Title

Computer Science 604

Advanced Algorithms

Lecture 8: Dealing with NP-completeness, cont'd

David Juedes

School of EECS

juedes@cs.ohiou.edu

Remarks

As you may have noticed already, the $O(1.47^n)$ upper bound on the running time of the recursive algorithm for Minimum Vertex Cover also applies to the branch-and-bound version. Hence, there are (think about it) input instances that cause both algorithms to perform poorly on large instances.

But, in practice, branch-and-bound performs relatively well on many instances, and hence is a good start for exploring ways of coping with NP-completeness.

In general, we'll examine what approaches will work, and try to find out what approaches will not.

Let's examine some other ways.

Strong NP-completeness

Recall our discussion concerning pseudo polynomial-time algorithms. Given an instance I, $length[I]$ denotes the length of the instance, and max$[I]$ denotes the value of the maximum integer value in the instance.

Let Π be a decision problem, and let p be a polynomial. We define the problem Π_p to be the problem Π restricted to all instances where $\max[I] \leq p(length[I])$.

Notice that if Π is solvable by a pseudo-polynomial time algorithm, then Π_p is solvable in polynomial-time.

Strongly NP-complete

A problem Π is NP-complete in the **strong sense** (or strongly NP-complete) if there is a polynomial such that Π_p is NP-complete.

Graph coloring is NP-complete in the strong sense because determining whether or not a graph is 3-colorable is NP-complete.

Vertex Cover **is strongly NP-complete**

Notice that since the only integer in the vertex cover problem (other than the vertex indices) is the integer k (we want to know whether G has a vertex cover of size k), that $VertexCover_n$ is NP-complete (bound k by n). Hence, Vertex Cover is NP-complete in the strong sense.

Hence, we cannot find a pseudo polynomial-time algorithm for Vertex Cover unless P=NP.

Other Approaches that Might Work

For the decision version of Vertex Cover, we can exploit certain "properties" of the input instance to make our algorithms run faster.

2 dimensional complexity theory

Instead of measuring the running-time of algorithms for Vertex Cover in terms of the size of the input (# of vertices, # of edges in G, etc.), we are going to measure the running-time of algorithms for these problems in terms of the size of the input (G) and the “parameter” k.

When k is small, we can get a polynomial-time algorithm for Vertex Cover that depends on k.

The Bounded Search Tree Approach

We will use the following observations to build a fast algorithm for Vertex Cover.

Observation 1: If $e = (x, y)$ is an edge in G, then either x or y (or both) must be in any valid vertex cover of G.

Observation 2: Let $e = (x, y)$ be an edge in G. Then, G has a vertex cover of size k iff $G - \{x\}$ has a vertex cover of size $k - 1$ or $G - \{y\}$ has a vertex cover of size $k - 1$.

These observations lead to the following recursive algorithm for Vertex Cover.

A Recursive Algorithm for Vertex Cover

```
Algorithm Check(G,k)
if k < 0 then
   return FALSE;
end if
COMMENT: k ≥ 0
if G has no edges then
   return TRUE;
else
   if k = 0 then
      return FALSE;
   else
      pick an edge e = (x, y) ∈ E;
      if CHECK(G − {x}, k − 1) or CHECK(G − {y}, k − 1) then
         return TRUE;
      else
         return FALSE;
      end if
   end if
end if
```

What is the running time of CHECK?

2 Dimensional Recurrence Relations

Since Check has two parameters, it is natural to analyze the running-time of check via a 2 dimensional recurrence relation. Let $T(n, k)$ be an upper bound on the running-time of Check on graphs G of size n and parameter of value k.

Then, what is

$$T(n, 0)?$$

The base case, $k = 0$

When $k = 0$, the algorithm never makes any recursive call. Hence, $T(n, 0) =$ time needed to determine whether G has no edges. Conservatively, this takes $O(n)$ steps using the adjacency list representation of G and $O(n^2)$ steps if we represent G as an adjacency matrix.

Hence, $T(n, 0) \leq O(n^2)$.

The general case

What is $T(n, k)$ in terms of T with smaller values of n and k?

Notice that if $k \neq 0$, then Check makes 2 recursive calls.

Hence, $T(n, k) =$?

The general case, cont'd

In the general case,

$$T(n, k) = 2T(n - 1, k - 1) + cn^2).$$

Why?

If we assume that $n \geq k$, then the solution to $T(n, k) = O(2^k n^2)$. We can show this via the iteration method, since

$$T(n, k) \leq \sum_{j=0}^{i-1} 2^j \cdot c(n - j)^2 + 2^i T(n - i, k - i).$$

This iteration stops when we hit the base case $i = k$.

Cont'd

Hence,

$$
\begin{aligned}
T(n,k) &\le \sum_{j=0}^{k-1} 2^j \cdot c(n-j)^2 + 2^k n^2 \\
&\le cn^2 \sum_{j=0}^{k-1} 2^j + 2^k n^2 \\
&\le 2^k cn^2 + 2^k n^2 \\
&= O(2^k n^2).
\end{aligned}
$$

Finding the Minimum Vertex Cover

How can we modify this algorithm to find the Minimum Vertex Cover.

How long does it take?

When is the parameterized algorithm faster?

We've seen two exact algorithms for Minimum Vertex Cover.

For which graphs is the guarantee of the parameterized algorithm faster than the original.