Noname manuscript No.
(will be inserted by the editor)

Maximum-Weight Stable Sets and Safe Lower Bounds For Graph Coloring

Stephan Held · William Cook
Edward C. Sewell

Received: date / Accepted: date

Abstract The best method known for determining lower bounds on the vertex coloring number of a graph is the linear-programming column-generation technique, where variables correspond to stable sets, first employed by Mehrotra and Trick in 1996. We present an implementation of the method that provides numerically-safe results, independent of the floating-point accuracy of linear-programming software. Our work includes an improved branch-and-bound algorithm for maximum-weight stable sets and a parallel branch-and-price framework for graph coloring. Computational results are presented on a collection of standard test instances, including the unsolved challenge problems created by David S. Johnson in 1989.

Keywords graph coloring, fractional chromatic number, column generation, maximum-weight stable set, safe computations.

Mathematics Subject Classification (2000) 90-04, 90-08, 90C27 , 90C35

Stephan Held's research was supported by a postdoctoral fellowship grant from the DAAD. William Cook's research was supported by NSF Grant CMMI-0726370 and ONR Grant N00014-12-1-0030.

S. Held
Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2,
53113 Bonn, Germany
Tel.: +49 228 738740, Fax: +49 228 738771
E-mail: held@or.uni-bonn.de

W. Cook
H. Milton Stewart School of Industrial and Systems Engineering,
Georgia Institute of Technology, 765 Ferst Drive, NW, Atlanta, GA 30332-0205, USA
E-mail: bico@isye.gatech.edu

E. C. Sewell
Department of Mathematics and Statistics,
Southern Illinois University Edwardsville, Edwardsville, IL 62026, USA
E-mail: esewell@siue.edu

1 Introduction

Let $G = (V, E)$ be an undirected graph with a set V of vertices and a set E of edges. We follow the usual notation $n = |V|$ and $m = |E|$. A *stable set* is a subset $S \subset V$ composed of pairwise non-adjacent vertices, that is, $\{v, w\} \notin E$ for all $v, w \in S$. A *coloring* of G, or a *k-coloring*, is a partition of V into k stable sets $S_1, \ldots, S_k$. The minimum k such that a k-coloring exists in G is called the *chromatic number* of G and is denoted $\chi(G)$.

A *clique* is a subset $C \subset V$ composed of pairwise adjacent vertices, that is, $\{v, w\} \in E$ for all $v, w \in C$. Cliques are stable sets in the complement $\overline{G} := (V, \{\{v, w\} \in V \times V \ : \ \{v, w\} \notin E, v \neq w\})$. In a coloring the vertices in a clique C must be in different stable sets and thus $|C| \leq \chi(G)$. Consequently, the *clique number* $\omega(G)$, defined as the size of a largest clique in G, is a lower bound for $\chi(G)$. However, there are graphs with clique size two and arbitrarily high chromatic number [18]. Similarly, the *stability number* $\alpha(G)$, defined as the maximum size of a stable set in G, provides another lower bound $\lceil n/\alpha(G) \rceil \leq \chi(G)$.

Letting $\mathcal{S}$ denote the set of all maximal stables sets in G, it is well known that $\chi(G)$ is the optimal value of the following integer-programming problem (e.g. see [16])

$$\begin{aligned} \chi(G) = \min \quad & \sum_{S \in \mathcal{S}} x_S \\ s.t. \quad & \sum_{S \in \mathcal{S}: v \in S} x_S \geq 1 \qquad \forall v \in V \\ & x_S \in \{0, 1\} \ \forall S \in \mathcal{S}. \end{aligned} \tag{CIP}$$

Indeed, a solution to (CIP) with objective value k can be transformed into a k-coloring by removing each vertex from all but one of the stable sets to which it is assigned. Vice versa, by extending all stable sets of a k-coloring to become maximal stable sets, a feasible solution of (CIP) with objective value k is generated.

The optimal value, $\chi_f(G)$, of the linear-programming (LP) relaxation of (CIP)

$$\begin{aligned} \chi_f(G) := \min \quad & \sum_{S \in \mathcal{S}} x_S \\ s.t. \quad & \sum_{S \in \mathcal{S}: v \in S} x_S \geq 1 \ \forall v \in V \\ & 0 \leq x_S \leq 1 \ \forall S \in \mathcal{S} \end{aligned} \tag{CLP}$$

is called the *fractional chromatic number* of G. It defines the lower bound $\lceil \chi_f(G) \rceil$ for $\chi(G)$. In [13] it was shown that the integrality gap between $\chi(G)$ and $\chi_f(G)$ is $\mathcal{O}(\log\ n)$ wherefore it is *NP*-hard to compute $\chi_f(G)$. In fact, for all $\epsilon > 0$, approximating $\chi(G)$, $\chi_f(G)$, or $\alpha(G)$ with an approximation ratio of $n^{1-\epsilon}$ is *NP*-hard [29].

Mehrotra and Trick [16] proposed to solve (CLP) via *column generation* and, accordingly, (CIP) via *branch and price*. Their process is the most successful exact coloring method proposed to date, including impressive results

obtained recently by Gualandi and Malucelli [10] and by Malaguti, Monaci, and Toth [14]. See also Hansen, Labbé, and Schindl [11] for polyhedral studies and a comparison with an alternative packing formulation.

A correct implementation of the Mehrotra-Trick process, using modern LP software, must necessarily involve a mechanism for handling errors that can occur in floating-point computation. Indeed, LP solvers typically do not provide guarantees of either the feasibility or optimality of solutions computed for instances of LP models. To illustrate the potential problem, consider the 112,517-variable LP instance `sgpf5y6` from the Mittelmann test suite [17]. This instance has a very sparse constraint matrix and is solved easily with variants of the simplex algorithm. The difficulty is that software can produce varying results, for example, the CPLEX 12.1 code returns the optimal objective value -6425.87 when solved with the primal simplex method and -6484.47 when solved with the dual simplex method; results reported by other simplex LP solvers are listed in Steffy [23]. This example is not an instance of the fractional chromatic number, but it suggests that it would be an unsafe practice to use $\lceil \chi_f(G) \rceil$ as a lower bound on $\chi(G)$ in practical computations.

A standard response, in an implementation of Mehrotra-Trick, would be to select a fixed $\epsilon > 0$ and to use the truncated value $\lceil \chi_f(G) - \epsilon \rceil$ as a lower bound on $\chi(G)$. For a sufficiently large choice of ϵ, this method can avoid typical floating-point errors, but its correctness depends on LP software that is not guaranteed to produce solutions accurate to within the ϵ tolerance. In contrast, the focus of our study is an alternative implementation of Mehrotra-Trick that is guaranteed to produce correct results, independent of the floating-point accuracy of LP software employed in the computation; the methodology is applicable to other settings where column-generation is employed. The technique is to avoid inaccuracy by computing a numerically-safe lower bound on $\chi_f(G)$, using a floating-point LP solution as a guide. To drive the process, we also present a new combinatorial branch-and-bound algorithm to compute maximum-weight stable sets in graphs; the new method is particularly well suited for instances of the problem that arise in the Mehrotra-Trick procedure. With this safe methodology, we are able to verify results reported in previous studies as well to obtain new best-known bounds for a number of instances from the standard DIMACS test collection. In particular, we have improved previously reported results on four of the eight open `DSJCxxx` instances created by David S. Johnson in 1989, including the optimal solution of `DSJC250.9`.

The paper is organized as follows. In Section 2 we describe the Mehrotra-Trick algorithm, including the new algorithm for solving stable-set subproblems. Numerically safe computations are discussed in Section 3 and an improved process to compute lower bounds is presented in Section 4, together with a branch-and-price implementation. Experimental results are presented in Section 5.

2 Column generation

Let $\mathcal{S}' \subseteq \mathcal{S}$ contain a feasible solution to (CLP), that is, $V = \bigcup_{S \in \mathcal{S}'} S$, and consider the restricted LP problem defined as

$$\begin{aligned} \chi_f(G, \mathcal{S}') := \min \quad & \sum_{S \in \mathcal{S}'} x_S \\ s.t. \quad & \sum_{S \in \mathcal{S}' : v \in S} x_S \geq 1 \quad \forall v \in V \\ & 0 \leq x_S \leq 1 \; \forall S \in \mathcal{S}'. \end{aligned} \tag{CLP-r}$$

Let (x, π) be an optimum primal-dual solution pair to (CLP-r), where the dual solution vector $\pi = (\pi_v)_{v \in V}$ contains a value $\pi_v \in [0, 1]$ for every $v \in V$. By setting $x_S = 0$ for all $S \in \mathcal{S} \setminus \mathcal{S}'$, x can be extended naturally to a feasible solution of (CLP). Now, either (x, π) is also optimum or π is dual infeasible with respect to (CLP). In the latter case, there is a stable set $S \in \mathcal{S} \setminus \mathcal{S}'$ with

$$\pi(S) > 1, \tag{1}$$

where we use the notation $\pi(X) := \sum_{v \in X} \pi_v$ for a subset $X \subseteq V$. A stable set satisfying (1) exists if and only if the *weighted stability number*

$$\begin{aligned} \alpha_\pi(G) := \max \quad & \sum_{v \in V} \pi_v y_v \\ s.t. \quad & y_v + y_w \leq 1 \qquad \forall \; \{v, w\} \in E \\ & y_v \in \{0, 1\} \qquad \forall \; V \in V \end{aligned} \tag{MWSS}$$

is greater than one.

Note that any $\pi \geq 0$ with $\alpha_\pi(G) \leq 1$ is dual feasible to (CLP) and thus defines a lower bound $\pi(V)$ for $\chi_f(G)$. Consequently, with $\pi_v = 1/\alpha(G)$ for all $v \in V$, the aforementioned lower bound $n/\alpha(G)$ for the chromatic number is also valid for the fractional chromatic number: $n/\alpha(G) \leq \chi_f(G)$.

2.1 Finding maximum-weight stable sets

As mentioned in the introduction the maximum-cardinality stable-set problem and its weighted variant defined by (MWSS) are among the hardest combinatorial optimization problems. However, for very dense graphs, for example with edge-density

$$\rho(G) := m/(n(n-1)/2) \sim 0.9,$$

the size and number of maximal stable sets is quite low and can be enumerated. A particularly efficient way of solving (MWSS) in dense graphs is via Östergård's CLIQUER algorithm [30]. For sparse graphs CLIQUER becomes less efficient and for such instances we employ a new algorithm described below.

2.1.1 Combinatorial branch and bound

The branch-and-bound algorithm presented here uses depth-first search and adopts ideas from algorithms presented in [4,3,22,27].

A subproblem in the branch-and-bound tree consists of a lower bound, denoted LB, which is the weight of the heaviest stable set found so far, the current stable set $S = \{v_1, v_2, \ldots, v_d\}$ (where d is the depth of the subproblem in the search tree), the set of free vertices F, and a set of vertices X that are excluded from the current subproblem (which will be explained below). The goal of the subproblem is to either prove that this subproblem cannot produce a heavier stable set than the heaviest one found so far (that is, $\pi(S) + \alpha_\pi(G[F]) \leq LB$) or find a maximum-weight stable set in $G[F]$ (given a vertex set $W \subseteq V$, its induced subgraph $G[W]$ is defined as $G[W] := (W, \{\{v, w\} \in E : v, w \in W\})$).

An overview is given in Algorithm 1. The algorithm consists of a recursive subfunction MWSS_RECURSION(S, F, X) that is called with $S = \emptyset, F = V$ and $X = \emptyset$.

Algorithm 1 An Exact Maximum-Weight Stable Set Algorithm.

```
function MWSS_RECURSION(S,F,X)
    LB = max(LB, π(S));
    if F = ∅ then return;
    end if
    if ∃ x ∈ X with π_x ≥ π((S ∪ F) ∩ N(x)) then return;
    end if
    Find a weighted clique cover of G[F];
    if weight of the clique cover ≤ LB − π(S) then return;
    end if
    Determine the branch vertices F'' = {f_1, f_2, ..., f_p} ⊂ F
        using the three branching rules;
    for i = p down to 1 do
        F_i = F \ (N(f_i) ∪ {f_i, f_{i+1}, ..., f_p});
        MWSS_RECURSION(S ∪ {f_i}, F_i, X);
        X = X ∪ {f_i};
    end for
end function
MWSS_RECURSION(∅, V, ∅);
```

The algorithm uses two methods to prune subproblems. The first method works as follows. Let X be the set of vertices that have been excluded from consideration in the current subproblem because they have already been explored in an ancestor of the current subproblem (see Algorithm 1 to see how X is created). If there exists a vertex $x \in X$ such that $\pi_x \geq \pi((S \cup F) \cap N(x))$, then the current subproblem cannot lead to a heavier stable set than has already

been found. To see this, let S' be the heaviest stable set that can be created by adding vertices from F to S. Now consider the stable set $S'' = \{x\} \cup \mathrm{S}'\backslash N(x)$ created by adding x to S' and removing any of its neighbors from S'. Then

$$\begin{aligned}\pi(S'') &= \pi(\{x\} \cup S'\backslash N(x)) \\ &= \pi_x + \pi(S'\backslash N(x)) \\ &= \pi_x + \pi(S') - \pi(S' \cap N(x)) \\ &\geq \pi_x + \pi(S') - \pi((S \cup F) \cap N(x)) \\ &\geq \pi(S'),\end{aligned}$$

where the second to last inequality follows from the fact that S' is contained in $S \cup F$ and the last inequality follows from the hypothesis that $\pi_x \geq \pi((S \cup F) \cap N(x))$. Furthermore, every vertex in S'' was available when x was explored as a branch vertex, thus LB must have been greater than or equal to $\pi(S'')$ when the algorithm returned from exploring x as the branch vertex. Consequently, $LB \geq \pi(S'') \geq \pi(S')$. Hence, this subproblem can be pruned.

The second method of pruning subproblems uses weighted clique covers. A *weighted clique cover* for a set of vertices F is a set of cliques $K_1, K_2, \ldots, K_r$ together with a positive weight Π_i for each clique K_i such that $\sum_{i:f \in K_i} \Pi_i \geq \pi_f$ for each vertex $f \in F$. The *weight of the clique cover* is defined to be $\sum_{i=1}^{r} \Pi_i$. It is easy to show that $\alpha_\pi(G[F])$ is less than or equal to the weight of any clique cover of F. Hence, if a clique cover of weight less than or equal to $LB - \pi(S)$ can be found for F, then this subproblem can be pruned.

An iterative heuristic is used to find weighted clique covers, as presented in Algorithm 2. The heuristic repeatedly chooses the vertex v with the smallest positive weight, finds a maximal or maximum clique K_i that contains v, assigns the weight $\Pi_i = \pi_v$ to K_i, and subtracts Π_i from the weight of every vertex in K_i.

Algorithm 2 Weighted Clique Cover Heuristic.

$\pi'_f = \pi_f \ \forall f \in F$
$i = 0$
while $\pi' \neq 0$ **do**
  $v = \arg\min \left\{ \pi'_f : \pi'_f > 0 \right\}$
  $i = i + 1$
  Find a clique $K_i \subseteq \{u \in N(v) \cap F : \pi'_u > 0\}$
  $K_i = K_i \cup \{v\}$
  $\Pi_i = \pi'_v$
  $\pi'_u = \pi'_u - \pi'_v \ \forall u \in K_i \backslash \{v\}$
  $\pi'_v = 0$
end while

The algorithm uses three branching rules to create subproblems. The first two rules adopt a weighted variation of a technique employed by Balas and Yu [5,4]. Suppose that $F' \subseteq F$ and it can be proved that

$$\alpha_\pi \left(G \left[F' \right] \right) \leq LB - \pi \left(S \right),$$

then a stable set containing S that is heavier than LB must contain at least one vertex from $F'' = F \backslash F'$. Therefore, we branch into all possibilities of extending S by a vertex from F'', avoiding multiple enumerations of the same sets. To this end let $F'' = \{f_1, f_2, \ldots, f_p\}$ and

$$F_i = F \backslash \left(N \left(f_i \right) \cup \{f_i, f_{i+1}, \ldots, f_p\} \right).$$

If $\alpha_\pi \left(G \left[F \right] \right) > LB - \pi \left(S \right)$, then

$$\alpha_\pi \left(G \left[F \right] \right) = \max_{i=1,\ldots,p} \pi_{f_i} + \alpha_\pi \left(G \left[F_i \right] \right).$$

Hence, one branch is created for each set $F_1, \ldots, F_p$. Note, if $\alpha_\pi \left(G \left[F \right] \right) \leq LB - \pi \left(S \right)$, the maximum-weight stable set in $G[F]$ might consist of vertices from F' only and the above equality might be false, but in this case no better stable set can be found in the current branch.

The first branching rule uses the weighted clique cover to create F'. The clique cover heuristic is halted as soon as the weight of the clique cover would exceed $LB - \pi \left(S \right)$. Then F' is defined as the set of vertices whose remaining weight is zero (that is, $F' = \left\{ f \in F : \pi'_f = 0 \right\}$) and $F'' = F \backslash F'$.

The second branching rule uses a method similar to the first method of pruning. If there exists a vertex $x \in X$ such that $\pi_x \geq \pi \left(S \cap N \left(x \right) \right)$, then it can be shown that if $\alpha_\pi \left(G \left[F \right] \right) > LB - \pi \left(S \right)$, then every maximum-weight stable set in $G \left[F \right]$ must contain at least one neighbor of x that is in F. The proof is similar to the proof for the first method of pruning. In such a case, F'' is set equal to $N \left(x \right) \cap F$.

The third branching rule searches for a vertex $f \in F$ such that $\pi_f \geq \pi \left(F \cap N \left(f \right) \right)$. If such a vertex exists, it is easy to prove that there exists an maximum-weight stable set of $G \left[F \right]$ that includes f, hence a single branch is created (that is, $F'' = \{f\}$).

The algorithm uses the rule that generates the smallest F'' (ties are broken in favor of the first rule and then the third rule). For both the second and the third branching rules, the set of vertices F'' are sorted in increasing order of their degree in $G \left[F \right]$.

In the context of column generation the running time can be reduced further because the actual maximum-weight stable set need not necessarily be found. Instead, it is sufficient to either find a stable set S with $\pi(S) > 1$ or decide that no such set exists. This gives two effective ways to decrease the number of branches that are processed during the course of Algorithm 1.

Firstly, LB can be initialized as 1, because only solutions of value bigger than one are of interest, while the exact value and elements of a maximum-weight stable set S are discarded if $\pi(S) \leq 1$. Secondly, it is sufficient to stop the algorithm once a stable set S with $\pi(S) > 1$ is found.

With these two modifications the running time of Algorithm 1 can be reduced significantly, for example, from almost 3 hours to 7 seconds for solving the final maximum-weight stable set instance on `3-Insertions_5` from the DIMACS benchmarks.

2.1.2 Heuristics

Within the column-generation process, a stable set with $\pi(S) > 1$ can often be found by heuristic methods. The heuristics we use create an initial solution by a greedy strategy and then improve this solution with local search. The greedy algorithms build a stable set S by starting with an empty set and adding vertices one by one. A vertex $v \in V \setminus S$ is added to S if $S \cup \{v\}$ is a stable set. Mehrotra and Trick proposed to traverse the vertices in non-decreasing order of their weight [16]. Another strategy is to consider the weight of a vertex related to the weights of its neighbors. The surplus of a vertex $v \in V$ is defined as $\pi_v^+ = \pi_v - \sum_{w \in N(v)} \pi_w$. If a preliminary stable set S is already found, the surplus can be refined by neglecting neighbors of S that are not eligible for insertion. We use the following three greedy orderings: as the next vertex, try a not yet processed vertex $v \in V \setminus (N(S) \cup S)$ for which

1. π_v (maximum weight strategy)
2. $\pi_v - \sum_{w \in N(v) \setminus N(S)} \pi_w$ (maximum dynamic surplus strategy)
3. $\pi_v - \sum_{w \in N(v)} \pi_w$ (maximum static surplus strategy)

is maximum.

The result of the greedy algorithm is then improved by local search. First, we perform $(1,2)$-swaps, where a vertex $v \in S$ is replaced by two neighbors $w_1, w_2 \in N(v)$ if $\pi_{w_1} + \pi_{w_2} > \pi_v$. This is done until no more $(1,2)$-swaps can be found. Using efficient data structures developed in [1] it can be tested in linear time whether $(1,2)$-swaps exists. Then, we perform $(2,k)$-swaps, where two vertices $v_1, v_2 \in S$ are replaced by $k \geq 1$ vertices $w_1, \ldots, w_k \in N(v) \cup N(w)$ with $w_1 \in N(v) \cap N(w)$ if $\sum_{i=1}^{k} \pi(w_i) > \pi(v_1) + \pi(v_2)$.

If the greedy algorithm followed by local search does not find a stable set of weight greater than one, then we perform several additional searches using the same strategy but a modified greedy order in which the first considered vertex is skipped and appended to the tail of the list of vertices.

This method is similar to the tabu search in [14]; it differs in not starting from random stable sets but greedy stable sets and by more general swaps that allow the addition of k vertices instead of just one vertex in a swap.

Note that one could also try further heuristics, e.g. the merging of two stable sets into a new one by a maximum-flow computation as described by Balas and Niehaus for cliques [6], or by using the QUALEX-MS algorithm of Busygin [8]. However, as we observed that on almost all instances having a stable set heavier than one the above greedy procedure would find it, we did not try to improve it further.

2.1.3 Hybrid maximum-weight stable-set solver

The overall approach to solve the maximum-weight stable-set problem is summarized in Algorithm 3. On sparse graphs most stable sets are found by the greedy algorithms, although the solvability and overall running times depend heavily on Algorithm 1.

Algorithm 3 MWSS-Wrapper

```
if ρ(G) < 0.8 then                                   ▷ sparse graphs
    for i = 1 → 3 do
        S ← Run greedy strategy i.
        if π(S) > 1 then return S
        end if
    end for
    Run Algorithm 1
else                                                 ▷ dense graphs
    Run CLIQUER
end if
```

Most DIMACS instances have either a density of 0.5 or less or a density of approximately 0.9. Only on the latter does CLIQUER outperform Algorithm 1. The threshold of 0.8 for separating sparse and dense graphs in Algorithm 3 was chosen arbitrarily between 0.5 and 0.9.

3 Numerically safe bounds

Up to this point we have assumed that all computations are performed in exact arithmetic, but competitive LP codes for solving (CLP-r) use floating-point representations for all numbers. This causes immediate difficulties in the column-generation process. Indeed, let π^{float} denote the vector of dual variables in floating-point representation as returned by an LP-solver. Based on these inexact values, $\alpha_\pi(G) > 1$ can hardly be decided and this can lead to premature termination or to endless loops (if the same stable set is found again and again).

One way to circumvent these problems would be to solve (CLP-r) exactly, for example with a solver such as [2]. However, exact LP-solvers suffer significantly higher running times, and in column generation, where thousands of restricted problems must be solved, these solvers would be impractical. Thus, instead of computing $\chi_f(G)$ exactly, we compute a numerically-safe lower bound $\underline{\chi_f}(G)$ in exact integer (fixed point) arithmetic, where the floating-point variables π^{float} serve only as a guide.

Recall that any vector $\pi \in [0,1]^n$, with $\alpha_\pi(G) \leq 1$ is a dual feasible solution of (CLP) and defines a lower bound regardless whether it is optimum or not.

Accordingly, given a scale factor $K > 0$, a vector $\pi^{int} \in \mathbb{N}_0^{V(G)}$ proves the lower bound $K^{-1}\pi^{int}(V)$ if and only $\alpha_{\pi^{int}}(G) \leq K$.

Now, the goal is to conduct the maximum-weight stable-set computations with integers $\pi_v^{int} := \lfloor K\pi_v^{float} \rfloor$ $(v \in V)$. Thus, achieving a lower $\frac{n}{K}$-approximation of $\pi_v^{float}(V)$:

$$\pi_v^{float}(V) - \frac{n}{K} \leq \frac{1}{K}\pi_v^{int}(V) \leq \pi_v^{float}(V). \tag{2}$$

The question is how to represent the integers π_v^{int} $(v \in V)$ and how to choose K. For performance reasons, it is preferable to use integer types that are natively supported by the computer hardware, for example 32- or 64-bit integers in two's complement for current x86 processors.

More generally, let us assume that all integers are restricted to an interval $[I_{\min}, I_{\max}]$ with $I_{\min} < 0$ and $I_{\max} > 0$. To avoid integer overflows, we have to ensure that during the computations of maximum-weight stable sets the intermediate results neither fall below $I_{\min}$ nor exceed $I_{\max}$. The smallest intermediate results occur while computing surpluses with the greedy strategies 2 and 3. The largest intermediate results are either given by $\pi^{int}(X)$ for some $X \subset V$ or as the weight of the weighted clique covers in Algorithm 1. As $\pi_v^{float} \in [0, 1]$ $(v \in V)$, setting $K := \min\{-I_{\min}, I_{\max}\}/n$ guarantees that any intermediate result will be representable within $[I_{\min}, I_{\max}]$. Note that the dual variables returned as floating point numbers by the LP solver might exceed the interval $[0, 1]$ slightly, although this would be dual infeasible (as all vertices are covered by at least one stable set). Thus, they are shifted into $[0, 1]$ before scaling.

By (2) the deviation from the floating-point representation of the fractional chromatic number is at most $n^2/\min\{-I_{\min}, I_{\max}\}$. Note that the denominator grows exponentially in the number of bits that are spent to store numbers, allowing a reduction in the error without much memory overhead.

Column generation creating a safe lower bound is summarized in Algorithm 4. Initially, a coloring is determined with the greedy algorithm DSATUR [7]. It provides the initial set $\mathcal{S}'$ and an upper bound for $\chi(G)$. The column-

Algorithm 4 Column Generation for Computing $\underline{\chi_f}(G)$

$\mathcal{S}' \leftarrow$ Compute initial coloring (DSATUR).
$S \leftarrow \emptyset$
repeat
  $\mathcal{S}' \leftarrow \mathcal{S}' \cup S$
  $\pi^{float} \leftarrow$ Solve (CLP-r) in floating-point arithmetic
  $\pi^{int} \leftarrow \lfloor K\pi^{float} \rfloor$
  $(S, \alpha_{\pi^{int}}(G)) \leftarrow$ Algorithm 3
until $\alpha_{\pi^{int}}(G) \leq K$
$\underline{\chi_f}(G) \leftarrow K^{-1}\pi^{int}(V)$

generation process terminates when $\alpha_{\pi^{int}}(G) \leq K$ with a lower bound of $\underline{\chi_f}(G) := K^{-1}\pi^{int}(V) \leq \chi_f(G)$.

Note that it is difficult to bound the difference $\chi_f(G) - \underline{\chi_f}(G)$ without further assumptions on the LP solver. However, a close upper bound $\overline{\chi_f}(G)$ for $\chi_f(G)$ can be computed by solving the final restricted LP (CLP-r) once in exact arithmetic [2]. Thereby, an interval $[\underline{\chi_f}(G), \overline{\chi_f}(G)]$ containing $\chi_f(G)$ can be determined, allowing us to obtain the precise value of $\lceil \chi_f(G) \rceil$ on most test instances.

4 Improved computation of lower bounds

4.1 Decreasing dual weights for speed

If the weight of a maximum-weight stable set in Algorithm 4 is slightly larger than the scale factor K from Section 3, it can potentially be reduced to K, or less, by decreasing the integer variables π^{int}. This way an earlier termination of the column-generation approach might be possible, because $K^{-1}\pi^{int}$ would become dual feasible for (CLP). Of course, such reduced weights will impose a lower fractional bound. However, the entries of π^{int} can be reduced safely by a total amount of

$$\begin{aligned} \operatorname{frac}(\pi^{int}, K) &:= \max\{k \in \mathbb{N}_0 \ : \ \lceil K^{-1}(\pi^{int}(V) - k)\rceil = \lceil K^{-1}\pi^{int}(V)\rceil\} \\ &= \max\left\{0, \left(\textstyle\sum_{v\in V} \pi_v^{int} - 1\right)\right\} \mod K, \end{aligned} \tag{3}$$

while generating the same lower bound of $\lceil K^{-1}\pi^{int}(V)\rceil$. The difficulty is to decide how to decrease entries in π_v^{int}. Ideally, one would like to achieve a largest possible ratio between the reduction of the value of the maximum-weight stable set and the induced lower bound for the chromatic number.

Gualandi and Malucelli [10] proposed a *uniform rounding* style, rounding down all values π_v^{int} $(v \in V)$ uniformly by $\operatorname{frac}\left(\pi^{int}, K\right)/n$. This way the weight of a stable set $S \in \mathcal{S}$ decreases by $\frac{|S|}{n} \operatorname{frac}(\pi^{int}, K)$.

An alternative technique works as follows. Consider a $v \in V$ with $\pi_v > 0$, then at least one vertex from $V' := v \cup \{w \in N(v) \ : \ \pi_w > 0\}$ will be contained in a maximum-weight stable set. Thus, to reduce the value of the maximum-weight stable set, it is sufficient to reduce weights in V' only. In our implementation, we always select a set V' of smallest cardinality. We refer to this rounding style as *neighborhood rounding*.

Table 1 demonstrates the importance of rounding for some instances from the DIMACS benchmark set, covering several instance classes. It reports the number of calls of the exact maximum-weight stable-set solver (Algorithm 1) needed to terminate column generation, in column 4 without any dual weight reduction (beyond safe weights according to Section 3), in column 5 with uniform rounding, and in column 6 with neighborhood rounding. However, neither of the two dual variable reduction styles dominates the other.

Table 1 Number of calls to Algorithm 1 depending on decreased dual weights

Instance	$\|V\|$	$\|E\|$	# calls to Algorithm 1		
			None	Uniform	Neighborhood
`latin_square_10`	900	307350	1	1	1
`queen16_16`	256	12640	1	1	1
`1-Insertions_6`	607	6337	8354	147	211
`DSJC250.1`	250	3218	50	1	1
`DSJC250.5`	250	15668	34	10	17
`DSJC500.5`	500	62624	106	42	38
`flat300_28_0`	300	21695	32	6	3
`myciel7`	191	2360	186	111	19

For most instances the total running time of column generation is dominated by the time spent to solve the maximum-weight stable-set problems exactly.

4.2 Branch and price

The lower bound can be improved further by branch and bound, eventually yielding an optimum coloring. As in [16] we pick two vertices $v, w \in V$ with $\{v, w\} \notin E$ and create the two coloring problems

- $G_{DIFF} := (V, E \cup \{v, w\})$, enforcing different colors for v and w, and
- $G_{SAME} := G[\{v, w\}]$, enforcing the same color for v and w.

Mehrotra and Trick [16] choose v from a most fractional column $S_1 \in \mathcal{S}$, then select a second column $S_2 \in \mathcal{S}$ covering v and choose w from $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$.

In our computations we adopt an alternative choice of v and w. For each pair $v, w \in V$, define

$$p(v,w) := \frac{\sum_{S \in \mathcal{S}: v,w \in S} x_S}{\frac{1}{2}\left(\sum_{S \in \mathcal{S}: v \in S} x_S + \sum_{S \in \mathcal{S}: w \in S} x_S\right)}. \tag{4}$$

Note that $p(v, w) \in [0, 1]$ is well defined. By (CLP), both sums in the denominator are greater than or equal to one. In our experimental runs, the denominator is equal to one for almost all instances, as it would be the case if x would define an integral coloring.

The rationale behind our branching rule is that if $p(v, w)$ is close to 0, then the current solution assigns quite different fractional colors to v and w. If $p(v, w)$ is close to one, then the two vertices are assigned nearly-equal fractional colors. In both cases, one of the branches G_{DIFF} and G_{SAME} will likely give a similar bound as their parent node. In contrast, if $p(v, w)$ is more fractional, that is, close to 0.5, it is not clear whether v and w should end up in a common stable set. Empirically, by performing strong branching on all pairs on several

instances, we found it slightly favorable to consider 0.55 as "most fractional". Our tests indicate that the new rule does not dominate the performance of the original method of Mehrotra and Trick, suggesting further work is needed in this direction.

As we are focusing on lower bounds, we use the best-first-search branching strategy. That is we always branch on that open leave node in the branch and price tree that determines the lowest fractional lower bound π^{int}/K. Ties are broken arbitrarily.

When running the column generation in a sub-node of the branch-and-bound tree, we do not compute the restricted set of stable sets $\mathcal{S}'$ from the scratch. Instead, we initialize $\mathcal{S}'$ with the basic stable sets at the parent node, that is for which the primal variables are non-zero. Of course, they may not be valid for the child nodes, but can easily be modified. In the G_{DIFF}-branch we remove w from S if both v and w occur in a common stable S, while ensuring that w will occur in at least one stable set, adding one extra set $\{w\}$ if necessary. In the G_{SAME}-branch, we remove occurrences of v and w from each set, replacing them by the new contraction node if this results in a stable set in G_{SAME}. Again, if the new node remains uncovered we add one extra singular set containing it.

5 Experimental results

The described algorithms were implemented in the C programming language; our source code is available online [12]. The LP problems (CLP-r) are solved with Gurobi 3.0.0 in double floating-point precision. Experiments were carried out on the DIMACS graph-coloring instances [26], using a 2.268 GHz Intel Xeon E5520 server, compiling with gcc -O3. For comparability with past and future work see Table 5 row "Coloring" for how this machine performs on the DFMAX benchmarks from `http://mat.gsia.cmu.edu/COLOR04/BENCHMARK`. To compute $\overline{\chi_f}(G)$ by solving (CLP-r) exactly we used the exact LP-solver QSopt_ex [2].

5.1 Results of column generation

We were able to compute $\underline{\chi_f}(G)$ and $\overline{\chi_f}(G)$ for 119 out of 136 instances, limiting the running time for computing $\underline{\chi_f}(G)$ to three days per instance. Solving (CLP-r) exactly can be quite time consuming, for example, on `wap02a` it takes 34 hours, compared to 10 minutes using doubles (QSopt_ex first solves the problem in floating-point arithmetic). This demonstrates that the use of an exact LP-solver for every instance of (CLP-r) would be impractical. As we compute $\overline{\chi_f}(G)$ only for the academic purpose of estimating the differences $\chi_f(G) - \underline{\chi_f}(G)$, we do not report its running times from here on, but only those for computing $\underline{\chi_f}(G)$.

For the 119 solved DIMACS instances the distance to the upper bound $\overline{\chi_f}(G)$ was very small. In fact, it turned out that $\lceil \underline{\chi_f}(G) \rceil = \lceil \overline{\chi_f}(G) \rceil$ for all these instances. Thus, we obtained safe results for $\lceil \underline{\chi_f}(G) \rceil$. But there were many instances where $\overline{\chi_f}(G) < \pi^{float}(V)$, and the floating-point solutions implied by the LP-solver would have been wrong. However, we did not find previously reported results for $\lceil \chi_f(G) \rceil$ that were incorrect.

Here, we focus on those instances for which the chromatic number is or was unknown. Table 2 shows the results on these instances.

Table 2 Computational results on open benchmarks

Instance	$\|V\|$	$\|E\|$	$\lceil \underline{\chi_f}(G) \rceil$	$\omega(G)$	old LB	old UB	Time (seconds)
DSJC250.5	250	15668	26	12	26[10]	28[10]	18
DSJC250.9	250	27897	71	42	71[10]	72[10,15]	8
DSJC500.1	500	12458	*	5	5[19]	12[21]	*
DSJC500.5	500	62624	**43**	13	16[14]	48[21]	439
DSJC500.9	500	224874	123	54	123[10,14]	126[21]	100
DSJC1000.1	1000	49629	*	6	6[19]	20[21]	*
DSJC1000.5	1000	249826	**73**	14	15[19]	83[21]	142014
DSJC1000.9	1000	449449	215	63	215[10]	222[25]	5033
r1000.1c	1000	485090	96	89	96[10]	98[10]	2634
C2000.5	2000	999836	*	16	16	146[28]	*
C4000.5	4000	4000268	*	≥ 17	17	260[28]	*
latin_square_10	900	307350	90	90	90[19]	97[24]	76
abb313GPIA	1557	65390	8	8	8[19]	9[14]	3391
flat1000_50_0	1000	245000	**50**	14	14	50[10]	3331
flat1000_60_0	1000	245830	**60**	14	14	60[10]	29996
flat1000_76_0	1000	246708	**72**	14	14	82[10]	190608
wap01a	2368	110871	41	41	41[19]	43[14]	20643
wap02a	2464	111742	40	40	40[19]	42[14]	236408
wap03a	4730	286722	*	40	40[19]	47[14]	*
wap04a	5231	294902	*	40	40[19]	44[14]	*
wap07a	1809	103368	40	40	40[19]	42[14]	25911
wap08a	1870	104176	40	40	40[19]	42[14]	18015
1-Insertions_6	607	6337	4	2	4[19]	7[19]	1167
3-Insertions_5	1406	9695	3	2	3[19]	6[19]	6959

(The column Time reports the running times for computing $\lceil \underline{\chi_f}(G) \rceil$)

Columns 2 and 3 give the number of vertices and edges, column 4 shows $\lceil \chi_f(G) \rceil$ from our computations, where bold numbers are those where we could improve best-known lower bounds. Column 5 shows the clique numbers from the literature or computed with CLIQUER, columns 6 and 7 summarize the best lower and upper bounds that can be found in the literature [10,15,14, 19,20,21,25,24,28]. The last column shows the running time for computing $\underline{\chi_f}(G)$.

For the instances

DSJC500.5, DSJC1000.5, flat1000_50_0, flat1000_60_0, flat1000_76_0, wap01a, wap02a, wap07a, wap08a, 1-Insertions_6, and 3-Insertions_5

Table 3 Improvements by Branch and Price

Instance	LB	UB	B&B nodes	Time (seconds)
DSJC250.9	**72**	72	3225	11094
DSJC1000.9	**216**	222	29	12489

we could compute $\lceil \chi_f(G) \rceil$ for the first time and, thereby, improve known lower bounds on DSJC500.5, DSJC1000.5, flat1000_50_0, flat1000_60_0, and flat1000_76_0 significantly.

On flat1000_50_0 and flat1000_60_0, $\lceil \chi_f(G) \rceil$ proves the optimality of known upper bounds. For latin_square_10, abb313GPIA, and all wap*a instances except for wap03a and wap04a, which did not finish, the fractional chromatic number did not improve lower bounds imposed by the clique number, that is $\omega(G) = \chi_f(G)$. Here cliques of size $\omega(G)$ can be computed within seconds by heuristics from Section 2.1.2 on the complement graphs. [30]

On most instances that are not listed $\chi_f(G)$ is computed much faster than within three days. The geometric mean of the running times of the 119 solved instances is 6.5 seconds. 17 DIMACS instances were not finished within three days. These are

- the Leighton graph instances le450_5a, le450_5b, le450_5c, le450_5d, le450_15a, le450_15b, le450_15c, le450_15d, le450_25c, and le450_25d. On these instances column generation stalls, generating more and more columns without changing the value of (CLP-r). However, the clique number $\omega(G)$ can be computed within seconds with CLIQUER [30] matching upper bounds of the instances [9] and, thus, $\omega(G) = \chi_f(G) = \chi(G)$ for these instances.
- the large instances DSJC500.1, DSJC1000.1, C2000.5, and C4000.5. On these instances the maximum-weight stable-set problems that need to be solved exactly become too numerous and too difficult.
- the very large instances wap03a, wap04a, and qg.order100. Here, the LP problems (CLP-r) become very large and dense. With state-of-the-art LP-solvers column generation would take months to terminate. On qg.order100, the 100×100 latin square , $\omega(G) = 100$ can be computed in seconds and provides a tight lower bound.

5.2 Results of branch and price

For the twenty-four open benchmark instances listed in Table 2, we attempted to improve the lower bounds by branch and price as described in Section 4.2, allowing a time limit of three days. Table 3 shows two instances for which the lower bound could be lifted by one, proving optimality of the known upper bound for DSJC250.9.

We also did more extensive runs on up to 60 parallel processors for several days, but for most of the instances branch and price did not help much. A

Table 4 Lower bounds $\lceil \chi_f(G[X]) \rceil$ from induced subgraphs

Instance	$\lvert X \rvert$	$\lvert E(G[X]) \rvert$	$\lceil \chi_f(G[X]) \rceil$	old LB	UB	Time
DSJC500.1	300	5436	**9**	5	12	16 days
DSJC1000.1	350	8077	**10**	6	20	< 36 days
C2000.5	1000	261748	77	16	148	< 1 days
C2000.5	1250	403289	91	16	148	< 11 days
C2000.5	1400	502370	**99**	16	148	< 24 days
C4000.5	1000	268033	80	17	271	< 1 days
C4000.5	1500	589939	**107**	17	271	< 26 days
wap03a	2500	164008	40	40	47	< 3 days
wap04a	2500	159935	40	40	44	< 1 days

major problem is that the fractional lower bounds increase by smaller and smaller amounts. For the promising instance DSJC250.5 (with gap 2), no child could be pruned after evaluating 916,877 branch-and-bound nodes. Also the integral lower bound did not improve.

5.3 Results on dense subgraphs

As already noted in Section 5.1, for 17 very large DIMACS instances we were not able to compute $\lceil \chi_f(G) \rceil$. For 11 of these instances, $\omega(G)$ is easy to compute and yields a tight lower bound. For each of the remaining six instances DSJC500.1, DSJC1000.1, C2000.5, C4000.5, wap03a, and wap04a the gap between the published lower and upper bounds is particularly large.

However, on these instances column generation can still be applied if restricted to tractable subgraphs. It is easy to see that for any subgraph $G[X]$ induced by $X \subset V$ (see Section 2.1.1), $\chi_f(G[X]) \leq \chi_f(G)$ and, thus, $\lceil \chi_f(G[X]) \rceil$ imposes a lower bound for $\chi(G)$.

The set X should be chosen such that $\lceil \chi_f(G[X]) \rceil$ is large, but still solvable. For the first goal a dense subgraph $G[X]$ would be favorable. We use a simple greedy strategy that starts with $X = V$ and iteratively deletes a vertex of minimum degree until $|X|$ has a given size.

Table 4 shows the lower bounds, we could obtain this way. Columns 2 and 3 give the sizes of the induced subgraph. Column 4 reports the lower bounds obtained from the subgraphs, while column 5 reports previously published lower bounds, corresponding to the respective maximum clique numbers.

From the unfinished column generation runs on the full instances, the expected lower bounds are at most 10 (DSJC500.1), 17 (DSJC1000.1), 128 (C2000.5), 253 (C4000.5), 47 (wap03a), and 45 (wap04a). This gives an indication of the loss incurred when using subgraphs.

Table 5 DFMAX benchmark results for the calibration of running times

Machine	r100.5	r200.5	r300.5	r400.5	r500.5
Coloring	0.00	0.04	0.36	2.21	8.43
MWSS	0.00	0.01	0.16	1.02	3.89

(see DIMACS challenge: `http://mat.gsia.cmu.edu/COLOR04/BENCHMARK`)

5.4 Maximum-weight stable set results

Finally, we demonstrate the efficiency of Algorithm 1 for solving maximum-weight stable set problems in Table 6. We compared the new algorithm with the branch and cut solvers Gurobi 3.0.0 and CPLEX 12.2, as well as CLIQUER 1.21 [31], which solved the maximum-weight clique problems in the complement graphs. Gurobi and CPLEX are among the fastest branch and cut solvers for integer programming. These experiments were carried out on a different machine than the coloring experiments, namely a 3.33 GHz Intel Xeon W5590 server. See Table 5 row "MWSS" for calibration results.

As test instances we used maximum-weight stable set instances as they occurred on the DIMACS coloring benchmarks during the course of Algorithm 4, when the greedy algorithm failed to provide a stable set of weight greater than one. Where possible, we picked the final instance where the exact solution terminated the column generation and, thus, determined $\lceil \chi_f(G) \rceil$. These are the instances where the name matches that of the coloring instance in the table.

For large instances, where $\lceil \chi_f(G) \rceil$ could not be computed, we took some instance that occurred after at least several hundred rounds of column generation, and where the Greedy Algorithm was unable to find an improving stable set. The complete set of instances can be obtained at

`http://code.google.com/p/exactcolors/wiki/MWISInstances`.

We performed two experiments per instance and solver. First, we computed the maximum-weight stable set as is. These runs are represented by the columns labeled STD. Second, in the columns labeled LB, we used the solvers in the same setting as in Algorithm 4. Here, the maximum-weight stable set solvers were called with an initial lower bound of $LB = 1$ (respectively, the corresponding scaled integer K defined in Section 3) to enable more efficient pruning of branches in the branch and bound trees. Furthermore, we did stop the computation once a solution greater than LB was found. However, the second stopping criterion applies only to the four large instances `C2000.5.1029`, `DSJC1000.1.3915`, and `DSJC500.1.117`.

Table 6 shows the instance names, edge densities $\rho(G)$ (see Section 2.1), and the respective running times in seconds on those instances where the maximum running time of at least one solver was more than one minute. So all instances that could be solved within a minute by all solvers are omitted. If an entry in Table 6 consists of three stars (***), this means that the instance could not be solved within ten hours of running time, a hard limit we set for all runs.

Table 6 Running times of various MWSS solvers on hard instances in seconds.

Instance	$\rho(G)$	Gurobi 3.0.0		CPLEX 12.2		CLIQUER 1.21		Algorithm 1	
	in %	STD	LB	STD	LB	STD	LB	STD	LB
1-Insertions_6	3.4	1250	1005	2969	2065	***	***	61	65
2-Insertions_4	4.9	2	2	1	1	70	70	1	1
2-Insertions_5	2.2	25	32	7	1	***	***	115	1
3-Insertions_4	2.6	2	2	1	1	***	***	1	1
3-Insertions_5	0.9	31	26	4	1	***	***	10576	7
4-Insertions_4	1.5	3	4	3	1	***	***	14	1
C2000.5.1029	50	***	***	***	***	***	30586	***	11373
DSJC1000.1.3915	9.9	***	***	***	***	***	***	***	***
DSJC1000.5	50	***	***	***	***	1076	1057	3634	3547
DSJC1000.9	89.9	***	***	***	***	1	1	2	2
DSJC250.1	10.3	31278	34901	***	16288	***	***	5941	2281
DSJC250.5	50.3	1825	1963	2737	2557	1	1	1	1
DSJC250.9	89.6	1382	1442	319	317	1	1	1	1
DSJC500.1.117	9.9	***	***	***	***	***	***	***	***
DSJC500.5	50.1	***	***	***	***	9	9	32	32
DSJC500.9	90.1	***	***	24318	22105	1	1	1	1
DSJR500.1c	97.2	86	88	17	17	1	1	1	1
flat1000_50_0	49	***	***	***	***	50	52	765	752
flat1000_60_0	49.2	***	***	***	***	262	273	1943	1916
flat1000_76_0	49.3	***	***	***	***	1332	1444	4349	4320
flat300_28_0	48.3	7540	6743	13200	14571	1	1	3	3
latin_square_10	75.9	147	146	1	1	1	1	1	1
r1000.1c	2.8	10343	10353	1279	3642	1	1	1	1
r1000.5	47.7	88	88	1	1	1	1	1	1
school1	25.8	681	676	540	563	79	86	15	17
unsolved		10	10	10	9	10	9	3	2

With only three unsolved instances in the STD runs and two unsolved instances in the LB runs, Algorithm 1 could solve significantly more instances than any other solver, which left nine or ten unsolved instances each.

Some interesting results become apparent when characterizing the graphs by edge density $\rho(G)$. CLIQUER is most efficient for dense graphs with edge densities of 50% and higher, with the exception of `C2000.5.1029` where Algorithm 1 found a solution greater than one (K respectively) more quickly. On the other hand, the branch and cut solvers are the most efficient for sparse instances with edge densities of less than 3%. Both fail in the opposite extreme. Algorithm 1 is the best performing for instances in between, but in contrast to the other solvers it shows a stable competitive performance throughout all edge densities.

Furthermore, it gets the most benefit if a lower bound is specified from the very beginning. This turns it into a solver comparable to branch and cut on sparse instances. Generally, the benefit from the lower bound is the higher the sparser the instance is. The reason is probably that the upper bounds computed for subproblems are tighter for sparse instances.

Note that Gurobi and CPLEX provide unsafe results as they rely on floating point arithmetic. Additional computational time would be needed to verify the optimality of a solution in safe arithmetic.

Finally, we remark that we have chosen a density threshold of 80% and not 50% for using CLIQUER during column generation in Algorithm 3, because our own implementation of CLIQUER that we are using in Algorithm 3 is not as elaborately tuned as the original code ([31]) that was used here. Furthermore, we observed that if an instance contains stable sets of weight greater than one (K respectively), such as `C2000.5.1029`, Algorithm 1 would usually find such sets more quickly than CLIQUER, yielding an overall better performance in the context of fractional coloring.

Acknowledgements We thank Andrew King for discussions on techniques to obtain dense subgraphs in large test instances.

References

1. Andrade, D.V., Resende, M.G.C., Werneck, R.F.: Fast local search for the maximum independent set problem. In: Proceedings of Workshop on Experimental Algorithms, pp. 220–234 (2008)
2. Applegate, D.L., Cook, W., Dash, S., Espinoza, D.G.: Exact solutions to linear programming problems. Operations Research Letters **35**(6), 693–699 (2007)
3. Babel, L.: A fast algorithm for the maximum weight clique problem. Computing **52**, 31–38 (1994)
4. Balas, E., Xue, J.: Minimum weighted coloring of triangulated graphs with application to maximum weight vertex packing and clique finding in arbitrary graphs. SIAM Journal of Computing **20**(2), 209–221 (1991)
5. Balas, E., Yu, C.: Finding a maximum clique in an arbitrary graph. SIAM Journal of Computing **15**(4), 1054–1068 (1986)
6. Balas E. and Niehaus, W.: A Max-Flow Based Procedure for Finding Heavy Cliques in Vertex-Weighted Graphs. Tech. Rep. MSRR No. 612, GSIA, Carnegie-Mellon University (1995)

7. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22**(4), 251–256 (1979)
8. Busygin, S.: A new trust region technique for the maximum weight clique problem. Discrete Appl. Math. **154**, 2080–2096 (2006)
9. F.T. Leighton: A graph coloring problem for large scheduling problems. Journal of Research of the National Bureau of Standards **84**, 489–505 (1979)
10. Gualandi, S., Malucelli, F.: Exact solution of graph coloring problems via constraint programming and column generation. INFORMS Journal on Computing **24**(1), 81–100 (2012)
11. Hansen, P., Labbé, M., Schindl, D.: Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. Discrete Optimization **6**(2), 135–147 (2009)
12. Held, S., Sewell, E.C., Cook, W.: Exact colors project webpage (2010). `http://code.google.com/p/exactcolors/`
13. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. Journal of the ACM **41**(5), 960–981 (1994)
14. Malaguti, E., Monaci, M., Toth, P.: An exact approach for the vertex coloring problem. Discrete Optimization **8**(2), 174–190 (2011)
15. Malaguti, E., Toth, P.: A survey on vertex coloring problems. International Transactions in Operational Research **17**, 1–34 (2010)
16. Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. INFORMS Journal on Computing **8**(4), 344–354 (1996)
17. Mittelmann, H.: Benchmarks for optimization software (2011). `http://plato.asu.edu/bench.html`
18. Mycielski, J.: Sur le coloriage des graphes. Colloq. Math. **3**, 161–162 (1955)
19. Méndez-Díaz, I., Zabala, P.: A branch-and-cut algorithm for graph coloring. Discrete Applied Mathematics **154**(5), 826–847 (2006)
20. Méndez-Díaz, I., Zabala, P.: A cutting plane algorithm for graph coloring. Discrete Applied Mathematics **156**(2), 159–179 (2008)
21. Porumbel, D.C., Hao, J.K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers and Operations Research **37**(10), 1822–1832 (2010)
22. Sewell, E.C.: A branch and bound algorithm for the stability number of a sparse graph. INFORMS Journal on Computing **10**(4), 438–447 (1998)
23. Steffy, D.E.: Topics in Exact Precision Mathematical Programming. Ph.D. thesis, Georgia Institute of Technology (2011)
24. Titiloye, O., Crispin, A.: Graph coloring with a distributed hybrid quantum annealing algorithm. In: Proceedings of the 5th KES international conference on Agent and multi-agent systems: technologies and applications, KES-AMSTA'11, pp. 553–562. Springer-Verlag, Berlin, Heidelberg (2011)
25. Titiloye, O., Crispin, A.: Quantum annealing of the graph coloring problem. Discrete Optimization **8**(2), 376–384 (2011)
26. Trick, M.: DIMACS Graph Coloring Instances (2002). `http://mat.gsia.cmu.edu/COLOR02/`
27. Warren, J., Hicks, I.: Combinatorial branch-and-bound for the maximum weight independent set problem. Technical report, Texas A&M University (2006)
28. Wu, Q., Hao, J.K.: Coloring large graphs based on independent set extraction. Computers & Operations Research **39**(2), 283–290 (2012)
29. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing **3**(1), 103–128 (2007)
30. Östergård, P.R.J.: A new algorithm for the maximum-weight clique problem. Electronic Notes in Discrete Mathematics **3**, 153–156 (1999)
31. Östergård, P.R.J., Niskanen, S.: Cliquer home page (2010). `http://users.tkk.fi/pat/cliquer.html`