Backtracking: Largest maximal independent set

A simple example of average-case analysis

A simple example of a *backtracking algorithm* is the *N-Queens problem* in recreational mathematics. Here, we look at another problem:

PROBLEM: Find a largest *maximal independent set* (MIS) of a given simple connected undirected graph G.

The above problem is known to be hard in the sense that it is believed that no polynomial-time solution exists. The backtracking algorithm given here following [1]

- will always find a solution - this follows directly from the algorithm.
- will run fast *on the average* - the analysis is surprisingly simple.

Working of the backtracking algorithm:

Let the input graph be $G = (V, E)$, where $V = \{v_1, \cdots, v_6\}$ and $E = \{v_1v_2, v_1v_4, v_1v_5, v_2v_3, v_2v_6, v_3v_6, v_4v_6, v_5v_6\}$.

Denote by S an MIS. Initialize $S = \phi$. In the algorithm, we always attempt to enlarge S by ading the next possible (available) higher sub-scripted vertex. Thus, after initialization, we do $S = S \cup \{v_1\}$. With $S = \{v_1\}$, we cannot add v_2 because $v_1v_2 \in E(G)$; we try v_3, which can be added to S as $v_1v_3 \notin E(G)$. Now $S = \{v_1, v_3\}$. We next try v_4; it cannot be added to S as $v_1v_4 \in E(G)$.

We next consider v_5 and then v_6 both of which cannot be added because $v_1v_5 \in E(G)$ and $v_2v_6 \in E(G)$. We are now struck and therefore we *backtrack* – that is, we remove from S, the latest addition and we try the other choices. In the example, we remove v_3 from S making $S = \{v_1\}$. The next two choices v_4 and v_5 cannot be added to S. We then add v_6 and get $S = \{v_1, v_6\}$ which is an MIS as there are no further choices. Since we are again at a dead end, we backtrack, now removing v_6 as well as v_1. We next add v_2 and now we have $S = \{v_2\}$. This gives $S = \{v_2, v_4, v_5\}$. The complete list of $MISs$ that arise from the start to the finish of the algorithm is

$$\{v_1\},\ \{v_1, v_3\},\ \{v_1, v_6\},\ \{v_2\},\ \{v_2, v_4\},\ \{v_2, v_4, v_5\},\ \{v_2, v_5\},$$
$$\{v_3\},\ \{v_3, v_4\},\ \{v_3, v_4, v_5\},\ \{v_3, v_5\},\ \{v_4\},\ \{v_4, v_5\},\ \{v_5\},\ \{v_6\}.$$

Backtrack search tree T:

T is one way to represent the search process.

T is a rooted tree with levels $L = 0, 1, 2, \cdots, n$, where $|V| = n$.

Each node in $T \equiv$ an independent set in G.

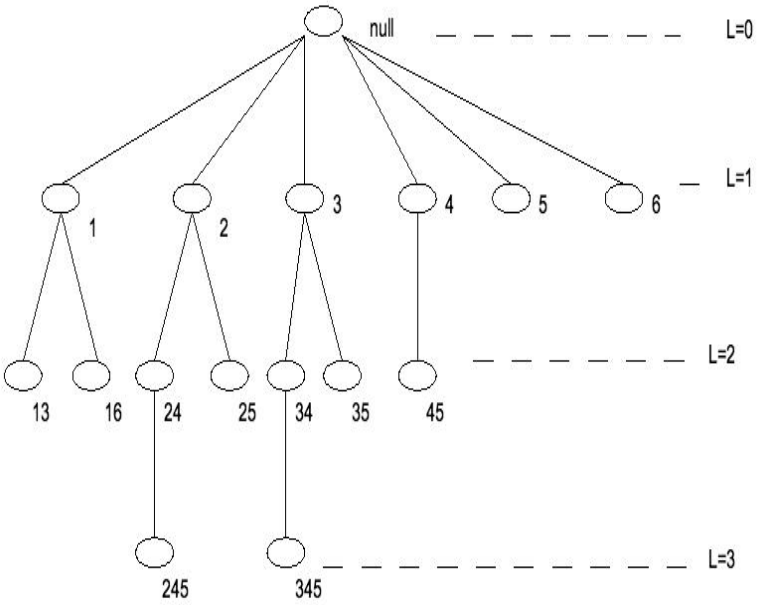
In T, the root corresponds to the *null* set.

Let S' and S'' be two nodes of T connected by an edge. Then $S' \subset S''$ and $S'' - S'$ consists of a single vertex of G – the highest numbered vertex in S''.

On level L in T, we find nodes of T corresponding to every independent set with exactly L vertices of G. At level 0 is the null set ϕ.

The backtracking algorithm amounts to just visiting every vertex of T without having to build T in advance.

The next page depicts T for the example graph.



Backtrack search tree T for the example graph G
Each node in T represents an independent set

Complexity of the algorithm:

The list given above comprise the nodes of T – includes every independent set of G. The search will go through all these nodes of T.

Therefore a reasonable measure of complexity of the search is the number of independent sets of G.

Thus

Finding complexity $\equiv$ Counting the number of independent sets of G.

One extreme: the graph $\overline{K}_n$ with 2^n independent sets - exponentially long search.

Other extreme: the graph K_n with just $n+1$ independent sets - fast search. In general, a random graph G will have a number of independent sets between $n+1$ and 2^n.

We ask: how fast is the backtracking, *on the average*?

Thus our new problem is: what is the average number I_n of independent sets that a graph on n vertices can have?

The answer is given by

$$I_n = \sum_{S \subseteq \{1,\cdots,n\}} Prob\,(S \text{ is an independent set}). \tag{1}$$

If $|S| = k$, then

$$\text{Prob}(\ S \text{ is an independent set}) = \text{Probability that among the } \binom{k}{2} \text{ potentially possible edges in } S, \text{ none is actually present.} \tag{2}$$

Note: each of the $\binom{k}{2}$ edges has a probability of $\frac{1}{2}$ of appearing in S.

Therefore

$$\text{Probability that no edge is present in } S = \left(\frac{1}{2}\right)^{k(k-1)/2} \tag{3}$$

Using (3) and (2) and combining with (1), we get

$$I_n \;=\; \sum_{k=0}^{n} \binom{n}{k} 2^{-k(k-1)/2} \tag{4}$$

We can quickly build the following table using (4):

n	2	3	4	5	10	20
I_n	3.5	5.6	8.5	12.3	52.0	350.6
2^n	4	8	16	32	1024	1048576

It follows that:

Average number of independent sets << Maximum number possible.

The backtracking algorithm is subexponential, but is slower than polynomial-time on the average.

Exercise 1: Write a C or C++ code implementing the backtracking algorithm.

Exercise 2: Starting with (4), show that: as n grows large, rate of growth of I_n is in $O(n^{log\, n})$.

References

[1] H. S. Wilf, *Algorithms and complexity*, A.K.Peters, Natick, MA, 2002