduzido em meados da década de 70 por Hartigan (1975), mas seu primeiro uso se deu no contexto da biologia computacional com o trabalho de Cheng & Church (2000). Desde então algoritmos têm sido propostos e utilizados neste e em outros campos de aplicação (Abdullah & Hussain, 2006; Faure et al., 2007; Bisson & Hussain, 2008).

Nomes como co-clusterização, clusterização bidimensional, entre outros, são frequentemente utilizados na literatura para se referir ao mesmo problema. Em suma, este conceito denota a existência de submatrizes "significativas" em uma dada matriz, ou seja, um subconjunto de linhas e de colunas que possuem padrões únicos baseados em um certo método de classificação.

No entanto, existem diversas formulações e contextos para o problema de biclusterização. Por exemplo, podemos representar a matriz através de um grafo bipartido, considerando as linhas e colunas como vértices no qual cada conjunto de vértices (linhas e colunas) pertencem a bipartições diferentes.

Ainda neste contexto, podemos considerar os pesos das arestas como a interação entre esses vértices. Particularmente, em um grafo bipartido sem pesos, um *bicluster* ideal é um subgrafo bipartido completo (biclique). A presença de bicliques indicaria um alto grau de similaridade entre os dados agrupados. Em especial, quando todas as componentes conexas de um grafo bipartido formam uma biclique, o chamamos de grafo biclusterizado.

Neste trabalho, nos focaremos na versão de otimização de um dos diversos problemas de biclusterização apresentados e definidos em Amit (2004), o *Bicluster Graph Editing Problem*. Neste problema é dado um grafo bipartido $G = (U, V, E)$ e deseja-se saber qual o número mínimo de edições de arestas (remoções e adições) que deve-se fazer em $G$ para que este se torne um grafo biclusterizado.




(a) Instância (b) Solução

Figura 1: Exemplo

A figura 1 mostra um exemplo onde uma adição (azul tracejado) e uma remoção (vermelho pontilhado) tornam o grafo em uma partição de biclique. Observe que esta não é a solução ótima, uma vez que, bastaria remover a aresta $(3, 7)$ para torná-lo um grafo *bicluster*.

Amit (2004) provou que este problema, formulado em sua versão de decisão, é NP-completo. Ele propõe uma formulação em programação inteira binária e um algoritmo 11-aproximado para o problema. Com base no trabalho de Amit (2004), Guo et al. (2008) propõem um algoritmo randômico 4-aproximado.

Outras variações do problema são apresentadas por Amit (2004). Algumas variações são polinomiais, como por exemplo, encontrar a biclique com a máxima soma de pesos em seus vértices. Porém, a maioria continua NP-completo como:

- encontrar uma biclique cujas partições possuem cardinalidades $u$ e $v$ respectivamente;
- encontrar uma biclique com partições com cardinalidades iguais e maior que um inteiro $k$;
- encontrar uma biclique com mais que $k$ arestas;
- encontrar a biclique com a máxima soma de pesos em suas arestas.

### 1.1 Aplicações

O conceito de agrupamento de dados em *clusters* surge em muitos contextos e disciplinas diferentes. A modelagem através do BGEP produz boas soluções para um problema da biologia computacional que consiste em encontrar um bom agrupamento de genes na qual satisfaça um critério de homogeneidade, ou seja, genes dentro de um agrupamento devem ter uma alta semelhança entre si.

Para resolver este problema utilizando o BGEP devemos ter como entrada uma matriz gene (linha) x característica ou fenótipo (coluna). Com a matriz gene-característica podemos criar um grafo bipartido onde os genes e as características serão as bipartições. Transformando assim o problema do agrupamento de genes no BGEP.

Já Faure et al. (2007) mostram que uma variante do BGEP ajuda no projeto de uma rede *multicast*. Uma sessão *multicast* é definida como um subconjunto de clientes que requerem a mesma informação. Cada cliente pode exigir várias sessões *multicast*. A principal limitação é que uma rede de telecomunicações não pode controlar muitas sessões *multicast*, ao mesmo tempo. A solução encontrada é agrupar as sessões em um número limitado de *clusters*. Neste problema os as bipartições são formadas pelos conjuntos de clientes e sessões.

## 2 Proposta B&C

*Branch and Cut* (B&C) é um método exato utilizado para resolver problemas de programação inteira binária (Wolsey, 1998), no qual o problema é dividido em sub-problemas com domínios disjuntos. Cada sub-problema é representado por um nó em uma estrutura de árvore, na qual cada em nó é encontrado um limite primal e dual. Com os limites determinados — no caso deste trabalho uma solução inteira e o valor da solução sem as restrições de integralidade — o algoritmo elimina uma grande quantidade de soluções sem nem mesmo avaliá-las através de podas na árvore. Com a finalidade de torna o limite dual mais próximo, em cada nó executa-se um algoritmo de plano de corte que encontra novas restrições (cortes) e as adiciona ao sub-problema.

Neste trabalho utiliza-se a formulação matemática proposta por Amit (2004). Ela se baseia no fato que partições de *bicluster* tem o $P_4$ como subgrafo de proibição. Em outras palavras, toda vez que houver um subgrafo induzido $P_4$ em $G$, significa que $G$ não é uma partição de *biclusters*. $P_4$ é um grafo linear ou caminho composto por 4 vértices, como mostra a figura 2.

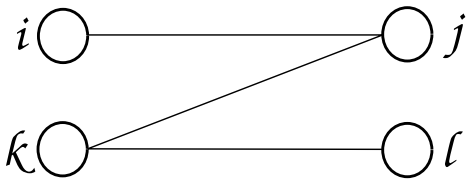



Figura 2: Subgrafo de proibição

A formulação é definida como:

$$\min \sum_{+(ij)} x_{ij} + \sum_{-(ij)} 1 - x_{ij} \tag{1}$$

$$\text{s.a } x_{ij} \leq x_{il} + x_{kj} + x_{kl} \qquad \forall i,k \in U \text{ e } j,l \in V \tag{2}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in U \text{ e } j \in V. \tag{3}$$

Em que, $x_{ij}$ é uma variável binária que assume o valor 0 se a solução contém a aresta entre os vértices $i$ e $j$; $+(ij) = \{ij|w(ij) = 1\}$ e $-(ij) = \{ij|w(ij) = 0\}$. Em (1) temos a função objetivo que indica quantas edições foram realizadas. A restrição (2) elimina os subgrafos induzidos $P_4$. E por fim (3) é a restrição de integralidade.

Veja que o número de restrições desta formulação é de $|U|^2|V|^2$. Dessa forma é necessário um pré-processamento que possa fixar ou gerar novos cortes para o problema. Como base do nosso pré-processamento, utilizamos o teorema da seção 4. Com ele, poderemos saber *a priori* que alguns pares de vértices estarão em bicliques diferentes.

Para acelerar o processo do B&C começamos com o modelo vazio, ou seja, com nenhuma restrição. Todas as restrições são armazenadas num *pool*. No decorrer do processo é verificado quais restrições do *pool* são violadas e as mesmas são adicionadas ao modelo. O responsável por gerenciar tudo isso é o *solver*, conforme será explicado na seção 5.1.

## 3 Proposta GRASP

*Greedy Randomized Adaptive Search Procedure* (GRASP) é uma metaheurística em que cada iteração consiste de duas fases: uma para a construção de uma solução inicial e outra que busca melhores soluções a partir desta (Feo & Resende, 1995). A solução geral é atualizada sempre que a fase de busca encontrar uma solução melhor que todas as outras já encontradas. O processo é repetido até um certo número de iterações ou até um critério de parada estabelecido ser satisfeito. Nos ultimos anos, o GRASP tem sido aplicado com sucesso em uma grande variedade de problemas de otimização (Resende & Ribeiro, 2005).

Neste trabalho, desenvolvemos um algoritmo GRASP para o BGEP. A entrada consiste de um grafo bipartido sem pesos $G = (U, V, E)$ com $m = |U|$ e $n = |V|$. Inicialmente, transformamos $G$ em uma clique $C = (U, F)$ com pesos $p_{ij}$, ou seja, um grafo composto pelos vértices da bipartição $U$ do grafo $G$ e um conjunto de arestas $F$ que ligam cada um desses vértices a todos os outros, com pesos $p_{ij}$ tal que $i, j \in U$. Cada peso estará representando a quantidade de vértices de $V$ que não são vizinhos de $i$ e $j$ ao mesmo tempo (é vizinho de um, mas não o é do outro). Os pesos $p_{ij}$ podem ser representados matematicamente da forma $p_{ij} = \sum_{k=1}^{n} ||e_{ik} - e_{jk}||$ tal que $e \in E$ (por $G$ ser bipartido, $k \in V$) e possui valor 1 caso exista esta aresta e 0 caso não exista.

Com base na clique $C$ obtemos a árvore geradora mínima $T$ por meio do algoritmo de *Kruskal*. Nesta árvore, cada aresta conterá os pares de vértices de $U$ mais similares entre si quanto à sua vizinhança. Portanto, esses vértices tem uma grande chance de estarem na mesma biclique.

Esta árvore $T$ fornecerá sementes para a criação das soluções iniciais no GRASP, explicado com detalhes na seção 3.1. Em seguida, inicia-se uma busca por melhores soluções com base nesta solução inicial encontrada, através do processo apresentado na seção 3.2. No algoritmo 1 apresentamos a visão geral da heurística.

**Algoritmo 1** Visão Geral da Heurística

**procedure** MST+BL($G, k$)
  $it \leftarrow 0$
  $\mu^* \leftarrow \infty$
  transforme o grafo $G$ numa clique $C$ com pesos $d_{ij}$
  calcule a árvore geradora de custo mínimo $T$ de $C$
  **repeat**
    $C_{it} \leftarrow$ CONSTRUTIVO($T, k$) ▷ $k = 1$, guloso; $k = m$, aleatório; $1 < k < m$, $k$-melhores
    $C'_{it} \leftarrow$ BUSCALOCAL($C_{it}$)
    $\mu_{it} \leftarrow$ NUMERROS($C'_{it}$) ▷ número de edições (remoção ou adição de arestas)
    **if** $\mu_{it} < \mu^*$ **then**
      $C^* \leftarrow C'_{it}$
    **end if**
    $it \leftarrow it + 1$
  **until** $it = 1 + (k-1) \cdot m$
  **return** $C^*$
**end procedure**

### 3.1 Método Construtivo

Dado como entrada a árvore geradora mínima $T$, as soluções iniciais de cada iteração do GRASP são construídas com base em remoções aleatórias das arestas desta árvore. Na floresta geradora resultante, cada árvore definirá um agrupamento $U_i$ para compor a biclique.

Com base nisso, para cada vértice de $V$ verifica-se em qual subconjunto de $U_i$ deverá fazer parte, de maneira a conter o menor número de erros. Isto é feito associando o vértice em questão a cada uma dos conjuntos de $U_i$ e verificando o quanto de remoções e adições de arestas são necessárias para que esse vértice se ligue com todos os vértices do conjunto e apenas estes.

O agrupamento de $U$ em que seja necessário o menor número de edições será o ideal. Em seguida, combinamos $U$ e $V$ para formarmos a biclusterização $\{U, V\}$ ligando as arestas apenas quando houver no grafo $G$. No algoritmo 2 apresentamos o método construtivo.

**Algoritmo 2** Método Construtivo

**procedure** CONSTRUTIVO($T, k$)
  **while** $E(T) \neq \emptyset$ **do**
    $\mu \leftarrow \{\infty, \cdots, \infty\}_{1 \times |E(T)|}$
    **for all** $i \in E(T)$ **do**
      $T_i \leftarrow T - \{i\}$
      construa $U_i$ a partir de $T_i$ ▷ um bicluster para cada componente conexa de $T_i$
      construa $V_i$ ideal a partir de $U_i$ ▷ $V_i$ que tem menor número de erros dado $U_i$
      $C_i \leftarrow$ BICLUSTER($U_i, V_i$) ▷ combina os agrupamentos $U_i$ e $V_i$ para gerar o bicluster $C_i$
      $\mu_i \leftarrow$ NUMERROS($C_i$)
    **end for**
    escolha $\tilde{C}$ aleatoriamente entre os $k$ melhores $C_i$ ▷ beaseado nos valores em $\mu$
    atualiza $T$ com base no $\tilde{C}$ escolhido
    **if** $\tilde{\mu} < \mu^*$ **then**
      $C^* \leftarrow \tilde{C}$
    **end if**
  **end while**
  **return** $C^*$
**end procedure**

### 3.2 Busca Local

Dada uma solução para o problema, nesta fase realiza-se uma busca por melhores soluções a cada iteração do GRASP. Esta busca consiste em alternadamente inverter o processo de construção da biclusterização (descrito na seção 3.1). Em vez de tomarmos o agrupamento de $U$ como entrada, tomaremos o de $V$, e para cada vértice de $U$ verifica-se em qual agrupamento de $U$ deve-se ficar de modo a minimizar o número de erros.

Como resultado deste procedimento, teremos uma outra biclusterização $C$. Repetimos novamente este processo com $C$, e caso a resultante $C'$ tenha o agrupamento de $U$ igual ao de $C$ ou tenha um número de erros não melhor que a da iteração anterior a busca para. No algoritmo 3 apresentamos a método de busca local empregado.

**Algoritmo 3** Busca Local

**procedure** BUSCALOCAL($C_0$)
  $U'_0 \leftarrow U_0$; $C^* \leftarrow C_0$; $\mu_0 \leftarrow \mathsf{NumErros}(C_0)$; $it \leftarrow 1$ ▷ $C_0 = \mathsf{BiCluster}(U_0, V_0)$
  **repeat**
    $U_{it} \leftarrow U'_{it-1}$
    construa $V_{it}$ ideal a partir de $U_{it}$
    $C_{it} \leftarrow$ BICLUSTER($U_{it}, V_{it}$)
    $\mu_{it} \leftarrow$ NUMERROS($C_{it}$)
    construa $U'_{it}$ ideal a partir de $V_{it}$
    $C'_{it} \leftarrow$ BICLUSTER($U'_{it}, V_{it}$)
    $\mu'_{it} \leftarrow$ NUMERROS($C'_{it}$)
    **if** $\mu_{it} < \mu_{it-1}$ **then**
      $C^* \leftarrow C_{it}$
    **end if**
    **if** $\mu'_{it} < min\{\mu_{it}, \mu_{it-1}\}$ **then**
      $C^* \leftarrow C'_{it}$
    **end if**
    $it \leftarrow it + 1$
  **until** $U'_{it} = U_{it}$
  **return** $C^*$
**end procedure**

## 4 Resultados Teóricos

Nesta seção provaremos o teorema que nos ajudará na criação do pré-processamento e o definiremos a seguir.

**Teorema 4.1.** *Sejam $a$ e $b$ vértices quaisquer em um grafo bipartido $G(V, U, E)$ e $d(a, b)$ a distância entre eles. Se $d(a, b) \geq 4$. Então existe uma solução ótima na qual $a$ e $b$ pertencem a bicliques distintas.*

**PROVA.** Devemos provar que sempre será mais custoso manter $a$ e $b$ na mesma biclique, quando $d(a, b) \geq 4$, do que mantê-las em bicliques distintas. Para tal, definiremos duas funções $rem(X, Y)$ e $adc(X, Y)$.

Sejam $X \subseteq V$ e $Y \subseteq U$. A função $rem(X, Y)$ representa o custo de remover todas as arestas entre $X$ e $Y$. A função $adc(X, Y)$ representa o custo de adicionar todas as arestas entre $X$ e $Y$ que faltam para criar uma biclique $B = \{X, Y\}$. Com isso, definimos também $N(a)$ como a vizinhança de $a$ e $N^2(a) = \{v \in V | d(a, v) = 2\}$. Para os casos em que $d(a, b) = \infty$, ou seja, não existe um caminho entre eles, então eles já estão em bicliques diferentes. Restam então, dois possíveis casos: $a$ e $b$ na mesma partição (Caso 1) e em partições diferentes (Caso 2). O caso em que $a$ e $b$ estão na mesma partição é dividido em dois sub-casos: $d(a, b) > 4$ (Caso 1a) e $d(a, b) = 4$ (Caso 1b).

Nesta prova será considerado a remoção ou inclusão de $N(a)$, $N(b)$, $N^2(a)$ e $N^2(b)$ nos *bicluster* de forma integral, já que adicionar/remover apenas algumas arestas continuará sendo uma solução pior do que a apresentada em cada caso.

**Caso 1a:**

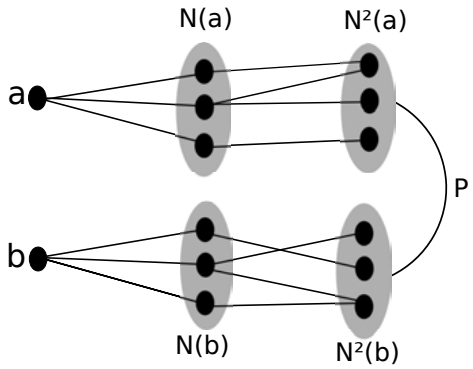



Figura 3: Caso 1a.

Neste caso temos que $a$ e $b$ pertencem a mesma partição e $N^2(a) \cap N^2(b) = \emptyset$. Ou seja, $d(a,b) > 4$, em que $P$ representa um caminho qualquer entre $N^2(a)$ e $N^2(b.)$

- Para criar o *bicluster* $B = \{a, b, N(a), N(b)\}$ temos o custo:

$$\begin{aligned} &adc(a, N(b)) + adc(b, N(a)) + rem(N(a), N^2(a)) + rem(N(b), N^2(b)) \\ &= |N(b)| + |N(a)| + rem(N(a), N^2(a)) + rem(N(b), N^2(b)). \end{aligned}$$

  Observe que se $N^2(a)$ ($N^2(b)$) fizesse parte da biclique teríamos um custo adicional $adc(N^2(b), N(a))$.

  Porém, apenas a criação das bicliques $\{a, N(a)\}$ e $\{b, N(b)\}$ tem custo igual a $rem(N(a), N^2(a)) + rem(N(b), N^2(b))$.

- Para criar o *bicluster* $B = \{a, b, N(a)\}$ temos o custo:

$$adc(b, N(a)) + rem(N(a), N^2(a)) + rem(b, N(b))$$

  Porém, apenas a criação das bicliques $\{a, N(a)\}$ e $\{b\}$ tem custo igual a $rem(N(a), N^2(a)) + rem(b, N(b))$ (de forma similar a $B = \{a, b, N(a)\}$).

**Caso 1b:**

Neste caso temos que $a$ e $b$ pertencem a mesma partição e que $N^2(a) \cap N^2(b) = C$. Ou seja, $d(a,b) = 4$.

- Para criar o *bicluster* $B = \{a, b, N(a), N(b), C\}$ temos o custo:

$$\begin{aligned} &adc(b, N(a)) + adc(a, N(b)) + rem(N(a), N^2(a)) + \\ &+ rem(N(b), N^2(b)) + adc(C, N(a)) + adc(C, N(b)) \\ &= |N(a)| + |N(b)| + rem(N(a), N^2(a)) + rem(N(b), N^2(b)) + \\ &+ adc(C, N(a)) + adc(C, N(b)). \end{aligned}$$

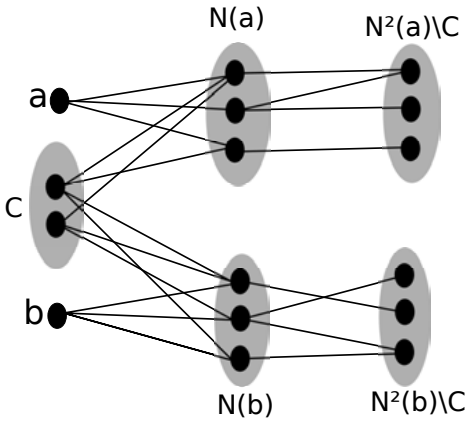



Figura 4: Caso 1b.

Por outro lado, sem perda de generalidade, se $|N(b)| \leq |N(a)|$ então o *biclusters* $\{a, N(a), N(b), C\}$ e $\{b\}$ teriam o custo:

$$\begin{aligned} & rem(b, N(b)) + adc(a, N(b)) + rem(N(a), N^2(a)) \\ & + rem(N(b), N^2(b)) + adc(C, N(a)) + adc(C, N(b)) \\ & = |N(b)| + |N(b)| + rem(N(a), N^2(a)) + rem(N(b), N^2(b)) + \\ & + adc(C, N(a)) + adc(C, N(b)). \end{aligned}$$

- De forma semelhante para criar o *bicluster* $B = \{a, b, N(a), N(b)\}$ temos o custo:

$$\begin{aligned} & adc(b, N(a)) + adc(a, N(b)) + rem(N(a), N^2(a)) \\ & + rem(N(b), N^2(b)) + rem(C, N(a)) + rem(C, N(b)) \\ & = |N(a)| + |N(b)| + rem(N(a), N^2(a)) + \\ & + rem(N(b), N^2(b)) + rem(C, N(a)) + rem(C, N(b)). \end{aligned}$$

Ainda considerando $|N(b)| \leq |N(a)|$, os *biclusters* $\{a, N(a), N(b)\}$ e $\{b\}$ teriam o custo:

$$\begin{aligned} & rem(b, N(b)) + adc(a, N(b)) + rem(N(a), N^2(a)) \\ & + rem(N(b), N^2(b)) + rem(C, N(a)) + rem(C, N(b)) \\ & = |N(b)| + |N(b)| + rem(N(a), N^2(a)) + \\ & + rem(N(b), N^2(b)) + rem(C, N(a)) + rem(C, N(b)). \end{aligned}$$

Assim, $a$ e $b$ em bicliques distintas possui um custo menor.

**Caso 2:**

Neste caso, temos que $a$ e $b$ pertencem a partições diferentes, tentaremos então, colocá-las na mesma biclique. Analisemos alguns casos para esta biclique:

- $B = \{a, b\}$

O custo desta biclique será:
$adc(a, b) + rem(N(b), b) + rem(N(a), a) = 1 + |N(b)| + |N(a)|$.

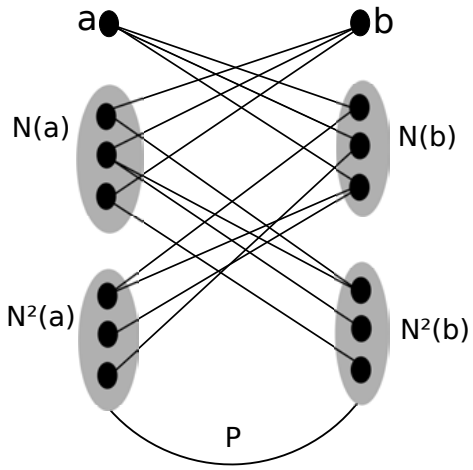



Figura 5: Caso 2.

Porém separar em $\{a\}$ e $\{b\}$ possui um custo menor:
$rem(N(b), b) + rem(N(a), a) = |N(b)| + |N(a)|$.

- $B = \{a, b, N(a)\}$

  O custo desta biclique será:
  $adc(a, b) + rem(N(b), b) + rem(N(a), N^2(a)) = 1 + |N(b)| + rem(N(a), N^2(a))$.

  Porém separar em $\{a, N(a)\}$ e $\{b\}$ possui um custo menor:
  $rem(N(b), b) + rem(N(a), N^2(a)) = |N(b)| + rem(N(a), N^2(a))$ (de forma similar a $B = \{a, b, N(b)\}$).

- $B = \{a, b, N(a), N(b)\}$

  O custo desta biclique será:
  $adc(a, b) + adc(N(a), N(b)) + rem(N(a), N^2(a)) + rem(N(b), N^2(b)$.

  Porém separar em $\{a, N(a)\}$ e $\{b, N(b)\}$ possui um custo menor:
  $rem(N(a), N^2(a)) + rem(N(b), N^2(b))$.

- $B = \{a, b, N(a), N^2(a)\}$

  O custo desta biclique será:
  $adc(a, b) + rem(b, N(b)) + adc(N(a), N^2(a)) + adc(b, N^2(a)) + rem(N^2(a), P)$.

  Porém separar em $\{a, N(a), N^2(a)\}$ e $\{b\}$ possui um custo menor:
  $rem(b, N(b)) + adc(N(a), N^2(a)) + rem(N^2(a), P)$.

- $B = \{a, b, N(a), N(b), N^2(a)\}$

  O custo desta biclique será:
  $adc(a, b) + adc(N(a), N(b)) + adc(N(a), N^2(a)) + rem(N(b), N^2(b)) + rem(N^2(a), P)$.

  Porém separar em $\{a, N(a), N^2(a)\}$ e $\{b, N(b)\}$ possui um custo menor:
  $adc(N(a), N^2(a)) + rem(N(b), N^2(b)) + rem(N^2(a), P)$.

- $B = \{a, b, N(a), N(b), N^2(a), N^2(b)\}$

  O custo desta biclique será:
  $adc(a,b) + adc(N(a), N(b)) + adc(N(a), N^2(a)) + adc(N(b), N^2(b)) +$
  $+adc(N^2(a), N^2(b)) + rem(N^2(a), P) + rem(N^2(b), P)$.

  Porém separar em $\{a, N(a), N^2(a)\}$ e $\{b, N(b), N^2(b)\}$ possui um custo menor:
  $adc(N(a), N^2(a)) + adc(N(b), N^2(b)) + rem(N^2(a), P) + rem(N^2(b), P)$.

□

Como já dito na seção 2, o resultado do teorema será a base de um pré-processamento. Com isso poderemos fixar vértices em bicliques diferentes antes de executar o B&C.

Na prática, será calculada a distância entre cada par de vértices. Para os pares que atendam o teorema podemos fixar variáveis de decisão ou criar cortes. Caso $u$ e $v$ estejam na mesma partição, podemos simplesmente fixar a variável $x_{uv} = 1$. Já no caso em que $u$ e $v$ estejam na mesma partição $U$, introduziremos os cortes da forma: $x_{uw} + x_{vw} \geq 1, \forall w \in V$.

## 5 Experimentos Computacionais

Nesta seção mostraremos as ferramentas utilizadas, a forma que criamos as instâncias além dos resultados alcançados com o método exato e o GRASP.

### 5.1 Ferramentas

Como ferramenta utilizaremos o CPLEX, que é um dos pacotes de *software* de otimização linear mista mais utilizados na literatura. O CPLEX é responsável por gerenciar todo o processo e, se necessário, adiciona novos cortes. No nosso caso os únicos cortes que ele adicionará são as próprias restrições do modelo, já que iniciamos com o modelo vazio. Todas os métodos foram desenvolvidos em C++ (gcc-4.7.1) e executados numa máquina Intel Core i7-2600 com 3.40GHz e 32 Gb de memória RAM no sistema operacional Arch Linux 3.3.4. A versão utilizada do CPLEX foi a 12.3.

### 5.2 Instâncias

Como o problema é pouco explorado, não existem instâncias criadas na literatura. Dessa forma, criamos grafos bipartidos aleatórios utilizando o modelo $\mathbb{G}(m, n, p)$, também conhecido como modelo binomial. Com ele são construídos grafos bipartidos $G(U, V, E)$ em que $|U| = m$, $|V| = n$ e cada aresta de $E$ é incluida no grafo com probabilidade $p$, independente das outras arestas. Neste modelo, o número de arestas de um grafo $G$, $E_G = |E(\mathbb{G}(m, n, p))|$, é uma variável aleatória com esperança $\mathbb{E}(E_G) = mnp$.

Inicialmente com base no trabalho de Bastos (2012), para o *Cluster Editing Problem*, criamos 20 instâncias seguindo o modelo binomial com $p = \{0,2; 0,4; 0,6; 0,8\}$ e tamanhos $m$ e $n$ aleatórios entre 10 e 30. No entanto, como veremos na seção 5.3, as instâncias mais difíceis foram as criadas utilizando $p = 0,4$. Então criamos mais 10 instâncias com $p = 0,4$ para analisarmos o comportamento dos métodos desenvolvidos nestas instâncias.

### 5.3 Exato

Na tabela 1 temos o número de variáveis fixadas (VF) e cortes gerados (CG) usando o teorema 4.1, além do tempo computacional e do valor ótimo. No nome de cada instância temos a

identificação da instância juntamente com o tamanho de cada partição e a probabilidade de existência de arestas, por exemplo 10x11z_i1_p0,4 significa que a instância 1 possui $|U| = 10$, $|V| = 11$ e $p = 0,4$. Como criamos instância com diferentes probabilidades de existir aresta, o resultado do pré-processamento se mostra mais eficaz em instâncias esparsas.

Tabela 1: Resultados do B&C

| Instância | VF | CG | Ótimo | Tempo (s) | Instância | VF | CG | Ótimo | Tempo (s) |
|---|---|---|---|---|---|---|---|---|---|
| 10x11_i1_p0,4 | 15 | 291 | 17 | 0,04 | 10x11_i16_p0,6 | 0 | 30 | 26 | 0,35 |
| 10x11_i2_p0,4 | 2 | 208 | 20 | 0,09 | 10x14_i17_p0,6 | 0 | 0 | 40 | 2,57 |
| 10x18_i3_p0,4 | 0 | 366 | 36 | 7,24 | 11x11_i18_p0,6 | 0 | 0 | 34 | 2,87 |
| 11x13_i4_p0,4 | 0 | 48 | 34 | 4,25 | 12x14_i19_p0,6 | 0 | 0 | 49 | 18,97 |
| 11x15_i5_p0,4 | 0 | 217 | 35 | 11,22 | 12x15_i20_p0,6 | 0 | 12 | 51 | 161,51 |
| 11x16_i6_p0,4 | 0 | 404 | 30 | 0,52 | 15x25_i21_p0,8 | 0 | 0 | 88 | 0,73 |
| 11x18_i7_p0,4 | 0 | 187 | 50 | 743,90 | 15x26_i22_p0,8 | 0 | 0 | 74 | 0,39 |
| 12x13_i8_p0,4 | 6 | 404 | 28 | 1,52 | 16x16_i23_p0,2 | 56 | 2112 | 24 | 0,22 |
| 12x13_i9_p0,4 | 0 | 176 | 34 | 13,32 | 16x18_i24_p0,2 | 77 | 2726 | 24 | 0,11 |
| 12x19_i10_p0,4 | 0 | 230 | 52 | 1479,82 | 16x29_i25_p0,8 | 0 | 0 | 94 | 1,06 |
| 12x19_i11_p0,4 | 0 | 168 | 53 | 3132,17 | 17x18_i26_p0,2 | 98 | 3002 | 23 | 0,10 |
| 13x19_i12_p0,4 | 0 | 84 | 61 | 29188,78 | 17x29_i27_p0,8 | 0 | 0 | 111 | 1,84 |
| 15x16_i13_p0,4 | 0 | 267 | 54 | 2835,34 | 20x28_i28_p0,2 | 47 | 5452 | 62 | 35844,99 |
| 16x17_i14_p0,4 | 0 | 215 | 63 | 32335,47 | 21x22_i29_p0,2 | 32 | 3590 | 56 | 14575,64 |
| 16x17_i15_p0,4 | 0 | 217 | 61 | 9090,40 | 21x22_i30_p0,8 | 0 | 0 | 56 | 0,86 |

### 5.4 GRASP

Na tabela 2 temos o número de edições encontradas pelo algoritmo apresentado, o GAP em relação ao ótimo apresentado na tabela 1 e o tempo computacional. O GAP apresentado na tabela é relativo ao resultado em que houve o menor número de edições, além disso a média dos tempos é apresentado.

Tabela 2: Resultados do GRASP

| Instância | Edições | GAP (%) | Tempo (s) | Instância | Edições | GAP (%) | Tempo (s) |
|---|---|---|---|---|---|---|---|
| 10x11_i1_p4 | 17 | 0,00 | 7,23 | 10x11_i16_p6 | 28 | 7,14 | 3,40 |
| 10x11_i2_p4 | 20 | 0,00 | 7,00 | 10x14_i17_p6 | 40 | 0,00 | 4,02 |
| 10x18_i3_p4 | 36 | 0,00 | 10,09 | 11x11_i18_p6 | 35 | 2,86 | 5,26 |
| 11x13_i4_p4 | 34 | 0,00 | 9,97 | 12x14_i19_p6 | 49 | 0,00 | 5,28 |
| 11x15_i5_p4 | 36 | 2,78 | 12,90 | 12x15_i20_p6 | 52 | 1,92 | 7,05 |
| 11x16_i6_p4 | 30 | 0,00 | 13,89 | 15x25_i21_p8 | 88 | 0,00 | 31,80 |
| 11x18_i7_p4 | 51 | 1,96 | 7,88 | 15x26_i22_p8 | 74 | 0,00 | 24,27 |
| 12x13_i8_p4 | 28 | 0,00 | 13,13 | 16x16_i23_p2 | 24 | 0,00 | 26,45 |
| 12x13_i9_p4 | 34 | 0,00 | 12,53 | 16x18_i24_p2 | 24 | 0,00 | 25,56 |
| 12x19_i10_p4 | 53 | 1,89 | 20,31 | 16x29_i25_p8 | 94 | 0,00 | 26,48 |
| 12x19_i11_p4 | 54 | 1,85 | 23,41 | 17x18_i26_p2 | 23 | 0,00 | 43,42 |
| 13x19_i12_p4 | 62 | 1,61 | 24,50 | 17x29_i27_p8 | 111 | 0,00 | 51,80 |
| 15x16_i13_p4 | 54 | 0,00 | 47,19 | 20x28_i28_p2 | 63 | 1,59 | 77,30 |
| 16x17_i14_p4 | 64 | 1,56 | 48,24 | 21x22_i29_p2 | 56 | 0,00 | 71,91 |
| 16x17_i15_p4 | 61 | 0,00 | 65,95 | 21x22_i30_p8 | 97 | 0,00 | 77,21 |

## 6 Conclusão

Vimos que a resolução do BGEP através de métodos exatos é satisfatória em instâncias de 21 a 48 vértices. Percebemos que é viável a integração do teorema 4.1 com o método exato, já que cortes foram gerados e variáveis fixadas. Apesar do teorema ter apresentado fixações de variáveis em poucos casos, isso pode ser justificado pela densidade das instâncias utilizadas. Instâncias

densas tendem a ter uma número menor de fixações, pois há poucos vértices com distâncias menores que 4, diferentemente do que ocorre em instâncias esparsas.

O GRASP apresentou um GAP relativamente pequeno, chegando ao ótimo em 20 das 30 instâncias, com GAP médio de 0,84% e GAP máximo de 7,14%, na única instância em que houve uma diferença de duas edições para o ótimo. Em outras 9 instâncias, houve uma diferença de uma edição. Além disso, o algoritmo executa em um tempo computacional baixo.

Para trabalhos futuros iremos propor melhorias que acelerem o *branch*, como por exemplo uma heurística ou incorporá-lo à meta-heurística GRASP. Faremos um estudo mais aprofundado com relação às instâncias mais difíceis. Com relação a meta-heurística, proporemos novos métodos construtivos e novas buscas locais

## Referências

**Abdullah, A. & Hussain, A.** (2006), 'A new biclustering technique based on crossing minimization', *Neurocomputing* **69**(16–18), 1882 – 1896.

**Alpert, C. J. & Kahng, A. B.** (1995), 'Recent directions in netlist partitioning: a survey', *Integration, the VLSI Journal* **19**, 1–81.

**Amit, N.** (2004), The bicluster graph editing problem, Master's thesis, Tel Aviv University.

**Bastos, L.** (2012), Novos Algoritmos e Resultados Teóricos para o Problema de Particionamento de Grafos por Edição de Arestas, PhD thesis, Universidade Federal Fluminense.

**Bisson, G. & Hussain, F.** (2008), Chi-sim: A new similarity measure for the co-clustering task, *in* 'Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications', ICMLA '08, IEEE Computer Society, Washington, DC, USA, pp. 211–217.

**Cheng, Y. & Church, G. M.** (2000), Biclustering of expression data, *in* 'Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology', AAAI Press, pp. 93–103.

**Faure, N., Chretienne, P., Gourdin, E. & Sourd, F.** (2007), 'Biclique completion problems for multicast network design', *Discrete Optimization* **4**(3-4), 360–377.

**Feo, T. & Resende, M.** (1995), 'Greedy randomized adaptive search procedures', *Journal of Global Optimization* **6**, 109–133.

**Guo, J., Hüffner, F., Komusiewicz, C. & Zhang, Y.** (2008), Improved algorithms for bicluster editing, *in* '5th International Conference on Theory and Applications of Models of Computation', Vol. 4978, Springer Berlin Heidelberg, pp. 445–456.

**Gupta, A. & Palit, A.** (1979), 'Clique generation using boolean equations', *Proceedings of The IEEE* **67**(1), 178–180.

**Hartigan, J. A.** (1975), *Clustering Algorithms*, John Wiley and Sons.

**Pothen, A.** (1997), 'Graph partitioning algorithms with applications to scientific computing', *ICASE LaRC Interdisciplinary Series in Science and Engineering* **4**, 323–368.

**Resende, M. & Ribeiro, C.** (2005), *Metaheuristics: Progress as real problem solvers*, Springer, New York, chapter GRASP whit path-relinking: recent advances and applications, pp. 29–63.

**Wolsey, L. A.** (1998), *Integer Programming*, Wiley-Interscience, Series in Discrete Mathematics Optimization.