Introduction to exact algorithms

Based on handout on parameterised algorithms and handout on exact exponential algorithms

---

Every problem in NP can be solved in exponential time:

Let $L \in NP$ and let $p(k)$ be a polynomial such that $x \in L \iff \exists$ a string $y = y(x)$ of length at most $p(|x|)$ for with $\mathcal{A}(x,y) = 1$, where $A = \mathcal{A}(L)$ is the certificate checking verifier for L.

$y(x)$ is a bitstring of length at most $p(|x|)$

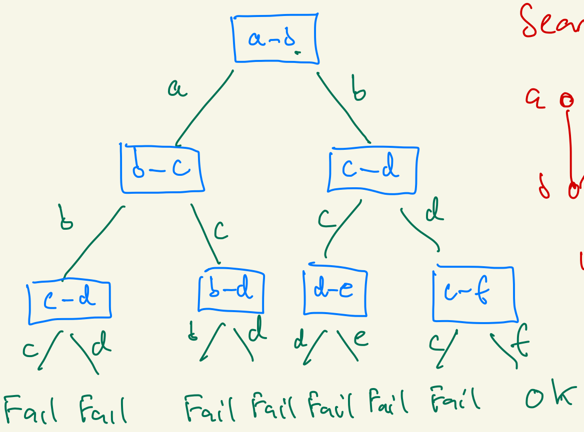
So by checking for at most $2^{p(|x|)}$ bitstrings ω whether $\mathcal{A}(x,\omega) = 1$ we can check whether $x \in L$ in time $O(2^{p(|x|)} |x|^c)$ for some constant c

Here we assume that $\mathcal{A}$ runs in time $|x|^c$

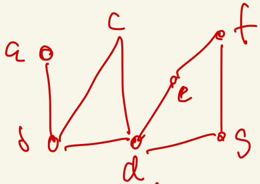
O^* notation : ignore polynomial factor

e.g $O(n^3 \log^2 n \, 3^{n/2}) = O^*(3^{n/2})$

Recall the Tree search that we applied to Vertex.cover



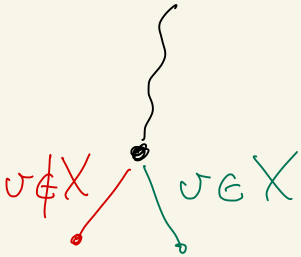
Search tree for

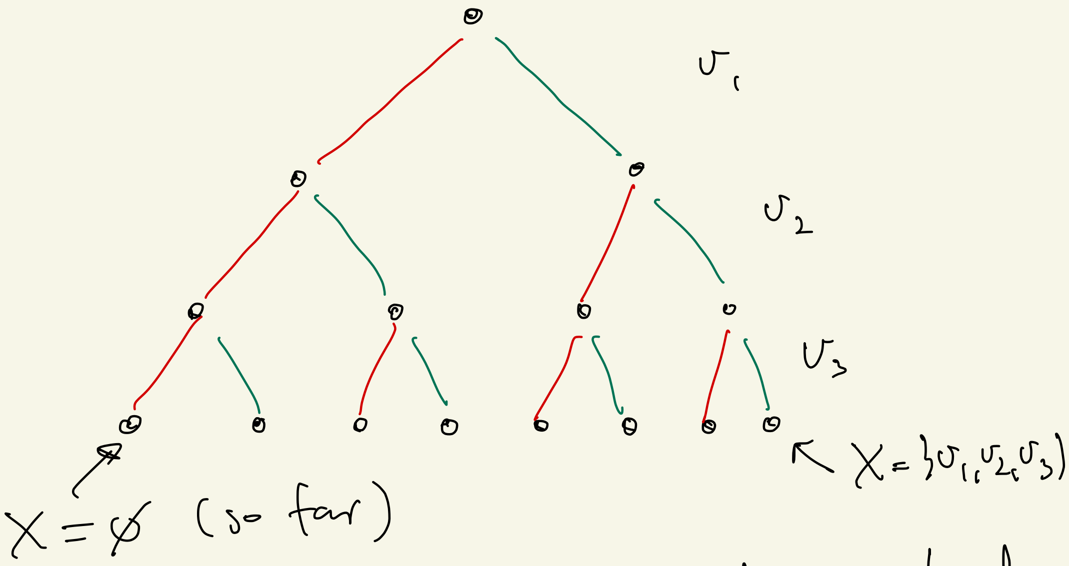


when trying a VC of size 3

We can use the same strategy to find an optimal vertex cover

Now we pick a vertex v not considered yet and branch on whether v is or is not in the VC X that we try





Let G have n vertices and recall that we can bound the work we do by looking at # leaves in the binary search tree:

If this has ℓ leaves then we solve at most $2\ell - 1$ subproblems (# nodes in search tree)

For vertex cover the search tree has depth at most $n = |V(G)|$

In the worst case we must solve all subproblems at the leaves

Let $T(n)$ denote # leaves in the search tree for graphs on n vertices

Clearly $T(n) \leq T(n-1) + T(n-1)$ and $T(1) = 2$

Thus $T(n) \leq 2^n$

How to improve the trivial $O^*(2^n)$ algorithm?

2 approaches

A. traverse the subtree in a clever order and use knowledge about vertexcovers seen so far

B. use problem specific observations to reduce # leaves in search tree

Ad A: Suppose we have already found a VC of size r, then we do not have to take more than r-1 accept steps (green edges in search tree)

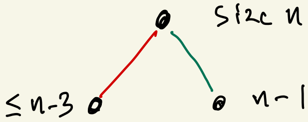
looking for some vertexcover: either use Depth-First-search to build the search tree or use a heuristic such as the 2-approx alg for VC to find a vertexcover Y and let $r = |Y|$

Key ingredient in the general search technique called Branch and bound

can also use a BFS strategy for searching but this requires a lot of space to store subproblems

A & B When we used reduction rules in the FPT algorithm for VC, we found that we could reduce to an instance where all vertices have degree at least 2

Thus when we reject a vertex v (take a red edge in search tree) we must include at least 2 vertices in the vertex cover we are building for that subtree.



This implies that we can set

$T(n) \le T(n-1) + T(n-3)$ and $T(1)=2$

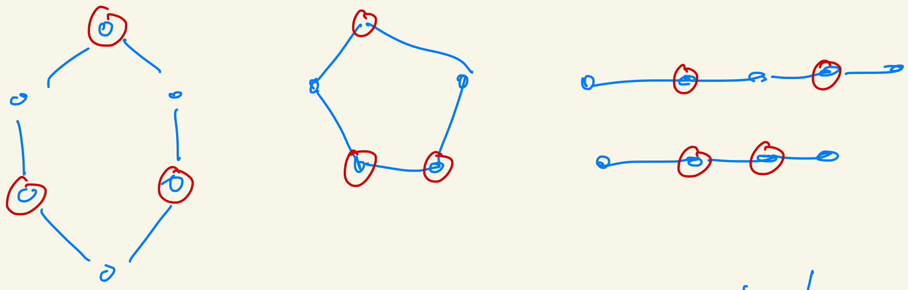
From DM551 you know how to solve $a_n = a_{n-1} + a_{n-3}$ $a_1 = 2$

Characteristic equation $x^3 - x^2 - 1 = 0$

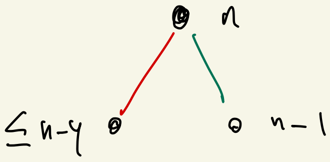
largest real root is less than 1.4656

Hence $T(n) \le 1.4656^n$ implying that we can solve Vertex cover in time $O^*(1.4656^n)$

Lemma We can find a minimum vertex cover in a graph H on n vertices and no vertex of degree larger than 2 in time $O(n^2)$



Hence we don't need to branch in search tree if no vertex in remaining graph has degree >2
⇒ if we reject v we have to include at least 3 vertices in the VC for that subtree



Now $T(n) \le T(n-1) + T(n-4)$

largest real root of $x^4 - x^3 - 1 = 0$ is less than 1.3803

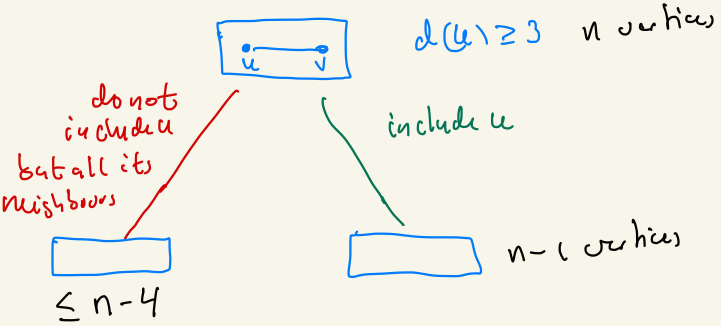
⇒ $T(n) \le 1.3803^n$ and we can solve VC in time $O^*(1.3803^n)$

Back to Vertex cover with parameter k

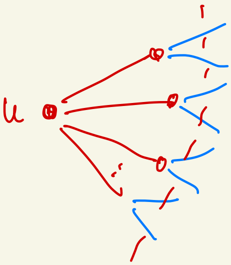
Idea only branch on edge $u - v$ if

$\max\{d(u), d(v)\} \geq 3$

(if no such vertex use the lemma from previous page)



we removed u and all its neighbours



So remaining graph has at least 4 vertices less

We only go to depth k (looking for VC of size $\leq k$)

So $T(k) \leq T(k-1) + T(k-4)$

$\Downarrow$ $T(k) \leq 1.3803^k$ and we can decide $\langle G, k \rangle$ in time $O^*(1.3803^k)$

Solving TSP exactly using dynamic programming

Algorithmic idea due to Bellman, Held and Karp

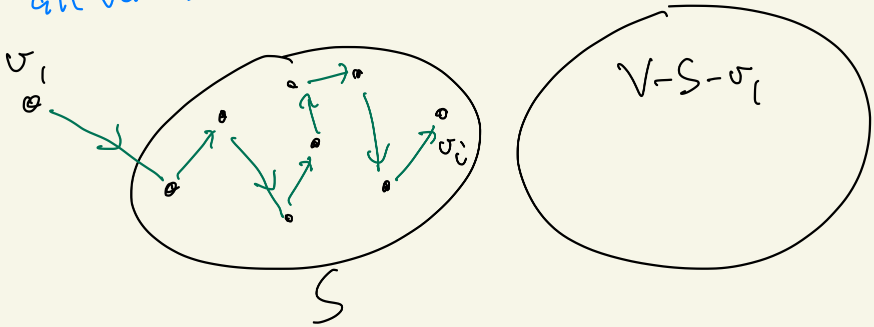
Given n vertices $v_1, v_2, \dots, v_n$ and their distances $d(v_i, v_j)$ for all $i \neq j$

We seek a permutation π of $\{1, 2, \dots, n\}$ such that

$$M = d(v_{\pi(n)}, v_{\pi(1)}) + \sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)}) \qquad (\boxdot)$$

is minimized

Idea: For every subset $S \subseteq \{v_2, v_3, \dots, v_n\}$ and $v_i \in S$ let $OPT[S, v_i]$ be the length of a shortest path that starts in v_1 and then visits all vertices in S and ends in v_i



Then

$$M = \min\{ OPT[\{v_2, v_3, \dots, v_n\}, v_i] + d(v_i, v_1) \mid i \in \{2, 3, \dots, n\} \}$$

How do we compute $OPT[\{v_2, v_3, \ldots v_n\}, v_i]$?

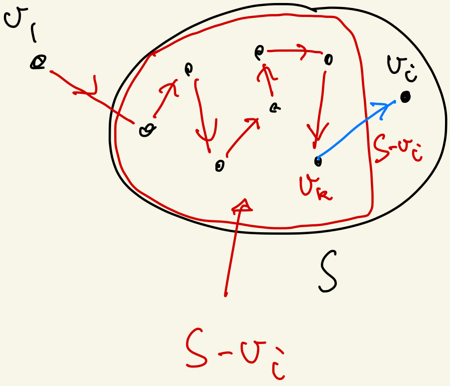
Use dynamic programming:

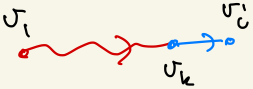
Lemma

$$OPT[S, v_i] = \begin{cases} d(v_1, v_i) & \text{if } S = \{v_i\} \\ \min\{ OPT[S - v_i, v_k] + d(v_k, v_i) \mid v_k \in S - v_i \} & \text{if } \{v_i\} \subset S \end{cases}$$

Proof: If $S = v_i$ it follows from its definition that
$OPT[S, v_i] = d(v_1, v_i)$

so assume $|S| > 1$



If 

is optimal for S, v_i then 

is optimal for $S - v_i, v_k$

Dynamic programming algorithm for TSP

```
1: Function TSP($\{v_1, v_2, \dots v_n\}, d$)
2:     For $i \leftarrow 2$ to $n$ do
3:         $OPT[\{v_i\}, v_i] \leftarrow d(v_1, v_i)$
4:     For $j \leftarrow 2$ to $n-1$ do
5:         For $S \subseteq \{v_2, \dots v_n\}$ with $|S| = j$ do
6:             For $v_i \in S$ do
7:                 $OPT[S, v_i] \leftarrow \min\{ OPT[S - v_i, v_k] + d(v_k, v_i) \mid v_k \in S - v_i \}$
8:     Return $\min\{ OPT[\{v_2, v_3, \dots v_n\}, v_i] + d(v_i, v_1) \mid v_i \in \{v_1, \dots, v_n\} \}$
```

Lemma Function TSP calculates a min cost TSP tour by computing $O(n^2 2^n)$ shortest paths ← (OPT calculations)

Proof: The # of path lengths computed in line 7 is

$$\sum_{j=2}^{n-1} \underbrace{\binom{n-1}{j}}_{\text{\# of } j\text{-subsets of an } (n-1)\text{-set}} \cdot \underbrace{\sum_{i=1}^{j}}_{j \text{ choices for } v_i} \underbrace{(j-1)}_{j-1 \text{ choices for } v_k}$$

using that $\sum_{j=1}^{n} \binom{n}{j} = 2^n$ (binomial formula)

we get

$$\sum_{j=2}^{n-1} \binom{n-1}{j} \cdot \sum_{i=1}^{j} (j-1) \leq n^2 \sum_{j=1}^{n} \binom{n}{j} = n^2 2^n$$

In lines 3 and 8 we compute a total of $2(n-1)$ path length $\square$

Conclusion : We can solve TSP in time

$$O(n^2 2^n) = O^*(2^n)$$

Recall the naive algorithm for TSP

check $(n-1)!$ permutation.

This has running time $O(n!) \sim O(e^{n \ln n})$

as $\ln(n!) \sim n \ln n$

So the dynamic programming algorithm is much better!

FPT versus XP

Definition A parameterized problem Q with parameter k is slicewise polynomial (XP) if can be solved in time $O(f(k)\,n^{g(k)})$ for some functions f, g

Note: $Q \in FPT \Rightarrow Q \in XP$ as we can let g be a constant c

Clique is in XP: given $\langle G, k \rangle$ try all $\binom{n}{k}$ subsets when $n = |V(G)|$. Then are $O(n^k)$ k-subsets of an n-set

So clique is solvable in time $O(n^k \cdot k^2)$ check if given k-subsets is a clique

Open: is Clique in FPT?

widely believed that the answer is no!

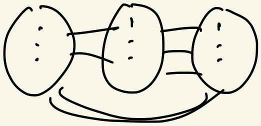
Suppose we parametrize k-clique by the maximum degree Δ of the input graph.

For each vertex $v \in V(G)$ we can check whether v is in a k-clique by checking all the $O(2^\Delta)$ subsets of its neighbours

So when input has maximum degree Δ we can check for a k-clique in time $O(n \cdot 2^\Delta \cdot \Delta^2)$ so k-clique is FPT when parametrized by maximum degree Δ

Colouring is difficult

Recall that already 3-colouring is NPC



So we can conclude:

Lemma Unless P = NP there cannot exist an algorithm for solving k-colouring in time $O(f(k)\, n^{g(k)})$ for general k.

(this would imply that 3-colouring would be polynomial)