Algorithmica (1996) 15: 397–412

Algorithmica

Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring[1]

E. Balas[2] and Jue Xue[3]

Abstract. The linear programming relaxation of the minimum vertex coloring problem, called the fractional coloring problem, is NP-hard. We describe efficient approximation procedures for both the weighted and unweighted versions of the problem. These fractional coloring procedures are then used for generating upper bounds for the (weighted or unweighted) maximum clique problem in the framework of a branch-and-bound procedure. Extensive computational testing shows that replacing the standard upper bounding procedures based on various integer coloring heuristics with the somewhat more expensive fractional coloring procedure results in improvements of the bound by up to one-fourth in the unweighted and up to one-fifth in the weighted case, accompanied by a decrease in the size of the search tree by a factor of almost two.

Key Words. Maximum clique, Minimum coloring, Fractional coloring.

1. Introduction. Consider an undirected graph $G = (V, E)$ and its complement $\bar{G} = (V, \bar{E})$, where $\bar{E} := \{(i, j)\colon (i, j) \neq E\}$. For any $S \subseteq V$, $G(S)$ denotes the subgraph of G induced by S. A *clique* of G is a set of pairwise adjacent vertices, whereas a *stable set* (independent set, vertex packing) is a set of pairwise nonadjacent vertices. From the definitions, S is a clique of G if and only if it is a stable set of $\bar{G}$. The maximum clique/maximum stable set problem asks for a clique/stable set of maximum size. Obviously, the former problem on G is the latter problem on $\bar{G}$ and vice versa.

A (vertex) *coloring* of G is a partition of the vertex set into stable subsets, each of which is called a color class. A *clique covering* of G is a partition of the vertex set into cliques. Obviously, $\{S_1, \ldots, S_p\}$ is a (vertex) coloring of G if and only if it is a clique covering of $\bar{G}$. The minimum coloring/minimum clique covering problem asks for a coloring/clique covering of minimum cardinality. Again, the former problem on G is the latter problem on $\bar{G}$.

We address the above pair of problems in the formulation

$$z_0 = \max\{1x\colon Ax \leq 1,\ x_j \in \{0, 1\},\ j \in V\}, \tag{1}$$

for the maximum clique problem, and

$$t_0 = \min\{y1\colon yA \geq 1,\ y_S \in \{0, 1\},\ S \in \mathcal{S}\}, \tag{2}$$

for the minimum coloring problem; where A is the incidence matrix of stable sets versus vertices, and $\mathcal{S}$ is the family of all stable sets of G. Clearly, $t_0 \geq z_0$, and the minimum

[1] This research was supported by the National Science Foundation under Grant No. DDM-9201340 and the Office of Naval Research through Contract N00014-85-K-0198.

[2] Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA.
[3] Graduate School of Management, Clark University, 950 Main Street, Worcester, MA 01610, USA.

Received May 20, 1994; revised September 15, 1994. Communicated by N. Megiddo.

coloring problem has been frequently used as an upper bounding device for the maximum clique problem.

If the 0–1 conditions in both problems are relaxed, then (1) and (2) become a pair of dual linear programs; hence the *fractional coloring* problem

$$t_1 = \min\{y1\colon\ yA \geq 1,\ y_S \geq 0,\ S \in \mathcal{S}\} \tag{3}$$

also provides a valid upper bound t_1 on the value of z_0, and in fact a tighter one than t_0. Therefore we wish to use (3) to obtain a stronger upper bound on the value of z_0. The linear program (3) should be easier to solve than the integer program (2); but (3) has exponentially many variables and there is no known algorithm bounded by a polynomial in $|V|$ for solving it. In fact (3) is known to be NP-complete [11]. Therefore, rather than solving it exactly, we use a *fractional coloring heuristic* to find an approximate solution to (3).

The above problems have their weighted counterparts, where the weight of a clique is the sum of weights of its vertices. Thus the *maximum weight clique problem* is

$$\tilde{z}_0 = \max\{wx\colon\ Ax \leq 1,\ x_j \in \{0, 1\},\ j \in V\}, \tag{4}$$

while the *minimum weighted coloring problem* and its *fractional* counterpart are

$$\tilde{t}_0 = \min\{y1\colon\ yA \geq w,\ y_S \geq 0 \text{ integer},\ S \in \mathcal{S}\} \tag{5}$$

and

$$\tilde{t}_1 = \min\{y1\colon\ yA \geq w,\ y_S \geq 0,\ S \in \mathcal{S}\}, \tag{6}$$

respectively. Unlike in the unweighted case, a color class S appears in the solution to (5) with a weight y_S possibly different from 1 (or 0).

For any $C \subseteq \mathcal{S}$, we denote $y(C) := \sum(y_S\colon\ S \in C)$.

In this paper we present approximate procedures for solving the fractional coloring problem (3) and its weighted version (6). We then embed these procedures as upper bounding devices into a branch-and-bound algorithm for solving the maximum clique problem in its weighted and unweighted versions. Finally, we present extensive computational experience with the resulting algorithms, both on randomly generated graphs and on the DIMACS Challenge benchmark problems.

The fractional coloring procedure for unweighted graphs and the corresponding branch-and-bound procedure were presented, along with computational results, in [4] and [16]. Further details can be found in [17]. Essentially the same procedure for the unweighted case has been rediscovered independently in the context of vertex packing by Mannino and Sassano [13].

2. A Fractional Coloring Heuristic. The basic idea of our approach is this. Suppose we have an (integer) coloring $C_1 := (S_1, \ldots, S_q)$ of G, where each S_i is a stable set (color class). Since C_1 is a partition of V, typically some of the color classes S_i are not maximal. Suppose we now augment these color classes by coloring as many vertices as we can with a second color from among the q color classes. Let V_2 be the set of these vertices

that now belong to two color classes. The coloring C_1 thus becomes a collection of overlapping rather than disjoint subsets, but its weight (cardinality) remains unchanged. If we now find a new coloring C_2 of the vertices in $V \setminus V_2$, then every vertex will have been colored twice, and thus using one-half times the first and one-half times the second coloring, we obtain a full fractional coloring of total weight $(|C_1)| + |C_2|)/2$. However, since C_2 is a coloring of fewer than $|V|$ vertices, typically $|C_2| < |C_1|$ and therefore $(|C_1| + |C_2|)/2 < |C_1|$, i.e., our fractional coloring is better than C_1. This process can be repeated by using a third, fourth, etc., coloring as long as improvements can be obtained.

The (integer) colorings needed at each iteration will be generated by an Integer Coloring Heuristic (ICH). We denote by $C := C_1 \cup C_2 \cup \cdots \cup C_k$ the collection of color classes generated during the procedure. Notice that the color classes S_i of each coloring C_j are generated at iteration j and augmented at subsequent iterations. At iteration k, we denote by U_k the set of vertices not yet colored (during that iteration), and by t_1^k the current value of the fractional coloring at hand.

Fractional Coloring Procedure (FCP)

0. Initialize: $t_1^0 := \infty$, $C := \emptyset$, $y := 0$, $k = 1$.

At iteration k:

1. Set $U_k := V$.
2. For each $v \in U_k$, choose a color class $S_i \in C$, if one exists, such that $S_i \cup \{v\}$ is stable, and set

$$S_i := S_i \cup \{v\}, \qquad U_k := U_k \setminus \{v\}.$$

3. Apply ICH to $G(U_k)$; let C_k be the coloring found. If $(|C|+|C_k|)/k < t_1^{k-1}$, set

$$t_1^k := (|C| + |C_k|)/k, \qquad C := C \cup C_k, \qquad k := k + 1$$

and go to 1.

Otherwise let $C^* := C$, $t_1^* = t_1^{k-1}$,

$$y_S^* = \begin{cases} 1/(k-1) & \text{for } S \in C^*, \\ 0 & \text{for } S \notin C^* \end{cases}$$

and stop.

Note that the color classes generated by FCP need not be distinct, i.e., repetition is allowed.

THEOREM 2.1. *y^* is a feasible fractional coloring, with $t_1^* = y^*(C^*)$; hence $\lfloor t_1^* \rfloor$ is an upper bound on the cardinality of a clique in G.*

PROOF. Each $S \in C^*$ is stable by construction; hence y^* with support C^* satisfies all

constraints (3) if and only if

$$\sum(y_S^*:\ S \in C^* \text{ and } v \in S) \geq 1 \qquad \text{for all} \quad v \in V.$$

During each iteration except for the last one, every $v \in V$ is included in exactly one color class either in step 2 or in step 3. If the number of iterations is p, each $v \in V$ is included in p or $p-1$ color classes $S \in C^*$; and since $y_S^* = 1/(p-1)$ for all $S \in C^*$,

$$\sum(y_S^*:\ S \in C^* \text{ and } v \in S) \geq (p-1)/(p-1) = 1 \qquad \text{for all} \quad v \in V. \qquad \square$$

It is desirable to have the stopping rule amended to the effect that the maximum number of color classes allowed is $O(|V|)$.

THEOREM 2.2. *If the number of color classes generated is $O(|V|)$ and the ICH used has complexity $O(h)$, then FCP can be executed in $O(\min\{|E| \cdot |V|,\ \alpha(G) \cdot |V|^2\} + h \cdot |V|)$ time, where $\alpha(G)$ is the size of a maximum stable set in G.*

PROOF. At every iteration, step 2 requires checking for every pair v, S whether $N(v) \cap S = \emptyset$, where $N(v) := \{w \in V:\ (v, w) \in E\}$. This can be done either by checking for each $u \in N(v)$ whether it has the color S, which takes $O(\deg(v))$ time, or by checking for each $u \in S$ whether it belongs to $N(v)$, which requires $O(\alpha(G))$ time. Thus the complexity of executing step 2 throughout the procedure (i.e. at most $O(|V|)$ times) is $O(|V| \cdot \min\{|E|, \alpha(G) \cdot |V|\})$.

On the other hand, the effort involved in step 3 depends on the complexity of the ICH used. If the latter is $O(h)$, then the complexity of step 3 throughout the procedure is $O(h \cdot |V|)$. Thus the complexity of FCH is as claimed. $\square$

For dense graphs $\alpha(G)$ is small and the complexity of the procedure is only slightly worse than $O(|V|^2 + h|V|)$.

We have tested two heuristics in the role of ICH. One is DSATUR (for Degree of Saturation), a procedure due to Brelaz [8]; which consists of applying n times the following step:

- Choose an uncolored vertex v such that the vertices in $N(v)$ represent a maximum number of color classes, and put v in the first color class where it fits. To break ties, give preference to vertices with higher degree.

The other one is the simple coloring heuristic SCH, which can be stated this way:

- Open a color class and put in it as many vertices as possible, in order of nonincreasing degree. Repeat this as long as possible.

DSATUR typically finds a better coloring than SCH. In [8] it was found the most efficient among five heuristics tested (see [12], [14], and [15] for other coloring heuristics). However, the complexity of DSATUR is $O(|V^2|)$, whereas that of our SCH is $O(\min\{|E|, |V|^2 - |E|\})$. For very sparse and very dense graphs, DSATUR is an order of magnitude more expensive than SCH.

Before continuing, we illustrate FCP on a graph with nine vertices.

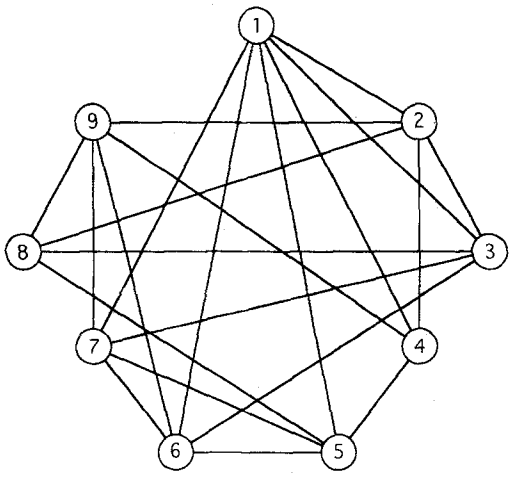

Fig. 1. Graph for Example 1.

EXAMPLE 1. Consider the graph of Figure 1.

$k = 1$. $U_1 = V$ and the first coloring generated is $C_1 = \{S_1, \ldots, S_5\}$, with $S_1 = \{1, 9\}$, $S_2 = \{2, 5\}$, $S_3 = \{3, 4\}$, $S_4 = \{6, 8\}$, $S_5 = \{7\}$. This gives a starting bound of $t_1^1 = |C| = |C_1| = 5$.

$k = 2$. Vertex 2 can be added to S_5 and vertex 4 to S_4, i.e. $S_4 := \{4, 6, 8\}$, $S_5 := \{2, 7\}$. $U_2 = \{1, 3, 5, 6, 7, 8, 9\}$. ICH yields $C_2 := \{S_6, S_7, S_8, S_9\}$, with $S_6 = \{1, 8\}$, $S_7 = \{3, 5, 9\}$, $S_8 = \{6\}$, $S_9 = \{7\}$. $(|C| + |C_2|)/2 = 4.5 < 5$, so $t_1^2 := 4.5$, $C = C \cup C_2$.

$k = 3$. Vertex 2 can be added to S_8, vertices 4 and 8 to S_9; i.e., $S_8 := \{2, 6\}$, $S_9 := \{4, 7, 8\}$. $U_3 := \{1, 3, 5, 6, 7, 9\}$. $C_3 := \{S_{10}, S_{11}, S_{12}, S_{13}\}$, with $S_{10} := \{1, 9\}$, $S_{11} := \{3, 5\}$, $S_{12} := \{6\}$, $S_{13} := \{7\}$. $(|C| + |C_3|)/3 = 4.33 < 4.5$, so $t_1^3 := 4.33$, $C := C \cup C_3$.

At this point the number of color classes generated exceeds our limit, so we stop:

$$C^* := \{S_1, \ldots, S_{13}\}, \qquad y^*_{S_i} = \tfrac{1}{3} \text{ for } i = 1, \ldots, 13, \qquad \text{and} \qquad \lfloor t_1^* \rfloor = 4.$$

The fractional coloring procedure can be generalized so as to apply to the weighted case, i.e., to finding an approximate solution to problem (6). However, this generalization is not trivial. The reason for this is that in the weighted case every vertex v has to be colored with enough colors of enough weight so that their total weight adds up to the weight of v. Therefore the integer coloring heuristic used in FCP needs to be replaced by a weighted integer coloring heuristic (WICH), where each vertex may belong to several (weighted) color classes. For $v \in V$, we call the *residual weight* of v, and denote by $r(v)$, the weight that needs to be "covered" by weighted color classes during the current iteration. Like in the unweighted case, a full integer coloring is generated at every iteration; but the weight $r(v)$ to be covered (absorbed) by weighted color classes for each $v \in V$ may be less than the initial weight $w(v)$. This is so because, unlike in the unweighted case, the integer (weighted) coloring at a given iteration may "overcover" (i.e., color with excess

weight) some of the vertices. In particular, this may happen if in step 2 we fit a vertex v into a color class S_i whose weight y_{S_i} exceeds $w(v)$. In such a case the residual weight of vertex v at the next iteration will be $r(v) := w(v) - (y_{S_i} - w(v)) < w(v)$.

By analogy with the unweighted case, at iteration k we denote by U_k the set of vertices not yet fully colored (i.e., having positive residual weight) during that iteration, and by $\tilde{t}_1^k$ the current value of the fractional coloring at hand. Unlike in the unweighted case, however, we have to keep track explicitly of the weights y_{S_i} generated during successive iterations. These weights are kept integer until the last iteration, when the appropriate fractional values are calculated.

Weighted Fractional Coloring Procedure (WFCP)

0. Initialize: $\tilde{t}_1^0 := \infty$, $C := \emptyset$, $y^0 := 0$, $r(v) := 0$, $v \in V$, $U_1 = V$, $k := 1$.

At iteration k:

1. Let

$$r(v) := r(v) + w(v), \quad v \in V; \qquad U_k := \{v \in V\colon r(v) > 0\}.$$

2. For every $v \in U_k$, choose a color class $S_i \in C$, if one exists, such that $S_i \cup \{v\}$ is stable. Set $S_i := S_i \cup \{v\}$, $r(v) := r(v) - y_{S_i}^{k-1}$, and, if $r(v) \leq 0$, set $U_k := U_k \backslash \{v\}$.
3. Apply WICH to $G(U_k)$; let y^k be the weighted coloring found, with C_k the corresponding set of color classes.

If $(y^{k-1}(C) + y^k(C_k))/k \leq \tilde{t}_1^{k-1}$, set

$$\tilde{t}_1^k := (y^{k-1}(C) + y^k(C_k))/k,$$

$$y_S^k := \begin{cases} y_S^{k-1} & \text{for } S \in C, \\ y_S^k & \text{for } S \in C_k, \\ 0 & \text{for } S \notin C \cup C_k, \end{cases}$$

$$C := C \cup C_k, \qquad k := k + 1,$$

and go to 1.

Otherwise let $C^* := C$, $\tilde{t}_1^* := \tilde{t}_1^{k-1}$,

$$y_S^* := \begin{cases} y_S^{k-1}/(k-1) & \text{for } S \in C^*, \\ 0 & \text{for } S \notin C^*, \end{cases}$$

and stop.

THEOREM 2.3. *y^* is a feasible weighted fractional coloring, with $\tilde{t}_1^* = y^*(C^*)$; hence $\lfloor \tilde{t}_1^* \rfloor$ is an upper bound on the weight of a clique in G.*

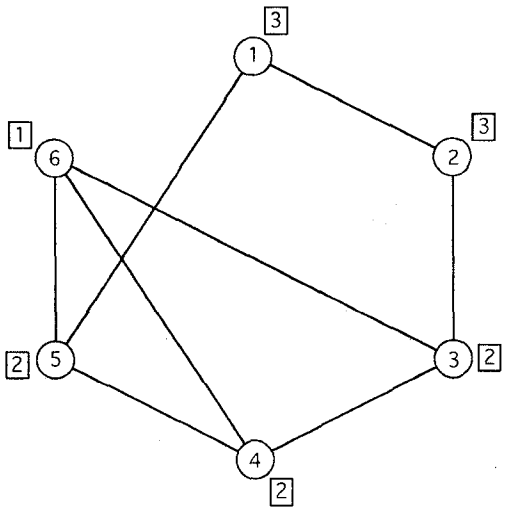

Fig. 2. Graph for Example 2.

PROOF. Parallels that of Theorem 1. □

The complexity of the weighted fractional coloring procedure is the same as that of its unweighted analog.

The integer coloring heuristic that we use in the role of WICH is a generalization to the weighted case of SCH, which can be stated as follows:

- Open a color class S and put in S as many vertices $v \in V_k$ as possible, in order of decreasing residual weight. Then reduce the residual weight $r(v)$ of each $v \in S$ by $r(v_0) := \min\{r(v)\colon v \in S\}$ and remove from U_k all vertices v such that $r(v) = 0$. Repeat this procedure until $U_k = \emptyset$.

Next we illustrate WFCP on the graph with six vertices in Figure 2, where the numbers in the boxes are the weights.

EXAMPLE 2.

$k = 1$. $r = w = (3, 3, 2, 2, 2, 1)$, $U_1 = \{1, \ldots, 6\}$. WICH yields the integer coloring defined by $C_1 = \{S_1, \ldots, S_5\}$, with $S_1 = \{1, 4\}$, $y^1_{S_1} = 2$; $S_2 = \{1, 3\}$, $y^1_{S_2} = 1$; $S_3 = \{2, 5\}$, $y^1_{S_3} = 2$; $S_4 = \{2, 6\}$, $y^1_{S_4} = 1$; $S_5 = \{3\}$, $y^1_{S_5} = 1$.

We have $\tilde{t}^1_1 = \sum(y^1_{S_i}\colon i = 1, \ldots, 5) = 7$, $y^1_{S_i}$ as defined above for $i = 1, \ldots, 5$, $y^1_S = 0$ for all other S, and $C := C_1$.

$k = 2$. $r = (3, 3, 2, 2, 2, 1)$, $U_2 = \{1, \ldots, 6\}$. We can add vertex 5 to S_5, so $S_5 := \{3, 5\}$, $r(5) = 2 - 1 = 1$. U_2 remains unchanged, but $r = (3, 3, 2, 2, 1, 1)$.

WICH yields the coloring $C_2 = \{S_6, \ldots, S_9\}$, with $S_6 = \{1, 3\}$, $y^2_{S_6} = 2$; $S_7 = \{1, 6\}$, $y^2_{S_7} = 1$; $S_8 = \{2, 5\}$, $y^2_{S_8} = 1$; $S_9 = \{2, 4\}$, $y^2_{S_9} = 2$.

We have $(y^1(C) + y^2(C_2))/2 = (7 + 6)/2 = 6.5 < 7$, so we set $\tilde{t}_1^2 := 6.5$,

$$y_S^2 := \begin{cases} y_S^1 & \text{for } S \in C, \\ y_S^2 & \text{for } S \in C_2, \\ 0 & \text{for } S \notin C \cup C_2, \end{cases}$$

and $C := C \cup C_2$.

$k = 3$. $r = (3, 3, 2, 2, 2, 1)$, $U_3 = \{1, \ldots, 6\}$. The color classes in C are all maximal. Applying WICH to $G(U_3)$ yields C_3 which is the same as C_1, i.e., $S_{10} = S_1, \ldots, S_{14} = S_5$, with $y_{S_i}^3 = y_{S_{i-9}}^1$, $i = 10, \ldots, 14$. Further,

$$(y^2(C) + y^3(C_3))/3 = (13 + 7)/3 = 6.67 > 6.5,$$

hence we stop with $C^* := C$, $y_S^* = y_S^2$ for $S \in C^*$, $y_S^* = 0$ otherwise, and $\lfloor \tilde{t}_1^* \rfloor = 6$.

It is shown in the last section on computational results, that the fractional coloring procedure provides a substantially tighter bound, sometimes by as much as one-fourth, on the maximum clique size or clique weight, than the best integer coloring heuristics. The ratio between the two bounds seems to improve with the size of the graph in favor of the fractional coloring bound.

3. Using FCP and WFCP in a Branch and Bound Algorithm. In this section we embed FCP and WFCP into a branch and bound algorithm for finding a maximum clique, or a maximum-weight clique, in an arbitrary graph $G = (V, E)$. The branch-and-bound algorithm has the same structure as that of Balas and Yu [6] for the unweighted case and Balas and Xue [5] for the weighted case. FCP in the unweighted case, and WFCP in the weighted case, are used as upper bounding devices. Because of the important differences between these two procedures, as well as between the weighted and unweighted lower bounding devices, we have actually developed two distinct algorithms, MAXCLQ1 for the unweighted and MAXCLQ2 for the weighted case. We discuss the weighted case only; the corresponding steps of the algorithm for the unweighted case can easily be substituted on the basis of our discussion in Section 2.

At the root node of the search tree, we start by finding an edge-maximal triangulated subgraph G' of G (see [3] and [18]) and a maximum-weight clique K^* of G'. Then $LB := w(K^*)$ is a lower bound on the weight of a maximum-weight clique in G. Next we apply WFCP to G in order to obtain an upper bound, UB_G, on the weight of any clique in G. If $UB_G \leq LB$, we are done; otherwise we branch, based on the following branching scheme (see [6] and [5] for a proof of validity and discussion):

THEOREM 3.1. *Let $G' := G(V')$ be an induced subgraph of G, let $UB_{G'}$ be an upper bound on the weight of a clique in G', and let $v_1, \ldots, v_m$ be an arbitrary ordering of the vertices in $V \backslash V'$. If G has a clique K such that $w(K) > UB_{G'}$, then K is contained in one of the m sets*

$$V_i := \{v_i\} \cup N(v_i) \backslash \{v_1, \ldots, v_{i-1}\}, \qquad i = 1, \ldots, m, \tag{7}$$

where, for $i = 1$, we define $\{v_1, \ldots, v_{i-1}\} = \emptyset$.

At an arbitrary node of the search tree other than the root, we have a subproblem $P' := (G', I', UB_{G'})$, where G' is a subgraph of G induced by some vertex set V_i of the form (7), I' is a subset of the vertices in $V \backslash V_i$ to be added to any clique of G' (to yield a larger clique of G), and $UB_{G'}$ is an upper bound on $w(K)$ for any clique K of G'.

The statement of the algorithm follows, with L denoting the list of active subproblems (nodes of the search tree).

Maximum-Weight Clique Algorithm (MAXCLQ2)

0. *Initialize.* Find an edge-maximal triangulated subgraph G^* of G, and a maximum-weight clique K^* of G^*. Set $LB := w(K^*)$. Put into L the problem $P := (G, \emptyset, \infty)$ and go to 1.
1. *Subproblem selection.* If $L = \emptyset$, stop: K^* is a maximum-weight clique of G. Otherwise, choose a subproblem $P' := (G', I', UB_{G'})$ in L and remove P' from L.
   If $UB_{G'} + w(I') \leq LB$, discard P' and go to 1.
2. *Lower bounding.* Use a heuristic to find a maximal clique K' in G'. If $w(K') + w(I') > LB$, set $K^* := K' \cup I'$ and $LB := w(K^*)$.
3. *Upper bounding.* Apply WFCP to G' to get an upper bound $UB_{G'}$ on the weight of a clique in G'. If $UB_{G'} + w(I') \leq LB$, discard P' and go to 1.
4. *Branching.* Let C_1 be the integer coloring generated in the first iteration of WFCP, and let C_1' be a maximal subset of C_1 such that $y(C_1') \leq LB - w(I')$. Further, let $v_1, \ldots, v_m$ be an arbitrary ordering of the vertices in $V \backslash \{v\colon\ y(C_1') \geq w(v)\}$. For $i = 1, \ldots, m$, put into L the subproblem $P_i' := (G_i', I_i', UB_{G_i'})$, where $G_i' := G'(N(v_i) \backslash \{v_1, \ldots, v_{i-1}\})$, $I_i' := I' \cup \{v_i\}$, and $UB_{G_i'} := UB_{G'} - w(v_i)$. Then go to 1.

Our current implementation uses a subproblem selection rule (step 1) based on a depth-first search strategy, and a lower bounding heuristic (step 2) that constructs a clique at every node of the search tree by the same procedure as in [5]. In the unweighted case the lower bounding heuristic constructs a clique as a by-product of DSATUR. The main feature that distinguishes this algorithm from those of [6] and [5] is its upper bounding technique. The fractional coloring procedure provides a stronger upper bound than the integer coloring used in [6] and [5], but is of course computationally more expensive.

4. Computational Results. The procedures FCP and WFCP were implemented in C and tested both as stand-alone procedures for finding an approximate solution to the (unweighted and weighted) fractional coloring problem, and as bounding devices in the framework of a branch and bound procedure. For the purposes of testing, random graphs were generated, having between 100 and 500 vertices, and densities of 0.4 to 0.95. Here density means the probability that a certain edge is present. The tests were run on a NeXTstation. In addition, the collection of DIMACS Challenge benchmark problems was solved on a DEC alpha 300–400 AXP.

Several currently available branch and bound codes for finding maximum cliques (see [1], [2], [6], and [9]) can solve without difficulty problems on *sparse* random graphs even for sizes well above 1000 vertices. Dense graphs are an altogether different matter.

The expected size of a minimum (integer) coloring of a random graph is $O(n/\log_a n)$, and the corresponding number for a maximum clique is $O(\log_b n)$ (see [7]), where the bases of the two logarithms are $a = 1/(1-d)$ and $b = 1/d$, respectively, with d the density of the random graph, i.e. the probability that a particular edge is present. It follows that the ratio between the expected size of a minimum coloring and a maximum clique is $O(n/(\log_a n \cdot \log_b n))$. This shows that the degree of difficulty in solving the problem increases rapidly with n. Now, for fixed n, the above ratio attains its maximum for $a = b = 1/(\frac{1}{2}) = 2$. This suggests that the hardest problems are those on graphs whose density is 0.5. This conclusion, however, is not corroborated by computational experience with any of the above listed algorithms, which are all branch and bound procedures, differing only in their branching rules and bounding devices. The general experience with all these algorithms has been that the size of the search tree, and therefore the computational effort needed to solve the problem, grows with density not just up to 0.5 but even faster beyond that, and that this trend continues well beyond the density of 0.9. (This might be different for an approach based primarily on cutting planes.) Actually, for densities of 0.8–0.9, problems on graphs with 200–300 vertices are already very hard.

First we examine the performance of the fractional coloring procedures by themselves. Tables 1 and 2 summarize our findings in this respect, for the unweighted and weighted cases, respectively. We have examined a larger number of variants and carried out more extensive testing for the unweighted case. The first two columns of Table 1 describe the

Table 1. Fractional versus integer coloring procedures

Graphs		Upper bounds				Computational effort			
						Iterations		CPU (seconds)	
$\|V\|$	Density (%)	t_{SCH}	t_{DSAT}	t_{FCP1}	t_{FCP2}	FCP1	FCP2	FCP1	FCP2
100	10	6.0	6.0	6.0	5.0	2.0	3.5	0.031	0.047
	20	9.5	9.0	8.0	8.0	5.0	3.5	0.055	0.069
	30	13.0	12.0	11.0	10.0	4.0	6.0	0.039	0.102
	40	15.0	15.0	13.0	13.0	7.5	4.0	0.063	0.070
	50	19.5	18.5	16.5	15.5	5.5	9.0	0.055	0.164
	60	23.5	23.0	20.0	19.5	7.5	7.0	0.077	0.148
	70	27.5	27.5	25.0	25.0	5.5	5.0	0.062	0.124
	80	35.0	34.5	31.5	30.0	5.0	7.5	0.071	0.195
	90	43.0	44.5	41.0	39.5	4.5	6.5	0.061	0.212
500	10	18.0	16.5	14.5	13.5	8.5	7.0	1.024	2.336
	20	30.0	27.0	24.0	23.0	13.0	8.5	1.429	3.343
	30	42.5	40.5	35.0	34.0	7.5	8.0	0.984	3.577
	40	57.0	51.0	46.5	45.0	8.0	9.5	1.171	4.691
	50	69.0	66.0	58.5	57.5	11.5	12.5	1.742	6.686
	60	86.0	81.0	74.0	72.5	9.0	10.0	1.570	6.119
	70	105.5	99.5	91.5	90.8	11.5	10.0	2.138	6.823
	80	130.5	126.0	114.5	113.5	14.5	12.0	3.880	8.899
	90	166.0	166.5	151.0	151.5	11.0	6.5	2.928	5.694

random graphs in terms of their size and density. Columns 3–6 show the upper bounds (cardinality of colorings) obtained by SCH (t_{SCH}), by DSATUR (t_{DSAT}), by FCP1 (t_{FCP1}), and by FCP2 (t_{FCP2}), respectively, with each entry representing the average of two runs. The first two procedures are the two integer coloring heuristics discussed in Section 2. FCP1 is the version of the fractional coloring procedure that uses SCH as its integer coloring heuristic, whereas FCP2 is the version that uses DSATUR in the same role.

As the numbers show, the integer coloring obtained by DSATUR is typically better than the one obtained by SCH, though not without exceptions. The improvements in the bound obtained by FCP1 over SCH range from 0 to $\frac{1}{5}$, and FCP2 sometimes goes even beyond $\frac{1}{4}$. The number of iterations for FCP1 increases somewhat with problem size, but remains below 15. Computing times increase both with problem size and density. There is a large discrepancy between the computing times of the two versions of FCP. This is partly due to the fact that DSATUR is more time consuming than SCH, but mainly to the fact that in the experiment reported here we let FCP2 run for 50 iterations in each case, to make sure we got the lowest value obtainable by our approach.

Table 2 reports, in a more summary fashion, the corresponding results for the weighted case. Here $\tilde{t}_{WICP}$ and $\tilde{t}_{WFCP}$ stand for the weights of the (weighted) integer and fractional colorings, respectively, obtained by the procedures WICP and WFCP discussed in Section 2. The last column shows the number of iterations of WFCP. Again, each entry represents the average of two runs. As in the unweighted case, the improvement in the bound obtained by the fractional coloring procedure over the integer one varies between 0 and $\frac{1}{5}$.

Next we turn to the tables describing the performance of the fractional coloring procedure as an upper bounding device in the framework of a branch and bound algorithm. For the unweighted case, we tested various ways of using this upper bounding device,

Table 2. Weighted fractional coloring procedure

Graphs		Upper bounds		
$\|V\|$	Density (%)	$\tilde{t}_{WICP}$	$\tilde{t}_{WFCP}$	Iterations
100	10	35.0	29.5	14.0
	20	52.5	46.5	7.0
	50	106.0	92.5	7.0
	80	195.0	176.5	11.0
	90	255.5	237.5	16.0
200	10	57.0	45.5	9.0
	20	84.5	78.5	3.5
	50	180.5	159.0	7.0
	80	347.5	306.5	18.0
	90	454.5	407.5	11.0
300	10	69.5	55.0	15.5
	20	111.0	92.0	12.0
	50	249.0	218.0	13.0
	60	306.0	267.0	12.5
	70	382.0	343.0	12.0

and the one that gave the best results (summarized in Table 3) has step 3 modified as follows:

3. *Upper bounding.* Apply ICH to G' to get a first upper bound $UB'_{G'}$ on the size of a clique in G'.
   If $UB'_{G'} + |I'| \leq LB$, discard P' and go to 1.
   If $(UB'_{G'} + |I'|) \cdot \gamma > LB$, go to 4.
   Otherwise apply FCP to G' to get a second upper bound $UB_{G'}$.
   If $UB_{G'} + |I'| \leq LB$, discard P' and go to 1."

The code with this modified step 3 is MAXCLQ1, and its results are reported in Table 3. The value used for the multiplier γ in these runs was 0.85.

Every entry represents the average of ten runs. The column "LB at the root node" shows the lower bound obtained by finding an edge maximal triangulated (EMT) subgraph of G and a maximum clique in that subgraph. Comparing the entries of this column with those of the column showing the actual maximum clique sizes, we find that the EMT subgraph yields a very strong lower bound: for a total of 220 problems run, the difference between this lower bound and the actual maximum clique size is as follows:

0 in 61 cases,
1 in 92 cases,
2 in 62 cases,
3 in 5 cases.

The fourth column, "UB at the root node," shows the upper bound obtained by the fractional coloring procedure FCP2, which uses DSATUR as its integer coloring heuristic. Although this procedure, as shown in Table 1, improves the upper bound sometimes by more than one-fifth, nevertheless the gap between it and the lower bound is very significant, the ratio between the two being in the range 1.13–4.89. As a result of this situation, the number of search tree nodes that need to be explored is very large and grows fast with the graph density.

The last two columns of Table 3 compare the results of MAXCLQ1 with those of the branch-and-bound procedure of Babel [1], which seems to be the most efficient code in the literature. We see that in all cases but one MAXCLQ1 generates smaller search trees than Babel's code, sometimes by a factor of 2. This is due primarily to the stronger upper bounds obtained by the fractional coloring procedure. A second explanation lies in the strong lower bound generated at the root node by solving the EMT problem. The computing times are shown only for completeness; they are hard to compare, because the runs were performed on radically different computers. It should also be mentioned that the computational results of Babel are taken from [1], and the random problems solved with the two procedures, although of the same size and density, are not the same.

Table 4 reports on our computational experience with the DIMACS benchmark problems, which were contributed by various participants of the DIMACS Challenge. Unlike the randomly generated problems drawn from a uniform distribution, these problems have strong structure. Some come from coding theory, some are Steiner triple problems, some use graphs with carefully hidden large cliques. These problems are available from DIMACS on FTP.

Table 3. Unweighted maximum clique algorithm—random graphs

Graph		MAXCLQ1					Babel	
$\|V\|$	Density (%)	LB at root node	UB at root node	Maximum clique size	Search tree nodes	CPU (seconds)[1]	Search tree nodes	CPU (seconds)[2]
100	40	7.0	13.0	7.3	82.1	0.132	116	0.33
	50	8.9	15.9	9.3	156.2	0.225	252	0.79
	60	11.5	19.9	11.8	328.1	0.548	542	2.03
	70	14.0	24.0	14.8	489.6	1.287	1,018	5.41
	80	19.3	30.0	20.0	877.9	4.244	2,192	17.69
	90	29.3	39.7	30.5	757.2	8.260	1,698	28.40
	95	42.2	47.8	43.1	47.8	1.245	—[3]	—
200	40	8.0	21.6	8.8	1,294.4	1.732	1,241	4.78
	50	10.2	27.3	10.9	3,081.5	5.716	3,721	17.32
	60	12.8	34.0	14.0	10,666.7	26.897	15,747	87.33
	70	16.8	42.2	18.0	47,641.7	180.553	79,734	630.10
	80	23.5	53.0	25.5	327,933.5	2,331.262	893,158	10,761.17
300	40	9.1	30.3	9.5	5,278.3	10.031	6,307	28.49
	50	10.8	37.8	12.0	25,541.0	50.001	29,256	152.20
	60	13.9	47.3	15.3	116,220.7	378.750	183,454	1,242.19
	70	18.1	59.3	20.0	1,309,412.9	6,107.087	1,521,542	16,221.38
400	40	9.3	37.7	10.1	17,964.0	39.236	24,606	113.35
	50	11.8	48.0	12.9	126,496.1	283.775	141,892	831.20
	60	14.2	59.9	16.0	951,989.8	3,337.087	1,333,978	10,043.45
500	40	9.5	45.3	10.6	60,502.4	123.510	66,494	327.80
	50	11.9	58.2	13.1	429,055.9	1,118.942	476,392	3,082.30
	60	15.3	72.6	17.0	4,540,539.4	17,142.842	—	—

[1] NeXTstation.
[2] CDC CYBER 995.
[3] Not solved.

Table 4. Unweighted maximum clique algorithm—DIMACS benchmarks

$\|V\|$	Density (%)	LB at root node	UB at root node	Maximum clique size	Search tree nodes	CPU (seconds)	DIMACS problem ID
200	75	19	48	21	113,244	172.19	brock200_1
200	60	9	27	12	2,965	1.90	brock200_2
200	65	12	35	15	8,155	7.96	brock200_3
200	66	15	38	17	25,705	26.00	brock200_4
400	75	22	85	33	4,825,525	17,558.39	brock400_4
125	90	34	47	34	8,186	22.33	C125.9
200	8	12	12	12	1	0.03	c-fat200-1
200	16	24		24	1	0.02	c-fat200-2
200	43	58	67	58	29	0.13	c-fat200-5
500	4	14		14	1	0.08	c-fat500-1
500	7	26		26	1	0.11	c-fat500-2
500	19	64		64	1	0.24	c-fat500-5
500	37	126		126	1	0.64	c-fat500-10
500	50	12	59	13	505,355	338.06	DSJC500.5
200	90	38	56	44	140,966	719.67	gen200_p0.9_44
200	90	55	65	55	467	5.96	gen200_p0.9_55
64	90	32	32	32	1	0.01	hamming6-2
64	35	4	7	4	48	0.01	hamming6-4
256	97	128	128	128	1	0.34	hamming8-2
256	64	16	20	16	373	1.74	hamming8-4
1024	99	512	512	512	1	30.89	hamming10-2
28	56	4	4	4	1	0.00	johnson8-2-4
70	77	14	14	14	1	0.01	johnson8-4-4
120	76	8	8	8	1	0.01	johnson16-2-4
496	88	16	16	16	1	0.47	johnson32-2-4
171	65	11	18	11	4,164	3.39	keller4
45	93	16	19	16	23	0.02	MANN_a9
378	99	125	141	126	14,145	1,362.82	MANN_a27
300	24	8	19	8	832	0.80	p_hat300-1
300	49	25	41	25	1,613	5.83	p_hat300-2
300	74	34	66	36	171,229	826.27	p_hat300-3
500	25	9	29	9	6,105	4.83	p_hat500-1
500	50	35	63	36	31,746	228.11	p_hat500-2
700	25	9	37	11	14,040	16.08	p_hat700-1
700	50	43	82	44	252,892	2,848.49	p_hat700-2
1000	24	9	48	10	93,004	97.38	p_hat1000-1
1500	25	11	72	12	738,370	946.70	p_hat1500-1
200	70	16	33	30	635	1.69	san200_0.7_1
200	70	14	23	18	852	4.91	san200_0.7_2
200	90	46	73	70	10	0.26	san200_0.9_1
200	90	41	67	60	1,825	12.27	san200_0.9_2
200	90	36	56	44	353,617	1,678.79	san200_0.9_3
400	50	7	18	13	1,194	5.83	san400_0.5_1
400	70	21	51	40	20,913	190.59	san400_0.7_1
400	70	17	44	30	75,773	347.19	san400_0.7_2
400	70	16	31	22	161,585	585.28	san400_0.7_3
400	90	54	116	100	1,434,607	31,282.62	san400_0.9_1*
1000	50	8	22	15	21,897	365.26	san1000
200	70	17	42	18	40,496	52.91	sanr200_0.7
200	90	39	69	42	2,067,336	10,451.38	sanr200_0.9
400	50	11	48	13	112,932	105.43	sanr400_0.5
400	70	19	75	21	19,385,778	36,656.41	sanr400_0.7*

Table 5. Weighted maximum clique algorithm.

Graph		MAXCLQ2					BX92	
$\|V\|$	Density (%)	LB at root node	UB at root node	Maximum clique weight	Search tree nodes	CPU (seconds)[1]	Search tree nodes	CPU (seconds)[2]
100	10	28.5	29.5	28.5	9	1.28	13	0.06
	20	35.5	46.5	37.0	29	0.69	32	0.09
	50	62.0	87.5	69.5	76	1.87	99	0.27
	80	135.0	176.5	137.0	146	10.84	387	1.71
	90	209.9	237.5	213.5	148	21.42	269	1.84
200	10	34.5	46.0	34.5	67	1.87	68	0.26
	20	43.0	78.5	49.0	104	1.99	113	0.47
	50	77.0	159.0	77.5	1,254	36.52	1,726	5.76
	70	123.5	240.0	129.5	7,477	443.97	18,011	87.96
	80	184.5	306.5	186.5	27,605	2,789.75	60,038	536.52
300	10	34.5	55.0	36.5	152	5.84	152	0.75
	20	45.0	92.0	46.5	207	8.53	258	1.29
	50	80.5	218.0	86.5	7,859	282.66	13,611	48.73
	60	103.5	267.0	117.5	24,052	1,364.69	42,078	208.66
	70	141.0	343.0	150.0	175,996	14,246.57	427,806	2,590.35
400	10	35.5	67.0	35.5	238	9.67	250	1.48
	20	49.0	115.5	50.0	367	18.07	539	2.94
	40	76.0	222.5	78.5	8,197	324.64	11,379	43.08
	50	86.0	272.5	96.0	31,558	1,735.71	58,286	232.41
	60	110.0	336.0	117.5	178,089	12,636.95	345,979	1,880.25
500	10	40.0	86.5	42.5	305	13.16	316	2.32
	20	47.5	133.5	53.0	695	51.67	1,510	6.19
	30	59.0	194.0	68.5	4,815	216.29	6,419	25.52
	40	75.0	261.5	83.5	17,395	872.58	25,902	120.25
	50	89.5	325.0	101.5	97,318	6,223.33	179,050	842.21

[1] NeXTstation.
[2] HP 9000/835.

In running the DIMACS problems on the DEC alpha 300–400 AXP, we set a time limit of 18,000 seconds (5 hours), except for two cases: sanr400_0.7 and san400_0.9_1. An asterisk (*) marks the cases where this time limit was exceeded.

As Table 4 shows, our general-purpose maximum clique algorithm solves most of the DIMACS benchmark problems with a reasonable computational effort.

Finally, Table 5 describes the performance of MAXCLQ2, our branch-and-bound code for the weighted maximum clique problem. Again we find that the lower bound obtained by generating a triangulated subgraph of G and finding a maximum-weight clique in it, is pretty strong, though not to the same degree as in the unweighted case: in a couple of instances the ratio between LB and the maximum clique weight is less than 0.9. The gap between the upper and lower bounds remains large in the weighted case, too, with the ratio of the two bounds exceeding 3.0 in some cases. The last two columns compare the performance of MAXCLQ2 with the branch and bound code of Balas and Xue [5] (BX92), whose structure is similar to that of MAXCLQ2, except for

the fact that it does not use the fractional coloring procedure. We see that use of this improved upper bounding procedure substantially reduces the size of the search tree, sometimes by a factor of more than 2. Again, the computing times are not comparable because of differences in the computers. However, the computing times of MAXCLQ2 can be compared with those of MAXCLQ1, and the comparison shows that the time spent per search tree node is about five times higher in the weighted case than in the unweighted one. This is partly due to the fact that the weighted fractional coloring problem is inherently more difficult than the unweighted one, and partly to differences in the intensity of the implementation effort in the two cases.

To conclude, the fractional coloring procedure is undoubtedly a powerful device for strengthening the upper bound obtained by integer coloring. Its incorporation into a branch-and-bound framework has produced algorithms that compare favorably with other state-of-the-art procedures.

References

[1] L. Babel, Finding maximum cliques in arbitrary and special graphs, *Computing*, 46:321–341, 1991.

[2] L. Babel and G. Tinhofer, A branch and bound algorithm for the maximum clique problem, *ZOR-Methods and Models of Operations Research*, 34:207–217, 1990.

[3] E. Balas, A fast algorithm for finding an edge-maximal subgraph with a TR-formative coloring, *Discrete Applied Mathematics*, 15:123–134, 1986.

[4] E. Balas and J. Xue, Fast Maximum Clique Algorithms, TB17.4, TIMS/ORSA, Las Vegas, May 7–9, 1990.

[5] 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*, 20:209–221, 1991. Addendum, *SIAM Journal on Computing*, 21:1000, 1992.

[6] E. Balas and C. S. Yu, Finding a maximum clique in an arbitrary graph, *SIAM Journal on Computing*, 14:1054–1068, 1986.

[7] B. Bollobás, *Random Graphs*, Academic Press, New York, 1985.

[8] D. Brelaz, New methods to color the vertices of a graph, *Communications of the ACM*, 22:251–256, 1979.

[9] R. Carraghan and P. M. Pardalos, A Parallel Algorithm for the Maximum Weight Clique Problem, Technical Report CS-90-40, Department of Computer Science, Pennsylvania State University, 1990.

[10] F. D. J. Dunstan, Sequential Colorings of Graphs, *Proceedings of the 5th British Combinatorial Conference*, Vol. 19, 1975, pp. 456–463.

[11] M. Grötschel, L. Lovász, and A. Schrijver, Polynomial algorithms for perfect graphs, *Annals of Discrete Mathematics*, 21:325–356, 1989.

[12] F. T. Leighton, A graph coloring algorithm for large scheduling problems, *Journal of Research of the National Bureau of Standards*, 84:489–506, 1979.

[13] C. Mannino and A. Sassano, An Exact Algorithm for the Stable Set Problem, IASI-CNR Report No. 334, Rome, 1992.

[14] D. W. Matula, G. Marble, and J. D. Isaacson, Graph coloring algorithms, in R. C. Read (ed.), *Graph Theory and Computing*, Academic Press, London, 1972, pp. 109–122.

[15] D. J. A. Welsh and M. B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, *The Computer Journal*, 10:85–86, 1967.

[16] J. Xue, Fractional Coloring Heuristic with Application to the Maximum Clique Problem, ARIDAM V, Abstracts, RUTCOR Report No.2-90, May–June 1990, p. 67.

[17] J. Xue, Fast algorithms for vertex packing and related problems, Ph.D. thesis, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1991.

[18] J. Xue, Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem, *Networks*, to appear.