J Heuristics (2013) 19:549–564
DOI 10.1007/s10732-011-9176-0

Test-assignment: a quadratic coloring problem

Jelle Duives · Andrea Lodi · Enrico Malaguti

Received: 23 June 2010 / Accepted: 16 May 2011 / Published online: 1 June 2011

Abstract We consider the problem of assigning the test variants of a written exam to the desks of a classroom in such a way that desks that are close-by receive different variants. The problem is a generalization of the Vertex Coloring and we model it as a binary quadratic problem. Exact solution methods based on reformulating the problem in a convex way and applying a general-purpose solver are discussed as well as a Tabu Search algorithm. The methods are extensively evaluated through computational experiments on real-world instances.

The problem arises from a real need at the Faculty of Engineering of the University of Bologna where the solution method is now implemented.

Keywords Binary quadratic programming · Reformulations · Heuristics

1 Introduction

During a written exam, the teacher usually prepares some variants of a test to be solved by the students. It normally happens that some variants are quite similar while others are completely different. We consider the problem of assigning the test variants to the desks of a classroom in such a way that desks that are close-by receive different variants. The problem arises from a real need at the University of Bologna, where

J. Duives
University of Twente, Enschede, The Netherlands
e-mail: j.duives@student.utwente.nl

A. Lodi (✉) · E. Malaguti
DEIS, Università di Bologna, Bologna, Italy
e-mail: andrea.lodi@unibo.it

E. Malaguti
e-mail: enrico.malaguti@unibo.it

some classrooms are equipped with networked desktop computers, and the teacher can remotely assign the test variants to each desk.

More precisely, the *Test-Assignment* problem can be modeled as a Vertex Coloring problem on an undirected graph, where vertices correspond to desks and edges connect desks which are close-by. A positive weight, associated with each edge, represents the proximity of the vertices at the endpoints, the weight is increasing with the proximity, so as to model the "temptation" that a student may have in taking inspiration from its neighbor. Colors are associated with the exam variants: for every pair of colors, a positive weight defines the similarity of the corresponding variants. By defining the *vicinity* of an edge-color assignment as the edge weight times the similarity of the colors assigned to the endpoints, the problem asks to color a subset of the graph vertices, whose cardinality corresponds to the number of students attending the exam, in such a way that the *overall vicinity* is minimized.

Formally, we define an undirected graph $G = (V, E)$ with positive weights w associated with edges, and a set of available colors H. Let $|V| = n$ and $|E| = m$ be the number of nodes and edges, respectively. For each pair of colors (h, k) we have a positive weight γ_{hk} which defines the similarity of the two colors. If node i gets color h and node j gets color k the vicinity of the edge-color assignment is $w_{ij}\gamma_{hk}$. In general, the students will not completely fill the classroom, and there will be T empty desks, thus we have to color $n - T$ nodes of G.

The Test-Assignment problem is a generalization of the Vertex Coloring problem (see Malaguti and Toth 2010), which arises (in decision version) when all the edge weights are equal and there are k completely different exam variants (i.e., we can set $w_{ij}\gamma_{hk} = 1\ \forall (i, j) \in E, \forall h, k \in H$) and $T = 0$. A coloring with null vicinity corresponds in this case to a feasible *k-coloring* (i.e., a coloring using k colors). Thus, the problem is $NP - Hard$ in the strong sense. Moreover, the problem is a variant of the Quadratic Assignment problem (see Burkard et al. 2009), because the penalty (cost) in the objective function depends on the colors assigned to the two endpoints of every edge. In particular, similar problems are encountered in the domain of Frequency Assignment, and concern the allocation of frequencies to transmitters, with the aim of avoiding or minimizing interference (see the survey by Aardal et al. 2003 for an overview).

The paper is organized as follows. In Sect. 2 we propose the possible quadratic formulations of the problem. In Sect. 3 we discuss practical details on how to solve the quadratic problems, while in Sect. 4 a Tabu Search approach is proposed. Extensive computational results on real-world instances are presented in Sect. 5. Finally, some short conclusions are drawn in Sect. 6.

2 Test-assignment as a binary quadratic program

A natural quadratic model for the Test-Assignment problem uses binary variables x_{ih}, taking value 1 when node i gets color h and 0 otherwise, and reads as:

$$\min \sum_{(i,j)\in E} w_{ij} \sum_{h\in H} \sum_{k\in H} \gamma_{hk} x_{ih} x_{jk}, \tag{1}$$

$$\sum_{h \in H} x_{ih} \leq 1, \quad i \in V, \tag{2}$$

$$\sum_{i \in V} \sum_{h \in H} x_{ih} = n - T, \tag{3}$$

$$x_{ih} \in \{0, 1\}, \quad i \in V,\ h \in H, \tag{4}$$

where constraints 2 state that each vertex can receive at most one color, and constraint 3 states that $n - T$ vertices must be colored (i.e., $n - T$ students must be seated). An alternative to constraint 3 is to define a *dummy* color 0 which is the color given to the T uncolored nodes, where $\gamma_{0h} = 0\ \forall h \in H$. Constraints 2 and 3 are then rewritten as:

$$\sum_{h \in H} x_{ih} = 1, \quad i \in V, \tag{5}$$

$$\sum_{i \in V} x_{i0} = T, \tag{6}$$

where constraints 5 state that each vertex must receive one color, and constraint 6 states that T vertices (i.e., the empty desks) receive the dummy color 0.

2.1 Reformulating the binary quadratic program

A textbook linearization consists in replacing every product of variables $x_{ih}x_{jk}$ in the objective function with an additional binary variable (see, e.g., Aardal et al. 2003):

$$y_{ij}^{hk} = x_{ih}x_{jk} \quad (i, j) \in E,\ h, k \in H, \tag{7}$$

$$y_{ij}^{hk} \in \{0, 1\} \quad (i, j) \in E,\ h, k \in H. \tag{8}$$

The non-linear condition 7 can be imposed by the following (weak) linear constraints:

$$y_{ij}^{hk} \geq x_{ih} + x_{jk} - 1, \quad (i, j) \in E,\ h, k \in H, \tag{9}$$

while a stronger way to impose condition 7 is through the following constraints:

$$x_{ih} = \sum_{l \in H} y_{ij}^{hl} \quad i \in V,\ j \in \delta(i),\ h \in H \tag{10}$$

where $\delta(i)$ denotes the set of edges having node i as one endpoint.

Note that our objective function is symmetric, i.e., we only care about the colors that are assigned to a given edge. Thus we could associate y variables to colors assigned to edges (instead of vertices) and define $y_{ij}^{hk} = x_{ih}x_{jk} = x_{ik}x_{jh}$. In this way the overall number of y variables is divided by two, because we only have to define y_{ij}^{hk} variables with $i < j$ and $h < k$. The drawback of the y variables reduction is that we cannot express anymore the relation with the x variables by means of constraints 10. In any case, it is well-known that these reformulations produce models

with a weak continuous relaxation which are not suited for being solved through Branch-and-Bound.

Tackling directly model 1–4 with a generic *Mixed Integer Quadratic Problem* (MIQP) solver might be hard due to the fact that the objective function 1 is in general not convex. Let Q and c be the symmetric matrix and the vector encoding the quadratic and the linear term of the objective function of a generic MIQP, respectively. The objective function of a generic MIQP can be encoded as $z(x) = x^t Qx + c^t x$. In the Test-Assignment problem we have that $x^t Qx = \sum_{(i,j)\in E} w_{ij} \sum_{h\in H} \sum_{k\in H} \gamma_{hk} x_{ih} x_{jk}$, while the linear term is null. In addition, our variables are binary, i.e., we are considering a *Binary Quadratic Problem* (BQP).

A well-known trick that can be used to transform a BQP into an equivalent one with positive semi-definite (i.e., convex) objective function consists in subtracting the smallest eigenvalue λ of Q from all the entries of its main diagonal, and adding λ to all the entries of the linear term (see, e.g., Hammer and Rubin Hammer and Rubin 1970), thus obtaining an equivalent positive semi-definite objective function on the domain of the problem variables $z_\lambda(x) = x^t(Q - (\lambda e))x + (c + \lambda e)^t x$, where e is an n-vector of 1 entries. However, this method tends to produce weak lower bounds when the BQP is relaxed to a QP by dropping the integrality requirement on the variables.

Billionnet and Elloumi (2007) show how to reformulate an unconstrained non-convex BQP as an equivalent positive semi-definite BQP, in such a way that the bound obtained by relaxing the BQP to a QP is the best possible. Billionnet et al. (2009) extend the analysis to the constrained binary case (equality constraints), and propose what they call the *Quadratic Convex Reformulation* (QCR) method. Finally, Billionnet et al. (2010) extend the method to general *Mixed Integer Quadratic Problems* (MIQP).

More formally, let P be a (constrained) BQP, and let $z(C(P))$ be the value of its continuous relaxation. Without loss of generality, consider a minimization problem. We want to reformulate P as an equivalent positive semi-definite problem P' (with associated matrix Q') so as to obtain the best bound from $z(C(P'))$. This is extremely useful when the obtained formulation has to be solved through Branch-and-Bound, whose performance is deeply influenced by the quality of the computed (lower) bounds. Obtaining the best bound consists in solving the following maximization problem:

$$LB_{best} = \max_{Q'\succeq 0} z(C(P')). \tag{11}$$

We will discuss the details of the QCR method in Sect. 3.2. Roughly speaking, the method requires the solution of a specific *semi-definite problem* (SDP) relaxation of P. Billionnet et al. (2009) prove (see Theorem 1, Billionnet et al. 2009) that the bound produced by such an SDP relaxation equals LB_{best}, and that the dual variables associated with the constraints of the SDP can be used to compute the reformulation Q' of Q.

3 Solving the binary quadratic problem

In this section we will mention the way IBM-CPLEX v.12.1 (IBM-CPLEX, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, sometimes indicated as CPLEX for short) handles the original formulation, give a practical description of how to construct the SDP and finally discuss several noncommercial solvers used to handle the specific SDPs needed for the reformulation.

3.1 Solving the original formulation

We observed that the simple procedure outlined in the previous section to transform the objective function 1 in a convex one by using the smallest eigenvalue is indeed the one implemented into the IBM-CPLEX MIQP solver (at least since version 10.0). Thus, the solver is able to automatically apply it and solve the original formulation to optimality. However, the bound that is obtained when the BQP is relaxed to a QP is normally quite weak, producing a very long computing time for the subsequent Branch-and-Bound algorithm. Producing a tighter bound would of course make the problem much easier to solve in the subsequent phase and the details on how to do that are presented in the next section.

3.2 The SDP-reformulation

To construct the SDP used to reformulate matrix Q according to Billionnet et al. (2009) the problem that is being reformulated has to be a BQP with equality constraints. Our system, namely Eqs. 1, 5, 6 and 4 is indeed in this form, thus the technique (Billionnet et al. 2009) can be effectively applied.

Consider the following linearly-constrained binary quadratic program:

$$(P) : \min \left\{ g(x) = x^t Q x + c^t x : Ax = b, x \in \{0,1\}^p \right\} \tag{12}$$

where $c \in \mathbb{R}^p$, $b \in \mathbb{R}^o$, $Q \in \mathbb{R}^{p \times p}$ and symmetric and $A \in \mathbb{R}^{o \times p}$. Now consider an associated binary quadratic problem equivalent to P depending on two parameters $\alpha \in \mathbb{R}^{o \times p}$ and $u \in \mathbb{R}^p$

$$(P_{\alpha,u}) : \min \left\{ g_{\alpha,u}(x) : Ax = b, x \in \{0,1\}^p \right\}, \tag{13}$$

where

$$g_{\alpha,u} = g(x) + \sum_{k=1}^{o} \left(\sum_{i=1}^{p} \alpha_{ki} x_i \right) \left(\sum_{j=1}^{p} \alpha_{kj} x_j - b_k \right) + \sum_{i=1}^{p} u_i \left(x_i^2 - x_i \right). \tag{14}$$

Clearly, for any binary vector $\widetilde{x}$ satisfying the system $Ax = b$, then $g_{\alpha,u}(\widetilde{x}) = g(\widetilde{x})$. Billionnet et al. (2009) have shown that the "best" pair (α, u), i.e., that defining the associated problem $P_{\alpha,u}$ with the highest continuous relaxation value, can be computed by solving the following SDP:

$$\min c^t x + \sum_{i=1}^{p} \sum_{j=1}^{p} q_{ij} X_{ij},$$

$$-b_k x_i + \sum_{j=1}^{p} a_{kj} X_{kj} = 0, \quad k = 1, \ldots, o; i = 1, \ldots, p, \tag{15}$$

$$X_{ii} = x_i, \quad i = 1, \ldots, p, \tag{16}$$

$$Ax = b, \tag{17}$$

$$\begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0,$$

$$x \in \mathbb{R}^p, \quad X \in S_p,$$

where S_p represents the set of $p \times p$ real symmetric matrices. More precisely, the dual variables associated with constraints 15 and 16 are the values α and u, respectively, and the optimal solution value of the SDP equals LB_{best}.

3.3 On the choice of an SDP solver

Solving the above SDP requires attention. First, a commonly used format for expressing the SDP is the so-called "sparse SDP format" (see, e.g., SDPA, http://sdpa.sourceforge.net). This implies formulating the problem by only means of "greater or equal" constraints, thus each of the equalities 15, 16 and 17 above has to be translated into two equivalent "greater or equal" inequalities. Second, notice that X is symmetric and therefore the part under the main diagonal does not need to be defined. This greatly affects the number of SDP variables and therefore also the solving time of the SDP, independently of the used SDP solver.

Of course, however, the most important decision is the choice of the SDP solver among the noncommercial ones which are available. Because the efficiency of the solver depends on the type of problem, several noncommercial solvers have been tested. All the tested solvers accept the SDP instance in "sparse SDPA format". Beside this similarity there are some important differences among the solvers concerning the algorithmic approaches and the use. In the remainder of the section we briefly discuss the solvers we tested so as to justify our choice and provide useful information to the readers.

- CSDP v.6.1.0 by Borchers (COIN-OR-CSDP, https://projects.coin-or.org/Csdp/). The algorithm is a predictor-corrector version of the primal-dual barrier method of Helmberg et al. (1996) and uses the HKM search direction (see, Helmberg et al. 1996 for details). It can be used as a callable library for MATLAB but it can also be called from command line. Furthermore, CSDP has a parallel version for systems with multiple processors and shared memory.
- SDPLR v.1.03 by Burer, Monteiro, and Choi (SDPLR, http://dollar.biz.uiowa.edu/~burer/software/SDPLR/). This is different from the other solvers in the sense that it reformulates the SDP instance as a nonlinear program, and then solves it by the augmented Lagrangian method. It is a C package and a MATLAB interface is also provided.

- DSDP v.5.8 by Benson, Ye, and Zhang (DSDP, http://www.mcs.anl.gov/hs/software/DSDP/). It implements the dual-scaling algorithm for semi-definite programming. Unlike primal-dual interior point methods, this algorithm uses only the dual solution to generate a step direction. The software can be used as a set of subroutines, through MATLAB, or by reading and writing to data files. It also has a parallel version which is called PDSDP.
- SDPT3 v.4.0 by Toh, Todd, and Tütüncü (SDPT3, http://www.math.nus.edu.sg/~mattohkc/sdpt3.html). It is a MATLAB implementation which uses a Mehrotra-type predictor-corrector variant of interior-point methods (Mehrotra 1992) and two types of search directions, namely HKM and NT (see, Nesterov and Todd 1997).
- SeDuMi v.1.3 by Sturm, Romanko and Pólik (SeDuMi, http://sedumi.ie.lehigh.edu/). It is a MATLAB toolbox which implements the self-dual embedding technique and uses the NT search direction with a predictor-corrector scheme.
- SDPA v.7.3.1 by Fujisawa, Fukuda, Futakata, Kobayashi, Kojima, Nakata, Nakata, Yamashita (SDPA, http://sdpa.sourceforge.net). It is based on a Mehrotra-type predictor-corrector infeasible primal-dual interior-point method. It uses the HKM search direction and includes a callable library and a MATLAB interface. Besides the presence of a parallel version (SDPARA), versions 7.3.x of the software are specialized for multi-thread computing.

For the specific Test-Assignment problem SDPA happens to be the best solver. More precisely, for the Test-Assignment instances it has the smallest solving time among the noncommercial SDP solvers, its sequential version is also able to exploit multi-threading and finally a useful "SDPA Online Solver" is available for quick testing purposes.

4 A tabu search algorithm

It will be shown in Sect. 5 that the exact formulations described in the previous sections are good enough to solve medium-size instances of practical interest. However, in case of large-size instances one can resort to effective and fast heuristics to obtain feasible solutions of good quality.

We tackle the Test-Assignment problem with an algorithm based on the Tabu Search framework. Tabu Search is a Local Search technique that consists in exploring the neighborhood $N(s)$ of a given solution s, and choosing as new solution s' the solution in $N(s)$ which minimizes a given objective function $f(s)$. We say that the algorithm moves from s to s'. Tabu Search can move to a solution s' such that $f(s') \geq f(s)$, and thus the algorithm may cycle at the next iteration by choosing again s as best solution in $N(s')$. In order to avoid cycling, Tabu Search algorithms maintain a so-called *tabu list*, which is used to store some feature of the previously explored solutions. For a given number of iterations, called *tabu tenure*, a solution having a feature which is in the tabu list cannot be chosen as new current solution.

The Tabu Search algorithm we propose produces feasible solutions of the Test-Assignment problem, i.e., according to model 1, 5, 6 and 4, colorings of the nodes of $G = (V, E)$ such that T nodes receive color 0. In the following we will denote by $c(i)$ the color assigned to a given node $i \in V$ in a specified coloring. In particular,

begin
1. randomly generate a solution s with no uncolored nodes, all nodes are non-tabu;
2. $s^* = s$, $f(s^*) = +\infty$;
3. **for** *iter* $= 1$ to *iteration number*
4. order non-tabu nodes according to non-increasing score in s;
5. randomly select a node $i \in V$ among the first α non-tabu nodes;
6. i is tabu for the following *tenure* iterations;
7. assign i to the best color $h > 0$ and obtain s';
8. $s = s'$; $s'' = s'$;
9. **for** $j = 1$ to T
10. uncolor in s'' the node with largest score;
11. update nodes scores in s''
12. **end for**;
13. **if** $f(s'') < f(s^*)$ **then** $s^* = s''$
14. **end for**

end.

Fig. 1 Pseudo-code of the Tabu Search algorithm

our algorithm moves between solutions where no node receives color 0, while the solution value is computed through Eq. 1 by uncoloring a posteriori T nodes. Given a solution s and a node $i \in V$, we define $N(s)$ as the set of solutions which can be obtained from s by changing the color of node i. This neighborhood was originally proposed by Hertz and de Werra (1987) for the first Tabu Search algorithm for the classical Vertex Coloring problem.

Our algorithm, sketched in Fig. 1, starts by generating a random coloring of the graph (line 1). During the iteration cycle (lines 3-12), we keep the nodes ordered according to non-increasing values of their $score$ in the current solution s. The score of each node is defined as its contribution to the objective function $f(s)$ defined by Eq. 1, i.e.:

$$score(i) = \sum_{j \in \delta(i)} w_{ij} \gamma_{c(i)c(j)} \tag{18}$$

where $\delta(i)$ is the set of nodes j such that $(i, j) \in E$. A node i with current color $c(i)$ is randomly selected among the first α nodes and colored with the best color, i.e., with a color $c'(i)$ such that:

$$c'(i) = \arg \min_{h \in H, h \neq c(i)} \sum_{j \in \delta(i)} w_{ij} \gamma_{hc(j)}. \tag{19}$$

For the next *tenure* iterations node i is considered *tabu*, and thus cannot be selected for recoloring. The new solution where node i has color $c'(i)$ is denoted as s'. Note that all nodes in s' have a color larger than 0; in lines 9–11 we iteratively select the node j with the largest score, uncolor it (i.e., set $c(j) = 0$), and update the scores of nodes in $\delta(j)$, thus obtaining a solution s'' with T uncolored nodes. Finally, in line 13 we update the incumbent solution s^*, if needed.

5 Computational results

The reformulations and the heuristic approach described in the previous sections were computationally tested on a set of instances derived from 4 real classrooms in the Engineering Faculty of the University of Bologna. The classrooms have from 20 to 79 desks, and are usually used for written exams. From each classroom we derive the underlying graph G by associating a node with each desk and by defining edges between pairs of nodes (desks) that are close-by. The edge weights have values in $\{0.3, 0.5, 0.8, 1\}$ according to the distance and the relative location of the desks. Note that, since every desk has a similar number of neighbors (depending on the room geometry), small graphs are more dense. For every graph, we consider the first 2, 3 and 4 consecutive colors in the set $\{1, 2, 3, 4\}$, in addition to the dummy color (0) assigned to empty desks. With a slight abuse of notation, in this section we denote by H the largest color available. As previously specified, the dummy color does not affect the objective function, while the similarity of the colors larger than 0 is defined as $\gamma_{hk} = 1 - |h - k|/H$. Finally, for every graph/color combination, we consider 0, 5, 10, or 0, 10, 20 empty desks (this is the number T of vertices which must receive color 0), thus obtaining 36 instances.

Every BQP counts $n(H + 1)$ variables and $n + 1$ constraints. Recall that p is the number of variables and o is the number of constraints of the BQP. Then, every SDP counts $p + \frac{1}{2}(p^2 + p)$ variables, $2(p(o + 1) + o)$ constraints and a semi-definite constraint matrix of size $(p + 1) \times (p + 1)$. Note that with the number of variables and constraints for the SDP is meant the number of variables and constraints which are used in the "sparse SDPA format" SDP (see Sect. 3.3 for details). The sizes of the problems are summarized in Table 1, where for every instance we report in the first three columns the number of nodes n, the number of edges m and the density δ of graph G, in column four the number of available colors H and in column five the number of uncolored nodes T. Column six and seven report the number of variables and constraints of the SDPs, respectively, while column eight reports the size of the semi-definite constraint matrix of the SDPs. Similarly, in column nine and ten the number of variables and constraints is given for the BQP. All the problems, i.e., the original graphs, the SDPs, the original and reformulated BQPs, are publicly available at the DEIS-OR web page (DEIS-OR, http://www.or.deis.unibo.it/research.html).

The SDPs, whose size is extremely challenging, are solved with a computer equipped with 16 Intel Xeon X5570 processors at 2.93 GHz and 48 GB RAM under LINUX operating system, and the SDPA v.7.3.1 solver (SDPA, http://sdpa.sourceforge.net). The solver needs a set of parameters, which are explained in detail in the corresponding documentation. Although a standard parameter file is distributed with the software, for our experiments we used an adjusted version of this file, available at DEIS-OR, http://www.or.deis.unibo.it/research.html.

The SDPA solver performs successive iterations until the gap between the primal and dual solutions falls under a specified threshold. However, the solver can be stopped after a given number of iterations, and a primal and dual feasible solutions can then be retrieved before convergence. In our test we tried two configurations. The first requires solving the SDP up to convergence, while in the second we halt the SDPA solver after 8 iterations, so as to obtain a (weaker) lower bound

Table 1 Details on the instances

Properties					SDP			BQP	
n	m	δ	H	T	# Var	# Const	mat.size	# Var	# Const
20	53	0.28	2	{0, 5, 10}	1,890	2,682	3721	60	21
20	53	0.28	3	{0, 5, 10}	3,320	3,562	6561	80	21
20	53	0.28	4	{0, 5, 10}	5,150	4,442	10201	100	21
47	174	0.16	2	{0, 10, 20}	10,152	13,914	20164	141	48
47	174	0.16	3	{0, 10, 20}	17,954	18,520	35721	188	48
47	174	0.16	4	{0, 10, 20}	27,965	23,126	55696	235	48
60	187	0.11	2	{0, 10, 20}	16,470	22,442	32761	180	61
60	187	0.11	3	{0, 10, 20}	29,160	29,882	58081	240	61
60	187	0.11	4	{0, 10, 20}	45,450	37,322	90601	300	61
79	252	0.08	2	{0, 10, 20}	28,440	38,554	56644	237	80
79	252	0.08	3	{0, 10, 20}	50,402	51,352	100489	316	80
79	252	0.08	4	{0, 10, 20}	78,605	64,150	156816	395	80
122	424	0.06	2	{0, 10, 20}	67,527	90,280	134689	366	122
122	424	0.06	3	{0, 10, 20}	119,804	120,292	239121	488	122
122	424	0.06	4	{0, 10, 20}	186,965	150,304	373321	610	122

in a shorter computing time. We are actually interested in studying how much the value of the lower bound is weakened and how much this influences the performance of IBM-CPLEX v. 12.1 when solving the reformulated problem. Our experiments are summarized in Table 2, where we report the instance name in the first column. The instance name, whose format is n_H_T, summarizes the problem features: node number, available colors and uncolored nodes. In column two we report the value of the solution at the root node (*root*) obtained by IBM-CPLEX v. 12.1 by applying the textbook reformulation of the problem based on the eigenvalue method (Sect. 2.1); the corresponding computing time is always smaller than 0.25 seconds. In column three we report the optimal value of the SDP (*opt*), the computing time in seconds (t) in column four and the corresponding number of iterations ($it.s$) in column five. The remainder of the table considers the case in which the solver is stopped after 8 iterations. In column six we report the value of the dual objective function D_{obj}, which still represents a valid lower bound for the problem, and in column seven we report the computing time after 8 iterations have been performed. The largest SDP which can be solved counts 79 nodes and 2 colors, while for 3 and 4 colors the corresponding SDP exceeds the size which can be managed by the SDPA solver.

The effectiveness of the QCR with respect to the eigenvalue method is pointed out by the comparison of the initial bounds in the two cases (columns *root* and *opt*). The lower bound represented by the *opt* value is in all cases much better than the one in the *root* column, however, computing this bound can be very expensive, from the computational viewpoint, even for medium-size instances. In particular, we observe that while changing the number of uncolored nodes T has no effect on the computing time for solving the corresponding SDP, the addition of one color (i.e., increasing H) makes the problem much more difficult to solve. We also observe that the computing

Table 2 Computational results for the SDP reformulations

Instance	CPLEX	Opt SDP			Cutoff SDP (8 it.s)	
name	*root*	*opt*	*t*	*it.s*	D_{obj}	*t*
20_2_0	9.97	20.67	1.7	10	20.67	1.2
20_2_5	−0.80	7.19	2.5	15	7.18	1.3
20_2_10	−5.04	0.49	2.5	16	0.47	1.3
20_3_0	−4.20	14.18	8.2	15	14.16	4.1
20_3_5	−9.54	2.02	8.1	15	1.98	4.1
20_3_10	−10.30	−2.49	8.2	15	−2.53	4.1
20_4_0	−15.06	8.67	20.6	15	8.65	11.0
20_4_5	−16.81	−0.18	23.4	16	−0.21	11.1
20_4_10	−15.26	−3.54	22.1	16	−3.58	11.2
47_2_0	48.92	72.19	82.7	10	72.18	59.0
47_2_10	15.90	34.06	118.4	16	33.95	59.0
47_2_20	−1.87	11.30	102.5	13	11.18	59.4
47_3_0	12.34	52.18	713.5	17	52.13	320.6
47_3_10	−7.98	17.94	787.1	19	17.57	308.8
47_3_20	−16.89	1.06	762.3	19	0.63	314.6
47_4_0	−14.57	36.19	2,706.0	17	36.06	1,199.8
47_4_10	−26.51	10.41	2,093.9	13	10.04	1,192.7
47_4_20	−29.42	−2.86	2,093.6	13	−3.13	1,197.5
60_2_0	40.08	73.04	370.2	11	73.02	246.6
60_2_10	12.69	41.01	583.8	19	40.87	243.5
60_2_20	−3.96	18.64	572.0	18	18.52	247.3
60_3_0	−5.41	50.98	3,032.8	17	50.89	1,346.4
60_3_10	−20.74	19.78	2,694.3	15	19.26	1,349.8
60_3_20	−28.21	3.13	3,198.6	18	2.50	1,345.2
60_4_0	−39.96	32.54	10,566.6	18	32.40	4,442.8
60_4_10	−47.17	10.24	10,548.7	18	9.84	4,441.4
60_4_20	−48.50	−2.74	8,932.3	15	−3.11	4,438.8
79_2_0	58.82	107.85	1,737.6	10	107.83	1,282.0
79_2_10	24.53	68.25	2,057.3	12	67.75	1,271.4
79_2_20	2.72	39.58	2,208.4	13	38.79	1,263.4

time of the SDPA solver is approximately linear in the number of iterations, which ranges in the interval 11–19. By stopping the solver after 8 iterations, a good lower bound is quite always provided at lower computational cost.

The BQPs were solved on a standard PC equipped with a PIV at 3.4 GHz and 2 GB RAM under LINUX operating system. We are interested in studying how much the performance of the IBM-CPLEX v. 12.1 MIQP solver is influenced by the adopted reformulation and by the improved lower bound, and in comparing these results with the performance of the Tabu Search algorithm. In Table 3, after the problem name which provides the problem characteristics, we consider:

Table 3 Computational results for the quadratic coloring problem: reformulations and Tabu Search

Instance	CPLEX				CPLEX + reformulation				CPLEX + partial reformulation				Tabu Search		
name	*gap*	*UB*	*t*	*nodes*	*gap*	*UB*	*t*	*nodes*	*gap*	*UB*	*t*	*nodes*	*UB*	*t*	UB_{CPLEX}
20_2_0	0.00	20.90	5.7	33,526	0.00	20.90	0.1	145	0.00	20.90	0.1	110	20.90	3.2	20.90
20_2_5	0.00	7.95	213.6	1,078,274	0.00	7.95	0.9	1,650	0.00	7.95	0.5	939	7.95	5.1	8.15
20_2_10	0.00	1.85	241.8	1,354,793	0.00	1.85	12.0	30,517	0.00	1.85	10.7	29,613	1.85	5.1	1.85
20_3_0	9.95	15.15	3,600.0	15,515,986	0.00	15.15	3.7	8,677	0.00	15.15	3.7	8,723	15.35	4.0	15.15
20_3_5	86.43	5.59	3,600.0	13,825,199	0.00	5.58	1,414.4	2,697,684	0.00	5.58	1,682.3	3,288,762	5.59	5.3	5.65
20_3_10	195.95	1.22	3,600.0	16,601,626	6.20	1.22	3,600.0	8,064,766	17.67	1.22	3,600.0	8,014,543	1.22	5.4	1.25
20_4_0	75.16	11.95	3,600.0	13,210,430	0.00	11.95	1,349.7	2,746,950	0.00	11.95	1,120.5	2,298,906	12.10	4.6	12.13
20_4_5	236.41	3.98	3,600.0	11,542,131	30.87	3.98	3,600.0	5,176,331	30.44	3.98	3,600.0	5,460,687	4.05	5.3	4.15
20_4_10	593.55	0.93	3,600.0	14,414,118	174.14	0.93	3,600.0	6,475,618	166.90	0.93	3,600.0	6,933,296	0.93	5.2	1.05
47_2_0	12.46	72.60	3,600.0	10,499,825	0.00	72.60	2.7	1,940	0.00	72.60	1.7	1,653	73.70	30.1	72.60
47_2_10	33.63	35.60	3,600.0	9,223,882	0.00	35.45	750.1	446,400	0.00	35.45	1,359.0	805,542	35.60	38.2	35.60
47_2_20	68.87	12.75	3,600.0	10,137,663	0.00	12.65	503.5	276,793	0.00	12.65	873.4	499,768	12.80	44.1	13.30
47_3_0	51.76	54.20	3,600.0	6,297,112	0.67	53.72	3,600.0	1,891,002	0.65	53.72	3,600.0	1,909,715	55.45	29.1	54.17
47_3_10	101.43	25.11	3,600.0	4,875,050	17.73	24.84	3,600.0	1,079,444	19.96	25.07	3,600.0	1,094,840	25.29	38.0	25.76
47_3_20	222.14	8.55	3,600.0	5,321,731	62.34	8.59	3,600.0	1,254,663	64.68	8.52	3,600.0	1,146,090	8.68	45.3	8.59
47_4_0	102.86	44.00	3,600.0	4,169,741	13.51	43.93	3,600.0	1,129,116	13.42	43.88	3,600.1	1,155,307	44.60	30.4	43.93
47_4_10	193.93	19.28	3,600.0	3,765,036	37.59	19.23	3,600.0	653,529	37.83	19.05	3,600.0	745,651	19.20	38.0	20.15
47_4_20	445.03	6.40	3,600.0	3,669,344	122.17	6.38	3,600.0	764,858	119.78	6.58	3,600.0	826,140	6.40	45.7	6.90
60_2_0	25.57	74.05	3,600.0	9,131,708	0.00	74.05	102.5	59,268	0.00	74.05	141.7	103,852	76.30	51.2	74.25
60_2_10	50.30	43.00	3,600.0	7,571,364	1.64	43.00	3,600.0	1,321,462	1.89	43.00	3,600.0	1,478,046	44.70	62.0	43.20
60_2_20	88.56	20.35	3,600.0	6,135,847	0.81	20.35	3,600.0	1,023,605	1.45	20.35	3,600.0	1,161,395	22.20	72.3	20.95
60_3_0	84.83	54.83	3,600.0	5,101,020	3.45	54.03	3,600.0	1,162,763	3.65	54.03	3,600.0	1,156,750	55.140	52.4	54.99

Table 3 (*Continued*)

Instance	CPLEX				CPLEX + reformulation				CPLEX + partial reformulation				Tabu Search		
name	*gap*	*UB*	*t*	*nodes*	*gap*	*UB*	*t*	*nodes*	*gap*	*UB*	*t*	*nodes*	*UB*	*t*	UB_{CPLEX}
60_3_10	139.50	30.73	3,600.0	4,005,041	27.32	30.26	3,600.0	760,470	28.18	29.95	3,600.0	630,290	31.31	62.0	31.14
60_3_20	250.43	14.11	3,600.0	3,393,645	63.80	14.11	3,600.1	577,483	65.67	14.11	3,600.0	596,661	14.93	73.1	14.67
60_4_0	159.89	42.93	3,600.0	3,331,206	19.78	42.93	3,600.0	672,960	20.55	42.93	3,600.0	715,001	44.45	52.8	43.73
60_4_10	262.42	23.18	3,600.0	2,848,485	50.85	23.70	3,600.0	409,982	50.95	23.60	3,600.0	390,029	24.33	63.3	25.00
60_4_20	453.89	11.30	3,600.0	3,089,287	112.79	10.78	3,600.0	406,926	112.91	10.70	3,600.0	409,187	11.23	73.3	11.73
79_2_0	30.91	109.20	3,600.0	7,529,132	0.00	109.20	3,100.4	1,052,726	0.00	109.20	1,255.5	607,379	111.63	100.0	109.20
79_2_10	51.99	71.55	3,600.0	6,554,829	3.07	71.35	3,600.0	648,573	3.76	71.35	3,600.0	645,921	72.08	100.0	72.20
79_2_20	76.55	43.33	3,600.0	5,093,811	6.24	43.33	3,600.0	748,757	7.56	43.33	3,600.0	740,368	44.10	100.0	44.65
79_3_0	89.82	80.83	3,600.0	4,003,553	–	–	–	–	–	–	–	–	84.20	100.0	81.33
79_3_10	135.01	50.26	3,600.0	2,977,168	–	–	–	–	–	–	–	–	52.50	100.0	54.87
79_3_20	199.49	29.87	3,600.0	2,567,270	–	–	–	–	–	–	–	–	30.94	100.0	31.46
79_4_0	167.66	64.40	3,600.0	2,544,600	–	–	–	–	–	–	–	–	67.33	100.0	65.75
79_4_10	252.05	38.65	3,600.0	1,983,724	–	–	–	–	–	–	–	–	40.43	100.0	41.86
79_4_20	385.29	21.94	3,600.0	1,754,500	–	–	–	–	–	–	–	–	23.13	100.0	26.60

- The solution of the BQP "as is" with IBM-CPLEX v. 12.1. Since the objective function is not convex, IBM-CPLEX v. 12.1 modifies the Q matrix so as to obtain a positive semi-definite matrix Q', as explained in Sect. 2.1. In column two and three we report the final percentage optimality gap (*gap*, computed as $100\frac{UB-LB}{UB}$) and the value of the best feasible solution (UB) computed with a time limit of 1 hour. Column four reports the corresponding computing time in seconds (t), and column five the number of explored nodes (*nodes*).
- The solution of the BQP according to the QCR method (Billionnet et al. 2009), where the associated SDP is solved to optimality. In column six and seven we report the final optimality gap (*gap*) and the value of the best feasible solution (UB) computed within a time limit of 1 hour. Columns eight and nine report the computing time in seconds (t) and the number of nodes explored by IBM-CPLEX v. 12.1 (*nodes*), respectively.
- The solution of the BPQ according to the QCR method (Billionnet et al. 2009), where the associated SDPs are not solved to optimality, but the solver is stopped after 8 iterations. The same information as in the previous case is reported in columns ten to thirteen.
- Finally, in columns fourteen we report the value of the best solution (UB) found by the Tabu Search algorithm within a limit of $20000n$ iterations or 100 seconds. The overall computing time in seconds (t) is reported in column fifteen. Column sixteen reports the best upper bound produced by IBM-CPLEX v. 12.1 within the same computing time. Extensive computational tests were performed so as to define the best parameters' configuration for the set of instances of this study. We tested all possible combinations of parameters α and *tenure* with $\alpha \in \{0.2n, 0.25n, 0.3n, 0.35n, 0.4n\}$ and *tenure* $\in \{3, 5, 10, 15, 20, 25, 30\}$, for the subset of instances where 3 colors are available. For each combination we computed the percentage gap with respect to the best solution computed by IBM-CPLEX v. 12.1 in one hour (with and without reformulation), and we selected the best configuration, that is, $\alpha = 0.3$ and *tenure* $= 5$, which obtains a gap of 2.64% on the considered subset of instances. Note that the performance of the algorithm does not depend heavily on a change in the parameters settings, actually the worst tested configuration gives a gap of 4.17%.

The table shows the strong effect of the reformulation on the computational effort needed to solve the BQP with the Branch-and-Bound algorithm embedded in IBM-CPLEX v. 12.1. While IBM-CPLEX v. 12.1 stand alone can solve to proven optimality only 3 instances, the reformulation raises the optimally solved instances to 11, and this is the case also when the SDP is not solved to optimality. In addition, the optimality gap of the reformulation and the partial reformulation are almost always similar, which shows that the optimal solution of the SDP is not needed for the effectiveness of the method. Finally, the upper bound produced by the reformulations is 13 times lower and only 1 time larger than the one produced by IBM-CPLEX v. 12.1 stand alone. On the other hand, IBM-CPLEX v. 12.1 can produce a feasible solution even when the reformulation cannot be computed, namely for instances with $n = 79$ and $H > 2$.

Instead, if one is interested in producing a good quality solution in short computing time, the Tabu Search algorithm can produce excellent solutions within the time limit

Table 4 Computational results for the quadratic coloring problem: IBM-CPLEX v. 12.1 and Tabu Search on a large graph of 122 nodes

Instance	CPLEX				Tabu Search		
name	*gap*	*UB*	*t*	*nodes*	*UB*	*t*	UB_{CPLEX}
122_2_0	40.59	165.95	3,600.0	4,366,640	171.90	100.0	168.25
122_2_10	54.62	134.40	3,600.0	3,919,117	136.75	100.0	136.30
122_2_20	67.54	103.95	3,600.0	3,390,092	104.95	100.0	106.20
122_3_0	108.63	123.42	3,600.0	2,014,706	129.05	100.0	128.71
122_3_10	133.51	98.22	3,600.0	1,315,174	100.69	100.0	103.48
122_3_20	164.69	73.09	3,600.0	1,175,857	76.20	100.0	79.78
122_4_0	194.58	100.00	3,600.0	892,250	103.90	100.0	104.73
122_4_10	237.83	77.40	3,600.0	533,352	80.18	100.0	82.45
122_4_20	294.38	57.80	3,600.0	369,401	59.45	100.0	65.43

of 100 seconds. When compared with IBM-CPLEX v. 12.1 within the same time limit (without reformulation), the value of the *UB* produced by the Tabu Search is better in 18 cases, it is worse in 15 cases and it is the same in the remaining 3 cases.

In Table 4 we report the results obtained by IBM-CPLEX v. 12.1 and the Tabu Search algorithm on the large graph of 122 nodes, for which the SDP becomes intractable. Column headers have the same meaning as in the previous table. For this graph, IBM-CPLEX v. 12.1 always reaches the time limit and has a very large optimality gap after one hour of computation. For this graph, the upper bound produced by the Tabu Search algorithm in 100 seconds is better than the one produced by IBM-CPLEX v. 12.1 within the same time in 6 cases, and it is worse in the remaining 3.

6 Conclusions

We have considered a Vertex Coloring generalization arising in the context of assigning the test variants of a written exam to the desks of a classroom so as to minimize the improper collaboration among the students. We have considered some binary quadratic formulations of the problem going from a trivial one to more complex reformulations in which the quadratic objective function is convex. Moreover, we have proposed a simple and fast Tabu Search algorithm.

All methods have been tested on real-world instances and the strong reformulation proved to be useful within a general-purpose mixed-integer quadratic solver to address medium-size instances of practical interest for the application at the Faculty of Engineering of the University of Bologna. The Tabu Search algorithm is shown to produce good approximate solutions in short computing times for large-size instances.

Future work can be devoted to integrate the Tabu Search algorithm within the Branch-and-Bound scheme of the general-purpose solver, and to the design of very fast greedy heuristics that can be obtained, e.g., by adapting those for the Vertex Coloring Problem (Malaguti and Toth 2010). Concerning greedy heuristics, preliminary

attempts have shown that it is not trivial to obtain solutions with quality comparable to those of the Tabu Search but effectively integrating those algorithms within both the Branch-and-Bound scheme and the Tabu Search might be highly beneficial.

Acknowledgements The present work has been conducted when the first author was visiting the University of Bologna under the Erasmus project agreement. We warmly thank our colleagues at the University of Bologna Luca Ghedini and Alessio Ferretti for asking us the questions that led to this study. Many thanks go to Angelika Wiegele, Christoph Buchheim, Marc Uetz, Katsuki Fujisawa, Brian Borchers, Sam Burer and Michael Sting for interesting discussions and help with the SDP solvers. A final thank to our students Fabrizio Buccella and Sonia Cabassi for helping in the setup of the laboratory instances, and to two anonymous referees for useful comments.

References

Aardal, K.I., van Hoesel, S.P.M., Koster, A.M.C.A., Mannino, C., Sassano, A.: Models and solution techniques for the frequency assignment problem. 4OR **1**, 261–317 (2003)

Billionnet, A., Elloumi, S.: Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. Math. Program. **109**, 55–68 (2007)

Billionnet, A., Elloumi, S., Plateau, M.-C.: Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: the QCR method. Discrete Appl. Math. **157**, 1185–1197 (2009)

Billionnet, A., Elloumi, S., Lambert, A.: Extending the QCR method to general mixed-integer programs. Math. Program. (2010). doi:10.1007/BF01586044

Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. SIAM, Philadelphia (2009)

Hammer, P.L., Rubin, A.A.: Some remarks on quadratic programming with 0-1 variables. Rev. Fr. Inform. Rech. Oper. **4**, 69–74 (1970)

Helmberg, C., Rendl, F., Vanderbei, R.J., Wolkowicz, H.: An interior-point method for semidefinite programming. SIAM J. Optim. **6**, 342–361 (1996)

Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**, 345–351 (1987)

Malaguti, E., Toth, P.: A survey on vertex coloring problems. Int. Trans. Oper. Res. **17**, 1–34 (2010)

Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM J. Optim. **2**, 575–601 (1992)

Nesterov, Y.E., Todd, M.J.: Self-scaled barriers and interior-point methods for convex programming. Math. Oper. Res. **22**, 1–42 (1997)