



An SDP-based approach for computing the stability number of a graph

Elisabeth Gaar¹ · Melanie Siebenhofer² · Angelika Wiegele²

Received: 12 August 2021 / Revised: 25 January 2022 / Accepted: 27 January 2022 /
Published online: 12 March 2022
© The Author(s) 2022

Abstract

Finding the stability number of a graph, i.e., the maximum number of vertices of which no two are adjacent, is a well known NP-hard combinatorial optimization problem. Since this problem has several applications in real life, there is need to find efficient algorithms to solve this problem. Recently, Gaar and Rendl enhanced semidefinite programming approaches to tighten the upper bound given by the Lovász theta function. This is done by carefully selecting some so-called exact subgraph constraints (ESC) and adding them to the semidefinite program of computing the Lovász theta function. First, we provide two new relaxations that allow to compute the bounds faster without substantial loss of the quality of the bounds. One of these two relaxations is based on including violated facets of the polytope representing the ESCs, the other one adds separating hyperplanes for that polytope. Furthermore, we implement a branch and bound (B&B) algorithm using these tightened relaxations in our bounding routine. We compare the efficiency of our B&B algorithm using the different upper bounds. It turns out that already the bounds of Gaar and Rendl drastically reduce the number of nodes to be explored in the B&B tree as compared to the Lovász theta bound. However, this comes with a high computational cost. Our new relaxations improve the run time of the overall B&B algorithm, while keeping the number of nodes in the B&B tree small.

Keywords Stable set · Semidefinite programming · Lovász theta function · Branch and bound · Combinatorial optimization

✉ Angelika Wiegele
angelika.wiegele@aau.at

Elisabeth Gaar
elisabeth.gaar@jku.at

Melanie Siebenhofer
melanie.siebenhofer@aau.at

¹ Institute of Production and Logistics Management, Johannes Kepler University Linz, Altenberger StraÙ 69, 4040 Linz, Austria

² Institut für Mathematik, Alpen-Adria-Universität Klagenfurt, Universitätsstraße 65-67, 9020 Klagenfurt, Austria

1 Introduction

The stable set problem is a fundamental combinatorial optimization problem. It is capable of modeling other combinatorial problems as well as real-world applications and is therefore widely applied in areas like operations research or computer science. We refer to the survey (Wu and Hao 2015) for more information and a review of exact and heuristic algorithms. Most of the exact algorithms are based on branch and bound (B&B) and differ mainly by different upper and lower bound computations. A recent paper using a MIP solver is e.g. (Letchford et al. 2020). The models used in that paper yield computation times from less than a second up to half an hour on a selection of DIMACS instances.

Outstanding results are obtained by an algorithm of Depolli et al. (2013). They introduced an algorithm using parallel computing for finding maximum cliques in the context of protein design. The algorithm consists of carefully implemented algorithmic building blocks such as an approximate coloring algorithm, an initial vertex ordering algorithm and the use of bit-strings for encoding the adjacency matrix.

In the 2015 survey (Wu and Hao 2015), no exact algorithms using semidefinite programming (SDP) are mentioned. One reason for the rare literature on SDP based B&B algorithms is the high computational cost for computing these bounds. In this work we introduce an SDP based B&B algorithm. We formulate new SDP relaxations and develop solution algorithms to compute these bounds with moderate computational expense, making them applicable within a B&B scheme.

Before introducing the stable set problem, sometimes also referred to as vertex packing problem, we give the definition of a stable set. Let $G = (V, E)$ be a simple undirected graph with $|V| = n$ vertices and $|E| = m$ edges. A set $S \subseteq V$ is called stable if no vertices in S are adjacent. S is called a maximal stable set if it is not possible to add a vertex to S without losing the stability property. The stability number $\alpha(G)$ denotes the maximum size of a stable set in G , where size means the cardinality of the set. A stable set S is called a maximum stable set if it has size $\alpha(G)$.

For convenience, from now on we always label the vertices of a graph with n vertices from 1 to n . Computing $\alpha(G)$ can be done by solving the following optimization problem.

$$\begin{aligned} \alpha(G) = \max \quad & \sum_{i=1}^n x_i & (1) \\ \text{s. t. } & x_i + x_j \leq 1 \quad \forall \{i, j\} \in E(G) \\ & x \in \{0, 1\}^n \end{aligned}$$

For a graph $G = (V, E)$, the set of all stable set vectors $\mathcal{S}(G)$ and the stable set polytope $\text{STAB}(G)$ are defined as

$$\mathcal{S}(G) = \{s \in \{0, 1\}^n : s_i s_j = 0 \quad \forall \{i, j\} \in E\} \text{ and}$$

$$\text{STAB}(G) = \text{conv} \{s : s \in \mathcal{S}(G)\}.$$

Determining $\alpha(G)$ is NP-complete and the decision problem is among Karp’s 21 NP-complete problems (Karp 1972). Furthermore, Håstad (1999) proved that $\alpha(G)$ is not approximable within $n^{1-\varepsilon}$ for any $\varepsilon > 0$ unless $P=NP$. A well known upper bound on $\alpha(G)$ is the Lovász theta function $\vartheta(G)$. Grötschel et al. (1988) proved that

$$\begin{aligned} \vartheta(G) = \max \quad & \mathbb{1}_n^T x & (2) \\ \text{s. t. } \text{diag}(X) = x & \\ & X_{i,j} = 0 \quad \forall \{i, j\} \in E \\ & \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succcurlyeq 0 \\ & X \in \mathcal{S}_n, \quad x \in \mathbb{R}^n \end{aligned}$$

and hence provided a semidefinite program (SDP) to compute $\vartheta(G)$. We define the feasible region of (2) as

$$\begin{aligned} \text{TH}^2(G) = \{ & (x, X) \in \mathbb{R}^n \times \mathcal{S}_n : \text{diag}(X) = x, \\ & X_{i,j} = 0 \quad \forall \{i, j\} \in E, \quad X - xx^T \succcurlyeq 0 \}. \end{aligned}$$

Clearly for each element (x, X) of $\text{TH}^2(G)$ the projection of X onto its main diagonal is x . The set of all projections

$$\text{TH}(G) = \left\{ x \in \mathbb{R}^n : \exists X \in \mathcal{S}_n : (x, X) \in \text{TH}^2(G) \right\}$$

is called theta body. More information on $\text{TH}(G)$ can be found for example in Conforti et al. (2014). It is easy to see that $\text{STAB}(G) \subseteq \text{TH}(G)$ holds for every graph G , see (Grötschel et al. 1988). Thus $\vartheta(G)$ is a relaxation of $\alpha(G)$.

This paper is structured as follows. In Sect. 2 we introduce two new relaxations using the concept of exact subgraph constraints. A branch and bound algorithm that uses these relaxations is described in Sect. 3, followed by the discussion of numerical results in Sect. 4. Sect. 5 concludes this paper.

2 New relaxations of the exact subgraph constraints

In this section we present two new approaches to find upper bounds on the stability number $\alpha(G)$ of a graph G starting from the Lovász theta function $\vartheta(G)$ with so-called exact subgraph constraints, one based on violated facets and one based on separating hyperplanes. After introducing these approaches, we compare them both theoretically and practically.

2.1 Basic setup for exact subgraph constraints

Our approach is based on the idea of exact subgraph constraints that goes back to Adams, Anjos, Rendl and Wiegele (Adams et al. 2015) for combinatorial optimization problems that have an SDP relaxation and was recently computationally investigated by Gaar and Rendl (2019, 2020) for the stable set, the Max-Cut and the coloring problem as a basis. Starting from this, we present two relaxations of including exact subgraph constraints into the SDP for calculating $\vartheta(G)$ that are computationally more efficient.

We first recapitulate the basic concepts of exact subgraph constraints with the notation from Gaar (2020). An upper bound on $\alpha(G)$ is given by the Lovász theta function $\vartheta(G)$. Due to the SDP formulation (2) it can be computed in polynomial time. Adams, Anjos, Rendl and Wiegele (Adams et al. 2015) proposed to improve $\vartheta(G)$ as an upper bound by adding so-called exact subgraph constraints. These exact subgraph constraints can be used to strengthen SDP relaxations of combinatorial optimization problems with a certain property by including subgraph information. For the stable set problem we need the following definitions in order to introduce the exact subgraph constraints. For a graph G the squared stable set polytope $\text{STAB}^2(G)$ is defined as

$$\text{STAB}^2(G) = \text{conv} \left\{ ss^T : s \in \mathcal{S}(G) \right\}$$

and matrices of the form ss^T for $s \in \mathcal{S}(G)$ are called stable set matrices. Let G_I denote the subgraph induced by the vertex set $I \subseteq V(G)$ with $|I| = k_I$. With X_I we denote the submatrix of X that results when we delete each row and column corresponding to a vertex that is not in I . In other words, X_I is the submatrix of X where we only choose the rows and columns corresponding to the vertices in I . Then the constraint that asks the submatrix X_I of (2) for an induced subgraph G_I to be in the squared stable set polytope $\text{STAB}^2(G_I)$ is called exact subgraph constraint (ESC).

The k -th level of the exact subgraph hierarchy introduced in Adams et al. (2015) is the Lovász theta function (2) with additional ESC for each subgraph of order k . In Gaar and Rendl (2019, 2020) this hierarchy is exploited computationally by including the ESC only for a set J of subgraphs and then considering

$$z_J(G) = \max \left\{ \mathbb{1}_n^T x : (x, X) \in \text{TH}^2(G), X_I \in \text{STAB}^2(G_I) \quad \forall I \in J \right\}. \quad (3)$$

Clearly, $\alpha(G) \leq z_J(G)$ holds for every set J of subsets of $V(G)$, so $z_J(G)$ is an upper bound on $\alpha(G)$. One of the key remaining questions is how to solve (3). We will compare different implementations and relaxations of this problem in the rest of the paper and start by considering existing methods.

The most straightforward way to solve (3) is to include the ESCs in a convex hull formulation as presented in Gaar and Rendl (2019, 2020); Gaar (2020). We now recall the basic features and follow the presentation from Gaar (2020). As the ESC for a subgraph G_I makes sure that $X_I \in \text{STAB}^2(G_I)$ holds and the polytope $\text{STAB}^2(G_I)$ is defined as the convex hull of the stable set matrices, the most intuitive way to formulate the ESC is as a convex combination. Towards that end, for a subgraph G_I

of G induced by the subset $I \subseteq V$, let $|\mathcal{S}(G_I)| = t_I$ and let $\mathcal{S}(G_I) = \{s_1^I, \dots, s_{t_I}^I\}$. Then the i -th stable set matrix S_i^I of the subgraph G_I is defined as $S_i^I = s_i^I (s_i^I)^T$. As a result, the ESC $X_I \in \text{STAB}^2(G_I)$ can be rewritten as

$$X_I \in \text{conv} \left\{ S_i^I : 1 \leq i \leq t_I \right\}$$

and it is natural to implement the ESC for the subgraph G_I as

$$X_I = \sum_{i=1}^{t_I} [\lambda_I]_i S_i^I, \quad \lambda_I \in \Delta_{t_I},$$

where Δ_{t_I} is the t_I -dimensional simplex.

This means that when including the ESC for the subgraph G_I into (2) we have t_I additional non-negative variables, one additional linear equality constraint for λ_I and the matrix equality constraint which couples X_I and λ_I . We denote the number of equality constraints that are induced by the matrix equality constraint by b_I and note that $b_I \leq \binom{k_I+1}{2}$ holds. With this formulation (3) can equivalently be written as

$$z_J^C(G) = \max \left\{ \mathbb{1}_n^T x : (x, X) \in \text{TH}^2(G), X_I = \sum_{i=1}^{t_I} [\lambda_I]_i S_i^I, \lambda_I \in \Delta_{t_I} \quad \forall I \in J \right\}, \tag{4}$$

so $z_J(G) = z_J^C(G)$ holds. In practice, this SDP can be solved by interior point methods as long as the number of ESC constraints is of moderate size.

Due to the fact that (4) becomes a huge SDP as soon as the number of ESCs $|J|$ becomes large, Gaar and Rendl (2019, 2020) proposed to use the bundle method to solve this SDP. The bundle method is an iterative procedure to find a global minimum of a non-smooth convex function and has been adapted for SDPs by Helmberg and Rendl (2000). As we use the bundle method only as a tool and do not enhance it any further, we refrain from presenting details here.

2.2 Relaxation based on inequalities that represent violated facets

We will see later on that the computational costs of a B&B algorithm are enormous in the original version with the convex hull formulation (4) and they are still substantial with the bundle approach from Gaar and Rendl (2019, 2020). Therefore, we suggest two alternatives.

First, we present a relaxation of calculating the Lovász theta function with ESCs (3) that has already been mentioned in Gaar (2020), but has never been computationally exploited so far. The key ingredient for this relaxation is the following observation. The polytope $\text{STAB}^2(G_I)$ is given by its extreme points, which are the stable set matrices of G_I . Due to Weyl's theorem (see for example (Nemhauser and Wolsey 1988)) it can also be represented by its facets. This means that there are (finitely many) inequalities, such that the constraint $X_I \in \text{STAB}^2(G_I)$ can be represented by these inequalities.

However, the facets and hence the inequalities depend on the stable set matrices and therefore on the subgraph G_I . Thus different subgraphs need different calculations that will lead to different inequalities. Gaar (2020)[Lemma 3] showed that adding the ESC $X_I \in \text{STAB}^2(G_I)$ to the SDP calculating the Lovász theta function (2) is equivalent to adding the constraint $X_I \in \text{STAB}^2(G_{k_I}^0)$ where $G_{k_I}^0 = (V_{k_I}^0, E^0)$ with $V_{k_I}^0 = \{1, \dots, k_I\}$ and $E^0 = \emptyset$.

This implies that it is enough to calculate the facets of $\text{STAB}^2(G_{k_I}^0)$ and include these facets for each subgraph G_I on k_I vertices, instead of calculating the facets of $\text{STAB}^2(G_I)$ for each subgraph G_I separately. Let r_{k_I} be the number of facets of $\text{STAB}^2(G_{k_I}^0)$ and let $F_i^{k_I} \in \mathbb{R}^{k_I \times k_I}, f_i^{k_I} \in \mathbb{R}$ for $1 \leq i \leq r_{k_I}$ such that

$$\text{STAB}^2(G_{k_I}^0) = \left\{ X \in \mathbb{R}^{k_I \times k_I} : \left\langle F_i^{k_I}, X \right\rangle \leq f_i^{k_I} \quad \forall 1 \leq i \leq r_{k_I} \right\},$$

so $(F_i^{k_I}, f_i^{k_I})$ is an inequality representing the i -th facet of $\text{STAB}^2(G_{k_I}^0)$. We obtained $r_{k_I}, F_i^{k_I}$ and $f_i^{k_I}$ for $k_I \leq 6$ in the way suggested in Gaar (2020). For $k_I \geq 7$ this computation is beyond reach, as r_7 is conjectured to be 217093472 (Christof (2020)).

If we would include all facets of $\text{STAB}^2(G_{k_I}^0)$ for each subgraph G_I to replace the ESCs in (3), then we would include a huge number of inequalities ($r_5 = 368$ and $r_6 = 116764$) and reach the limits of computing power rather soon. In order to reduce the number of inequalities, for each subgraph we include only those inequalities that represent facets that are violated by the current solution X^* . To be more precise, let X^* be the optimal solution of (3) for $J = \emptyset$, i.e., the optimal solution of calculating the Lovász theta function. Then we define the indices of significantly violated facets of G_I , i.e., facets where the corresponding inequalities are violated at least by ε_F , as

$$\mathcal{V}'_I = \left\{ 1 \leq i \leq r_{k_I} : \left\langle F_i^{k_I}, X^* \right\rangle > f_i^{k_I} + \varepsilon_F \right\},$$

where ε_F is a small constant to take care of numerical inaccuracies of calculating X^* .

Now we can further reduce the number of included inequalities in the following way. Although all $(F_i^{k_I}, f_i^{k_I})$ are different for different values of i , it could happen that for a subgraph G_I there exist $1 \leq i \neq i' \leq r_{k_I}$ such that $(F_i^{k_I}, f_i^{k_I})$ and $(F_{i'}^{k_I}, f_{i'}^{k_I})$ induce the same inequality. This is possible because they might differ only in positions (j, j') with $j, j' \in I$ and $\{j, j'\} \in E$. Therefore, these different entries are multiplied with zero due to $[X^*]_{j,j'} = 0$. Hence, let $\mathcal{V}_I \subseteq \mathcal{V}'_I$ be a set such that only one index among all indices in \mathcal{V}'_I which induce the same inequality is in \mathcal{V}_I . Then we obtain the following relaxation of (3), in which we include only inequalities that induce significantly violated facets of G_I

$$z_J^F(G) = \max \left\{ \mathbb{1}_n^T x : (x, X) \in \text{TH}^2(G), \left\langle F_i^{k_I}, X_I \right\rangle \leq f_i^{k_I} \quad \forall i \in \mathcal{V}_I \quad \forall I \in J \right\}. \tag{5}$$

Unfortunately for $k_I \geq 7$ it is not possible to store and check the facets of $\text{STAB}^2(G_{k_I}^0)$ for violation in reasonable memory and time due to the huge number of facets. Hence, we can perform this relaxation only for subgraphs G_I of order $k_I \leq 6$.

2.3 Relaxation based on separating hyperplanes

Next we consider another approach to implement a relaxation of (3) which can also be used for subgraphs G_I of order $k_I \geq 7$ and which is based on including separating hyperplanes.

It uses the following fact. Let \tilde{X} be any matrix in $\in \mathcal{S}_n$ and let P_I be the projection of \tilde{X}_I onto $\text{STAB}^2(G_I)$. Then we can calculate the projection distance of \tilde{X} to $\text{STAB}^2(G_I)$ as

$$\begin{aligned} \|P_I - \tilde{X}_I\|_F^2 &= \min_{\lambda_I \in \Delta_{t_I}} \left\| \left(\sum_{i=1}^{t_I} [\lambda_I]_i S_i^I \right) - \tilde{X}_I \right\|_F^2 = \min_{\lambda_I \in \Delta_{t_I}} \left\| \sum_{i=1}^{t_I} [\lambda_I]_i (S_i^I - \tilde{X}_I) \right\|_F^2 \\ &= \min_{\lambda_I \in \Delta_{t_I}} \sum_{j=1}^{k_I} \sum_{j'=1}^{k_I} \left(\sum_{i=1}^{t_I} [\lambda_I]_i [S_i^I - \tilde{X}_I]_{j,j'} \right)^2 \\ &= \min_{\lambda_I \in \Delta_{t_I}} \sum_{j=1}^{k_I} \sum_{j'=1}^{k_I} \left(\sum_{i=1}^{t_I} \sum_{i'=1}^{t_I} [\lambda_I]_i [\lambda_I]_{i'} [S_i^I - \tilde{X}_I]_{j,j'} [S_{i'}^I - \tilde{X}_I]_{j,j'} \right) \\ &= \min_{\lambda_I \in \Delta_{t_I}} \sum_{i=1}^{t_I} \sum_{i'=1}^{t_I} [\lambda_I]_i [\lambda_I]_{i'} \left(\sum_{j=1}^{k_I} \sum_{j'=1}^{k_I} [S_i^I - \tilde{X}_I]_{j,j'} [S_{i'}^I - \tilde{X}_I]_{j,j'} \right) \\ &= \min_{\lambda_I \in \Delta_{t_I}} \lambda_I^T Q_I \lambda_I, \end{aligned} \tag{6}$$

where $Q_I \in \mathbb{R}^{t_I \times t_I}$ and $[Q_I]_{i,i'} = \langle S_i^I - \tilde{X}_I, S_{i'}^I - \tilde{X}_I \rangle$. Q_I is symmetric and positive semidefinite because it is a Gram matrix, so (6) is a convex-quadratic program with t_I variables, a convex-quadratic objective function and one linear equality constraint. With the optimal solution λ_I of (6) the projection of \tilde{X}_I onto $\text{STAB}^2(G_I)$ can be obtained by $P_I = \sum_{i=1}^{t_I} [\lambda_I]_i S_i^I$. By defining

$$H_I = \frac{1}{\|\tilde{X}_I - P_I\|_F} (\tilde{X}_I - P_I) \quad \text{and} \quad h_I = \frac{1}{\|\tilde{X}_I - P_I\|_F} \langle \tilde{X}_I - P_I, P_I \rangle$$

due to the separating hyperplane theorem (see for example Boyd and Vandenberghe 2004)

$$\langle H_I, X_I \rangle \leq h_I \tag{7}$$

is a hyperplane that separates \tilde{X}_I from $\text{STAB}^2(G_I)$ such that $X_I = P_I$ fulfills the inequality with equality. Obviously (7) is a relaxation of the ESC $X_I \in \text{STAB}^2(G_I)$, so

$$z_J^H(G) = \max \left\{ \mathbb{1}_n^T x : (x, X) \in \text{TH}^2(G), \langle H_I, X_I \rangle \leq h_I \quad \forall I \in J \right\} \tag{8}$$

is another relaxation of (3) that depends on the chosen \tilde{X} .

2.4 Theoretical comparison of the relaxations

We briefly comment on some theoretical properties of $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$. We start by analyzing the upper bounds we obtain. Due to the fact that $z_J^F(G)$ and $z_J^H(G)$ are relaxations of $z_J^C(G)$, we know that

$$\alpha(G) \leq z_J^C(G) \leq z_J^F(G), z_J^H(G) \leq \vartheta(G)$$

holds for every graph G and every set J .

Another important observation is the following. Whenever we include the ESC of the subgraph G_I into the SDP computing $z_J^C(G)$, the stable set problem is solved exactly on this subgraph G_I . However, when computing $z_J^F(G)$ and $z_J^H(G)$ we do not include the ESC but only a relaxed version of it. Hence, in the optimal solutions of these two relaxations, it could still be the case that the ESC is not fulfilled, i.e., for the subgraph G_I we do not have an exact solution. Hence, it is possible that we still find violated inequalities (representing facets or hyperplanes) in these cases. As a consequence, for $z_J^C(G)$ it does not make sense to include the ESC for the same subgraph twice, but for $z_J^F(G)$ and $z_J^H(G)$ it is possible that we want to include a relaxation of the very same ESC twice with different facets or a different separating hyperplane.

Finally let us consider the sizes of the SDPs to solve. In all three versions $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ we solve the SDP of the Lovász theta function (2) with additional constraints, so in all three SDPs we have a matrix variable of dimension $n + 1$ which has to be positive semidefinite (psd) and $n + m + 1$ linear equality constraints. Additionally to that we have $\sum_{I \in J} t_I$ non-negative variables and $|J| + \sum_{I \in J} b_I$ equality constraints for $z_J^C(G)$, $\sum_{I \in J} |\mathcal{V}_I|$ inequalities for $z_J^F(G)$, and $|J|$ inequalities for $z_J^H(G)$. Table 1 gives an overview of the different sizes of the SDPs.

Table 1 Sizes of the SDPs to compute $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$

	$z_J^C(G)$	$z_J^F(G)$	$z_J^H(G)$
Dimension psd matrix variable	$n + 1$	$n + 1$	$n + 1$
# Non-negative variables	$\sum_{I \in J} t_I$	0	0
# Linear equality constraints	$n + m + 1 + J + \sum_{I \in J} b_I$	$n + m + 1$	$n + m + 1$
# Linear inequality constraints	0	$\sum_{I \in J} V_I $	$ J $

Table 2 The values of $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ for $G = G_{100,15}$ for different sets J_q

	J_{221}	J_{443}	J_{664}	J_{886}	J_{1107}
$z_J^C(G)$	26.9905	26.9299	26.8684	26.8496	26.8278
$z_J^F(G)$	26.9975	26.9393	26.8807	26.8602	26.8397
$z_J^H(G)$	27.0104	26.9741	26.9215	26.8992	26.8898

2.5 Computational comparison of the relaxations

Before we perform a large scale comparison of $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ within a B&B algorithm, we investigate a small example.

Example 1 We consider a random graph $G = G_{100,15}$ from the Erdős-Rényi model $G(n, p)$ with $n = 100$ and $p = 0.15$. A random graph from this model has n vertices and every edge is present with probability p independently from all other edges. For the chosen graph, $\vartheta(G_{100,15}) = 27.2003$ and $\alpha(G_{100,15}) = 24$ holds, so

$$24 \leq z_J^C(G) \leq z_J^F(G), z_J^H(G) \leq 27.2003$$

holds for every set J .

All the computations were performed on an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz with 32 GB RAM with the MATLAB version R2016b and with MOSEK version 8. In the computations, we use $\epsilon_F = 0.00005$ and we include a separating hyperplane for a subgraph whenever the projection distance is greater or equal to 0.00005.

We compute $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ for different sets J , which all consist of subsets of vertices of size $k_I = 5$ and only differ in the number $q = |J|$ of included ESCs. To be more precise, we consider five different sets $J = J_q$ with $q = |J_q| \in \{221, 443, 664, 886, 1107\}$. These values of q are chosen in such a way that the number of linear equality constraints which are induced by the matrix equalities from the ESCs in the convex hull formulation, i.e., $\sum_{I \in J_q} b_I$, is in $\{3000, 6000, 9000, 12000, 15000\}$. To choose the subsets in J_q , we first determine X^* as the optimal solution of (3) for $J = \emptyset$, i.e., the optimal solution of calculating the Lovász theta function (2). Then we generated $3q$ subgraphs G_I of order k_I randomly and included those q subsets I into J_q , where the corresponding X_I^* have the largest projection distances to $\text{STAB}^2(G_I)$. For computing $z_J^H(G)$ we choose $\tilde{X} = X^*$.

Table 3 The running times in seconds for computing $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ of Table 2

	J_{221}	J_{443}	J_{664}	J_{886}	J_{1107}
$z_J^C(G)$	8.14	29.30	66.95	139.64	279.11
$z_J^F(G)$	0.61	1.15	1.94	2.72	3.81
$z_J^H(G)$	0.75	1.25	1.93	2.57	3.34

Table 4 The average projection distances of X_I to $\text{STAB}^2(G_I)$ over all $I \in J_q$ before (i.e., $X = X^*$ is the optimal solution of (2)), and after (i.e., $X \in \{X^{C^*}, X^{F^*}, X^{H^*}\}$ is the optimal solution of $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$) including the ESCs

	J_{221}	J_{443}	J_{664}	J_{886}	J_{1107}
Before including ESCs	0.03095	0.03014	0.03057	0.02982	0.03032
After computing $z_J^C(G)$	0.00005	0.00004	0.00004	0.00004	0.00004
After computing $z_J^F(G)$	0.00151	0.00087	0.00115	0.00080	0.00051
After computing $z_J^H(G)$	0.00290	0.00256	0.00252	0.00196	0.00183

If we consider Table 2 with the improved upper bounds then we see that if q increases, all upper bounds $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ improve. Furthermore, one can observe that for a fixed set J_q the obtained bounds of $z_J^C(G)$ are best, those of $z_J^F(G)$ are a little bit worse and those of $z_J^H(G)$ are even a little bit more worse, i.e., empirically the bounds obtained by using $z_J^F(G)$ are better than those coming from $z_J^H(G)$ in our example.

Next we consider the running times for computing $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$ in Table 3. Here we see that the time it takes to solve $z_J^C(G)$ is extremely high and increases drastically if the number of included ESCs gets larger. Both our relaxations $z_J^F(G)$ and $z_J^H(G)$ reduce the running times enormously. The running times for $z_J^F(G)$ and $z_J^H(G)$ are comparable, but computing $z_J^H(G)$ is slightly faster for including a large number of ESCs as we only include one additional inequality in $z_J^H(G)$ whereas we may include several inequalities that represent facets in $z_J^F(G)$.

As a next step we investigate the projection distances. Recall that X^* is the optimal solution of calculating the Lovász theta function (2). Let X^{C^*} , X^{F^*} and X^{H^*} be the optimal solution of calculating $z_J^C(G)$, $z_J^F(G)$ and $z_J^H(G)$, respectively. In Table 4 we see that the average projection distance of X^* is significantly larger than 0 before including the ESCs, so there are several violated ESCs. As soon as the ESCs are included the average projection distance for X^{C^*} is almost zero, so the ESCs are almost satisfied. In theory they should all be zero, but as MOSEK is not an exact solver, the optimal solution is subject to numerical inaccuracies. If we turn to $z_J^F(G)$, then the projection distances of X^{F^*} are not as close to zero as those for X^{C^*} , because $z_J^F(G)$ is only a relaxation of $z_J^C(G)$. Also the average projection distance of X^{H^*} after solving $z_J^H(G)$ is greater than the one obtained with $z_J^F(G)$. This is in tune with the fact that the upper bounds obtained in the latter case are better for this instance. Furthermore, note that the average projection distances for X^{F^*} and X^{H^*} decrease as

Table 5 The average number of violated facets $|\mathcal{V}_I|$ over all subgraphs G_I with $I \in J_q$ before and after including the ESCs and computing $z_J^F(G)$

	J_{221}	J_{443}	J_{664}	J_{886}	J_{1107}
Before including ESCs	1.53	1.56	1.57	1.56	1.56
After computing $z_J^F(G)$	0.23	0.14	0.16	0.15	0.09

q increases. This is not surprising, as more ESCs mean that a bigger portion of the graph is forced into the stable set structure.

Finally we present in Table 5 the average number of violated facets. As one can see the average number of violated facets before including the ESCs is already very low. This means that we do not include too many inequalities that represent facets in the computation of $z_J^F(G)$. Furthermore, the average number of facets that are violated by X^{F*} decreases significantly compared to the average number of violated facets before including the relaxations of the ESCs. This is very encouraging because one could imagine a scenario where we iteratively add violated facets of one subgraph and then the optimal solution violates different facets. However, the computations suggest that this does not happen too often. Like before in Table 4 we see that the more ESCs are included, the more stable set structure is captured and therefore the less facets are violated after including the relaxations of the ESCs. ○

Let us briefly summarize the key points of Example 1. Usually the upper bounds obtained by $z_J^F(G)$ are only slightly worse than those of $z_J^C(G)$, but the running times are only a fraction. Unfortunately, this approach works only for subgraphs of order at most 6. Also $z_J^H(G)$ yields good upper bounds in slightly better running time than $z_J^F(G)$, but the bounds are a little bit worse. A major benefit of this approach is that it can be used for subgraphs of any order.

In a nutshell, both $z_J^F(G)$ and $z_J^H(G)$ are relaxations of $z_J^C(G)$ that reduce the running times drastically by worsening the bounds only a little bit. As a result these bounds are very promising for including them into a B&B algorithm for stable set.

3 Branch and bound for the stable set problem

The aim of this section is to present our implementation of an exact branch and bound (B&B) algorithm for the stable set problem (1).

3.1 Our branch and bound algorithm

We start by detailing the general setup of our B&B algorithm. Towards this end, keep in mind that in a solution of (1) the binary variable x_i is equal to 1 if vertex i is in the stable set, and $x_i = 0$ otherwise.

For our B&B algorithm for the stable set problem we choose a vertex $i \in V(G)$ and divide the optimization problem in a node of the B&B tree into the subproblem where vertex i is in the stable set (i.e., set the branching variable $x_i = 1$) and a second subproblem where i is not in the stable set (i.e., set the branching variable $x_i = 0$).

It turns out that in each node of the B&B tree the optimization problem is of the form

$$\begin{aligned}
 P(G, c) = c + \max \quad & \sum_{i \in V(G)} x_i & (9) \\
 \text{s. t.} \quad & x_i + x_j \leq 1 \quad \forall \{i, j\} \in E(G) \\
 & x_i \in \{0, 1\} \quad \forall i \in V(G)
 \end{aligned}$$

for some graph G , so in each node we have to solve a stable set problem and add a constant term c to the objective function value. Indeed, by fixing a branching variable x_i to 0 or 1, we shrink the graph and create subproblems that are again stable set problems of the form (9) but with a smaller graph and some offset c . To be more precise, for the subproblem with $x_i = 1$ the objective function value of (1) increases by 1 because there is one more vertex in the stable set. Furthermore, all neighbors of i can not be in the maximum stable set because i is already in the stable set. So we can set $x_j = 0$ for all $j \in N_G(i)$ if $N_G(i) = \{j \in V(G) \mid \{i, j\} \in E(G)\}$ denotes the set of neighbors of the vertex i in G . Furthermore, we can delete i and all neighbors of i in the current graph G and search for a maximum stable set in the new graph G' of smaller order, where $G' = G[U']$ is the subgraph of G induced by $U' = V(G) \setminus (\{i\} \cup N_G(i))$. Hence, the subproblem to solve in the new node has the form $P(G', c + 1)$.

In the subproblem for $x_i = 0$ the vertex i is not in the stable set. We can remove the vertex i from the graph and search for a maximum stable set in the induced subgraph $G'' = G[U'']$ with $U'' = V(G) \setminus \{i\}$. This boils down to solving $P(G'', c)$ in the new node of the B&B tree.

Note that whenever we delete a vertex i from the graph in the branching process, we set the according variable x_i to a fixed value. Consequently, in every node, all vertices of the original graph G are either still present, or the value of the variable corresponding to them is implicitly fixed. Furthermore, we exclude all non-feasible solutions by deleting all neighbors in case of setting the branching variable to 1. Hence, every time we set a variable of a vertex to 1 the set of all vertices of which the variable is set to 1 remains stable and we only obtain feasible solutions of (1). Therefore, from a feasible solution of (9) in any node we can determine a feasible solution of (1) with the same objective function value.

The order of the graph to consider in a node shrinks whenever we branch. As a consequence the B&B tree is of finite size. Whenever we reach a node with a subproblem on a graph with less or equal to 23 vertices, we solve the problem by a fast enumeration procedure that can be employed whenever the subproblems become sufficiently small. To do so, we iterate easily—and especially fast—over all subsets of V in descending order with an implementation of Hinnant (2011) in C++.

Bounds We do not want to solve the subproblems (9) in each node to optimality, but only compute an upper and a lower bound on the optimal objective function value. This boils down to obtaining bounds on the stability number of the graph considered in (9). We present details on lower bounds obtained by heuristics in Sect. 3.2.

As upper bounds we use the relaxations based on ESCs in four different versions, namely the convex hull formulation or the bundle method as detailed in Sect. 2.1, the violated facets version as described in Sect. 2.2 or the separating hyperplanes version as presented in Sect. 2.3. For choosing the subset J of ESCs, we follow the approach of Gaar and Rendl (2020) in our computations and perform several cycles, i.e., iterations of the repeat until loop, of solving (a relaxation of) (3) and then adjusting the set J , as illustrated in Algorithm 1.

In particular, we start with $J = \emptyset$, as preliminary computations have shown that carrying over ESCs to subproblems does not pay off, and in each cycle we update J depending on the current optimal solution X^* of the SDP solved. We remove all previously added ESCs where the associated dual variables of the optimal solution have absolute value less than 0.01. For finding violated subgraphs (i.e., subgraphs for which the ESC does not hold in X^*) we use the methods presented in Gaar and Rendl (2020), so we use a local search heuristic to find submatrices of X^* that minimize the inner product with some matrices. We let the local search heuristic run for $9n$ times and add random subgraphs to obtain $9n$ subgraphs without duplicates. From these subgraphs we add the $3n$ most violated ones (subsets I with largest projection distance of X_I^* to $\text{STAB}^2(G_I)$) to J .

To reduce computational effort, we stop cycling as soon as we do not expect to be able to prune within the next 5 cycles assuming that the decrease of the upper bound z^* in each future cycle is 0.75 of the average decrease we had in the previous cycles.

Algorithm 1: Upper bound computation at a B&B node

Input: Graph G at current node, method *convex-hull*, *bundle-method*, *violated-facets*, or *separating-hyperplanes*

Output: Upper bound z^*

```

1  $J = \emptyset$ 
2 repeat
3   if convex-hull then let  $X^*$  be an optimal solution of (4) with objective function value  $z^*$ 
   if bundle-method then let  $X^*$  be the solution of approximately solving (4) using the bundle
   method with objective function value  $z^*$ 
   if violated-facets then let  $X^*$  be an optimal solution of (5) with objective function value  $z^*$ 
   if separating-hyperplanes then let  $X^*$  be an optimal solution of (8) with objective function
   value  $z^*$ ,
   where we choose  $\tilde{X}$  as  $X^*$  of the last cycle
   remove ESCs from  $J$  if associated dual variables have small absolute value
4   search for ESCs violated by  $X^*$  and add them to  $J$ 
5 until optimistic forecast does not suggest pruning
6 return  $z^*$ 

```

For our computations we use the implementation of the bundle method as it is detailed in Gaar and Rendl (2020), in particular with all specifications given in Section 6.3 therein and we let the bundle method run for at most 15 iterations in each cycle.

Branching Rule An important question in the B&B algorithm is how to choose the branching variable. In our implementation we follow the approach to first deal with

vertices, for which we know least whether they will be in a maximum stable set or not (“*difficult first*”) in order to find an optimal solution soon.

All our upper bounds are based on the Lovász theta function (2), so we can use the intuition that the closer an entry x_i is to 1 in an optimal solution of (2), the more likely it is that this vertex i is in a maximum stable set. Hence, we choose the variable x_i with value closest to 0.5 as branching variable.

More on branching rules for the stable set problem can for example be found in Fondzefe (2016); Xiao (2009).

Diving Strategy We implemented a best first search strategy, where we always consider the open subproblem with the largest upper bound next. We expect that we find a large stable set in this branch of the B&B tree because the difference between the global lower bound and the upper bound for this branch is the highest of all.

3.2 Heuristics to find large stable sets

It is crucial to find a good lower bound on $\alpha(G)$ early in the B&B algorithm to prevent the growth of the B&B tree and therefore solve the stable set problem more efficiently. In Singh and Pandey (2015) and Wu and Hao (2015), for example, one can find references to some heuristics to find a large stable set. In our implementation we use several different heuristics.

The first heuristic makes use of the vector x from the SDP formulation (2) of $\vartheta(G)$, which is available from the upper bound computation. This vector consists of n elements between 0 and 1. The value x_i gives us some intuition about the i -th vertex of the graph, namely the closer it is to 1, the more likely it is that the vertex is in a maximum stable set. Hence, we sort the vertices in descending order according to their value in x and then add the vertices in this order to a set S , such that the vertices of S always remain a stable set. In the following we refer to this heuristic as (HT).

Furthermore, we use a heuristic introduced by Khan and Khan (2013) based on vertex covers and vertex supports. A subset C of the vertices of a graph is called vertex cover if for each edge at least one of the two incident vertices is in C . The vertex support of a vertex is defined as the sum of the degrees of all vertices in the neighborhood of this vertex. If C is a vertex cover, then clearly $V(G) \setminus C$ is a stable set, so instead of searching for a maximum stable set we can search for a vertex cover of minimum cardinality. In a nutshell, the heuristic of Khan and Khan (2013) searches for a vertex with maximum vertex support in the neighborhood of the vertices with minimum vertex support. If there is more than one vertex with maximum support, one with maximum degree is chosen. This vertex is added to the vertex cover C and all incident edges are removed. The above steps are repeated until there are no edges left in the graph. In the end we obtain a hopefully large stable set with $V(G) \setminus C$. We denote this heuristic by (HVC).

Finally we use a heuristic proposed by Burer et al. (2002). Their heuristic is based on the SDP formulation of the Lovász theta function with additional restriction to the matrix variable to be of low rank. With rank one, a local maximizer of the problem yields a maximal stable set, whereas with rank two the stable set corresponding to the local maximizer does not necessarily have to be maximal, but one can escape to

a higher local maximizer which corresponds to a maximal stable set. The C source code of this heuristic is online available at Burer and Monteiro (2014). We use this code with the parameters rank set to 2 and the number of so-called escapes set to 1 in a first version and set to 5 in a second version. Both parameter settings are among the choices that were tested in Burer et al. (2002). We will refer to these versions with (H21) and (H25).

In the B&B algorithm we perform the heuristic (H25) in the root node with a time limit of 20 seconds. Then we only perform the heuristics in every third node of the B&B tree. (HT) and (HVC) are very fast, so we let them run in each node we run heuristics. Furthermore, in the first 10 nodes of the B&B tree we perform (H25) with the hope to find a stable set of cardinality $\alpha(G)$ as soon as possible, but we do not allow the heuristic to iterate longer than 7 seconds. For graphs with less than 200 vertices we additionally perform (H21) with the running time limited to 1 second. On graphs with more vertices we perform with probability 0.05 (H25) and a time limit of 7 seconds and in the other cases (H21) with a time limit of 3 seconds. In a computational comparison in the master's thesis of Siebenhofer (2018), this turned out to be the best combination of heuristics.

4 Computational experiments

In this section we finally compare the B&B algorithms using the different upper bounds presented so far. In Table 6 and Fig. 1 we compare the number of nodes generated in the B&B algorithm as well as the CPU time and the final gap. The abbreviations refer to the following bounds used.

- (CH) We consider the upper bound obtained by the ESCs in the convex hull formulation (CH) described in Sect. 2.1,
- (BD) solving this formulation with the bundle method (BD) as presented in Sect. 2.1,
- (VF) relaxing this formulation by considering only violated facets (VF) as described in Sect. 2.2,
- (SH) and using only separating hyperplanes (SH) as presented in Sect. 2.3.
- (TH) For better comparability we also consider our B&B algorithm with only the Lovász theta function (2) as an upper bound and denote this version with (TH). Note that this boils down to solving (3) with $J = \emptyset$.

If we are not able to solve an instance within the timelimit, we indicate this in Table 6 by a cell that is colored . Whenever a cell is colored it means that the run did not finish correctly, for example because MOSEK produced an error or ran out of memory. Before discussing the results, we give the details on the instances as well as on the soft- and hardware.

4.1 Benchmark set and experimental setup

We consider several different instances. First, we consider the instances used in Gaar and Rendl (2020), i.e., torus graphs, random near- r -regular graphs and random graphs from the Erdős-Rényi model and also several instances from the literature. Additionally

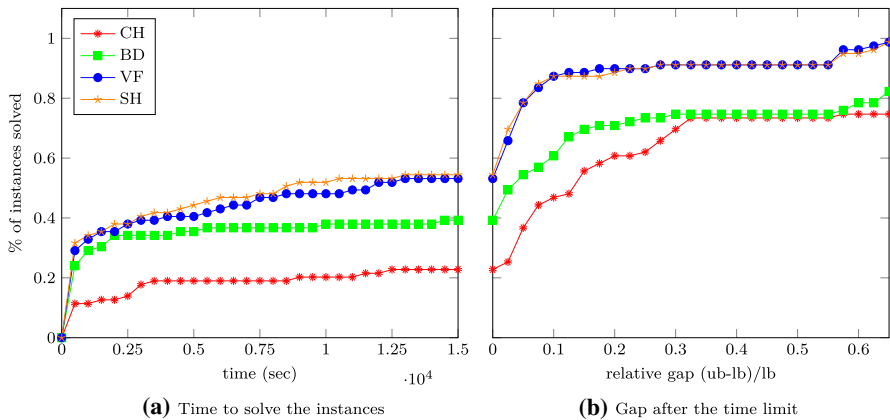


Fig. 1 The performance comparison of the bounds (CH), (BD), (VF) and (SH)

we consider all instances from the DIMACS challenge for which the gap between $\vartheta(G)$ and $\alpha(G)$ is larger than one (i.e., all instances which are not solved in the root node of our B&B algorithm) and that have at most 500 vertices. Moreover, we consider spin graphs, which are produced with the command `./rudy -spinglass3pm x x x 50 xxx1` (for $x \in \{5, 7, 9, 11\}$) by the graph generator `rudy`, which was written by Giovanni Rinaldi.¹

We implemented our B&B algorithm with different upper bounds in C++. All computations were performed on an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz with 32 GB RAM. All programs were compiled with `gcc` version 5.4.0 with the optimization level `-O3` and the CPU time was measured with `std::clock`. We set the random seed to zero. We use MOSEK [22] 8.1 in the methods (CH), (VF), and (SH) to solve the SDPs (4), (5), and (8), respectively. Furthermore, we use it within the method (BD) for solving the subproblems within the bundle method for computing an approximate solution of (4). Moreover, we use it to solve the QP (6) to compute the projection distance in (SH) and when updating J , i.e., while searching for subgraphs with violated ESCs and adding these subgraphs to J . The execution time of our B&B algorithm is limited to 4 hours, i.e., after this computation time we allow the B&B to finish solving the SDP of the already started node in the B&B tree and then stop.

4.2 First computational experiments

We first want to compare the two different versions of the B&B algorithm that use (CH) and (BD) to compute upper bounds, i.e., we compare those versions that have already been established as upper bounds in the literature, but are now for the first time used within a B&B algorithm.

First, by looking at Table 6 we observe that 20 instances were not solved correctly with (CH), which is due to the fact that the SDPs to solve are huge and therefore MOSEK runs out of memory very often. Indeed, by using (BD) and hence not having

¹ Available at www-user.tu-chemnitz.de/~helmborg/rudy.tar.gz

Table 6 The number of nodes in our B&B algorithm

Graphs	<i>n</i>	<i>m</i>	α	(TH)	(CH)	(BD)	(VF)	(SH)
ER_100_15	100	737	24	25	19	19	19	19
ER_100_25	100	1262	17	27	49	23	21	23
ER_100_50	100	2501	9	23	21	21	21	21
ER_125_15	125	1123	27	243	22	173	139	149
ER_125_25	125	1928	20	29	21	25	23	21
ER_125_50	125	3904	9	69	63	63	63	63
ER_150_15	150	1664	28	1469	41	-	835	871
ER_150_25	150	2810	19	543	47	417	401	429
ER_150_50	150	5608	10	79	58	77	77	77
ER_175_15	175	2213	30	4633	35	-	1856	2488
ER_175_25	175	3777	20	1191	32	818	985	995
ER_175_50	175	7578	11	91	24	87	87	87
ER_200_15	200	2917	33	3149	21	479	1342	1771
ER_200_25	200	4924	21	2883	18	274	1196	1861
ER_200_50	200	9913	11	135	12	52	131	131
torus5	25	50	10	3	1	1	1	1
torus7	49	98	21	11	1	1	1	1
torus9	81	162	36	33	1	7	1	1
torus11	121	242	55	79	1	19	1	3
torus13	169	338	78	211	6	59	5	5
torus15	225	450	105	551	3	-	11	21
torus17	289	578	136	1453	2	-	23	47
torus19	361	722	171	-	-	-	31	98
torus21	441	882	210	2956	-	-	23	105
torus23	529	1058	253	1750	2	59	16	89
torus25	625	1250	300	1140	-	47	10	61
torus27	729	1458	351	774	-	33	7	41
torus29	841	1682	406	520	-	23	5	28
torus31	961	1922	465	369	12	17	4	20
torus33	1089	2178	528	261	-	12	2	14
torus35	1225	2450	595	188	-	9	2	10
torus37	1369	2738	666	134	-	6	2	7
reg_n100_r4	100	195	40	145	8	71	11	15
reg_n100_r6	100	294	34	251	32	123	47	51
reg_n100_r8	100	377	31	141	18	85	37	39
reg_n100_r10	100	474	28	147	45	95	51	55
reg_n200_r4	200	400	81	9873	24	-	417	647
reg_n200_r6	200	593	$69 \leq \alpha \leq 72$	28221	24	642	1023	1447
reg_n200_r8	200	792	$60 \leq \alpha \leq 63$	27441	24	630	1242	1663
reg_n200_r10	200	980	$57 \leq \alpha \leq 59$	26591	24	606	1257	1657
rand_n100_p004	100	212	45	3	1	1	1	1
rand_n100_p006	100	303	38	23	11	13	5	9
rand_n100_p008	100	443	32	35	11	19	11	11
rand_n100_p010	100	489	32	15	2	9	3	3
rand_n200_p003	200	631	81	39	6	29	9	11
rand_n200_p004	200	816	67	6165	23	-	423	999
rand_n200_p005	200	991	64	879	19	-	279	295
CubicVT26_5	26	39	10	7	1	3	1	1
HoG_6575	45	225	10	189	79	105	77	77
Circulant47_030	47	282	13	5	1	1	1	1
PaleyGraph61	61	915	5	49	47	47	47	47
spin5	125	375	50	325	1	1	1	1
spin7	343	1029	$147 \leq \alpha \leq 151$	5992	-	-	79	181
spin9	729	2187	$324 \leq \alpha \leq 333$	670	-	23	4	41
spin11	1331	3993	$605 \leq \alpha \leq 625$	126	-	4	1	8
MANN_a9	45	72	16	7	3	5	3	3
MANN_a27	378	702	126	-	4	188	131	580
hamming6_4	64	1312	4	17	1	1	1	1
MANN_a45	1035	1980	345	280	-	14	17	43
sanr200_0_9	200	2037	42	5623	22	475	1102	1693
brock200_1	200	5066	21	1691	18	296	1073	1379
keller4	171	5100	11	167	24	103	119	129
sanr200_0_7	200	6032	18	1099	18	254	905	935
brock200_4	200	6811	17	709	16	185	619	627
brock200_3	200	7852	15	469	14	122	403	409
brock200_2	200	10024	12	127	14	54	123	123
c_fat200_5	200	11427	58	11	1	7	7	7
p_hat300_3	300	11460	36	622	4	-	58	102
brock400_2	400	20014	29	34	-	4	10	14
brock400_4	400	20035	33	34	-	4	10	13
brock400_1	400	20077	27	36	-	2	9	12
brock400_3	400	20119	31	35	-	2	10	12
p_hat300_2	300	22922	25	24	1	-	8	10
sanr400_0_7	400	23931	21	22	-	2	6	8
san400_0_7_3	400	23940	22	65	-	3	1	14
p_hat500_3	500	30950	49	8	-	1	2	4
p_hat300_1	300	33917	8	6	1	-	2	2
sanr400_0_5	400	39816	13	5	4	1	2	2

to solve that large SDPs only 10 instances are not solved correctly, most of them due to other MOSEK errors. When we compare the number of B&B nodes for the instances where both (CH) and (BD) finished we see that typically the number is the same, or there are slightly more nodes for (BD). This is plausible, as we only have an approximate solution of (3) when using (BD), but an exact solution of (3) with (CH).

We next take a closer look on the lines labeled (CH) and (BD) in the performance profiles in Fig. 1. The B&B code using (BD) as bounding routine can solve much more instances within a given time than when (CH) is used. Once the time limit is reached, the gap is typically much lower for (BD) than it is for (CH).

In a nutshell, even though (BD) solves only a relaxation of (3), using it is much faster than using (CH) while it does not increase the number of B&B nodes a lot. This justifies considering only a relaxation of (3).

4.3 Computational experiments with new relaxations

Up to now we have used the exact subgraph approach of Adams et al. (2015) with the implementation proposed by Gaar and Rendl (2019, 2020) in order to get tight upper bounds on the stability number within a B&B algorithm for solving the stable set problem. So far we have proven the strength of the bounds by showing that the number of nodes in a B&B algorithm reduces drastically by using these bounds, however the computational costs are enormous in the original version with the convex hull formulation (CH) and they are still substantial with the bundle approach (BD) from Gaar and Rendl (2019, 2020). Therefore, we now discuss the numerical results of the B&B algorithm using the new relaxations (VF) and (SH).

Looking at Table 6, the first thing we observe is that both (VF) and (SH) never lead to a MOSEK error, hence they are more robust than the other versions, presumably due to their smaller size of the SDPs to solve in the B&B nodes. For 9 instances both (VF) and (SH) and in 13 instances at least one of (VF) and (SH) is able to finish within the time limit for an instance where both (CH) and (BD) were not able to finish.

If we compare the number of B&B nodes in Table 6 for the finished instances, then we see that typically the number of B&B nodes for (VF) is smaller than those of (BD), which makes sense because in (BD) we only approximately solve (3) whereas in (VF) we solve a possibly very tight relaxation of (3) exactly. The number of B&B nodes needed by (SH) is typically a little bit larger than the one of (VF), which is in tune with the empirical finding that (5) gives better bounds than (8) in the small example considered in Sect. 2.5. In a nutshell, for finished runs typically (CH) and (VF) need roughly the same number of nodes, (SH) needs a little bit and (BD) needs many more nodes in the B&B tree.

As for the running times, in Fig. 1 we see that both (VF) and (SH) are faster than (BD) and considerably faster than (CH). (VF) is a little bit slower than (SH). For those instances that cannot be solved to optimality, the gap when the time limit is reached is roughly the same for (VF) and (SH), and it is considerably tighter than for (BD).

We have demonstrated that within a B&B algorithm both our relaxations (VF) and (SH) work better than already existing SDP based methods. In particular using (SH) allows to keep the majority of the strength of the upper bound (3) (i.e.,

keeping the number of vertices in the B&B tree low) by reducing the running time so that within the time limit almost 60% of the instances are solved, as compared to (CH) that only manages to solve a bit more than 20%.

5 Conclusions

We introduced an algorithm for computing the stability number of a graph using semidefinite programming. While there exist several exact solution methods for finding the stability number, those based on semidefinite programming are rather rare.

We implemented a B&B algorithm using the SDP relaxations introduced in Gaar and Rendl (2019, 2020) together with heuristics from the literature. Moreover, we further relaxed the SDPs, getting more tractable SDPs still producing high-quality upper bounds. This is confirmed by the numerical experiments where we compare the number of nodes to be explored in the B&B tree as well as the CPU times.

While SDPs produce strong bounds, the computational expense for solving the SDPs is sometimes huge. In particular, there is potential for improvement of the running time for solving SDPs with many constraints. We use MOSEK as a solver, which uses the interior point method to solve an SDP. For large instances it would be beneficial to use a solver based on the boundary point method (Povh et al. 2006; Malick et al. 2009) or DADAL (De Santis et al. 2018). Moreover, the solver computing ϑ^+ as an upper bound (Cerulli et al. 2020) combined with the relaxations above, may push the performance of the B&B solver even further. Also, these other solvers are capable of doing warm starts, that can have big advantages within a B&B framework. Since all these implementations are available as MATLAB source code only, they need to be translated to C or C++ first. This will be part of our future work.

Another line of future research is working out different strategies for identifying violated subgraphs, that should also lead to a more efficient overall algorithm.

Acknowledgements This project has received funding from the Austrian Science Fund (FWF): I3199-N31. Moreover, the second author has received funding from the Austrian Science Fund (FWF): DOC 78. Furthermore, we thank two anonymous referees for their valuable input.

Funding Open access funding provided by University of Klagenfurt. This project has received funding from the Austrian Science Fund (FWF): I3199-N31. Moreover, the second author has received funding from the Austrian Science Fund (FWF): DOC 78.

Availability of data and material All data comes from publicly available sources.

Code availability The source code is available upon request from the authors.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence,

and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Adams E, Anjos MF, Rendl F, Wiegale A (2015) A Hierarchy of subgraph projection-based semidefinite relaxations for some NP-hard graph optimization problems. *INFOR Inf Syst Op Res* 53(1):40–47. <https://doi.org/10.3138/infor.53.1.40>
- Boyd S, Vandenberghe L (2004) *Convex optimization*. Cambridge University Press, Cambridge
- Burer S, Monteiro RDC (2014) Max-AO, Georgia Tech, v0.120301. <https://github.com/sburer/Max-AO>. Accessed 2020-11-06
- Burer S, Monteiro RDC, Zhang Y (2002) Maximum stable set formulations and heuristics based on continuous optimization. *Math Program* 94:137–166. <https://doi.org/10.1007/s10107-002-0356-4>
- Cerulli M, Santis MD, Gaar E, Wiegale A (2020) Improving ADMMs for solving doubly nonnegative programs through dual factorization. *4OR*. <https://doi.org/10.1007/s10288-020-00454-x>
- Christof T SMAPO: Cut Polytope. <http://comopt.ifl.uni-heidelberg.de/software/SMAPO/cut/cut.html>. Accessed 2020-11-06
- Conforti M, Cornuejols G, Zambelli G (2014) *Integer Programming*. Springer, Berlin
- De Santis M, Rendl F, Wiegale A (2018) Using a factored dual in augmented Lagrangian methods for semidefinite programming. *Oper Res Lett* 46(5):523–528
- Depolli M, Konc J, Rozman K, Trobec R, Janezic D (2013) Exact parallel maximum clique algorithm for general and protein graphs. *J Chem Inf Model* 53(9):2217–2228. <https://doi.org/10.1021/ci4002525>
- Fondzefe F.T. Advanced branching rules for maximum stable set integer programs. Master's thesis, Università degli Studi dell'Aquila (2015/16)
- Gaar E (2020) On different versions of the exact subgraph hierarchy for the stable set problem. <https://arxiv.org/abs/2003.13605>
- Gaar E, Rendl F (2019) A bundle approach for SDPs with exact subgraph constraints. In: Lodi A, Nagarajan V (Eds) *Integer Programming and Combinatorial Optimization*. Springer, pp 205–218
- Gaar E, Rendl F (2020) computational study of exact subgraph based SDP bounds for Max-Cut, stable set and coloring. *Math Program* 183:283–308. <https://doi.org/10.1007/s10107-020-01512-2>
- Grötschel M, Lovász L, Schrijver A (1988) *Geometric algorithms and combinatorial optimization*. In: *Algorithms and combinatorics: study and research texts*. vol. 2. Springer, Berlin
- Håstad J (1999) Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math* 182(1):105–142. <https://doi.org/10.1007/BF02392825>
- Helmberg C, Rendl F (2000) A spectral bundle method for semidefinite programming. *SIAM J Optim* 10(3):673–696
- Hinnant H (2011) *Combinations and permutations*. <https://howardhinnant.github.io/combinations.html>. Accessed 2020-11-06
- Karp R.M. (1972) Reducibility among combinatorial problems. In: *Complexity of computer computations* (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), pp. 85–103
- Khan I, Khan H (2013) Modified vertex support algorithm: a new approach for approximation of minimum vertex cover. *Res J Comput Inf Technol Sci* 1(6):1–6
- Letchford AN, Rossi F, Smriglio S (2020) The stable set problem: clique and nodal inequalities revisited. *Comput Op Res*. <https://doi.org/10.1016/j.cor.2020.105024>
- Malick J, Povh J, Rendl F, Wiegale A (2009) Regularization methods for semidefinite programming. *SIAM J Optim* 20(1):336–356. <https://doi.org/10.1137/070704575>
- MOSEK ApS: MOSEK Fusion API for C++ 8.1.0.53. <https://docs.mosek.com/8.1/cxxfusion/index.html>. (2018). Accessed 2020-11-06
- Nemhauser GL, Wolsey LA (1988) *Integer and combinatorial optimization*. Wiley-interscience series in discrete mathematics and optimization. John Wiley & Sons Inc., New York
- Povh J, Rendl F, Wiegale A (2006) A boundary point method to solve semidefinite programs. *Computing* 78(3):277–286. <https://doi.org/10.1007/s00607-006-0182-2>

- Siebenhofer M (2018) An SDP-based approach for finding the stability number of a graph. Master's thesis, Alpen-Adria-Universität Klagenfurt
- Singh KK, Pandey AK (2015) Survey of algorithms on maximum clique problem. *Int Adv Res J Sci Eng Technol* 2:15–20
- Wu Q, Hao JK (2015) A review on algorithms for maximum clique problems. *Eur J Oper Res* 242(3):693–709. <https://doi.org/10.1016/j.ejor.2014.09.064>
- Xiao M (2009) New branching rules: Improvements on independent set and vertex cover in sparse graphs. ArXiv e-prints

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.