Integer Programming Formulation and GRASP for the Non-Automatic Bicluster Editing Problem

Teobaldo L. Bulhões Júnior[1],Gilberto F. de Sousa Filho[2], Lucídio dos Anjos F. Cabral[1], Fábio Protti[3], Luiz Satoru Ochi[3]

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

[2]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niteroi - RJ - Brazil

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

Abstract. *This work addresses the Non-Automatic Bicluster Editing Problem (NABEP). Given a bipartite graph G and an integer k (the number of biclusters), the objective is to make the least number of editions (additions or deletions of edges) in G in order to make it a biclustered graph, i.e., a disjoint union of exactly k complete bipartite subgraphs. This problem belongs to the class NP-Hard, once it can be reduced to the Bicluster Editing Problem, in which the number of biclusters is not fixed. To our knowledge, we are the first to deal with this problem. To solve it, we propose an Integer Programming Formulation and a GRASP containing a construction heuristic based on the intersection neighborhood set and a local search phase composed by three neighborhood movements procedures. The mathematical model and algorithm were tested on a set of randomly generated instances and have shown to be efficient in solving the problem.*
KEYWORDS. Biclustering, GRASP, Integer Programming.

1. Introduction

We address in this paper the study of the Non-Automatic Bicluster Editing problem (NABEP). In this problem, given a bipartite graph $G = (V_1, V_2, k)$ and an positive integer $k \leq |V_1| + |V_2|$ (number of biclusters), the goal is to achieve the smallest number of editions (additions or deletions of edges) in G in order to make it a biclustered graph, i.e., a disjoint union of exactly k complete bipartite subgraphs. The NABEP is a variation of the Bicluster Editing Problem (BEP). The idea, in both problems, is to perform partitioning in a collection of biclustered data, so that the elements belonging to the same bicluster have greater similarity to each other. To our knowledge, we are the first to deal with this problem.

The concept of grouping data into clusters arises in numerous contexts and disciplines. This subject has been extensively studied and various exact algorithms [Rahmann et al. 2007, Böcker et al. 2008], approximations and heuristics [Rahmann et al. 2007, Wittkop et al. 2010] were proposed, in which the goal is to partition a data set into clusters such that elements within a cluster are similar, while between clusters there is 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 in which every connected component is a clique. A simple clustering model is then the Cluster Editing problem [Bansal et al. 2004, Shamir et al. 2004]: find a minimum set of edges to add or delete to make the graph a cluster graph.

In some settings, the standard clustering model is not satisfactory. An important example, showed by [Guo et al. 2008], is clustering of gene expression data, in which, under a 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].

We can identify some applications for the NABEP in different areas of knowledge, among them:

Data Mining: data is typically stored in a database (data matrix), in which each record has a set of attributes. In applications with hundreds and thousands of records, an alternative to discover new information and knowledge occurs through biclustering.

Imagine books to be sold in a bookstore. The goal is to discover hidden patterns. Discover similar purchases can help the bookstore in recommending new products, identifying patterns in its customers, creating association rules to assist in promotions etc. All this information would not be noticeable, since many of the buying patterns are not apparent. By pairing shops and products (shop register versus product purchased), a biclustering algorithm can give us a good view of the hidden relationships in data.

Multicast network design: a multicast session is defined as a subset of clients that require the same information. Each client may require several multicast sessions. The main limitation is that a telecommunication network does not support multicast control over many sessions simultaneously. The solution is to group sessions in a limited number of sessions.

An example use for a multicast session is TV over IP, in which a subset of clients may require the same channel or the same subset of channels during a given period of time. In this problem, the biclusters are formed by sets of clients and their required channels.

In this context, we propose an Integer Programming Formulation and approximative strategies to solve the NABEP, which is described in section 2. Section 3 describes the proposed formulation, while sections 4 discusses the GRASP for the NABEP. Furthermore, we present computational results in section 5 and concluding remarks in section 6.

2. Non-Automatic Bicluster Editing Problem

Preliminaries. We consider only undirected bipartite graphs $G = (V_1, V_2, E)$. Let P_4 an induced path containing 4 vertices, $ijkl$, such that 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 $N[v]$. Moreover, we extend this notation to

vertex sets, i.e., for a vertex set S, $N(S) = (\bigcup_{v \in S} N(v)) \backslash S$. For a vertex v, $N_2(v) = N(N(v)) \backslash \{v\}$ is the set of vertices that are two edges apart from v. Analogously to $N[v]$, $N_2[v] = N(N(v)) \cup \{v\}$. For a graph $G = (V_1, V_2, E)$, if the edge $(i, j) \in E$, then its weight ($w(i, j)$) is $+1$ and it is called an *active* edge; if $(i, j) \notin E$, $w(i, j) = -1$ and it is called an *inactive* edge.

The Figure 1(a) shows a small instance of the problem, in which the partitions V_1 and V_2 have dimensions 4 and 3, respectively, and the desired number of biclusters is $k = 2$. The edges cross the partitions and connections between vertices of the same partition are forbidden.

For a given solution to be viable, the following conditions must be satisfied:

- Each partition must be divided in k subsets B whose elements have the same neighborhood;
- Each vertex $v \in B$ has $N_2[v] = B$;
- There is no formation of P_4.

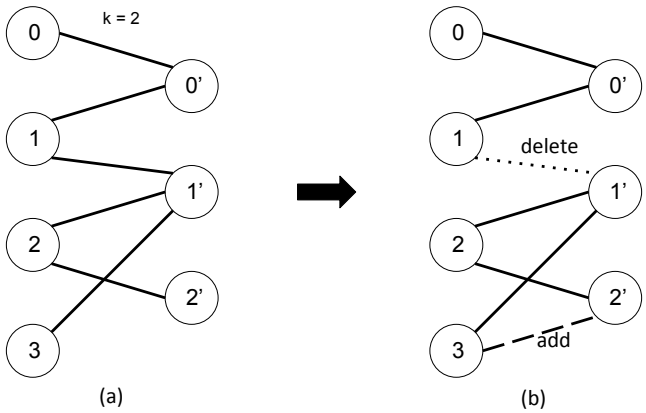

Figure 1. (a) An instance of the problem. (b) Optimal solution of the instance.

The figure 1(b) shows a solution composed by two biclusters, bicluster K_1, which has vertex sets $\{0, 1\} \subseteq V_1$ and $\{0'\} \subseteq V_2$, and bicluster K_2, which has vertex sets $\{2, 3\} \subseteq V_1$ and $\{1', 2'\} \subseteq V_2$. As we can see, the edge $(1, 0)$ was removed and the edge $(3, 2)$ was added to the solution; for this reason, we have two editions, exactly the optimal solution of the instance.

The related works in the literature tackle the generalization of NABEP, the Bicluster Edition Problem - BEP, in which the number of biclusters in the solution is not fixed and whose unique goal is to minimize the number of editions needed to make the graph biclustered. [Amit 2004] proves the NP-Hardness of BEP and presents a 11 factor approximative procedure based on the relaxation of the linear programming model. Using a simple *branching* strategy, the BEP can be solved with complexity $O(4^k + m)$ [Protti et al. 2006], where m is the number of edges of the graph and k is the number of biclusters. [Guo et al. 2008] proposed two data reductions rules to the problem and a 4 factor approximative procedure based on a random heuristic. [Sousa et al. 2012] pro-

posed a new data reduction rule and the metaheuristics GRASP and VNS (besides their hybridization GRASP + VNS) for the BEP.

3. Integer Programming Formulation for the NABEP (IP-NABEP)

In this formulation, the biclusters are indexed by the integers in the set $\{1, ..., k\}$. Let B_i be the bicluster indexed by the integer i. The formulation is as follows:

Data:

k : number of biclusters of the solution.

Sets:

V_1, V_2: vertices partitions of the graph.
$+ij$: $\{(i,j) \mid i \in V_1 \wedge j \in V_2 \wedge w(i,j) = +1\}$, i.e, active edges between V_1 and V_2.
$-ij$: $\{(i,j) \mid i \in V_1 \wedge j \in V_2 \wedge w(i,j) = -1\}$, i.e, inactive edges between V_1 and V_2.
K: {1,2,..,k}. Indices of the biclusters.

Decision variables:

x_{ijk} : if the edge (i,j) belongs to the bicluster B_k, then $x_{ijk} = 1$. Otherwise, $x_{ijk} = 0$.
y_{ik} : if the vertex i is in the bicluster B_k, then $y_{ik} = 1$. Otherwise, $y_{ik} = 0$.

$$\text{Minimize} \sum_{+ij}(1 - \sum_{k \in K} x_{ijk}) + \sum_{-ij} \sum_{k \in K} x_{ijk} \tag{1}$$

subject to:

$$x_{ijk} + 1 \geq y_{ik} + y_{jk}, \forall i \in V_1, \forall j \in V_2, \forall k \in K, \tag{2}$$

$$2x_{ijk} \leq y_{ik} + y_{jk}, \forall i \in V_1, \forall j \in V_2, \forall k \in K, \tag{3}$$

$$\sum_{k \in K} y_{ik} = 1, \forall i \in (V_1 \cup V_2), \tag{4}$$

$$\sum_{i \in (V_1 \cup V_2)} y_{ik} \geq 1, \forall k \in K, \tag{5}$$

$$|V_2| * \sum_{i \in V_1} y_{ik} + 1 \geq \sum_{j \in V_2} y_{jk}, \forall k \in K, \tag{6}$$

$$|V_1| * \sum_{j \in V_2} y_{jk} + 1 \geq \sum_{i \in V_1} y_{ik}, \forall k \in K, \tag{7}$$

$$x_{ijk}, y_{lk} \in \{0,1\}, \forall i \in V_1, \forall j \in V_2, \forall l \in (V_1 \cup V_2), \forall k \in K, \tag{8}$$

The objective function (1) minimizes the number of edges removed from the graph (first sum) and the number of edges added to the graph (last sum). The constraint sets (2) and (3) ensure that the edge (i,j) belongs to the bicluster B_k if, and only if, the vertices i and j are in the bicluster B_k - that is, $y_{ik} = 1$ and $y_{jk} = 1$. Note that (2)

and (3) automatically guarantee that there is no P_4 in the solution. The constraint set (4) associates each vertex of the graph G to exactly one bicluster, while the constraint set (5) avoids the presence of empty biclusters in the solution. Finally, constraint sets (6) and (7) forbid biclusters with zero vertices in one partition and more than one vertex in the another partition. (8) concerns about the integrality of the variables.

4. GRASP

GRASP [Resende 2001] is an iterative procedure in which each iteration consists of two phases: a solution construction phase and a local search phase. The best solution obtained between all iterations is considered the final solution. This section presents the proposed GRASP for the NABEP.

4.1. Construction Phase

The construction phase, illustrated in Figure 2, starts with an empty graph *G'*. At each iteration, it creates a candidate list (*CL*), consisting of the set $\{(i,j) \mid i \in V_1 \wedge j \in V_2\}$, and a restricted candidate list (*RCL*), from which an edge (i,j) is randomly chosen. The *RCL* corresponds to the best candidates of the *CL* according to the greedy function $g(i,j)$ that is applied to each edge:

$$g(i,j) = w(i,j) + in(i,j) + diff(i,j) - out(i,j), \tag{9}$$

where:

- $w(i,j)$: represents the weight of the edge (i,j);
- $in(i,j)$: sum of weights of active edges between i and $N_2(j)$ and between j and $N_2(i)$;
- $diff(i,j)$: sum of weights of inactive edges between i and $N_2(j)$ and between j and $N_2(i)$;
- $out(i,j)$: sum of weights of active edges between i and $\{v \mid v \notin \{N_1(i) - N_2(j)\}\}$ and between j and $\{v \mid v \notin \{N_1(j) - N_2(i)\}\}$.

The elements in *RCL* satisfy the condition:

$$g(i,j) \geq g_{min} + \alpha(g_{max} - g_{min}), \tag{10}$$

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

After obtaining *RCL* and choosing a random $(i,j) \in RLC$, the bicluster C, which has vertex set $N(i) \cup N(j)$, is added to the solution G' and all vertices $v \in N(i) \cup N(j)$ are removed from G. The construction algorithm repeats this iteration while there are candidates edges.

There is no guarantee that, when there is no more candidate edges, the produced graph, G', has the desired number (k) of biclusters. Thus, it is necessary to call the procedure $AdjustBiclusters$ to perform ***Break-Bicluster*** movements (which break a bicluster into two biclusters), if the number of biclusters of G' is less than k, or ***Join-Bicluster*** movements (which joins two biclusters), if the number of biclusters of G' is greater than k. We give further details about ***Break-Bicluster*** and ***Join-Bicluster*** in the next subsection.

procedure ConstructGRASP($G = (V_1, V_2, E)$, α, $g(.)$)
1. G' $\leftarrow (\emptyset, \emptyset, \emptyset)$;
2. **while** $G \neq \emptyset$ **do**
3. $CL \leftarrow \{(i,j) | i \in V_1 \wedge j \in V_2\}$;
4. $RCL \leftarrow \{(i,j) \in CL \mid g(i,j) \geq g_{max} - \alpha(g_{max} - g_{min})\}$;
5. $(i,j) \leftarrow$ random selection in *RCL*;
6. $C \leftarrow N(i) \cup N(j)$;
7. transform $G[C]$ into an isolated biclique;
8. $G' \leftarrow G' \cup G[C]$;
9. $G \leftarrow G[V \backslash C]$;
10. **end-while**
11. $AdjustBiclusters(G')$;
12. return(G').

end ConstructGRASP.

Figure 2. Algorithm GRASP: construction phase.

4.2. Local Search Phase

Once a feasible solution is available, some neighborhood movements can be applied. For all neighborhood movements procedures, consider a solution *G* formed by a set *C* of biclusters. Figure 3 illustrates all movements.

Mov-Vertex: for each bicluster $c_i \in C$, consider each vertex $v \in c_i$, cut v from c_i and add it to another bicluster $c_j \in C \backslash c_i$.

Join-Bicluster: for each bicluster pair $\{c_i, c_j\} \in C$, create a new bicluster c_k by the union of biclusters c_i and c_j, add c_k to *C* and remove c_i and c_j from *C*.

Break-Bicluster: for each bicluster $c_i \in C$, create two new biclusters, c_j and c_k, by the separation of the vertices of c_i, add c_j and c_k to *C*, remove c_i from *C*.

To divide a bicluster into two biclusters, we use the function $bind(v)$, which examines how the vertex *v* is connected to the elements of its current bicluster:

$$bind(v) = bindin(v) - bindout(v), \tag{11}$$

where $bindin(v)$ represents the number of edges *ij* $\in$ *+ij* between *v* and any vertex within its current bicluster and $bindout(v)$ represents the number of edges *ij* $\in$ *+ij* between *v* and any vertex outside its bicluster. The breakdown of bicluster c_i uses the following algorithm: be *V* a vertex partition of bicluster c_i; all vertices $v \in V$ with $bind(v) < 0$ will be cut from c_i to form the bicluster c_j alongside every vertex $i \in N(v)$ such that $bind(i) <= +1$. The remaining elements in c_i form the second bicluster c_k.

All neighborhood movements must preserve the viability of the solution. Therefore, for the number of biclusters of the solution remains equal to *k*, the movement ***Mov-Vertex*** can not be applied to a bicluster that has only one vertex, as well as any movement ***Join-Bicluster*** must be followed by a movement ***Break-Bicluster*** (and vice versa). These restrictions culminate in movements ***JoinBreak-Bicluster***, ***Join-Bicluster*** followed by ***Break-Bicluster***, and ***BreakJoin-Bicluster***, ***Break-Bicluster*** followed by ***Join-Bicluster***.

In the Local Search phase, neighbors (close solutions) of the solution obtained

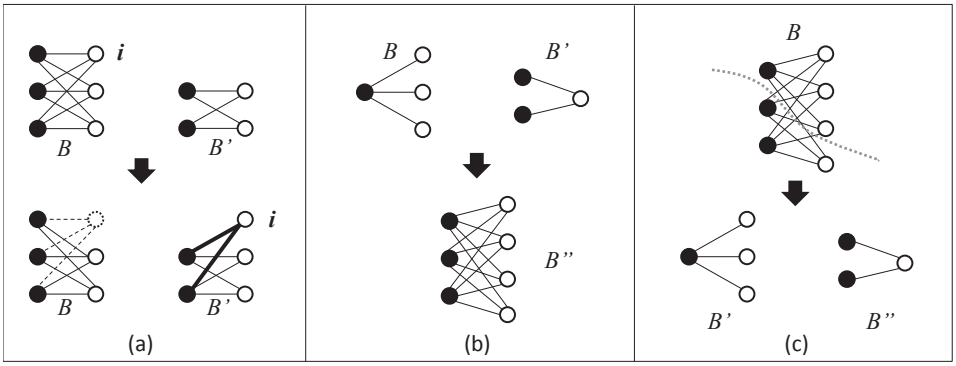

Figure 3. Mov-Vertex (a), Join-Bicluster (b) and Break-Bicluster (c).

during the construction phase are generated through the method *Variable Neighbourhood Descent* (VND), described in [Hansen et al. 2008], applying the neighborhoods of the set $N = \{$ *BreakJoin-Bicluster*, *JoinBreak-Bicluster*, *Mov-Vertex* $\}$, in the order they are.

5. Computational Results

All algorithms tested in this work were developed with the C++ language and with the aid of the mathematical solver CPLEX 11. All computational experiments were done on a Intel Core 2 Quad machine (4 processors of 2.33 GHz and 4 GB of RAM), running the operating system Linux Ubuntu 9.04.

The instances used in this work were defined in [Sousa et al. 2012]. Each value pair (n, m) represents $n = |V_1|$ and $m = |V_2|$, being chosen instances with density 0.5, which represents the probability of existence of the edge during the random construction. These instances have the following dimensions: $\{(5, 7); (7, 11); (6, 12); (6, 20); (10, 16); (16, 30); (24, 40); (20, 35)\}$.

For the comparison between the proposed Integer Programming Formulation (IP-NABEP), which was implemented using the solver CPLEX, and the GRASP, were used the instances $\{(5, 7); (7, 11); (10, 16)\}$, varying the number of biclusters k, in each instance, from 1 to $n + m$. The IP-NABEP, due to its deterministic nature, was executed only once, while the GRASP was performed 20 times for each instance with 100 iterations in each execution, obtaining its minimum and medium solution. The results are shown in the table below.

For each line of Table 1, the first three columns represent the dimensions of the tested instance. The remaining columns are divided into two groups: *IP-NABEP* and *GRASP*. In the *IP-NABEP* group, the column E^* denotes the optimal value and the column *Time* indicates the computing time (in milliseconds) spent solving the instance (we set the maximum computing time to 6 hours). In the *GRASP* group, the column E_{min} indicates the best value found by the method, E_{avg} represents the average value of its solution, $GAP_{avg}(\%)$ shows the percentage reduction between E_{avg} and E^* and T_{avg} is the average computing time in milliseconds.

The Table 1 does not bring all values of k for each instance due to space restrictions

Instance			IP-NABEP		GRASP			
$\|V_1\|$	$\|V_2\|$	k	E*	Time	E_{min}	E_{avg}	GAP_{avg} (%)	T_{avg}(ms)
5	7	3	**7**	275	**7**	7	0	49.1
		4	**8**	523	**8**	8	0	47.7
		6	**10**	5740	**10**	10	0	52.1
		7	**11**	5755	**11**	11	0	54.55
		8	**12**	6812	**12**	12	0	56.9
		9	**14**	9000	**14**	14	0	57.65
7	11	3	**11**	252	**11**	11	0	143.1
		4	**13**	1135	**13**	13	0	140.05
		6	**17**	18223	**17**	17	0	166.15
		7	**18**	29860	**18**	18.85	4.7	163.3
		9	**22**	46175	**22**	22	0	157.65
		10	**23**	103743	**23**	23	0	159
10	16	2	**46**	3873	**46**	46	0	278.2
		4	**39**	49042	**39**	39.25	0.6	274.5
		5	**40**	242543	**40**	40.35	0.87	298.7
		7	**42**	617805	**42**	42.55	1.3	340
		8	**43**	2201062	**43**	43.85	1.9	367.5
		10	**46**	12962461	**46**	47	2.17	381

Table 1. Computational results between IP-NABEP and GRASP.

in this paper [1]. The GRASP procedure found the best result in all instances (E_{min}) - even its average value reached optimal value in 12 of the 18 instances tested. The GRASP still showed to be efficient in its computational time. As an example, the IP-NABEP spent 12,962 seconds to execute the instance (10, 16) with $k = 10$, while the GRASP required, in average, 381 milliseconds.

The larger instances were not executed by IP-NABEP due to its high demand of computational resources (memory and CPU time). Therefore, we compared only the construction heuristic (CH) to the GRASP with the instances of medium size { (6, 12); (6, 20); (16, 30); (24, 40); (20, 35) }. The construction heuristic is the construction procedure of GRASP with $\alpha = 0$. The heuristic CH was executed only once (since it is deterministic) and the GRASP was performed 20 times for each instance with 100 iterations in each execution, obtaining its minimum and medium solution.

The table 2 demonstrates that the heuristic CH produces solutions closed to the GRASP solutions. For example, for the instance (20,35) with $k = 47$, the GAP between CH and GRASP is of only 3.5%. This fact becomes clearer if we consider the average GAP of all instances tested, which is 14%. As a consequence, the effort of the local search phase is reduced, allowing the GRASP to reach good computational efficiency even over medium instances, obtaining, in the worst case, an average time of 5.3 seconds ((24, 40) with k=23).

[1] More results can be found at *https://sites.google.com/site/biclustereditingproblem/technical-reports*.

Instance			CH		GRASP			
$\|V_1\|$	$\|V_2\|$	k	E	T(ms)	$\mathbf{E_{min}}$	E_{avg}	GAP_{avg} (%)	$\mathbf{T_{avg}(ms)}$
6	12	5	26	0	**15**	15	-42.3	116.9
		9	25	0	**19**	19	-24	141.05
		13	30	1	**27**	27	-10	146.2
6	20	2	54	1	**34**	34	-37.03	222.4
		7	38	0	**28**	28	-26.31	268.35
		12	36	1	**34**	34	-5.55	335.2
		17	46	1	**41**	41.25	-10.32	343.95
16	30	5	215	2	**156**	164.75	-23.37	1120.7
		11	201	6	**165**	170.2	-15.32	1527
		17	201	9	**176**	179.4	-10.74	1782.45
		23	207	13	**186**	189.35	-8.52	1960.4
		29	209	16	**196**	199.95	-4.33	1955.45
		35	225	17	**212**	214.1	-4.84	1866.35
24	40	5	478	5	**347**	359.05	-24.88	2733.5
		11	437	11	**350**	356.55	-18.4	4125.4
		17	421	19	**355**	362.1	-13.99	4981.45
		23	422	26	**368**	374.2	-11.32	5334.25
		29	421	32	**384**	386.65	-8.15	5510.4
		35	440	36	**393**	399.35	-9.23	5161.65
		41	448	40	**413**	418.7	-6.54	4981.95
		47	459	44	**429**	434.25	-5.39	4814.9
		53	466	45	**446**	446	-4.29	4722.1
20	35	5	312	2	**244**	248.45	-20.36	1741.75
		11	305	6	**243**	247.15	-18.96	2436.6
		17	299	10	**252**	255.5	-14.54	2914.3
		23	292	15	**262**	264.95	-9.26	3238.8
		29	310	21	**270**	272	-12.25	3330.25
		35	323	24	**288**	291.2	-9.84	3340.05
		41	334	26	**304**	308.85	-7.52	3176.65
		47	342	28	**330**	330	-3.5	3124.55

Table 2. Computational results between construction heuristic CH and GRASP.

6. Concluding remarks

In this paper we have addressed the Non-Automatic Bicluter Editing Problem (NABEP), which aims at making a bipartite graph G a vertex-disjoint union of k complete bipartite subgraphs, by editing the smallest possible number of edges.

We have proposed an Integer Programming Formulation for the NABEP, a construction heuristic based on the intersection neighborhood set, three neighborhood movements procedures and a GRASP. For the first 18 tested instances, GRASP achieved the optimal solution in all problems, maintaining the average computational time below 1 second. For the 29 instances of medium size, the IP-NABEP failed to find the optimal solution after 6 hours of processing.

As a future work, we intend to propose new neighborhood movements and to develop new metaheuristics for the NABEP with phases of shake of solutions, such as

Iterated Local Search (ILS) and Variable Neighborhood Search (VNS).

References

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.

Böcker, S., Briesemeister, S., and Klau, G. W. (2008). Exact algorithms for cluster editing: evaluation and experiments. In *Proceedings of the 7th international conference on Experimental algorithms*, WEA'08, pages 289–302, Berlin, Heidelberg. Springer-Verlag.

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.

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

Hansen, P., Mladenovic, N., and Moreno Perez, J. (2008). Variable neighbourhood search: methods and applications. *4OR: A Quarterly Journal of Operations Research*, 6:319–360. 10.1007/s10288-008-0089-1.

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.

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.

Rahmann, S., Wittkop, T., Baumbach, J., Martin, M., Truss, A., and Böcker, S. (2007). Exact and heuristic algorithms for weighted cluster editing. *Comput Syst Bioinformatics Conf*, 6(1):391–401.

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.

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

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.

Wittkop, T., Emig, D., Lange, S. J., Rahmann, S., Albrecht, M., Morris, J. H., Böcker, S., Stoye, J., and Baumbach, J. (2010). Partitioning biological data with transitivity clustering. *Nature Methods*, 7(6):419–420.