# Um Algoritmo Eficiente para o Problema de Biclusterização em Grafos

**Rian G. S. Pinheiro[1,2] , Ivan César Martins[1] , Fábio Protti[1] , Luiz Satoru Ochi [1]**

[1]Instituto de Computação – Universidade Federal Fluminense
Niterói – RJ – Brasil
[2]Universidade Federal Rural de Pernambuco
Garanhuns – PE – Brasil

{rgpinheiro,imatins,fabio,satoru}@ic.uff.br

## RESUMO

Neste trabalho é proposta uma meta-heurística GRASP com (e sem) módulo exato baseado em Set Partitioning (GRASP+SP) para o *Bicluster Graph Editing Problem* (BGEP). O BGEP é um problema de biclusterização semelhante ao *Cluster Graph Editing*, no entanto é menos explorado pela literatura. Além do GRASP, é proposto um pré-processamento que identifica bicliques que serão componentes conexas na solução ótima do BGEP. Com o resultado foi visto que tanto o GRASP simples encontra a solução ótima em todas as instâncias da literatura, com exceção de uma e que o GRASP+SP encontra a solução ótima em todas as instâncias. Além disso, em ambos os métodos o tempo computacional é baixo comparado ao da literatura.
**PALAVRAS CHAVE**. Biclusterização. Meta-heurísticas. Matheuristic.
**Área principal** OC - Otimização Combinatória.

## ABSTRACT

We present a GRASP metaheuristic and a GRASP with a Set Partition method (GRASP+SP) for the Bicluster Graph Editing Problem (BGEP). The BGEP is similar to the Cluster Graph Editing but less explored in the literature. We also proposed a new preprocessing that identifies bicliques that will be connected components in the optimal solution of the BGEP. The proposed algorithms are compared with some available algorithms in the literature, presenting meaningful results in a reasonable computational time.
**KEYWORDS**. Biclustering. Metaheuristics. Matheuristic.
**Main area** OC - Combinatorial Optimization.

## 1 Introdução

O *Bicluster Graph Editing Problem* (BGEP) é um problema $\mathcal{NP}$-completo em que, dado um grafo bipartido $G = (V, U, E)$ e um inteiro $k \geq 0$, sendo $V$ e $U$ conjuntos não vazios de vértices e $E$ um conjunto de arestas, tem como objetivo adicionar ou remover no máximo $k$ arestas de forma a tornar $G$ uma união disjunta de subgrafos bipartidos completos maximais (bicliques).

Um problema semelhante é o CLUSTER GRAPH EDITING PROBLEM, primeiramente estudado por Gupta & Palit (1979). Ele consiste em tornar $G$ uma união disjunta de subgrafos completos (cliques), neste problema o grafo de entrada $G$ não é necessariamente bipartido (Bastos et al., 2014). Ambos são casos de problemas de partição em grafos.

No trabalho de Pothen (1997), são descritos vários tipos de algoritmos de particionamento e aplicações em diversas áreas da ciência da computação. Já 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 só foi introduzido em meados da década de 70 por Hartigan (1975). Já o seu primeiro uso se deu no contexto da biologia computacional através do 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). Outros nomes como co-clusterização, clusterização bidimensional, entre outros, são frequentemente utilizados na literatura para se referir ao mesmo problema. Em suma, pode-se dizer que este conceito denota a existência de submatrizes "significativas" em uma dada matriz, em outras palavras, um subconjunto de linhas e de colunas que possuem padrões únicos baseados em um certo método de classificação.

Existem diversas formulações e contextos para problemas de biclusterização. Por exemplo, pode-se 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.

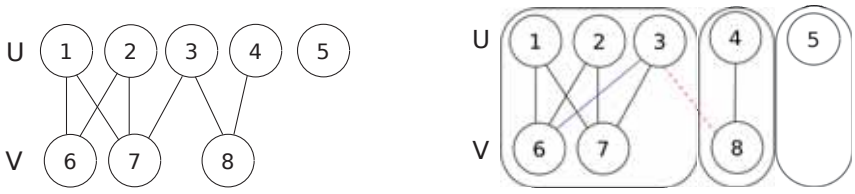
Em um grafo bipartido sem pesos, um *bicluster* ideal é um subgrafo bipartido completo (biclique). Pode-se dizer que a presença de bicliques indica um alto grau de similaridade entre os dados agrupados. Em especial, chama-se de grafo *bicluster* (ou biclusterizado) o grafo bipartido no qual cada uma de suas componentes conexas forma uma biclique.

Este trabalho foca na versão de otimização de um dos diversos problemas de biclusterização apresentados e definidos em Amit (2004), o BGEP. 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.

**PROBLEMA DE BICLUSTERIZAÇÃO POR EDIÇÃO DE ARESTAS**
INSTÂNCIA: Um grafo bipartido $G = (U, V, E)$ e um inteiro $k \geq 0$.
QUESTÃO: É possível realizar no máximo $k$ edições de arestas (remoções e adições) de forma a tornar $G$ um grafo *bicluster*?

A Figura 1 mostra um exemplo de um grafo que com uma adição, aresta $(3, 6)$, e uma remoção, aresta $(3, 8)$, tornam-o em um grafo biclusterizado. Observe que esta não é a solução ótima, uma vez que bastaria uma edição, remover a aresta $(3, 7)$, para torná-lo um grafo *bicluster*. Além disso, considera-se um único vértice como uma biclique, representada na figura pelo vértice 5.

Amit (2004) provou que o BGEP (em sua versão de decisão) é um problema $\mathcal{NP}$-completo, a partir de uma redução polinomial do problema 3-EXACT 3-COVER, e propõe uma formulação em programação inteira binária e um algoritmo 11-aproximado para o problema. Já o trabalho




(a) Instância. (b) Solução.

Figura 1: Exemplo do BGEP.

de Protti et al. (2006) propõe um algoritmo que usa uma nova estratégia baseada em técnicas de decomposição modular. O algoritmo resolve o problema em tempo $O(4^k + n + m)$, em que $k$ é o número de edições no grafo, $n$ é o número de vértices e $m$ o número de arestas. Com base no trabalho de Amit (2004), Guo et al. (2008) propõem um algoritmo randômico 4-aproximado.

Com relação à meta-heurísticas, Pinheiro et al. (2012) e Sousa Filho et al. (2012) abordam o problema através do método GRASP. Em Sousa Filho et al. (2012) comparam seu GRASP com as heurísticas da literatura e propõem uma regra de redução. Já Pinheiro et al. (2012), o GRASP é comparado com um método exato adicionado ao pré-processamento descrito neste trabalho.

### 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 de forma a satisfazer um critério de homogeneidade, ou seja, genes dentro de um agrupamento devem ter uma alta semelhança entre si.

A análise de dados biológicos tem como entrada uma matriz gene (linha) $\times$ característica ou fenótipo (coluna). As características podem ser de indivíduos diferentes ou até de diferentes condições experimentais do mesmo indivíduo, como por exemplo, tecidos diferentes: um canceroso e um saudável. A linha de um determinado gene é chamada de padrão de expressão do referido gene. Já a coluna é chamada de perfil de expressão da condição. Com a matriz gene–característica cria-se um grafo bipartido onde os genes e as características serão as partes da bipartição. Isto transforma o problema do agrupamento de genes no BGEP.

Outra aplicação para o BGEP se dá em sessões de *multicast*. Uma sessão de *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 sessões.

Um exemplo de sessão *multicast* é TV sobre IP. Nele, um subconjunto de clientes pode requerer o mesmo canal ou o mesmo subconjunto de canais, durante um dado período de tempo. Faure et al. (2007) mostram que uma variante do BGEP ajuda no projeto de uma rede *multicast*. Neste problema, as partes da bipartição são formadas pelos conjuntos de clientes e sessões.

Por fim, o BGEP também pode ser aplicado no contexto de mineração de dados. Dados são normalmente armazenados em uma base de dados (ou uma matriz de dados) na qual cada registro possui uma coluna de atributos (Abdullah & Hussain, 2006). Em aplicações com centenas e milhares de registros, uma alternativa para descobrir novas informações e conhecimento pode ser feito através da biclusterização.

## 2 Pré-Processamento para o BGEP

Nesta seção é proposto um pré-processamento que identifica bicliques que serão componentes conexas na solução ótima do BGEP. O pré-processamento é uma aplicação direta do Teorema 2.1, apresentado a seguir.

Para o entendimento do teorema, considere a seguintes notações. Sejam $X \subseteq V$ e $Y \subseteq U$; a função $rem(X,Y) = |\{xy \in E \mid x \in X \text{ e } y \in Y\}|$, representa o custo da remoção de todas as arestas em $E$ entre os conjuntos $X$ e $Y$. Da mesma forma, $add(X,Y) = |X||Y| - rem(X,Y)$, representa o custo de adicionar as arestas restantes entre $X$ e $Y$ no intuito de criar a biclique $B$ com os vértices $V(B) = X \cup Y$. Nos casos em que $X = \{x\}$, será utilizada a notação $add(x,Y)$ e $rem(x,Y)$ no lugar de $add(\{x\},Y)$ e $rem(\{x\},Y)$, da mesma forma para os casos em que $Y = \{y\}$. Por fim, $N(B)$ representa os vértices que possuem pelo menos um vizinho em $B$ e não estão em $B$.

**Teorema 2.1.** *Sejam a biclique $B$ e o corte $C$ contidos em um grafo bipartido $G(U,V,E)$ de tal forma que $C = \{uv|u \in B \text{ e } v \in N(B)\}$ e $t_{\min} = |B \cap U| < |B \cap V|$. Se $t_{\min} > |C|$ e $add(i,B) > rem(i,B) \quad \forall i \in N(B)$ então existirá uma solução ótima em que $B$ será uma componente conexa.*

*Demonstração.* Suponha que a solução ótima tenha uma biclique da forma $B \cup N_B$, tal que $N_B \subseteq N(B)$. O custo $c_1$ para formar $B \cup N_B$ é definido por:

$$add(B,N_B) + rem(B, N(B) \setminus N_B) + rem(N_B, (U \cup V) \setminus B)$$

Porém, o custo $c_2$ para formar $B$ é apenas:

$$rem(B, N(B)) = |C|$$

Como $add(i,B) > rem(i,B) \quad \forall i \in N(B)$ então $add(B,N_B) + rem(B, N(B) \setminus N_B) > rem(B,N(B))$

Agora suponha que a solução ótima é da forma $B' \cup N_B$ em que $B' \subseteq B$. O custo de $B' \cup N_B$ vale pelo menos $t_{\min}$, mas $t_{min} > |C|$, logo o custo de formar $B$ sozinho é menor. □

Como o Teorema 2.1, pode-se determinar algumas bicliques que podem ser removidas do grafo de entrada. Para identificá-las, é proposto o seguinte modelo matemático que encontra a maior biclique (em número de arestas) que atende ao teorema.

$$\max \quad \sum_{i \in U} \sum_{j \in V} x_{ij} \tag{1}$$

$$y_i \leq \sum_{j \in V} x_{ij} \qquad \forall i \in U \tag{2}$$

$$y_j \leq \sum_{i \in U} x_{ij} \qquad \forall j \in V \tag{3}$$

$$y_i + y_j - x_{ij} \leq 1 \qquad \forall (ij) \in E \tag{4}$$

$$x_{ij} - y_i \leq 0 \qquad \forall (ij) \in E \tag{5}$$

$$x_{ij} - y_j \leq 0 \qquad \forall (ij) \in E \tag{6}$$

$$x_{ij} = 0 \qquad \forall (ij) \notin E \tag{7}$$

$$(1+\delta(i))y_i + \sum_{(ij)\notin E} y_j \geq \sum_{(ij)\in E} y_j + 1 \qquad \forall i \in U \qquad (8)$$

$$(1+\delta(j))y_j + \sum_{(ij)\notin E} y_i \geq \sum_{(ij)\in E} y_j + 1 \qquad \forall j \in V \qquad (9)$$

$$v_i \geq \sum_{(ij)\in E} y_j - \delta(i)y_i \qquad \forall i \in U \qquad (10)$$

$$v_j \geq \sum_{(ij)\in E} y_i - \delta(j)y_j \qquad \forall j \in V \qquad (11)$$

$$\sum_{i\in U} v_i + \sum_{j\in V} v_j \leq \sum_{j\in V} y_j \qquad (12)$$

$$\sum_{i\in U} v_i + \sum_{j\in V} v_j \leq \sum_{i\in V} y_i \qquad (13)$$

$$x_{ij}, y_i, y_j \in \mathbb{B} \qquad \forall (ij) \in U \times V \qquad (14)$$

$$v_i, v_j \in \mathbb{N} \qquad \forall (ij) \in U \times V \qquad (15)$$

Nesta formulação são utilizadas três famílias de variáveis: $x_{ij}$ que representa se a aresta $ij$ pertence ou não a solução final, $y_i$ que representa se o vértice $i$ pertence ou não a solução final e $v_i$ que é uma variável inteira que representa o "corte" no vértice $i$. A seguir, a função de cada restrição será detalhada.

(1) Função Objetivo, maximiza o número de arestas.

(2) Se não há aresta incidente ao vértice $i$ na solução então $y_i$ vale 0.

(3) Se não há aresta incidente ao vértice $j$ na solução então $y_j$ vale 0.

(4) Se $i$ e $j$ pertencem a biclique então a aresta $x_{ij}$ também estará.

(5) Se a aresta $x_{ij}$ está na solução então o vértice $i$ também estará.

(6) Se a aresta $x_{ij}$ está na solução então o vértice $j$ também estará.

(7) Se a aresta $ij$ não pertence ao grafo original então $x_{ij}$ vale 0.

(8) Representa a restrição $add(i, B) > rem(i, B)$, $add$ é o 1º somatório e $rem$ é o 2º. Já $\delta(i)$ representa o grau do vértice $i$, a restrição não terá efeito se o vértice pertencer à biclique ($y_i = 1$).

(9) Mesmo significado que a família de restrições anterior, só que para a outra partição.

(10) O corte em $j$ é representado por $v_j$ deve ser no mínimo $rem$. Se $j$ estiver na biclique a diferença pelo grau faz com que a restrição não tenha efeito.

(11) Mesmo significado que a família de restrições anterior, só que para a outra partição.

(12) O corte deve ser menor que $t_{\min}$.

(13) Mesmo significado que a restrição anterior, só que para a outra partição.

(14) As variáveis $x_{ij}, y_i$ e $y_j$ são binárias.

(15) As variáveis $v_i$ e $v_j$ são inteiras.

Ao executar o modelo, as variáveis $y_i$ e $y_j$ irão determinar a maior biclique que atende ao Teorema 2.1.

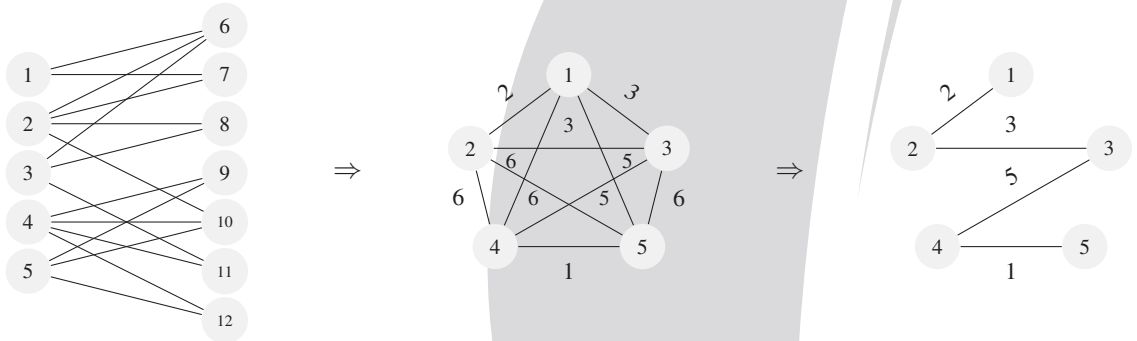
## 3 GRASP

*Greedy Randomized Adaptive Search Procedure* (GRASP) é uma meta-heurística 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 últimos anos, o GRASP tem sido aplicado com sucesso em uma grande variedade de problemas de otimização (Resende & Ribeiro, 2005).

Neste trabalho, o método construtivo será baseado de Pinheiro et al. (2012). A entrada consiste de um grafo bipartido sem pesos $G = (U, V, E)$ com $m = |U|$ e $n = |V|$. Inicialmente, transforma-se $G$ 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 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 $C$ obtém-se a árvore geradora mínima $T$. 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. O processo de criação da árvore $T$ pode ser observado na Figura 2.

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. O Algoritmo 3 apresenta a visão geral da heurística.




grafo de entrada $G$ | clique $C$ | árvore geradora $T$

Figura 2: Execução do Algoritmo Construtivo

### 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 em $V$, verifica-se em qual agrupamento $U_i$ deverá fazer parte, de maneira a conter o menor número de edições. 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 associado.

O agrupamento $U_i$ em que seja necessário o menor número de edições será o ideal e portanto escolhido. Em seguida, combinamos $U$ e $V$ para formarmos a biclusterização $\{U, V\}$ ligando as arestas no grafo $G$. No algoritmo 1 apresentamos o método construtivo.

**Algoritmo 1** 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 biclique 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 a biclique $C_i$
      $\mu_i \leftarrow$ NUMEDICOES($C_i$)
    **end for**
    escolha $\tilde{C}$ aleatoriamente entre os $k$ melhores $C_i$ ▷ baseado 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

Após o método construtivo, inicia-se o método de Busca Local baseado no VND (Mladenovic & Hansen, 1997). Tendo como entrada um solução inicial, a Busca Local tenta melhorar a solução através da exploração do espaço de solução. Dado um conjunto de estruturas de vizinhança, o método de Busca Local explora o espaço de solução trocando sistematicamente as estruturas de vizinhança. O método aceita somente soluções que melhorem a solução corrente. Nestes casos, o método volta à primeira estrutura de vizinhança e inicia uma nova busca. Usamos aqui uma variante do VND que consiste em usar uma ordenação aleatória dessas estruturas, como mostrado no Algoritmo 2.

O método proposto utiliza três estruturas de vizinhança: "Reagrupe"; "Mude de Biclique" e "Troca de Vértice". A complexidade computacional da busca em cada uma dessas vizinhanças é de $O(n^2)$. Elas são presentadas a seguir.

**Reagrupe:** Dado os conjuntos de grupos de vértices, formados pela intersecção de cada uma das partes, $U$ por exemplo, com cada biclique $B$ da solução atual. A Busca Local Reagrupe aloca os vértices da outra parte, no caso $V$, ao grupo de vértice de forma a minimizar o número de edições.

**Muda de Biclique:** Esta Busca Local move um vértice de uma biclique para outra.

**Troca de Vértice:** Esta Busca Local troca dois vértices (de mesma parte) de uma biclique para a outra. Ao contrário da "Mude de Biclique" não possibilita a eliminação de bicliques previamente existentes.

**Algoritmo 2** VND com ordem aleatórias

```
1: procedure VND(A)
2:     inicialize V randomicamente
3:     k ← 1
4:     μ* ← μ(A)
5:     repeat
6:         A' ← V^k(A)
7:         if μ(A') > μ* then
8:             A* ← A'
9:             k ← 1
10:        else
11:            k ← k + 1
12:        end if
13:    until k = k_max
14:    return A*
15: end procedure
```

**Algoritmo 3** Visão Geral da Heurística GRASP

```
1: procedure MST+BL(G, k)
2:     it ← 0
3:     μ* ← ∞
4:     transforme o grafo G numa clique C com pesos d_ij
5:     calcule a árvore geradora de custo mínimo T de C
6:     repeat
7:         C_it ← CONSTRUTIVO(T, k)        ▷ k = 1, guloso; k = m, aleatório; 1 < k < m, k-melhores
8:         C'_it ← BUSCALOCAL(C_it)
9:         μ_it ← NUMEDICOES(C'_it)        ▷ número de edições (remoção ou adição de arestas)
10:        if μ_it < μ* then
11:            C* ← C'_it
12:        end if
13:        it ← it + 1
14:    until it = 100
15:    return C*
16: end procedure
```

## 4 Módulo Híbrido baseado no Set Partitioning para o PBEA

Nesta seção, é proposto um método exato de refinamento baseado no Set Partitioning. Tendo como entrada um conjunto de soluções intermediárias válidas geradas por uma heurística do BGEP o refinamento retorna uma solução formada pela melhor combinação do conjunto de entrada.

Em outras palavras, considerando que uma heurística $H$ aplicada a uma entrada $G$ gerou um conjunto $S$ formado pelas soluções intermediárias válidas $S = \{G_1, G_2, \ldots, G_k\}$ e que cada solução intermediária pode ser particionada em um conjunto de bicliques $\{B_i^1, \ldots, B_i^{k_i}\}$. A união de todas essas bicliques será definida como: $\mathcal{B} = \{B_1, B_2, \ldots, B_\ell\}$. Note que cada biclique $B_i$ é candidata a estar na solução final. Além disso, note que todas as bicliques repetidas de $\mathcal{B}$ podem ser descartadas.

A ideia do refinamento consiste em selecionar um conjunto de bicliques de $\mathcal{B}$ que formem uma partição de $V(G) \cup U(G)$. Seja $\mathcal{B}_i \subseteq \mathcal{B}$ o conjunto de todas as bicliques que contenham o vértice $i$. Define-se $y_j$ como uma variável binária associada à biclique $B_j \in \mathcal{B}$, e $c_j = custo(B_j)$ é definida como o custo relativo à biclique $B_j$. Com isso, o refinamento retorna a solução encontrada pelo seguinte PI:

$$\text{min.} \quad \sum_{j=1}^{\ell} c_j y_j \tag{16}$$

s. a

$$\sum_{j \in \mathcal{B}_i} y_j = 1 \qquad \forall i \in V(G) \cup U(G) \tag{17}$$

$$y_j \in \{0,1\} \qquad 1 \leq j \leq \ell \tag{18}$$

A Função Objetivo (16) minimiza a soma dos custos da solução. A Restrição (17) garante que cada vértice $i \in V(G) \cup U(G)$ deve pertencer a exatamente uma única biclique do conjunto $\mathcal{B}_i$. Por fim, a Restrição (18) define o domínio das variáveis.

Note que a função objetivo minimiza o número de operações nas arestas das possíveis soluções do BGEP contidas no conjunto $\mathcal{B}$.

**Algoritmo 4** GRASP+SP

```
1:  procedure GRASP+SP(G)
2:      while (condição de parada não for satisfeita) do
3:          sol = construtiva()
4:          sol = busca local(sol)
5:          adicione as bicliques da solução no Pool
6:          if sol < best then
7:              best = sol
8:          end if
9:      end while
10:     modeloSP = criarModelo(Pool)
11:     sol = solver(modeloSP)
12:     sol = busca local(sol)
13:     if sol < best then
14:         best = sol
15:     end if
16:     return best
17: end procedure
```

O Algoritmo 4 apresenta um pseudocódigo do algoritmo proposto. Note que a heurística construtiva, a busca local e o critério de parada é o mesmo do Algoritmo 3. As diferenças se encontram: na linha 5, em que as bicliques da solução corrente são adicionadas ao *pool* de Bicliques; na linha 10, em que o modelo *Set Partitioning* é criado de acordo com as bicliques contidas no *pool*; na linha 11, em que o modelo de Programação Linear Inteira é resolvido; e por fim na linha 12, na qual a busca local é chamada para uma ultima tentava de melhorar a solução

## 5 Experimentos Computacionais

Nesta seção são apresentadas as ferramentas utilizadas, além dos resultados computacionais do GRASP e do GRASP+SP.

### 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-4.9.2) e executados numa máquina Intel Core i7-4500U com 1.80GHz e 16 Gb de memória RAM no sistema operacional Arch Linux 4.0.1. A versão do CPLEX utilizada no módulo Set Partitioning foi a 12.5. As instâncias utilizadas nos testes computacionais foram obtidas pelo trabalho de Pinheiro et al. (2012), são 20 instâncias criadas com densidade de arestas definidas no conjunto $\{0,2; 0,4; 0,6; 0,8\}$ e tamanhos $m$ e $n$ aleatórios entre 10 e 30.

### 5.2 Resultado

A Tabela 1 apresenta a instância, seguida do número de edições encontradas pelo algoritmo exato (Pinheiro et al., 2012), a solução encontrada na literatura (Pinheiro et al., 2012), juntamente com a solução do GRASP proposto e do GRASP+SP. Observa-se que o algoritmo GRASP encontra o valor ótimo em todas as instâncias com exceção da `i11`. Já o GRASP+SP, encontrou a solução ótima em todas as instâncias.

Tabela 1: Resultados do GRASP e GRASP+SP

| Instância | Exato | (Pinheiro et al., 2012) | GRASP | GRASP+SP |
|---|---|---|---|---|
| i1 | 17 | 17 | 17 | 17 |
| i2 | 20 | 20 | 20 | 20 |
| i3 | 36 | 36 | 36 | 36 |
| i4 | 34 | 34 | 34 | 34 |
| i5 | 35 | 36 | 35 | 35 |
| i6 | 30 | 30 | 30 | 30 |
| i7 | 50 | 51 | 50 | 50 |
| i8 | 28 | 28 | 28 | 28 |
| i9 | 34 | 34 | 34 | 34 |
| i10 | 52 | 53 | 52 | 52 |
| i11 | 53 | 54 | 54 | **53** |
| i12 | 61 | 62 | 61 | 61 |
| i13 | 54 | 54 | 54 | 54 |
| i14 | 63 | 64 | 63 | 63 |
| i15 | 61 | 61 | 61 | 61 |
| i16 | 26 | 28 | 26 | 26 |
| i17 | 40 | 40 | 40 | 40 |
| i18 | 34 | 35 | 34 | 34 |
| i19 | 49 | 49 | 49 | 49 |
| i20 | 51 | 52 | 51 | 51 |
| i21 | 88 | 88 | 88 | 88 |
| i22 | 74 | 74 | 74 | 74 |
| i23 | 24 | 24 | 24 | 24 |
| i24 | 24 | 24 | 24 | 24 |
| i25 | 94 | 94 | 94 | 94 |
| i26 | 23 | 23 | 23 | 23 |
| i27 | 111 | 111 | 111 | 111 |
| i28 | 62 | 63 | 62 | 62 |
| i29 | 56 | 56 | 56 | 56 |
| i30 | 97 | 97 | 97 | 97 |

A Tabela 2 apresenta a instância, junto com o tempo de execução do algoritmo de Pinheiro et al. (2012), o tempo de execução do GRASP proposto e o tempo de execução do GRASP+SP. Note que, o tempo de GRASP proposto foi muito inferior ao tempo de execução da literatura. Observa-se ainda que os tempos do GRASP e do GRASP+SP foram próximos. Além disso, algoritmo de foi executado em uma máquina Intel Core i7-2600 com 3.40GHz, enquanto que o deste artigo foi executado em uma máquina Intel Core i7-4500U com 1.80GHz.

No que diz respeito ao pré-processamento da Seção 2, ele se mostrou eficaz em instâncias esparsas, porém sem muita efetividade para as instâncias densas.

Tabela 2: Tempos do GRASP e GRASP+SP

| Instância | (Pinheiro et al., 2012) | GRASP | GRASP+SP |
|---|---|---|---|
| i1 | 7,23 | 0,26 | 0,30 |
| i2 | 7,00 | 0,21 | 0,24 |
| i3 | 10,09 | 0,39 | 0,45 |
| i4 | 9,97 | 0,35 | 0,41 |
| i5 | 12,90 | 0,41 | 0,46 |
| i6 | 13,89 | 0,48 | 0,49 |
| i7 | 7,88 | 0,49 | 0,52 |
| i8 | 13,13 | 0,39 | 0,42 |
| i9 | 12,53 | 0,47 | 0,50 |
| i10 | 20,31 | 0,98 | 1,09 |
| i11 | 23,41 | 1,35 | 1,17 |
| i12 | 24,50 | 1,40 | 1,26 |
| i13 | 47,19 | 2,66 | 2,39 |
| i14 | 48,24 | 2,82 | 2,78 |
| i15 | 65,95 | 3,27 | 3,25 |
| i16 | 3,40 | 0,09 | 0,10 |
| i17 | 4,02 | 0,05 | 0,05 |
| i18 | 5,26 | 0,12 | 0,14 |
| i19 | 5,28 | 0,06 | 0,08 |
| i20 | 7,05 | 0,36 | 0,39 |
| i21 | 31,80 | 0,15 | 0,14 |
| i22 | 24,27 | 0,17 | 0,14 |
| i23 | 26,45 | 1,71 | 1,48 |
| i24 | 25,56 | 3,28 | 3,24 |
| i25 | 26,48 | 0,16 | 0,19 |
| i26 | 43,42 | 2,95 | 3,05 |
| i27 | 51,80 | 0,20 | 0,22 |
| i28 | 77,30 | 15,52 | 16,09 |
| i29 | 71,91 | 13,73 | 13,65 |
| i30 | 77,21 | 0,34 | 0,37 |

## 6 Conclusão

Neste trabalho foram propostos novos métodos de busca local, além de um método híbrido baseado na solução exata do modelo Set Partitioning. Comparando com os trabalhos da literatura, foi observado que os métodos propostos apresentam um ótimo desempenho, encontrando o valor ótimo em todas as instâncias, e com um tempo bem abaixo da literatura.

Para trabalhos futuros serão propostas novas melhorias que acelerem o método, como por exemplo novo módulos exatos. Pretende-se estender o pré-processamento proposto para os coasos em que a biclique não é completa. Além disso, um estudo mais aprofundado com relação à elaboração de instâncias mais difíceis. Com relação a meta-heurística, pode-se pensar em 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., Ochi, L., Protti, F., Subramanian, A., Martins, I. & Pinheiro, R.** (2014), 'Efficient algorithms for cluster editing', *Journal of Combinatorial Optimization* pp. 1–25.

**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, 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 & Sons.

**Mladenovic, N. & Hansen, P.** (1997), 'Variable neighborhood search', *Computers and Operations Research* **24**(11), 1097–1100.

**Pinheiro, R. G. S., Martins, I. C., Protti, F., Ochi, L. S. & Simonetti, L. G.** (2012), Métodos exatos e heurísticos para biclusterização em grafos, *in* 'XVI CLAIO/XLIV SBPO'.

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

**Protti, F., Dantas da Silva, M. & Szwarcfiter, J. L.** (2006), Applying modular decomposition to parameterized bicluster editing, *in* 'Parameterized and Exact Computation', Vol. 4169 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–12.

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

**Sousa Filho, G. F., dos Anjos F. Cabral, L., Ochi, L. S. & Protti, F.** (2012), 'Hybrid metaheuristic for bicluster editing problem', *Electronic Notes in Discrete Mathematics* **39**(0), 35 – 42.