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.

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’ ∈ V(G) | {v, v’} ∈ 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.

![Figure 1: Max. clique, max. independent set, min. coloring, and min. clique partition.](image-url)
function $SC(G)$;
$C \leftarrow$ a clique of $G$;
$k \leftarrow 0$;
foreach $v \in C$ {
  /* color the clique */
  $k \leftarrow k + 1$;
  color $v$ with $k$;
  /* a color is an integer $\geq 1$ */
}
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 */
if $G$ is entirely colored return $k$;
/* new best coloring */
$v \leftarrow$ an uncolored vertex of $G$;
for ($c \leftarrow 1 ; c \leq \min(k + 1, best - 1); c \leftarrow c + 1$) {
  if ($\forall v' \in N(v), color(v') \neq c$) {
    /* for each potential color */
    best $\leftarrow SCrec(G, max(c, k), best, lb)$;
    uncolor $v$;
    if $lb = best$ return $best$;  /* $\gamma(G) = \chi(G)$: abort */
  }
}
return $best$;

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

Figure 3: Sequential coloring.

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 enu-
merating maximal independent sets. Or one can color the
vertices one at a time: this is called sequential coloring.

This section discusses the sequential coloring algo-
rithm. 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

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

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 $\geq 1$) non-conflicting with its
neighbors’ colors.

An efficient heuristic, the well known DSATUR al-
gorithm [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 sequen-
tially colored with this heuristic.

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

3.2 Why is Sequential Coloring Hard?

The way the lower bound is used in $SC$ is largely inef-
fective. 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 com-
puted 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 col-
ored with unsaturated colors. Indeed, quickly estimat-
ing a lower bound on the number of colors necessary to
optimally 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 second and third
graphs 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 (unsuc-
cessful) 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:

$$\gamma(G) < 4 \log n < \frac{n}{3 \log n} < \chi(G).$$
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 $v.\text{degree} < |\text{best}| - |C|$ must be removed from the graph, because it cannot be a member of a larger clique.

(c) Every vertex $v$ such that $v.\text{degree} \geq |V(G)| - 2$ 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

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

or

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

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

Rules (a)-(e) 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.

\[ |C| = |C \cup \{v\} \cup MaxClique(N(v))| \]

Inequality (4) holds because \( N(v) \) is 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.

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).

Table 2: Heuristic coloring.

| name           | | \( |V| \) | \( |E| \) | \( \gamma \) | \( |\chi| \) | H1 | H2 | H3 |
|----------------|------|----------|---------|----------|----------|----|----|----|
| DSJC125.1      | 125  | 736      | 4       | 5        | 8        | 7  | 6  |
| DSJ1R500.1     | 500  | 3555     | 12      | 12       | 16       | 13 | 12 |
| MANN_a9        | 45   | 918      | 16      | 18       | 18       | 20 | 19 |
| R100.1         | 1000 | 14378    | 20      | 20       | 23       | 20 | 20 |
| R125.5         | 250  | 3838     | 36      | 36       | 51       | 39 | 39 |
| R250.1c        | 250  | 30227    | 64      | 64       | 72       | 68 | 65 |
| c-fat200-1     | 200  | 1534     | 12      | 12       | 15       | 15 | 13 |
| 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_L0| 1000 | 245000   | 14      | 50       | 104      | 110| 113|
| flat300_20_20 | 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       | 20 | 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 |
| san300_0.7_2   | 200  | 13930    | 18      | 18       | 20       | 26 | 23 |
| sgelg2.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 |

For each graph, we give: its number of vertices (|V|), its number of edges (|E|), its clique number (\( \chi \)), its chromatic number (\( \gamma \)), 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.
5 Experimental Results

This section presents experiments done with real-life applications, combinatorics instances, and (artificial\textsuperscript{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\textsuperscript{3}, $H1$, $H2$, and $H3$. $H3$ consists of forbidding any backtrack in the sequential coloring $SC$.

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

\begin{tabular}{|c|c|c|c|c|c|}
\hline
name & \(|V|\) & \(|E|\) & \(|\gamma|\) & without #back & CPU & with #back & CPU \\
\hline
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 \\
sanr200\_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 \\
sanr200\_0\_9\_1 & 200 & 17910 & 70 & 11236823 & 17h 30min & 507 & 5.61 \\
MANN\_a27 & 378 & 70551 & 126 & – & > 2 days & 3451 & 98.4 \\
\hline
\end{tabular}

For each graph, we give: its number of vertices ($|V|$), its number of edges ($|E|$), its clique number ($|\gamma|$), 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.

\begin{figure}[h]
\begin{center}
\begin{verbatim}
function ColorWithIndSet(G);
  I ← Ø;
  while G is not empty {
    I ← a maximal independent set of G;
    I ← I ∪ {i};
    G ← graph induced by $V(G) - I$;
  }
  return I;
\end{verbatim}
\end{center}
\caption{Heuristic coloring with independent sets.}
\end{figure}

\begin{figure}[h]
\begin{center}
\begin{verbatim}
function FindIndSetH1(G);
  I ← Ø;
  while G is not empty {
    v ← vertex of minimum degree;
    I ← I ∪ {v};
    G ← graph induced by $V(G) - \{v\} - N(v)$;
  }
  return I;
\end{verbatim}
\end{center}
\caption{Color class for heuristics $H1$ and $H2$.}
\end{figure}

minimizes the number of edges connected to uncolored 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:
### Table 3: Coloring of real-life application graphs (left), and of hard artificial graphs (right).

<table>
<thead>
<tr>
<th>name</th>
<th></th>
<th></th>
<th></th>
<th>back</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V</td>
<td>E</td>
<td>γ</td>
<td>χ</td>
<td>#back</td>
</tr>
<tr>
<td>school1</td>
<td>352</td>
<td>14012</td>
<td>14</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>school2</td>
<td>385</td>
<td>19095</td>
<td>14</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>register allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>musol.1</td>
<td>197</td>
<td>3925</td>
<td>49</td>
<td>49</td>
<td>2</td>
</tr>
<tr>
<td>interp.1</td>
<td>253</td>
<td>5039</td>
<td>39</td>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td>d2exp.1</td>
<td>319</td>
<td>8534</td>
<td>61</td>
<td>61</td>
<td>1</td>
</tr>
<tr>
<td>sgemm.1</td>
<td>439</td>
<td>8458</td>
<td>55</td>
<td>55</td>
<td>0</td>
</tr>
<tr>
<td>fspol.1</td>
<td>496</td>
<td>11654</td>
<td>65</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>slahr.1</td>
<td>557</td>
<td>11535</td>
<td>29</td>
<td>29</td>
<td>7</td>
</tr>
<tr>
<td>spbryf.1</td>
<td>823</td>
<td>16250</td>
<td>30</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>conduct.1</td>
<td>1185</td>
<td>27013</td>
<td>54</td>
<td>54</td>
<td>7</td>
</tr>
<tr>
<td>slaer.1</td>
<td>1752</td>
<td>72265</td>
<td>87</td>
<td>87</td>
<td>2</td>
</tr>
<tr>
<td>slaein.1</td>
<td>2337</td>
<td>71600</td>
<td>73</td>
<td>73</td>
<td>2</td>
</tr>
<tr>
<td>instd.1</td>
<td>2408</td>
<td>114388</td>
<td>136</td>
<td>136</td>
<td>4</td>
</tr>
<tr>
<td>deseco.1</td>
<td>2826</td>
<td>86688</td>
<td>117</td>
<td>117</td>
<td>3</td>
</tr>
<tr>
<td>h2d.1</td>
<td>3072</td>
<td>228151</td>
<td>171</td>
<td>171</td>
<td>3</td>
</tr>
<tr>
<td>t0ldrv.1</td>
<td>4905</td>
<td>383709</td>
<td>227</td>
<td>227</td>
<td>3</td>
</tr>
<tr>
<td>fpplp.1</td>
<td>5439</td>
<td>543223</td>
<td>212</td>
<td>212</td>
<td>1</td>
</tr>
<tr>
<td>wanal1.1</td>
<td>6760</td>
<td>190975</td>
<td>71</td>
<td>71</td>
<td>2</td>
</tr>
<tr>
<td>planar routing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>burs</td>
<td>24</td>
<td>133</td>
<td>9</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>ext1</td>
<td>21</td>
<td>77</td>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>ex3a</td>
<td>44</td>
<td>176</td>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>ex3b</td>
<td>47</td>
<td>283</td>
<td>9</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>ex3c</td>
<td>54</td>
<td>336</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>ex4b</td>
<td>54</td>
<td>298</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>ex5</td>
<td>64</td>
<td>405</td>
<td>9</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>ex5b</td>
<td>64</td>
<td>427</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>delet</td>
<td>72</td>
<td>763</td>
<td>16</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>exam1</td>
<td>200</td>
<td>17124</td>
<td>126</td>
<td>126</td>
<td>2</td>
</tr>
<tr>
<td>exam2</td>
<td>250</td>
<td>26801</td>
<td>141</td>
<td>141</td>
<td>10</td>
</tr>
<tr>
<td>exam3</td>
<td>300</td>
<td>36801</td>
<td>162</td>
<td>162</td>
<td>6</td>
</tr>
<tr>
<td>frequency assignment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>man7</td>
<td>548</td>
<td>3250</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>man8</td>
<td>858</td>
<td>4023</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

For each graph, we give: its number of vertices (\(|V|\)), its number of edges (\(|E|\)), its clique number (\(\gamma\)), and its chromatic number (\(\chi\)). 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 85.4 SpecInt, and includes: reading the graph description, building the internal data structure, solving the minimum coloring problem, and finally freeing the memory.

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 particu-
lar 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


