PTT algos

Want to solve NP-hard problems faster

Eg. vertex cover: delete min # vts s.t. no edges remaining

Idea 1: Let's approximate soln.

- 2-apx soln in polytime
- 2 is tight assuming UGC

Idea 2: Let's restrict the input somehow.

- Disallow "lower bound instances"

This lecture: restrict input by parameterization

- Restrict to instances where \( \text{OPT} \leq k \)

Vertex cover parameterized by \( \text{OPT} \)

Input: (unweighted) graph \( G \)

Output: If \( \text{OPT} \leq k \), output \( \text{OPT} \)

Else, output \( 1 \)

**Def:** Parameterized problem

Input: \( (I, k) \) where \( I \) is an instance and \( k \in \mathbb{N} \) is a parameter

Output: (specified by problem)

Runtime: \( f(k)n^c \) for absolute fn \( f: \mathbb{N} \rightarrow \mathbb{N} \)

Constant \( c \in \mathbb{N} \)

"Exponential in \( k\), poly in \( n\)"

"Capture the hardness of the problem into a param \( k\)"

1. Parameterized vertex cover:
   - We'll see: 1. \( 2^k \ polym(n) \)
   - \( \text{poly}(n) + 2^k \ polym(k) \)

2. \( k \)-path
   - Input: \( (G, k) \)
   - Output: a simple path of length \( k \) in \( G \)
   - (or 1 if none exist)

Runtime:
   - 1. \( k \ polym(n) - k^k \ polym(n) \)
   - 2. \( c^k \ polym(n) \)

3. Feedback vertex set: delete min # vts to make graph acyclic

\[ \begin{array}{ccc}
\text{OPT} = 1 & \text{OPT} = 1 & \text{OPT} = 3 \\
\end{array} \]

Input: \( (G, k) \)

Output: if \( \text{OPT} \leq k \), then output \( \text{OPT} \)
elx, output 1
Runtime: \(4^k \text{poly}(n)\)

2. k-path
- Brute-force: \(n^k\) (not FPT!)
- Idea: Randomization!
- Alg:
  - Every vrtx gets random label in \([k]\)
  - We want:
    - \(1 \rightarrow 2 \rightarrow 3 \rightarrow \cdots \rightarrow k\)
    - Prob that fixed k-path looks like this is \(\frac{1}{k! k^k}\)
  - Given labeling, find longest path w/ increasing labels
  - Can be done using DP
  - One easy way to look at it:
    - Direct the edges and look for
      \(1 \rightarrow 2 \rightarrow 3 \rightarrow \cdots \rightarrow k\)
    - For each edge in \(G\), direct it to higher labeled vertex:
      \(k \rightarrow k' \Leftrightarrow l_k < l_{k'}\)
      - If \(l_k = l_{k'}\), then remove edge.
      - Graph is acyclic (very important!)
      - Find longest path in DAG [DP]
  - Runtime: \(\text{poly}(n)\)
  - Success prob: \(\frac{1}{k!}\)
  - Repeat \(\Theta(k! \log n)\) times
    - \(\Rightarrow\) Fail prob \(\leq (1 - \frac{1}{k!})^{\frac{1}{2}n \log n} \leq e^{-\frac{1}{2}n \log n} = \frac{1}{n^k}\).

Main insight: By using randomization, we were able to ensure that EVERY size-\(k\) subset of vrtxs is “rainbow-colored” with prob \(\frac{1}{k!}\). Hard to ensure deterministically: there are \(n^k\) such subsets!

Speed up? Key idea: Requiring sorted order on k-path was too strict. Let’s relax to:
- We want:
  \[2 \rightarrow 6 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 4 \quad \text{K=6}\]
  - All vertices on fixed k-path have unique labels
    - Prob is \(\frac{k!}{k^k} \approx \frac{n^{\log(k)} e^k}{k^k} = 2 \left(\frac{1}{e}\right)^k\).
  - Task: given labeled graph, find a k-path w/ all vrtxs different labels
    - Higher success prob, but harder
Algorithmic task
- We'll show $2^k \text{poly}(n)$ time
  \[ \Rightarrow \text{repeat } \Theta(2^k \text{poly}(n)) \text{ times} \]
  \[ \Rightarrow (2e)^k \text{poly}(n) \text{ time}. \]

- "Subset DP"

\[ \text{DP}(v, I) : v \in V, I \subseteq [k] \]

\[ \text{ending at } v \]

- "a path whose vertices have labels in } I, \]
  - one of each (in particular path has length } |I|, \]
  - or $\perp$ if none exist"

Final answer: find a $v$ s.t. $\text{DP}(v, [k]) \neq \perp$
- else, output $\perp$

\[ \begin{align*}
  u & \text{ 2 } \text{DP}(u, \{2,3\}) = (u) \\
  v & \text{ 3 } \text{DP}(v, \{3,6\}) = (u, v) \\
  w & \text{ 4 } \text{DP}(v, \{2,3,6\}) = (u,v,w) \\
  x & \text{ 4 } \text{DP}(v, \{6\}) = (u,v,w,x,y,z) \\
  y & \text{ 4 } \text{DP}(v, \{6\}) = (u,v,w,x,y,z) \\
  z & \text{ 4 } \text{DP}(v, \{6\}) = (u,v,w,x,y,z) \\
\end{align*} \]

Initialize: $\text{DP}(v, \{v\}) = (v)$

Recursion $\text{DP}(v, S)$, $S \subseteq S$:
- If $\exists$ neighbor $u$ s.t. $\text{DP}(u, S \cup \{v\}) \neq \perp$, then set $\text{DP}(v, S) = (\text{DP}(u, S \cup \{v\}), v)$
- Else, $\text{DP}(v, S) = \perp$

- Technique called "color coding" [Alon, Yuster, Zwick '95]

- Further improvements:
  $2^k \text{poly}(n)$ [Williams '09]
  $2^{k/4} \text{poly}(n)$ undirected [Bjorklund '10]
- Also implies $2^{3n/4}$ algo for Hamilton path (first $2^{-\sqrt{k}}$ algo)

Vertex Cover
- Let's solve decision version for simplicity:
  - Input: $(G, k)$
  - Output: Does $G$ have a VC of size $k$?

- Simple randomized algo:
  - For $k$ iter: pick an arbitrary edge, choose random vertex
Prob of success: $\frac{1}{2^k}$

- Repeat $\Theta(\log n)$ times $\Rightarrow$ runtime $2^{\Theta(\log n)}$.

"Derrandomize"? Branching "brute force to bounded depth"

- Pick an arbitrary edge $uv$, branch on whether to clone $u$ or $v$, recurse on each

- **Algorithm VC**: $(G, k)$
  - If $k = 0$:
    - If $G = \emptyset$: return YES
    - Else: return NO
  - $uv \leftarrow$ arbitrary edge in $G$
  - return $V(K(G - u, k - 1)) \lor V(K(G - v, k - 1))$

- **Recursion tree**:

  $$
  \begin{array}{c}
  \text{node} \quad \text{level} \\
  -k & 1 \\
  -k+1 & 2 \\
  \vdots & \vdots \\
  2k & 2^k \\
  \end{array}
  $$

  $\Rightarrow$ runtime $2^k \cdot \text{poly}(n)$. 

**Kernelization**

- Obs: If $\forall v: \deg(v) \geq k + 1$, then greedily
  - choose $v$.

  "Reduction 1": $(G, k)$ is YES $\iff (G - v, k - 1)$ is YES

- Reduction 2: if $\exists$ isolated $v$ in $G$, remove $v$.

- Suppose answer is YES, but neither reduction holds.

  $\leq k \quad \text{OPT (size $\leq k$)}$

  $\Rightarrow$ every vertex not in OPT is in here.

- **Lemma**: If answer is YES and neither reduction holds, then $n \leq k^2 + k$.

- **Algorithm VC-kernel**: $(G, k)$
  - (Red. 1) If $\forall v: \deg(v) \geq k + 1$, return $V(K(G - v, k - 1))$
  - (Red. 2) If $\exists$ isolated, return $V(K(G - v, k))$
  - If $n > k^2 + k$, return NO
  - Return $VC(G, k)$
**DEF (Kernelization)**
A kernelization algo inputs \((G, k)\), runs in poly time, and outputs \((G', k')\) s.t.
- \((G, k)\) is YES \(\iff (G', k')\) is YES
- \(|G| \leq f(k)
- \(k' \leq k\)
- polynomial time kernelization algo if \(f(k) = \text{poly}(k)\)
- VC has a polynomial kernel.

---

**(3) Feedback Vertex set**

**Decision version:**
Input: \((G, k)\)
Output: Can we delete \(k\) vtes in \(G\) to make it acyclic?

1. Randomized branching with reductions
   - Setting: multigraphs (loops & multi-edges)

   - **Reduction 1:** If \(v\) has a loop, then greedily choose \(v\).
     \((G, k) \rightarrow (G-v, k-1)\)

   - **Reduction 2:** If \(uv\) has edge multiplicity \(\geq 3\), then reduce it to \(2\).
     \(u \leftrightarrow v \rightarrow u \leftrightarrow v\)

   - **Reduction 3:** If \(v\) has degree \(\leq 1\), delete \(v\).
     \((G, k) \rightarrow (G-v, k)\)

   - **Reduction 4:** If \(v\) has degree \(= 2\), delete \(v\) and connect its neighbors by an edge.
     \(u \leftrightarrow v \rightarrow u \leftrightarrow w\)

   \((G, k) \rightarrow (G-v, u \leftrightarrow w, k)\)

- Obs: if no more reductions, \(G\) has min deg \(\geq 3\).
  \(\Rightarrow G\) has \(\geq \frac{3}{2}n\) edges

**acyclic**

\(\Rightarrow \#\text{edges} \leq |V-\text{OPT}| - 1\)
\(\leq n - k\)

\(\Rightarrow \text{at least} \left(\frac{3}{2}n - (n-k)\right)\) edges not inside \(G[V-\text{OPT}]\)
\(\geq \frac{1}{2}n \geq \frac{1}{3}|E|\).

- Pick random edge. With prob \(\frac{1}{2}\), pick one adjacent to OPT.
- Pick random endpoint. With prob \(\frac{1}{2}\), pick canonically.
Success prob $\geq \frac{1}{2}$, decrease $k$ by 2

$\Rightarrow$ overall success prob $\geq \frac{1}{6k}$

repeat $\Theta(6^k \log n)$ times.