Coloring Real-Life Graphs

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

Abstract

Graph coloring has several applications in compilation and VLSI CAD [15, 9]. Since it is NP-complete, heuristics are used in practice 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 instances appear to be 1-perfect, one can indeed solve them exactly for a small overhead.

1 Introduction

Graph coloring consists of assigning a color to every vertex of a graph so that no two vertices linked by an edge have the same color, and by using a minimum number of colors. The purpose of this paper is to show that one can solve exactly real-life coloring instances in no more time than heuristics, while heuristics are on average 10% off, up to 100%, from the optimum.

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

2 Notations

A 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, i.e., $|N(v)|$. 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.

In the sequel, $n$ is the number of vertices, and $k$ the number of colors used by a coloring. Given a set of vertices $V$, we will often use the notation $G − V$ to denote the graph induced by $(V(G) − V, E(G))$. When the context is not ambiguous, we will denote a subgraph by its set of vertices.

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. Fig. 1 illustrates these NP-complete problems [10].

Let $\gamma(G)$ be the size of the maximum clique of $G$, and $\chi(G)$ be the chromatic number of $G$. Since every element 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.

3 Sequential Coloring

Fig. 2 shows the exact sequential coloring algorithm [4] SC. 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 are assigned a color (an integer $\geq 1$) non-conflicting with its neighbors’ colors.

An efficient heuristic, the well known DSATUR algorithm [3], 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.

1$G$ is perfect iff every subgraph of $G$ is 1-perfect. Exact coloring of perfect graphs is polynomial [11, 12], but much too slow in practice.
function \textit{SC}(G);
C \leftarrow \text{a clique of } G;
k \leftarrow 0;
\text{foreach } v \in C \{ /* color the clique */
k \leftarrow k + 1;
\text{color } v \text{ with } k;
/* a color is an integer } \geq 1 /*
\}
\text{return } \text{SCrec}(G, k, |V(G)| + 1, |C|);
/
/* G is a graph partially colored, using } k \text{ colors, and } /*
/* best \text{ is the chromatic number found so far } */
\text{if } G \text{ is entirely colored } \text{return } k;
/* } \text{new best coloring } */
v \leftarrow \text{an uncolored vertex of } G;
\text{for } (c \leftarrow 1; c \leq \min(k + 1, \text{best } - 1); c \leftarrow c + 1) \{ /*
\text{for each potential color } */
\text{if } (\forall v' \in N(v), \text{color}(v') \neq c) \{ /* \text{c is non-conflicting } */
\text{color } v \text{ with } c;
\text{best } \leftarrow \text{SCrec}(G, \text{max}(c, k), \text{best }, \text{lb});
\text{uncolor } v;
\text{if } \text{lb } = \text{best } \text{return } \text{best } ; /* \gamma(G) = \chi(G) : abort */
\}
\text{return } \text{best } ;

Figure 2: \textit{SC}, the exact sequential coloring.

Figure 3: Sequential coloring.

3.1 Why is Coloring Hard?

The way the lower bound is used in \textit{SC} is largely ineffective. As a comparison, consider a branch-and-bound algorithm to solve maximum clique: a coloring is computed at each step of the recursion, and is used as an upper bound to prune the search tree (e.g., Fig. 5). 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 optimally complete an unsaturated coloring is an open problem.

\textit{SC} uses several unsaturated colors at the same time, and thus have only one static lower bound. We therefore have the following fact (e.g., [13, pp. 220]):

\textbf{Fact 1} If } \gamma(G) < \chi(G), \text{ 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), \text{ which can take an exponential time.}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{sequential-coloring.png}
\caption{Sequential coloring.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{sequential-coloring2.png}
\caption{Sequential coloring.}
\end{figure}

4 Maximum Clique

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

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{max-clique.png}
\caption{Maximum clique.}
\end{figure}

\section{Maximum Clique}

Fig. 5 shows a simplified branch-and-bound algorithm solving maximum clique. The following result presents an original pruning method which can be efficiently implemented, and which dramatically reduces the search space.

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

| examples | name       | |E| | V | γ | #back | CPU  | with #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 | 260811 | 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_Jat300-2 | 300 | 21928 | 25 | 57761 | 481.0 | 1211 | 4.21 |
| san200_0.9_1 | 200 | 17910 | 70 | 1123683 | 17h 30mn | 507 | 5.61 |
| MANN_4a27 | 378 | 70551 | 126 | – | > 2 days | 3451 | 98.4 |

For each graph, we give its number of vertices (|V|), its number of edges (|E|), and its clique number (γ). We give the number of backtracks (#back) performed to solve maximum clique, and the CPU time is given 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.

```plaintext
function MaxClique(G):
    return MaxCliqueRec(G, O, O, +∞);

/* G is the remaining graph, C is the clique under construction, and best is the largest clique found so far. */

function MaxCliqueRec(G, C, best, ub):
    if G is empty return C;
    /* new best solution */
    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.

**Proof.** Fig. 6 shows the k-coloring of G, i.e., the partition of the vertices of G into k independent sets I1, . . . , Ik. Assume that the vertex v can be colored with q colors. Without loss of generality, this means that Ij ∪ {v} is an independent set for 1 ≤ j ≤ q. Let C1 be the largest clique that can be obtained by adding v to C. We then obtain:

\[
|C1| = |C ∪ \{v\} ∪ MaxClique(N(v))| \quad (1)
\]

\[
|C1| = |C| + 1 + γ(N(v)) \quad (2)
\]

\[
|C1| ≤ |C| + 1 + χ(N(v)) \quad (3)
\]

\[
|C1| ≤ |C| + 1 + k - q \quad (4)
\]

\[
|C1| ≤ |best| \quad (5)
\]

Inequality (4) holds because |Iq+1, . . . , Ik| necessarily contains N(v), and thus 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 to unsuccessful branches, and can be removed. This reduces the number of choice points, but the effectiveness of this pruning technique is its snowball effect. Vertices that are removed are also uncolored, which frees some colors for their neighbors, which increases their q’s, which infers more removal. Removing vertices can empty an independent set, which decreases k, which looses 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) with color constraints propagation to find the k-coloring, the value q of 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 numbers, 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.

Experiences shows that thanks to this pruning technique, the search space is reduced by several orders of magnitude, drastically speeding up maximum clique (Table 1). On real-life examples, this quickly leads the algorithm to a maximum clique. Where one previously needed up to 10000 backtracks, less than 10 are now necessary to find (not necessarily prove) an optimum solution.

5 Experimental Results

Table 2 shows the performance of the exact coloring algorithm on some real-life application instances. The planar routing instances come from [16, 5, 7]. The frequency assignment problems come from [14]. The other instances come from [8]. The exact algorithm is the sequential coloring shown in Section 3, using the clique generated by algorithm of Section 4 in no more than 10 backtracks.

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). If the clique is not maximum, the algorithm has to enumerate all the optimum coloring before terminating, which can be exponential (Fact 1 of Section 3.1). Assuming that one only finds a clique of size $\gamma(G) - 1$, most of the examples cannot be solved in less than hours, and many of them remains unsolved after 2 days (e.g., the two scheduling examples, most of the resource allocation problems, le450.5c, R125.5, R250.1c, etc).

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 any 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 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

Table 2: Coloring of real-life applications.

| name            | $|V|$  | $|E|$ | $\gamma$ | $\chi$ | #back | CPU   |
|-----------------|------|------|--------|--------|-------|-------|
| scheduling      |      |      |        |        |       |       |
| school1_nsh     | 352  | 14612| 14     | 14     | 11    | 0.25  |
| school1          | 385  | 19095| 14     | 14     | 12    | 0.41  |
| register allocation |    |      |        |        |       |       |
| musol.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  |
| sgenom.i.1      | 439  | 8458 | 55     | 55     | 0     | 0.17  |
| fisol2.i.1      | 496  | 11654| 65     | 65     | 4     | 0.27  |
| slabr2.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  |
| slacon.i.1      | 2337 | 71600| 73     | 73     | 2     | 4.93  |
| midat.i.1       | 2408 | 114388| 136  | 136    | 4     | 15.4  |
| deseco.i.1      | 2826 | 86688| 117   | 117    | 3     | 12.4  |
| bld.i.1         | 3072 | 224151| 171  | 171    | 3     | 19.8  |
| twldr.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            | 21   | 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  |

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 these 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, and freeing the memory.