CSE 101 Class Notes
Three Coloring, Greedy Algorithms

October 30, 2006

Backtracking: Graph coloring

Backtracking is a search and optimization procedure where at each step we consider a number of possible choices, and for each one *recursively* solve a smaller version of the original problem. If the subproblems are *not* similar to the original, we may still be able to restate the problem to give it this similarity, usually by generalizing our original problem statement. Map coloring, our final example of a backtracking algorithm, is one such problem.

Given a map, we want to find a coloring using 3 colors such that no two adjacent countries share a color. That is, given a graph $G = (V, E)$ and a set of labels $\{R, G, B\}$, can we guarantee that $(v, w) \in E \implies \text{color}(v) \neq \text{color}(w)$?

This can be formulated as a backtracking problem where we choose a color for some node, then look for a solution for the rest of the graph. However, this subgraph will contain some already-colored nodes. Therefore, we generalize our original problem statement to include a partial color assignment as well as a graph: given a graph $G = (V, E)$, a set of labels $\{R, G, B\}$, and a partial labeling $C : V \to \{R, G, B, nil\}$, can we guarantee that $(v, w) \in E \implies \text{color}(v) \neq \text{color}(w)$ and either $C(v) = \text{color}(v)$ or $C(v) = nil$?

This leads to the following backtracking algorithm:

```
C <- { array of nil for each node of G }
BT3Color(G, C)
    if C has no nil's
        return true
    if (there is some x with 3 neighbors (a,b,c)
            and C[a] = R, C[b] = G, C[c] = B)
        return false
    if (there is some uncolored x with 2 colored neighbors (a,b)
            and C[a] != C[b])
        C[x] <- { R,G,B } - { C[a], C[b] }
        return BT3Color(G, C)
    if (there is some uncolored x with 1 colored neighbor a)
        for each color k such that k != C[a]
            C' <- C
```

```
15              C'[x] <- k
16              if BT3Color(G, C')
17                  return true
18          return false
19      x <- any node in G
20      C[x] <- R
21      return BT3Color(G, C)
```

Running time

Exhaustive search would try every color for every node, for a cost of $O(3^n)$. In our backtracking solution, the case at lines 12-18 contains the most recursive calls. Each of these calls has size $n - 1$, so the running time is

$$T(n) = 2T(n-1) = O(2^n)$$

an improvement comparable to those we saw last time (e.g. 5x faster for size 4, 656x faster for size 16, 431,000x for size 32).