Alcom-FT Technical Report Series
ALCOMFT-TR-03-180

Simple and Fast: Improving a Branch-And-Bound Algorithm for Maximum Clique[⋆]

Torsten Fahle

University of Paderborn
Department of Mathematics and Computer Science
Fürstenallee 11, D-33102 Paderborn, Germany
tef@uni-paderborn.de

To Appear at ESA 2002 / Rom

Abstract. We consider a branch-and-bound algorithm for maximum clique problems. We introduce cost based filtering techniques for the so-called *candidate set* (i.e. a set of nodes that can possibly extend the clique in the current choice point). Additionally, we present a taxonomy of upper bounds for maximum clique. Analytical results show that our cost based filtering is in a sense as tight as most of these well-known bounds for the maximum clique problem.
Experiments demonstrate that the combination of cost based filtering and vertex coloring bounds outperforms the old approach as well as approaches that only apply either of these techniques. Furthermore, the new algorithm is competitive with other recent algorithms for maximum clique.

Keywords: maximum clique, branch-and-bound, constraint programming, cost based filtering.

1 Introduction

A *Clique* is an undirected graph $C = (V_C, E_C)$ where all nodes in V are pairwise adjacent. The *Maximum Clique Problem (MC)* on a graph $G = (V, E)$ asks for a clique $C = (V_C, E_C)$ in G having a largest node set $|V_C|$. It is known that MC is NP-hard [3].

Despite its complexity, the maximum clique problem is important in many fields. Cliques are used in integer programming for presolving [19] and deriving valid cuts [1]. They appear in coding theory, fault diagnostics, or pattern recognition. For an overview on applications we refer to [7].

In this paper, we introduce some simple cost based domain filtering techniques [13]. We test them on a branch-and-bound algorithm proposed by Carraghan and Pardalos [9]. Their algorithm is still believed to be one of the fastest clique solver for sparse graphs. It recursively enlarges a clique by adding nodes of a candidate set. Pruning based on simple bounds is used to cut off useless parts of the search tree.

Our cost based filtering tightens the candidate set by removing or fixing certain nodes. We show analytically that this domain filtering in a sense is as tight as typical upper bounds for MC. As we will see later, our filtering dominates 7 out of 8 bounds

⋆ Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

proposed for MC. The bound that is not dominated (but also does not dominate the filtering) is the vertex coloring bound. We add it to the algorithm and achieve a significant improvement on the DIMACS benchmark set. Within a 6 hours time limit the enhanced algorithm can prove optimal results for 45 out of 66 benchmark instances, whereas the old approach is only able to finalize 33 instances. Furthermore, it turns out that domain filtering improves 56 out of 66 test cases compared to applying coloring bounds alone. (An earlier version of this work was presented at the CPAIOR'02 workshop [10]).

1.1 Literature review

Cost based filtering [13] aims at fixing variables with respect to the objective function. It typically combines techniques from Optimization with domain filtering stemming from Constraint Programming. It is an important technique when solving complex optimization problems. Optimization constraints for various applications have been addressed e.g. in [13, 14, 16, 11, 12, 18].

The maximum clique problem has attracted researchers in computer science, operations research and other fields for many years. We can therefore only scratch some results and have to refer to some recent overview in [7] for further details.

Balas and Yu [5] presented a new idea for implicitly enumerating cliques. They search for a maximal induced triangulated subgraph T of G in which then a maximum clique C is searched. In a second phase they extend T in a way that does not enlarge the max. clique. Applied in a branch-and-bound scheme their approach was quite successful as the bounds generated turned out to be quite tight. A branch-and-bound algorithm using vertex coloring bounds was presented by Wood [20].

Cuts for an integer programming formulations of the maximum clique problem have been investigated in [4]. The linear independent set formulation, $\max\{\sum_{i=1}^{n} x_i \mid x_i + x_j \leq 1, \forall\, \{i,j\} \notin E,\ x \in [0\ldots1]^n\}$ was used in their approach.

Recently, Caprara, Pisinger and Toth proposed a solver for the quadratic Knapsack problem which was applied also to the maximum clique problem [8].

Enumerating algorithms have been contributed by many researchers. Of certain interests for this work are the methods of Carraghan and Pardalos, and Östergård. We will return to those algorithms in Sec. 2. Before that we will briefly introduce some notation. In Sec. 2.1 we will present the filtering constraints. Some analytical results are given in Sec. 3, empirical studies are presented in Sec. 4. Finally, we conclude.

1.2 Notation

Throughout the paper we denote the neighborhood of a node $v \in V$ as $N(v) = \{u \in V | \{u,v\} \in E\}$ and the degree of a node $v \in V$ as $\delta(v) = |N(v)|$. Given a node set U, the graph $G_U = (U, E \cap (U \times U))$ is the graph *induced* by U. We write $N_U(v) = \{u \in U | \{u,v\} \in (U \times U) \cap E\}$ and the $\delta_U(v) = |N_U(v)|$ if we speak of the neighborhood, or degree, resp. of a node v in the subgraph of G induced by U. For a clique C we call $P = \bigcap_{v \in C} N(v)$ the *candidate set of* C. A clique $\widetilde{C}$ is an *extension* of a clique C if $C \subseteq \widetilde{C}$. We use the notation $\widetilde{C}(C)$ for extensions. To ease the notation, we often identify a clique C with its set of nodes. Finally, $\mathcal{Z}(G) = \{I \subseteq V \mid G_I \text{ is a connected component in } G\}$ contains the *node sets of connected components* of the graph $G = (V,E)$

2 Simple Branch-And-Bound Algorithms for Maximum Clique

Carraghan and Pardalos [9] enumerate the cliques of a graph $G = (V, E)$ according to some lexicographical order. Without pruning, the algorithm finds the largest clique in G_V containing node v_1, then the largest clique in $G_{V \setminus \{v_1\}}$ containing v_2, etc. A candidate set $P = \bigcap_{v \in C} N(v)$ containing all nodes that are adjacent to the clique C currently under construction is used for bounding. A simple depth first search and static variable ordering is used to guide the branch-and-bound (Alg. 1). The well-known `dfmax` [22] program (D. Applegate and D. Johnson) is an efficient implementation of that idea.

Algorithm 1 A simple Algorithm to find the Maximum Clique C^*

function findClique(set C, set P)

1: **if** $(|C| > |C^*|)$ **then**
2: $\quad C^* \leftarrow C$
3: **if** $(|C| + |P| > |C^*|)$ **then**
4: $\quad$ **for all** $p \in P$ in predetermined order: **do**
5: $\quad\quad P \leftarrow P \setminus \{p\}$
6: $\quad\quad C' \leftarrow C \cup \{p\}$
7: $\quad\quad P' \leftarrow P \cap N(p)$
8: $\quad\quad$ findClique(C', P')

function main( )

1: $C^* \leftarrow \emptyset$
2: findClique($\emptyset$, V)
3: return C^*

Östergård [17] developed a variant of the previous method. Instead of determine the cliques in a decreasing node set, his algorithm starts on $G_{\{v_n\}}$ and determines the largest clique on that graph. Then it considers $G_{\{v_{n-1}, v_n\}}$ and determines the largest clique there. By adding more and more nodes, and determine cliques on each of those sets, it ends up with a largest clique in G. During that incremental process, bounds on the largest cliques containing a certain node can be determine as a by-product. With these bounds at hand, Östergård can speed up finding cliques significantly.

2.1 Observations on the Candidate Set

Both algorithms sketched above share the idea of a candidate set $P = \bigcap_{v \in C} N(v)$ containing only those nodes that may extend the clique C currently under construction. P is also used for pruning (line 3), since the largest clique to be discovered in the current part of the search tree can contain $|C| + |P|$ nodes at most. We will show now that two simple observations can help to tighten the candidate set, and thus, potentially reduce the number of choice points explored.

Looking at the nodes in P in more detail, we can characterize those that can never extend a given clique to a maximum clique:

Lemma 1. *Let $G = (V, E)$ be a graph, C be a clique on G, and P be a candidate set, i.e. $P = \bigcap_{v \in C} N(v)$. Furthermore, let $\sigma \in \mathbb{N}$ be a lower bound on the size of a maximum clique in G. Then, for every $v \in P$ such that $|C| + \delta_P(v) < \sigma - 1$, v cannot extend C to a maximum clique of G.*

(proof omitted due to space limitations)

The next lemma identifies nodes that will be members of any extension of the current clique to a maximum one:

Lemma 2. *Let $G = (V, E)$ be a graph, C be a clique on G, and P be a candidate set, i.e. $P = \bigcap_{v \in C} N(v)$. Then, every $v \in P$ such that $\delta_P(v) = |P| - 1$, is contained in any maximum clique of G that also contains C.*

(proof omitted due to space limitations)

Figure 1(a) represents the settings of the algorithms described above, Figure 1(b),(c) sketches the situation described by lemma 1, 2.

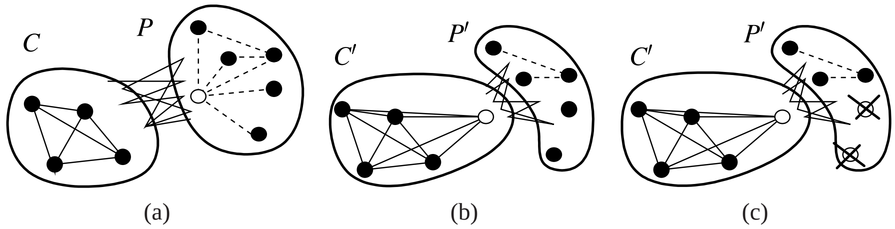


Fig. 1. (a) a clique C and its candidate set P. (b) applying lemma 2 fixes one more node. (c) as a result, two nodes have degree zero now, and can be eliminated according to lemma 1.

2.2 An Improved Branch-And-Bound Algorithm

With these observations at hand we are ready to present an improved enumeration algorithm for MC (Alg. 2, on page 5). Between lines 7 and 8 of the first algorithm we insert some domain filtering technique based on lemma 1 and 2. We know that the next improvement has to be better than the best clique found so far. Hence, we use $(|C^*| + 1)$ as a lower bound for the necessary checks.

Domain filtering does not depend on the order in which the two lemmas are applied. The only situation where the lemmas can be in conflict arises when there is $v \in P'$ such that $|P'| - 1 = \delta_{P'}(v)$ and $\delta_{P'}(v) + |C'| < |C^*|$. Then $|P'| - 1 + |C'| < |C^*| \Rightarrow |P'| + |C'| \leq |C^*|$. But in such a case we cannot improve on C^* anymore, and that part of the search tree is pruned in the next recursion level by line 3 of Alg. 2. Furthermore, it is easy to see that the order in which nodes are considered within each lemma does not change the final result. Also, it is clear that when applying lemma 1 first, then lemma 2, no node will fulfill the requirements of lemma 1 again in that branch-and-bound node.

3 Upper Bounds vs. Domain Filtering: Some Analytical Results

In this section we present some analytical results on the two constraints. Especially, we show that some upper bounds for the MC used in OR methods are already subsumed by the domain filtering.

Algorithm 2 An Improved Algorithm for the Maximum Clique Problem

function findClique2(set C, set P)

if $(|C| > |C^*|)$ **then**
    $C^* \leftarrow C$
if $(|C| + |P| > |C^*|)$ **then**
    for all $p \in P$ in predetermined order: **do**
        $P \leftarrow P \setminus \{p\}$
        $C' \leftarrow C \cup \{p\}$
        $P' \leftarrow P \cap N(p)$
        // domain filtering
        // reduce possible set
        while $(\exists\, v \in P' : \delta_{P'}(v) + |C'| < |C^*|)$ **do**
            $P' \leftarrow P' \setminus \{v\}$ *// lemma 1*
        // increase required set
        while $(\exists\, v \in P' : \delta_{P'}(v) = |P'| - 1)$ **do**
            $C' \leftarrow C' \cup \{v\}$ *// lemma 2*
            $P' \leftarrow P' \setminus \{v\}$ *// lemma 2*
        // here it holds: $(\forall\, v \in P' : (|C^*| - |C'|) \le \delta_{P'}(v) < (|P'| - 1)$
        findClique2(C', P')

3.1 Accuracy

We still assume the setting presented in Sec. 2: C is the clique currently under construction, $P = \bigcap_{v \in C} N(v)$ the corresponding candidate set. The following lemma lists some well-known upper bounds $\mathcal{U}_i(C,P)$ for a maximal clique $C^*(C)$ (see also [7]):

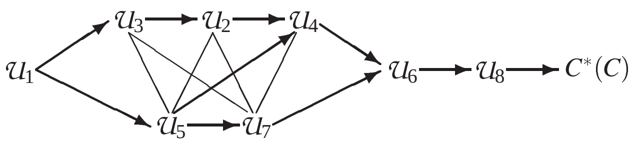
Lemma 3 (Upper Bounds for MC). *The following upper bounds hold for MC:*

1. *Only nodes from the candidate set can extend C:* $\mathcal{U}_1(C,P) = |C| + |P|$.
2. *A node with maximum degree in P limits the size of any extension of C to a maximal clique:* $\mathcal{U}_2(C,P) = |C| + \max\{\delta_P(v) + 1 \mid v \in P\}$.
3. *Only one connected component in P can extend C:* $\mathcal{U}_3(C,P) = |C| + |I^{max}|$, *where* I^{max} *denotes the node set of the largest connected component in* $\mathcal{Z}(G_P)$
4. *A k-clique requires k nodes with degree at least* $k-1$*:*
$$\mathcal{U}_4(C,P) = |C| + \max\{k \mid \exists\, v_1 < \cdots < v_k \in P,\ \delta_P(v_i) \ge k-1\}$$
5. *A k-clique has* $k(k-1)/2$ *edges:* $\mathcal{U}_5(C,P) = |C| + \max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \le |E_P|\}$, *where* E_P *is the edge set of the graph* $G_P = (P, E_P)$ *induced by P.*
6. *Apply* $\mathcal{U}_4(C,P)$ *on connected components of* G_P *only:*
$$\mathcal{U}_6(C,P) = |C| + \max_{I \in \mathcal{Z}(G_P)} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_P(v_i) \ge k-1\}$$
7. *Apply* $\mathcal{U}_5(C,P)$ *on connected components of* G_P *only:*
$$\mathcal{U}_7(C,P) = |C| + \max_{I \in \mathcal{Z}(G_P)} \max\{k \in \mathbb{N} \mid \tfrac{k(k-1)}{2} \le |E_I|\},$$
where E_I *is the edge set of the graph* $G_I = (P, E_I)$ *induced by I.*
8. *Any k-clique needs k colors in a vertex coloring:* $\mathcal{U}_8(C,P) = |C| + \chi(G_P)$, *where* $\chi(G_P)$ *denotes the (vertex) chromatic number of the graph induced by P.*

Proof. For applications of these bounds and further information we refer to [9] for $\mathcal{U}_1$, [5, 20] for $\mathcal{U}_8$, and [7] for some bound similar to $\mathcal{U}_5$. The other bounds are simple extensions of $\mathcal{U}_1$.

Obviously, some of these bounds are contained in others leading to the following relationship (to our knowledge, such a taxonomy has not been presented before, though some relations are obvious and have probably been used earlier):

Lemma 4 (Taxonomy of Bounds). *Let $G = (V,E)$ be a graph, C a (not necessarily maximum) clique in G, and $P = \bigcap_{v \in C} N(v)$ the corresponding candidate set. Then*



($\mathcal{U}_i \rightarrow \mathcal{U}_j \;:\Leftrightarrow\; \mathcal{U}_i \geq \mathcal{U}_j$ and $\mathcal{U}_i - \mathcal{U}_j \;:\Leftrightarrow$ neither $\mathcal{U}_i \geq \mathcal{U}_j$, nor $\mathcal{U}_i \leq \mathcal{U}_j$).

(proof omitted due to space limitations)

Next, we show that the filters described in Sec. 2.1 already include bounds $\mathcal{U}_1$ – $\mathcal{U}_7$. I.e. after applying lemma 1, 2 to (C,P) and obtaining a new (C',P'), those bounds cannot prune the current part of the subtree.

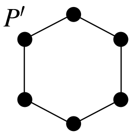
Theorem 1. *Let $G = (V,E)$ be a graph, C be a clique on G, and P be a candidate set, i.e. $P = \bigcap_{v \in C} N(v)$. Furthermore, let $\sigma \in \mathbb{N}$ be a lower bound on the size of a maximum clique in G. Let $|C| + |P| > \sigma$ and let C' and P' be the node sets after applying the domain filtering of algorithm 2, lines 8 – 16. Then, $\mathcal{U}_i(C',P') > \sigma$ for $i = 1,\dots,7$.*

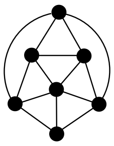
Proof. According to lemma 4 it is sufficient to prove $\mathcal{U}_6(C',P') > \sigma$. So let us assume, $\mathcal{U}_6(C',P') \leq \sigma$, i.e. $|C'| + \max\limits_{I \in \mathcal{Z}(G_{P'})} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_{P'}(v_i) \geq k-1\} \leq \sigma$.
After applying lines 8 – 16 of algorithm 2 we have: $\forall\, v \in P' :\ (\sigma - |C'|) \leq \delta_{P'}(v)$.
$\Rightarrow\ \forall\, v \in P' :\ |C'| + \delta_{P'}(v) \geq \sigma \geq |C'| + \max\limits_{I \in \mathcal{Z}(G_{P'})} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_{P'}(v_i) \geq k-1\}$
$\Rightarrow\ \forall\, v \in P' :\ \delta_{P'}(v) \geq \max\limits_{I \in \mathcal{Z}(G_{P'})} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_{P'}(v_i) \geq k-1\} =: k'$

So the degree of all nodes in P' is at least k', especially, there is a connected component in P' having at least $k'+1$ nodes with degree larger than k' — being a contradiction to the maximality of $\mathcal{U}_6(C',P')$ □

Bound $\mathcal{U}_8$ is not included in the above theorem. The following two examples show that depending on the situation we may subsume the coloring bound, or it may be more accurate than filtering alone:
Assume, the candidate set after domain filtering is given as sketched in the right graph. Furthermore, we have guessed $\sigma = 4$, and we have already $|C'| = 2$. P' cannot be reduced further by domain filtering, as $\forall\, v \in P' :\ \underbrace{\sigma - |C'|}_{=2} \leq \underbrace{\delta_{P'}(v)}_{=2} < \underbrace{|P'| - 1}_{=5}$. Now, as $\chi(G_{P'}) = 2$ we get $\mathcal{U}_8(C',P') = 4 = \sigma$. Thus, applying the coloring bound here can prune parts of the search tree that would still be considered when using filtering only.

P'



P' 

Things change when considering the left graph: Having $\sigma = 3$ and $|C'| = 1$, domain filtering cannot fix any further node in P' as $\forall\, v \in P' : \underbrace{\sigma - |C'|}_{=2} \leq \underbrace{\delta_{P'}(v)}_{3,4,5} < \underbrace{|P'| - 1}_{=6}$. As $\chi(G_{P'}) = 4$ here, we obtain $\mathcal{U}_8(C', P') = 5 > \sigma$ showing that in this case the coloring bound cannot prune the remaining subtree.

3.2 Computational Complexity

Better accuracy of bounds is usually payed by higher running time when calculating bounds. We briefly compare running time per choice point for calculating bounds or domain filtering, respectively.

As benchmark graphs for cliques are typically rather dense, we assume that the graph G is stored as an adjacency matrix. Also, we assume, that $|C'|$ can be determined in $O(|C'|)$. $\mathcal{U}_1$ requires the size of P, which can be determined in $O(|P|)$ within a reasonable data structure. Also the max. degree, required by $\mathcal{U}_2$, can be determined in that time if degree-information is available (otherwise: $O(|P|^2)$).

A largest connected component in P can be detected by a DFS in a dense graph in time $O(|P|^2)$. However, $\mathcal{U}_4$ being tighter than $\mathcal{U}_3$ can be computed in time $O(|P| \log |P|)$ if degree information is available (or in quadratic time otherwise). Also $\mathcal{U}_6$ and $\mathcal{U}_7$ base their information on connected components, and need therefore time $O(|P|^2)$.

If degree information is at hand, bound $\mathcal{U}_5$ can be determined in $O(|P|)$, otherwise this bound needs to visit G, giving time complexity $O(|P|^2)$.

Vertex coloring, needed for $\mathcal{U}_8$ is NP-hard in general. For bound calculation, however, heuristics are used. They need to visit each node and each edge a constant number of time, giving running time $O(|P|^2)$ in a dense graph (see e.g. [2, 5, 20]).

Our domain filtering, as described in algorithm 2, lines 8 – 16, does perform $|P|$ loops at most, as each loop excludes one node from P. Being dependent on degree information, each exclusion of a node requires an update of the degrees of adjacent nodes. That step requires at most time $O(|P|)$ leading to a total running time of $O(|P|^2)$ for the entire path from the root of the search tree to one of its leaves.

The domain filtering is therefore at least as accurate as bounds $\mathcal{U}_1$–$\mathcal{U}_7$, and requires not more running time asymptotically, than the more effective bounds of these seven candidates.

4 Experiments

Obviously, as the tighter bounds need some more computational effort, there is a trade-off between the effectiveness of pruning techniques and the overall efficiency of the approach. In this section we study the empirical behavior of the new approach. We use the well-established DIMACS benchmark set [15, 21] for the comparison. It consists of 66 instances, ranging from 28 – 3361 nodes, and 420 – 11 million edges. The density of the underlying graphs range from 3.5% – 99.8%. To our knowledge for some of these instances optimal solutions have not been proven so far.

4.1 The Implementation

An efficient implementation has to consider some more "tricks" than those described in the algorithms 1 and 2. When determine $P' = P \cap N(v)$, we use a loop running over all elements $p \in P$. If $\{p,v\} \in E$, p is copied to P'. For non-adjacent elements we decrease a counter m that contains the number of elements still needed in order to obtain $|C'|+|P'| > |C^*|$. Should m ever be decremented below zero we skip that recursion-level immediately. (We refer to [22] for more details).

For the algorithm using domain filtering we also update degree information on the subgraph induced by P. That is, we decrease the degree of a node $p' \in P'$ if an adjacent node $p \in P$ is not copied to P'. By keeping a copy of P in each recursion level we can easily perform incremental updates on the degrees before entering the next recursion level.

In lines 8 – 16 of algorithm 2 we perform domain filtering. We implement the two while loops using a stack to store those nodes, that either have not been copied to P' (lines 5 – 7), or have been removed during domain filtering (lines 8 – 16). In the implementation, we take the next element p from the stack, decrease the degree of all adjacent nodes p', perform the degree tests on those nodes, and in case of removal, push p' onto the stack. We continue until the stack is empty.

For the coloring bounds we use four heuristics: Two of them follow the DSATUR approach of Brelaz [2], e.g. choose the node with a maximum number of colored neighbors and assign the minimal unused color to that node. The other two methods build an independent set for each color class. If the current set cannot be extended anymore, a new set is opened, and the number of sets required is the heuristic chromatic number. As we apply nodes in increasing, and decreasing order of degrees, we obtain four heuristics, and the minimum number calculated is used as the bound.

4.2 Numerical Results

All algorithms were coded in C++ and compiled by the GNU g++ 2.95.3 compiler using full optimization. Our benchmark tests run on a Pentium III-933 PC with 512 MB ram operating Linux 2.4.17. We stopped each run after 6 hours (21 600 sec).

Effects of Domain Filtering In table 2 we compare our implementation[1] of `dfmax` using only bound $\mathcal{U}_1$ (Alg. 1), an implementation using coloring bounds only (χ), another using domain filtering only (*DF*), and a forth using both ($\chi + DF$). The latter three implementations correspond to Alg. 2.

As expected, the number of choice points decreased considerably when applying domain filtering. The savings range from 3.6 (mann_a9) to 30.4 (p_hat1500_1) when comparing `dfmax` with the *DF* approach. In contrast, running time of the current code is typically a factor of 2 times slower than the `dfmax` code. Here, domain filtering is

[1] For a fare comparison, we decided to use identical subroutines for all implementations. We adapted `dfmax` to our settings resulting in a slightly faster algorithm ($< 2\%$) than the original `dfmax`

computationally too expensive compared to the simple $\mathcal{U}_1$ bound: It is still cheaper to traverse larger useless parts of the search tree than detecting these parts.

The c-fat*xxx* instances play a special role in this context. They are quite easy to solve using domain filtering. Each instance requires 5 choice points only to prove optimality, with running time being negligible. The `dfmax` algorithm uses dramatically more choice points (a factor of more than one million in the case of c-fat200-5, and more than 10^9 for c-fat500-10). Wood [20] also reports only a few number of choice points (1–27) for his approach on those instances. Hence, we consider this behavior as a drawback of `dfmax` rather than as an advantage of our filtering.

Using vertex color bounds heavily helps to detect useless parts of the search tree. Out of the 66 benchmark instances, 45 could be solved to optimality when using coloring bounds, whereas only 33 can be finalized by `dfmax`. Again, it turns out that DF helps to reduce choice points considerable. Additionally, also a positive impact on running time can be seen. Since the coloring bounds a rather expensive to compute, any reduction of the remaining graph helps to decrease overall running time. In 56 out of 66 cases, the overall running time benefits from domain filtering (measured as: (1) better result, (2) (if results equal) being faster, (3) (if results and running time equal) finding best solution earlier).

It should be noted that especially for simple instances running time is negatively affected by more sophisticated approaches (e.g. some of the p_hat*xx* instances). For those cases some control mechanism that switches off expensive techniques should be used.

Comparison to other Algorithms Though the focus of this paper is not on designing a fastest clique solver, we also compare our results to some recent results presented in the literature. We compare results of Wood [20], Balas and Xue [6], and Östergård [17] against our $\chi + DF$ results. Since the results in column one of table 2 refer to the Carraghan/Pardalos algorithm, we have shown already, that we beat that algorithm in many cases. Unfortunately, for the other approaches different machines where used as well as different time limits. Hence, we can only try to show a general tendency, rather than comparing details. Also, the papers mentioned do not present results on all 66 instances, whereas we have good results on some of the omitted instances. It should be noted here, that we do not use lower bound heuristics, that sometimes give good or even optimal results in the first node of the branch-and-bound tree.[2]

Wood's approach is a branch-and-bound using fractional coloring and lower bound heuristics. It can solve 38 instances to optimality using running times up to 19 hours on a SUN S10. In 10 cases, the heuristics can prove optimality in the root node already (i.e. lower and upper bounds are the same). In the remaining 29 cases our approach always uses less choice points than his fastest method (MC_C), the typical factor being 4–8. When considering MC_D (the approach using the fewest number of choice points), our approach wins in 14 cases, MC_D has less choice points in 15 cases. However, in at least 20 cases the running time is higher than ours.[3].

[2] The reason being that we want to show the effects of bounds and filtering clearly

[3] A SUN Sparc 10/51 is about 10 times slower than our computer, and we transformed the running times with this factor

Balas and Xue developed a heuristics to determine fractional coloring bounds within a branch-and-bound framework. They used a DEC Alpha 300–400 AXP and a 5 hours time limit. It is not noted whether the result given are all proven optimal results, or best results after stopping the algorithm. Again, we omit those cases, where heuristics already prove optimality in the root node. In 19 out of 38 cases we use less choice points than their approach. Assuming that their computer is about 4 times slower than ours, we beat that algorithm in 13 cases.

Östergård's approach was already described in Sec. 2. As no numbers on choice points are given, we can only compare running times. The computer used is a 500 MHz PC, roughly 2 times slower than ours. He considers 38 instances, of which we can solve 8 faster than his approach. Since our domain filtering approach is compatible with the ideas of Östergård, combining both would be an interesting experiment. However, Östergård also mentions that his approach sometimes uses much more time than other solvers, so a detailed experimental evaluation on all DIMACS instances is necessary here.

5 Conclusions and Future Work

We presented some simple constraints that tighten the candidate set used in solvers for maximum clique instances. Using a taxonomy of typical bounds for MC we were able to show that the tightened candidate set is in a sense as least as tight as seven of these bounds.

A brief complexity analysis furthermore showed, that the asymptotic running time is not higher than the time complexity of efficient bounds used. In an experimental study we showed significant reduction of choice points when using the new approach. Introducing some tighter bounds further improves the algorithm, and the combination of tight bounds and domain filtering is the most promising approach tested in this paper. Hence, the combination of a simple branch-and-bound, domain filtering, and coloring bounds defines a simple and fast solver for MC.

Obviously, the bounds' quality is most important for the overall efficiency. They prune the search tree and lead to the promising parts. Domain filtering fixes variables in a choice point. As we thereby avoid branching on those variables, domain filtering helps to stabilize the search within the solution space.

Since the domain filtering approach is rather generic, it can be combined easily with other MC algorithms – like the one of Östergård.

In this paper, we focused on the effects of domain filtering for MC, and the algorithms tested were designed to show these effects rather then to be extremely fast. When considering other techniques as well, running time of the algorithm can be reduced further. As shown by others, lower bound heuristics can prove optimality in the root node for 10 DIMACS instances. Also for the p_hat*xx* and the mann_a*xx* instances applying lower bound heuristics provides a solution better than the best solution found after 6 hours of pure systematic search. Since our domain filtering depends on a lower bound for the maximum clique, such a heuristic also improves effects of domain filtering.

The ordering in which nodes are considered during search can have a severe impact on running time, too. We found in an additional study that some instances benefit from

				our dfmax			χ			*DF*			$\chi+DF$	
file	$\lvert V\rvert$	**density**	$\lvert C^*\rvert$	**ChPts**	**time**	$\lvert C^*\rvert$	**ChPts**	**time**	$\lvert C^*\rvert$	**ChPts**	**time**	$\lvert C^*\rvert$	**ChPts**	**time**
brock200_1	200	74.54%	21	38043497	**28.82**	21	74613	181.82	21	3350353	50.61	21	66042	139.46
brock200_2	200	49.63%	12	54314	**0.05**	12	530	0.64	12	3539	0.16	12	437	0.46
brock200_3	200	60.54%	15	453265	**0.41**	15	2487	4.93	15	32170	0.99	15	2332	3.34
brock200_4	200	65.77%	17	2161909	**1.73**	17	9708	16.86	17	167750	3.64	17	8779	12.27
brock400_1	400	74.84%	≥**27**	24423311512	≥21600	≥24	6682621	≥21600	≥25	1115779306	≥21600	≥24	7513237	≥21600
brock400_2	400	74.92%	≥**29**	19952147706	≥21600	≥**29**	7159353	≥21600	≥**29**	809348442	≥21600	≥**29**	6743670	≥21600
brock400_3	400	74.79%	≥**31**	25092151368	≥21600	≥24	6208742	≥21600	≥24	1142054135	≥21600	≥24	7061823	≥21600
brock400_4	400	74.89%	**33**	20152485865	20577.7	≥25	5370110	≥21600	≥25	1016877999	≥21600	≥25	5950301	≥21600
brock800_1	800	64.93%	≥**21**	21193921452	≥21600	≥20	7825966	≥21600	≥**21**	579955608	≥21600	≥**21**	9205790	≥21600
brock800_2	800	65.13%	≥**21**	23396947454	≥21600	≥20	7861777	≥21600	≥20	684885264	≥21600	≥20	9747602	≥21600
brock800_3	800	64.87%	≥**21**	22077418714	≥21600	≥20	7596314	≥21600	≥20	645967913	≥21600	≥20	9464943	≥21600
brock800_4	800	64.97%	≥**26**	21514428210	≥21600	≥20	7688179	≥21600	≥21	643687866	≥21600	≥20	9529585	≥21600
c-fat200-1	200	7.71%	12	52	0.01	12	34	0	12	5	0	12	5	0.01
c-fat200-2	200	16.26%	24	444	0	24	70	0.01	24	5	0	24	5	0.01
c-fat200-5	200	42.58%	58	268435599	1307.45	58	172	0.04	58	5	0	58	5	0.01
c-fat500-1	500	3.57%	14	71	0.01	14	40	0.02	14	5	0.01	14	5	0.02
c-fat500-10	500	37.38%	≥124	15086121946	≥21600	**126**	376	0.3	**126**	5	0.01	**126**	5	0.03
c-fat500-2	500	7.33%	26	683	0.01	26	76	0.03	26	5	0.01	26	5	0.02
c-fat500-5	500	18.59%	64	5703074	5.71	64	190	0.06	64	5	0.01	64	5	0.03
hamming10-2	1024	99.02%	≥512	2963756831	≥21600	**512**	513	8.92	≥512	902493400	≥21600	**512**	257	**7.74**
hamming10-4	1024	82.89%	≥**32**	35273440082	≥21600	≥**32**	7782157	≥21600	≥**32**	2933556637	≥21600	≥**32**	7677209	≥21600
hamming6-2	64	90.48%	32	42787	0.03	32	33	0	32	1891	0.01	32	17	0.01
hamming6-4	64	34.92%	4	221	0	4	28	0	4	47	0	4	31	0
hamming8-2	256	96.86%	≥128	13603244188	≥21600	**128**	129	0.14	≥128	1582399395	≥21600	**128**	65	**0.11**
hamming8-4	256	63.92%	16	3742143	**4.05**	16	2063	9.97	16	252418	7.47	16	1950	9.16
johnson16-2-4	120	76.47%	8	4177630	**1.51**	8	96767	12.48	8	904446	3.57	8	126460	11.87
johnson32-2-4	496	87.88%	≥**16**	49631406919	≥21600	≥**16**	133437402	≥21600	≥**16**	4918222342	≥21600	≥**16**	139645893	≥21600
johnson8-2-4	28	55.56%	4	90	0	4	13	0	4	21	0	4	15	0
johnson8-4-4	70	76.81%	14	12544	**0.01**	14	50	0.05	14	1205	0.02	14	39	0.04
keller4	171	64.91%	11	1275236	**0.79**	11	2664	3.93	11	107086	1.53	11	1771	3.79
keller5	776	75.15%	≥24	23430537443	≥21600	≥24	2694646	≥21600	≥24	737283168	≥21600	≥**25**	2671681	≥21600
keller6	3361	81.82%	≥**43**	17888655376	≥21600	≥**43**	1390946	≥21600	≥**43**	540201863	≥21600	≥**43**	1362220	≥21600
MANN_a9	45	92.73%	16	329235	0.1	16	49	0.03	16	92210	0.11	16	31	**0.01**
MANN_a27	378	99.01%	≥116	15557799773	≥21600	≥126	40001	≥21600	≥116	23120760471	≥21600	**126**	39351	15523.3
MANN_a45	1035	99.63%	≥220	12119272496	≥21600	≥327	13098	≥21600	≥234	23274505520	≥21600	≥**331**	10367	≥21600
MANN_a81	3321	99.88%	≥438	5947703016	≥21600	≥994	30709	≥21600	≥467	23035700588	≥21600	≥**996**	8113	≥21600
p_hat1000-1	1000	24.48%	10	1801019	**2.18**	10	28375	28.99	10	81573	9.9	10	19430	24.65
p_hat1000-2	1000	49.01%	≥42	19361957667	≥21600	≥**44**	990436	≥21600	≥41	1149227032	≥21600	≥**44**	968615	≥21600
p_hat1000-3	1000	74.42%	≥49	21936773057	≥21600	≥**50**	928896	≥21600	≥47	1603605303	≥21600	≥**50**	920905	≥21600
p_hat1500-1	1500	25.34%	12	13825580	**20.77**	12	140943	299.62	12	454914	97.29	12	136620	179.65
p_hat1500-2	1500	50.61%	≥46	24069645860	≥21600	≥**52**	705644	≥21600	≥46	1893106666	≥21600	≥**52**	656300	≥21600
p_hat1500-3	1500	75.36%	≥53	26020312163	≥21600	≥55	849096	≥21600	≥53	1920941837	≥21600	≥**56**	885013	≥21600
p_hat300-1	300	24.38%	8	11634	**0.02**	8	271	0.17	8	450	0.06	8	254	0.11
p_hat300-2	300	48.89%	25	1553140	**1.25**	25	1443	5.34	25	182168	2.52	25	1121	4.52
p_hat300-3	300	74.45%	36	1960518988	1614.35	36	210265	1668.4	36	251846120	2561.11	36	171086	**1285**
p_hat500-1	500	25.31%	9	89908	**0.11**	9	778	1.28	9	4978	0.43	9	690	0.9
p_hat500-2	500	50.46%	36	292274682	**253.76**	36	38031	363.22	36	35324032	459.89	36	32413	305.9
p_hat500-3	500	75.19%	≥44	23974852012	≥21600	≥**48**	1113372	≥21600	≥43	2103038234	≥21600	≥**48**	993862	≥21600
p_hat700-1	700	24.93%	11	343857	**0.44**	11	2617	4.7	11	20630	1.94	11	2195	4.01
p_hat700-2	700	49.76%	44	10969888234	10397.8	44	231488	3921.47	44	1394776054	16637.8	44	188823	**3129.95**
p_hat700-3	700	74.80%	≥49	22686699084	≥21600	≥**54**	737898	≥21600	≥49	1851669803	≥21600	≥**54**	653774	≥21600
san1000	1000	50.15%	≥10	11298828641	≥21600	**15**	29502	**4158.17**	≥10	207288356	≥21600	**15**	35189	4566.13
san200_0.7_1	200	70.00%	30	14265131069	5713.23	30	348	**2.32**	30	3805796652	8848.04	30	301	2.35
san200_0.7_2	200	70.00%	≥18	36793469795	≥21600	**18**	419	1.07	≥18	5332111203	≥21600	**18**	394	**0.99**
san200_0.9_1	200	90.00%	≥48	44733138057	≥21600	**70**	32148	171.54	≥48	16789724889	≥21600	**70**	20239	**93.92**
san200_0.9_2	200	90.00%	≥41	47851554296	≥21600	**60**	446005	4096.28	≥41	3788397406	≥21600	**60**	309378	**2896.35**
san200_0.9_3	200	90.00%	≥44	22287415000	≥21600	**44**	38240	347.31	≥44	1804345383	≥21600	**44**	32327	**292.44**
san400_0.5_1	400	50.00%	13	1414713059	924.5	13	703	**8.9**	13	155275559	2095.96	13	882	10.11
san400_0.7_1	400	70.00%	≥22	86272548474	≥21600	**40**	12553	672.43	≥22	17971994383	≥21600	**40**	11830	**638.98**
san400_0.7_2	400	70.00%	≥17	60943311431	≥21600	**30**	21511	**218.47**	≥17	8165125369	≥21600	**30**	26818	239.58
san400_0.7_3	400	70.00%	≥22	25369766347	≥21600	**22**	141744	**833.87**	≥18	2116314580	≥21600	**22**	213162	925.61
san400_0.9_1	400	90.00%	≥49	51823450631	≥21600	**100**	610230	17850.1	≥49	6422756477	≥21600	**100**	291195	**10829.3**
sanr200_0.7	200	69.69%	18	7985123	**5.96**	18	28835	51.19	18	665013	11.55	18	25582	37.68
sanr200_0.9	200	89.76%	≥40	27782611608	≥21600	≥**41**	2634239	≥21600	≥40	2453566354	≥21600	≥**41**	2649347	≥21600
sanr400_0.5	400	50.11%	13	4283999	**4.07**	13	36414	53.04	13	251463	11.74	13	32883	34.64
sanr400_0.7	400	70.01%	21	5624332491	**4700.66**	21	11010391	31686.3	21	408188818	8765.3	21	9759158	23887.5

Fig. 2. Results on the Dimacs Instances. We compare the number of choice points and running time in seconds of our implementation of dfmax, our algorithm using coloring bounds only (χ), our algorithm using domain filtering only (DF), and a forth using both ($\chi+DF$).

different orderings. For mann_a81 e.g. we find a solution of 1096 within 10 sec. when using a reverse ordering, whereas the best solutions found by the other approaches after 6 hours is still below 1000 (and below 500 when not using coloring bounds). It is not clear though how to detect these effects while searching, and utilize them to improve convergence.

References

1. A. Atamtürk and G.L. Nemhauser and M.W.P. Savelsberg. Conflict Graphs in Integer Programming. *European Journal of Operations Research*, 121:40–55, 2000.
2. D. Brelaz. New methods to color the vertices of a graph. *Communcations of the ACM*, 22:251–256, 1979.
3. M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman & Co., 1979.
4. E. Balas, S. Ceria, G. Couruéjols and G. Pataki. Polyhedral Methods for the Maximum Clique Problem. in [15, p. 11–28].
5. E. Balas and C.S. Yu. Finding a Maximum Clique in an Arbitrary Graph. *SIAM Journal Computing*, 14(4):1054–1068, 1986.
6. E. Balas and Xue. Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring. *Algorithmica*, 15:397–412, 1996.
7. I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo. *The Maximum Clique Problem*. Handbook of Combinatorial Optimization, volume 4. Kluwer Academic Publishers, 1999.
8. A. Caprara and D. Pisinger and P. Toth. Exact Solutions on the Quadratic Knapsack Problem. *Informs Journal on Computing*, 11(2):125–137, 1999.
9. R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters* 9:375–382, 1990.
10. T. Fahle. Cost Based Filtering vs. UpperBounds for Maximum Clique *CP-AI-OR'02 Workshop*, Le Croisic/France, 2002.
11. T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics* 8(1):59–81, 2002.
12. T. Fahle and M. Sellmann. Constraint Programming Based Column Generation with Knapsack Subproblems. *Annals of Operations Reserach, Vol 114*, 2003, to appear.
13. F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Proc. CP'99* LNCS 1713:189–203, 1999.
14. F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proceedings of CP'00*, Springer LNCF 1894:187–200, 2000.
15. D.S. Johnson and M.A. Trick. *Cliques, Colorings and Satisfiability*. 2nd DIMACS Implementation Challenge, 1993. American Mathematical Society, 1996.
16. U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint programming based column generation. *Proc. CP'99* LNCS 1713:261–274, 1999.
17. P.R.J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, to appear.
18. G. Ottosson and E.S. Thorsteinsson. Linear Relaxation and Reduced-Cost Based Propagation of Continuous Variable Subscripts. *CP-AI-OR'00*, Paderborn, 2000, submitted.
19. M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
20. D.R. Wood. An algorithm for finding a maximum clique in a graph. *Operations Research Letters*, 21:211–217, 1997.
21. ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/ Dimacs Clique Benchmark Instances.
22. `dfmax.c`. ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/