CSE 101 Class Notes

Backtracking

May 13, 2009

Backtracking is a generic method that can be applied to many problems. Finding a backtracking algorithm can often be a first step towards finding a greedy or dynamic-programming algorithm.

However, backtracking does not usually result in optimal algorithms or dramatic running-time improvements over the naive approach. Furthermore, exact time analysis can be difficult, since it is difficult to exactly determine the range of the search.

Examples of backtracking include depth-first search, branch-and-bound, and other similar approaches that perform a search by exhaustive case analysis.

Example: Maximal Independent Set

Given a graph with nodes representing people, and edges representing enmities, find the largest set of people having no mutual animosity, i.e. the largest set of unconnected nodes.

Greedy approach

While some people remain, choose the person with the fewest enemies.

```
S <- { all nodes of G }
P <- { }
while S not empty
    s <- node in S with smallest degree
    add(P, s)
    S <- S - s - adj(s)
```

This is fast. However, it doesn't always find the be best solution, because it is possible to back oneself into an algorithmic corner by making a choice that prevents you from making better choices later.

Exhaustive search

Make up a list of every possible subset of people, and choose the largest compatible subset. This will find the solution, but requires exponential time ($O(2^n)$).

Backtracking approach

This exhaustive approach is wasteful, though: by choosing one person, we eliminate all of its neighbors at once – i.e. $2^{|\mathrm{adj}(s)|}$ rows of the exhaustive truth table can be handled equivalently. The backtracking approach systematizes this process of exhaustive, sequential choice as a tree of decisions.

More precisely, let $G = (V, E)$ be an undirected graph. We want to find a set $I \subset V$ such that if $(u, v) \in E$, then either $u \notin I$ or $v \notin I$. Call this I a *maximal independent set* for G (note that I may not be unique). The backtracking approach finds I with the following algorithm:

```
1     MIS(G = (V, E))
2         if V = NIL
3             return NIL
4         if |V| = 1
5             return V
6         for x in V
7             S_in <- union(MIS(G - x - adj(x)), x)
8             S_out <- MIS(G - x)
9             return largest(S_in, S_out)
```

This takes $T(n) \leq 2T(n-1) + O(1)$, since each of the two recursive calls has size at most $n - 1$. We can't apply the master theorem (*why?*), but by noting that in the worst-case the call tree forms a complete binary tree of depth n, `MIS` is $O(2^n)$.

While this is not better than the exhaustive approach above, a more careful implementation can yield a win by handling a special case: if x has degree 0, then it should always be included in I, so we can skip the recursive call on line 8. So the recurrence becomes $T(n) \leq T(n-1) + T(n-2) + O(1)$ (hm... Fibonacci strikes again), i.e. $T(n) = O(F_n) = O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) \approx O(2^{0.7n})$. We can prove the magic number by guess-and-check, guessing that $T(n)$ is exponential in n:

$$\begin{aligned} T(n) &= \omega^n \\ &= T(n-1) + T(n-2) = \omega^{n-1} + \omega^{n-2} \\ &\Longrightarrow \omega^2 - \omega - 1 = 0 \\ &\Longrightarrow \omega = \frac{1+\sqrt{5}}{2} \end{aligned}$$

Aside: DES Challenge

To get some idea how much of an improvement $2^{0.7n}$ is over 2^n, the DES challenge cracked a 56-bit key by exhaustive search over 2^56 using 1000 computers and 3 months. At this rate, breaking an 80-bit key takes 2^80 steps (56/0.7), or 16 million times more resources. You might as well just set your computers on fire. In the other direction, 2^40 $(56 * 0.7)$ would take only a few hours.

In other words, those constants in the exponent matter! Can we find other heuristics to improve the constant? The current worst-case graph is a straight line of linked nodes. In this case, we really only need to consider either the even or the odd elements in the "list".

Generalizing this idea, if x has only a single neighbor y in G, then we can return the MIS for $G - \{x\}$, which must be of the same size as `union(MIS(G - x - y ), x)`. This reduces the running time to $O(2^{0.6n})$. However, we don't have an instance of a graph exhibiting this worst-case behavior, so we might be able to do better.