A Branch and Price Approach to the k-Clustering Minimum Biclique Completion Problem

Stefano Gualandi[a,1], Francesco Maffioli[b], Claudio Magni[c]

[a]*Dipartimento di Matematica, Università degli Studi di Pavia, Via Ferrata 1, 27100, Pavia, Italy*
[b]*Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci 32, 20133 Milano, Italy*
[c]*Max Planck Institute for Computer Science, Department 1: Algorithms and Complexity, Campus E1 4, 66123 Saarbrücken, Germany*

Abstract

Given a bipartite graph $G = (S, T, E)$, we consider the problem of finding k bipartite subgraphs, called "clusters", such that each vertex i of S appears in exactly one of them, every vertex j of T appears in each cluster in which at least one of its neighbors appears, and the total number of edges needed to make each cluster complete (i.e., to become a biclique) is minimized. This problem is known as k-clustering Minimum Biclique Completion Problem and has been shown strongly NP-hard. It has applications in bundling channels for multicast transmissions. Given a set of demands of services from clients, the application consists of finding k multicast sessions that partition the set of demands. Each service has to belong to a single multicast session, while each client can appear in more sessions. We extend previous work by developing a Branch and Price algorithm that embeds a new metaheuristic based on Variable Neighborhood Infeasible Search and a non-trivial branching rule. The metaheuristic is also adapted to solve efficiently the pricing subproblem. In addition to the random instances used in the literature, we present structured instances generated using the MovieLens data set collected by the GroupLens Research Project. Extensive computational results show that our Branch and Price algorithm outperforms the approaches proposed in the literature.

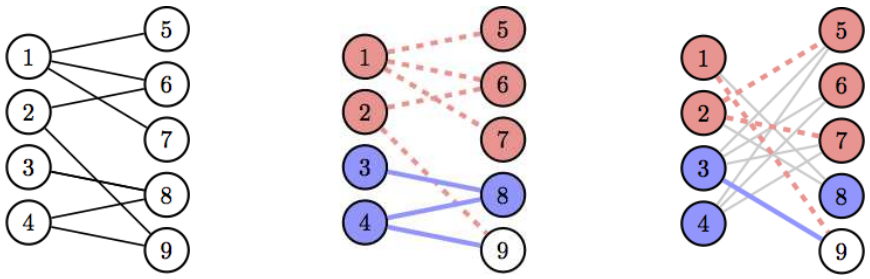
Key words: Biclique, Branch and Price, Clustering, Local Search

1. Introduction

Given a bipartite graph $G = (S, T, E)$, the k-clustering Minimum Biclique Completion Problem (k-MinBCP) consists of finding k bipartite subgraphs (clusters), such that each vertex i of S appears in exactly one subgraph, every vertex j in T appears in each cluster in which at least one of its neighbors appears, and the total number of edges to add for each subgraph to become a complete bipartite subgraph, i.e., a biclique, is minimized. This problem was introduced in (Faure et al., 2007), where it was shown to be NP-hard, and its approximability, to the best of our knowledge, remains unknown. In the literature, k-MinBCP is tackled with two approaches: in (Faure et al., 2007), it is solved with Integer Linear Programming to optimality, and it is solved heuristically using a column generation approach; in (Gualandi, 2009), it is solved with an hybrid Constraint Programming–Semidefinite Programming exact approach.

The k-MinBCP problem has a significant application in telecommunications, as shown in (Faure et al., 2007), for bundling channels in multicast transmissions. Given a set of demands of services from clients, the application consists of finding k multicast sessions that partition the set of demands. Each service has to belong to a single multicast session, while each client can appear in more sessions. This problem is represented on a bipartite graph $G = (S, T, E)$ as follows: every service i is represented by a vertex in S, and every client j by a vertex in T. The demand of a service i from a client j is represented by the edge $(i, j) \in E$. A bipartite subgraph G' of G represents a multicast session that transmits c times an unrequested service, where c is equal to the number of complementary

Email addresses: stefano.gualandi@unipv.it (Stefano Gualandi), maffioli@elet.polimi.it (Francesco Maffioli), magni@mpi-inf.mpg.de (Claudio Magni)
[1]Corresponding author.



(a) (b) (c)

Figure 1: An example of the k-MinBCP problem for k = 2: Figure 1.a shows a bipartite graph G with $S = \{1, \ldots, 4\}$ and $T = \{5, \ldots, 9\}$. Figure 1.b represents a possible 2-clustering induced by $S_1 = \{1, 2\}$ and $S_2 = \{3, 4\}$. The dashed edges belong to the first cluster induced by S_1 and those in bold to the second cluster induced by S_2. Note that vertex 9 belongs to both clusters. The complementary graph $\bar{G}$ is shown in Figure 1.c. The cost of this 2-clustering is equal to four, given by three missing edges of the first cluster and one missing edge in the second cluster, that are respectively $(1, 9)$,$(2, 5)$, $(2, 7)$, and $(3, 9)$.

edges that must be added to G' to obtain a biclique. The cost c gives a measure of the waste of bandwidth of the corresponding multicast session. Solving the k-MinBCP problem on this bipartite subgraph, is equivalent to finding k multicast sessions that minimize the overall waste of bandwidth.

In this work, we start from the column generation approach proposed in (Faure et al., 2007) and we develop a Branch and Price algorithm that embeds a new metaheuristic based on Variable Neighborhood Search (VNS) (e.g., Hansen and Mladenovic, 2001) and a non-trivial branching rule. The metaheuristic is also adapted to solve efficiently the pricing subproblem.

The outline of this paper is as follows. In Section 2, we review different mathematical formulations of k-MinBCP, and for each formulation we briefly discuss the main advantages and drawbacks. In Section 3, we present a new metaheuristic based on VNS accepting also infeasible solutions, herein called Variable Neighborhood Infeasible Search (VNIS), and we describe how it is adapted to solve the pricing subproblem in a column generation approach to k-MinBCP. In Section 4, we describe our Branch and Price algorithm along with a non trivial branching rule. In Section 5, we discuss the computational results, and, in Section 6, we mention some direction for further research.

2. Problem Formulations

Let $G = (S, T, E)$ be a bipartite graph and let $\bar{E}$ be the edge set of the complementary bipartite graph, i.e. $\bar{E} = \{S \times T\} \setminus E$. Let $K = 1, \ldots, k$ be the set of cluster indices. Figure 1 shows an example of k-MinBCP. Belowe, we review existing mathematical formulations of this problem.

2.1. Bilinear Programming Formulation

We start by reviewing the first model presented in (Faure et al., 2007). Let x_{ip} and y_{jp} be 0–1 variables indicating the p-th cluster containing node i of the shore S or node j of the shore T, respectively. Using these variables we have the following Integer Bilinear Program:

$$(\text{QP}) \quad w^* = \min \quad \sum_{p \in K} \sum_{(i,j) \in \bar{E}} x_{ip}\, y_{jp} \tag{1}$$

$$\text{s.t.} \quad \sum_{p \in K} x_{ip} = 1, \qquad \forall i \in S, \tag{2}$$

$$x_{ip} \leq y_{jp}, \qquad \forall (i, j) \in E, \forall p \in K, \tag{3}$$

$$x_{ip}, y_{jp} \in \{0, 1\}, \qquad \forall i \in S, \forall j \in T, \forall p \in K. \tag{4}$$

When the bilinear term $x_{ip}\, y_{jp}$ in the objective function is equal to 1 the edge $(i, j) \in \bar{E}$ is assigned to the p-th cluster. Therefore, the objective function (1) sums up for each cluster p the number of edges necessary to turn the bipartite subgraph into a biclique. Constraints (2) assign each node $i \in S$ to a single cluster. Constraints (3) force every $j \in T$

adjacent to a node $i \in S$ assigned to the p-th cluster to be assigned to the same cluster. Note that some nodes of T can be assigned to more than a cluster.

While the problem (1)–(4) can be solved by modern quadratic programming solver, such solvers can handle only small instances. The first ILP formulation is obtained by applying standard linearization techniques to problem (1)–(4).

Let z_{ijp} be a 0–1 variable equal to 1 whenever both x_{ip} and y_{jp} are. We obtain the following ILP formulation:

$$\text{(ILP)} \quad w^* = \min \quad \sum_{p \in K} \sum_{(i,j) \in \bar{E}} z_{ijp} \tag{5}$$

$$\text{s.t.} \quad x_{ip} + y_{jp} \leq 1 + z_{ijp}, \qquad \forall (i,j) \in \bar{E}, \forall p \in K, \tag{6}$$

$$\sum_{p \in K} x_{ip} = 1, \qquad \forall i \in S, \tag{7}$$

$$x_{ip} \leq y_{jp}, \qquad \forall (i,j) \in E, \forall p \in K, \tag{8}$$

$$x_{ip}, y_{jp} \in \{0,1\}, \qquad \forall i \in S, \forall j \in T, \forall p \in K, \tag{9}$$

$$z_{ijp} \in \{0,1\}, \qquad \forall (i,j) \in \bar{E}, \forall p \in K. \tag{10}$$

The implication constraints (6) link the variable z_{ijp} to both x_{ip} and y_{jp} by forcing z_{ijp} to 1 whenever the other two variables are. The integrality constraints for the variables y_{jp} can be relaxed into simpler constraints $y_{jp} \geq 0$, since the objective function (5), constraints (8) together with the integrality constraints of x_{ip} ensure that y_{jp} takes either value 0 or 1. In addition, it is possible to reduce both the number of constraints (6) and variables (10) by projecting for every pair (i, j) the variables z_{ijp} into a single variable z_{ij}. However, computational experiments shown that is unclear which between the two alternative is better handled by modern ILP solvers.

Unfortunately, the linear relaxation of model (5)–(10) is very weak since it is half-integer and provides a lower bound equal to zero. In fact, it is always possible to assign to every variable z_{ijp} value 0, and to the other variables x_{ip} and y_{jp} a value among $\{0, \frac{1}{2}, 1\}$.

Another weakness of the model (5)–(10) is in the number of symmetries introduced by the p indices: every permutation of the variables over those indices produces an equivalent solution. The symmetry issues arise above all during the branching phase. This issue is handled in (Faure et al., 2007) introducing breaking symmetry constraints within the model. However, in our computational experience using CPLEX as ILP solver, we always obtained better results without including such constraints. This is in contrast to what reported in (Faure et al., 2007) where the experiments were run with XPress. We suppose that the advanced techniques for handling variable symmetries introduced recently in CPLEX are more effective than the symmetry breaking constraints used in (Faure et al., 2007), which therefore are no longer used here.

2.2. Asymmetric representative formulation

An alternative formulation is introduced in (Faure, 2006) that reduces the symmetries directly into the model. The formulation is related to a formulation introduced in (Campelo et al., 2008) for the graph coloring problem. The idea is to use a 0–1 variable x_{ij} for every pair of vertices in $S \times S$ such that $i \leq j$. When variable x_{ii} is equal to one, then the vertex i is called the *representative* of its cluster. When variable x_{ij} is equal to one, then the two vertices i and j are in the same cluster; the first vertex, in this case vertex i, is the representative of the other vertex. In addition there is a 0–1 variables z_{ij} for every $(i, j) \in E$ that indicates whether we have to pay for the edge (i, j). Using these variables

we can introduce the following ILP model:

$$\text{(A-ILP)} \quad w^* = \min \quad \sum_{(i,j)\in\bar{E}} z_{ij} \tag{11}$$

$$\text{s.t.} \quad \sum_{i\in S} x_{ii} = k, \tag{12}$$

$$\sum_{i\in S, i\le l} x_{il} = 1, \qquad \forall l \in S, \tag{13}$$

$$x_{il} \le x_{ii}, \qquad \forall i, l \in S, i < l, \tag{14}$$

$$x_{hi} + x_{hl} \le z_{ij} + 1, \qquad \forall h \in S, \forall (i,j) \in \bar{E}, (l,j) \in E, \tag{15}$$

$$x_{il} \in \{0,1\}, \qquad \forall i, l \in S \times S, i \le l, \tag{16}$$

$$z_{ij} \ge 0, \qquad \forall (i,j) \in \bar{E}. \tag{17}$$

The constraint (12) ensures that there are exactly k representative, i.e., there are exactly k clusters. Constraints (13) impose that each vertex is either its own representative or it has a single representative. Constraints (14) state that if the vertex i is the representative of the vertex l, then vertex i has to be the representative of itself, as well. Constraints (15) activate the variable z_{ij} whenever the vertex $h \in S$ is the representative of the two vertices $i \in S$ and $l \in S$, and there exists a fourth vertex $j \in T$ that is adjacent to l but it is not adjacent to i. That is, the edge (i, j) is needed to complete the subgraph represented by h into a biclique. Figure 2 shows the relations among the four vertices i, j, h and l. Note that the objective function (11), constraints (15), and the integrality constraints (16) imply the integrality of the variables (17).

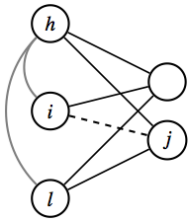


Figure 2: Asymmetric representative formulation: if node h is the representative of both node i and l, then we have to pay for the (complementary) edge (i, j).

2.3. Semidefinite Programming relaxation

A Semidefinite Programming (SDP) formulation of k-MinBCP was introduced in (Gualandi, 2009). The SDP formulation provides stronger lower bounds than both the linear relaxations of model (5)–(10) and (11)–(17). The formulation is inspired by the SDP formulations of the Max-k-Cut problem (e.g., see (Frieze and Jerrum, 1997) and (Ghaddar et al., 2009)).

Let us consider k unit vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{R}^{k-1}$ satisfying $\mathbf{a}_i^T \mathbf{a}_j = -\frac{1}{k-1}$, for $1 \le i \ne j \le k$. We label each vertex $i \in S$ with a vector of real variable $\mathbf{x}_i$. The variable vector has the domain ranging in $\{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$, i.e., $\mathbf{x}_i \in \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$. Two vertices i and j are in the same cluster if $\mathbf{x}_i^T \mathbf{x}_j = 1$, while they are in different clusters whenever $\mathbf{x}_i^T \mathbf{x}_j = -\frac{1}{k-1}$. Note that for $k = 2$, we get exactly $a_1 = -1$, $a_2 = 1$, and $x_i \in \{-1, 1\}$.

Let z_e be a variable that indicates whether we pay for the edge $e = (i, j) \in \bar{E}$ or not. The value of z_e is either $z_e = -\frac{k}{k-1}$ if there exists an edge $(l, j) \in E$ such that the vertex l is in the same cluster of i, i.e., $z_e = \mathbf{x}_i^T \mathbf{x}_l = -\frac{k}{k-1}$; or

$z_e = 1$ if such edge does not exist. Using these variables the formulation of the k-MinBCP problem is as follows:

$$w^* = \min \quad \sum_{e \in \bar{E}} \frac{1 + (k-1)z_e}{k} \tag{18}$$

$$\text{s.t.} \quad z_e \geq \mathbf{x}_i^T \mathbf{x}_l, \qquad \forall e = (i,j) \in \bar{E}, (l,j) \in E, \tag{19}$$

$$\mathbf{x}_i \in \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}, \qquad \forall i \in S, \tag{20}$$

$$z_e \in \mathbb{R}, \qquad \forall e \in \bar{E}. \tag{21}$$

Each single term in the summation of the objective function is equal to either 1, if $z_e = 1$, or is equal to 0, if $z_e = \frac{-1}{k-1}$. The SDP relaxation of the k-MinBCP problem is obtained using the labeling technique introduced for the Max-Cut problem in (Goemans and Williamson, 1995). Every vertex i of S is labeled with a unit vector $\mathbf{v}_i \in \mathbb{R}^{|S|}$. Then, two vertices i and j are considered to be in the same cluster if the angle between them is small enough, that is, if $\mathbf{v}_i^T \mathbf{v}_j = 1$. Let V be a matrix such that column i is given by vector $\mathbf{v}_i$, and let $X = V^T V$. Let $\mathbf{e} \in \mathbb{R}^{|S|}$ be a vector of all ones. The SDP relaxation of problem (18)–(21) is as follows:

$$w_{sdp} = \min \quad \sum_{e \in \bar{E}} \frac{1 + (k-1)z_e}{k} \tag{22}$$

$$z_e \geq X_{il}, \qquad \forall e = (i,j) \in \bar{E}, (l,j) \in E, \tag{23}$$

$$\text{diag}(X) = \mathbf{e}, \tag{24}$$

$$X_{ij} \geq -\frac{1}{k-1}, \qquad \forall i, j \in S, i \neq j, \tag{25}$$

$$X \succeq 0, \tag{26}$$

$$z_e \in \mathbb{R}, \qquad \forall e \in \bar{E}. \tag{27}$$

Together constraints (24)–(26) relax constraints (20) and (21) into $-\frac{1}{k-1} \leq X_{ij} \leq 1$.

2.4. Column Generation

Finally, we present a formulation with an exponential number of variables similar to the one proposed in (Faure et al., 2007): the master problem is a set partitioning problem where each column represents the subset of vertices of S that induces a cluster t. Let c_t be the cost of the t-th cluster that is equal to the number of edges that would complete the corresponding subgraph into a biclique. Let λ_t be a 0–1 variable, equal to 1 if the cluster t is part of the solution, and 0 otherwise. Let $\mathcal{T}$ be the collection of every possible cluster, and let S_t be the subset of vertices of S that belong to the t-th cluster. The master problem formulation is as follows:

$$\min \quad \sum_{t \in \mathcal{T}} c_t \lambda_t \tag{28}$$

$$\text{s.t.} \quad \sum_{t \in \mathcal{T} | i \in S_t} \lambda_t = 1, \qquad \forall\, i \in S, \tag{29}$$

$$\sum_{t \in \mathcal{T}} \lambda_t = k, \tag{30}$$

$$\lambda_t \in \{0,1\}, \qquad \forall\, t \in \mathcal{T}. \tag{31}$$

Constraints (29) are the partitioning constraints, one for each vertex i in S. Constraint (30) is the cardinality constraint on the number of cluster to be selected. Let π_i and ν be the dual multipliers of constraints (29) and (30), respectively. Then, the pricing subproblem is the problem of finding a vertex and edge weighted biclique of negative reduced cost. The vertex weights are given by the dual multipliers π_i, while the edge weights are equal to the number of edges that

would turn the subgraph into a biclique. The pricing subproblem is as follows:

$$\min \quad \sum_{(i,j)\in\bar{E}} z_{ij} - \sum_{i\in I} \pi_i x_i - \nu \tag{32}$$

$$\text{s.t.} \quad x_i + x_l \le z_{ij} + 1, \qquad \forall (i,j) \in \bar{E},\ \forall (l,j) \in E, \tag{33}$$

$$x_i \in \{0,1\}, \qquad \forall i \in S, \tag{34}$$

$$z_{ij} \ge 0, \qquad (i,j) \in \bar{E}. \tag{35}$$

Constraints (33) force the binary variable z_{ij} to be 1 if the corresponding edge is part of the biclique, and 0 otherwise. Note that an edge belongs to a biclique if it exists at least a pair of vertices i and l both in S such that a vertex $j \in T$ exists with $(i,j) \in \bar{E}$ and $(l,j) \in E$.

3. Computing Upper Bounds

We introduce in this section a new metaheuristic based on VNIS that finds primal solutions to k-MinBCP. Since the implementation of the algorithm is quite standard, we report in the following only the main features of our algorithm. For an in-depth survey on VNS algorithms see (Hansen and Mladenovic, 2001). The metaheuristic is also adapted to solve the pricing subproblem (32)–(35).

3.1. Neighborhood Functions

Any partition of the vertices of S into k disjoint (not-empty) subsets gives a feasible solution s of k-MinBCP. Let $\delta(i)$ denote the neighbors of vertex i in the shore T. The cost of a solution s is:

$$w(s) = |(\{(i,j) \mid i \in s,\ j \in \cup_{i\in s}\delta(i), (i,j) \notin E\}|. \tag{36}$$

We are interested in neighborhood functions that map a solution s into a new solution s' of (likely) lower cost. The proposed metaheuristic algorithm is based on the following three neighborhood functions:

Neighborhood $N_1(s)$: select a node in a cluster of size at least two and move it in a different cluster.

Neighborhood $N_2(s)$: swap a pair of nodes belonging to different clusters (of size at least two).

Neighborhood $N_3(s)$: select a pair of nodes and move both of them in a new cluster.

The smallest neighborhood is $N_1(s)$ that has a size $O(k|S|)$. The neighborhood $N_2(s)$ has a size $O(k^2|S|^2)$, but it does not share any solution with $N_1(s)$. The neighborhood $N_3(s)$ has size $O(k^4|S|^2)$ and contains the other two: moving two nodes allows to reach any solution in $N_1(s)$ an $N_2(s)$. However, in order to explore disjoint neighborhoods, we define $N_3(s)$ so that it does not contain any solution in $N_1(s)$ nor $N_2(s)$. It is sufficient to select two distinct nodes and moving them to two clusters, both different from the original ones.

3.2. Variable Neighboorhood Infeasible Search

In addition to the three neighborhood functions $N_1(s)$, $N_2(s)$, and $N_3(s)$, we consider a fourth neighborhood that allows to explore for few iterations the space of infeasible solutions. The idea is simply to increase by one the number of possible clusters, to perform a few iterations in this new solution space with the other two neighborhoods, and then to restore a feasible solution by merging two clusters. The motivation for performing such infeasible search is, as usual, to try to escape from local minima. In order to make this idea effective the merging of two clusters has to be done carefully. Basically, once in the space of $k+1$ cluster the VNS algorithm has reached a local minimum, we perform an exhaustive search by evaluating the cost of merging any possible pair of the $k+1$ clusters, and by selecting the pair that yields the lowest cost.

3.3. VNS for the pricing subproblem

The VNS algorithm that finds upper bound to k-MinBCP is also used in a simplified version to heuristically solve the pricing subproblem (32)–(35). In the pricing subproblem, we are interested in finding a single bipartite subgraph that minimize (32). We can consider this problem as k-MinBCP with $k = 2$, and we use (32) to evaluate one of the two clusters. In this way, we can still use the neighborhood functions $N_1(s)$ and $N_2(s)$, and perform a standard VNS. Since we are mainly interested in negative reduced cost solutions, whenever after a single descent we already have found such a solution, we stop the algorithm.

In addition, column generation algorithms converges faster when the generated columns (i.e., solutions of the pricing subproblems) are diverse among each other. For this reason, the initial solution is randomly drawn from the subset of vertices that were not belonging to the previous solution call of the algorithm.

4. A Branch and Price algorithm for k-MinBCP

The column generation formulation (28)–(31) and the new metaheuristic presented in Section 3 are integrated into a Branch and Price algorithm for k-MinBCP. Note that the column generation formulation was introduced in (Faure et al., 2007), but, there, it was only used to compute lower bounds at the root node and to get heuristic solutions, and it was not used to develop an exact algorithm.

Our Branch and Price algorithm is based mainly on three features: the metaheuristic that finds upper bounds, the algorithms used to solve the pricing subproblem, and the branching rule. We describe next each of these features.

The VNIS metaheuristic is used to compute upper bounds and to gather a fist pool of columns that defines the initial restricted master problem. That is, we store also intermediate solutions with cost higher than the lowest upper bound. This is repeated at each node of the branch-and-bound tree, even if the VNIS heuristic is executed at the root node with a higher limit on the number of iterations.

The second feature is in the solution of the pricing subproblem (32)–(35). The solution of the pricing subproblem is currently the true bottleneck of our exact approach to k-MinBCP. We have implemented two methods for solving the pricing problem. The first method is the VNS heuristic described in Section 3.3: we basically look for a single subgraph with negative reduced cost. Whenever the heuristic is unable to find a negative reduced cost solution, we use an ILP solver to solve the pricing subproblem. The formulation (32)–(35) is extended with the constraint on the objective function value that must be negative. Moreover, since we are interested in any solution of negative reduced cost, we stop the ILP solver once it has found such a solution. Figure 3 shows the main procedures of our implementation of the column generation.

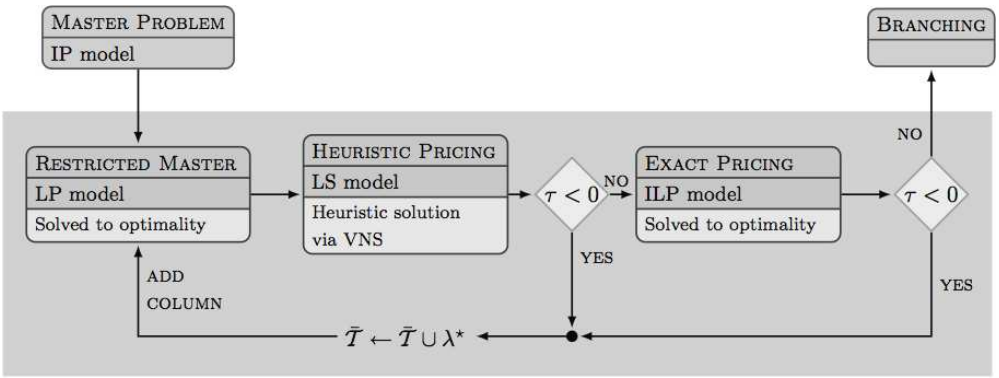


Figure 3: Scheme of the Column Generation algorithm.

Several instances are solved at the root node, since the upper bounds obtained with our VNIS heuristic are equal to the lower bounds obtained by Column Generation. However, since this is not always the case, we have devised a branching rule that exploits the problem structure. Once a pair of vertices i and j of S appearing in a fractional solution of the restricted master problem are selected, the algorithm adds two branching constraints: either i and j must appear in the same cluster, or they do not. In the first case, we merge the two nodes in a single new node, obtaining a new

instance of the same problem, as shown in Figure 4. In this case we have to take into account the cost of merging, and to sum it up in the objective function. Apart from that, we can directly reuse the same algorithms used at the root node to compute lower and upper bounds.

In the second case, we force the two selected vertices i and j to appear in different clusters. This branch deserves more attention in the VNIS algorithm and in the pricing subproblem. The branching decision corresponds to add a constraint $x_i + x_j \leq 1$ in the pricing subproblem (32)–(35). In the exact solution of the pricing subproblem this is directly implemented by adding such a constraint. However, the VNIS algorithm for k-MinBCP and the VNS algorithm for the pricing subproblem need a small modification: they have to remove from the neighborhood every move that is in contrast with any branching decision. Therefore, the corresponding branching constraints are stored in an appropriate data structure (a hash table) that permits look up in constant time. Before evaluating a move of any neighborhood function, the algorithm checks if two vertices can be placed in the same cluster according to the branching constraints.

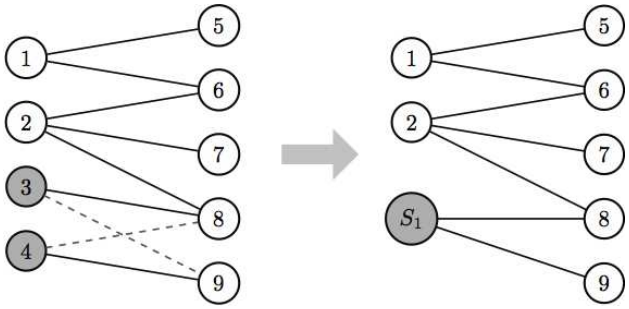


Figure 4: Left Branching rule: merge of two vertices of the shore S into a single new vertex. Vertices 3 and 4 are merged in the new node S_1. The new instance of k-MinBCP has at least cost 2 due to the complementary edges $(3, 9)$ and $(4, 8)$.

5. Computational Results

The computational results are divided into three parts: the first shows the strength of VNIS, the second part compare the strength of the lower bounds of the models presented in Section 2, and the last part presents the results for our Branch and Price algorithm. We used two classes of instances: the first set of instances consists of random bipartite graphs, the second of instances generated using the MovieLens data set. We selected randomly generated bipartite graphs with $|S| = |T|$, because, in (Faure et al., 2007), they were shown to be the most challenging. We remark that previous work solved to optimality this type of random graphs only with size up to $|S| = |T| = 12$ (Faure et al., 2007; Gualandi, 2009).

The MovieLens data set was collected by the GroupLens Research Project[2] at the University of Minnesota through the MovieLens web site[3] during the seven-month period from September 19th, 1997 through April 22nd, 1998. The data set contains 100,000 ratings (from 1 to 5) on movies provided by 943 users on 1682 movies. Each user has rated at least 20 movies and simple demographic information for the users are reported. The data set can be translated easily to our problem: the users are the clients, while the movies are the sessions. A rating can be seen as a preference expressed by the client/user about the session/movie. In terms of k-MinBCP, users form the left shore S, the items the right shore T, and the ratings are the edges. The collection of data and the resulting graphs are large. However, since the ratings are evenly spread, it has been possible to extract meaningful subgraphs to be used in our tests.

The Branch and Price algorithm is implemented in Comet[4], using the Comet interface to `lp_solve` for solving the restricted master problem. The lower bounds of the linear models are computed using AMPL+CPLEX 11.0. The SDP relaxation is solved with the DSDP5.8 solver (Benson and Ye, 2008). A random problem generator along with a set

[2] `http://www.grouplens.org/`, last visited September 2011.
[3] `http://movielens.umn.edu/`, last visited September 2011.
[4] Comet web site, `http://comet-online.org`, last visit September 2011.

$\|S\|$	$\|T\|$	k	time	it	n. opt	gap
15	15	3	0.5	265	**12**	**0**
		4	0.7	365	**12**	**0**
		5	0.8	372	**12**	**0**
18	18	3	1.5	390	**12**	**0**
		4	1.9	526	**12**	**0**
		5	2.0	531	**12**	**0**
20	20	3	2.5	504	*11*	*0.01*
		4	3.3	670	**12**	**0**
		5	3.5	678	**12**	**0**

Table 1: Number of times (n. opt) the VNIS algorithm finds the global optimum.

$\|S\|$	$\|T\|$	k	time	it	UB	opt
15	15	2	0.02	35	73	73
		3	0.01	14	49	49
		4	0.01	12	32	32
		5	0.17	148	22	22
18	18	2	0.01	8	101	101
		3	0.06	40	54	54
		4	0.09	74	35	35
		5	0.01	17	23	23
20	20	2	0.01	13	124	124
		3	0.09	29	77	77
		4	0.01	20	47	47
		5	0.11	40	34	34

Table 2: Performance of the VNIS algorithm on small graphs extracted from the MovieLens data set.

of challenging instances is made available at `http://www-dimat.unipv.it/~gualandi/Resources.html` (last visited, September 2011). The experiments were run on an standard desktop computer with an Intel(R) Xeon(TM) 2.80GHz CPU and 2GB of memory.

5.1. VNIS

The VNIS metaheuristic is implemented in COMET using its high-level programming abstractions to implement local search and metaheuristic algorithms (Van Hentenryck and Michel, 2005; Van Hentenryck et al., 2005).

Table 1 shows the results obtained with our VNIS algorithm on a set of randomly generated instances. The VNIS algorithm is stopped whenever it reaches a local minimum. Each row reports the averages over 12 instances with the same size, for a total of 108 different instances. The first three columns of the table report the characteristics of each instance. The fourth column gives the computation time in seconds and the number of iterations that the algorithm requires to reach a local optimum. The sixth column reports the number of time the VNIS algorithm does find the optimum, and the last column reports the gap with the lower bound computed with the column generation algorithm. Note that over 108 instances only once our algorithm did not find the optimum, that is for an instance with $|S| = |T| = 20$ and $k = 3$. However, for this instance, the percentage gap with the lower bound is of 1%.

Table 2 and 3 show the results on instances extracted from the MovieLens data set. Table 2 refers to instances with the same dimensions of those used in Table 1 and for which we were able to compute the optimum solution via Branch and Price. For those instances, the VNIS algorithm finds always the optimum in less than 1 second, taking only few iterations. Table 3 reports the results of instances defined on bigger graphs, whose optimum solution is unknown. We report only the best results obtained with VNIS within a timeout of two hours. For each instance, the time of the last improvement is given in the table, that is the time at which the best solution was found.

Figure 5 plots the normalized execution time with the longest time (at which is assigned as reference the value 10 reported on the vertical axis) of the VNIS algorithm on instances with different characteristics. The plot on the left shows that indeed by increasing the number of clusters the problem becomes harder. The plot on the left shows

$\|S\|$	$\|T\|$	d	k	time	UB	k	time	UB
50	50	0.3	5	107	1321	10	137	938
		0.5	5	11	1072	10	50	876
		0.7	5	16	672	10	43	577
80	80	0.3	5	872	3819	10	780	3202
		0.5	5	167	2862	10	400	2571
		0.7	5	46	1769	10	144	1618
100	100	0.3	5	452	6248	10	893	4298
		0.5	5	198	4658	10	5427	5428
		0.7	5	87	2842	10	441	2644

Table 3: Performance of the VNIS algorithm on big graphs extracted from the MovieLens data set.

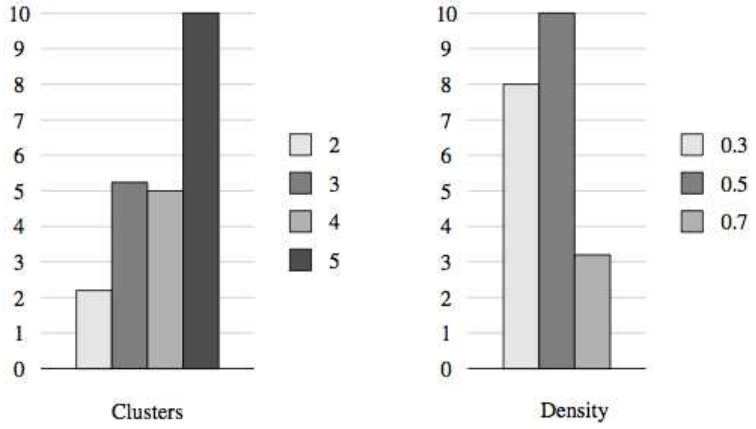


Figure 5: Impact of the number of clusters (left) and the density (right) on the performance of VNIS.

instead that the random instances with density around 50% are most challenging, and the sparse graphs are harder than the dense graphs.

5.2. Lower bounds

Table 4 shows the lower bounds obtained with the relaxations of the ILP models presented in Section 2. The results are obtained on random bipartite graphs with density ranging in the set $\{0.3, 0.5, 0.7\}$ and with the number of clusters between 2 and 5. The ILP model (5)–(10) gives the weakest lower bounds. The asymmetric representative relaxation (11)–(17) and the SDP relaxation are complementary: the first gives better lower bounds on dense bipartite graphs, while the latter on sparse bipartite graphs. However, the (A-ILP) formulation gives better bounds than the (SDP) relaxation when the number of clusters increases, e.g., $k = 5$. The column generation (CG) formulation outperforms the other models and therefore was further investigated.

Table 5 shows the results obtained with our column generation algorithm over a selection of random instances. For each instance, we first report the optimum value (in column "opt"), the upper bound and the computation time (in seconds) obtained with VNIS. Then, we report the lower bound, and the overall computation time and number of iterations of the column generation algorithm. The last six columns distinguish among the computation time, the number of iteration (it), and the average time per iteration (avg), of the heuristic pricing solver and of the exact pricing solver. The lower bounds are tight and in most iteration equal the optimum values. The bottleneck of our approach is clearly in the exact solution of the pricing subproblem required in the very last iterations of the column generation algorithm. Note that the exact pricing is executed at most 10 times, but it takes most of the computation time (see instance with $|S| = |T| = 20$, $k = 3$, and $d = 0.7$, where each iteration requires in average 4.4 seconds, instead of the 1.03 seconds of the heuristic pricer).

				Lower Bounds			
$\|S\|$	$\|T\|$	d	k	(ILP)	(A-ILP)	(SDP)	(CG)
20	20	0.3	2	68	73	137	217
			3	0	59	86	168
			4	0	53	60	129
			5	0	49	45	105
		0.5	2	26	80	97	173
			3	0	58	61	150
			4	0	52	43	129
			5	0	48	32	112
		0.7	2	25	64	54	102
			3	0	54	34	92
			4	0	48	24	83
			5	0	44	18	75

Table 4: Comparing lower bounds obtained with different formulations.

							Column Generation								
					VNIS		Total			Heuristic Pricing			Exact Pricing		
$\|S\|$	$\|T\|$	k	d	opt	UB	time	LB	time	it	time	it	avg	time	it	avg
15	15	3	0.3	84	84	0.04	82	14	62	9.7	60	0.17	4.3	2	2.3
			0.5	72	72	0.03	72	15	64	10	60	0.18	5.0	4	1.3
			0.7	53	53	0.08	51	11	70	10	67	0.15	1.5	3	0.5
		4	0.3	66	66	0.04	64	11	49	7.2	47	0.15	4.0	2	2.0
			0.5	59	59	0.12	59	12	56	9.6	53	0.18	2.7	3	0.9
			0.7	46	46	0.08	45	10	68	9.0	66	0.13	1.1	2	0.5
		5	0.3	51	51	0.01	51	8.1	35	5.7	33	0.17	2.4	2	1.2
			0.5	50	50	0.01	49	7.9	35	6.0	32	0.19	1.9	3	0.6
			0.7	39	39	0.27	39	6.9	38	5.2	33	0.16	1.7	5	0.3
18	18	3	0.3	137	137	0.10	135	83	94	30	92	0.33	52	2	26.5
			0.5	109	109	0.25	109	73	104	55	100	0.55	17	4	4.4
			0.7	79	79	0.10	79	47	111	39	105	0.37	8.6	6	1.4
		4	0.3	112	112	0.12	110	38	71	26	70	0.38	11	1	11.5
			0.5	96	96	0.35	96	54	76	34	72	0.48	19	4	4.8
			0.7	71	71	0.56	71	32	92	24	87	0.29	7.2	5	1.4
		5	0.3	92	92	0.17	92	40	80	28	78	0.36	12	2	6.3
			0.5	86	86	0.02	84	58	88	40	85	0.48	18	3	6.0
			0.7	63	63	0.06	63	29	76	22	68	0.33	6.3	8	0.8
20	20	3	0.3	178	178	1.39	176	145	109	70	106	0.62	75	3	25.0
			0.5	156	156	0.03	154	146	147	117	145	0.82	28	2	14
			0.7	104	104	0.14	104	171	134	127	124	1.03	44	10	4.4
		4	0.3	145	145	0.09	142	83	88	55	87	0.64	27	1	27.5
			0.5	138	*139	1.23	137	140	134	114	132	0.86	26	2	13.1
			0.7	92	92	0.14	92	101	111	93	109	0.86	7.2	2	3.6
		5	0.3	119	*120	1.32	117	182	74	155	72	2.16	27	2	13.6
			0.5	123	123	1.09	121	118	92	92	89	1.04	25	3	8.6
			0.7	82	82	0.31	82	131	148	107	139	0.77	23	9	2.6

Table 5: Lower and upper bounds obtained via Column Generation.

Table 6: Results of the Branch and Price algorithm.

(a) Random instances.

$\|S\|$	$\|T\|$	k	LB_0	opt	nodes	time
15	15	3	72	72	0	6.4
		4	59	59	0	2.9
		5	50	50	0	4.1
18	18	3	109	109	0	26
		4	96	96	0	20
		5	85	86	8	47
20	20	3	154	156	12	275
		4	137	138	8	206
		5	121	123	38	175

(b) MovieLens instances.

$\|S\|$	$\|T\|$	k	LB_0	opt	nodes	time
15	15	3	49	50	4	38
		4	32	32	0	6.6
		5	22	23	2	12
18	18	3	54	57	2	61
		4	35	35	0	21
		5	23	25	2	49
20	20	3	77	77	0	41
		4	47	47	0	45
		5	34	35	2	91

5.3. Branch and Price

Tables 6.a and 6.b show the computational results obtained with our Branch and Price algorithm (described in Section 4) on random and MovieLens instances with the same size. The first three columns give the size of the two shores $|S|$ and $|T|$, and the number k of required clusters. Then, the table reports the lower bounds obtained via column generation at the root node (LB_0), the optimal solution (opt) obtained via Branch and Price, the number of branching nodes (nodes), and the computation time in seconds. Note that our Branch and Price algorithm is able to solve all the instances in less than five minutes, while the instances of dimension 20×20 were out of reach in previous work.

6. Conclusions

We have presented a Branch and Price algorithm to k-MinBCP that embeds a new metaheuristic called VNIS and a non-trivial branching rule. The metaheuristic is effective on large instances derived from the MovieLens data set. The Branch and Price algorithm outperforms previous approaches presented in the literature. The approximability of the problem remains open and constitutes an obvious direction for further research. The corresponding non-bipartite version of the problem, to the authors knowledge, has not yet been considered and constitutes another interesting research topic.

It could be worthwhile also to extend our approach to a weighted version of this problem.

References

N. Faure, P. Chrétienne, E. Gourdin, F. Sourd, Biclique completion problems for multicast network design, Disc. Optim. 4 (3) (2007) 360–377.

S. Gualandi, k-Clustering Minimum Biclique Completion via a Hybrid CP and SDP Approach, in: Proc. Integration of AI and OR Techniques in CP for Combin. Optim., vol. LNCS 5547, 87–101, 2009.

P. Hansen, N. Mladenovic, Variable neighborhood search: Principles and applications, European Journal of Operational Research 130 (3) (2001) 449–467.

N. Faure, Contribution à la résolution de problèmes de regroupement de sessions multicasts, Ph.D. thesis, Université Paris VI, (written in French), 2006.

M. Campelo, V. Campos, R. Correa, On the asymmetric representatives formulation for the vertex coloring problem, Disc. Appl. Math. 156 (7) (2008) 1097–1111.

A. Frieze, M. Jerrum, Improved approximation algorithms for max k-cut and max-bisection, Algorithmica 18 (1997) 67–81.

B. Ghaddar, M. Anjos, F. Liers, A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem, Annals of Operations Research (2009) 1–20(Available on-line).

M. Goemans, D. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, J. ACM 42 (1995) 1115–1145.

S. J. Benson, Y. Ye, Algorithm 875: DSDP5—software for semidefinite programming, ACM Trans. Math. Software 34 (3) (2008) 1–20.

P. Van Hentenryck, L. Michel, Control abstractions for local search, Constraints 10 (2) (2005) 137–157.

P. Van Hentenryck, L. Michel, L. Liu, Contraint-Based Combinators for Local Search, Constraints 10 (4) (2005) 363–384.