

A Branch and Cut Solver for the Maximum Stable Set Problem

Steffen Rebennack · Marcus Oswald · Dirk Oliver
Theis · Hanna Seitz · Gerhard Reinelt · Panos
M. Pardalos

Received: date / Accepted: date

Abstract This paper deals with the cutting-plane approach to the maximum stable set problem. We provide theoretical results regarding the facet-defining property of inequalities obtained by a known project-and-lift-style separation method called edge-projection, and its variants. An implementation of a Branch and Cut algorithm is described, which uses edge-projection and two other separation tools which have been discussed for other problems: local cuts (pioneered by Applegate, Bixby, Chvátal and Cook) and mod- k cuts. We compare the performance of this approach to another one by Rossi and Smiriglio (2001) and discuss the value of the tools we have tested.

Keywords maximum stable set problem · cutting-plane algorithm · Branch and Cut · separation algorithm · edge-projection

Panos M. Pardalos is partially supported by Airfoce and DTRA grants

Steffen Rebennack
Department of Industrial & Systems Engineering, University of Florida, Gainesville, FL, USA
E-mail: steffen@ufl.edu

Marcus Oswald
Discrete Optimization Research Group, Ruprecht-Karls Universität Heidelberg, Heidelberg, Germany
E-mail: marcus.oswald@informatik.uni-heidelberg.de

Dirk Oliver Theis
Fakultät für Mathematik (IMO), OvG-Universität Magdeburg, Magdeburg, Germany
E-mail: dirk.theis@ovgu.de

Hanna Seitz
Discrete Optimization Research Group, Ruprecht-Karls Universität Heidelberg, Heidelberg, Germany
E-mail: hanna.seitz@informatik.uni-heidelberg.de

Gerhard Reinelt
Discrete Optimization Research Group, Ruprecht-Karls Universität Heidelberg, Heidelberg, Germany
E-mail: gerhard.reinelt@informatik.uni-heidelberg.de

Panos M. Pardalos
Department of Industrial & Systems Engineering, University of Florida, Gainesville, FL, USA
E-mail: pardalos@ise.ufl.edu

1 Introduction

This paper develops a Branch & Cut algorithm to solve the stable set problem. Therefore, let $G = (V, E)$ be an undirected graph with node set V and edge set E . Then, a stable set of graph G is defined as a (sub)set \mathcal{S} of node set V with the property that each pair of nodes of \mathcal{S} are not connected via an edge in graph G ; *i.e.*, the nodes are pairwise non adjacent. A stable set is sometimes also called independent set, vertex packing, co-clique or anticlique. We associate a node weight c_i with each node $i \in V$ in graph G ; the case of $c \equiv \mathbf{1}$ is allowed. The maximum stable set problem aims to find a stable set \mathcal{S} which maximizes the node weights of \mathcal{S} ; *i.e.*, $\sum_{i \in \mathcal{S}} c_i$.

The maximum stable set problem is one of the ‘classical’ combinatorial optimization problems and has many applications, see for instance [Strickland et al. \(2005\)](#), [Vries and Vohra \(2003\)](#), [Bomze et al. \(1999\)](#), or [Rebennack \(2006\)](#). However, the stable set problem still remains computationally difficult to solve. It belongs to the class of \mathcal{NP} -hard problems ([Garey and Johnson 1979](#)). Moreover, it is even hard to approximate ([Arora and Safra 1992](#); [Fujisawa et al. 1995](#)).

Different approaches have been discussed in the literature to solve the stable set problem exactly. For a list of exact methods, we refer to [Bomze et al. \(1999\)](#), [Butenko \(2003\)](#), [Warren and Hicks \(2006\)](#), [Rebennack \(2008\)](#), and [Avenali \(2007\)](#). Computational results for different stable set linear programming relaxations have been reported by [Gruber and Rendl \(2003\)](#). The development of Branch & Cut solvers was driven by [Mannino and Sassano \(1996\)](#) through their introduction of the edge projection. This work was adapted to separation routines by [Rossi and Smriglio \(2001\)](#). Their resulting Branch & Cut algorithm still provides one of the best dual bounds obtained by cutting plain approaches and their handling of the Branch & Bound tree has no competition. Therefore it is used as a benchmark for our computations. Branch & Price methods have also been successfully applied to the stable set problem ([Warrier et al. 2005](#); [Warrier 2007](#)). Recently, promising results have been obtained by semidefinite programming relaxations of the stable set polytope ([Dukanovich and Rendl 2007](#)) and Lagrangian relaxations ([Campelo and Correa 2009](#)).

This paper extends the theory of the edge-projection methodology introduced by [Mannino and Sassano \(1996\)](#). In particular, we provide examples of a facet-defining inequality for the projected graph which is not facet-defining for the original graph and vice versa. Furthermore, we are able to give a sufficient condition for an inequality, obtained from the edge-projection, to be facet-defining for the original graph. As a third result, the edge-projection is extended to be valid for general inequalities, under some defined conditions.

The implemented Branch & Cut algorithm in this paper combines several different separation routines: the odd-hole inequalities, clique inequalities, rank inequalities, mod- k inequalities, local cuts and large cuts are separated. The separation of mod- k inequalities and local cuts is new for the stable set problem. Extensive computational results are presented, using some of the DIMACS benchmark graphs. The upper bounds are good compared to the Branch & Cut algorithm by [Rossi and Smriglio \(2001\)](#).

This paper is organized as follows. In Section 2, the edge projection and the concept of local cuts are discussed. Section 3 describes the implemented Branch & Cut algorithm while computational results are presented in Section 4. We conclude with Section 5.

2 Separation Routines

In this section, we discuss the separation of the rank inequalities (Section 2.1) and so-called local cuts (Section 2.2).

2.1 Rank Inequalities

The so-called edge-projection was introduced by [Mannino and Sassano \(1996\)](#) in order to separate the rank inequalities

$$x(W) := \sum_{v_i \in W} x_i \leq \alpha(G[W]), \quad (1)$$

for graph $G = (V, E)$ and subset $W \subseteq V$; where $\alpha(G)$ denotes the stability number of graph G . The rank inequalities are a large class of valid inequalities for the stable set polytope; *i.e.*, the edge, odd-cycle, clique, antihole, web and antiweb inequalities belong to this class.

The appealing idea of the edge-projection is to reduce the size of the graph and to make it denser at the same time. This process, which is called *edge-projection*, should fasten the separation routines for a class of rank inequalities for the resulting graph, for instance the clique separation. Once a violated inequality for that smaller graph has been found, it is adjusted to be valid for the whole graph. This process is called *anti-projection*. After reviewing the latest results of [Rossi and Smriglio \(2001\)](#) we will discuss some new polyhedral aspects.

We start with an exact definition of the projection of a graph G . For a given edge $e = uv$ the new graph is constructed in the following way. Both endnodes u, v of edge e and their *common neighborhood* $\Gamma_{uv} := \Gamma(u) \cap \Gamma(v)$ are deleted from the graph. This has the effect that the edges in the set

$$E_{uv} := \{v_i v_j \in E \mid v_i \text{ or } v_j \in \Gamma_{uv} \cup \{u, v\}\}$$

are removed, too. In addition, new edges, the so-called *false edges*, are added. In the new graph, every node which is a neighbor of node u but not of v is adjacent to all nodes which are neighbors of v but not of u . The corresponding edge set is defined as

$$\bar{E}_{uv} := \{v_i v_j \mid v_i v_j \in V \setminus (\{u, v\} \cup \Gamma_{uv}) \text{ and } \Gamma(\{v_i, v_j\}) \supseteq \{u, v\}\}.$$

The elements e of \bar{E}_{uv} are called false edges, if $e \notin E$. We get the formal

Definition 1 The graph $G|e := (V|e, E|e)$ with $V|e := V \setminus (\Gamma_{uv} \cup \{u, v\})$ and edge set $E|e := E \setminus E_{uv} \cup \bar{E}_{uv}$ is called the *projection* of e in G .

In order to have a chance to make a rank inequality of $G|e$ valid for the polytope of the original graph G , we only consider edges e with a special property. An edge $e = uv \in E$ is called *projectable* in G , if there is a maximum stable set \mathcal{S}^* in G such that $\mathcal{S}^* \cap \{u, v\} \neq \emptyset$. It is called *strongly projectable* in G if it is projectable in every induced subgraph of G containing both u and v . These definitions indicate the following

Lemma 1 ([Mannino and Sassano 1996](#)) *Let $e = uv$ be a projectable edge in G . Then $\alpha(G) = \alpha(G|e) + 1$.*

This implies that a valid rank inequality for the projection $G|e$ can be anti-projected by adding the deleted nodes of the projection step. This is stated in the following theorem.

Theorem 1 ([Rossi and Smriglio 2001](#)) *Let $e = uv$ be a projectable edge in G and $W \subseteq V|e$. If $x(W) \leq l$ is a valid rank inequality for $P_{STAB}(G|e)$, then $x(W) + x(\Gamma_{uv}) + x_u + x_v \leq l + 1$ is valid for $P_{STAB}(G)$.*

The next question is how the projectable edges can be characterized. Therefore we consider first the case of strongly projectable edges.

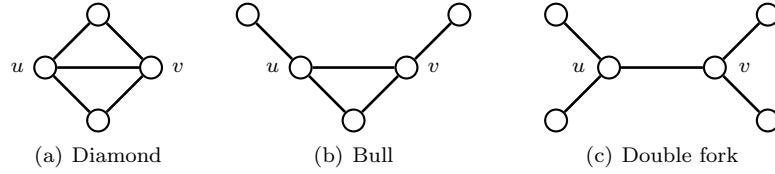


Fig. 1 Diamond, bull and double fork; see also (Rossi and Smriglio 2001, Fig. 1)

Theorem 2 (Rossi and Smriglio 2001) *An edge $e = uv \in E$ is strongly projectable in G , if and only if it is not the central edge of an induced subgraph isomorphic to a diamond, Fig. 1 (a), a bull, Fig. 1 (b), or a double fork, Fig. 1 (c).*

Theorem 2 indicates the following necessary condition for a strongly projectable edge.

Corollary 1 *If edge $uv \in E$ is strongly projectable in G , then the common neighborhood Γ_{uv} is a clique.*

Proof Let $uv \in E$ be a strongly projectable edge and Γ_{uv} be no clique. Then there are two nodes v_i, v_j in set Γ_{uv} which are not adjacent. This implies that the graph indicated by the nodes v_i, v_j, u and v is a diamond with central edge uv which is a contradiction to Theorem 2.

According to its definition, every strongly projectable edge is projectable. This implies

Theorem 3 (Mannino and Sassano 1996) *An edge $e = uv \in E$ is projectable in G , if it is not the central edge of an induced subgraph isomorphic to a diamond or a bull or a double fork.*

Note that in this case we only have one direction. In fact, it is \mathcal{NP} -hard to check whether an edge is projectable or not (Mannino and Sassano 1996). In contrast, Theorem 2 provides a method to test strong projectability in polynomial time. For practical use however, it turns out that the number of strongly projectable edges for the relevant graphs is quite small. A check of the DIMACS Challenge benchmark graphs (DIMACS 1992/1993) shows that they do not contain any strongly projectable edge¹. The MANN graphs form an exception according to there special structure arising from the transformation of the set covering problem. In these four graphs half of the edges are strongly projectable. To use the projection nevertheless, we help us with a trick. The following corollary will enable us to make any edge uv strongly projectable in a smaller subgraph. The idea is to remove some edges which destroys the strong projectability. Now consider

Corollary 2 (Rossi and Smriglio 2001) *Let $e = uv \in E$. If $\Gamma(u) \setminus \{v\}$ is a clique, then uv is strongly projectable.*

To make an edge uv strongly projectable, one can select a clique Q from set $\Gamma(u) \setminus \{v\}$ and delete all edges uw where w is in set $\Gamma(u) \setminus (\{v\} \cup Q)$ —the same could be done for node v . The drawback is that the deletion of edges changes the structure of the resulting graph and decreases the density which negatively effects the projection. To reduce this drawback, one idea could be

¹ Tested graphs: brock200_1, brock200_2, brock200_3, brock200_4, brock400_1, brock400_2, brock400_3, brock400_4, brock800_1, brock800_2, brock800_3, brock800_4; c-fat200-1, c-fat200-2, c-fat200-5, c-fat500-1, c-fat500-2, c-fat500-5, c-fat500-10; hamming6_2, hamming6_4, hamming8_2, hamming8_4, hamming10_2, hamming10_4; johnson8-2-4, johnson8-4-4, johnson16-2-4, johnson32-2-4; keller4, keller5; p-hat300-1, p-hat300-2, p-hat300-3, p-hat500-1, p-hat500-2, p-hat500-3, p-hat700-1, p-hat700-2, p-hat700-3, p-hat1000-1, p-hat1000-2, p-hat1000-3, p-hat1500-1, p-hat1500-2, p-hat1500-3; san200_0.7_1, san200_0.7_2, san200_0.9_1, san200_0.9_2, san200_0.9_3, san400_0.5_1, san400_0.7_1, san200_0.7_2, san200_0.7_3, san200_0.9_1, san1000; sanr200_0.7, sanr200_0.9, sanr400_0.5, sanr400_0.7; C125.9, C250.9.

to compute a large clique. Unfortunately, the computation of a maximum clique in an arbitrary graph is \mathcal{NP} -hard and one has to use a heuristic.

Now consider an example. In Fig. 2 (a) we select the edge v_3v_5 as a candidate for the projection. As this edge is not the central edge of an induced subgraph isomorphic to a diamond, bull or double fork, it is strongly projectable. We recognize that $\Gamma(v_5) \setminus \{v_3\} = \{v_2, v_4\}$ is no clique but the criteria of Corollary 2 holds for node v_3 . The projection of v_3v_5 removes nodes v_3, v_4 and v_5 and adds the false edge v_2v_6 . This can be seen in Fig. 2 (b). A separation routine could find the clique inequality $x_1 + x_2 + x_6 \leq 1$. Note that this inequality is not valid for the whole graph G . The anti-projection adds the deleted nodes to the inequality and increases the right hand side by value one. We obtain inequality $\sum_{i=1}^6 x_i \leq 2$ which is facet-defining for $P_{STAB}(G)$. Removing of edge v_2v_4 changes the problem completely. Consider therefore Fig. 2 (c). The computed inequality is no longer valid as the edge v_3v_5 is not strongly projectable any more. It is the central edge of a bull. We use the idea indicated by Corollary 2 and compute a (maximum) clique Q . In this case we choose $Q = \{v_2\}$. Edge v_3v_4 is removed and after the projection we obtain the graph of Fig. 2 (d). Using again the clique inequality indicated by v_1, v_2 and v_6 leads to inequality $x_1 + x_2 + x_3 + x_5 + x_6 \leq 2$ which is now valid for $P_{STAB}(G)$.

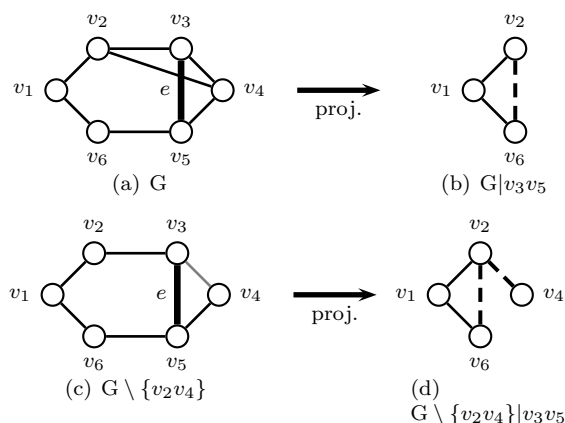


Fig. 2 Edge-projections. The dashed-line are the false edges added during the projection.

In both examples of Fig. 2 the rank inequality for $P_{STAB}(G|e)$ cannot be trivially lifted, because it is not valid for $P_{STAB}(G)$ without anti-projection. The reason is that the graph induced by the inequality contains a false edge. Clearly, a rank inequality $\sum_{v_i \in V} \beta_i x_i \leq b_0$, denoted by (β, b_0) , in the graph $G|e$, whose induced subgraph does not contain any false edge, is also valid for $P_{STAB}(G)$ without anti-projection. Observe that the anti-projected inequality is dominated by (β, b_0) and $x(\Gamma_{uv}) \leq 1$, according to Corollary 1. Hence, it is better not to anti-project the inequality (β, b_0) .

Let us now consider the case that we have found a facet-defining inequality in $G|e$, for instance a maximal clique. This is illustrated in Fig. 3 (a) and (b). The inequality after the anti-projection is not facet-defining for $P_{STAB}(G)$, as it is dominated by the two edge-inequalities $x_1 + x_4 \leq 1$ and $x_2 + x_3 \leq 1$. In addition, it is not even a necessary condition for a facet-defining rank inequality of G that the inequality on $G|e$ before the anti-projection is facet-defining. This is shown by Fig. 3 (c) and (d). The inequality in (d) is dominated by the two clique inequalities $x_1 + x_4 + x_5 \leq 1$ and $x_6 + x_7 + x_8 \leq 1$, while the inequality according to (c) is facet-defining. Let us summarize this in the following lemma.

Proof Let $\beta \in \mathbb{R}^n$, $b_0 \in \mathbb{R}$ with

$$\text{for all stable sets } \mathcal{S} \text{ in } G - \Gamma_{uv} : \chi^{\mathcal{S}}(W \cup \{uv\}) = l + 1 \implies \beta^\top \chi^{\mathcal{S}} = b_0. \quad (2)$$

We have to show that there is a constant $\xi \in \mathbb{R}$ with $\beta = \xi \cdot \chi^{W \cup \{u,v\}}$ and $b_0 = \xi \cdot (l + 1)$. This will be done in three steps.

First step. We will show that $\beta_u = \beta_v$. Let $\tilde{\mathcal{S}}$ be as assumed. Then $\tilde{\mathcal{S}} \cup \{u\}$ is a stable set on G with cardinality $l + 1$. This implies together with (2) that $\beta^\top (\chi^{\tilde{\mathcal{S}}} + \chi^{\{u\}}) = b_0$. The same can be shown for v which gives $\beta^\top (\chi^{\tilde{\mathcal{S}}} + \chi^{\{v\}}) = b_0$. These two identities imply $\beta_u = \beta_v$.

Second step. The existence of a constant $\xi \in \mathbb{R}$ with $\beta|_{V|e} = \xi \cdot \chi^W$ will be shown where $\beta|_{V|e}$ means the restriction of β to the set $V|e \subset V$. According to the definition of the projection $G|e$, the addition of one of the two nodes u or v to an arbitrary stable set \mathcal{S} in $G|e$ will lead to a stable set in G (this is only valid for one of the two nodes in general). Otherwise \mathcal{S} would contain a node $w_1 \in \Gamma(u)$ and a node $w_2 \in \Gamma(v)$ which is a contradiction as w_1 and w_2 are adjacent in $G|e$. Therefore for all stable sets $\mathcal{S} \subseteq W$ in $G|e$ with cardinality l there exists a node $w \in \{u, v\}$ with

$$(\chi^{\mathcal{S}} + \chi^w)(W \cup \{u, v\}) = l + 1.$$

This implies together with (2) that $\beta^\top (\chi^{\mathcal{S}} + \chi^w) = b_0$. With the result of the first step we get that $\beta^\top|_{V|e} \chi^{\mathcal{S}} = b - \beta_u$ for all stable sets \mathcal{S} in $G|e$ satisfying $\chi^{\mathcal{S}}(W) = l$. Since $x(W) \leq l$ is facet-defining on $P_{STAB}(G|e)$ there is a constant $\xi \in \mathbb{R}$ with $\beta|_{V|e} = \xi \cdot \chi^W$.

Third step. It will be shown that ξ from step two satisfies $\xi = \beta_u (= \beta_v)$. Let $\tilde{e} = w_1 w_2$ be a critical false edge. Then there is a stable set $\mathcal{S} \subseteq W$ in $G|e$ with size $l - 1$ and the property that after deleting \tilde{e} from $G|e$ the extension $\mathcal{S} \cup \{w_1, w_2\}$ is a stable set. It can be assumed that $w_1 \in \Gamma(u)$. Then

$$\begin{aligned} \chi^{\mathcal{S} \cup \{w_1, w_2\}}(W \cup \{u, v\}) &= l + 1 \\ \text{and } \chi^{\mathcal{S} \cup \{w_1, v\}}(W \cup \{u, v\}) &= l + 1. \end{aligned}$$

This enables us to use (2) and we obtain $\beta^\top \chi^{\mathcal{S} \cup \{w_1, w_2\}} = b$ and $\beta^\top \chi^{\mathcal{S} \cup \{w_1, v\}} = b$. Comparison of the coefficients yields the equality $\beta_{w_1} + \beta_{w_2} = \beta_{w_1} + \beta_v$ which can be simplified to $\beta_{w_2} = \beta_v$. As w_2 is in $V|e$ we get from step two that $\xi = \beta_v$.

Up to this point we have shown that there is a constant ξ with $\beta = \xi \cdot \chi^{W \cup \{u,v\}}$. As the inequality $\chi^{W \cup \{u,v\}} \leq l + 1$ is valid and the face is not empty, $b_0 = \xi \cdot (l + 1)$ and the proof is complete.

The assumptions of Proposition 1 are satisfied if W is a maximal clique in $G|e$ with size 3 or more and one of its edges \tilde{e} is a false edge. In this case, \tilde{e} is a critical edge in $G|e$ and one can choose $\tilde{\mathcal{S}} = \emptyset$. If in addition the nodes of Γ_{uv} are only adjacent to $u, v, \Gamma(u)$ or $\Gamma(v)$ the anti-projected clique inequality is facet-defining for $P_{STAB}(G)$. If $W = V|e$ induces an odd hole in $G|e$ and $E|e$ contains a false edge, then the anti-projected inequality is facet-defining for $P_{STAB}(G - \Gamma_{uv})$. For lifted odd-cycle inequalities in $G|e$ this is not generally true. There is an counterexample with only 6 nodes.

One observes that all the presented results were restricted to rank inequalities. The reason is that the anti-projection may fail for general inequalities. An example is given in Fig. 4. The graph of sub figure (a) is projected with the strongly projectable edge $v_7 v_8$. The result can be seen in sub figure (b). This graph contains one 5-cycle and the two nodes v_7, v_8 which are adjacent to all nodes of the graph. The resulting wheel inequality reads

$$\sum_{i=1}^5 x_i + 2x_6 + 2x_9 \leq 2.$$

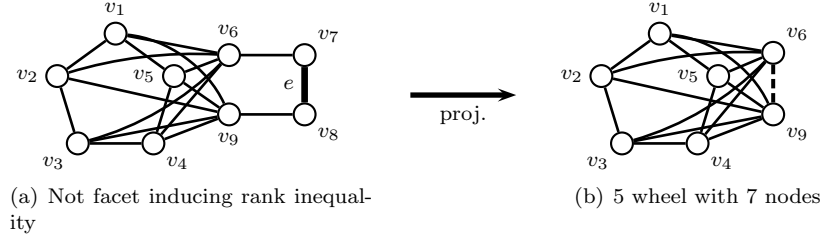


Fig. 4 Edge-projection for general inequality

The anti-projection with Theorem 1 would result in the inequality

$$\sum_{i=1}^5 x_i + 2x_6 + x_7 + x_8 + 2x_9 \leq 3.$$

This inequality is no longer valid since $\{v_6, v_9\}$ is a stable set in G which violates the inequality. With this example it is obvious that the anti-projection fails whenever there is a false edge with the property that the coefficients of both of its endnodes have a value greater than 1. The next proposition deals with these cases and extends the edge-projection to general inequalities. We define $\max \emptyset := 0$.

Proposition 2 *Let $e = uv \in E$ be a strongly projectable edge and $\sum_{v_i \in V|e} a_i x_i \leq l$ be valid for $P_{STAB}(G|e)$ and not dominated by a non-negativity facet. The inequality*

$$\sum_{v_i \in V} a_i x_i \leq l + a \text{ with } a_j := a := \max \{a_i \mid v_i \text{ is an endnode of a false edge} \}$$

for all $v_j \in V \setminus V|e$ is valid for $P_{STAB}(G)$.

Proof Without loss of generality we can assume that $a_i \in \mathbb{N}_0$ for all $v_i \in V|e$. Let \mathcal{S} be a stable set on G_c with a node-weighting c which is given through the coefficients a_i of the inequality

$$\sum_{v_i \in V} a_i x_i \leq l + a. \quad (3)$$

We construct a stable set $\tilde{\mathcal{S}}$ on $G|e$ and show that if $\tilde{\mathcal{S}}$ satisfies the inequality

$$\sum_{v_i \in V|e} a_i x_i \leq l, \quad (4)$$

then \mathcal{S} satisfies inequality (3). Therefore we consider the following cases and assume first that $\Gamma_{uv} = \emptyset$.

Case 1: $u \in \mathcal{S}$. This implies that $v \notin \mathcal{S}$ and that there exists no $v_i \in \Gamma_u$ with $v_i \in \mathcal{S}$. Hence, the set $\tilde{\mathcal{S}} := \mathcal{S} \setminus \{u\}$ is a stable set on $G|e$ with $\alpha(G_c[\tilde{\mathcal{S}}]) = \alpha(G_c[\mathcal{S}]) + a_u$. This shows that \mathcal{S} satisfies (3) if $\tilde{\mathcal{S}}$ satisfies (4).

Case 2: $u \notin \mathcal{S}$ and $\exists v_i \in \Gamma_v$ with $v_i \in \mathcal{S}$. Let $w \in \Gamma_u$ be in \mathcal{S} . If there is no such w the set \mathcal{S} is a stable set in $G|e$ and there is nothing to show. Since edge e is strongly projectable it cannot be the central edge of a double fork which implies that

$$\begin{aligned} (\mathcal{S} \setminus \{v_i, w\}) \cap \Gamma_v &= \emptyset \\ \text{or } (\mathcal{S} \setminus \{v_i, w\}) \cap \Gamma_u &= \emptyset. \end{aligned}$$

Without loss of generality let $(\mathcal{S} \setminus \{v_i, w\}) \cap \Gamma_v = \emptyset$. Then $\tilde{\mathcal{S}} := \mathcal{S} \setminus \{v_i\}$ defines a stable set in $G|e$ which shows that in this case inequality (3) is valid for \mathcal{S} .

We only have to discuss the case $\Gamma_{uv} \neq \emptyset$ as all other cases are trivial or equivalent to the two cases discussed above. Let $w \in \Gamma_{uv}$ be in \mathcal{S} . We claim that $\tilde{\mathcal{S}} := \mathcal{S} \setminus \{w\}$ is a stable set on $G|e$. First of all $\tilde{\mathcal{S}} \subseteq V|e$ as $\Gamma_{uv} \cup \{u, v\}$ is a clique, compare Lemma 1. It defines a stable set on $G|e$ because e is strongly projectable and therefore not the central edge of a bull. This closes the proof.

In case the computed inequality in the projected graph does not contain any false edge, Proposition 2 uses the observation that this inequality is also valid for the graph before projection. In this case inequalities (3) and (4) are identical, as $a = 0$. Consider again Fig. 4. The use of Proposition 2 gives $\sum_{i=1}^5 x_i + \sum_{i=6}^9 2x_i \leq 4$. This inequality is valid for $P_{STAB}(G)$ and even facet-defining.

We close this section with two remarks. Up to this point we have considered only one edge-projection but it can also be done iteratively. The theory can be adopted straight forward without any new result or restriction but it becomes a little bit technical. For all its details we refer again to Rossi and Smriglio (2001). The second remark is on the weighted stable set problem. Since the computed inequalities are valid for the stable set polytope, they can also be used by separation routines of a maximum-weight stable set problem.

2.2 Local Cuts

The principle of this method goes back to a separation routine for the traveling salesman problem introduced by Applegate et al. (2001). The proposed method is similar to the one presented by Warrior (2007). However, in contrast to our approach, the point x^* has to be an inner or interior point of the polytope.

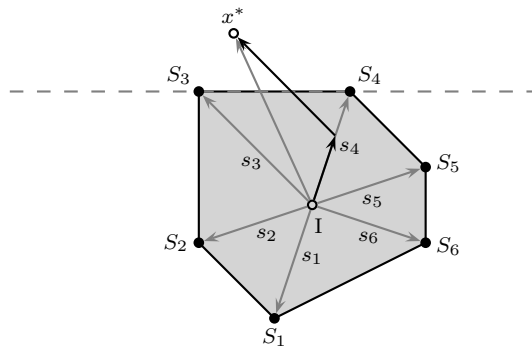


Fig. 5 Local Cut

We explain the principle with Fig. 5. If one knows all feasible solutions S_1, \dots, S_m of a particular problem, for instance the stable set problem, one could check if a vector x^* lies in its convex hull or not. If it lies outside, the aim is to compute a face (with high dimension) separating this point from the convex hull. Therefore, one has to find an interior point I of the convex hull and calculate all vectors s_j from I to the feasible solution S_j . Let i be the vector according to point I . A cone combination of $x^* - i$ with the vectors s_j provides the information,

if I is inside or outside the convex hull. Therefore, consider the following LP

$$\min \mathbf{1}^\top \lambda \qquad \qquad \qquad \max (x^* - i)^\top y \qquad (5)$$

$$\text{s.t. } (s_1, \dots, s_m)\lambda = x^* - i \qquad \text{and its dual} \qquad \text{s.t. } (s_1^\top, \dots, s_m^\top)^\top y \leq \mathbf{1} \qquad (6)$$

$$\lambda \in \mathbb{R}_+^m \qquad \qquad \qquad y \in \mathbb{R}^n. \qquad (7)$$

If $\mathbf{1}^\top \lambda > 1$, x^* lies outside the polytope. Combined with the complementary slackness, we obtain

$$y^\top (x^* - i) \leq 1 \qquad (8)$$

which defines, according to construction, a face of the polytope. The left hand side of (8) depends on i and the dual variables may be fractional. As this inequality defines a face, the coefficients have a greatest common divisor (gcd), which may be fractional, too. Finally, we get the so-called local cut

$$\tilde{y}^\top x \leq \left\lfloor \frac{1}{gcd} + \tilde{y}^\top i \right\rfloor. \qquad (9)$$

The feasible solutions S_1, \dots, S_m correspond to vertices of the corresponding polytope. This enables us to formulate the following corollary.

Corollary 3 *Given is a polytope P , all vertices S_1, \dots, S_m of P , an interior point i and a point x^* . If problem (5)–(7) has an optimal solution for x^* , then the local cut (9) is a feasible inequality for polytope P .*

Proof Assume that (3) is not valid for a point \bar{x} which lies inside the polytope P . Then,

$$\tilde{y}^\top x \leq \frac{1}{gcd} + \tilde{y}^\top i \qquad (10)$$

is also not valid. Inserting \bar{x} and multiplying by gcd yields to (8) which can be written as

$$y^\top (\bar{x} - i) > 1. \qquad (11)$$

Now, let y^* be an optimal solution to the dual (5) - (7). Then, as y is a feasible solution to the dual and the dual is a maximization problem, we get

$$y^{*\top} (\bar{x} - i) \geq y^\top (\bar{x} - i) > 1. \qquad (12)$$

Strong duality implies that for the primal problem $\mathbf{1}^\top \lambda > 1$ which contradicts the assumption that \bar{x} is valid for polytope P .

The obtained inequality (9) depends on the interior point i . Furthermore, choosing point i from the boundary of the polytope may not lead to a facet. Therefore, consider the stable set polytope for a triangle. Choosing the origin as interior point leads to an edge-inequality which is dominated by the clique inequality. Also, recognize that the primal problem (5) - (7) is not feasible for all interior points i - due to the non-negativity constraints for λ .

The complexity of this method is dominated by the enumeration of all feasible solutions and by the solution time for the linear program. Therefore, it is obvious that such a local cut (9) will not be computed for the whole graph. Hence, one has to select a subgraph and compute the cut there. This explains the name “local cut” since it is valid “locally,” for the smaller graph. To obtain a valid inequality for the polytope P , one has to lift inequality (9).

3 Branch & Cut Algorithm

In this section, we discuss the implemented Branch & Cut algorithm for the stable set problem. Some computational results are then presented in Section 4. We assume that all graphs are connected.

The implemented Branch & Cut algorithm starts with a preprocessing phase; see Section 3.1. After the preprocessing has been performed, the first LP-relaxation is set-up via a clique covering. For each node of the graph, one maximal clique containing it is constructed.

After an LP in the Branch & Bound tree has been solved, all edge-inequalities are tested to see if they are satisfied. Whenever an edge-inequality is violated, a maximal clique containing this edge is computed and added to the LP.

In general, after an LP has been solved, one can check if the solution is integer feasible. If this test fails, the separation or branching step is called. Between these steps one can use heuristics to increase the global lower bound (for a maximization problem). With the information of the LP solution, we can construct a maximal stable with the following rounding heuristics. First, put all nodes in a set S which solution vector has value 1. Then, set S is a stable set as it is assumed that solution x satisfies the edge-inequality. The neighbors of the added nodes are marked that they cannot get a member of S . Then, step by step, unmarked nodes are added to set S according to a defined weight (*i.e.*, $c_i x_i$) until only marked nodes are left.

As a first heuristics, we round the fractional LP solution until a stable set is computed. After this rounding heuristics has been executed, a so-called improving heuristics is called. For the improving heuristics, a stable set is given and the aim is to compute a stable set of larger weight (size). Therefore, three nodes in the current stable set are selected where one is labeled as ' t '. If the distance² of node t to the other two nodes is greater than 3, the paths are node disjoint, no other node of the stable set is included in these two paths and the nodes in both paths are not connected via an edge, then the stable set can be increased by exchanging the node t of the stable set by the two adjacent nodes of t in the computed paths. This procedure is repeated until a time limit is reached.

If the gap cannot be closed by the heuristics, then the separation process starts. The implemented Branch & Cut algorithm performs the following separation routines (in this order)

1. odd-hole inequalities and lifts the computed 3-cycles to maximal cliques (lifting is in linear time),
2. clique inequalities (via heuristics),
3. rank inequalities (via edge-projection),
4. mod- $\{2, 3, 5, 7\}$ cuts,
5. local-cuts, and a
6. large rank inequality.

Whenever less violated cuts than $|G|/10$ are computed, the next separation procedure in this list is called; except the last method. This is only called if none of the other separation routines can find any violated cut.

A polynomial separation algorithm for the class of odd-cycle inequalities was given by Gerards and Schrijver (1986). A necessary condition for an odd cycle inequality to be facet-defining for the stable set polytope is to be chordless. Hence, we prefer the separation of the odd-hole inequalities instead of the odd-cycle inequalities in a Branch & Cut framework. Separation of the odd-hole inequalities are given by Grötschel et al. (1988), Nemhauser and Sigismondi (1992), and Rebennack (2006). In general, a separation routine aims to find a *most* violated inequality of a certain class for a given fraction solution. However, in order to compute a solution for a stable set problem via a

² The *distance* between two nodes in a graph is the length of a shortest path connecting these two nodes.

Branch & Cut algorithm, it is more interesting to get any violated inequality, within a tolerance $\varepsilon > 0$. The odd-hole separation going back to [Gerards and Schrijver \(1986\)](#) is based on solving several shortest path problems on a modified graph ([Rebennack 2006](#), Chapter 4.2.1). This allows to obtain several odd-hole inequalities instead of only one, with the no additional computational effort. If the size of any cycle is equal to 3, we lift it to a maximal clique, since we know that the resulting inequality is facet-defining; this little ‘trick’ is easy to implement, computationally fast and also effective for the tested graphs; see Section 4.3.

The separation of the clique inequalities is itself \mathcal{NP} -hard. Therefore we use a heuristic to separate the clique inequalities. Randomly, an edge is selected and, step by step, a maximal clique containing this edge is constructed. If one is lucky, the clique inequality is violated by the current solution and it can be added to the linear program formulation.

We use the edge-projection in order to separate the rank inequalities. Notice that this separation is not exact in the sense that if the routine does not find a violated rank inequality, it is not guaranteed that non exists. As mentioned in Section 2.1, the strength of edge-projection lies in the reduction of the size of the graph and the increase of the density of the smaller graph. This gets even stronger, when the edge-projection is used iteratively. The theory does not bring anything new, but the implementation itself gets a bit tricky. As the separated inequalities have to be anti-projected, the information how this inequality is computed has to be stored. For this, we used a special tree structure where each node of the tree stores the information of one projection. Once an inequality has been found, it can be anti-projected by walking up the tree, see [Rebennack \(2006, Chapter 5.2.3\)](#).

The appealing idea of the mod- k cuts is to find a multiplier μ such that a particular inequality system $Ax \leq b$ multiplied with this vector μ can be strengthened by dividing it by a positive integer k . For more information, see [Caprara et al. \(2000\)](#), [Fricke \(2007\)](#), or [Rebennack \(2008\)](#). We apply these cuts to the stable set polytope.

In order to compute a local cut, one has to enumerate all stable sets explicitly. As the number of stable sets grows with $O(2^n)$ this cannot be done for the whole graph. Therefore, a subgraph has to be selected. We computationally observed that a size of 13 to 16 nodes for the subgraph is computationally affordable; this leads to an LP with around 10000 inequalities; depending on the density of the graph. We selection a subgraph as follows. First, select a node which LP solution has a value close to 0.5 and add it to the graph. Second, select now some nodes of its neighborhood and prefer again those nodes which LP value is close to 0.5. This is done iteratively until the wished size of the subgraph is reached.

A so-called ‘large’ rank inequality means that the edge-projection is iteratively executed until a graph of at most 75 nodes is reached. Then, this Branch & Cut solver is again used to solve the maximum-cardinality stable set problem on this smaller graph. The optimum value together with the nodes of the graph imply a rank inequality. This valid inequality is expected to have a high support after the anti-projection and is therefore called ‘large’ rank inequality.

All computed inequalities are stored in a cut pool. Each inequality is only added if it is not contained in the pool. After an LP has been solved, all inequalities which have a slack $\neq 0$ are removed from the formulation. If no cuts during the separation can be computed or if the improvement of the LP solution is not better than 10^{-4} , the branching phase starts.

The method of [Balas and Yu \(1986\)](#) is used for branching. The nodes of the induced subgraph of the current subproblem are sorted in ascending order of their degree. After that a covering of these nodes is constructed with the use of the rank inequalities stored in the pool. Step by step the inequality which maximizes the ratio of the right hand side and the number of nodes containing it is added. The Branch & Bound tree is scanned with the best-bound-first strategy.

3.1 Preprocessing

Let \mathcal{LP} denote the linear programming relaxation, containing (only) all edge-inequalities. Consider now the following theorem.

Theorem 4 (*Nemhauser and Trotter 1975; ?*) *Let x^* be an optimal $(0, \frac{1}{2}, 1)$ -valued solution of \mathcal{LP} . There is a maximum stable set in G that contains $\mathcal{S} := \{v_j \in V \mid x_j^* = 1\}$.*

In this section, we also allow for a clique the degenerated cases of one or two nodes. Imagine there is a clique Q and a node $v \in Q$, whose neighborhood is a subset of the clique. If the weight of this node is greater or equal to the weight of all other nodes of the clique, this node can be added to a stable set and the whole clique can be deleted from the graph. This is summarized in the following lemma.

Lemma 3 *Let $G_c = (V, E, c)$ be a node-weighted graph. Exists a node $v_i \in V$ and a clique Q of G with the property $\Gamma(v_i) \subseteq Q$ and $c_i \geq c_j$ for all $v_j \in Q$, then there is a maximum stable set of G which contains node v_i .*

Proof Use (Nemhauser and Trotter 1975, Theorem 1). Then node v_i is then a maximum stable set in $\hat{G}_{\{v_i\}}$.

As preprocessing, we iteratively use Theorem 4 to remove nodes from the graph which have value 1 and Lemma 3 to remove cliques from the graph, until both methods do not allow any further fixing of nodes.

4 Computational Results

In this section, we present some computational results of the implemented Branch & Cut algorithm for DIMACS benchmark graphs and uniform benchmark graphs.

The Branch & Cut algorithm is implemented in C++ with the ABACUS framework (Elf et al. 2001). We used ILOG CPLEX as the LP solver. The processor of the test run is an Intel Xeon with 2.8 GHz and 2.0 GB RAM. The CPU time limit is set to 12 hours. We tested the implemented algorithm on DIMACS Challenge benchmark graphs (DIMACS 1992/1993) and uniform random graphs (Sewell 1998, Chap. 4.1). The Branch & Cut algorithms of Rossi and Smriglio (2001) are used as benchmarks. The DIMACS machine benchmarks have shown that our computer is approximately 6.6 times faster than the one used by Rossi and Smriglio in 2001.

All computed inequalities are checked with an independent program. For each inequality a stable set problem on its induced subgraph is solved with the CPLEX mixed-integer program solver to validate its feasibility. In addition, for all problems which could be solved in the root node, an LP with the computed inequalities is solved. This LP solution always lies between the optimum value and the LP solution of the Branch & Cut solver. The difference is due to the removal of inequalities with slack zero after an LP has been solved.

Consider now Table 1 and 2. These tables contain the computational results of the implemented Branch & Cut algorithm which is in the following abbreviated by “BC”. As benchmark, we use “BC_{rank}” and “BC_{clique}” of Rossi and Smriglio (2001), separating the rank inequalities, or only the clique inequalities, respectively. In the first 4 columns there is the information about the problem instances. The density, “Den.,” is the ratio of $\frac{2|E|}{|V|^2 - |V|}$ and α is the stability number of the particular graph. The column “LB_{root}” provides the (global) lower bound in the root of the Branch & Bound tree. The next three columns give the LP solution value before branching. This is rounded down equal to the dual bound. The values of BC are stated next to the results

of the algorithm of the literature. This is followed by the information about the number of nodes in the Branch & Bound tree which is the number of subproblems. The column “# LP” gives the number of solved linear programs. The last three columns provide the needed CPU time in seconds of the three algorithms. The graphs of Table 1 are the inverse graphs of the DIMACS clique benchmark graphs.

4.1 DIMACS Graphs

Let us look at the test results for the DIMACS graphs in Table 1. The comparison of the dual bounds of the three algorithms shows that BC is always better than BC_{clique} . This is not astonishing as BC_{clique} is mainly based on clique separation which is only one separation routine in BC. However, the results of BC_{clique} can be used to evaluate the other separation routines of BC, as one could expect that the results of BC would be similar to BC_{clique} if all other separation routines of BC were turned off. For the whole algorithm, it is more interesting to compare BC with the results of BC_{rank} . Observe that in most cases the dual bounds of BC are even better than the bounds of BC_{rank} . An exception builds the two graphs “C125.9” and “C250.9.” The last graph could not even be solved to optimality by BC. The upper bound after 12 CPU hours was still 57 and the gap 25.5%. All four “brock” graphs have quite good bounds compared to BC_{rank} . The “c_fat” graphs are all solved in the root node. Especially “c_fat500-2” and “c_fat500-10” have a very good result. Interestingly, the number of computed inequalities is quite small, see Table 3. We again point out that all inequalities are checked. The LP with all computed inequalities for “c_fat500-2” has value 26.48 and the solution for “c_fat500-10” is integer feasible. The optimality proof for all “san” graphs can be done in the root node if the optimum is known. Unfortunately, the lower bounds for “san200_0.7_2,” “san200_0.9_3” and “san400_0.5_1” differ from the optimum and the branching phase has to be entered. The gaps for the “p_hat300” graphs are quite big. Nevertheless they can both be solved to optimality.

To some extent the downside of the good upper bounds is an increase in the CPU time. The separation of the mod- k cuts has no effect on the upper bounds but were quite time consuming, see Table 3. The computation of the local cuts and the large rank inequalities are also very time consuming with only very little contribution to closing the gap.

Table 1: Computational results on DIMACS graphs

Name	Instances			Branch & Cut										
				LB _{root}	LP value in root				# B & B nodes			# LP	Time	
	V	Dens.	α	BC	BC	BC _{rank}	BC _{clique}	BC	BC _{rank}	BC _{clique}	BC	BC	BC _{rank}	BC _{clique}
brock200_2	200	0.50	12	12	20.99	22.01	22.17	1002	1957	1510	2726	477	292	424
brock200_4	200	0.34	17	14	29.93	31.54	33.87	8227	8695	23457	13855	2128	661	683
brock400_2	400	0.25	29	22	63.84	67.66	70.89	161363 ^b	+++	+++	37746 ^b	45207 ^b	+++	+++
brock400_4	400	0.25	33	23	63.89	67.98	71.85	186247 ^b	+++	+++	48882 ^b	45714 ^b	+++	+++
c_fat200-1	200	0.92	12	12	12.71	12.86	12.98	1	1	1	3	1	33	27
c_fat200-2	200	0.84	24	22	24.00	24.00	24.00	1	1	1	5	1	38	25
c_fat200-5	200	0.57	58	58	58.89	65.25	66.66	1	1	42	90	16	146	92
c_fat500-1	500	0.96	14	11	14.00	14.98	14.90	1	1	1	16	323	697	1012
c_fat500-2	500	0.93	26	26	26.97	57.78	90.51	1	130	304	13	81	1038	1171
c_fat500-5	500	0.81	64	64	64.70	67.08	67.58	1	2	2	10	26	1751	1754
c_fat500-10	500	0.63	126	118	126.00	223.29	223.32	1	672	547	5	4	209	211
C125.9	125	0.10	34	34	41.26	37.40	43.15	1097	80	3136	2234	29	18	28
C250.9	250	0.10	44	43	69.76	58.30	71.91	207462 ^b	141954	+++	83612 ^b	49047 ^b	115099	+++
hamming8-4	256	0.36	16	9	16.00	16.00	16.00	1	1	1	3	1	89	89
keller4	171	0.35	11	11	14.83	14.95	14.96	990	2070	4256	2085	136	187	261

Table 1: Continued

san200.0.7_1	200	0.30	30	16	30.00	30.71	30.69	1	1	1	4	1	17	24
san200.0.7_2	200	0.30	18	14	18.50	19.18	19.24	577	38	38	347	11	11	15
san200.0.9_1	200	0.10	70	69	70.00	70.00	70.00	1	1	1	4	1	2	2
san200.0.9_2	200	0.10	60	38	60.00	60.00	60.00	1	1	1	6	1	6	6
san200.0.9_3	200	0.10	44	39	44.00	44.80	45.68	10620	1	46	5359	74	8	5
san400.0.5_1	400	0.50	13	10	13.24	17.14	27.68	228	139	194	416	348	147	315
san400.0.9_1	400	0.10	100	73	100.00	100.40	100.40	1	1	1	12	2	131	131
p_hat300-2	300	0.51	25	17	33.81	34.19	34.42	1124	686	844	2327	912	1015	973
p_hat300-3	300	0.26	36	30	54.12	53.19	55.55	28064	32756	5667	44749	11703	8226	13228

+++ Could not be solved to optimality
^b Statistics collected at CPU time limit

4.2 Uniform Random Graphs

The computational results of the uniform random graphs can be seen in Table 2. The graphs are randomly constructed where each edge is selected with a fixed probability to be in the graph. As the results of BC with BC_{rank} and BC_{clique} are not 100% comparable they are listed separately. Indeed, the average size of α differs. We recognize that the problems become harder to solve if the density grows from 0.1 to 0.2. This is confirmed by the results in literature (Rossi and Smriglio 2001; Sewell 1998; Mannino and Sassano 1996). The lower boundsturn out to be close to the optimal solution on average. However, the upper bounds are not so strong which explains the high number of subproblems and solved linear programs.

A first impression of the performance of the two heuristics is given through the value of “ LB_{root} ” in Table 1 and 2. It shows that in several cases the optimal solution is already found in the root of the Branch & Bound tree. This is very important for the total running time of the algorithm. The lower bounds not only affect the gap but also have a strong impact on the size of the branching tree. Especially for the random graphs the improvement heuristic is very effective and finds the optimal solution in many cases. For the graphs with 100 nodes and a probability of 10%, “G100-10,” in 20 out of 25 cases the improvement heuristic found the optimum. From the much harder graphs “G150-20” still 13 instances were solved by this heuristic. In all other cases the optimum has been computed via an integer solution. This changes for the DIMACS graphs. Here, the rounding heuristic performed better than the improvement heuristic.

The preprocessing of Section 3.1 fails for the presented graphs. Not a single node can be fixed. Only the two graphs “hamming6-2” and “hamming8-2” of the DIMACS graphs are solved to optimality in the preprocessing step (not shown in Table 1). It turns out that the clique fixing works for random graphs with up to 80 nodes and 0.15 density. The fixing due to the structure of the \mathcal{LP} is only effective for bipartite graphs. However, for the computation of the large rank inequality this preprocessing is useful. As the graphs have a size equal to or less than 75 nodes and the graphs have a different structure than the original graphs, some nodes can be fixed which reduces the running time.

Table 2: Computational results on uniform random graphs

Name	Instances			Branch & Cut										
				LB _{root}	LP value in root				# B & B nodes			# LP	Time	
	V	Dens.	α	BC	BC	BC _{rank}	BC _{clique}	BC	BC _{rank}	BC _{clique}	BC	BC	BC _{rank}	BC _{clique}
G100_10	100	0.10	30.9	30.2	34.93	–	–	233	–	–	405	4	–	–
G100_20	100	0.20	19.9	19.3	25.73	–	–	346	–	–	763	12	–	–
G125_10	125	0.10	34.4	33.5	41.55	–	–	1634	–	–	2274	35	–	–
G125_20	125	0.20	21.7	20.5	30.40	–	–	1766	–	–	3552	116	–	–
G150_10	150	0.10	37.2	36.1	48.05	–	–	8078	–	–	12890	350	–	–
G150_20	150	0.20	23.2	21.7	34.46	–	–	8448	–	–	13994	1211	–	–
g100_10	100	0.10	31.4	–	–	33.30	36.42	–	16	323	–	–	4	5
g100_20	100	0.20	19.2	–	–	23.13	26.66	–	26	676	–	–	32	18
g125_10	125	0.10	33.3	–	–	36.92	42.87	–	144	4046	–	–	23	75
g125_20	125	0.20	21.2	–	–	26.07	31.11	–	222	3006	–	–	123	101
g150_10	150	0.10	36.8	–	–	42.14	48.14	–	803	19160	–	–	361	702
g150_20	150	0.20	22.3	–	–	30.34	35.54	–	1121	44081	–	–	475	976

– Data not available

4.3 Generated Cuts

Now consider the generated cuts for all tested graphs which is shown in Table 3. It has already been mentioned that whenever an edge-inequality is violated, a maximal clique containing this edge is computed. The number of these cliques is given in column two. In the next column the number of odd-hole inequalities with support greater than or equal to 5 can be seen. This is followed by the number of lifted 3-cycles to maximal cliques. The number of clique inequalities computed by the clique heuristics can be seen in column “clique”. This is followed by the number of computed rank inequalities, mod- k cuts, local cuts and large inequalities. The smooth numbers in the table come from the restriction of the separation routines. For instance the clique separation is stopped if $|G|/5$ inequalities have been computed at the latest. Each inequality counts only once in Table 3, even if it is computed several times.

Let us interpret the results. One observes that the number of odd-holes with size greater than 3 is not so large compared to the total number of computed odd-cycles, which is the sum of column three and four. In most cases the odd-cycles turn out to be triangles which can be lifted to maximal cliques. To get the total number of computed clique inequalities, one has to add the numbers given in columns two, three, and five. This shows that the total number of inequalities is close to the sum of computed clique inequalities and rank inequalities. An exception is the two graphs “C125.9” and “C250.9.” The number of computed rank inequalities is quite high. One should remember that this separation is only called if the odd-cycle and clique separation could not compute enough inequalities. Not only the quantity of the generated clique and rank inequalities but also their quality is very high. Their impact in reducing the upper bound is much more important than all other generated inequalities. The maximally violated mod- $\{2, 3, 5, 7\}$ cuts separation is inefficient for the tested graphs. Only for random graphs with density 0.2 have they been helpful. Interestingly, it can be observed that the maximally violated mod- k cut separation finds violated cuts very late in the separation process. This means that if the tailing off is reduced and the instances get hard, this separation will compute cuts. However, these cuts are very weak as their effect on the LP solution is small. For the local cuts and the large rank cuts we recognize something similar. Their contribution to the reduction of the gap is very small compared to their running time.

5 Conclusions

A Branch & Cut solver for the maximum stable set problem was implemented and tested on the DIMACS benchmark and random graphs. The key computational results are the following:

- The fraction of 3-cycles in the odd-hole separation is quite large, which supports the idea of lifting. The lifting of 3-cycles is effective with regard to upper bounds. Moreover, it works very fast.
- The improvement heuristic together with the rounding heuristic produced satisfactory lower bounds.
- This research gives evidence that mod- k cuts are of very little use for the stable set problem.
- The local cuts help to reduce the upper bounds a bit but their separation is very time consuming. Hence, we suggest to use this separation routine only in the root node of the Branch & Bound tree.
- The edge-projection arguably provides good cuts and is an important part of the implemented Branch & Cut algorithm.

Further Branch & Cut solvers could focus on a combination of clique separation and edge-projection. Another result of this research is that the odd-cycle separation should be replaced by

Table 3 Number of generated cutting-planes for the tested graphs

Instances Name	# Inequalities							
	edge	hole	3-cycle	clique	rank	mod- k	local	large
brock200_2	3043	0	567	22476	23753	0	45	40
brock200_4	66426	3450	10537	52037	57399	0	121	40
brock400_2	322701	12716	138114	437211	325870	0	177	46
brock400_4	428397	15019	181752	591953	399548	0	205	23
c.fat200-1	100	0	8	400	0	0	0	0
c.fat200-2	102	0	0	800	0	0	0	0
c.fat200-5	2327	0	3092	6572	4056	0	0	0
c.fat500-1	250	0	0	3607	423	0	0	0
c.fat500-2	250	0	3	3600	0	0	0	0
c.fat500-5	504	0	51	2700	0	0	0	0
c.fat500-10	998	0	0	1199	0	0	0	0
C125.9	271	641	3	254	219	0	17	16
C250.9	861	148832	172	1434	15041	0	143	34
hamming8-4	112	0	32	488	0	0	0	0
keller4	1674	147	745	4612	11775	0	2	0
san200_0.7_1	184	0	0	549	0	0	0	0
san200_0.7_2	1849	57	82	2537	163	0	0	0
san200_0.9_1	177	0	0	217	0	0	0	0
san200_0.9_2	341	0	0	373	0	0	0	0
san200_0.9_3	922	100	0	250	0	0	0	0
san400_0.5_1	5399	4	112	61611	3699	0	0	0
san400_0.9_1	1120	0	0	1711	0	0	0	0
p.hat300-2	14093	1397	3308	34492	5761	0	109	60
p.hat300-3	351214	23087	5493	437683	31589	0	48	21
g100_10	239	241	1	87	30	0	3	0
g100_20	319	626	72	423	940	0	9	0
g125_10	303	762	3	208	117	0	16	15
g125_20	533	3194	163	781	2509	50	24	16
g150_10	367	3380	8	365	340	0	17	16
g150_20	788	12285	285	1364	8608	150	88	19

an exact separation of the triangles only, which should then be lifted to maximal cliques. As the number of the computed rank inequalities is very high, a promising idea could be to strengthen these inequalities. Since the exact lifting method is practically inefficient, a clique covering should be used to compute better coefficients. Another idea could be to generate some clique coverings and store them. They could then be used to compute a covering for the induced subgraph of the inequalities. This would lead to a (non exact) lifting in linear time. A third approach could be to combine the edge-projection with the separation of general inequalities.

Acknowledgements The authors would like to thank Rossi and Smriglio from the Univeristy of L'Aquila for their support on the separation of rank inequalities.

References

- ABACUS—A Branch And CUt Solver, Version 2.3.0, 2006. <http://www.informatik.uni-koeln.de/abacus/>.
- David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. TSP Cuts Which Do Not Conform to the Template Paradigm. *Computational Combinatorial Optimization*, LNCS 2241:157–222, 2001.
- S. Arora and S. Safra. Probabilistic Checking of Proofs; a new Characterization of NP. In *Proceedings 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13. IEEE Computer Society, Los Angeles, CA, 1992.
- Alessandro Avenali. Resolution Branch and Bound and an Application: The Maximum Weighted Stable Set Problem. *Operations Research*, 55(5):932–948, September October 2007.
- Egon Balas and Chang Sung Yu. Finding a Maximum Clique in an Arbitrary Graph. *Siam Journal on Computing*, 14(4):1054–1068, 1986.
- I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. *The Maximum Clique Problem*. Handbook of Combinatorial Optimization. Kluwer Academic Publishers, Boston, 1999.
- Sergiy Butenko. *Maximum Independent Set and Related Problems, with Applications*. PhD thesis, University of Florida, USA, 2003.
- Manoel Campelo and Ricardo C. Correa. A Lagrangian Relaxation for the Maximum Stable Set Problem. http://arxiv.org/PS_cache/arxiv/pdf/0903/0903.1407v1.pdf, 2009.
- ILOG CPLEX, Version 8.100. <http://www.ilog.com/products/cplex/>.
- Second DIMACS Challenge, 1992/1993. <http://mat.gsia.cmu.edu/challenge.html>.
- I. Dukanovich and F. Rendl. Semidefinite programming relaxations for graph coloring and maximal clique problems. *Math. Prog. B*, 109:345–365, 2007.
- Matthias Elf, Carsten Gutwenger, Michael Jnger, and Giovanni Rinaldi. Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS. *Computational Combinatorial Optimization*, LNCS 2241:157–222, 2001.
- Lars Fricke. 2007.
- Katsuku Fujisawa, S. Morito, and Miko Kubo. Experimental Analyses of the Life Span Method for the Maximum Stable Set Problem. *The Institute of Statistical Mathematics Cooperative Research Report*, 75:135–165, 1995.
- Michael R. Garey and David S. Johnson. *Computers and Intractability, A guide to the Theory of NP-Completeness*. A series of books in the mathematical sciences, Vicot Klee, Editor. W. H. Freeman and Company, 1979.
- A. M. H. Gerards and A. Schrijver. Matrices with the Edmonds-Johnson property. *Combinatorica*, 6:365–379, 1986.
- M. Grötschel, L. Lovasz, and A.Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.
- Gruber and F. Rendl. Computational experience with stable set relaxations. *SIAM J. Opt.*, 13:1014–1028, 2003.
- A. Caprara, M. Fischetti, and A. N. Letchford. On the Separation of Maximally Violated mod- k Cuts. *Mathematical Programming*, 87(1):37–56, 2000.
- C. Mannino and A. Sassano. Edge Projection and the Maximum Cardinality Stable Set Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:249–261, 1996.
- G.L. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *J. Opl Res. Soc.*, 43:443–457, 1992.
- G. L. Nemhauser and L. E. Trotter, Jr. Vertex Packings: Structural Properties and Algorithms. *Mathematical Programming*, 8:232–248, 1975.
- Manfred W. Padberg. On the Facial Structure of Set Packing Polyhedra. *Mathematical Programming*, 5:199–215, 1973.
- Steffen Rebennack. Maximum Stable Set Problem: A Branch & Cut Solver. Diplomarbeit, Ruprecht–Karls Universität Heidelberg, Heidelberg, Germany, 2006.
- Steffen Rebennack. Stable Set Problem: Branch & Cut Algorithms. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3676–3688. Springer, 2nd edition, 2008.
- Fabrizio Rossi and Stefano Smriglio. A Branch-and-Cut Algorithm for the Maximum Cardinality Stable Set Problem. *Operations Research Letters*, 28:63–74, 2001.
- E. C. Sewell. A Branch and Bound Algorithm for the Stability Number of a Sparse Graph. *INFORMS Journal on Computing*, 10(4):438–447, 1998.
- Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal Protein Structure Alignment Using Maximum Cliques. *Operations Research*, 53:389–402, 2005.

- Sven de Vries and Rakesh V. Vohra. Combinatorial Auctions: A Survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003.
- Jeffrey S. Warren and Illya V. Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. <http://ie.tamu.edu/people/faculty/Hicks/jeff.rev.pdf>, 2006.
- Deepak Warriar. *A branch, price, and cut approach to solving the maximum weighted independent set problem*. PhD thesis, Texas A&M University, 2007.
- Deepak Warriar, Wilbert E. Wilhelm, Jeffrey S. Warren, and Illya V. Hicks. A Branch-and-Price Approach for the Maximum Weight Independent Set Problem. *Networks*, 46(4):198–209, 2005.