Safe Lower Bounds For Graph Coloring

Stephan Held[*], Edward C. Sewell, William Cook[†]

November 8, 2010

Abstract

The best known method for determining lower bounds on the vertex coloring number of a graph is the linear-programming column-generation technique 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 standard test instances, including the unsolved challenge problems created by David S. Johnson in 1989.

[*]Research supported by DAAD

[†]Research supported by NSF Grant CMMI-0726370 and ONR Grant N00014-08-1-1104.

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 $v, w \in C$. The *clique number* $\omega(G)$, defined as the size of a largest clique in G, is a lower bound for $\chi(G)$. Similarly, the *stability number* $\alpha(G)$, defined as the maximum size of a stable set in G, provides another lower bound $\lceil n/\alpha(G) \rceil \le \chi(G)$.

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 following integer-programming problem (e.g. see [14])

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

The optimal value, $\chi_f(G)$, of the linear-programming (LP) relaxation, denoted by (CLP), obtained by replacing the integrality condition by $0 \le x_S \le 1$ for all $S \in \mathcal{S}$, is called the *fractional chromatic number* of G. It defines the lower bound $\lceil \chi_f(G) \rceil$ for $\chi(G)$. In [11] 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 $\chi(G)$ as well as for $\chi_f(G)$ and all $\epsilon > 0$ there does not exist a polynomial-time approximation algorithm that achieves an approximation ratio of n^ϵ unless $P = NP$ [20].

Mehrotra and Trick [14] 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 [9] and by Malaguti, Monaci, and Toth [12].

Our study focuses on an implementation of Mehrotra-Trick that is guaranteed to produce correct results, independent of the floating-point accuracy of LP software employed in the computation. To see the possible difficulty, consider the `queen16_16` instance from the DIMACS test collection. Using the state-of-the art LP solver Gurobi 3.0.0, at the termination of the column-generation process the floating-point representation $\chi_f^{float}(G)$ of the fractional chromatic number for this graph is $\chi_f^{float}(G) = 16.0000000000001315$. But $17 = \lceil 16.0000000000001315 \rceil$ is not a valid lower bound, since there is a known 16-coloring for `queen16_16`. In general, $\chi_f^{float}(G)$ can be smaller than, equal to , or larger than $\chi_f(G)$. This difficulty is compounded by the need to accurately run the column-generation process when dual LP solutions are available only as floating-point approximations.

We propose a technique to avoid this inaccuarcy 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 particually well suited for the 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 six of the eight open `DSJCxxx` instances created by David S. Johnson in 1989, including the optimal solution of `DSJC250.9`.

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

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

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*

$$\begin{array}{rlrll} \alpha_\pi(G) := & \max & \sum\limits_{v\in V} \pi_v y_v & & \\ & s.t. & y_v + y_w & \leq 1 & \forall \ \{v,w\} \in E \\ & & y_v & \in \{0,1\} & \forall \ V \in V \end{array} \tag{MWSS}$$

is greater than one.

2.1 Finding maximum-weight stable sets

The maximum-cardinality stable-set problem and its weighted version (MWSS) are among the hardest combinatorial optimization problems. For any $\epsilon > 0$, $\alpha(G)$ cannot be approximated within a factor $O(n^{1-\epsilon})$ unless $P= NP$ [20]. 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 [21], which we employ on dense instances. 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 [3, 4, 19].

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)
    $LB = \max(LB, \pi(S))$;
    if $F = \emptyset$ then return;
    end if
    if $\exists\, x \in X$ with $\pi_x \geq \pi((S \cup F) \cap N(x))$ then return;
    end if
    Find a weighted clique cover of $G[F]$;
    if weight of the clique cover $\leq LB - \pi(S)$ then return;
    end if
    Determine the branch vertices $F'' = \{f_1, f_2, \ldots, f_p\} \subset F$
        using the three branching rules;
    for $i = p$ down to 1 do
        $F_i = F \setminus (N(f_i) \cup \{f_i, f_{i+1}, \ldots, f_p\})$;
        MWSS_RECURSION($S \cup \{f_i\}, F_i, X$);
        $X = X \cup \{f_i\}$;
    end for
end function
MWSS_RECURSION($\emptyset, V, \emptyset$);
```

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 $S'' = \{x\} \cup S' \setminus N(x)$ created by adding x to S' and removing any of its neighbors from S'. Then

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

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 supposition 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 $\pi\left(S''\right)$ when the algorithm returned from exploring x as the branch vertex. Consequently, $LB \geq \pi\left(S''\right) \geq \pi\left(S'\right)$. 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 $\alpha_\pi\left(G\left[F\right]\right)$ 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 $LB - \pi\left(S\right)$ can be found for F, then this subproblem can be pruned.

An iterative heuristic is used to find weighted clique covers. The heuristic repeatedly chooses the vertex v with the smallest positive weight, finds a maximal 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.

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

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

Let $F'' = F \backslash F' = \{f_1, f_2, \ldots, f_p\}$ and let $F_i = F \backslash \left(N\left(f_i\right) \cup \{f_i, f_{i+1}, \ldots, f_p\}\right).$
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 $F_1, \ldots, F_p$.

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.

Hence, LB can be initialized as 1, because only solutions of value bigger than one are of interest. Furthermore, it is sufficient to stop the algorithm once a stable set S with $\pi(S) > 1$ is found.

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 $S \in \mathcal{S} \setminus \mathcal{S}'$ 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 [14]. 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 similar to the local swaps in [1]. If we do not find a stable set of weight greater than one, then we perform several additional searches using slightly perturbed greedy orders.

3 Numerically safe bounds

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, e.g. 32- or 64-bit integers in two's complement.

More generally, 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 permissible interval $[0, 1]$ slightly. 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 including safe lower bounds is summarized in Algorithm 2. Initially, a coloring is determined with the greedy algorithm DSATUR [6]. It provides the initial set $\mathcal{S}'$ and an upper bound for $\chi(G)$. The column-generation process terminates when

Algorithm 2 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
    $\pi^{int} \leftarrow \lfloor K\pi^{float} \rfloor$
    $S \leftarrow$ search for an improving stable set by heuristic (Section 2.1.2) or Algorithm 1
**until** $\pi^{int}(S) \leq K$
$\underline{\chi_f}(G) \leftarrow K^{-1}\pi^{int}(V)$
```

$\alpha_{\pi^{int}}(G) \leq K$ 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 2 is slightly larger than K, 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. Of course, such reduced weights will impose a lower fractional bound. However, the entries of π^{int} can be

Instance	$\lvert V\rvert$	$\lvert E\rvert$	None	Uniform	Neighborhood
`latin_square_10`	900	307350	1	1	1
`queen16_16`	256	12640	1	1	1
`1-Insertions_5`	202	1227	67	1	1
`1-Insertions_6`	607	6337	> 18046	9	40
`DSJC250.1`	250	3218	> 301	1	1
`DSJC250.5`	250	15668	18	13	13
`DSJC500.5`	500	62624	75	39	38
`flat300_28_0`	300	21695	25	5	4
`myciel7`	191	2360	79	33	5

Table 1: Impact of reducing dual weights on # calls to Algorithm 1

reduced safely by a total amount of

$$\operatorname{frac}(\pi^{int}, K) := \max\left\{0, \left(\sum_{v\in V} \pi_v^{int} - 1\right)\right\} \mod K, \tag{3}$$

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 [9] 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 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.

5 Experimental results

The described algorithms were implemented in the C programming language; our source code is available online [10]. 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 [18], using a 2.268 GHz Intel Xeon E5520 server, compling with gcc -O3. 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 in doubles (using QSopt_ex in both cases). 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.

For all the 119 solved DIMACS instances it turned out that $\lceil \underline{\chi_f}(G) \rceil = \lceil \overline{\chi_f}(G) \rceil$. Thus, we obtained safe results for $\lceil \chi_f(G) \rceil$. 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, as in the example from Section 3: queen16_16. 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. For space reasons, we skip those open `*-Insertions_*` and `*-FullIns_*` instances where $\lceil \chi_f(G) \rceil$ was already reported in [9] or [12]. Table 2 shows the results on the remaining open instances. 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 [7, 9, 15, 16, 17]. The last column shows the running time for computing $\underline{\chi_f}(G)$.

For the instances `DSJC1000.5`, `flat1000_50_0`, `flat1000_60_0`, `flat1000_76_0`, `wap01a`, `wap02a`, `wap06a`, `wap07a`, `wap08a`, `1-Insertions_6`, and `3-Insertions_5` we could compute $\lceil \chi_f(G) \rceil$ for the first time, improving known lower bounds on `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.

On most instances that are not listed $\underline{\chi_f}(G)$ 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. For 11 of these instances (`le450_5a`, `le450_5b`, `le450_5c`, `le450_5d`, `le450_15a`, `le450_15b`, `le450_15c`, `le450_15d`, `le450_25c`, and `le450_25d`, and `qg.order100`) the clique numbers $\omega(G)$ can be computed within seconds by CLIQUER [21] and match known upper bounds and proving $\omega(G) = \chi_f(G) = \chi(G)$.

5.2 Results of branch and price

For all open benchmark instances, we attempted to improve the lower bounds by branch and price as described in [14], allowing a time limit of three days. This way we could improve the lower bounds of `DSJC1000.9` and `DSJC250.9` by one to 216 and **72** respectively, proving optimality of a known upper bound for `DSJC250.9`.

We also did excessive branching experiments with up to 60 parallel processors, but for other instances the lower bounds grow too slow to achieve better integral bounds within a few weeks.

Instance	$\|V\|$	$\|E\|$	$\lceil\chi_f(G)\rceil$	$\omega(G)$	old LB	old UB	Time (sec.)
`DSJC250.5`	250	15668	26	12	26[9]	28[9]	18
`DSJC250.9`	250	27897	71	42	71[9]	72[9]	8
`DSJC500.1`	500	12458	*	5	6[7]	12[17]	*
`DSJC500.5`	500	62624	**43**	13	16[7]	48[17]	439
`DSJC500.9`	500	224874	123	54	123[9]	126[17]	100
`DSJC1000.1`	1000	49629	*	6	6[7]	20[17]	*
`DSJC1000.5`	1000	249826	**73**	14	17[7]	83[17]	142014
`DSJC1000.9`	1000	449449	215	63	215[9]	223[17]	5033
`r1000.1c`	1000	485090	96	89	96[9]	98[9]	2634
`C2000.5`	2000	999836	*	16	16	148[17]	*
`C4000.5`	4000	4000268	*	≥ 17	17	271[17]	*
`latin_square_10`	900	307350	90	90	90[15]	98[13]	76
`abb313GPIA`	1557	65390	8	8	8[15]	10[15]	3391
`flat1000_50_0`	1000	245000	**50**	14	14	50[9]	3331
`flat1000_60_0`	1000	245830	**60**	14	14	60[9]	29996
`flat1000_76_0`	1000	246708	**72**	14	14	82[9]	190608
`wap01a`	2368	110871	41	41	41[15]	45[8]	20643
`wap02a`	2464	111742	40	40	40[15]	44[8]	236408
`wap03a`	4730	286722	*	40	40[15]	50[8]	*
`wap04a`	5231	294902	*	40	40[15]	46[8]	*
`wap06a`	947	43571	40	40	40[15]	43[8]	382
`wap07a`	1809	103368	40	40	40[15]	45[8]	25911
`wap08a`	1870	104176	40	40	40[15]	45[8]	18015
`1-Insertions_6`	607	6337	3	2	3[15]	7[15]	1167
`3-Insertions_5`	1406	9695	3	2	3[15]	6[15]	6959

Table 2: Computational results on open benchmarks

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 3 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

Instance	$\lvert X\rvert$	$\lvert E(G[X])\rvert$	$\lceil\chi_f(G[X])\rceil$	old LB	UB	Time
`DSJC500.1`	300	5436	**9**	6	12	16 days
`DSJC1000.1`	350	8077	**10**	6	20	< 36 days
`C2000.5`	1400	502370	**99**	16	148	< 24 days
`C4000.5`	1500	589939	**107**	17	272	< 26 days
`wap03a`	2500	164008	40	40	48	< 3 days
`wap04a`	2500	159935	40	40	46	< 1 days

Table 3: Lower bounds $\lceil\chi_f(G[X])\rceil$ from induced subgraphs

Instance	$\rho(G)$	Gurobi 3.0.0		CPLEX 12.2		CLIQUER 1.2		Algorithm 1	
	in %	STD	LB	STD	LB	STD	LB	STD	LB
`C2000.5.1029`	50	***	***	***	***	***	30586	***	11373
`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
`DSJC500.5`	50.1	***	***	***	***	9	9	32	32
`DSJC500.9`	90.1	***	***	24318	22105	1	1	1	1
`DSJC1000.5`	50	***	***	***	***	1076	1057	3634	3547
`DSJC1000.9`	89.9	***	***	***	***	1	1	2	2

Table 4: Running times of various MWIS solvers on hard instances in seconds.

subgraphs, while column 5 reports previously published lower bounds, corresponding to the respective maximum clique numbers.

5.4 Maximum-weight stable set results

Finally, we demonstrate the efficiency of Algorithm 1 for solving maximum-weight stable set problems. We compared the new algorithm with the currently fastest integer-programming solvers Gurobi 3.0.0 and CPLEX 12.2, as well as CLIQUER 1.21 [22], which solved the maximum-weight clique problems in the complement graphs. Where available, we used the final maximum-weight stable set instances as they occure in Algorithm 2. They can be downloaded from `http://code.google.com/p/exactcolors/wiki/MWISInstances`. Table 4 shows the results on the `DSJC*` instances and `C2000.5.1029`, to which we restrict ourselves here for space reasons. Comprehensive test results will be reported in the full version of the paper.

We performed two experiments per instance and solver. First, in the columns labeled STD, we computed the maximum-weight stable set as is. Second, in the columns labeled LB, we used the solvers in the same setting as in Algorithm 2 with an initial lower bound of $LB = 1$. We gave a time limit of ten hours to each run. `C2000.5.1029` is the only instance with $\alpha_\pi(G) > 1$, here Algorithm 1 was the fastest to find such a set. Algorithm 1 is competetive throughout all edge-density classes and was a main ingredient for improving known lower bounds.

References

[1] D. V. Andrade, M. G. C. Resende, and R. F. Werneck. Fast local search for the maximum independent set problem. In *Proceedings of Workshop on Experimental Algorithms*, pages 220–234, 2008.

[2] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35(6):693–699, 2007.

[3] L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52:31–38, 1994.

[4] E. Balas and J. Xue. 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] E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15(4):1054–1068, 1986.

[6] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[7] M. Caramia and P. Dell'Olmo. Bounding vertex coloring by truncated multistage branch and bound. *Networks*, 44(4):231–242, 2004.

[8] M. Caramia and P. Dell'Olmo. Coloring graphs by iterated local search traversing feasible and infeasible solutions. *Discrete Appl. Math.*, 156(2):201–217, 2008.

[9] S. Gualandi and F. Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *Optimization Online*, 2010.

[10] S. Held, E. C. Sewell, and W. Cook. Exact colors project webpage, 2010. `http://code.google.com/p/exactcolors/`.

[11] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[12] E. Malaguti, M. Monaci, and P. Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, in press, 2010.

[13] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17:1–34, 2009.

[14] A. Mehrotra and M. A. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[15] I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.

[16] I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159 – 179, 2008.

[17] D. C. Porumbel, J.-K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10):1822–1832, 2010.

[18] M.A. Trick. DIMACS Graph Coloring Instances, 2002.
`http://mat.gsia.cmu.edu/COLOR02/`.

[19] J.S. Warren and I.V. Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. *Technical report, Texas A&M University*, 2006.

[20] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

[21] P. R. J. Östergård. A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, 3:153–156, 1999.

[22] P. R. J. Östergård and S. Niskanen. Cliquer home page, 2010.
`http://users.tkk.fi/pat/cliquer.html`.