BRANCH-AND-PRICE APLICADO AO PROBLEMA DE PARTICIONAMENTO DE BIGRAFOS POR EDIÇÃO DE ARESTAS

Gilberto Farias de Sousa Filho[2,3], Hugo Harry Kramer[4], Thiago Gouveia da Silva[1,3], Teobaldo L. Bulhões Júnior[2], Fábio Protti[3], Lucídio dos Anjos F. Cabral[2], Luiz Satoru Ochi[3]
[1]Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB)
João Pessoa – PB – Brasil
[2]Centro de Informática – Universidade Federal da Paraíba (UFPB)
João Pessoa – PB – Brasil
[3]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil
[4]Departamento de Engenharia de Produção - Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil
{gilberto,teobaldoleite,lucidio}@ci.ufpb.br, hugoharry@gmail.com
thiago.gouveia@ifpb.edu.br, {satoru,fabio}@ic.uff.br

RESUMO

O problema de particionamento de bigrafos por edição de arestas, do inglês *Bicluster Editing Problem* (BEP), é NP-difícil e consiste na edição de um número mínimo de arestas de um bigrafo de entrada $G = (V_1, V_2, E)$ com o objetivo de transformá-lo em uma união disjunta de subgrafos bipartidos completos. Aplicações do BEP incluem mineração de dados e análise de dados de expressões genéticas. Neste trabalho, propomos um novo modelo de programação linear inteira com crescimento exponencial do número de variáveis em relação à dimensão do problema. No entanto, propomos abordagens de Geração de Colunas (GC) para resolver a relaxação linear deste modelo em um algoritmo de *Branch-and-Price*. Para a GC, apresentamos um modelo para o subproblema de *pricing*, bem como uma meta-heurística GRASP. Resultados computacionais mostram que nossas propostas são mais eficientes que os procedimentos da literatura, encontrando soluções ótimas anteriormente não provadas para algumas das instâncias testadas.

PALAVRAS CHAVE. Bicluster editing problem, Branch-and-price, Meta-heurística.
Área Principal: Otimização Combinatória

ABSTRACT

The NP-hard Bicluster Editing Problem (BEP) consists of editing a minimum number of edges of an input bigraph G in order to transform it into a vertex-disjoint union of complete bipartite subgraphs. Its applications include data mining and analysis of genetic expression data. In this work, we propose a new mixed integer programming (MIP) formulation in which the number of variables grows exponentially according to the problem dimension. Due to this drawback, we propose Column Generation (CG) approaches to solve the linear relaxation of this model within a branch-and-price algorithm. For the CG, we propose a formulation for the pricing subproblem, as well as a GRASP metaheuristic. Computational results show that our approaches are more efficient than those in the literature, finding previously unknown optimal solutions for the tested instances.

KEYWORDS. Bicluster editing problem, Branch-and-price, Metaheuristics.
Main area: Combinatorial Optimization

1. Introdução

O conceito de agrupar dados em *clusters* aparece em inúmeros contextos e disciplinas. Este tema tem sido estudado extensivamente e vários algoritmos exatos, aproximativos e heurísticos foram propostos, onde o objetivo é encontrar uma partição no conjunto de dados de entrada cujos elementos de um mesmo *cluster* sejam similares, enquanto elementos de *clusters* distintos tenham pouca similaridade. Esta similaridade é geralmente modelada como um grafo: cada vértice representa uma entidade do dado, e dois vértices são conectados por uma aresta se as entidades que eles representam possuem alguma similaridade em um contexto específico. Se os dados são perfeitamente agrupados, então o resultado será um grafo *clusterizado*, que é um grafo no qual cada componente conexa é uma clique. Neste contexto, um simples modelo de *clusterização* é definido como o problema de particionamento de grafos por edição de arestas (em inglês *Cluster Editing Problem* (CEP) (Bansal *et al*., 2004; Shamir *et al*., 2004)). Este problema tem como objetivo encontrar um conjunto mínimo de arestas a serem editadas a fim de transformar o grafo de entrada $G = (V, E)$ em um grafo *clusterizado*. Editar uma aresta (i, j) consiste em adicioná-la (se $(i, j) \notin E$) ou deletá-la (se $(i, j) \in E$).

Em algumas situações, o modelo padrão de *clusterização* não é satisfatório. Um importante exemplo, apresentado por Guo *et al*. (2008), é a clusterização de dados de expressão genética, onde sob certas condições o nível da expressão de um número de genes é medido. Neste caso, *clusterizar* apenas genes ou condições não fornece significado suficiente. Pretende-se encontrar um subconjunto de genes e um subconjunto de condições que juntos se comportem de uma maneira consistente. Tal situação configura o particionamento de bigrafos, do inglês *biclustering* (Madeira e Oliveira, 2004; Tanay *et al*., 2006).

O conceito de *biclusterização* foi introduzido na década de 1970 (Kluger *et al*., 2003), mas seu primeiro uso no contexto da biologia computacional foi realizado por Cheng e Church (2000). O problema de particionamento de bigrafos, ou grafos bipartidos, é uma simples representação de *biclusterização*, em inglês *Bicluster Editing Problem* (BEP). Neste caso, a solução é composta por bicliques, que são subgrafos bipartidos completos, disjuntos em vértices.

Outras aplicações de *biclusterização* surgem em mineração de dados, sistemas de recomendação e filtragem colaborativa. Apesar de sua importância, existem poucos resultados para o BEP em comparação ao CEP. Em Amit (2004), o autor prova que o BEP é NP-difícil e apresenta um algoritmo de aproximação com fator 11 baseado na relaxação de um modelo de programação linear. Usando técnicas de decomposição de grafos, o problema pode ser resolvido com complexidade $O(4^k + m)$ (Protti *et al*., 2009), onde k é o número máximo de edições de arestas permitidas e m representa o número de arestas no grafo. Além disso, Sun *et al*. (2013) propõe um algoritmo exato de fixação de parâmetros para o BEP. Em Guo *et al*. (2008), duas regras de redução do grafo de entrada e um algoritmo aproximativo de fator 4 baseado em uma heurística aleatória são apresentados para o BEP. Ailon *et al*. (2011) apresenta um contra exemplo para o algoritmo aproximativo de Guo *et al*. (2008), além de um novo algoritmo aproximativo de fator 4 utilizando a relaxação do modelo dual para a identificação de seu fator de aproximação. Pinheiro *et al*. (2013) apresentam um algoritmo exato *branch-and-cut* adotando uma programação dinâmica como algoritmo de separação de inequações violadas. Finalmente, nos últimos anos, heurísticas foram propostas, entre as quais a meta-heurística GRASP (Sousa *et al*., 2012), a heurística de remoção de arestas (EDH) (Sun *et al*., 2013) e a heurística Bi-Force (Sun *et al*., 2014).

Este trabalho está estruturado da seguinte forma: na Seção 2, apresentamos definições formais do BEP e propomos um novo modelo de programação linear inteira (PLI). Como o modelo proposto possui uma quantidade muito grande de variáveis, na Seção 3 apresentamos o subproblema de *pricing* para resolver este PLI com a técnica de *branch-and-price*. As técnicas de Geração de Colunas necessárias para resolver esta nova formulação são apresentadas na seção 4. Apresentamos na seção 5 uma meta-heurística GRASP para o subproblema de *pricing*. Os resultados computacionais demonstram uma evolução das técnicas propostas em comparação com

os métodos exatos da literatura; estes resultados são discutidos na Seção 6. Por fim, na Seção 7, apresentamos as conclusões e propostas de trabalhos futuros.

2. Definição do problema

Dado um grafo $G = (V, E)$, editar uma aresta (i, j) consiste em adicioná-la ao grafo (se $(i, j) \notin E$) ou deletá-la (se $(i, j) \in E$). O BEP consiste em editar o número mínimo de arestas a fim de transformar um bigrafo de entrada $G = (V_1 \cup V_2, E)$ em uma união disjunta de subgrafos bipartidos completos; em outras palavras, em uma solução final, cada componente conexa é uma biclique (grafo bipartido completo). As bicliques que formam uma solução são chamadas de *biclusters*.

Consideramos apenas bigrafos não direcionados. Denote por P_4 um caminho com 4 vértices, onde $ijkl$ é um P_4 em que os vértices i e l têm grau 1, e j e k têm grau 2, $i, k \in V_1$ e $j, l \in V_2$. A vizinhança de vértices (aberta) do vértice j é representada por $N(j)$, e a vizinhança fechada $N(j) \cup \{j\}$ é representada por $N[j]$. Estendemos esta notação para o conjunto de vértices S, $N(S) = \bigcup_{j \in S} N(j)$. Para um vértice j, $N^2[j] = N(N(j))$ denota o conjunto de vértices que contém j e os vértices que estão exatamente a distância 2 de j, onde a distância entre dois vértices é definida pelo número de arestas do caminho de comprimento mínimo que os liga. Portanto, $j \in N^2[j]$. Para um grafo $G = (V_1, V_2, E)$, define-se $\overline{E} = \{(i, j) \mid (i, j) \notin E\}$. A Figura 1(a) apresenta uma instância G para o BEP, na qual as partições V_1 e V_2 contém 4 e 3 vértices, respectivamente.

Note que uma solução viável do BEP satisfaz as seguintes condições:

- Cada par de vértices i e j da mesma parte contidas na mesma componente B tem a mesma vizinhança ($N(i) = N(j)$).
- Cada vértice j em uma componente conexa B satisfaz $N(j) \cup N^2[j] = B$.

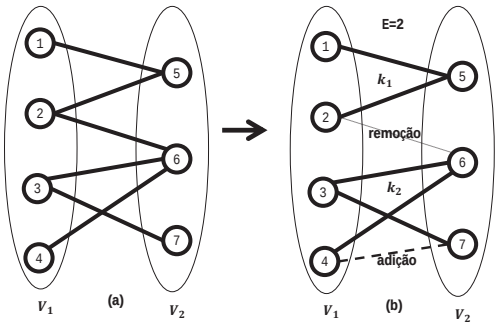

Figura 1: (a) Pequena instância para o BEP. (b) Solução ótima para a instância exemplo.

A Figura 1(b) apresenta uma solução ótima formada por dois *biclusters*: *bicluster* k_1 com vértices $\{1, 2\} \in V_1$ e $\{5\} \in V_2$, e *bicluster* k_2 com vértices $\{3, 4\} \in V_1$ e $\{6, 7\} \in V_2$. Como podemos ver, a aresta $(2, 6)$ foi removida e a aresta $(4, 7)$ adicionada. Por isso, duas edições foram realizadas, e este é o valor da solução ótima.

2.1. Modelo de Programação Linear Inteira

Amit (2004) apresenta uma formulação para o BEP, onde suas inequações evitam a existência de qualquer P_4 $ijkl$, onde $i, k \in V_1$ e $j, l \in V_2$, como um subgrafo induzido de G.

Esta condição é suficiente para garantir que a solução é uma união disjunta de bicliques. Propomos neste trabalho uma outra abordagem para modelar o BEP.

Seja K o conjunto de todos os *biclusters* viáveis, onde um *bicluster* $k \in K$ consiste de um subconjunto de vértices em $V_1 \cup V_2$. Uma variável binária λ_k é usada para decidir se o *bicluster* $k \in K$ é parte da solução. O coeficiente a_{vk} assume valor 1 se o vértice $v \in V_1 \cup V_2$ estiver no *bicluster* $k \in K$ e 0 caso contrário. O custo C_k do *bicluster* k é definido como:

$$C_k = \left|(ijk)^-\right| + \frac{1}{2}\left|(ijk)^+\right|,$$

onde

- $(ijk)^- = \{\bar{E} | i \in k \text{ e } j \in k\}$ e
- $(ijk)^+ = \{E | (i \in k \text{ e } j \notin k) \text{ ou } (i \notin k \text{ e } j \in k)\}$.

Sendo assim, C_k computa as adições ($(ijk)^-$) realizadas dentro do *bicluster* e metade do custo das remoções realizadas entre o *bicluster* k e os outros que se ligavam através de arestas ($(ijk)^+$).

A seguinte formulação tem como objetivo selecionar um conjunto de *bicluters* viáveis que minimize a soma de seus custos.

$$(F_\lambda) \min \sum_{k \in K} C_k \lambda_k \tag{1}$$

$$\text{sujeito a} \quad \sum_{k \in K} a_{vk} \lambda_k = 1 \qquad \forall v \in V_1 \cup V_2 \tag{2}$$

$$\lambda_k \in \{0, 1\} \qquad \forall k \in K \tag{3}$$

A função objetivo (1) minimiza a soma dos custos dos *bicluters* na solução. As restrições (2) garantem que todos os vértices $v \in V_1 \cup V_2$ sejam cobertos pelo conjunto de *bicluters* que fazem parte da solução.

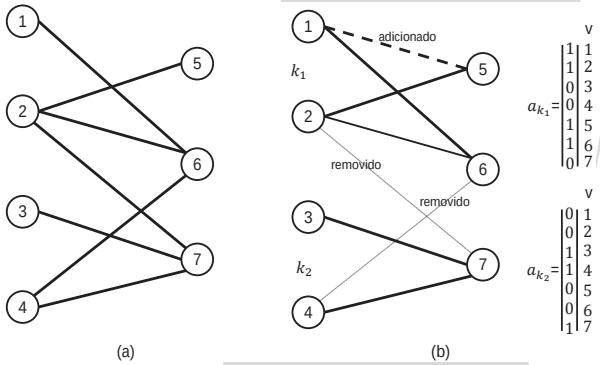

Figura 2: (a) Bigrafo de entrada. (b) Solução s contendo os *biclusters* k_1 e k_2.

A Figura 2(b) apresenta a solução s para o BEP com dois *biclusters* k_1 e k_2. É possível perceber que: $|(ijk_1)^+| = 2$; $|(ijk_1)^-| = 1$; $|(ijk_2)^+| = 2$; e $|(ijk_2)^-| = 0$. São apresentadas também as colunas a_{k_1} e a_{k_2} associadas a estes *biclusters*, donde é fácil perceber que ao fazer $\lambda_{k_1} = \lambda_{k_2} = 1$ teremos coberto todos os vértices do grafo de entrada obtendo uma solução viável. Sendo assim, o custo $z_{\mathcal{F}_\lambda}$ da solução s será dado por:

$$z_{\mathcal{F}_\lambda} = C_{k_1}\lambda_{k_1} + C_{k_2}\lambda_{k_2} = (1 + \frac{1}{2}2) + (0 + \frac{1}{2}2) = 3.$$

As colunas na formulação $\mathcal{F}_\lambda$ são associadas ao conjunto de todos os *biclusters* viáveis. Com o crescimento das instâncias, o número de colunas no modelo se torna muito grande, ficando impossível resolvê-lo, já que depende diretamente da enumeração de todos os *biclusters* viáveis. Para tratar este problema, propomos resolvê-lo por técnicas de *branch-and-price*, onde a relaxação linear de $(\mathcal{F}_\lambda)$ é resolvida em cada nó da árvore de *branch-and-bound* por Geração de Colunas (GC). No algoritmo de GC, uma versão restrita de $\mathcal{F}_\lambda$ com poucas colunas, chamada de Problema Mestre Restrito (PMR), é inicialmente considerada e novas colunas são adicionadas através da resolução do subproblema de *pricing* até que novas colunas não sejam mais necessárias. A seguir apresentaremos o subproblema de *pricing*. Os algoritmos de GC serão detalhados na Seção 4.

3. Subproblema de *Pricing*

O objetivo do subproblema de *pricing* é encontrar uma coluna (representada por um *bicluster*) para $\mathcal{F}_\lambda$ de custo reduzido mais negativo, procurando assim que a adição desta coluna ao PMR melhore sua solução corrente. Seja π o vetor das variáveis duais associadas às restrições (2), e seja a_k a k-ésima coluna contendo todos os vértice que fazem parte do *bicluster* k. Podemos definir o custo reduzido Cr_k da k-ésima coluna como

$$Cr_k = C_k - \sum_{v \in V_1 \cup V_2} \pi_v \, a_{vk}, \tag{4}$$

onde C_k é o custo associado à k-ésima coluna como definido na Subseção 2.1. Logo, iremos modelar o subproblema de *pricing* para construir uma coluna a_k avaliada pelo custo reduzido Cr_k.

Seja $G_{SP} = (V_1, V_2, E)$ um bigrafo de entrada para o subproblema de *pricing* e $N(v)$ a vizinhança de um vértice $v \in V_1 \in V_2$. Atribuímos $w(i,j) = -1$ se $(i,j) \in E$ e $w(i,j) = +1$ caso contrário. Seja x_v uma variável binária que toma valor 1 se o vértice $v \in V_1 \cup V_2$ faz parte da coluna (está no *bicluster*) ou valor 0, caso contrário. Sendo assim, o modelo do subproblema de *pricing* $(\mathcal{SP})$ é

$$(\mathcal{SP}) \min \sum_{v \in V_1 \cup V_2} (\frac{1}{2}|N(v)| - \pi_v)x_v + \sum_{(i,j) \in E \cup \bar{E}} w_{ij} x_i x_j \tag{5}$$

$$\text{sujeito a} \quad x_v \in \{0,1\} \qquad \forall v \in V_1 \cup V_2. \tag{6}$$

A solução de $\mathcal{SP}$ é um *bicluster* que representa uma coluna viável para o PMR com custo reduzido dado pela função objetivo. Qualquer combinação de vértices é solução viável para o subproblema, mesmo o *bicluster* vazio. Entretanto esta solução terá custo reduzido igual a 0 e será descartada pelo algoritmo de GC; por isto, as restrições (6) garantem apenas a integralidade da variável x.

A Figura 3(a) apresenta um bigrafo de entrada para o subproblema de *pricing*. Já a Figura 3(b) ilustra uma solução s para o subproblema, representando uma coluna a_k para o PMR contendo os vértices $\{1;2;5;6\}$. O custo z_{SP} da solução s, obtida pela função objetivo (5), é dada por

$$\begin{aligned} z_{SP} &= ((0,5-\pi_1)+(1,5-\pi_2)+(0,5-\pi_5)+(1,5-\pi_6))+(-1-1-1+1) \\ &= \underbrace{2}_{C_k} - \underbrace{(\pi_1+\pi_2+\pi_3+\pi_4)}_{\pi \cdot a_k}. \end{aligned}$$

Desta forma, observa-se que o custo reduzido dado por (5) é equivalente ao encontrado usando a expressão (4).

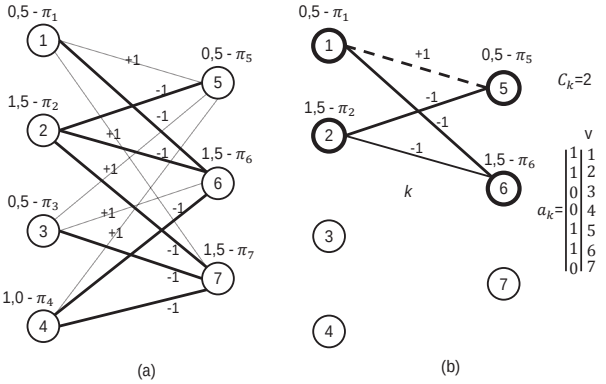

Figura 3: (a) Bigrafo G_{SP} de entrada para $\mathcal{SP}$. (b) Solução para $\mathcal{SP}$ e coluna viável a_k para o PMR.

Como o modelo $\mathcal{SP}$ é não linear, propomos sua linearização por meio da adição das variáveis y_{ij} da forma que segue.

$$(\mathcal{SP}) \min \sum_{v \in V_1 \cup V_2} (\frac{1}{2}|N(v)| - \pi_v)x_v + \sum_{(i,j) \in E \cup \bar{E}} w_{ij} y_{ij} \tag{7}$$

$$\begin{aligned}
\text{sujeito a} \quad y_{ij} &\leq x_i \qquad \forall i \in V_1, j \in V_2 && (8)\\
y_{ij} &\leq x_j \qquad \forall i \in V_1, j \in V_2 && (9)\\
y_{ij} &\geq x_i + x_j - 1 \qquad \forall i \in V_1, j \in V_2 && (10)\\
x_i, y_{ij} &\in \{0,1\} \qquad \forall i \in V_1, j \in V_2 && (11)
\end{aligned}$$

As restrições adicionais (8)–(10) garantem que a aresta (i, j) terá seu custo computado na solução apenas se ambos os vértices i e j também estiverem. Além disso, este subproblema de *pricing* pode ser modificado para que o algoritmo de GC tenha a capacidade de adicionar múltiplas colunas por iteração. Isto é possível pela adição de duas restrições à formulação. Seja $V(w)$ a parte (V_1 ou V_2) em que se encontra o vértice w. Temos que a primeira restrição fixa um vértice w para a solução, e a segunda proíbe os vértices $v \in V(w)$, com $v < w$, de aparecerem nesta solução. Então, durante o *pricing* de cada iteração do algoritmo de GC, um subproblema de *pricing* é resolvido para cada vértice fixo w. Estas restrições, que têm o objetivo de diversificar a busca durante o processo de geração de colunas, estão descritas a seguir:

$$\begin{aligned}
x_w &= 1 && (12)\\
x_v &= 0 \qquad \forall v \in V(w),\ v < w && (13)
\end{aligned}$$

4. Técnicas de Geração de Colunas

Esta seção descreve a técnica de GC proposta para resolver o BEP. Devido ao grande número de variáveis da formulação $\mathcal{F}_\lambda$, sua solução direta por meio de um resolvedor de PLI se torna proibitiva à medida que as instâncias crescem. Os algoritmos de GC são usados para resolver sua relaxação linear a cada nó da árvore de *branch-and-bound*, resultando em um algoritmo de *branch-and-price*.

4.1. Geração de Colunas Padrão

O algoritmo de Geração de Colunas padrão proposto para o BEP é detalhado no Alg. 1. O Alg. 1(a) é o algoritmo de GC padrão onde apenas uma coluna pode ser adicionada a cada iteração. Ele começa com a inicialização do PMR com um subconjunto das colunas da relaxação linear de $\mathcal{F}_\lambda$. Tais colunas iniciais são fornecidas a partir da melhor solução gerada por uma meta-heurística GRASP proposta em Sousa *et al*. (2012). A cada iteração, o PMR é resolvido e as variáveis duais π_v das restrições (2) são obtidas. O subproblema de *pricing* $\mathcal{SP}$ é alimentado com estas variáveis duais e resolvido até a otimalidade por um resolvedor de PLI. A solução do $\mathcal{SP}$ representa um *bicluster*, que é uma coluna do PMR com custo reduzido dado pela função objetivo (7). Se o custo reduzido for negativo, esta coluna será adicionada ao PMR e o algoritmo prossegue para a próxima iteração. Em uma dada iteração, se a solução do subproblema de *pricing* gerar uma coluna com custo reduzido não negativo, significa que esta coluna não será adicionada ao PMR. Além disso, significa que a adição de colunas ao PMR não é mais necessária e que a solução ótima da relaxação linear de $\mathcal{F}_\lambda$ foi encontrada. Então, o algoritmo para, retornando esta solução.

```
1: procedure SINGLECOLSTANDARDCG(PMR)
2:     sol ← ∅
3:     sol_SP ← ∅
4:     Inicialize PMR
5:     colAdded ← true
6:     while colAdded do
7:         sol ← resolvedorPL(PMR)
8:         colAdded ← false
9:         Atualize SP com os valores da solução dual π
10:        sol_SP ← resolvedorPLI(SP(π))
11:        if f(sol_SP) > 0 then
12:            Insere coluna associada a sol_SP para PMR
13:            colAdded ← true
14:        end if
15:    end while
16:    return sol
17: end procedure
```

(a) GC padrão adicionando uma coluna por iteração

```
1: procedure MULTICOLSSTANDARDCG(PMR)
2:     sol ← ∅
3:     sol_SP_w ← ∅
4:     Inicialize PMR
5:     colAdded ← true
6:     while colAdded do
7:         sol ← resolvedorPL(PMR)
8:         colAdded ← false
9:         Atualize SP com os valores da solução dual π
10:        for cada vértice w ∈ V1 ∪ V2 do
11:            sol_SP_w ← resolvedorPLI(SP_w(π))
12:            if f(sol_SP_w) > 0 then
13:                Insere coluna associada a sol_SP_w para PMR
14:                colAdded ← true
15:            end if
16:        end for
17:    end while
18:    return sol
19: end procedure
```

(b) GC padrão adicionando múltiplas colunas por iteração

Alg. 1: Algoritmos de Geração de Colunas padrão

No Alg. 1(b), a versão do algoritmo de GC padrão onde múltiplas colunas podem ser adicionadas ao PMR em cada iteração é apresentada. A única diferença está no *pricing*, onde um subproblema $\mathcal{SP}_w$ é resolvido para cada vértice $w \in V_1 \cup V_2$. Assim, a cada iteração deste algoritmo, até $|V_1| + |V_2|$ colunas podem ser adicionadas.

4.2. Geração de Colunas por fases

Podemos perceber que o algoritmo de GC padrão proposto para o problema na subseção anterior tem sua performance computacional dependente da performance do resolvedor de PLI usado no *pricing*. Por isto, com o objetivo de aliviar o peso computacional da GC padrão, propõe-se aplicar uma técnica descrita no Alg. 2 similar à apresentada por Kramer *et al*. (2014) ao Problema de *Clusterização* de *Software*.

Como nos algoritmos de GC padrão, duas versões são propostas, onde se adiciona uma única coluna por iteração e múltiplas colunas por iteração. Estas versões diferem das anteriores na adoção de duas fases de *pricing*, sendo uma heurística e outra exata, para a resolução do PMR. Primeiramente, o PMR é inicializado como previamente definido no Alg. 1. Vemos no Alg. 2(a) que na primeira fase (heurística) uma meta-heurística GRASP é aplicada na resolução do subproblema de *pricing*. Mais detalhes deste algoritmo GRASP serão apresentados na Seção 5. A cada iteração, a coluna associada à solução obtida pela meta-heurística GRASP com o custo reduzido mais negativo

é adicionada ao PMR. Se o custo reduzido desta coluna não for negativo, a coluna não é adicionada e a fase heurística é finalizada. Em seguida, os subproblemas remanescentes são resolvidos na otimalidade pelo resolvedor de problemas de PLI. Portanto, na fase exata, as iterações são realizadas até a convergência do mesmo modo que a GC padrão, retornando a solução ótima do PMR.

```
1: procedure SINGLECOLSTAGEDCG(PMR, maxStages)
2:     s ← 1
3:     sol ← ∅
4:     sol_SP ← ∅
5:     inicialize PMR
6:     colsAdded ← true
7:     while colsAdded do
8:         sol ← resolvedorPL(PMR)
9:         colsAdded ← false
10:        Atualize SP com valores da solução dual π
11:        while s < maxStages do
12:            sol_SP ← GRASP(G_SP, π)
13:            if f(sol_SP) > 0 then
14:                Insere coluna associada a sol_SP ao PMR
15:                colsAdded ← true
16:            else
17:                s ← s + 1
18:            end if
19:        end while
20:        sol_SP ← resolvedorPLI(SP(π))
21:        if f(sol_SP) > 0 then
22:            Insere coluna associada a sol_SP ao PMR
23:            colsAdded ← true
24:        end if
25:    end while
26:    return sol
27: end procedure
```

(a) GC por fases adicionando uma coluna por iteração

```
1: procedure MULTICOLSSTAGEDCG(PMR, maxStages)
2:     s ← 1
3:     sol ← ∅
4:     sol_SP ← ∅
5:     Inicialize RMP
6:     colsAdded ← true
7:     while colsAdded do
8:         sol ← LPSolver (PMR)
9:         colsAdded ← false
10:        Atualiza SP com valores da solução dual π
11:        while s < maxStages do
12:            for each node w ∈ V do
13:                sol_SP_w ← GRASP(G_SP, π, w)
14:                if f(sol_SP_w) > 0 then
15:                    Insere coluna associada a sol_SP_w ao PMR
16:                    colsAdded ← true
17:                end if
18:            end for
19:            if !colsAdded then
20:                s ← s + 1
21:            end if
22:        end while
23:        for each vértices w ∈ V1 ∪ V2 do
24:            sol_SP_w ← resolvedorPLI (SP_w(π))
25:            if f(sol_SP_w) > 0 then
26:                Insere a coluna associada a sol_SP_w ao PMR
27:                colsAdded ← true
28:            end if
29:        end for
30:    end while
31:    return sol
32: end procedure
```

(b) GC por fases adicionando múltiplas colunas por iteração

Alg. 2: Algoritmos de Geração de Colunas por fases

A versão do algoritmo que permite a adição de múltiplas colunas por iteração detalhada no Alg. 2(b) difere apenas no *pricing* do Alg. 2(a). Agora, na fase heurística, a meta-heurística GRASP cria um *bicluster*-solução que satisfaça as restrições (12) e (13). Na fase exata, o *pricing* realiza o mesmos passos do Alg. 1(b).

4.3. *Branching*

O esquema de *branching* no algoritmo *branch-and-price* é: após resolver cada nó da árvore de *branch-and-bound* usando um dos algoritmos de GC apresentados, dois nós filhos são criados. Para cada um destes novos nós, restrições disjuntivas de *branching* são adicionadas ao PMR na forma apresentada por Vanderbeck (2011), que generaliza o esquema de Ryan e Foster (1981). Para o BEP, estas restrições disjuntivas são:

$$\sum_{k \in \hat{K}} \lambda_k \leq 0 \quad \text{ou} \quad \sum_{k \in \hat{K}} \lambda_k \geq 1, \tag{14}$$

onde $\hat{K}$ é um subconjunto de colunas do PMR em que, para dois vértices $i \in V_1$ e $j \in V_2$, temos $a_{ik} = 1$ e $a_{jk} = 1$, ou seja, i e j fazem parte do mesmo *bicluster* nestas colunas.

Assim, adicionando as restrições disjuntivas da esquerda, as variáveis λ_k com $k \in \hat{K}$ são proibidas na solução do PMR, isto é, os vértices i e j não podem fazer parte do mesmo *bicluster*.

Pela adição das restrições da direita, pelo menos uma variável λ_k com $k \in \hat{K}$ deve estar na solução do PMR; isto significa que, nesta solução, os vértices i e j devem estar no mesmo *bicluster*.

5. Meta-heurística GRASP para o Subproblema de *Pricing*

Para resolver instâncias médias e grandes para o subproblema de *pricing* durante o algoritmo de GC, adaptamos a meta-heurística GRASP (Resende, 2001), um procedimento iterativo onde cada iteração consiste de duas fases: *construção* e *busca local*. A melhor solução obtida entre todas as iterações é considerada a solução final. Nesta seção apresentamos uma heurística construtiva e três movimentos de vizinhança adaptadas ao GRASP. Como entrada recebemos o bigrafo G_{SP} descrito na seção 3 e as variáveis duais π_v associadas as restrições do modelo $\mathcal{F}_\lambda$. Temos como representação da solução uma lista dos vértices que compõem um *bicluster*.

5.1. Fase de construção

A fase de construção é baseada na análise da vizinhança de vértices em G_{SP}. Ela inicia com um *bicluster* k vazio. Uma escolha gulosa de um par de vértices (i, j), $i \in V_1$ e $j \in V_2$, é realizada. Esta escolha é guiada com a ajuda de uma função gulosa $g(i, j)$. A cada iteração constrói-se uma lista de candidatos (*LC*) contendo todos os pares de vértices de G_{SP} ordenados (crescentemente) pela função gulosa $g(i, j)$. Depois, uma lista restrita de candidatos (*LRC*) é construída pela seleção dos candidatos em *LC* com os menores valores gulosos. Segue-se a definição da função gulosa:

$$g(i, j) = w_\pi(i, j) + in(i, j)$$

onde:

- $w_\pi(i, j) = \frac{1}{2}(|N(i)| + |N(j)|) - (\pi_i + \pi_j) + w(i, j)$;
- $in(i, j)$ é a soma dos pesos de todos os pares de vértices entre i e $N(j) \cap V_2$, e entre j e $N(i) \cap V_1$.

Após a escolha aleatória de um par de vértices (i, j) contidos na *LRC*, um novo *bicluster* isolado k é criado, composto pelo subconjunto de vértices $N[i] \cup N[j]$.

5.2. Fase de busca local

Na fase de busca local, novas soluções próximas à solução obtida na fase de construção são geradas. O método de busca em descida, *Variable Neighborhood Descent* (VND) descrito em (Hansen *et al*., 2010), é usado pela aplicação do conjunto de vizinhanças $\mathcal{N} = \{$*Remove-vértice, Adiciona-vértice, Troca-vértices*$\}$, na ordem em que aparecem. Para isto, são propostos os seguintes movimentos de vizinhança sobre um bicluster k: (a) ***Adiciona-vértice***: adiciona um vértice $v \notin k$ a k; (b) ***Remove-vértice***: remove um vértice $v \in k$ de k; (c) ***Troca-vértices***: remove um vértice $v_1 \in k$ e adiciona $v_2 \notin k$ a k.

6. Resultados Computacionais

Todos os algoritmos propostos neste trabalho foram desenvolvidos em C++. O resolvedor de problemas de PL e PLI utilizado foi o IBM ILOG CPLEX versão 12.5.1 64 bits. Para a implementação das abordagens de *branch-and-price* ($B\&P$) propostas neste trabalho, utilizamos o *framework* de geração de colunas BaPCod[1] em conjunto com as bibliotecas *Boost C++ libraries* 1.54. As abordagens propostas são comparadas com o *branch-and-bound* do CPLEX aplicado à formulação proposta por Amit (2004) ($B\&B$) e com o algoritmo de *branch-and-cut* proposto por Pinheiro *et al*. (2013) ($B\&C$). Esta última abordagem também foi desenvolvida em C++ e seu código foi disponibilizado pelo autor. Os experimentos computacionais foram realizados em uma

[1]Mais informações em `https://realopt.bordeaux.inria.fr/?page_id=2`

Tabela 1: Comparação entre $B\&B$, $B\&C$ e $B\&P$ para o BEP.

Instância				$B\&B$	$B\&C$	$B\&P$ **- 1Col**	$B\&P$ **- MultiCol**
n	m	d	e^*	*tempo*	*tempo*	*tempo*	*tempo*
10	11	0,4	17	0,45	**0,07**	0,18	0,46
10	11	0,6	26	0,86	**0,06**	0,3	0,83
10	14	0,6	40	7,78	**0,27**	1,36	2,84
10	18	0,4	36	76,80	5,18	**0,94**	1,98
11	11	0,6	34	10,18	1,00	**0,97**	1,31
11	13	0,4	34	12,99	1,36	**1,29**	1,72
11	15	0,4	35	62,01	7,65	**1,06**	1,89
11	16	0,4	30	13,74	**0,30**	0,92	1,79
11	18	0,4	50	978,23	83,13	**3,35**	5,28
12	13	0,4	28	10,63	**0,53**	0,58	1,15
12	14	0,6	49	34,60	**0,46**	2,53	2,87
12	15	0,6	51	648,48	65,39	**3,08**	4,4
12	19	0,4	52	3124,62	227,25	**2,81**	5,04
13	19	0,4	61	**LT**	2128,79	**3,74**	7,06
15	16	0,4	54	4094,37	362,74	**16,78**	45,19
15	25	0,8	88	3,87	**0,21**	8,98	11,49
15	26	0,8	74	3,01	**0,12**	4,14	3,21
16	16	0,2	24	25,03	**0,40**	0,48	1,13
16	17	0,4	63	**LT**	2689,38	**4,71**	6,55
16	18	0,2	24	6,41	**0,28**	0,47	1,52
16	29	0,8	94	5,41	**0,25**	7,09	6,35
17	18	0,2	23	5,45	0,51	**0,48**	1,89
17	29	0,8	111	8,25	**0,41**	24,64	12,36
20	28	0,2	**62**	**EM**	**LT**	**5,43**	18
21	22	0,2	56	**LT**	4875,39	**4,81**	8,93
21	22	0,8	97	6,18	**0,23**	12,93	13,02

máquina Intel Core i7-4790 64 bits 3,6 GHz com 16 GB de RAM, sob o sistema operacional Linux Ubuntu 12.04.2 64 bits.

Na Tabela 1 comparamos o resultado das estratégias $B\&B$, $B\&C$ e $B\&P$ utilizando as instâncias de Pinheiro *et al*. (2013). A tabela é dividida em cinco grupos de colunas. O conjunto de colunas *Instância* é dividida em n, m, d e e^*, onde $n = |V_1|$, $m = |V_2|$, $d = \frac{|E|}{n*m}$ e e^* é o valor ótimo relacionado a cada instância. As outras colunas exibem o tempo computacional em segundos para as estratégias $B\&B$ e $B\&C$ e para duas estratégias de $B\&P$ propostas, o $B\&P$ *- 1Col*, onde a cada iteração do procedimento apenas uma coluna é adicionado ao PMR, e o $B\&P$ *-MultiCol*, onde múltiplas colunas são adicionadas por iteração. Para ambas abordagens de $B\&P$, os resultados reportados foram obtidos considerando os algoritmos de GC por fases.

O parâmetro d, que indica a densidade de arestas dos grafos, é decisivo na métrica de dificuldade de uma instância BEP (vide Sousa *et al*., 2012), onde instâncias com $d \in \{0,5; 0,6\}$ são, em geral, mais difíceis de resolver, quando comparadas a instâncias com densidade baixa ou alta. O $B\&C$ obtém melhores tempos nas instâncias com $d \in \{0,2; 0,8\}$, tendo melhor tempo em 8 das 11 instâncias com esta característica. Já o procedimento $B\&P$ *- 1Col* obteve o melhor tempo em 10 dentre 15 instâncias com $d \in \{0,4; 0,6\}$. Um ganho mais expressivo no tempo se dá com instâncias maiores. Por exemplo, considere a instância ($n = 13, m = 19, d = 0,4$), onde o procedimento $B\&P$ *- 1Col* encontrou o ótimo em 3,74 segundos, e o $B\&C$ em 2128,79 segundos. Já na instância ($n = 20, m = 28, d = 0,2$), o procedimento $B\&P$ *- 1Col* encontrou o ótimo em 5,43 segundos, enquanto o $B\&C$ não atingiu o ótimo em mais de três horas de execução (LT). Já o procedimento $B\&B$ teve problemas com o uso de memória nesta instância (EM). A estratégia $B\&P$ *- MultiCol* obteve os mesmos resultados de $B\&P$ *- 1Col*, porém com maior tempo computacional.

A Tabela 2 exibe os resultados computacionais dos mesmos procedimentos do teste anterior para as instâncias propostas por Sousa *et al*. (2012), com valores $d \in \{0,5; 0,6\}$. Nesta tabela temos cinco conjuntos de colunas. O conjunto *Instância* é dividido nas colunas $n = |V_1|$, $m = |V_2|$ e $d = \frac{|E|}{n*m}$. As colunas $B\&B$, $B\&C$, $B\&P$ *- 1Col* e $B\&P$ *- MultiCol* são divididas nas colunas: *Mint*, que representa o melhor valor inteiro encontrado pelo procedimento; *tempo*,

Tabela 2: Comparação entre o $B\&B$, $B\&C$ e $B\&P$ para o BEP.

Instância			$B\&B$			$B\&C$			$B\&P$ **- 1Col**			$B\&P$ **- MultiCol**		
n	m	d	*Mint*	*tempo*	*GAP*	*Mint*	*tempo*	*GAP*	*Mint*	*tempo*	*GAP*	*Mint*	*tempo*	*GAP*
5	7	0,5	7	10	0%	7	13	0%	7	**0,02**	0%	7	0,12	0%
5	7	0,6	8	30	0%	8	11	0%	8	**0,03**	0%	8	0,11	0%
6	8	0,5	8	20	0%	8	5	0%	8	**0,03**	0%	8	0,13	0%
6	8	0,6	11	30	0%	11	15	0%	11	**0,03**	0%	11	0,15	0%
6	12	0,5	14	90	0%	14	33	0%	14	**0,05**	0%	14	0,23	0%
6	12	0,6	17	130	0%	17	25	0%	17	**0,08**	0%	17	0,52	0%
7	11	0,5	11	40	0%	11	6	0%	28	**0,5**	0%	11	1,14	0%
7	11	0,6	19	210	0%	19	49	0%	30	**0,26**	0%	19	1,14	0%
6	20	0,5	28	1990	0%	28	335	0%	11	**0,04**	0%	28	0,27	0%
6	20	0,6	30	1130	0%	30	139	0%	19	**0,12**	0%	30	0,44	0%
10	16	0,5	39	51560	0%	39	2.169	0%	39	**1,35**	0%	39	2,13	0%
10	16	0,6	47	62090	0%	47	3.129	0%	47	**1,95**	0%	47	3,58	0%
16	30	0,5	165	**LT**	46%	172	**EM**	43%	149	313,51	0%	149	**127,91**	0%
16	30	0,6	184	**LT**	45%	218	**EM**	48%	162	**3087,44**	0%	162	3983,43	0%
20	23	0,5	170	**LT**	46%	173	**EM**	41%	143	**8323,37**	0%	143	8750,74	0%
20	23	0,6	192	**LT**	46%	203	**EM**	43%	157	16753,99	0%	157	**1259,39**	0%

que representa o tempo computacional em segundos para executar o procedimento; e *GAP*, que representa a razão entre a diferença dos valores de *upper bound* e *lower bound* sobre o valor do *upper bound* encontrado por cada procedimento.

Podemos observar para estas instâncias que os procedimentos $B\&P$ - *1Col* e $B\&P$ - *MultiCol* obtiveram os melhores tempos computacionais, encontrando 4 novas soluções ótimas. Os procedimentos $B\&B$ e $B\&C$ falharam nas 4 últimas instâncias devido a problemas com memória (EM) ou limite de tempo de três horas ultrapassado (LT). O procedimento $B\&P$ - *MultiCol* obteve melhor tempo computacional em duas instâncias maiores, apresentando bom comportamento no crescimento das instâncias.

7. Conclusões e Trabalhos Futuros

Neste trabalho propomos um novo modelo de programação linear inteira (PLI) para o problema de particionamento de bigrafos por edição de arestas (BEP). Este modelo ($\mathcal{F}_\lambda$) apresenta um crescimento exponencial no número de variáveis com a dimensão da instância. Para tratar esta característica, propomos algoritmos de Geração de Colunas para resolver sua relaxação linear.

Para os algoritmos de Geração de Colunas, propomos um modelo de PLI para o subproblema de *pricing*, que tem como objetivo encontrar o *bicluster* de menor custo reduzido. Como este subproblema deve ser resolvido durante várias iterações do algoritmo de GC, o procedimento se torna muito ineficiente. Para superar esta desvantagem propomos uma meta-heurística GRASP para a resolução do subproblema de *pricing*.

Os resultados computacionais mostram que a abordagem $B\&P$ - *1Col* é mais eficiente que as da literatura obtendo melhores tempos computacionais com o crescimento da dimensão e dificuldade das instâncias, provando 5 soluções ótimas que anteriormente não haviam sido provadas. O $B\&P$ - *MultiCol* apresenta uma boa escalabilidade para instâncias maiores, reduzindo o tempo computacional em mais de 90% para a instância ($n = 16, m = 30, d = 0,6$).

Como trabalhos futuros, propomos: melhorias na formulação do subproblema de *pricing* e na meta-heurística para resolvê-lo; e definição de novas regras de *branching* para o procedimento $B\&P$.

Referências

Ailon, N., Avigdor-Elgrabli, N., Liberty, E. e van Zuylen, A. Improved approximation algorithms for bipartite correlation clustering. Demetrescu, C. e Halldórsson, M. (Eds.), *Algorithms ESA*, volume 6942 of *Lecture Notes in Computer Science*, p. 25–36. Springer Berlin Heidelberg, 2011.

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

Bansal, N., Blum, A. e Chawla, S. (2004), Correlation clustering. *Machine Learning*, v. 56, p. 89–113.

Cheng, Y. e Church, G. M. Biclustering of expression data. *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, p. 93–103. AAAI Press. ISBN 1-57735-115-0, 2000.

Guo, J., Hüffner, F., Komusiewicz, C. e Zhang, Y. Improved algorithms for bicluster editing. *TAMC'08 - 5th International Conference on Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, p. 445–456, 2008.

Hansen, P., Mladenović, N. e Moreno Perez, J. (2010), Variable neighbourhood search: methods and applications. *Annals of Operations Research*, v. 175, p. 367–407.

Kluger, Y., Basri, R., Chang, J. e Gerstein, M. (2003), Spectral biclustering of microarray data: Coclustering genes and conditions. *Genome Research*, v. 13, p. 703–716.

Kramer, H. H., Fampa, M., Köhler, V., Vanderbeck, F. e Uchoa, E. Column generation approaches for the software clustering problem. *Anais do XLVI Simpósio Brasileiro de Pesquisa Operacional*, 2014.

Madeira, S. C. e Oliveira, A. L. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, volume 1, p. 24–45. IEEE, 2004.

Pinheiro, R. G. S., Martins, I. C., Protti, F., Ochi, L. S., Simonetti, L. e Subramanian, A. On solving manufacturing cell formation via bicluster editing., 2013.

Protti, F., da Silva, M. D. e Szwarcfiter, J. L. (2009), Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, v. 44, p. 91–104.

Resende, M. Greedy randomized adaptive search procedures. Floudas, C. A. e Pardalos, P. M. (Eds.), *Encyclopedia of Optimization*, p. 913–922. Springer US, 2001.

Ryan, D. M. e Foster, B. A. (1981), An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, p. 269–280.

Shamir, R., Sharan, R. e Tsur, D. (2004), Cluster graph modification problems. *Discrete Applied Mathematics*, v. 144, p. 173–182.

Sousa, G. F., Cabral, L. d. A., Ochi, L. S. e Protti, F. (2012), Hybrid metaheuristic for the bicluster editing problem. *Eletronic Notes in Discrete Mathematics*, v. 39, p. 35 – 42.

Sun, P., Guo, J. e Baumbach, J. (2013), Biclue - exact and heuristic algorithms for weighted bi-cluster editing of biomedical data. *BMC Proceedings*, v. 7, n. Suppl 7, p. S9.

Sun, P., Speicher, N. K., Röttger, R., Guo, J. e Baumbach, J. (2014), Bi-force: large-scale bicluster editing and its application to gene expression data biclustering. *Nucleic Acids Research*.

Tanay, A., Sharan, R. e Shamir, R. Biclustering algorithms: a survey. Aluru, S. (Ed.), *Handbook of Computational Molecular Biology*. Chapman Hall/CRC Press, 2006.

Vanderbeck, F. (2011), Branching in branch-and-price: a generic scheme. *Mathematical Programming*, v. 130, n. 2, p. 249–294.