Alcom-FT Technical Report Series
ALCOMFT-TR-02-106

Proceedings CPAIOR'02

Cost Based Filtering vs. Upper Bounds for Maximum Clique*

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

Abstract

In this paper we consider a branch-and-bound algorithm for the maximum clique problem. 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). Doing this, we can reduce the number of choice points visited by a typical factor of 10 – 50.

Additionally, we present a taxonomy of upper bounds used in the OR community 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.

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

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 largest subgraph $C = (V_C, E_C)$ in G that forms a clique. It is known for long time that MC is NP-hard (see [2]). In this paper we consider the unweighted case, where the size of the maximum clique is the cardinality of V_C (the techniques can straightforwardly be adapted to the weighted case).

Despite its complexity, the maximum clique problem is important in many fields. Cliques are used in integer programming for presolving [17] and deriving valid cuts [1]. Sellmann and Harwey used them in a CP framework for the golfer problem as approvers on inconsistent assignments [19]. They appear in coding theory, fault diagnostics, or pattern recognition. For an overview on applications we refer to [5].

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

In this paper we introduce some simple cost based filtering techniques [10]. We test them on an enumeration algorithm proposed by Carraghan and Pardalos in 1990 [7]. 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 optimization constraints tighten the candidate set by removing or fixing certain nodes. We show analytically that domain filtering in a sense is as tight as typical upper bounds for MC. Preliminary tests on the DIMACS benchmark sets [12, 23] show that we can reduce the number of choice points visited by a typical factor of 10 – 50. Further improvement is gained by a dynamic reordering of variables in each choice point.

1.1 Literature review

Cost based filtering [10] aims at fixing variables with respect to the objective function. It typically combines bounds from the OR world with domain filtering stemming from CP. It is an important technique when solving complex optimization problems. Optimization constraints for traveling salesman, knapsack, airline scheduling, multimedia applications and many more problems have been addressed e.g. in [10, 11, 13, 8, 18, 9, 15].

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 overviews in [5] for further details. Some results refer to close relatives of the maximum cliques problem, namely the *vertex cover (VC)* and *independent set (IS)* problem. They stem from simple transformations: $V_C \subseteq V$ is a MC of $G = (V,E) \Leftrightarrow C$ is a maximum IS in $\overline{G} = (V, (V \times V) \setminus E) \Leftrightarrow V \setminus V_C$ is a minimum VC of $\overline{G}$.

Tarjan and Trojanowski [20] gave a recursive algorithm for the vertex cover problem that runs in time $O(2^{n/3}) \approx O(1.26^n)$ showing that NP-hard optimization problems can indeed be solved faster than by pure enumeration of the solution vectors. This result was later improved and the asymptotically fastest algorithm is devoted to Robson [16] having time complexity $O(2^{0.276n})$.

Balas and Yu [4] 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.

Cuts for an integer programming formulations of the maximum clique problem have been investigated in [3]. The linear independent set formulation, $\max\{\sum_{i=1}^{n} x_i \mid x_i + x_j \leq 1, \forall \{i,j\} \notin E,\ x \in [0\dots1]^n\}$ 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 [6].

Enumerating algorithms have been contributed by many researchers. Of certain interests for this work are the methods of Carragham and Paradalos, and Östergård. We will return to those algorithms in Sec. 2. Before that we will briefly introduce some notation. In Sec. 2.3 we will present the new constraints. Some analytical results are given in Sec. 3, whereas empirical experiments 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 $\Gamma(v) = \{u \in V | \{u,v\} \in E\}$ and the degree of a node $v \in V$ as $\delta(v) = |\Gamma(v)|$. Given a node set U, the graph $G_U = (U, E \cap (U \times U))$ is the graph *induced* by U. We write $\Gamma_U(v) = \{u \in U | \{u,v\} \in (U \times U) \cap E\}$ and the $\delta_U(v) = |\Gamma_U(v)|$ 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 $P = \bigcap_{v \in C} \Gamma(v)$ the *candidate set of* C. A clique $\tilde{C}$ is an *extension* of a clique C if $C \subseteq \tilde{C}$. We use the notation $\tilde{C}(C)$ 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 Enumeration Algorithms for Maximum Clique

2.1 The Carraghan/Pardalos Algorithm

In the following we present the basic idea of the enumeration algorithm presented in [7]. The famous `dfmax` [24] program is an efficient implementation of that algorithm.

Carrighan and Pardalos 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 $P = \bigcap_{v \in C} \Gamma(v)$ containing all nodes that are adjacent to the clique C currently under construction is used for bounding. A simple DFS and static variable ordering is used to guide the search. Algorithm 1 presents that idea.

Algorithm 1 A simple Algorithm for the Maximum Clique Problem

Require: $G = (V,E)$ undirected, unweighted graph
Ensure: $C^* \subseteq V$ node set forming a largest clique in G.

function main( )

$C^* \leftarrow \emptyset$
reorder(V)
findClique($\emptyset$, V)
return C^*

function findClique(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**
    $P \leftarrow P \setminus \{p\}$
    $C' \leftarrow C \cup \{p\}$
    $P' \leftarrow P \cap \Gamma(p)$
    findClique(C', P')

2.2 The Improvement of Östergård

Östergård [14] 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. Though not being mentioned by Östergård his approach follows the Russian Doll Search idea of Verfaillie et al. [21].

2.3 Observations on the Candidate Set

Both algorithms sketched above share the idea of a candidate set $P = \bigcap_{v \in C} \Gamma(v)$ 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. $P = \bigcap_{v \in C} \Gamma(v)$. 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: Assume we have a node $v \in P$ such that $|C| + \delta_P(v) < \sigma - 1$, but v extends C to a maximum clique $C^*(C)$ with $|C^*(C)| = k$ nodes. Then for all $u \in C^*(C)$ we have: $\delta_{C^*}(u) = k-1$. Since σ is a lower bound on $|C^*(C)|$ it holds $\sigma - 1 \leq k - 1 = \delta_{C^*(C)}(v)$. The degrees in $C^*(C)$ can be partitioned into $\delta_{C^*(C)}(v) = \delta_C(v) + \delta_{C^* \setminus C}(v)$. Hence we obtain

$$\sigma - 1 \leq k - 1 = \delta_{C^*(C)}(v) \quad = \quad \delta_C(v) + \delta_{C^* \setminus C}(v) \quad \leq \quad |C| - 1 + \delta_P(v) \quad < \sigma - 1$$

which is a contradiction. □

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. $P = \bigcap_{v \in C} \Gamma(v)$. 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: Let $C^*(C)$ be a maximum clique and $C \subseteq C^*$. Then $C^* \subseteq C \cup P$ (by definition of P). Assume, $v \in P$ such that $\delta_P(v) = |P| - 1$, but $v \notin C^*$. By construction, v is adjacent to all nodes in C (because it is a member of the candidate set) and all nodes in P (because of its degree in P). Hence, v is also adjacent to any node u in $C^* \subseteq C \cup P$ which means $C^* \cup \{v\}$ is a clique containing C, but being larger than C^*. This is a contradiction to the choice of C^* as a maximum clique. □

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

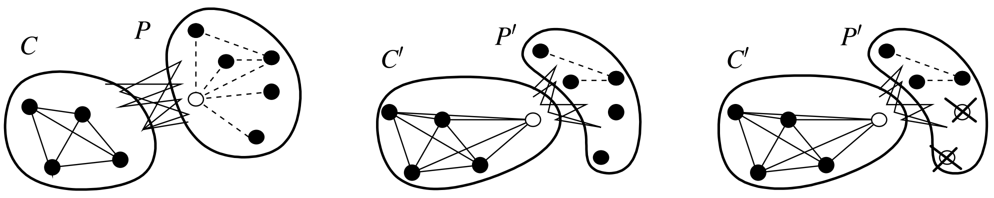


Figure 1: (a) a clique C and its candidate set P. The approaches of Carrighan/Pardalos and Östergård basically neglect the structure within P. (b) applying lemma 2 fixes one more node. (c) as a result, two nodes have degree zero now, and can be eliminated according to lemma 1.

2.4 An Improved Enumeration Algorithm

With these observations at hand we are ready to present an improved enumeration algorithm for MC (Alg. 2 on page). Between lines 7 and 8 of the first algorithm we insert some domain reduction 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.

The propagation 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. Hence, no special care is needed.

The algorithms presented in Sec. 2.1, 2.2 only use a static ordering of variables (i.e. nodes). We introduce some dynamic reordering in the algorithm in line 18, just before the next recursion is called. In Sec. 4 we will present results on both, the static and the dynamic ordering variant.

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.

3.1 Accuracy

We still assume the setting presented in Sec. 2: C is the clique currently under construction, $P = \bigcap_{v \in C} \Gamma(v)$ the corresponding candidate set. The following lemma lists some well-known upper bounds $\mathcal{U}_i(C,P)$ for a maximal clique $C^*(C)$ (some more, especially eigenvalue bounds, are listed in [5]):

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

1. *Only nodes from the candidate set can extend C:* $\mathcal{U}_1(C,P) = |C| + |P|$.

Algorithm 2 An Improved Algorithm for the Maximum Clique Problem

Require: $G = (V, E)$ undirected, unweighted graph
Ensure: $C^* \subseteq V$ node set forming a largest clique in G.

function main( )

$C^* \leftarrow \emptyset$
reorder(V)
findClique2($\emptyset$, V)
return C^*

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**
    $P \leftarrow P \setminus \{p\}$
    $C' \leftarrow C \cup \{p\}$
    $P' \leftarrow P \cap \Gamma(p)$
    // domain reduction
    repeat
      // 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*
    until $(\forall\, v \in P' : (|C^*| - |C'|) \le \delta_{P'}(v) < (|P'| - 1)$
    reorder (P')
    findClique2(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:*
$$\mathcal{U}_7(C,P) = |C| + \max_{I \in \mathcal{Z}(G_P)} \max\{k \in \mathbb{N} \mid \tfrac{k(k-1)}{2} \le |E_I|\},$$

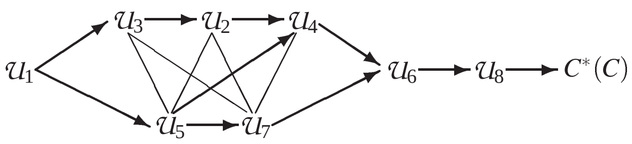
where E_I is the edge set of the graph $G_I = (P, E_I)$ induced by I.

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 [7] for $\mathcal{U}_1$, [4, 22] for $\mathcal{U}_8$, and [5] 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 $P = \bigcap_{v \in C} \Gamma(v)$ 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:

$\mathcal{U}_1(C,P) \geq \mathcal{U}_3(C,P)$**:** For any connected component G_I of P: $|P| \geq |I|$.

$\mathcal{U}_3(C,P) \geq \mathcal{U}_2(C,P)$**:** Let $k := \max\{\delta_P(v) + 1 \mid v \in P\}$. Then there exists a connected component with at least k nodes $\Rightarrow$ $|I^{max}| \geq k$.

$\mathcal{U}_2(C,P) \geq \mathcal{U}_4(C,P)$**:** Let $k := \max\{k \mid \exists\, v_1 < \cdots < v_k \in P,\ \delta_P(v_i) \geq k-1\}$
$\Rightarrow \max\{\delta_P(v) + 1 \mid v \in P\} \geq (k-1) + 1 = k$.

$\mathcal{U}_4(C,P) \geq \mathcal{U}_6(C,P)$**:** $\max\{k \mid \exists\, v_1 < \cdots < v_k \in P,\ \delta_P(v_i) \geq k-1\}$
$\geq \max\limits_{I \in \mathcal{Z}(G_P)} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_P(v_i) \geq k-1\}$

$\mathcal{U}_6(C,P) \geq \mathcal{U}_8(C,P)$**:** Let $k' := \max\limits_{I \in \mathcal{Z}(G_P)} \{\max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_P(v_i) \geq k-1\}\}$. Assume, $\chi(G_P) = k'' > k'$. As different connected components can be colored independently, there exists an $I \in \mathcal{Z}(G_P)$, such that $\chi(G_I) = \chi(G_P) = k''$. Then G_I contains $k'' > k'$ nodes of degree $k'' - 1 > k' - 1$, hence k' is not maximal — a contradiction.

$\mathcal{U}_8(C,P) \geq C^*(C)$**:** Any graph containing a k-clique requires at least k colors for vertex coloring.

$\mathcal{U}_1(C,P) \geq \mathcal{U}_5(C,P)$**:** Let $G_P = (P, E_P) \Rightarrow |E_P| \leq \frac{|P|(|P|-1)}{2} \Rightarrow \max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \leq |E_P|\}$
$\leq \max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \leq \frac{|P|(|P|-1)}{2}\} = |P|$.

$\mathcal{U}_5(C,P) \geq \mathcal{U}_7(C,P)$**:** $\max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \leq |E_P|\} \geq \max\limits_{I \in \mathcal{Z}(G_P)} \max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \leq |E_I|\}$

$\mathcal{U}_5(C,P) \geq \mathcal{U}_4(C,P)$**:** Let $k := \max\{k \mid \exists\, v_1 < \cdots < v_k \in P,\ \delta_P(v_i) \geq k-1\} \Rightarrow |E_P| \geq \frac{k(k-1)}{2}$
$\Rightarrow \max\{l \in \mathbb{N} \mid \frac{l(l-1)}{2} \leq |E_P|\} \geq k$.

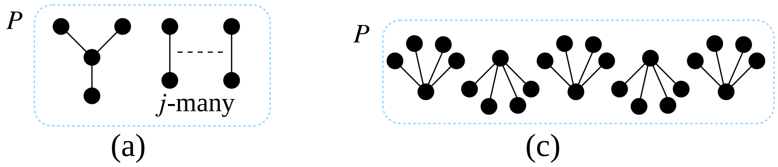


Figure 2:

$\mathcal{U}_7(C,P) \geq \mathcal{U}_6(C,P)$**:** Let I be the node set, where the maximum of bound $\mathcal{U}_6(C,P) =: k$ is found. On a connected component, $\mathcal{U}_6(C,P)$ and $\mathcal{U}_4(C,P)$, as well as $\mathcal{U}_7(C,P)$ and $\mathcal{U}_5(C,P)$ are the same. As proven above $\mathcal{U}_5(C,P) \geq \mathcal{U}_4(C,P)$. Therefore, $\mathcal{U}_7(C,P) \geq \mathcal{U}_6(C,P)$ on that component and (because k is optimal for P) also on P.

$\mathcal{U}_2(C,P) \lesseqgtr \mathcal{U}_5(C,P)$ and $\mathcal{U}_3(C,P) \lesseqgtr \mathcal{U}_5(C,P)$: Consider the candidate set of Fig. 2(a). There, $\mathcal{U}_2(C,P) = |C|+4$ and $\mathcal{U}_3(C,P) = |C|+4$ because of the "star" in P. We have $|E_P| = 3+j$, so we get $\mathcal{U}_5(C,P) = |C| + \max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \leq 3+j\}$. If $j=1$ we have $\mathcal{U}_5(C,P) = |C|+3 < \mathcal{U}_2(C,P), \mathcal{U}_3(C,P)$. If $j=7$ we have $\mathcal{U}_5(C,P) = |C|+5 > \mathcal{U}_2(C,P), \mathcal{U}_3(C,P)$. Hence, there is no general dominance between $\mathcal{U}_5(C,P)$ and $\mathcal{U}_2(C,P), \mathcal{U}_3(C,P)$.

$\mathcal{U}_4(C,P) \lesseqgtr \mathcal{U}_7(C,P)$**:** Consider the candidate set of Fig. 2(b). There, $\mathcal{U}_4(C,P) = |C|+5$ and $\mathcal{U}_7(C,P) = |C| + \max\{k \in \mathbb{N} \mid \frac{k(k-1)}{2} \leq 4\} = |C|+3$. In this case, $\mathcal{U}_4(C,P) \geq \mathcal{U}_7(C,P)$. Now, consider P as being a single connected component only. Then $\mathcal{U}_7(C,P) \equiv \mathcal{U}_5(C,P)$ and as proven above: $\mathcal{U}_5(C,P) \geq \mathcal{U}_4(C,P)$. Again, we cannot show dominance in general on two bounds.

$\mathcal{U}_2(C,P) \lesseqgtr \mathcal{U}_7(C,P)$ and $\mathcal{U}_3(C,P) \lesseqgtr \mathcal{U}_7(C,P)$: In the single connected component case it holds $\mathcal{U}_7(C,P) \equiv \mathcal{U}_5(C,P)$. But as proven earlier: $\mathcal{U}_2(C,P) \lesseqgtr \mathcal{U}_5(C,P)$ and $\mathcal{U}_3(C,P) \lesseqgtr \mathcal{U}_5(C,P)$, hence $\mathcal{U}_2(C,P) \lesseqgtr \mathcal{U}_7(C,P)$.

Next, we show that the filters described in Sec. 2.3 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. $P = \bigcap_{v\in C} \Gamma(v)$. 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 reduction of algorithm 2, lines 8 – 17. 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 – 17 of algorithm 2 we have: $\forall\, v \in P' : (\sigma - |C'|) \leq \delta_{P'}(v)$.

$$\Rightarrow\ \forall\, v \in P' :\ |C'| + \delta_{P'}(v) \geq \sigma \geq |C'| + \max_{I\in\mathcal{Z}(G_{P'})} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_{P'}(v_i) \geq k-1\}$$

$$\Rightarrow\ \forall\, v \in P' :\ \delta_{P'}(v) \geq \underbrace{\max_{I\in\mathcal{Z}(G_{P'})} \max\{k \mid \exists\, v_1 < \cdots < v_k \in I,\ \delta_{P'}(v_i) \geq k-1\}}_{=:\alpha}$$

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

Bound $\mathcal{U}_8$ is not included in the above theorem. There are indicators that for some situations domain filtering dominates $\mathcal{U}_8$ (in the sense of Theorem 1), whereas in other situations, $\mathcal{U}_8$ is tighter than filtering alone. The exact relation between domain filtering and coloring bounds is subject of further investigation.

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, approximation algorithms 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. [4, 22]).

Our domain filtering, as described in algorithm 2, lines 8 – 17, 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 again time $O(|P|)$ leading to a total running time of $O(|P|^2)$.

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 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 compared to our implementation[1] of `dfmax`. We use the well-known DIMACS benchmark set [12, 23] for the comparison.

It should be noted that we regard these experiments as preliminary since

- the current implementation is not as tuned as the opponent.
- tests have been performed using a node ordering that is known to be well suited for `dfmax` (see [7, p. 375]). It is not clear whether this is also a good starting point for the new approach.

[1] For a fare comparison, we decided to use identical subroutines for both enumeration approaches. Hence, we adapted `dfmax` to our setting. This results in a slightly faster algorithm ($< 2\%$) than the original `dfmax`, since we use a simpler graph representation for our approaches.

4.1 The Implementation

An efficient implementation has to consider some more "tricks" than those described in the algorithms 1 and 2. When determine $P' = P \cap \Gamma(v)$, we use a loop running over all elements in $p \in P$. 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 [24] for more details).

For the second algorithm 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 – 17 of algorithm 2 we perform domain filtering. We use a stack to store those nodes, that either have not been copied to P' (lines 5 – 7), or have been removed during domain reduction (lines 8 – 17). 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 algorithm using re-ordering we initially sort the nodes according to decreasing degrees. Re-ordering in a node is done by a standard quicksort algorithm, or — if $|P'| < 100$ — by an insertion sort. The latter one is expected to take advantage from having only few elements changing their relative position after updating degrees. Thus, by switching the algorithms, we typically gain some running time. As sorting is relatively expensive, more tests on that switching parameter should help saving running time.

4.2 Numerical Results

Both algorithms were coded in C++ and compiled by the GNU g++ 2.95.3 compiler using full optimization (`-O3`). Our benchmark tests run on a Pentium III-933 PC with 512 MB ram operating Linux kernel 2.4.17. We stopped each run after 6 hours (21 600 sec).

The benchmark set stems from the DIMACS challenge [12, 23]. It consists of 66 instances, ranging from 28 – 3361 nodes, and 420 – 11 million edges. The density $(|V|^2 - |V|)/|E|$ of the underlying graphs range from 3.5% – 99.8%. More details, e.g. minimal and maximal degree ($\Delta_{\min}$, $\Delta_{\max}$) are given in the tables below. To our knowledge for some of these instances optimal solutions have not been proven so far.

In the next tables we compare our `dfmax` variant using only bound $\mathcal{U}_1$ (similar to Alg. 1), an implementation using that bound plus domain filtering, but no re-ordering, and the same algorithm that re-orders nodes in every choice point. The latter two correspond to Alg. 2.

4.2.1 Instances Solved within Timelimit

Within 6 hours, 32 instances can be solved by all three approaches. For two additional instances only two approaches proved optimality. Interestingly, 20 of them, needed less than 10 seconds, 4 more can be solved within 30 seconds, and for 27 less than 30 minutes were sufficient. We give the instance, the clique size (C^*), the running time (time) and number of choice points (ChPts) in Table 3.

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 new approach. Improvements grow up to a factor of 51.7 (hamming6-2) when using

Instance							dfmax-like		filtering		filtering + reordering	
file	$\lvert V\rvert$	$\lvert E\rvert$	**density**	Δ_{min}	Δ_{max}	$\lvert C^*\rvert$	**ChPts**	**time**	**ChPts**	**time**	**ChPts**	**time**
brock200_1	200	29668	74.54%	130	165	21	38043497	28.82	3350353	50.61	2313467	35.69
brock200_2	200	19752	49.63%	78	114	12	54314	0.05	3539	0.16	3518	0.16
brock200_3	200	24096	60.54%	99	134	15	453265	0.41	32170	0.99	31515	0.88
brock200_4	200	26178	65.77%	112	147	17	2161909	1.73	167750	3.64	132692	3.11
brock400_4	400	119530	74.89%	275	326	33	20152485865	20577.70	1016877999	$\geq$21600.00[1]	859347507	21489.20
c-fat200-1	200	3068	7.71%	14	17	12	52	0.01	5	0.00	5	0.01
c-fat200-2	200	6470	16.26%	32	34	24	444	0.00	5	0.00	5	0.00
c-fat200-5	200	16946	42.58%	83	86	58	268435599	1307.45	5	0.00	5	0.00
c-fat500-1	500	8918	3.57%	17	20	14	71	0.01	5	0.01	5	0.01
c-fat500-10	500	93254	37.38%	185	188	126	15086121946	$\geq$21600.00[2]	5	0.01	5	0.02
c-fat500-2	500	18278	7.33%	35	38	26	683	0.01	5	0.01	5	0.02
c-fat500-5	500	46382	18.59%	92	95	64	5703074	5.71	5	0.01	5	0.03
hamming6-2	64	3648	90.48%	57	57	32	42787	0.03	1891	0.01	827	0.01
hamming6-4	64	1408	34.92%	22	22	4	221	0.00	47	0.00	36	0.00
hamming8-4	256	41728	63.92%	163	163	16	3742143	4.05	252418	7.47	325951	6.51
johnson16-2-4	120	10920	76.47%	91	91	8	4177630	1.51	904446	3.57	1176074	4.48
johnson8-2-4	28	420	55.56%	15	15	4	90	0.00	21	0.00	18	0.00
johnson8-4-4	70	3710	76.81%	53	53	14	12544	0.01	1205	0.02	1913	0.02
keller4	171	18870	64.91%	102	124	11	1275236	0.79	107086	1.53	111284	1.33
mann_a9	45	1836	92.73%	40	41	16	329235	0.10	92210	0.11	214424	0.25
p_hat1000-1	1000	244506	24.48%	105	408	10	1801019	2.18	81573	9.90	76995	9.81
p_hat1500-1	1500	569846	25.34%	157	614	12	13825580	20.77	454914	97.29	303225	96.93
p_hat300-1	300	21866	24.38%	23	132	8	11634	0.02	450	0.06	406	0.05
p_hat300-2	300	43856	48.89%	59	229	25	1553140	1.25	182168	2.52	161080	2.68
p_hat300-3	300	66780	74.45%	168	267	36	1960518988	1614.35	251846120	2561.11	189101562	1941.54
p_hat500-1	500	63138	25.31%	52	204	9	89908	0.11	4978	0.43	4954	0.46
p_hat500-2	500	125892	50.46%	117	389	36	292274682	253.76	35324032	459.89	30222902	401.23
p_hat700-1	700	121998	24.93%	75	286	11	343857	0.44	20630	1.94	22538	1.90
p_hat700-2	700	243456	49.76%	157	539	44	10969888234	10397.80	1394776054	16637.80	802206146	10663.40
san200_0.7_1	200	27860	70.00%	125	155	30	14265131069	5713.23	3805796652	8848.04	3005157489	11437.20
san400_0.5_1	400	79800	50.00%	174	225	13	1414713059	924.50	155275559	2095.96	87018003	3000.65
sanr200_0.7	200	27736	69.69%	120	161	18	7985123	5.96	665013	11.55	549887	9.03
sanr400_0.5	400	79968	50.11%	161	233	13	4283999	4.07	251463	11.74	263245	10.26
sanr400_0.7	400	111738	70.01%	252	310	21	5624332491	4700.66	408188818	8765.30	281397941	5708.89

[1] filtering on brock400_4: $\lvert C^*\rvert \geq 25$ [2] `dfmax` on c-fat500-10: $\lvert C^*\rvert \geq 124$

Figure 3: 34 instances can be solved within 6 hours by all three approaches.

re-orderings in every node. In contrast, running time of the current code is typically a factor of 2 times slower than the `dfmax` code. Though using some more time per choice point, re-ordering helps to reduce running time, unfortunately still staying mostly above the running times of `dfmax`.

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 [22] 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.

4.2.2 Instances Not Solved within Timelimit

For about half of the benchmark sets an optimal solution cannot be proved within the timelimit by any of the approaches considered. In table 4 we present the best solution found ($|C^*|$) after reaching the time limit, as well as the number of choice points (ChPts) used up to that point.

The hamming*xxx* benchmark instances seem not to profit from re-ordering nodes during search. In those three cases, the results obtained using re-ordering are far away from the results found using static ordering. On the contrary, san400_0.7_2 shows that filtering plus reordering can helps significantly: Thought using 20 times less choice points than `dvmax`, the best result found is 30, whereas `dfmax` only finds 17 in the same time.

Also, it turns out that after 6 hours for 24 instances best solutions are found by `dfmax`, 20 best solutions are found by filtering plus re-ordering, and 17 best solutions found by filtering only. This seems to indicate that `dfmax` is the better choice for MC. However, some care is needed here. Without giving the concrete results we only mention that after 2 hours the picture was as follows: 23 best solutions found by filtering plus reordering, 21 found by `dfmax`, and 20 by pure filtering. In 22 cases the best algorithms for an instance in the 2-hour snapshots was not longer the best approach after 6 hours. Given the discrete nature of the problem, the next snapshots after perhaps 6 hours and 10 minutes might give another, different impression.

Running some samples for 24 hours was not substantially changing the situation. Therefore, a serious conclusion on the impact of filtering on large instances and hence large search spaces has to remain open.

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 OR 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 OR. In a preliminary experimental study we showed significant reduction of choice points when using the new approach. A simple node reordering often helped to further reduce choice points and running time. However, the current code is not able to beat its opponent concerning running time so far.

Future effort will be spend to a more detailed complexity analysis, as well as a study on further domain reduction (maybe using coloring bound iedas). Another issue is node re-ordering: We found in an additional study that some instances benefit from different orderings. For

Instance						dfmax-like		filtering		filtering + reordering	
file	$\lvert V \rvert$	$\lvert E \rvert$	**density**	Δ_{min}	Δ_{max}	$\lvert C^{*} \rvert$	**ChPts**	$\lvert C^{*} \rvert$	**ChPts**	$\lvert C^{*} \rvert$	**ChPts**
brock400_1	400	119446	74.84%	272	320	$\geq$27	24423311512	$\geq$25	1115779306	$\geq$27	1151101803
brock400_2	400	119572	74.92%	274	328	$\geq$29	19952147706	$\geq$29	809348442	$\geq$29	800635654
brock400_3	400	119362	74.79%	275	322	$\geq$31	25092151368	$\geq$24	1142054135	$\geq$25	1233611595
brock800_1	800	415010	64.93%	477	560	$\geq$21	21193921452	$\geq$21	579955608	$\geq$21	694366310
brock800_2	800	416332	65.13%	472	566	$\geq$21	23396947454	$\geq$20	684885264	$\geq$20	895769632
brock800_3	800	414666	64.87%	474	558	$\geq$21	22077418714	$\geq$20	645967913	$\geq$21	764468066
brock800_4	800	415286	64.97%	481	565	$\geq$26	21514428210	$\geq$21	643687866	$\geq$21	767753797
hamming10-2	1024	1037312	99.02%	1013	1013	$\geq$512	2963756831	$\geq$512	902493400	$\geq$177	3255678941
hamming10-4	1024	868352	82.89%	848	848	$\geq$32	35273440082	$\geq$32	2933556637	$\geq$28	2006373042
hamming8-2	256	63232	96.86%	247	247	$\geq$128	13603244188	$\geq$128	1582399395	$\geq$107	668277943
johnson32-2-4	496	215760	87.88%	435	435	$\geq$16	49631406919	$\geq$16	4918222342	$\geq$16	5395390834
keller5	776	451980	75.15%	560	638	$\geq$24	23430537443	$\geq$24	737283168	$\geq$26	1719992940
keller6	3361	9239796	81.82%	2690	2952	$\geq$43	17888655376	$\geq$43	540201863	$\geq$49	1353081997
mann_a27	378	141102	99.01%	364	374	$\geq$116	15557799773	$\geq$116	23120760471	$\geq$114	16934063843
mann_a45	1035	1066230	99.63%	1012	1031	$\geq$220	12119272496	$\geq$234	23274505520	$\geq$220	24148632482
mann_a81	3321	11012760	99.88%	3280	3317	$\geq$438	5947703016	$\geq$467	23035700588	$\geq$438	16758223162
p_hat1000-2	1000	489598	49.01%	232	766	$\geq$42	19361957667	$\geq$41	1149227032	$\geq$41	1109383311
p_hat1000-3	1000	743492	74.42%	582	895	$\geq$49	21936773057	$\geq$47	1603605303	$\geq$47	1573103509
p_hat1500-2	1500	1137920	50.61%	335	1153	$\geq$46	24069645860	$\geq$46	1893106666	$\geq$46	1926905777
p_hat1500-3	1500	1694488	75.36%	912	1330	$\geq$53	26020312163	$\geq$53	1920941837	$\geq$53	1870837811
p_hat500-3	500	187600	75.19%	293	452	$\geq$44	23974852012	$\geq$43	2103038234	$\geq$44	1941804035
p_hat700-3	700	366020	74.80%	408	627	$\geq$49	22686699084	$\geq$49	1851669803	$\geq$49	1649739842
san1000	1000	501000	50.15%	445	550	$\geq$10	11298828641	$\geq$10	207288356	$\geq$10	30860052
san200_0.7_2	200	27860	70.00%	103	164	$\geq$18	36793469795	$\geq$18	5332111203	$\geq$18	3963265743
san200_0.9_1	200	35820	90.00%	159	191	$\geq$48	44733138057	$\geq$48	16789724889	$\geq$48	15249578090
san200_0.9_2	200	35820	90.00%	169	188	$\geq$41	47851554296	$\geq$41	3788397406	$\geq$50	9829628309
san200_0.9_3	200	35820	90.00%	166	187	$\geq$44	22287415000	$\geq$44	1804345383	$\geq$44	3986070188
san400_0.7_1	400	111720	70.00%	257	301	$\geq$22	86272548474	$\geq$22	17971994383	$\geq$22	23367299932
san400_0.7_2	400	111720	70.00%	257	304	$\geq$17	60943311431	$\geq$17	8165125369	$\geq$30	2985934092
san400_0.7_3	400	111720	70.00%	250	307	$\geq$22	25369766347	$\geq$18	2116314580	$\geq$17	2450986143
san400_0.9_1	400	143640	90.00%	341	374	$\geq$49	51823450631	$\geq$49	6422756477	$\geq$51	15552322617
sanr200_0.9	200	35726	89.76%	166	189	$\geq$40	27782611608	$\geq$40	2453566354	$\geq$41	2466068618

Figure 4: 32 instances of the DIMACS benchmark set cannot be solved within 6 hours

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 470. It is not clear though how to detect those effects while searching, and utilize them to improve convergence.

As indicated by one of the reviewers, studies on how to apply these filtering techniques to similar problems like coloring is interesting as well.

Acknowledgment We thank two anonymous reviewers for their helpful suggestions.

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] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, 1979.

[3] E. Balas, S. Ceria, G. Couruéjols and G. Pataki. Polyhedral Methods for the Maximum Clique Problem. in [12, p. 11–28].

[4] E. Balas and C.S. Yu. Finding a Maximum Clique in an Arbitrary Graph. *SIAM Journal Computing*, 14(4):1054–1068, 1986.

[5] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo. *The Maximum Clique Problem*. Handbook of Combinatorial Optimization, volume 4. Kluwer Academic Publishers, 1999.

[6] A. Caprara and D. Pisinger and P. Toth. Exact Solutions on the Quadratic Knapsack Problem. *Informs Journal on Computing*, 11(2):125–137, 1999.

[7] R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters* 9:375–382, 1990.

[8] 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.

[9] T. Fahle and M. Sellmann. Constraint Programming Based Column Generation with Knapsack Subproblems. *Proc. of the CP-AI-OR'00*, Paderborn, 2000.

[10] F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Proceedings of the CP'99* Springer LNCS 1713:189–203, 1999.

[11] F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proceedings of the CP-AI-OR'00*, Paderborn, 2000.

[12] D.S. Johnson and M.A. Trick. *Cliques, Colorings and Satisfiability*. 2nd DIMACS Implementation Challenge, 1993. American Mathematical Society, 1996.

[13] U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint programming based column generation. *Proceedings of the CP'99* Springer LNCS 1713:261–274, 1999.

[14] P.R.J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, to appear.

[15] G. Ottosson and E.S. Thorsteinsson. Linear Relaxation and Reduced-Cost Based Propagation of Continuous Variable Subscripts. *Proceedings of the CP-AI-OR'00*, Paderborn, 2000.

[16] J.M. Robson. Algorithms for Maximum Independent Sets. *Journal of Algorithms*, 7: 425-440, 1986.

[17] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[18] M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for a Multimedia Application *Proceedings CP-AI-OR'01*, Ashford/UK, April 2001

[19] M. Sellmann and W. Harvey. Heuristic Constraint Propagation (Using Local Search for Incomplete Pruning and Domain Filtering of Redundant Constraints for the Social Golfer Problem) *Proceedings CP-AI-OR'02*, Le Croisic/France, 2002.

[20] R.E. Tarjan and A.E. Trojanowski. Finding a Maximum Independent Set. *SIAM Journal on Computing*, 6:537–546, 1977.

[21] G. Verfaillie, M. Lemaître, T. Schiex. Russian doll search for solving constraint satisfaction problems. *Proceedings of AAAI-96*, 1:181–187, 1996.

[22] D.R. Wood. An algorithm for finding a maximum clique in a graph. *Operations Research Letters*, 21:211–217, 1997.

[23] Dimacs Clique Benchmark Instances. `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/`

[24] `dfmax.c`. `ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/`