Exact Coloring of Real-Life Graphs is Easy

Olivier Coudert
Synopsys, Inc., 700 East Middlefield Rd.
Mountain View, CA 94043

Abstract

Graph coloring has several important applications in VLSI CAD. Since graph coloring is NP-complete, heuristics are used to approximate the optimum solution. But heuristic solutions are typically 10% off, and as much as 100% off, the minimum coloring. This paper shows that since real-life graphs appear to be *1-perfect*, one can indeed solve them *exactly* for a small overhead.

1 Introduction

Coloring a graph consists of assigning a color to every vertex so that no two vertices linked by an edge have the same color. The associated optimization problem consists of minimizing the number of colors. Graph coloring is used in microcode optimization [15, pp. 168–169], scheduling [8, pp. 248–252], resource binding and sharing [8, pp. 277–294] [15, pp. 230–233], (un)constrained state encoding of (a)synchronous finite state machines [15, pp. 323–327], and planar routing [6]. Other non-CAD applications include code compilation, frequency assignment, and network optimization. Because graph coloring is NP-complete, heuristics are used to produce an approximate solution.

This paper shows that since real-life coloring instances appear to be *1-perfect*, one can solve them *exactly* in no more time than heuristics, while heuristics are on average 10% off, and as much as 100% off, from the optimum.

This paper is organized as follows. Section 2 gives some definitions and notations. Section 3 presents the well-known sequential coloring algorithm, and pinpoints its main weakness. Based on experimental evidence, it then explains why solving the maximum clique problem is a decisive factor when coloring real-life graphs. Section 4 introduces original pruning techniques to solve maximum clique. Section 5 gives experimental results. It shows that *all* the real-life application instances we had access to (>600) are solved *exactly* in a few seconds.

DAC 97, Anaheim, California
(c) 1997 ACM 0-89791-920-3/97/06 .. $3.50

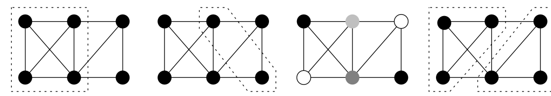

Figure 1: Max. clique, max. independent set, min. coloring, and min. clique partition.

2 Notations

A simple (i.e., undirected and self-loop free) graph G is denoted by $(V(G), E(G))$, where $V(G)$ is its set of vertices, and $E(G)$ its set of edges. We denote by $N(v)$ the set of neighbors of a vertex v in a given graph G, i.e., $N(v) = \{v' \in V(G) \mid \{v, v'\} \in E(G)\}$. The degree of a vertex is its number of neighbors, $|N(v)|$. Given a set of vertices V, we will often use the notation $G-V$ to denote the subgraph induced by $(V(G) - V, E(G))$. When the context is not ambiguous, we will denote a subgraph by its set of vertices.

In the sequel, n is the number of vertices, and k the number of colors used by a coloring. The *saturation* number of a vertex v is the number of colors used by its neighbors (i.e., the number of forbidden colors for v). We say that a color is *saturated* if it cannot be used anymore to extend a partial coloring.

A *clique* is a set of vertices that are all linked to each other by edges. An *independent set* is a set of vertices that are not connected by any edge. Partitioning the set of vertices into cliques is nothing but coloring the complementary graph. Fig. 1 illustrates these NP-complete problems [9]. An independent set is *maximal* iff it is not a proper subset of another independent set.

Let $\gamma(G)$ be the size of the maximum clique of G, and $\chi(G)$ be the chromatic number of G, i.e., the minimum number of colors needed to color G. Since every vertex of a clique must be assigned a different color, $\gamma(G) \leq \chi(G)$. When $\gamma(G) = \chi(G)$, we say that G is *1-perfect*[1].

3 Exact Coloring

Coloring a graph can be done in two ways. One can determine a color class one at a time: this consists of enumerating maximal independent sets. Or one can color the vertices one at a time: this is called sequential coloring.

[1] G is *perfect* iff *every* subgraph of G is 1-perfect. Exact coloring of perfect graphs is polynomial [10], but much too slow in practice.

```
function SC(G);
C ← a clique of G;
k ← 0;
foreach v ∈ C {                              /* color the clique */
   k ← k + 1;
   color v with k;                     /* a color is an integer ≥ 1 */
}
return SCrec(G, k, |V(G)| + 1, |C|);

/* G is a graph partially colored, using k colors, and     */
/* best is the chromatic number found so far               */
function SCrec(G, k, best, lb);
if G is entirely colored return k;      /* new best coloring */
v ← an uncolored vertex of G;
for  (c ← 1; c ≤ min(k + 1, best − 1); c ← c + 1) {
                                       /* for each potential color */
   if  (∀v' ∈ N(v), color(v') ≠ c) {   /* c is non-conflicting */
      color v with c;
      best ← SCrec(G, max(c, k), best, lb);
      uncolor v;
      if lb = best return best;       /* γ(G) = χ(G): abort */
   }
}
return best;
```

Figure 2: *SC*, the exact sequential coloring.

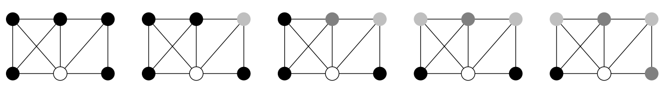

Figure 3: Sequential coloring.

This section discusses the sequential coloring algorithm. We pinpoint the main weakness of this algorithm, and explain why the maximum clique problem is a key player when coloring real-life graphs.

3.1 Sequential Coloring

Fig. 2 outlines the exact sequential coloring algorithm *SC* [5]. It first generates a clique, which is used both as a lower bound and as a starting point for the coloring, since every vertex of the clique must be assigned a different color and does not need to be recolored afterwards. Then uncolored vertices are picked one at a time, and each is assigned a color (an integer ≥ 1) non-conflicting with its neighbors' colors.

An efficient heuristic, the well known DSATUR algorithm [4], consists of picking the vertex that has the largest saturation number, and in breaking ties with the largest degree in the uncolored graph. The idea is to choose the vertex that is the most "difficult" to color, and that propagates as many constraints as possible. Fig. 3 (from left to right) shows how a simple graph is sequentially colored with this heuristic.

The reader is referred to [16] for an extensive description of some improvements and variations of sequential coloring (e.g., non-sequential backtracking [4, 18]).

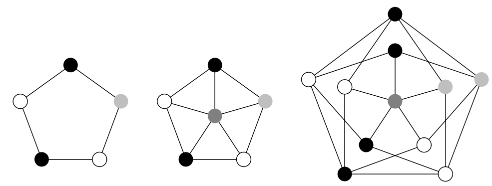

Figure 4: Three non 1-perfect graphs: $\gamma(G) < \chi(G)$.

3.2 Why is Sequential Coloring Hard?

The way the lower bound is used in *SC* is largely ineffective. As a comparison, consider a branch-and-bound algorithm that solves maximum clique (e.g., Fig. 5). Based on the inequality $\gamma(G) \leq \chi(G)$, a coloring is computed at each recursion and is used as an upper bound to prune the search tree of maximum clique. Conversely, a clique is a lower bound on the chromatic number of a graph. But the analogy ends here: a clique does not give any valuable information on a graph partially colored with *unsaturated* colors. Indeed, quickly estimating a lower bound on the number of colors necessary to optimaly complete an unsaturated coloring is an open problem.

SC uses several unsaturated colors at the same time (e.g., the two gray colors used in the middle graph of Fig. 3), and thus has only one *static* lower bound which is not reevaluated at each recursion, unlike "standard" branch-and-bound algorithms. We therefore have the following fact (e.g., [13, pp. 220]):

Fact 1 *If $\gamma(G) < \chi(G)$, then the lower bound does not influence the length of the computation at all, because the search must exhaustively enumerate all potential (unsuccessful) colorings that would improve on $\chi(G)$, which can take exponential time.*

Let us face the second fact ([2, 3], [13, pp. 243–247]):

Fact 2 *Almost all graphs G satisfy:*

$$\gamma(G) < 4 \log n < \frac{n}{3 \log n} < \chi(G).$$

This shows an actual large gap between $\gamma(G)$ and $\chi(G)$. Combined with Fact 1, this leaves little hope to address exact coloring in general.

3.3 Why is Maximum Clique Important?

However, Fact 3 gives a different perspective on exact graph coloring from the practical point of view:

Fact 3 All *the practical instances we found (more than 600 real-life examples in scheduling, register allocation, planar routing, and frequency assignment) are 1-perfect graphs, i.e., $\gamma(G) = \chi(G)$.*

For instance, the graph of Fig. 3 is 1-perfect. Fig. 4 shows non 1-perfect graphs (the one on the right is *myciel3*, see Section 5).

function $MaxClique(G)$;
return $MaxCliqueRec(G, \emptyset, \emptyset, +\infty)$;

```
/* G is the remaining graph, C is the clique under con-    */
/* struction, and best is the largest clique found so far.  */
function MaxCliqueRec(G, C, best, ub);
if G is empty return C;                  /* new best solution */
{I1, ..., Ik} ← a coloring of G;
ub   ← min(ub, |C| + k);        /* compute an upper bound */
if ub ≤ |best| return best;                         /* prune */
v    ← a maximum degree vertex of G;
G1   ← graph induced by N(v);      /* force v in the clique */
best ← MaxCliqueRec(G1, C ∪ {v}, best, ub);
if ub = |best| return best;                         /* prune */
G0   ← graph induced by V(G) − {v};             /* exclude v */
return MaxCliqueRec(G0, C, best, ub);
```

Figure 5: Maximum clique.

Finding a maximum clique is tremendously important when coloring 1-perfect graphs, since the search is aborted as soon as one finds a coloring whose cardinality is $\gamma(G)$. If the clique is not maximum, then Fact 1 applies, and the algorithm will not find the optimum solution and/or will not terminate within a reasonable time. Fact 3 makes maximum clique as important in practice as coloring itself.

4 Maximum Clique

This section shows how to solve maximum clique, and proposes an original pruning technique that drastically reduces the search space.

Fig. 5 shows a simplified branch-and-bound algorithm for solving maximum clique. One can add the following improvements:

(a) When $|C| + |V(G)| \leq |best|$, the recursion is pruned, because it is impossible to find a larger clique.

(b) Every vertex v such that $v.degree < |best| - |C|$ must be removed from the graph, because it cannot be a member of a larger clique.

(c) Every vertex v such that $v.degree \geq |V(G)| - 2$ must be put in the clique under construction, since excluding it cannot produce a larger clique.

(d) More generally, a vertex v such that $V(G) - N(v)$ is an independent set must be put in the clique under construction, since excluding it cannot produce a larger clique.

(e) One can force the choice of at least 2 non-neighbors of v in $G0$. In other words, the maximum clique of G is either

$$\{v\} \cup MaxClique(N(v)),$$

or

$$\{v_1, v_2\} \cup MaxClique(N(v_1) \cap N(v_2)),$$

where $v_1, v_2 \in V(G) - N(v) - \{v\}$, and $v_2 \in N(v_1)$.

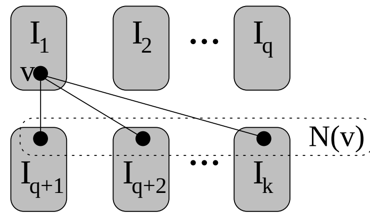

Figure 6: q-colorable vertices can be removed.

Rules (a)-(c) are trivial to implement. Rule (d) is in $O(|V(G)|^2)$, which introduces too large an overhead compared to the practical gain. Rule (e) is not costly, but is more delicate to implement.

The following result presents an original pruning method which can be efficiently implemented, and which dramatically reduces the search space.

Theorem 1 (q-color pruning) *Let G be the graph at some point of the recursion, C the clique under construction, and best the current best solution. Let $\{I_1, \ldots, I_k\}$ be a k-coloring obtained on G. Then every vertex v that can be colored with q colors, where $q > |C| - |best| + k$, can be removed from the graph.*

Proof. Fig. 6 shows the k-coloring of G, i.e., the partition of the vertices of G into k independent sets $I_1, \ldots, I_k$. Assume that the vertex v can be colored with q colors. Without loss of generality, this means that $I_j \cup \{v\}$ is an independent set for $1 \leq j \leq q$. Let $C1$ be the largest clique that can be obtained by forcing v in C. We then obtain:

$$\begin{aligned} |C1| &= |C \cup \{v\} \cup MaxClique(N(v))| & (1)\\ &= |C| + 1 + \gamma(N(v)) & (2)\\ &\leq |C| + 1 + \chi(N(v)) & (3)\\ &\leq |C| + 1 + k - q & (4)\\ &\leq |best| & (5) \end{aligned}$$

Inequality (4) holds because $N(v)$ is necessarily a subset of $\bigcup_{j=q+1}^{k} I_j$, and thus $\{I_{q+1}, \ldots, I_k\}$ is a valid $(k-q)$-coloring of $N(v)$. Inequality (5) holds because of the assumption on q. Since one cannot find a larger clique by selecting v, one can remove it from the graph. ■

Even if k is too large (i.e., $|C| + k > |best|$) to produce a "normal" pruning, Theorem 1 shows that q-colorable vertices yield unsuccessful branches, and can be removed. This reduces the number of choice points, but the effectiveness of this pruning technique is its snowball effect. Removing vertices gives more opportunities to apply rules (a)-(d). Vertices that are removed are also *uncolored*, which frees some colors for their neighbors, which increases their own q's, which infers more removal. Removing vertices can empty an independent set, which decreases k, which loosens the constraint on q and produces more removal. Eventually k becomes small enough to prune the recursion.

examples				without		with	
name	**|V|**	**|E|**	γ	**#back**	**CPU**	**#back**	**CPU**
school1_nsh	385	16710	14	2414	8.16	338	0.92
keller4	171	9435	11	30047	51.5	4964	4.87
sanr200_0.7	200	13868	18	206811	488.4	24780	23.0
brock200_1	200	14834	21	777895	2184.7	100900	112.9
san200_0.7_2	200	13930	18	12996	93.2	696	1.66
p_hat300-2	300	21928	25	57761	481.0	1211	4.21
hamming8-4	256	20864	16	4147	26.3	1	0.18
san200_0.9_1	200	17910	70	11236823	$17h\ 30mn$	507	5.61
MANN_a27	378	70551	126	–	> 2 *days*	3451	98.4

For each graph, we give: its number of vertices (|**V**|), its number of edges (|**E**|), its clique number (γ), the number of backtracks (**#back**) performed to solve maximum clique, and the **CPU** time in seconds on a 60 MHz SuperSparc (85.4 SpecInt). **without** is the "standard" branch-and-bound algorithm shown in Fig. 5, and **with** is the improved version described in Section 4.

Table 1: Solving Maximum Clique.

A notable aspect of this pruning technique is its no gain/no cost aspect. Using the *SC* algorithm without backtrack to find the k-coloring, the number of colors that can be used to color a vertex v is nothing but v's number of unconstrained colors, i.e., k minus v's saturation number, which is computed in $O(1)$. Using a priority queue that keeps the vertices in decreasing saturation number, one can test for the removal of the vertices from the tail of the queue up to its head. The first failure of the test indicates that one can stop the whole pruning procedure. Thus if no pruning is possible, the overhead is in $O(1)$. If r vertices can be removed (the last r vertices of the queue), the overhead is in $O(r \times |V(G)|)$ for a potentially exponential benefit.

Experience shows that thanks to this original pruning technique, the search space is reduced by several orders of magnitude, drastically speeding up maximum clique (Table 1).

On real-life examples, this pruning technique quickly leads the algorithm to a maximum clique. Where one previously needed about 1000 backtracks, and up to 10000 backtracks, less than 10 backtracks are now necessary to *find* (not necessarily *prove*) an optimum solution.

5 Experimental Results

This section presents experiments done with real-life applications, combinatorics instances, and (artificial[2]) hard examples. The planar routing instances come from [6]. The other instances come from [7].

5.1 Heuristic Coloring

We compared three widely used coloring heuristics[3], *H1*, *H2*, and *H3*. *H3* consists of forbidding any backtrack in the sequential coloring *SC*.

[2] Mycielski graphs [17] are difficult to color because their clique number is 2, while their chromatic number increases in problem size. Leighton graphs [14] are difficult to color because their optimum coloring is hidden among many suboptimal solutions.

[3] Graph coloring is not polynomially approximable within $n^{1/7-\epsilon}$ for any $\epsilon > 0$ [1]. The best known approximation ratio is in $O(n(\log\log n)^2/(\log n)^3)$ [11].

```
function ColorWithIndSet(G);
𝓘 ← ∅;
while G is not empty {
   I ← a maximal independent set of G;
   𝓘 ← 𝓘 ∪ {I};
   G ← graph induced by V(G) − I;
}
return 𝓘;
```

Figure 7: Heuristic coloring with independent sets.

```
function FindIndSetH1(G);
I ← ∅;
while G is not empty {
   v ← vertex of minimum degree;
   I ← I ∪ {v};
   G ← graph induced by V(G) − {v} − N(v);
}
return I;

function FindIndSetH2(G);
I ← ∅;
while G is not empty {
   if I = ∅
      v ← vertex of maximum degree;
   else
      v ← vertex of max. removed edges, then min. degree;
   I ← I ∪ {v};
   G ← graph induced by V(G) − {v} − N(v);
}
return I;
```

Figure 8: Color class for heuristics *H1* and *H2*.

Fig. 7 shows a heuristic coloring algorithm. It consists of adding a maximal independent set I (i.e., a saturated color class) to a coloring $\mathcal{I}$ under construction, removing I from G, and iterating this process until G is empty. Heuristic *H1* consists of using a greedy algorithm designed for *maximum* independent set (Fig. 8) to produce the maximal independent sets. *H1* is guaranteed to find a coloring within $O(n/\log n)$ of the optimum [12].

Instead of looking for a large maximal independent set, one can look for a maximal independent set that minimizes the number of edges connected to uncolored

name	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	γ	χ	*H1*	*H2*	*H3*
DSJC125.1	125	736	4	5	8	7	**6**
DSJR500.1	500	3555	12	12	16	13	**12**
MANN_a9	45	918	16	18	**18**	20	19
R1000.1	1000	14378	20	20	23	**20**	**20**
R125.5	250	3838	36	36	51	39	**38**
R250.1c	250	30227	64	64	72	68	**65**
c-fat200-1	200	1534	12	12	15	**13**	15
d2esp.i.1	319	8534	61	61	63	**61**	63
ex3a	44	176	10	10	11	11	**10**
ex3c	54	336	12	12	13	13	**12**
exam1	200	17124	126	126	137	127	**126**
exam2	250	26081	141	141	154	147	**142**
exam3	300	36801	162	162	177	164	**162**
flat1000_50_0	1000	245000	14	50	**104**	110	113
flat300_20_0	300	21375	11	20	**40**	41	42
fpsol2.i.2	451	8691	30	30	35	**30**	**30**
le450_15d	450	16750	15	15	31	**25**	**25**
le450_25a	450	8260	25	25	31	26	**25**
le450_25c	450	17343	25	25	38	30	**28**
le450_5c	450	9803	5	5	**9**	**9**	11
le450_5d	450	9757	5	5	**8**	10	11
queen6_6	36	290	6	7	**8**	9	9
queen7_7	49	476	7	7	**10**	**10**	**10**
queen8_8	64	728	8	9	12	**11**	13
queen9_9	81	2112	9	10	13	**12**	**12**
queen11_11	121	3960	10	11	16	16	**14**
queen13_13	169	6656	13	13	18	18	**17**
san200_0.7_2	200	13930	18	18	**20**	26	23
sgelq2.i.2	182	3254	26	26	29	**26**	28
school1	385	19095	14	14	36	30	**17**
school1_nsh	352	14612	14	14	32	**25**	26
average	313	16710	28.5	30.2	38.5	35.7	34.8

For each graph, we give: its number of vertices ($|\mathbf{V}|$), its number of edges ($|\mathbf{E}|$), its clique number (γ), its chromatic number (χ), and the number of colors obtained with heuristics *H1*, *H2*, and *H3*. Heuristics are 10% off, and as much as 100% off, the optimum solution.

Table 2: Heuristic coloring.

vertices (Fig. 8) [14]. This heuristic, *H2*, reduces the number of *conflicts* with the uncolored vertices so that less color classes are needed to complete the coloring.

Table 2 compares these three heuristics. Clearly, *H2* and *H3* are better than *H1*, but none of them wins consistently. It happens that there is a large gap between the heuristic colorings and the exact solution, even on real-life examples, e.g., the scheduling problem *school1_nsh*.

5.2 Exact Coloring

Table 3 gives the performance of exact coloring on real-life application instances (selected among more than 600 examples), and on combinatorics, hard, and random examples. The coloring algorithm is the sequential coloring described in Section 3.1, using the clique produced by algorithm of Section 4 in no more than 10 backtracks.

The combinatoric, artificial, and random examples are more difficult, especially when the graph is not 1-perfect: in that case, the algorithm has to enumerate all the optimum colorings before terminating, which can be exponential (Fact 1 of Section 3.2).

All the 600 real-life examples are solved exactly, even the large graphs (> 6000 nodes, > 500000 edges). This is because they are all 1-perfect, and because the clique algorithm introduced in Section 4 quickly finds the optimum lower bound. A way of comparing these results with the state-of-the-art consists of assuming that one finds a suboptimum clique (which is often the case with "standard" heuristics). Assuming that one only finds a clique of size $\gamma(G) - 1$, most of the examples cannot be solved in less than one hour, and many of them remain unsolved after 2 days (e.g., the scheduling examples and most of the resource allocation problems).

6 Discussion & Conclusion

This paper has explained how to improve on graph coloring, which is a key application in scheduling, resource allocation, constrained encoding, multi-layer topological routing, etc. When a graph is 1-perfect, and *providing that one finds a maximum clique*, the coloring is easy. Despite our effort, we did not find a real-life example that is not 1-perfect. Based on this experimental fact, and thanks to an improved maximum clique computation algorithm, a sequential coloring algorithm can solve all our real-life instances *exactly* in a matter of seconds.

This tends to show that, in practice, and in particular for CAD applications, one can afford to solve graph coloring exactly: for roughly the same CPU time, one is rewarded with an optimum result, while heuristic solutions are typically 10% off, and as much as 100% off, the minimum coloring.

References

[1] M. Bellare, O. Goldreich, M. Sudan, "Free bits, PCPs and non-approximability – towards tight results", Proc. of *36th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pp. 422–431, 1995.

[2] B. Bollobás, P. Erdös, "Cliques in Random Graphs", *Math. Proc. Camb. Phil. Soc.*, **80**, pp. 419–427, 1976.

[3] B. Bollobás, "The Chromatic Number of Random Graphs", *Combinatorica*, **8**-1, pp. 49–55, 1988.

[4] D. Brélaz, "New Methods to Color Vertices of a Graph", *Comm. of the ACM*, **22**-4, pp. 251–256, 1979.

[5] J. R. Brown, "Chromatic Scheduling and the Chromatic Number Problem", *Management Sci.*, **19**, pp. 456–463, 1972.

[6] J. Cong, M. Hossain, N. A. Sherwani, "A Provably Good Multilayer Topological Planar Routing Algorithm in IC Layout Designs", *IEEE Trans. on CAD*, **12**-1, pp. 70–78, Jan. 1993.

[7] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmark/, benchmark for graph coloring and clique, 1993.

[8] D. Gajski, N. Dutt, A. Wu, S. Lin, *High-Level Synthesis*, Kluwer Ac. Pub., 1992.

[9] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP–Completeness*, Freeman, 1979.

name	|V|	|E|	γ	χ	#back	CPU
scheduling						
school1_nsh	352	14612	14	14	11	0.25
school1	385	19095	14	14	12	0.41
register allocation						
mulsol.i.1	197	3925	49	49	2	0.10
interp.i.1	253	5039	39	39	3	0.15
d2esp.i.1	319	8534	61	61	1	0.16
sgemm.i.1	439	8458	55	55	0	0.17
fpsol2.i.1	496	11654	65	65	4	0.27
slahr2.i.2	557	11535	29	29	7	0.39
spbtrf.i.2	823	16250	30	30	4	0.67
conduct.i.1	1185	27013	54	54	7	1.33
slasbr.i.1	1752	72265	87	87	2	3.70
slaein.i.1	2337	71600	73	73	2	4.93
inidat.i.1	2408	114388	136	136	4	15.4
deseco.i.1	2826	86688	117	117	3	12.4
h2d.i.1	3072	228151	171	171	3	19.8
twldrv.i.1	4905	338709	227	227	3	37.5
fpppp.i.1	5439	543223	212	212	1	45.5
wanal1.i.1	6760	190975	71	71	2	39.6
planar routing						
burs	24	133	9	9	1	0.01
ex1	21	77	7	7	1	0.01
ex3a	44	176	10	10	1	0.01
ex3b	47	283	9	9	1	0.01
ex3c	54	336	12	12	0	0.02
ex4b	54	298	11	11	0	0.02
ex5	64	405	9	9	1	0.01
ex5b	64	427	10	10	0	0.02
deut	72	763	16	16	1	0.02
exam1	200	17124	126	126	2	0.46
exam2	250	26081	141	141	10	0.96
exam3	300	36801	162	162	6	1.45
frequency assignment						
man7	548	3250	10	10	0	0.11
man8	858	4023	10	10	0	0.30

name	|V|	|E|	γ	χ	#back	CPU
queen graphs						
queen5_5	25	160	5	5	4	0.01
queen6_6	36	290	6	7	897	0.03
queen7_7	49	476	7	7	1852	0.07
queen8_8	64	728	8	9	1824457	38.69
queen9_9	81	2112	9	10	561222078	$5h$ $12mn$
Mycielski transformation based graphs						
myciel3	11	20	2	4	23	0.01
myciel4	23	71	2	5	539	0.02
myciel5	47	236	2	6	191488	4.17
myciel6	95	755	2	7	3287401951	$35h$ $22mn$
Leighton graphs						
le450_25a	450	8260	25	25	0	0.13
le450_25b	450	8263	25	25	0	0.11
le450_5c	450	9803	5	5	232	3.04
le450_5d	450	9757	5	5	171271	31.40
misc. graphs						
MANN_a9	45	918	16	18	60	0.08
hamming6-4	64	704	4	7	1517	0.20
c-fat200-1	200	1534	12	12	325	0.43
c-fat200-2	200	3235	24	24	0	0.41
c-fat500-1	500	4459	14	14	1	1.72
san200_0.7_2	200	13930	18	18	42377	16.52
c-fat500-2	500	9139	26	26	1	2.63
c-fat500-5	500	23191	64	64	1	5.46
c-fat500-10	500	46627	126	126	2	4.35
random graphs						
DSJC125.1	125	736	4	5	2512	0.16
DSJR500.1	500	3555	12	12	0	0.12
R125.1	125	209	5	5	0	0.02
R125.5	250	3838	36	36	41167	2.60
R125.1c	125	7501	46	46	0	0.13
R250.1	250	867	8	8	0	0.05
R250.1c	250	30227	64	64	15	2.13
R1000.1	1000	14378	20	20	0	0.54

For each graph, we give: its number of vertices (|**V**|), its number of edges (|**E**|), its clique number (γ), and its chromatic number (χ). Note that all the real-life examples are 1-perfect. We give the number of backtracks (**#back**) performed to solve the minimum coloring. The **CPU** time is given in seconds on a 60 MHz SuperSparc (85.4 SpecInt), and includes: reading the graph description, building the internal data structure, solving the minimum coloring problem, and finally freeing the memory.

Table 3: Coloring of real-life application graphs (left), and of hard artificial graphs (right).

[10] M. Gröetschel, L. Lovász, A. Schrijver, "The Ellipsoid Method and its Consequences in Combinatorial Optimization", Combinatorica, **1**, pp. 169–197, 1981.

[11] M. M. Halldórsson, "A still better performance guarantee for approximate graph coloring", Inform. Process. Lett., **45**, pp. 19–23, 1993.

[12] D. S. Johnson, "Worst-Case Behavior of Graph-Coloring Algorithms", Proc. *5th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, pp. 513–528, Winnipeg, 1974.

[13] L. Kučera, *Combinatorial Algorithms*, Adam Hilger, 1990.

[14] F. T. Leighton, "A Graph Coloring Algorithm for Large Scheduling Problems", *J. Res. Nat. Bur. Standards*, **84**, pp. 489–506, 1979.

[15] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[16] C. A. Morgenstern, *Algorithms for General Graph Coloring*, Ph.D. thesis, Department of Computing Science, University of New Mexico, Albuquerque NM, 1990.

[17] J. Mycielski, "Sur le Coloriage des Graphes", Colloq. Math., **3**, 1955

[18] J. Peemöller, "A Correction to Brélaz's Modification of Brown's Coloring Algorithm", *Comm. of the ACM*, **26**, pp. 595–597, 1983.