Coping with NP-Hardness

- Strategies to deal with hard problems
- Algorithms for Special Cases
- Fixed Parameter Tractability
- Intelligent Exhaustive Search
  - Backtracking
  - Branch and Bound
- Dynamic Programming based pseudo polynomial algorithm TSP

Imdad ullah Khan

Coping with NP-Hardness

Approaches to tackle hard problems

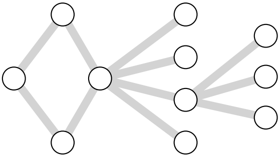
1. Special Cases: Relevant structure on which the problem is easy
   - Exact results in poly-time only for special cases or a range of parameters
2. Intelligent Exhaustive Search: Exponential time in worst case
   - The base and/or exponent are usually smaller
   - could be efficient on typical more realistic instances
   - Backtracking, Brand-and-Bound
3. Nearly exact solutions: Output is 'close' to exact (optimal) solution
   - Approximation Algorithms: Solutions of guaranteed quality in poly-time
   - Heuristic Algorithms: Solutions hopefully good in poly-time
4. Randomized Algorithms: Use coin flips for making decisions
   - Typically used for approximation, also used for easy problems

Fixed-Parameter Tractability

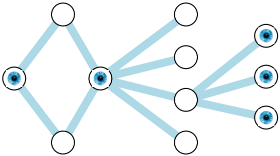
- The solution with approximation guarantees may be too expensive
- Parameterized complexity is a measure of complexity with more than 1 input parameters
- We want algorithms with running time exponential in one parameter but polynomial in the other parameter(s)
  - We would like algorithms with runtimes $2^k n^2$, $k! n \log n$, etc.
  - Acceptable runtimes when in realistic instance k is fixed (small)
- Such problems are called Fixed-Parameter Tractable
- Algorithms are called Fixed-Parameter Tractability (FPT) algorithms

Vertex Cover

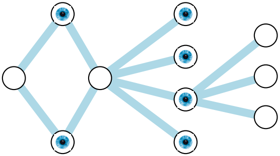
An vertex cover in a graph is subset C of vertices such that each edge has at least one endpoint in C



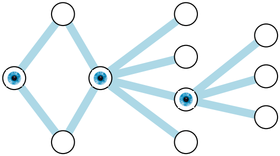
A graph on 11 vertices



A vertex cover of size 5



A vertex cover of size 6



A vertex cover of size 3

The VERTEX-COVER(G, k) problem: Is there a cover of size k in G?

- Focus on SRCH-VERTEX-COVER(G, k)

Brute-Force Algorithm

- For each possible k-subset S of V, check if it is a vertex cover
- Is S a vertex cover?
  - Traverse adj-list of each $v \in S$, count edges in S and in $[S, \overline{S}]$
  - If this count $= |E(G)|$, then S is a vertex cover
- Runtime is $O\left(\binom{n}{k} kn\right) = O\left(kn^{k+1}\right)$ $\triangleright$ polynomial in n for fixed k
- For larger k and large n, this is impractical
  - For $n = 10000, k = 20$, this runitme is $\sim 10^{82}$
- We will design a FPT algorithm with runtime $2^k nk$
  - For $n = 10000, k = 20$, it is $2^{20} \times 10000 \times 20 \ll 10^{82}$

FPT for SRCH-VERTEX-COVER

- Take full advantage of k being small
- Enumerate all possibilities for some k edges
- Pick an edge (u, v)
- For any k-cover S (vertex cover of size k), either $u \in S$ or $v \in S$
- For $x \in V$, $G - \{x\} := (V \setminus \{x\}, E \setminus \{(a, b) \in E : a = x \vee b = x\})$
- $G - \{x\}$: the graph after removing vertex x and edges incident on x

For any edge $(u, v) \in E$,
G has a k-cover if and only if $G - \{u\}$ or $G - \{v\}$ has a $k - 1$-cover

- $\Rightarrow$: If $u \in S$, then $S \setminus \{u\}$ is a $(k - 1)$-cover in $G - \{u\}$
- $\Leftarrow$: A $(k - 1)$-cover S' in $G - \{u\}$ covers all edges except those incident on u. $S' \cup \{u\}$ is a k-cover in G

We use this theorem in an algorithm by recursively trying both possibilities

FPT for SRCH-VERTEX-COVER

Algorithm Algorithm to find vertex cover of size k

function VERTEX-COVER(G, k)
  if $k = 0$ **then**
    if $E(G) = \emptyset$ **then** ▷ $O(n)$ time to check if all adj. lists are empty
      return $\emptyset$
    else
      return NF
  else
    $e = (u, v) \in E(G)$ ▷ Pick an arbitrary edge in G
    $S_u \leftarrow$ VERTEX-COVER($G - \{u\}, k - 1$)
    $S_v \leftarrow$ VERTEX-COVER($G - \{v\}, k - 1$) ▷ $O(n)$ time to make $G - \{x\}$
    if $S_u \neq$ **NF then**
      return $S_u \cup \{u\}$
    else if $S_v \neq$ **NF then**
      return $S_v \cup \{v\}$
    else
      return NF

FPT for SRCH-VERTEX-COVER

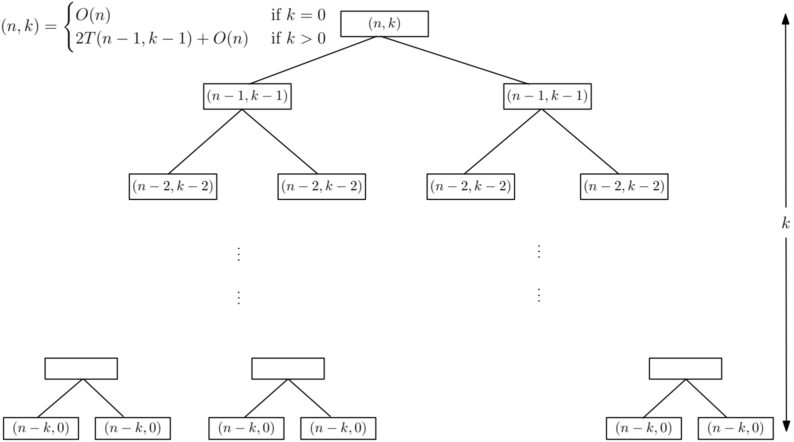
```
function VERTEX-COVER(G, k)
  if k = 0 then
    if E(G) = ∅ then                          ▷ O(n) time to check if all adj. lists are empty
      return ∅
    else
      return NF
  else
    e = (u, v) ∈ E(G)                         ▷ Pick an arbitrary edge in G
    S_u ← VERTEX-COVER(G − {u}, k − 1)
    S_v ← VERTEX-COVER(G − {v}, k − 1)        ▷ O(n) time to make G − {x}
    if S_u ≠ NF then
      return S_u ∪ {u}
    else if S_v ≠ NF then
      return S_v ∪ {v}
    else
      return NF
```

$T(n, k)$: runtime of this algorithm on input G and k

$$T(n,k) = \begin{cases} O(n) & \text{if } k = 0 \\ 2T(n-1,k-1) + O(n) & \text{if } k > 0 \end{cases}$$

FPT for SRCH-VERTEX-COVER

$$T(n,k) = \begin{cases} O(n) & \text{if } k = 0 \\ 2T(n-1,k-1) + O(n) & \text{if } k > 0 \end{cases}$$



- Recursion tree is a complete binary tree with height k
- 2^k leaves and 2^{k-1} internal nodes (recursive invocation)
- Runtime of each recursive invocation is at most $O(n)$

$T(n,k) = O(2^k n)$