Dealing with inequality constraints in large scale semidefinite relaxations for graph coloring and maximum clique problems

Federico Battista Marianna De Santis*

June 23, 2021

Abstract

Semidefinite programs (SDPs) can be solved in polynomial time by interior point methods. However, when the dimension of the problem gets large, interior point methods become impractical both in terms of computational time and memory requirements. First order methods, such as Alternating Direction Methods of Multipliers (ADMMs), turned out to be suitable algorithms to deal with large scale SDPs and gained growing attention during the past decade. In this paper, we focus on an ADMM designed for SDPs in standard form and extend it to deal with inequalities when solving SDPs in general form. This allows to handle SDP relaxations of classical combinatorial problems such as the graph coloring problem and the maximum clique problem, that we consider in our extensive numerical experience. The numerical results show the comparison of the method proposed equipped with a post-processing procedure with the state-of-the-art solver `SDPNAL+` [1].

Keywords: Semidefinite programming, Graph coloring problem, Maximum clique problem
MSC: 90C22, 90C27, 90C06

1 Introduction

Interest on semidefinite programming has considerably grown during the last two decades and this is partly due to the fact that many practical problems in operations research and combinatorial optimization can be modeled or approximated by semidefinite programs [2]. It is the purpose of the present paper to focus on the use of augmented Lagrangian methods for dealing with semidefinite programming relaxations of two well-known combinatorial problems: the graph coloring problem and the maximum clique problem. Augmented Lagrangian methods are known to be an alternative to interior point methods and currently represent the most popular first-order algorithms used to handle large scale semidefinite programs [3, 4, 5, 6]. Alternating direction methods of multipliers (ADMMs), which are a variant of augmented Lagrangian methods, gained growing attention in the last years [7, 1, 8, 9] and their success comes from the fact that as first-order methods avoid computing, storing and factorizing large Hessian matrices and are able to scale to much larger problems than interior point methods. This of course at some cost in accuracy, that should be avoided in case the semidefinite programming problem addressed is the relaxation of a combinatorial

*Dipartimento di Ingegneria Informatica Automatica e Gestionale, Sapienza Universit di Roma, Via Ariosto, 25 00185, Roma, Italy. E-mail: `battista@diag.uniroma1.it`, `mdesantis@diag.uniroma1.it`

problem and one aims at obtaining a valid bound on its optimal solution. In order to overcome this issue, post-processing procedures for ADMMs have been recently developed (see e.g. [10, 11, 12]). In this paper, we compare the performance of an ADMM equipped with a post-processing procedure with SDPNAL+ [1], the state-of-the art solver for large scale SDPs. SDPNAL+ combines an inexact symmetric Gauss-Seidel based semi-proximal ADMM with a semismooth Newton-Conjugate Gradient method and has been awarded with the Beale-Orchard-Hays Prize in 2018. Our study is motivated by the need of solvers for large scale SDPs, able to deal with inequality constraints, in particular when addressing SDP relaxations of combinatorial problems. This need is further demonstrated by similar studies in this respect, where SDP relaxations of the graph partitioning problem are solved [12]. We present numerical experiments on randomly generated instances and on instances from the SDP relaxations of the maximum clique problem and the graph coloring problem. Our aim is that of showing that even an inaccurate dual solution, obtained at a generic iteration of our ADMM, can represent a good and fast recovered bound on the optimal solution of the combinatorial problems considered.

1.1 Notation and outline

Let $\mathcal{S}_n$ be the set of n-by-n symmetric matrices, $\mathcal{S}_n^+ \subset \mathcal{S}_n$ be the set of positive semidefinite matrices and $\mathcal{S}_n^- \subset \mathcal{S}_n$ be the set of negative semidefinite matrices. In the following, we denote by $\langle X, Y\rangle = \text{trace}(XY)$ the standard inner product in $\mathcal{S}_n$. Whenever a norm is used, we consider the Frobenius norm in case of matrices and the Euclidean norm in case of vectors. Let $v \in \mathbb{R}^n$, we denote by $\text{Diag}(v)$ the diagonal matrix having v on the main diagonal. The vector e_i is defined as the i-th vector of the standard basis in $\mathbb{R}^n$. Let $S \in \mathcal{S}_n$. We denote the projection of S onto the positive semidefinite and negative semidefinite cone by $(S)_+$ and $(S)_-$, respectively. Moreover we denote by $\lambda(S)$ the vector of the eigenvalues of S and by $\lambda_{\min}(S)$ and $\lambda_{\max}(S)$ the smallest and largest eigenvalue of S, respectively. With $\mathbf{0}_n$ we denote the all-zeros column vector of size n.

The paper is organized as follows. In Section 2, we present the ADMM algorithm `ADAL` [5, 6, 8] for solving SDPs in standard form. We use this method to handle problems in general form and we detail how to deal with inequalities, avoiding the storage of the full matrices. We also discuss how to recover a valid dual bound on the optimal primal value, starting from an approximated solution. Some SDP relaxations of the maximum clique problem and of the graph coloring problem are reported in section 3. Our numerical experience is presented in section 4 and some conclusions are drawn in section 5.

2 An ADMM method for SDPs in general form

We focus on SDPs in general form:

$$\begin{array}{lll} \min & \langle C, X\rangle & \\ \text{s.t.} & \langle A^i, X\rangle \le b_i, & \forall i = 1, \dots, l \\ & \langle A^j, X\rangle = b_j, & \forall j = l+1, \dots, m \\ & X \in \mathcal{S}_n^+ & \end{array} \tag{1}$$

where $C \in \mathcal{S}_n$, $A^i \in \mathcal{S}_n$, $i = 1, \dots, m+l$ and $b \in \mathbb{R}^{m+l}$. In order to deal with problem (1) a standard way is to add slack variables $s_i \ge 0$, $i = 1, \dots, l$ and expand the matrix variable X

to $\bar{X} \in \mathcal{S}_{n+l}$:

$$\bar{X} := \begin{pmatrix} X & \mathbf{0}_{n,l} \\ \mathbf{0}_{l,n} & \mathrm{Diag}(s) \end{pmatrix}.$$

Recall that if B is a diagonal matrix, the constraint $B \succeq 0$ boils down to $B \geq 0$. In particular, imposing $\bar{X} \succeq 0$ is equivalent to consider $X \succeq 0$ and $s \geq 0$. By expanding the matrices $A^i,\ A^j$, and C, $i = 1, \ldots, l;\ j = l+1, \ldots, m$; to $\bar{A}^i$, $\bar{A}^j$ and $\bar{C}$ as

$$\bar{A}^i := \begin{pmatrix} A^i & \mathbf{0}_{n,l} \\ \mathbf{0}_{l,n} & e_i^T e_i \end{pmatrix}, \qquad \bar{A}^j := \begin{pmatrix} A^j & \mathbf{0}_{n,l} \\ \mathbf{0}_{l,n} & \mathbf{0}_{l,l} \end{pmatrix}, \qquad \bar{C} := \begin{pmatrix} C & \mathbf{0}_{n,l} \\ \mathbf{0}_{l,n} & \mathbf{0}_{l,l} \end{pmatrix}$$

problem (1) can be rewritten as an SDP in standard form as follows:

$$\begin{array}{ll} \min & \left\langle \bar{C}, \bar{X} \right\rangle \\ \text{s.t.} & \bar{\mathcal{A}} X = b \\ & \bar{X} \in \mathcal{S}^+_{n+l} \end{array} \tag{2}$$

where $b := (b_1, \ldots, b_m) \in \mathbb{R}^m$ and $\bar{\mathcal{A}} : \mathcal{S}_{n+l} \to \mathbb{R}^m$ is the linear operator $(\bar{\mathcal{A}} X)_i = \left\langle \bar{A}^i, X \right\rangle$ with $\bar{A}^i \in \mathcal{S}_{n+l}$, $i = 1, \ldots, m$. The dual problem of (2) is defined as

$$\begin{array}{ll} \min & b^T y \\ \text{s.t.} & \bar{\mathcal{A}}^\top y + \bar{Z} = \bar{C} \\ & \bar{Z} \in \mathcal{S}^+_{n+l}, \end{array} \tag{3}$$

where $\bar{\mathcal{A}}^\top : \mathbb{R}^m \to \mathcal{S}_{n+l}$ is the adjoint operator of $\bar{\mathcal{A}}$, namely $\bar{\mathcal{A}}^\top y = \sum_i y_i \bar{A}^i$ for $y \in \mathbb{R}^m$. Note that the matrix $\bar{Z} \in \mathcal{S}_{n+l}$ is a "surplus" matrix variable that can be written as

$$\bar{Z} := \begin{pmatrix} Z & \mathbf{0}_{n,l} \\ \mathbf{0}_{n,l} & \mathrm{diag}(p) \end{pmatrix},$$

with $p \in \mathbb{R}^l$. In particular, the equality constraint in (3) can be rewritten as

$$\bar{C} - \bar{\mathcal{A}}^T(y) - \bar{Z} = \begin{pmatrix} C - \mathcal{A}^T y - Z & & \mathbf{0}_{n,l} & \\ & -y_1 - p_1 & & \\ \mathbf{0}_{n,l} & & \ddots & \\ & & & -y_l - p_l \end{pmatrix} = 0.$$

Assuming that both the primal (2) and the dual (3) problems have strictly feasible points (i.e. Slater's condition is satisfied) strong duality holds and $(y, \bar{Z}, \bar{X})$ is optimal for (2) and (3) if and only if the following conditions hold:

$$\bar{\mathcal{A}} \bar{X} = b, \qquad \bar{\mathcal{A}}^\top y + \bar{Z} = \bar{C}, \qquad \bar{Z}\bar{X} = 0, \qquad \bar{X} \in \mathcal{S}^+_{n+l}, \qquad \bar{Z} \in \mathcal{S}^+_{n+l}. \tag{4}$$

In the following, we assume that the constraints formed through the operator $\bar{\mathcal{A}}$ are linearly independent.

2.1 ADAL: an ADMM for SDPs in standard form

We now present ADAL [5, 8], an alternating direction method of multipliers (ADMM) to address standard SDPs and in particular, able to deal with problem (2). The method we consider is based on the maximization of the augmented Lagrangian built over the dual problem. Let $\bar{X} \in \mathcal{S}_{n+l}$ be the Lagrange multiplier for the dual equation $\bar{\mathcal{A}}^\top y + \bar{Z} - \bar{C} = 0$ and $\sigma > 0$ be fixed. The augmented Lagrangian of the dual (3) is defined as

$$L_\sigma(y, \bar{Z}; \bar{X}) = b^T y - \langle \bar{\mathcal{A}}^\top y + \bar{Z} - \bar{C}, \bar{X} \rangle - \frac{\sigma}{2} \|\bar{\mathcal{A}}^\top y + \bar{Z} - \bar{C}\|^2.$$

In augmented Lagrangian methods applied to the dual (3) the problem

$$\begin{array}{ll} \max & L_\sigma(y, \bar{Z}; \bar{X}) \\ \text{s.t.} & y \in \mathbb{R}^m, \quad \bar{Z} \in \mathcal{S}^+_{n+l}, \end{array} \tag{5}$$

where $\bar{X}$ is fixed and $\sigma > 0$ is a penalty parameter is addressed at every iteration. When the maximization of the augmented Lagrangian $L_\sigma(y, \bar{Z}; \bar{X})$ is performed by optimizing first with respect to y and then with respect to $\bar{Z}$, we are considering the well known alternating direction method of multipliers (ADMM), first proposed in [5, 6] and then extended in [8]. In the following, we refer to this method as ADAL. To be more precise, the new point $(y^{k+1}, \bar{Z}^{k+1}, \bar{X}^{k+1})$ is computed by the following steps:

$$y^{k+1} = \operatorname*{argmax}_{y \in \mathbb{R}^m} L_{\sigma^k}(y, \bar{Z}^k; \bar{X}^k), \tag{6}$$

$$\bar{Z}^{k+1} = \operatorname*{argmax}_{Z \in \mathcal{S}^+_n} L_{\sigma^k}(y^{k+1}, \bar{Z}; \bar{X}^k), \tag{7}$$

$$\bar{X}^{k+1} = \bar{X}^k + \sigma^k(\mathcal{A}^\top y^{k+1} + \bar{Z}^{k+1} - \bar{C}). \tag{8}$$

The update of y in (6) can be performed in closed form, as it derives from the first-order optimality conditions of the problem on the right-hand side of (6): y^{k+1} is the unique solution of

$$\nabla_y L_{\sigma^k}(y, \bar{Z}^k; \bar{X}^k) = b - \bar{\mathcal{A}}(\bar{X}^k + \sigma^k(\bar{\mathcal{A}}^\top y + \bar{Z}^k - \bar{C})) = 0,$$

that is

$$y^{k+1} = (\bar{\mathcal{A}}\bar{\mathcal{A}}^\top)^{-1}\Big(\frac{1}{\sigma^k} b - \bar{\mathcal{A}}(\frac{1}{\sigma^k}\bar{X}^k + \bar{Z}^k - \bar{C})\Big).$$

The update of $\bar{Z}$ in (7) is conducted by considering the equivalent problem

$$\min_{\bar{Z} \in \mathcal{S}^+_{n+l}} \|\bar{Z} + W^{k+1}\|^2, \tag{9}$$

where

$$W^{k+1} = \frac{\bar{X}^k}{\sigma^k} - \bar{C} + \bar{\mathcal{A}}^\top y^{k+1}.$$

Solving problem (9), is equivalent to project $W^{k+1} \in \mathcal{S}_{n+l}$ onto the (closed convex) cone $\mathcal{S}^-_{n+l}$ and take its additive inverse (see Algorithm 1). Such a projection is computed via the spectral decomposition of the matrix W^{k+1}. Finally, it is easy to see that the update of $\bar{X}$ in (8) can be performed considering the projection of $W^{k+1} \in \mathcal{S}_{n+l}$ onto $\mathcal{S}^+_{n+l}$ multiplied by σ^k, namely

$$\bar{X}^{k+1} = \bar{X}^k + \sigma^k(\bar{\mathcal{A}}^\top y^{k+1} + \bar{Z}^{k+1} - \bar{C}) =$$

$$= \sigma^k(\bar{X}^k/\sigma^k - \bar{C} + \bar{\mathcal{A}}^\top y^{k+1} - (\bar{X}^k/\sigma^k - \bar{C} + \bar{\mathcal{A}}^\top y^{k+1})_-) =$$
$$= \sigma^k(\bar{X}^k/\sigma^k - \bar{C} + \bar{\mathcal{A}}^\top y^{k+1})_+.$$

We report in Algorithm 1 the scheme of `ADAL`. `ADAL` is stopped as soon as the following errors

Algorithm 1 Scheme of `ADAL` from [8]

Choose $\sigma > 0$, $\varepsilon > 0$, $\bar{X} \in \mathcal{S}^+_{n+l}$, $\bar{Z} \in \mathcal{S}^+_{n+l}$
$\delta = \max\{r_P, r_D\}$
while $\delta > \varepsilon$ **do**
  $y = (\bar{\mathcal{A}}\bar{\mathcal{A}}^\top)^{-1}\left(\frac{1}{\sigma}b - \bar{\mathcal{A}}(\frac{1}{\sigma}\bar{X} - \bar{C} + \bar{Z})\right)$
  $\bar{Z} = -(\bar{X}/\sigma - \bar{C} + \bar{\mathcal{A}}^\top y)_-$ and $\bar{X} = \sigma(\bar{X}/\sigma - \bar{C} + \bar{\mathcal{A}}^\top y)_+$
  $\delta = \max\{r_P, r_D\}$
  Update σ
end while

related to primal feasibility ($\bar{\mathcal{A}}\bar{X} = b$, $\bar{X} \geq 0$) and dual feasibility ($\bar{\mathcal{A}}^\top y + \bar{Z} + \bar{S} = \bar{C}$) are below a certain accuracy

$$r_P = \frac{\|\bar{\mathcal{A}}\bar{X} - b\|}{1 + \|b\|}, \qquad r_D = \frac{\|\bar{\mathcal{A}}^\top y + \bar{Z} - \bar{C}\|}{1 + \|\bar{C}\|}.$$

More precisely, the algorithm stops as soon as the quantity $\delta = \max\{r_P, r_D\}$ is less than a fixed precision $\varepsilon > 0$. The other optimality conditions (namely $\bar{X} \in \mathcal{S}^+_{n+l}$, $\bar{Z} \in \mathcal{S}^+_{n+l}$, $\bar{Z}\bar{X} = 0$) are satisfied up to machine accuracy throughout the algorithm thanks to the projections employed in `ADAL`. The numerical performance of ADMMs, including the one of `ADAL`, strongly depends on the update rule used for the penalty parameter σ. As in [11, 12], we follow the strategy by Lorenz and Tran-Dinh [13], considering at every iteration k the ratio between the norm of the primal variable $\bar{X}^k$ and norm of the dual variable $\bar{Z}^k$.

The memory required to store the augmented matrices $\bar{C}, \bar{A}_i, \bar{A}_j, \bar{Z}$ and $\bar{X}$ gets large with the number l of inequalities and even using efficient sparse matrix implementations may be insufficient to computationally deal with large scale problems. Our idea is then to rewrite the steps of `ADAL` in terms of the original matrices C, A^i and X, so that one can keep in memory only the matrices that are actually defining the problem. Indeed, let $1 \leq i \leq l$ be a generic index of an inequality constraint and let $l + 1 \leq j \leq m$ be a generic index of an equality constraint, then the following holds:

$$\begin{aligned}\langle \bar{A}^i, \bar{X}\rangle &= \langle A^i, X\rangle + s_i,\\ \langle \bar{A}^j, \bar{X}\rangle &= \langle A^j, X\rangle,\\ \langle \bar{C}, \bar{X}\rangle &= \langle C, X\rangle;\end{aligned}$$

The linear map applied to $\bar{X}$ becomes:

$$\bar{\mathcal{A}}(\bar{X}) = \begin{pmatrix} \langle A^1, X\rangle \\ \vdots \\ \langle A^l, X\rangle \\ \langle A^{l+1}, X\rangle \\ \vdots \\ \langle A^m, X\rangle \end{pmatrix} + \begin{pmatrix} s^T \\ \mathbf{0}_{m-l} \end{pmatrix} = \mathcal{A}(X) + \begin{pmatrix} s^T \\ \mathbf{0}_{m-l.} \end{pmatrix}$$

Similarly, the adjoint operator $\bar{\mathcal{A}}^T : \mathbb{R}^m \to \mathcal{S}_{n+l}$ of $\bar{\mathcal{A}}$ is defined as

$$\bar{\mathcal{A}}^T y := \sum_{m}^{i=1} y_i \bar{A}^i = \begin{pmatrix} \sum_m^{i=1} y_i A^i & & \mathbf{0}_{n,l} \\ & y_1 & \\ \mathbf{0}_{n,l} & & \ddots \\ & & & y_l \end{pmatrix} = \begin{pmatrix} \mathcal{A}^T y & & \mathbf{0}_{n,l} \\ & y_1 & \\ \mathbf{0}_{n,l} & & \ddots \\ & & & y_l \end{pmatrix}.$$

Using the operator vec, we can write $\bar{\mathcal{A}}(\bar{X}) = b$ as $\bar{A}\ \mathrm{vec}(\bar{X}) = b$, where

$$\bar{A} := \left(\mathrm{vec}(\bar{A}^1), \ldots, \mathrm{vec}(\bar{A}^m)\right)^T \in \mathbb{R}^{m \times (n+l)^2}.$$

Note that matrix $\bar{A}^i,\ i = 1, \ldots, l$, corresponding to the i-th inequality constraint, is the unique matrix having 1 in position $(n+i, n+i)$. Then, $\bar{A}\bar{A}^T$ can be expressed in terms of AA^T as follows:

$$\bar{A}\bar{A}^T = AA^T + \mathrm{diag}\begin{pmatrix} \mathbf{1}_l \\ \mathbf{0}_{m-l} \end{pmatrix},$$

as the zero entries of $\bar{A}^i\ i = 1, \ldots, m$, do not contribute in the row-by-column product and the 1 in position $(n+i, n+i)$ contributes only to the entry where $\mathrm{vec}(\bar{A}^i)$ is multiplied by itself, i.e., in position (i,i) of $\bar{A}\bar{A}^T$. According to the notation introduced, the update of the y variable can be rewritten as follows:

$$y^{k+1} = \left(AA^T + \mathrm{Diag}\begin{pmatrix} \mathbf{1}_l \\ \mathbf{0}_{m-l} \end{pmatrix}\right)^{-1} \left(\frac{1}{\sigma^k} b - A\ \mathrm{vec}\left(\frac{1}{\sigma^k} X^k - C + Z^k\right) + \begin{pmatrix} \frac{1}{\sigma^k} {s^k}^T + {p^k}^T \\ \mathbf{0}_{m-l} \end{pmatrix}\right)$$

Furthermore, the spectral decomposition of the matrix W, needed for updating the variable $\bar{X}$ and $\bar{Z}$ can be computed in a "block-wise" fashion. At a generic iteration of `ADAL`, matrix W can be written as follows:

$$W^{k+1} = \begin{pmatrix} \frac{X^k}{\sigma^k} - C + \mathcal{A}^\top y^{k+1} & & \mathbf{0}_{n,l} \\ & & \\ \mathbf{0}_{n,l} & & \mathrm{Diag}\left(\frac{{s^k}^T}{\sigma^k} + \begin{pmatrix} y_1^{k+1} \\ \vdots \\ y_l^{k+1} \end{pmatrix}\right) \end{pmatrix}.$$

Then, in order to compute the eigenvalues and eigenvectors of W^{k+1}, we can first compute the spectral decomposition of the matrix $\frac{X^k}{\sigma^k} - C + \mathcal{A}^T y^{k+1}$, then we trivially get the eigenvalues and eigenvectors of the diagonal part of W^{k+1} and eventually we adjust the dimension of the eigenvectors computed, in order to have them in $\mathbb{R}^{n+l}$.

The convergence of the scheme introduced is inherited by the convergence of `ADAL` [8]. In particular, Algorithm 1 can be interpreted as a fixed point method and we can state the following result

Theorem 1 *The sequence $\{(\bar{X}^k, y^k, \bar{Z}^k)\}$ generated by Algorithm 1 from any starting point $(\bar{X}^0, y^0, \bar{Z}^0)$ converges to a solution $(\bar{X}^*, y^*, \bar{Z}^*) \in \Omega^*$, where Ω^* is the set of primal and dual solutions of (2) and (3).*

2.2 Obtaining dual bounds

SDPs are used to approximate combinatorial problems: the optimal solution of a semidefinite relaxation can be computed in polynomial time and generally gives a better bound than that obtained solving a linear relaxation (see e.g. [14]). Given a pair of primal-dual SDPs, weak and strong duality hold under the assumption that both problems are strictly feasible. Duality results imply that the objective function value of every feasible solution of the dual SDP is a valid bound on the optimal objective function value of the primal. Therefore, every dual feasible solution and in particular the optimal dual solution of an SDP relaxation, gives a valid bound on the solution of the related combinatorial optimization problem. Hence, being able to compute dual feasible solutions - even of moderate quality - can be extremely useful when considering branch-and-bound frameworks to define exact solution methods for specific combinatorial optimization problems. Following ideas developed in [11], we define a post-processing procedure for `ADAL` on general SDPs, that allows to get a feasible dual solution starting from a positive semidefinite matrix $\tilde{Z} \in \mathcal{S}_n^+$. Let $\mathcal{A}_{ineq}$ and $\mathcal{A}_{eq}$ be the linear operators defining the inequality and equality constraints in problem (1): $\mathcal{A}_{ineq} = \langle A^i, X \rangle$ with $A^i \in \mathcal{S}_n$, $i = 1, \ldots, l$ and $\mathcal{A}_{eq} = \langle A^j, X \rangle$ with $A^j \in \mathcal{S}_n$, $j = l+1, \ldots, m$. Let b_{ineq} and b_{eq} be the right hand side vectors accordingly defined. Introducing the adjoint operators of $\mathcal{A}_{ineq}$ and $\mathcal{A}_{eq}$, the dual problem (3) can be equivalently written as

$$\begin{array}{ll} \max & -b_{ineq}^T \lambda + b_{eq}^T \mu \\ \text{s.t.} & C + \mathcal{A}_{ineq}^\top \lambda - \mathcal{A}_{eq}^\top \mu = Z \\ & Z \in \mathcal{S}_n^+, \ \lambda \geq 0, \end{array} \tag{10}$$

with $\lambda \in \mathbb{R}^l$ and $\mu \in \mathbb{R}^{m-l}$. We can then extend the results proposed in [11] and define a procedure to get feasible solutions of problem (10) and then, by weak duality, valid bounds on the optimal objective function value of the primal (1). Let $\tilde{Z} \in \mathcal{S}_n^+$. If the linear programming problem

$$\begin{array}{ll} \max & -b_{ineq}^T \lambda + b_{eq}^T \mu \\ \text{s.t.} & C + \mathcal{A}_{ineq}^\top \lambda - \mathcal{A}_{eq}^\top \mu = \tilde{Z} \\ & \lambda \geq 0 \end{array} \tag{11}$$

has an optimal solution $(\tilde{\lambda}, \tilde{\mu}) \in \mathbb{R}^m$, then $(\tilde{\lambda}, \tilde{\mu}, \tilde{Z})$ is a feasible solution for (10) and the value $-b_{ineq}^T \tilde{\lambda} + b_{eq}^T \tilde{\mu}$ is giving a dual bound. If (11) is unbounded, then also (10) is unbounded and hence the primal (1) is not feasible. If (11) is infeasible, it means that the starting $\tilde{Z} \in \mathcal{S}_n^+$ does not allow to find a feasible dual solution and then get a dual bound. From a practical point of view, once problem (1) is approximately solved by `ADAL`, we can try to get a feasible solution of problem (10), by addressing problem (11). This is what we have implemented, using GUROBI 9.1.1 [15] as solver for problem (11).

3 Bounding the clique number and the chromatic number of a graph

Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a subset of $W \subseteq V$ is a *clique* if every two vertices in W are adjacent while a subset of $W \subseteq V$ is called *stable* if no two vertices in W are adjacent. The *clique number* $\omega(G)$ and

the *stability number* $\alpha(G)$ are the maximum cardinality of a clique in G and the maximum cardinality of a stable set in G, respectively. A k-coloring is a partition of V into k stable sets. The *chromatic number* $\chi(G)$ is the smallest integer k for which G has a k-coloring. Denoting with $\bar{G} = (V, \bar{E})$ the complementary graph of G, it holds

$$\omega(\bar{G}) = \alpha(G) \leq \chi(\bar{G}).$$

Lovàsz [16] introduced the so called *theta number* $\vartheta(G)$ that is an upper bound for the clique number $\omega(\bar{G})$ and the stability number $\alpha(G)$ and is a lower bound for the chromatic number $\chi(\bar{G})$. The important property of $\vartheta(G)$ is that it can be computed with an arbitrary precision in polynomial time, as it is the optimal value of the following SDP [17]:

$$\begin{array}{rlll} \vartheta(G) = & \max & \langle J, X \rangle & \\ & \text{s.t.} & \text{trace}(X) = 1 & \\ & & X_{ij} = 0 & \{i,j\} \in E(G) \\ & & X \in \mathcal{S}_n^+, & \end{array}$$

where J is the n-by-n matrix of all ones, being $n = |V|$. Starting from this relaxation, several attempts for sharpening $\vartheta(G)$ as a bound for $\omega(\bar{G})$, $\alpha(G)$ and $\chi(\bar{G})$ have been made (see, e.g., [18, 19, 20, 21, 22, 23, 24]). As a first way to improve $\vartheta(G)$, we consider the numbers $\vartheta_+(G)$ and $\bar{\vartheta}_+(G)$ obtained as solutions of the following SDPs, where bounds on the entries of the matrix variables are introduced:

$$\begin{array}{rlll} \vartheta_+(G) = & \max & \langle J, X \rangle & \\ & \text{s.t.} & \text{trace}(X) = 1 & \\ & & X_{ij} = 0 & \{i,j\} \in E(G) \\ & & X \geq 0 & \\ & & X \in \mathcal{S}_n^+ & \end{array} \qquad \begin{array}{rlll} \bar{\vartheta}_+(G) = & \max & \langle J, X \rangle & \\ & \text{s.t.} & \text{trace}(X) = 1 & \\ & & X_{ij} \leq 0 & \{i,j\} \in E(G) \\ & & X \in \mathcal{S}_n^+. & \end{array}$$

The values $\vartheta_+(G)$ and $\bar{\vartheta}_+(G)$ are related to $\omega(\bar{G})$, $\alpha(G)$ and $\chi(\bar{G})$ as follows

$$\omega(\bar{G}) = \alpha(G) \leq \vartheta_+(G) \leq \vartheta(G) \leq \bar{\vartheta}_+(G) \leq \chi(\bar{G}).$$

In the literature, equivalent formulations for both $\vartheta_+(G)$ and $\bar{\vartheta}_+(G)$ have been proposed [2] and for our computational experience, we consider the following formulation for $\bar{\vartheta}_+(G)$:

$$\begin{array}{rlll} \bar{\vartheta}_+(G) = & \min & t & \\ & \text{s.t.} & X_{ii} = t - 1 & i \in V(G) \\ & & X_{ij} = -1 & \{i,j\} \in \bar{E}(G) \\ & & X_{ij} \geq -1 & \{i,j\} \in E(G) \\ & & X \in \mathcal{S}_n^+. & \end{array}$$

Note that in both the formulations of $\vartheta_+(G)$ and $\bar{\vartheta}_+(G)$ the entries of the matrix X are bounded from below. In the context of ADMMs defined over the dual problem, bounds on the matrix variable can be handled by introducing a further step, where a projection onto the nonnegative orthant is performed (see e.g. [8, 11, 12]). Although these 3-blocks ADMMs may not theoretically converge [25], they perform well in practice.

4 Numerical results

In this section we report our computational study: we compare the performance of `ADAL` and `SDPNAL+` [1] on randomly generated instances and on instances from SDP relaxations of the stable set problem (equivalent to the max clique problem) and the graph coloring problem. `SDPNAL+` implements an ADMM combined with a semismooth Newton-Conjugate Gradient method. `SDPNAL+` is implemented in MATLAB, with some subroutines in C language incorporated via Mex files. For our comparison, we considered the version of `SDPNAL+` available at `https://blog.nus.edu.sg/mattohkc/softwares/sdpnalplus/`. Our version of `ADAL` is implemented in MATLAB R2020a and uses its built-in functions. Both our implementation of `ADAL` and the instances used in our numerical experience can be downloaded from `https://github.com/batt95/ADAL-ineq`. In the implementation of `SDPNAL+` a refined management of the matrices is implemented exploiting their symmetry and allows the optimization of the subroutines used throughout the application of the algorithm.

The experiments were carried out on an Intel(R) Xeon(R) CPU E5-2698 v4 running at 2.20GHz, with 256GB of RAM, under Linux (Ubuntu 16.04.7).

We compare the performance of the algorithms using performance profiles as proposed by Dolan and Moré [26]. Given a set of solvers $\mathcal{S}$ and a set of problems $\mathcal{P}$, the performance of a solver $s \in \mathcal{S}$ on problem $p \in \mathcal{P}$ is compared against the best performance obtained by any solver in $\mathcal{S}$ on the same problem. The performance ratio is defined as $r_{p,s} = t_{p,s} / \min\{t_{p,s'} \mid s' \in \mathcal{S}\}$, where $t_{p,s}$ is the measure we want to compare, and we consider a cumulative distribution function $\rho_s(\tau) = |\{p \in \mathcal{P} \mid r_{p,s} \leq \tau\}|/|\mathcal{P}|$. The performance profile for $s \in S$ is the plot of the function ρ_s.

4.1 Comparison on randomly generated instances

The random instances considered in the first experiment are obtained from the instance generator used in [6]. Given n, m and a percentage p, we built 5 instances having $round(pm)$ number of inequalities. In Table 4.1, we report the comparison between `ADAL` and `SDPNAL+` in terms of number of iterations and CPU time needed in order to reach an accuracy of 10^{-5}. We consider instances with $n \in \{200, 250, 500, 1000\}$, $m \in \{5000, 10000, 25000, 50000, 100000\}$ and $p \in \{0.25, 0.5, 0.75\}$. We excluded those combinations of n and m leading to matrices A_i with linearly dependent rows. We set a time limit of 1800 seconds CPU time.

As a preliminary test, we ran the version of `ADAL` tailored for SDPs in standard form. However, this non-optimized version of `ADAL` did not allow us to solve any instance due to memory issues. Therefore, in the following comparisons, we only consider the version of `ADAL` described in Section 2.

In Table 4.1, for each solver and each combination of n, m and p, we report the number of instances solved within the time limit and the average running time. We notice that for $n = 250$ and $m = 25000$, `SDPNAL+` is not able to solve any instance within the time limit, while `ADAL` is able to solve all of them with a precision of 10^{-5}. For $n = 500$ and $m = 100000$ both algorithms are not able to solve any instance within the time limit. `SDPNAL+` performs better on instances with $n = 1000$ and $m = 10000$, while for the other instances either the two solvers show similar performances or `ADAL` outperforms `SDPNAL+`. The performance profiles of `ADAL` and `SDPNAL+` on random instances are reported in Figure 1, showing the better performance of `ADAL` with respect to `SDPNAL+`: on almost 60% of the instances `ADAL` is the fastest algorithm and it is also able to solve 90% of the instances while `SDPNAL+` solves only 80% of the instances

			ADAL		SDPNAL+	
n	m	p (%)	#sol	CPU time	#sol	CPU time
200	10000	25	5	39.24	5	33.05
		50	5	58.24	5	109.14
		75	5	67.14	5	713.82
250	5000	25	5	7.99	5	11.05
		50	5	9.87	5	15.51
		75	5	11.28	5	16.93
	25000	25	5	838.04	0	-
		50	5	1166.45	0	-
		75	5	1114.52	0	-
500	10000	25	5	15.52	5	15.54
		50	5	16.49	5	22.45
		75	5	28.87	5	23.94
	25000	25	5	18.11	5	31.33
		50	5	30.20	5	50.78
		75	5	45.53	5	52.57
	50000	25	5	217.61	5	106.28
		50	5	260.43	5	221.66
		75	5	325.71	5	250.97
	100000	25	0	-	0	-
		50	0	-	0	-
		75	0	-	0	-
1000	10000	25	5	136.63	5	49.52
		50	5	157.21	5	58.22
		75	5	242.63	5	71.38
	50000	25	5	57.19	5	60.96
		50	5	94.09	5	109.48
		75	5	110.00	5	111.29
	100000	25	5	83.15	5	136.53
		50	5	127.37	5	181.13
		75	5	155.05	5	184.21

Table 1: Results on 120 random instances

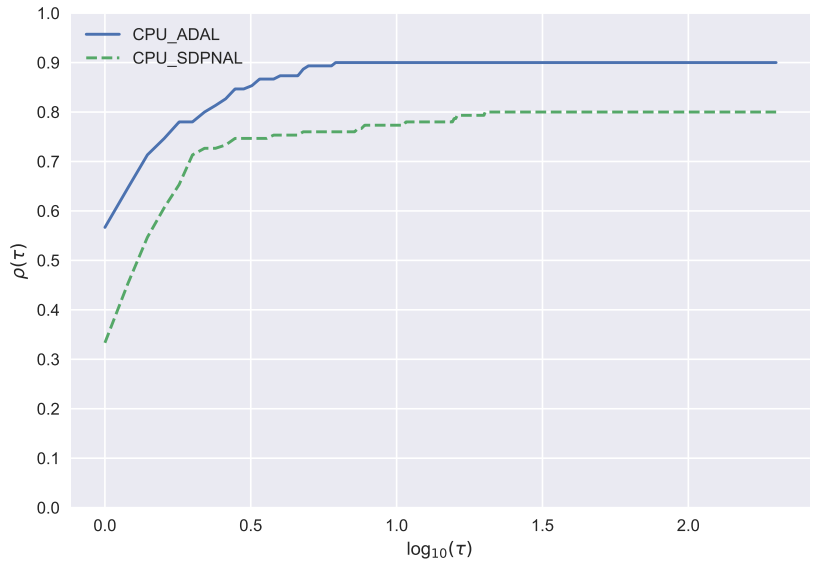

Figure 1: Performance profiles on CPU time. Comparison between `ADAL` and `SDPNAL+` on random instances.

within the time limit.

4.2 Comparison on instances from SDP relaxations of the maximum clique problem

In the following, we report the results on the SDP relaxation $\vartheta_+(G)$ for bounding the clique number (or the stability number) of a graph. We considered graphs from the second DIMACS implementation challenge [27], available at `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique`. These graphs form the standard benchmark for the maximum clique problem. In order to use them, we complemented the graphs to convert the maximum clique instances into stable set problem instances. Apart from applying `ADAL` and `SDPNAL+` for finding $\vartheta_+(G)$, stopping the algorithms as soon as the optimality conditions were satisfied with a precision of 10^{-6}, we applied the post-processing procedure every 200 iterations of `ADAL` and at the very last iteration performed by `ADAL`. Everytime the post-processing procedure is called, we give as input the matrix Z^k obtained by `ADAL` at the corresponding iteration k and solve the linear programming problem (10) using Gurobi [15]. Along the iterations of `ADAL`, we keep in memory the best dual bound found by the post-processing procedure, together with the CPU time needed to detect it. Note that every dual bound computed by the post-processing procedure and in particular the best dual bound is a valid upper bound on the stability number.

In Table 2, we report the following data for the comparison between `ADAL` and `SDPNAL+`: for each instance, we report its name (Graph), the dual objective function value obtained by `ADAL` and `SDPNAL+`, the best dual bound found along the iterations of `ADAL` ($\vartheta_+(G)$), the CPU time needed by `ADAL` and `SDPNAL+` to satisfy the stopping criterion, the CPU time needed to recover the best dual bound found by `ADAL` and the overall time needed to apply the post-processing procedure (CPU times).

We have that the post-processing procedure applied on `ADAL` is able to compute valid dual

bounds on every instance but on `keller6`, where `ADAL` shows a failure. On `p_hat1500-2` even if both `ADAL` and `SDPNAL+` did not converge within 3600 seconds, the post-processing procedure is able to compute a valid dual bound. Note that for huge graphs the time needed to find the best dual bound may be greater than the time needed by `ADAL` to converge and this comes from the fact that the best dual bound can be recovered at the last iteration computed. Note also the the overall time needed to apply the post-processing procedure in `ADAL` is small with respect the overall time needed by the algorithm, and clearly it may be lowered by seldom applying the procedure.

In Figure 2, we report the performance profiles obtained with respect to the CPU time needed by `ADAL`, the CPU time for computing the value of the best dual bound with `ADAL` and CPU time needed by `SDPNAL+`. It is clear that on these instances `SDPNAL+` outperforms `ADAL`. However, we want to underline the good performances of `ADAL` on the `p-hat` graphs, where we are often able to get the same bound as the optimal dual objective of `SDPNAL+` in a much lower CPU time.

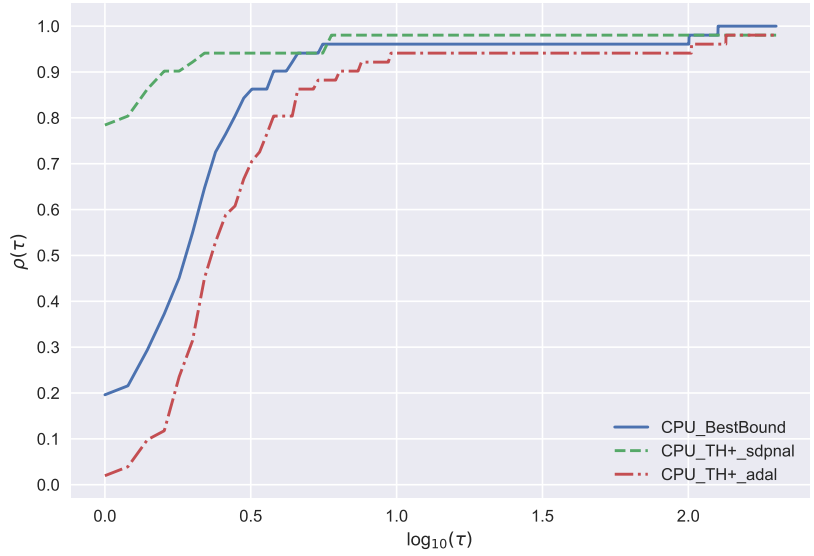

Figure 2: Performance profiles on CPU time. Comparison among `ADAL`, BestBound and `SDPNAL+` on the computation of $\vartheta_+(G)$.

4.3 Comparison on instances from SDP relaxations of the coloring problem

In the following, we report the results on the SDP relaxation $\bar{\vartheta}_+(G)$ for bounding the chromatic number of a graph. As before, we considered graphs from the second DIMACS implementation challenge [27], available at `https://sites.google.com/site/graphcoloring/files`. We applied `ADAL` and `SDPNAL+`, stopping the algorithms as soon as the optimality conditions were satisfied with a precision of 10^{-6}. As for bounding $\vartheta_+(G)$, we applied the post-processing procedure every 200 iterations of `ADAL` and at the very last iteration performed by `ADAL`. Every dual bound computed by the post-processing procedure and in particular the best dual bound is a valid lower bound on the chromatic number.

In Table 3 and Table 4, we report the following data for the comparison between `ADAL` and `SDPNAL+`: for each instance, we report its name (Graph), the dual objective function

	$\vartheta_+(G)$			CPU times			
Graph	ADAL	SDPNAL+	BestBound	ADAL	SDPNAL+	BestBound	post-proc
DSJC125.1	38.04	38.04	38.04	2.89	3.52	2.56	0.12
DSJC125.5	11.40	11.40	11.40	1.93	0.77	1.70	0.09
DSJC125.9	4.00	4.00	4.00	2.41	1.17	2.44	0.09
DSJC500-5	22.57	22.57	22.57	6.83	4.85	7.30	0.48
DSJC1000-5	31.67	31.67	31.67	41.60	34.32	43.69	3.59
C125-9	37.55	37.55	37.55	2.88	0.99	2.73	0.11
C250-9	55.82	55.82	55.82	7.82	3.12	7.15	0.43
C500-9	83.58	83.58	83.58	30.55	8.32	31.03	1.48
C1000-9	122.60	122.60	122.60	159.79	34.93	147.00	9.74
C2000-5	44.56	44.56	44.56	389.59	534.67	398.86	22.42
C2000-9	177.73	177.73	177.73	1238.53	278.60	1247.62	68.30
brock200_1	27.20	27.20	27.20	3.45	1.06	3.12	0.24
brock200_2	14.13	14.13	14.13	1.91	1.08	1.98	0.15
brock200_3	18.67	18.67	18.67	2.24	1.07	2.31	0.15
brock200_4	21.12	21.12	21.12	2.83	0.96	2.92	0.18
brock400_1	39.33	39.33	39.33	7.81	3.84	8.09	0.53
brock400_2	39.20	39.20	39.20	8.30	4.02	8.58	0.50
brock400_3	39.16	39.16	39.16	8.64	3.59	8.92	0.48
brock400_4	39.23	39.23	39.23	7.99	3.53	8.27	0.49
brock800_1	41.87	41.87	41.87	24.31	11.67	20.96	2.66
brock800_2	42.10	42.10	42.10	23.94	12.88	20.49	2.59
brock800_3	41.88	41.88	41.88	24.52	13.02	25.87	2.57
brock800_4	42.00	42.00	42.00	23.91	12.80	20.28	2.50
p_hat300-1	10.02	10.02	10.02	18.45	16.72	8.85	0.95
p_hat300-2	26.71	26.71	26.71	211.40	161.90	28.17	11.20
p_hat300-3	40.70	40.70	40.70	35.69	36.28	16.91	1.78
p_hat500-1	13.01	13.01	13.01	34.11	14.71	22.55	2.04
p_hat500-2	38.56	38.56	38.56	580.86	537.38	92.52	32.79
p_hat500-3	57.81	57.81	57.81	99.65	33.72	61.56	5.03
p_hat700-1	15.05	15.05	15.05	59.86	33.94	43.15	4.20
p_hat700-2	48.44	48.44	48.44	1161.67	295.99	218.26	71.55
p_hat700-3	71.76	71.76	71.76	293.90	93.48	162.79	15.91
p_hat1000-1	17.52	17.52	17.52	144.76	119.26	84.75	10.38
p_hat1000-2	54.84	54.84	54.84	1815.65	697.11	487.41	121.86
p_hat1000-3	83.53	83.53	83.53	473.77	243.21	293.35	28.77
p_hat1500-1	21.89	21.89	21.89	606.67	479.28	471.58	41.41
p_hat1500-2	-	-	76.46	-	-	1826.66	233.69
p_hat1500-3	113.65	113.65	113.65	3014.42	879.45	1886.51	202.90
keller4	13.47	13.47	13.47	3.35	1.46	2.69	0.17
keller5	31.00	31.00	31.00	503.36	53.31	289.31	30.06
keller6	-	63.00	-	-	1524.62	-	146.42
sanr200_0.7	23.63	23.63	23.63	3.44	1.26	3.21	0.25
sanr200_0.9	48.90	48.90	48.90	6.04	1.73	4.80	0.34
sanr400_0.5	20.18	20.18	20.18	6.60	3.80	6.19	0.57
sanr400_0.7	33.97	33.97	33.97	7.21	4.05	7.49	0.51
MANN_a9	17.48	17.48	17.47	0.48	0.28	0.46	0.05
MANN_a27	132.76	132.76	132.76	561.87	5.48	550.52	30.80
hamming6-2	32.00	32.00	32.00	1.49	0.33	1.25	0.10
hamming6-4	4.00	4.00	4.00	0.10	0.08	0.11	0.01
hamming8-2	128.00	128.00	128.00	532.92	3.96	500.17	30.03
hamming8-4	16.00	16.00	16.00	2.62	1.13	2.60	0.24
hamming10-4	42.67	42.67	42.67	97.36	31.77	93.90	7.46

Table 2: Results on $\vartheta_+(G)$, graphs from the second DIMACS implementation challenge.

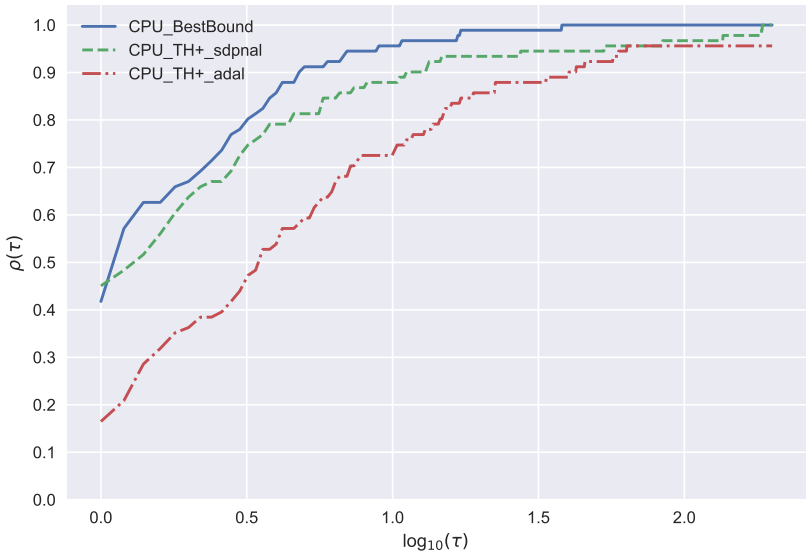

Figure 3: Performance profiles on CPU time. Comparison among ADAL, BestBound and SDPNAL+ on the computation of $\bar{\vartheta}_+(G)$.

value obtained by ADAL and SDPNAL+, the best dual bound found along the iterations of ADAL ($\bar{\vartheta}_+(G)$), the CPU time needed by ADAL and SDPNAL+ to satisfy the stopping criterion, the CPU time needed to recover the best dual bound found by ADAL and the overall time needed to apply the post-processing procedure (CPU times).

We notice that the post-processing procedure fails in finding bounds on the chromatic number for several graphs, 19 out of 113 graphs. This is due to the precision of the dual matrix given as input, that may be too low to detect a dual feasible solution. We also notice that on some graphs the bound obtained is slightly bigger than the dual objective function value obtained by ADAL. This is due to the precision asked to Gurobi to solve the LP in the post-processing phase. We imposed a feasibility precision of 10^{-5}. Asking for a higher precision would often result in a failure for the post-processing procedure. We are then showing what in our opinion is a good trade off between feasibility precision and quality of the bound. The CPU time needed to compute the BestBound is often much lower with respect to the time needed by SDPNAL+ to converge and this is confirmed by the performance profiles shown in Figure 3. In the performance profiles, we excluded those instances for which the difference in absolute value of the BestBound found by ADAL and the dual objective of SDPNAL+ is less than 0.5. In particular, we excluded all the instances where the post-processing procedure was not able to compute a bound.

As a further comparison between ADAL and SDPNAL+ on SDP relaxations of the chromatic number, we built instances adding 1000, 2500 and 5000 inequalities to $\bar{\vartheta}(G)$. The inequalities are chosen randomly from those proposed by Dukanovic and Rendl [18] to strengthen $\bar{\vartheta}(G)$:

$$X_{ij} + X_{ik} - X_{jk} \leq t - 1, \quad \forall\, i, j, k \in V(G).$$

In Table 5, we report the results on some classes of graphs where the CPU time needed to compute the BestBound is often lower with respect to the time needed by SDPNAL+ to converge with a precision of 10^{-6}.

	$\bar{\vartheta}_+(G)$			CPU times			
Graph	ADAL	SDPNAL+	BestBound	ADAL	SDPNAL+	BestBound	post-proc
DSJC125.1	4.14	4.14	4.14	69.01	22.88	1.72	0.91
DSJC125.5	11.87	11.87	11.87	1.02	1.36	0.74	0.06
DSJC125.9	37.80	37.80	37.80	1.87	2.03	1.31	0.09
DSJC250.1	4.94	4.94	4.94	6.65	6.70	2.19	0.16
DSJC250.5	16.35	16.35	16.35	1.90	2.84	2.00	0.10
DSJC250.9	55.22	55.22	55.22	4.76	3.76	3.76	0.37
DSJC500.1	6.25	6.25	6.25	8.66	16.57	5.91	0.16
DSJC500.5	22.90	22.90	22.90	5.41	9.64	5.71	0.30
DSJC500.9	84.14	84.14	84.14	17.04	16.36	17.53	1.21
DSJR500.1	12.00	12.00	12.00	35.18	10.00	23.59	0.34
DSJR500.1c	83.75	83.75	83.75	-	1231.74	190.31	294.26
DSJR500.5	122.01	122.00	122.00	198.08	16.95	181.80	6.48
DSJC1000.1	8.36	8.36	8.36	31.65	59.76	22.83	0.57
DSJC1000.5	32.11	32.11	32.11	18.30	38.11	19.46	1.17
DSJC1000.9	122.80	122.80	122.80	72.30	63.88	70.85	6.89
fpsol2.i.1	65.00	65.00	65.00	200.57	11.71	199.60	2.67
fpsol2.i.2	30.00	30.00	30.00	28.11	9.75	25.71	0.43
fpsol2.i.3	30.00	30.00	30.00	27.36	7.82	27.43	0.42
inithx.i.1	54.00	54.00	54.00	604.51	32.25	537.71	4.96
inithx.i.2	31.00	31.00	30.22	387.80	12.39	35.60	3.70
inithx.i.3	31.00	31.00	30.23	341.12	13.64	32.57	3.45
latin_square_10	90.00	89.99	-	48.40	41.12	-	2.91
le450_15a	15.00	15.00	-	6.37	5.18	-	0.12
le450_15b	15.00	15.00	15.00	7.06	5.61	7.13	0.14
le450_15c	15.00	15.00	15.00	3.92	4.70	4.02	0.10
le450_15d	15.00	15.00	15.00	3.86	4.71	3.96	0.10
le450_25a	25.00	25.00	25.00	19.73	7.54	19.53	0.29
le450_25b	25.00	25.00	-	18.44	7.27	-	0.23
le450_25c	25.00	25.00	25.00	9.67	7.03	9.78	0.20
le450_25d	25.00	25.00	25.00	9.15	6.82	9.25	0.19
mulsol.i.1	49.00	49.00	-	18.89	3.48	-	0.66
mulsol.i.2	31.00	31.00	31.00	9.02	2.92	9.04	0.28
mulsol.i.3	31.00	31.00	31.00	8.13	3.12	7.47	0.19
mulsol.i.4	31.00	31.00	31.00	7.97	2.40	6.31	0.22
mulsol.i.5	31.00	31.00	31.00	9.88	3.34	9.01	0.19
school1	14.00	14.00	14.00	14.74	65.03	8.08	0.40
school1_nsh	14.00	14.00	14.00	12.12	75.97	7.29	0.30
zeroin.i.1	49.00	49.00	49.00	24.91	2.47	21.91	0.80
zeroin.i.2	30.00	30.00	30.00	14.63	2.43	14.13	0.48
zeroin.i.3	30.00	30.00	30.00	14.62	2.80	13.53	0.46
anna	11.00	11.00	-	9.97	1.16	-	0.13
david	11.00	11.00	-	2.46	0.59	-	0.08
huck	11.00	11.00	-	1.60	0.43	-	0.04
jean	10.00	10.00	-	1.35	0.53	-	0.03
games120	9.00	9.00	-	3.23	0.86	-	0.07
miles250	8.00	8.00	8.00	7.17	0.94	6.30	0.11
miles500	20.00	20.00	20.00	6.78	1.69	6.26	0.11
miles750	31.00	31.00	31.00	4.75	2.73	4.77	0.09
miles1000	42.00	42.00	42.00	7.81	1.64	7.61	0.16
miles1500	73.00	73.00	73.00	10.36	1.48	10.28	0.27

Table 3: Results on $\bar{\vartheta}_+(G)$, graphs from the second DIMACS implementation challenge.

	$\bar{\vartheta}_+(G)$			CPU times			
Graph	ADAL	SDPNAL+	BestBound	ADAL	SDPNAL+	BestBound	post-proc
queen5_5	5.00	5.00	5.00	0.01	0.11	0.04	0.03
queen6_6	6.04	6.04	6.04	0.77	0.69	0.19	0.04
queen7_7	7.00	7.00	7.00	0.08	0.29	0.08	0.01
queen8_8	8.00	8.00	8.00	0.10	0.19	0.11	0.01
queen8_12	12.00	12.00	-	0.55	0.62	-	0.02
queen9_9	9.00	9.00	9.00	0.15	0.23	0.16	0.01
queen10_10	10.00	10.00	10.00	0.23	0.44	0.24	0.01
queen11_11	11.00	11.00	11.00	0.46	0.47	0.47	0.01
queen12_12	12.00	12.00	12.00	0.67	0.68	0.71	0.04
queen13_13	13.00	13.00	13.00	0.76	0.64	0.80	0.04
queen14_14	14.00	14.00	14.00	1.27	0.82	1.32	0.04
queen15_15	15.00	15.00	-	1.38	1.26	-	0.04
queen16_16	16.00	16.00	16.00	1.84	1.46	1.90	0.06
myciel3	2.40	2.40	2.40	0.01	0.13	0.04	0.03
myciel4	2.53	2.53	2.53	0.04	0.18	0.04	0.01
myciel5	2.64	2.64	2.64	0.41	0.41	0.23	0.02
myciel6	2.73	2.73	2.73	1.73	1.16	0.51	0.04
myciel7	2.82	2.82	2.82	7.35	7.60	1.34	0.24
mug88_1	3.00	3.00	3.00	11.78	29.45	0.35	0.24
mug88_25	3.00	3.00	3.00	20.81	47.43	0.35	0.43
mug100_1	3.00	3.00	3.00	19.59	84.51	0.46	0.31
mug100_25	3.00	3.00	3.00	26.20	84.97	0.46	0.39
abb313GPIA	8.00	8.00	8.01	615.04	2949.22	55.61	4.48
ash331GPIA	3.38	3.38	3.38	125.34	17.85	38.24	1.01
ash608GPIA	3.33	3.33	3.31	265.72	41.34	129.25	1.34
ash958GPIA	3.33	3.33	-	529.68	124.35	0.00	2.51
will199GPIA	6.10	6.10	6.10	156.39	32.11	124.61	1.22
1-Insertions_4	2.23	2.23	2.23	1.93	1.01	0.34	0.07
1-Insertions_5	2.28	2.28	2.28	19.71	15.04	2.64	0.56
1-Insertions_6	2.31	2.31	2.31	337.22	100.65	22.80	3.29
2-Insertions_3	2.10	2.10	2.10	0.38	0.57	0.18	0.04
2-Insertions_4	2.13	2.13	2.13	25.27	9.06	1.59	0.36
2-Insertions_5	2.16	2.16	2.16	544.91	109.90	52.67	4.99
3-Insertions_3	2.07	2.07	2.07	1.22	1.00	0.31	0.06
3-Insertions_4	2.09	2.09	2.09	125.48	29.79	8.39	2.37
3-Insertions_5	-	2.10	2.11	-	3568.47	130.38	17.94
4-Insertions_3	2.05	2.05	2.05	2.39	2.75	0.38	0.08
4-Insertions_4	2.06	2.06	2.06	563.58	130.23	8.89	6.64
1-FullIns_3	3.06	3.06	3.06	0.32	0.35	0.13	0.05
1-FullIns_4	3.12	3.12	3.12	4.37	2.45	1.39	0.10
1-FullIns_5	3.18	3.18	3.18	71.27	17.55	18.65	1.52
2-FullIns_3	4.03	4.03	4.03	1.34	0.39	0.74	0.08
2-FullIns_4	4.06	4.06	4.06	57.51	9.21	26.76	1.64
2-FullIns_5	4.08	4.08	4.08	2670.31	184.26	381.59	19.29
3-FullIns_3	5.02	5.02	5.02	6.12	1.18	4.47	0.15
3-FullIns_4	5.03	5.03	5.03	329.51	24.03	58.31	4.94
3-FullIns_5	-	5.05	5.04	-	1769.01	2965.54	18.40
4-FullIns_3	6.01	6.01	6.01	21.19	2.30	1.65	0.32
4-FullIns_4	6.02	6.02	6.02	1979.86	88.40	283.54	16.15
4-FullIns_5	-	-	-	-	-	-	14.11
5-FullIns_3	7.01	7.01	7.00	61.22	2.72	12.48	0.71
5-FullIns_4	-	7.01	7.01	-	207.83	137.49	21.11
wap01a	-	41.00	40.38	-	309.86	3575.61	20.34
wap02a	40.00	40.00	-	538.62	473.42	-	3.29
wap03a	40.00	40.00	40.00	1594.69	2668.31	1507.80	10.12
wap04a	40.00	40.00	40.00	2175.13	2658.54	2179.94	13.84
wap05a	50.00	50.00	50.00	1099.11	24.19	918.93	11.72
wap06a	40.00	40.00	-	63.35	69.47	-	0.74
wap07a	40.00	40.00	40.00	309.93	426.97	145.89	3.00
wap08a	40.00	40.00	-	278.67	224.35	-	2.26
qg.order30	30.00	30.00	-	32.22	21.30	-	0.30
qg.order40	40.00	40.00	-	153.25	82.68	-	1.08
qg.order60	60.00	60.00	-	1684.60	496.86	-	9.74

Table 4: Results on $\bar{\vartheta}_+(G)$, graphs from the second DIMACS implementation challenge.

	bounds on $\chi(G)$			CPU times			
Graph	ADAL	SDPNAL+	BestBound	ADAL	SDPNAL+	BestBound	post-proc
$\bar{\vartheta}(G)$ + 1000 inequalities from [18]							
DSJC500.5	22.90	22.90	22.90	15.97	38.73	13.77	0.54
DSJC1000.1	8.36	8.36	8.36	34.43	154.79	25.85	0.56
DSJC1000.5	32.11	32.11	32.11	33.33	154.82	34.50	1.18
myciel7	2.85	2.85	2.85	255.84	13.77	13.39	6.77
mug88_25	3.00	3.00	-	17.81	33.92	-	0.25
mug100_25	3.00	3.00	-	22.33	97.57	-	0.28
abb313GPIA	8.00	8.00	8.00	661.22	3466.57	178.77	4.49
1-Insertions_6	2.33	2.33	2.33	1001.29	233.15	65.62	8.98
2-Insertions_5	2.18	2.18	2.18	850.44	373.95	100.30	7.21
3-Insertions_5	-	-	2.11	-	-	415.32	18.32
4-Insertions_4	2.07	2.07	2.07	632.60	363.73	87.32	6.53
1-FullIns_5	3.19	3.19	3.19	1955.97	141.02	59.14	35.67
5-FullIns_4	-	7.01	7.01	-	257.83	132.12	21.35
wap03a	40.00	-	40.00	1629.48	-	1496.25	11.16
wap04a	40.00	-	40.00	2297.75	-	2302.30	13.66
$\bar{\vartheta}(G)$ + 2500 inequalities from [18]							
DSJC500.5	22.90	22.90	22.90	79.62	43.81	79.92	0.60
DSJC1000.1	8.36	8.36	8.36	38.74	167.61	29.44	0.57
DSJC1000.5	32.11	32.11	32.11	100.35	157.33	91.06	2.39
myciel7	2.87	2.87	2.87	341.20	13.80	63.14	3.51
mug88_25	3.00	3.00	3.00	83.45	81.22	36.93	1.22
mug100_25	3.00	3.00	3.00	91.18	97.93	85.68	1.43
abb313GPIA	8.00	8.00	8.00	685.60	2611.71	250.42	4.60
1-Insertions_6	2.34	2.34	2.34	827.90	553.82	111.23	6.91
2-Insertions_5	2.19	2.19	2.19	790.06	576.61	157.16	5.79
3-Insertions_5	-	2.13	2.12	-	-	1133.88	18.54
4-Insertions_4	2.08	2.08	2.08	625.78	537.75	143.96	4.82
1-FullIns_5	-	3.19	3.19	-	145.95	270.64	34.75
5-FullIns_4	-	7.01	7.01	-	308.79	149.04	20.10
wap03a	40.00	-	40.00	1932.11	-	1935.99	10.83
wap04a	40.00	-	40.00	2629.88	-	2011.30	14.42
$\bar{\vartheta}(G)$ + 5000 inequalities from [18]							
DSJC500.5	22.90	22.90	22.90	464.82	46.24	456.20	1.14
DSJC1000.1	8.36	8.36	8.36	63.50	163.05	48.59	0.70
DSJC1000.5	32.11	32.11	32.11	589.69	168.79	590.91	2.12
myciel7	2.89	2.89	2.89	583.55	15.44	132.16	3.90
mug88_25	3.00	3.00	3.00	199.32	116.73	78.35	1.56
mug100_25	3.00	3.00	3.00	216.21	154.84	190.52	1.39
abb313GPIA	8.00	-	8.00	744.60	-	356.07	4.36
1-Insertions_6	2.36	2.36	2.36	1351.80	258.02	218.89	8.25
2-Insertions_5	2.19	2.19	2.19	904.85	572.94	274.81	5.39
3-Insertions_5	-	-	2.12	-	-	1737.21	15.49
4-Insertions_4	2.09	2.09	2.09	736.69	831.07	315.23	4.01
1-FullIns_5	-	3.19	3.19	-	179.80	554.77	22.29
5-FullIns_4	-	7.01	7.01	-	344.28	293.47	17.54
wap03a	40.00	-	40.00	2591.76	-	2383.76	15.00
wap04a	-	-	40.00	-	-	3129.61	18.72

Table 5: Results on bounds on $\chi(\bar{G})$.

5 Conclusions

We propose a numerical comparison between `ADAL`, an ADMM method for SDPs in general form and `SDPNAL+`, the state-of-the-art method for solving large-scale SDPs that has been awarded with the Beale-Orchard-Hays Prize in 2018. We consider random instances as well as instances from the SDP relaxations of the graph coloring problem and the maximum clique problem. Despite `SDPNAL+` is more robust than `ADAL`, we are generally faster on the random instances considered and we are able to detect valid bounds on the chromatic number and the clique number in a much lower amount of time on some classes of graphs. The post-processing procedure used is developed to obtain a dual feasible solution which in turn gives a bound on the optimal primal value. From a practical point of view, as long as we use SDPs to address combinatorial optimization problems, the post-processing procedure allows to stop the execution of the ADMM as soon as a "good" bound is obtained, even if the convergence criterion is far to be met. Furthermore, the fact that a dual feasible solution is detected, allows to use reoptimization techniques within branch-and-bound frameworks and this is what we plan to focus on in the near future.

6 Acknowledgements

The authors acknowledge support within the project RM120172A2970290 which has received funding from Sapienza, University of Rome. They are also indebted to Fabrizio Rossi and Stefano Smriglio for their useful suggestions.

References

[1] L. Yang, D. Sun, K.-C. Toh, Sdpnal+: a majorized semismooth newton-cg augmented lagrangian method for semidefinite programming with nonnegative constraints, Mathematical Programming Computation 7 (3) (2015) 331–366.

[2] M. Laurent, F. Rendl, Semidefinite programming and integer programming, in: K. Aardal, G. Nemhauser, R. Weismantel (Eds.), Discrete Optimization, Vol. 12 of Handbooks in Operations Research and Management Science, Elsevier, Amsterdam, The Netherlands, 2005, Ch. 8, pp. 393–514.

[3] S. Burer, R. D. C. Monteiro, A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization, Math. Program. 95 (2, Ser. B) (2003) 329–357.

[4] S. Burer, R. D. C. Monteiro, Local minima and convergence in low-rank semidefinite programming, Math. Program. 103 (3, Ser. A) (2005) 427–444.

[5] J. Povh, F. Rendl, A. Wiegele, A Boundary Point Method to solve Semidefinite Programs, Computing 78 (2006) 277–286.

[6] J. Malick, J. Povh, F. Rendl, A. Wiegele, Regularization methods for semidefinite programming, SIAM J. Optim. 20 (1) (2009) 336–356.

[7] D. Sun, K.-C. Toh, L. Yang, A convergent 3-block semiproximal alternating direction method of multipliers for conic programming with 4-type constraints, SIAM Journal on Optimization 25 (2015) 882–915.

[8] Z. Wen, D. Goldfarb, W. Yin, Alternating direction augmented Lagrangian methods for semidefinite programming, Mathematical Programming Computation 2 (3) (2010) 203–230.

[9] M. De Santis, F. Rendl, A. Wiegele, Using a factored dual in augmented Lagrangian methods for semidefinite programming, Operations Research Letters 46 (5) (2018) 523 – 528.

[10] C. Jansson, D. Chaykin, C. Keil, Rigorous error bounds for the optimal value in semidefinite programming, SIAM Journal on Numerical Analysis 46 (1) (2007/08) 180–200. doi:10.1137/050622870.
URL https://doi.org/10.1137/050622870

[11] M. Cerulli, M. De Santis, E. Gaar, A. Wiegele, Improving admms for solving doubly nonnegative programs through dual factorization, 4OR (2020) 1–34.

[12] A. Wiegele, S. Zhao, Sdp-based bounds for graph partition via extended admm, arXiv preprint arXiv:2105.09075 (2021).

[13] D. A. Lorenz, Q. Tran-Dinh, Non-stationary Douglas–Rachford and alternating direction method of multipliers: adaptive step-sizes and convergence, Computational Optimization and Applications 74 (1) (2019) 67–92. doi:10.1007/s10589-019-00106-9.
URL https://doi.org/10.1007/s10589-019-00106-9

[14] F. Rendl, Matrix relaxations in combinatorial optimization, in: J. Lee, S. Leyffer (Eds.), Mixed Integer Nonlinear Programming, Springer New York, 2012, pp. 483–511.

[15] L. Gurobi Optimization, Gurobi optimizer reference manual (2021).
URL http://www.gurobi.com

[16] L. Lovász, On the shannon capacity of a graph, IEEE Transactions on Information theory 25 (1) (1979) 1–7.

[17] M. Grötschel, L. Lovász, A. Schrijver, Geometric algorithms and combinatorial optimization, Vol. 2, Springer Science & Business Media, 2012.

[18] I. Dukanovic, F. Rendl, A semidefinite programming-based heuristic for graph coloring, Discrete Applied Mathematics 156 (2) (2008) 180–189.

[19] G. Gruber, F. Rendl, Computational experience with stable set relaxations, SIAM Journal on Optimization 13 (4) (2003) 1014–1028.

[20] M. Giandomenico, A. N. Letchford, F. Rossi, S. Smriglio, An application of the lovász–schrijver m (k, k) operator to the stable set problem, Mathematical programming 120 (2) (2009) 381–401.

[21] M. Giandomenico, F. Rossi, S. Smriglio, Strong lift-and-project cutting planes for the stable set problem, Mathematical Programming 141 (1) (2013) 165–192.

[22] M. Giandomenico, A. N. Letchford, F. Rossi, S. Smriglio, Ellipsoidal relaxations of the stable set problem: theory and algorithms, SIAM Journal on Optimization 25 (3) (2015) 1944–1963.

[23] M. Locatelli, Improving upper bounds for the clique number by non-valid inequalities, Mathematical Programming 150 (2) (2015) 511–525.

[24] E. Gaar, F. Rendl, A bundle approach for SDPs with exact subgraph constraints, in: A. Lodi, V. Nagarajan (Eds.), Integer Programming and Combinatorial Optimization, Springer International Publishing, 2019, pp. 205–218.

[25] C. Chen, B. He, Y. Ye, X. Yuan, The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent, Mathematical Programming 155 (1-2) (2016) 57–79.

[26] E. Dolan, J.Moré, Benchmarking optimization software with performance profiles, Mathematical Programming 91 (2002) 201–213.

[27] D. J. Johnson, M. A. Trick (Eds.), Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993, American Mathematical Society, 1996.