Solving 2-List Coloring (by Reducing to 2-SAT)

Santiago Gil

March 2016

1 Introduction

The 2-List Coloring problem consists in finding a coloring of a graph $G = (V, E)$ where each node $v \in V(G)$ has a set $C(v)$ of possible colors assigned to it such that $|C(v)| = 2$. That is, each node has a list of two colors that we can use in a coloring.

In order to solve this problem, we will introduce the 2-SAT problem, to which we can reduce 2-List Coloring. We can then solve an instance of 2-SAT in polynomial time and obtain an answer that is equivalent to the answer of our original problem.

As we know, the Coloring problem is NP-Complete. However, when we restrict the possible colors that we can use to a list of two colors per node, there is a polynomial time solution.

2 2-SAT Problem

The general SAT problem consists in determining if a Boolean formula with restrictions on its variables can be assigned some set of values such that it evaluates to *true*. Equivalently, it is finding out whether the formula isn't a *contingency*—a formula that always evaluates to *false*. (In that case, it could make sense to replace the whole formula with a constant.)

The 2-SAT problem is a special case of SAT where each clause contains only two terms. For example, an instance of 2-SAT could be: $(a \vee b) \wedge (c \vee d) \wedge (e \vee f)$; or $(a \Rightarrow b) \wedge (c \Rightarrow d)$.

2.1 Complexity

While the SAT problem is NP-Complete, there are several algorithms that solve 2-SAT in polynomial time. One of those algorithms is **Aspvall, Plass and Tarjan's algorithm**.

2.2 Aspvall, Plass and Tarjan's Algorithm

This algorithm [1] works by finding the **strongly connected components** of a digraph that is constructed from the input formula. *Strongly* connected components are digraphs' counterpart of connected components, with the added requirement that nodes be reachable in both directions. That means that for any pair of nodes u and v in a strongly connected component there is a path $u \to v$ and another path $v \to u$.

The digraph that we construct from the input formula is called an **implication graph** and it consists of one node for each term (a variable and its negation are different nodes), and one edge for each implication (pointing in that same direction). So, for example, if the formula includes the clause $(a \Rightarrow \neg b)$, the digraph G has $\{a, \neg b\} \in V(G)$ and $(a, \neg b) \in E(G)$.

If a variable and its negation are in the same connected component there is a cycle of implications that includes them. That would imply that $x \Rightarrow \neg x$ and $\neg x \Rightarrow x$. In that case there is no possible assigment that satisfies the formula, because we cannot assign x and $\neg x$ the same value.

In order to find those strongly connected components, APT's algorithm uses another algorithm: **Kosaraju's algorithm**.

2.2.1 Kosaraju's Algorithm

This algorithm has two stages. It starts by marking all nodes as unvisited and creates an empty stack. Then it starts exploring the digraph using DFS (Depth-First Search) from unvisited nodes, adding the new-found nodes to the stack. It starts searches until all nodes in the graph are in the stack.

For the second stage, it inverts all the arcs in the digraph and marks all nodes as unvisited. Then, while the stack is not empty, it takes the first unvisited node from it and starts a DF search from that node (marking all reachable nodes as visited). The set of nodes obtained by each DF search is a strongly connected component of the digraph.

2.2.1.1 Complexity

Kosaraju's algorithm runs in $O(n+m)$ time if the digraph is represented using an adjacency list.

3 Reducing 2-List Coloring to 2-SAT

3.1 Constructing a Boolean Formula

Given a node $v \in V(G)$, let $c(v) = \{c_0, c_1\}$. For simplicity, we will assume, without loss of generality, that colors are ordered increasingly in each list.

We will now study what could happen with any neighbour w of v and its colors.

i. $c(v) \cap c(w) = \emptyset$

   In this case there are no clashes between the colors that we can choose for v and w. Any combination we select is valid.

ii. $c(v) \cap c(w) = \{c_0\}$

   If we color v with c_0, we cannot do the same with w. In all other cases there are no clashes.

iii. $c(v) \cap c(w) = \{c_1\}$

   Similarly to the previous item, if we assign c_1 to v then w needs a different color.

iv. $c(v) \cap c(w) = \{c_0, c_1\}$

   This is the most restrictive case. It forces us to assign alternating colors to v and w.

We can then state this terms in Boolean clauses.

Let X_i be the variable that represents node v_i of G and its coloring. We define the value of X_i to represent the way v_i is colored: If $X_i =$ false, v_i is colored with the first color in its list; otherwise, if $X_i =$ true, v_i is assigned its second color.

Taking those variables we construct the following terms for each case:

i. true

ii. $(\neg X_v \Rightarrow X_w)$

iii. $(X_v \Rightarrow \neg X_w)$

iv. $(\neg X_v \Rightarrow X_w) \wedge (X_v \Rightarrow \neg X_w)$

3.2 Associated Digraph

From this formula, we construct a digraph H where each term is a node and each arc follows the implications between them. Then we find its strongly connected components using APT's algorithm. As we stated before, if a variable and its negation are in the same strongly connected component, the formula is not satisfiable—and that means there isn't a possible coloring of the original graph G.

However, what if we determine that the formula is satisfiable? That would guarantee that there is *some* 2 list-coloring of G. But we are looking for an actual 2-list coloring, so we need to determine how to assign colors to nodes in $V(G)$ from the variables in this formula.

3.3 Constructing the Coloring

In order to do that, we take advantage of a property of Kosaraju's algorithm: the strongly connected components are returned in topological order. That is, if a node u is in the i-th strongly connected component, v is in the j-th one, with $i < j$, and we explore the digraph in topological order, u appears before v.

That tells us that, if we take those u and v, the original formula can have the implication $u \Rightarrow v$, but not in the other direction—if not they wouldn't be in separate components.

Following the truth table for the implication operation, we want to avoid the case where *true* $\Rightarrow$ *false*; the only case where the implication evaluates to *false*.

Therefore, if we explore strongly connected components in order, and we assign each source-node's term a value such that it evaluates to *false*, we avoid implications of the form *true* $\Rightarrow$ *false*, and always obtain clauses that evaluate to *true* in the formula.

Translating back from X_i values to the actual coloring, we select v's color as follows:

- If the strongly connected component that contains the node representing variable X_i is returned before $\neg X_i$, then $X_i :=$ *false*, and v is colored with its first color c_0.
- Otherwise, if the variable X_i comes after $\neg X_i$, v is colored with its second choice of color c_1.

In both cases, assuming that the arcs between terms exist in H, we obtain the implication (*false* $\Rightarrow$ *true*) $\equiv$ *true*.

4 Complexity

Kosaraju's algorithm, used in the second part of the process, runs in $O(n+m)$. Since the rest of the operations we do to construct the digraph are linear in the number of nodes and edges in the original graph, the total complexity is the same.

5 Recap

So, in order to solve 2-List Coloring by reducing it to 2-SAT we do the following:

1. Translate the relationship between pairs of connected nodes in G and their colors into implications of two terms.

2. Take all those terms and combine them into a formula F.
3. From F construct a digraph H where each node represents a term and each edge is an implication (pointing in the same direction).
4. Use APT's algorithm to find the strongly connected components of H.
5. Looking at each node v of H and its neighbors w, if SCC(v) = SCC(w), then stop: there is no possible coloring. If SCC(v) < SCC(w), color v with c_0 and w with c_1. If not, color v with c_1 and w with c_0.

References

[1] Bengt Aspvall, Michael F. Plass and Robert Endre Tarjan, *A linear-time algorithm for testing the truth of certain quantified boolean formulas*, 1979.