Metaheuristic GRASP for the Bicluster Editing Problem

Gilberto F. de Sousa Filho[1], **Lucidio dos Anjos F. Cabral**[2], **Luiz Satoru Ochi**[3], **Fábio Protti**[3]

[1]Departamento de Ciências Exatas – Universidade Federal da Paraíba(UFPB)
Rio Tinto – PB – Brazil

[2]Centro de Informática – Universidade Federal da Paraíba(UFPB)
João Pessoa – PB – Brazil

[3]Instituto de Computação – Universidade Federal Fluminense(UFF)
Niterói - RJ - Brazil

gilberto@dce.ufpb.br, lucidio@ci.ufpb.br, {satoru, fabio}@ic.uff.br

ABSTRACT

The NP-hard Bicluster Editing Problem consists of adding and/or removing at most k edges in order to transform an input bipartite graph $G = (V, E)$ into a vertex-disjoint union of complete bipartite subgraphs. It has applications in the analysis of gene expression data. We propose the generation and analysis of random bipartite graphs to perform empirical tests. A new reduction rule of graphs is proposed, based on the idea of *critical independent sets*; it allows more effective reduction in the size of the instances, without compromising the optimal solution. We further propose a metaheuristic GRASP for the Bicluster Editing Problem, containing a heuristic construction based on handling vertex neighborhoods.

Keyword. Bicluster Editing, Clustering, GRASP.

1. Introduction

The concept of grouping data into clusters arises in numerous contexts and disciplines. This subject has been extensively studied and various exact, approximation and heuristic algorithms were proposed, where the goal is to partition a data set into clusters such that elements within a cluster are similar, while elements in distinct clusters have less similarity. This similarity is often modeled as a graph: each vertex represents a data point, and two vertices are connected by an edge if the entities that they represent have some (context-specific) similarity. If the data were perfectly clustered, this would result in a cluster graph, that is, a graph where every connected component is a clique. A simple clustering model is then defined by the Cluster Editing Problem [Bansal et al. 2004, Shamir et al. 2004]: find a minimum set of edges to be added and/or deleted in order to transform the input graph into a cluster graph.

In some settings, the standard clustering model is not satisfactory. An important example, described by [Guo et al. 2008], is clustering of gene expression data, where under a certain number of conditions, the level of expression of a number of genes is measured. This yields a bipartite similarity graph. Here, clustering only genes or only conditions often does not yield sufficient insight; we would like to find subsets of genes and subsets of conditions that together behave in a consistent way. This is called biclustering [Madeira and Oliveira 2004, Tanay et al. 2006]. The concept of biclustering was first introduced in the seventies [Kluger et al. 2003], but its first usage in the context of computational biology was due to [Cheng and Church 2000].

A simple formulation of biclustering, analogous to cluster editing, is defined by the Bicluster Editing Problem. Here, as a consistency condition for a cluster, we demand that it forms a biclique, that is, a complete bipartite subgraph. Further, we do not allow intersection between any two clusters.

Further applications of biclustering arise in collaborative filtering, information retrieval, and data mining. Despite its importance, there are fewer results for Bicluster Editing than for Cluster Editing. Amit[Amit 2004] proved the NP-hardness of the Bicluster Editing Problem and described a 11-approximation algorithm based on the relaxation of a linear program. Using a simple branching strategy, the problem can be solved in $O(4^k + m)$ time [Protti et al. 2006], where m is the number of edges in the graph. The parameterized version of the Bicluster Editing Problem is fixed-parameter tractable (see e.g. [Protti et al. 2006]). In [Guo et al. 2008], two reduction rules and a 4-approximation algorithm based on a random heuristic for the Bicluster Editing Problem are described.

In this paper we present a linear programming model for the Bicluster Editing Problem, and its generalization to weighted graphs (Section 2). We propose an algorithm for generating random bipartite graphs and a new rule to reduce input bipartite graphs without affecting optimality (Section 3). Existing approximation algorithms are described and a metaheuristic GRASP is proposed to the Bicluster Editing Problem (Section 4). Computational results are presented in Section 5 and concluding remarks in Section 6.

2. Bicluster Editing Problem

The Bicluster Editing Problem consists of adding and/or removing at most k edges in order to transform an input bipartite graph $G = (V_1, V_2, E)$ into a vertex-disjoint union of complete bipartite subgraphs.

Preliminaries. We consider only undirected bipartite graphs $G = (V_1, V_2, E)$. Let P_4 denote an induced path with 4 vertices. Furthermore, let $ijkl$ denote a P_4 in which i and l have degree 1 and j and k have degree 2. The neighborhood of a vertex v is denoted by $N(v)$, and the closed neighborhood $N(v) \cup \{v\}$ is denoted by $N[v]$. We furthermore extend this notation to vertex sets, that is, for a vertex set S, $N(S) = (\bigcup_{v \in S} N(v)) \backslash S$. For a vertex v, $N_2(v) = N(N(v)) \backslash \{v\}$ denotes the set of vertices at distance exactly 2 from v.

2.1. The Linear Program

In [Amit 2004], a formulation is presented for the ±1 Bicluster Editing Problem, described as follows: assign a binary variable x_{ij} to every edge (i, j), such that $x_{ij} = 0$ if i and j end up in the same bicluster. Consider the integer programming problem below:

$$\textit{Minimize} \sum_{+(i,j)} x_{ij} + \sum_{-(i,j)} (1 - x_{ij}) \tag{1}$$

Subject to:

$$x_{ij} \leq x_{il} + x_{kj} + x_{kl}, \text{ for all } i, k \in V_1, j, l \in V_2 \tag{2}$$

$$x_{ij} \in \{0,1\}, \forall i \in V_1, \forall j \in V_2 \tag{3}$$

where $+(i,j) = \{(i,j) \mid w(i,j) = +1\}$, and $-(i,j) = \{(i,j) \mid w(i,j) = -1\}$.

To generalize the formulation by [Amit 2004], allowing values for $w(i,j)$ such that $w(i,j) \notin \{-1,+1\}$, we propose to replace the objective function (1) by the following function:

$$\mathit{Minimize} \sum_{+(i,j)} w(i,j)x_{ij} + \sum_{-(i,j)} |w(i,j)|\,(1 - x_{ij}) \tag{4}$$

Hereafter, $+(i,j) = \{(i,j) \mid w(i,j) > 0\}$, and $-(i,j) = \{(i,j) \mid w(i,j) < 0\}$.

The objective function measures the number of errors. This includes positive errors, i.e., positive edges between biclusters (positive edges (i,j) for which $x_{ij} = 1$), as well as negative errors, i.e., negative edges inside biclusters (negative edges (i,j) for which $x_{ij} = 0$). It is easy to see that inequalities $x_{ij} \leq x_{il} + x_{kj} + x_{kl}$ guarantee that if vertices i, l are in the same bicluster, as well as vertices k, l and k, j, then vertices i, j must be in the same bicluster.

3. Generation and Reduction of Instance

This section discuss how to generate random instances for the Bicluster Editing Problem, and presents an analysis of the generated instances with varying levels of difficulty. Below we describe existing reductions in the literature, and propose a new reduction rule.

3.1. Generation Algorithms

Studies concerning the Bicluster Editing Problem in the literature propose formal proofs of the effectiveness of their solutions without presenting empirical results. Therefore there is a need of creating test instances to perform our computational experiments.

We follow the model of random graphs proposed in [Gilbert 1959]. We denote by $G(n,m,p)$ a random bipartite graph with n vertices in partition V_1, m vertices in partition V_2 and such that each edge between partitions can occur independently with probability p. [Bastos 2012] proposed a simple and suitable framework for generation of random graphs. We adapt their approach for generation of bipartite graphs. First, initialize the set of vertices V_1 with n elements, V_2 with m elements, and $E = \emptyset$ (recall that E is the set of edges). Next, for each pair of vertices $i \in V_1$ and $j \in V_2$, decide with probability p if edge (i,j) is added to E. Finally, the bipartite graph G generated by this procedure is returned.

To determine the difficulty of an instance, [Bastos 2012] uses a criterion based on the number of editions of edges needed to achieve optimality (the value $\mathit{opt}(G)$). For two instances G_1 and G_2 with the same number of vertices $n * m$, we say that G_1 is more difficult than G_2 if $\mathit{opt}(G_1) > \mathit{opt}(G_2)$.

The value $\mathit{opt}(G(n,m,p))$ was calculated for various pairs ($n * m$, p). Figure 1 shows the results for some values of $n * m$. Note that, in general, the most difficult instances are concentrated a for values $p \in [0.6, 0.7]$.

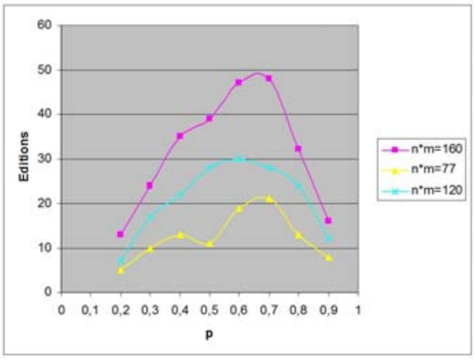

Figure 1. $opt(G(n,m,p))$ **values for** $n * m$ **=** $\{$**77, 120, 160**$\}$

3.2. Reduction Rules

[Guo et al. 2008] proposes and presents the formal proof of the following two data reduction rules; the second one works on critical independent sets.

Definition 1. *A set S of vertices is called a critical independent set if all vertices in S have the same open neighborhood and S is maximal under this property.*

Observe that every critical independent set is an independent set. The connection between critical independent sets and Bicluster Editing is given by the following lemma:

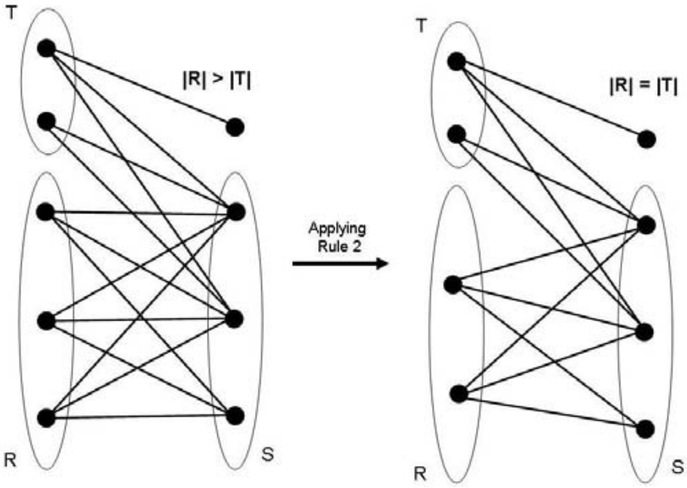

Figure 2. Applying Reduction Rule 2.

Lemma 1. *For any critical independent set B, there is an optimal solution of the Bicluster Editing Problem in which any two vertices v_1 and v_2 from B end up in the same biclique.* [Guo et al. 2008]

The following two data reduction rules were proposed in [Guo et al. 2008]:

Rule 1. *Remove from the graph all connected components that are bicliques.*

Rule 2. *Consider a critical independent set R. Let S = N(R) and T = $N(S) \backslash R$. If $|R| > |T|$, then remove arbitrary vertices from R until $|R| = |T|$.*

As illustrated in Figure 2, a vertex from R was removed of the problem without decreasing the solution optimality. [Guo et al. 2008] proves the correctness of Rule 2.

New proposed data reduction rule

Consider the generic Bicluster Editing problem, where the edges that cross partitions are weighted.

New Rule. *Consider a critical independent set R and S = N(R). Group all vertices of R into a single vertex v; all parallel edges should be grouped and their weights accumulated into a single edge e.*

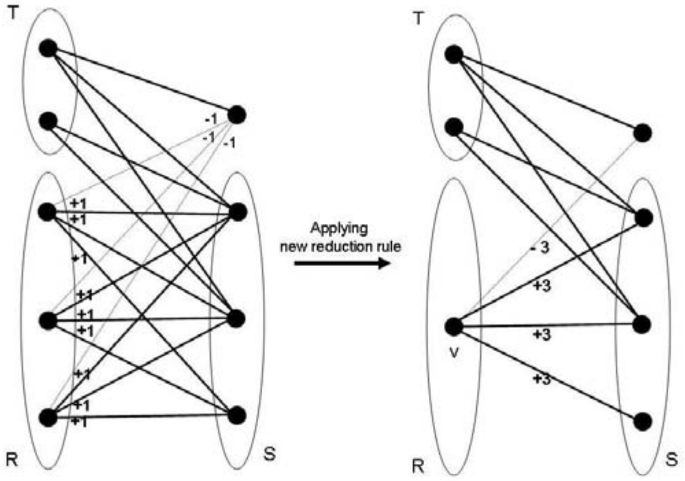

Figure 3. Applying the New Reduction Rule.

As illustrated in Figure 3, the three vertices from R were grouped into a single vertex v, and the parallel edges grouped as well; their accumulated weights are $+3$ or -3.

4. Approximation Algorithms

In this section, we present three approximation algorithms for the Bicluster Editing Problem. The first one, called Noga Approximation, is based on a LP relaxation; the second one, called Randomized 4-Approximation, was proposed by [Guo et al. 2008]; the third one is a metaheuristic GRASP proposed in this work.

4.1. Noga Approximation Algorithm

[Amit 2004] presents a polynomial-time algorithm that guarantees an approximation factor of 11, i.e, the algorithm solution is at most 11 times the LP solution.

First, a relaxation was defined for the LP model presented in section 2. The relaxation is obtained by replacing the integer constraints of $x_{ij} \in \{0, 1\}$ by linear programming (LP) constraints $0 \leq x_{ij} \leq 1$. Under this LP formulation, we refer to x_{ij} as the distance between i and j. Intuitively, points (nodes) that are close should be placed in the same bicluster ,and points that are far should be placed in different biclusters.

Using the concept of distance between vertices and the result obtained by the linear relaxation, [Amit 2004] describes the following algorithm:

1. Let $S = V \cup U$. Repeat the following steps:

2. Pick an edge (u, v) with $x_{uv} \leq \frac{1}{11}$.

Let N_u and N_v be the set of vertices within a distance of at most $\frac{5}{11}$ from u and v, respectively (not including u and v themselves).

Similarly, let N'_u and N'_v be the set of vertices within a distance of at most $\frac{3}{11}$ from u and v, respectively (again, not including u and v).

In addition, denote by αu (αv) the average distance of the vertices in N_u from u (N_v from v). If $N_u = \emptyset$ then define $\alpha u = 1$.

3. Let

$$B = \left\{ \begin{array}{cr} \{v, u\} & \text{if } \alpha_u, \alpha_v > \frac{3}{11} \\ & \text{or if } \frac{1}{11} < \alpha_u \leq \frac{3}{11}, \alpha_v > \frac{3}{11} \\ & \text{or if } \frac{1}{11} < \alpha_v \leq \frac{3}{11}, \alpha_u > \frac{3}{11} \\ \{v, u\} \cup N_v \cup N_v & \text{if } \alpha_u, \alpha_v \leq \frac{3}{11} \\ \{v, u\} \cup N'_v \cup N_u & \text{if } \alpha_u \leq \frac{1}{11}, \alpha_v > \frac{3}{11} \\ \{v, u\} \cup N_v \cup N'_u & \text{if } \alpha_v \leq \frac{1}{11}, \alpha_u > \frac{3}{11} \end{array} \right\}$$

Output B as a bicluster, let $S = S \backslash B$, and return to step 2.

3. When no edges with distance smaller than $\frac{1}{11}$ are left, output all the vertices of S as singletons.

4.2. Randomized 4-Approximation Algorithm (4Approx)

[Guo et al. 2008] presents a polynomial-time randomized 4-approximation algorithm for the Bicluster Editing Problem that is based on a technique introduced by [Ailom et al. 2005]. The basic strategy of the algorithm is to randomly pick a pivot vertex v, and then randomly destroy all P_4's that contain v. The pseudo-code of the algorithm is shown in Figure 4.

```
procedure ApproxBicluster($G = (V_1, V_2, E)$)
G' ← $(\emptyset, \emptyset, \emptyset)$;
while $V_1 \cup V_2 \neq \emptyset$ do
    randomly select a pivot vertex $i \in V_1 \cup V_2$;
    $C \leftarrow \{i\} \cup N(i)$;
    for all $j \in \{v \neq i \mid N(v) \cap N(i) \neq \emptyset\}$:
        if $N(j) = N(i)$ : add $j$ to $C$
        else: add $j$ to $C$ with probability $1/2$
    transform $G[C]$ into an isolated biclique
    $G' \leftarrow G' \cup G[C]$;
    $G \leftarrow G[V \backslash C]$;
end-while
output set of edge modifications from $G$ to $G'$
end ApproxBicluster.
```

Figure 4. A randomized 4-approximation algorithm for the Bicluster Editing Problem.

Eventually, after destroying all the P_4s, the algorithm creates an isolated biclique that contains v, since a connected component in which no vertex appears in a P_4 is a biclique. This procedure is applied until the graph is a bicluster graph. Below, we describe the procedure that creates a biclique containing the pivot vertex v.

Given a pivot vertex i, it creates a vertex set C that initially contains $N[i]$. In the end this set C contains the vertices that are in the same biclique as i in the final bicluster graph. First, it adds all vertices that are in the same *critical independent set* as i. Then it randomly decides for each vertex w that is adjacent to at least one vertex of $N(i)$ whether w should be added to C. Since w is adjacent to neighbors of $N(i)$ but is not in the same critical independent set as i, there must be a P_4 that contains i and w. By randomly deciding whether i and w end up in the same biclique, the algorithm randomly decides which edge modification is made in order to destroy the P_4. After this is done for all such vertices, we output C and remove C from G. This is done until G is empty.

4.3. Metaheuristic GRASP for Bicluster Editing

In this subsection, we describe the metaheuristic GRASP - Greedy Randomized Adaptive Search Procedure - proposed in this paper to solve the Bicluster Editing Problem.

GRASP [Resende 2001] is an iterative procedure where each iteration consists of two stages: a phase of construction of the solution and a local search phase. The best solution obtained among all the iterations is considered the final solution.

In the construction phase of a solution, start with an empty set which iteratively receives an element to form a feasible solution. In this step, two aspects are analyzed at each iteration: the randomness and the adaptation.

The solutions obtained in the construction phase of GRASP are not guaranteed to be local optima, considering a given neighborhood. Therefore, the use of the second phase of GRASP is done in order to improve the solution obtained during the construction phase.

4.3.1. Construction Phase

The construction phase, illustrated in Figure 5, starts with an empty graph G'. At each iteration, a candidate list (*CL*) is created, consisting of the set $\{(i,j) \mid i \in V_1 \text{ and } j \in V_2\}$. Then an edge (i,j) is randomly chosen from the restricted candidate list (*RCL*). The *RCL* consists of the best elements in *CL* according to the value of a greedy function, called $g(i,j)$, set for each edge (i,j):

$$g(i,j) = w(i,j) + in(i,j) + diff(i,j) - out(i,j), \tag{5}$$

where:

- $w(i,j)$: represents the weight of edge (i,j);
- $in(i,j)$: sum of weights of edges $+(i,j)$ between $N_2(i)$ and $N_2(j)$;
- $diff(i,j)$: sum of weights of edges $-(i,j)$ between $N_2(i)$ and $N_2(j)$;
- $out(i,j)$: sum of weights of edges $+(i,j)$ between $N_2(i)$ and $\{v \mid v \notin N_2(j)\}$.

The best candidates satisfy the condition:

$$g(i,j) \geq g_{min} + \alpha(g_{max} - g_{min}), \tag{6}$$

where $g_{min} = \min\{g(i,j) \mid (i,j) \in CL\}$, $g_{max} = \max\{g(i,j) \mid (i,j) \in CL\}$, and $\alpha \in (0,1)$.

```
procedure ConstructGRASP($G = (V_1, V_2, E), \alpha, g(.)$)
    G' $\leftarrow (\emptyset, \emptyset, \emptyset)$;
    while $G \neq \emptyset$ do
        $CL \leftarrow \{(i,j) | i \in V_1, j \in V_2\}$;
        $RCL \leftarrow \{(i,j) \in CL \mid g(i,j) \geq g_{max} - \alpha(g_{max} - g_{min})\}$;
        (i,j) $\leftarrow$ randomly selection in $RCL$;
        $C \leftarrow N(i) \cup N(j)$;
        transform $G[C]$ into an isolated biclique;
        $G' \leftarrow G' \cup G[C]$;
        $G \leftarrow G[V \backslash C]$;
    end-while
    return($G'$).
end ConstructGRASP.
```

Figure 5. Algorithm GRASP: construction phase.

Finally, after obtaining *RCL* and choosing a random $(i,j) \in RLC$, we add bicluster C, with vertex set $N(i) \cup N(j)$, into solution G', and remove C from G. The stop condition for the construction phase is $V(G) = \emptyset$.

4.3.2. Local Search

In the local search phase, solutions surrounding the solution obtained in the construction phase will be generated, using neighborhood movements proposed in this paper.

```
procedure Mov-Vertex($G_{constr} = (V_1, V_2, A), f(.)$)
    $G \leftarrow G_{constr}$;
    forall $c_i \in G_{constr}[C]$ do
        forall $v \in c_i$ do
            forall $c_j \in G_{constr}[C \backslash C_i]$ do
                $G_{ls} \leftarrow G_{constr}$;
                remove $v$ from $G_{ls}[c_i]$;
                $G_{ls}[c_j] \leftarrow G_{ls}[c_j] \cup \{v\}$;
                if $f(G) > f(G_{ls})$ then
                    $G \leftarrow G_{ls}$;
            end-forall
        end-foral
    end-forall
    return($G$).
end Mov-Vertex.
```

Figure 6. Local search *Mov-Vertex*.

Local Search *Mov-Vertex*: Consider a solution G formed by a set C of biclusters. For each bicluster $c_i \in C$, scroll all vertices $v \in c_i$, remove v from c_i, and add v to another bicluster $c_j \in C \backslash c_i$. The procedure returns the best solution, a neighbor of G, obtained by moving any vertex between any two biclusters. This algorithm is illustrated in Figure 6.

5. COMPUTATIONAL RESULTS

All algorithms tested in this work were developed in C++ with the aid of the mathematical solver CPLEX 11. All computational experiments were done on a machine consisting of four Intel Core 2 Quad, each with the following specification: 4 processors at the speed of 2.33 GHz with 4 GB of RAM, running the operating system Linux Ubuntu 9.04.

In Section 3.1, we have observed that the most difficult problems for Bicluster Editing are generated with probabilities $p \in [0.6, 0.7]$. For our computational experiments, four groups of random instances were generated using the algorithm described in Section 3.1. Each group has five pairs of values (n, m), where $n = |V_1|$ and $m = |V_2|$, and for each pair two values of p are considered. Thus, all groups contains 40 random instances. The sizes of the instances in each group BC_j are:

$$BC_1 = \{(5,7); (6,8); (6,12); (7,11); (6,20)\}$$
$$BC_2 = \{(10,16); (20,23); (16,30); (20,35); (24,40)\}$$
$$BC_3 = \{(28,46); (30,41); (30,50); (35,45); (40,40)\}$$
$$BC_4 = \{(37,54); (30,90); (40,70); (50,50); (40,100)\}$$

5.1. Comparison of computational results between Rule 2 and New Rule.

The comparison of the reduction rules described in Section 3.2 was performed as follows. We have implemented the LP for the Bicluster Editing Problem presented in Section 2 using the solver CPLEX. In the first run we have used an instance with no reductions, in the second we have applied **Rule 1** and **Rule 2** on the same instance before passing it to the solver, and in the third we have similarly applied **Rule1** and **New Rule** on the instance. The experiments used 12 instances whose size enabled the solver to find solutions. All executions achieved the same final value for the number of editions of the problem.

Instance	**BCE(CPLEX)**		**R1 + R2**			**R1 + NR**		
$\|V_1\| * \|V_2\|$	E^*	*Time(ms)*	$\|V_1\| * \|V_2\|$	*gap* (%)	*Time(ms)*	$\|V_1\| * \|V_2\|$	*gap* (%)	*Time(ms)*
35	8	37	35	0.00	37	**30**	**14.29**	20
35	7	12	35	0.00	10	**30**	**14.29**	8
48	11	79	48	0.00	85	48	0.00	86
48	12	41	48	0.00	36	**36**	**25.00**	24
72	17	951	72	0.00	950	**60**	**16.67**	279
72	20	1221	72	0.00	1198	72	0.00	1226
77	19	2173	77	0.00	2136	**70**	**9.09**	967
77	21	1196	77	0.00	1228	77	0.00	1222
120	30	15118	120	0.00	15132	**90**	**25.00**	2936
120	28	221	120	0.00	224	**78**	**35.00**	104
160	47	1136304	160	0.00	1119979	160	0.00	1088038
160	48	126672	160	0.00	130006	160	0.00	123177

Table 1. Comparison of computational results between Rule 2 and New Rule.

For each instance of Table 1, the first column shows the dimensions of the instance tested, and the remaining columns are divided into three groups: **BCE(CPLEX)**, **R1+R2** and **R1+NR**. In **BCE(CPLEX)**, column E^* denotes the optimal value and the column *Times(ms)* indicates the running time (in milliseconds) spent solving the instance. For the groups of columns **R1+R2** and **R1+NR**, we have: column $|V_1| * |V_2|$ indicates the dimension of the problem after the reduction, *gap*(%) indicates the percentage reduction with respect to the dimension of the original instance, and column *Times(ms)* indicates the running time in milliseconds.

Due to the complexity of the generated problems, Rule 2 did not achieve a reduction of the dimensions in any instance; moreover, its computational time is worse in some cases due to the processing overhead of the method. The new reduction rule have reduced on average the size of the original data by 11,6%, obtaining a reduction of 35% for the instance $n = 6$ and $m = 20$. The instances with their sizes reduced gained, on average, a reduction of 76,6% in computational time.

5.2. Comparison of computational results between approximation algorithms

To compare the approximation methods presented in Section 4, all 40 random instances were used. The NogaApprox method (a deterministic algorithm) was run one time for each instance. On the other hand, 4Approx and GRASP methods were run five times for each instance; each run performed 2000 iterations, in both algorithms. The value used for parameter α in GRASP was 0.5, which in empirical analysis presented better results.

Instance		NogaApprox		4Approx		GRASP	
$\lvert V_1\rvert$	$\lvert V_2\rvert$	E	*Time(ms)*	E	*Time(ms)*	E	*Time(ms)*
5	7	21	50	14	17	**10**	85
6	8	29	17	15	25	**11**	108
6	12	43	27	23	29	**17**	115
7	11	46	43	27	33	**20**	118
6	20	72	49	44	42	**32**	217
10	16	96	177	64	69	**51**	358
20	23	276	2934	186	167	**170**	1477
16	30	288	3368	196	174	**167**	1269
20	35	420	10891	296	253	**260**	1852
24	40	576	33725	398	352	**349**	2962
28	46	773	82766	565	484	**481**	4020
30	41	738	69028	522	453	**463**	5107
30	50	900	165799	636	573	**590**	5203
35	45	945	213577	682	592	**606**	6293
40	40	960	172651	694	608	**616**	6964
37	54	1199	662523	866	790	**777**	8377
30	90	-	-	1122	1203	**1070**	13939
40	70	-	-	1234	1175	**1076**	11237
50	50	-	-	1096	986	**1000**	4636
40	100	-	-	1766	1907	**1590**	23627

Table 2. Comparion of computational results between NogaApprox, 4Approx and GRASP $(p = 0.6)$**.**

For each instance of Tables 2 and 3, the first column shows the dimensions of the instances tested, and the remaining columns are divided into three groups: **NogaApprox**, **4Approx** and **GRASP**. In each group of columns we have: column E indicates the best solution found for each method, and column *Times(ms)* indicates the running time in milliseconds.

The Noga approximation algorithm fail to complete its execution in large problems (marked with -), because of the amount of memory allocated. The metaheuristic GRASP obtained the same result as the other approximation algorithms in 3 problems, obtaining better solutions in the remaining 37 problems. GRASP compared to the 4Approx algorithm gets, on average, a gain of 13.3% in their solutions, and spends a reasonable computational time, running the largest problem in 23.6 seconds.

Instance		NogaApprox		4Approx		GRASP	
$\|V_1\|$	$\|V_2\|$	*E*	*Time(ms)*	*avgE*	*Time(ms)*	*avgE*	*Time(ms)*
5	7	**7**	601	9	15	**7**	38
6	8	34	11	14	20	**12**	70
6	12	50	52	24	31	**22**	106
7	11	54	72	**22**	39	**22**	82
6	20	84	57	36	37	**28**	137
10	16	112	158	52	72	**48**	186
20	23	322	2366	138	179	**131**	512
16	30	336	2709	164	187	**144**	604
20	35	490	7117	246	268	**186**	1455
24	40	672	20692	302	381	**288**	1217
28	46	902	60889	472	528	**386**	1532
30	41	861	43665	405	487	**350**	2467
30	50	1050	96989	522	615	**450**	1962
35	45	1103	130104	522	640	**472**	1978
40	40	1120	128531	**480**	646	**480**	3597
37	54	1399	282157	599	846	**593**	4951
30	90	-	-	980	1291	**808**	8439
40	70	-	-	1036	1278	**840**	4459
50	50	-	-	808	1075	**748**	4529
40	100	-	-	1292	2037	**1190**	9813

Table 3. Comparion of computational results between NogaApprox, 4Approx and GRASP $(p = 0.7)$**.**

6. Conclusions and Future Work

In this paper we address the NP-hard Bicluster Editing Problem, which aims to transform an input bipartite graph $G = (V, E)$ into a vertex-disjoint union of complete bipartite subgraphs by editing (adding and/or removing) the smallest possible number of edges.

Due to lack of empirical testing in the literature, we have proposed and analyzed an algorithm for generating random bipartite graphs and, using the metric of the number of editions, we have identified that the more difficult problems are generated when $p \in [0.6, 0.7]$.

In order to reduce the complexity of the instances tested, reduction rules are defined, allowing the reduction of the size of input graphs without compromising the optimality of the solution. In this paper we have proposed a new rule that uses the concept of weighted graphs, accumulating the weights of the original edges by collapsing vertices belonging to the same critical independent set. We have compared the proposed reduction rule with Rule 2 described in the literature. Rule 2 failed to reduce the 12 instances used for the comparison, while the new proposed rule reduced 7 of the instances, reaching a reduction of 35% in the size of the problem $|V_1| * |V_2| = 120$.

We have also proposed a GRASP for the Bicluster Editing Problem, and compared its result with two approximation algorithms. For a total of 40 test instances, the GRASP method achieved better results than the algorithms in the literature in 37 problems. The average gap improvement of the solutions was 13.3%, and its computational time was below 30 seconds in all instances tested.

As a future work, we intend to propose new neighborhood movements, such as *join* and *break* biclusters, and to develop new metaheuristics for the Bicluster Editing Problem, such as Iterated Local Search.

References

Ailom, N., Charikar, M., and Newman, A. (2005). Aggregating inconsistent information: ranking and clustering. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 684–693, NY, USA. ACM.

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

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

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

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

Gilbert, E. N. (1959). Random graphs. In *Annals of Mathematical Statistics*, volume 3, pages 1141–1144.

Guo, J., Hüffner, F., Komusiewicz, C., and Zhang, Y. (2008). Improved algorithms for bicluster editing. In *TAMC'08*, pages 445–456.

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

Madeira, S. C. and Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. In *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, volume 1, pages 24–45. 10.1109/TCBB.2004.2.

Protti, F., da Silva, M., and Szwarcfiter, J. (2006). Applying modular decomposition to parameterized bicluster editing. In Bodlaender, H. and Langston, M., editors, *Parameterized and Exact Computation*, volume 4169 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg.

Resende, M. (2001). Greedy randomized adaptive search procedures. In Floudas, C. A. and Pardalos, P. M., editors, *Encyclopedia of Optimization*, pages 913–922. Springer US.

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

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