CSC 791/495

Dr. Kyle Kloster

October 20, 2017

Overview:

Up til now...

- Hard problems. Solving $2^{O(n)}$ in $f(k)\, n^{O(1)}$ (k = some parameter)
- Parameters: natural (size of vc), tw, structural, dual

Today: "Fully polynomial FPT": Solve already poly-time problems... faster! $k^{O(1)}\, n^{O(1)}$

- Convince you...
  - The FPT perspective is already all around you
  - useful for poly-time problems!
- Hot Topic™
  Applying FPT to P is a current hotbed of open problems and new papers.

Warm-up with matrices

Multiply this vector, $\underset{n\times 1}{\underline{x}}$, by this matrix $\begin{bmatrix} 2 & \cdots & 2 \\ \vdots & & \vdots \\ 2 & \cdots & 2 \end{bmatrix} = \overset{n\times n}{A}$ Usually: $O(n^2)$

- $A\underline{x} = \cdots \begin{bmatrix} \bullet \end{bmatrix}$ ↘ sum of $\underline{x}$, multiply by 2 $O(n)$
- more generally ... one distinct element? $\begin{bmatrix} 2 & 3 & 2 & 3 & 3 \\ 2 & 2 & 2 & 3 & 2 \\ 3 & 2 & 2 & 2 & 2 \end{bmatrix}$

↳ $A = 2 \cdot \underline{e}\underline{e}^T$, where "$\underline{e}$" $= \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$ → rank 1 matrix!

For any $A = \underline{u}\underline{v}^T$, $A\underline{x} = (\underline{u}\underline{v}^T)\underline{x} = \underline{u}(\underline{v}^T\underline{x}) = \gamma \cdot \underline{u}$ $O(n)$!

(*) • rank k: A rank k, $A = \overset{n\times k}{C} \cdot \overset{k\times n}{R}$ then $A\underline{x}$ in $O(kn)$

Example: Google's PageRank (initially) required lots of matvecs. (matrix-vector multiplication)
$n \approx 10^9$, so $O(n^2)$ is 10^{18}, no good.

* parameterizing an $O(n^2)$ alg for $O(kn)$

Example 2: Sorting!

- Comparison-based sorting is $\Theta(n \log n)$
- But... $[1, 1, 2, 2, 2, 2, 3, 7, 1, 1, 1]$

↓

If K = # distinct elements that we need to sort...

bucket sort / Radix sort / Counting sort is $O(Kn)$ or $O(K+n)$.

In-Class Exercise

Problem: PNI Pairwise-Neighborhood-Intersections.

- Given G with n nodes, a way to access neighborhood $N(v)$ in time $O(|N(v)|)$
- OUTPUT: size of $N(v_i) \cap N(v_j)$ for all pairs of nodes in G.
- Traditionally: $O(n^3)$
- Using parameter max degree $= K$, find $O(K \cdot n^2)$ algorithm (assume you can access nodes' neighborhood list efficiently)

Answer: For all $O(n^2)$ pairs, checking $N(v_i)$, $N(v_j)$, and $|N(v_i) \cap N(v_j)|$ is $O(K)$.

PNI (continued)

- $O(Kn^2)$:
  - iterate over all n^2 pairs
  - for each pair, $O(K)$ intersect.

- Now try $O(K^2 \cdot n)$ (assume you don't have to compute scores that are zero)

suggestions:

(1) For each node v, get its neighbors' neighbors
- those are all nodes that v has positive intersection with.

(2) For each node, for all pairs of its neighbors, increment their score by 1.

$O(K^2 \cdot n)$ because each node has $O(K)$ neighbors, so all pairs of neighbors means $O(K^2)$ work.

Hot Topic™

The "Start" of P-FPT:

- SODA 2016, Abboud, Williams, Wang: All-Pairs Shortest-Paths (APSP) and related problems (Radius, Diameter)
  - APSP traditionally $O(n^3)$, $O\left(\frac{n^3}{\log(2^{o(n)})}\right)$ is fastest (known)
    - → not $n^{3-\varepsilon}$ for $\varepsilon > 0$, not "truly sub-cubic".
  - Diameter of G is length of longest shortest path u to v for any u, v in G.

↓

Can we solve Diam and Radius faster than APSP?

Well, APSP is $\Omega(n^2)$ in connected graphs...

→ can we solve Diam, Rad in $O(f(k) \cdot n^{2-\varepsilon})$? (for some $\varepsilon > 0$)

(with fully polynomial complexity, $k^{O(1)}$)

Theorem: No Unless "common" complexity hypotheses fail.

But: there do exist FPT (exponential in k) algorithms that are "truly" sub quadratic: $O(2^{k \log k} \cdot n^{2-\epsilon})$ for Diam, Rad.

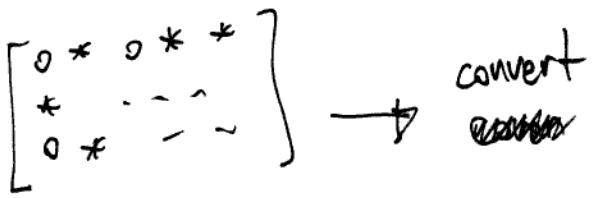
where k = tree-width

→ because tw can be $O(n)$, this is not sub-quadratic.

How about the rest of the field?

- k = tree-width [Fomin et al 2017]
  - compute rank, det of matrix, solve $Ax = b$ } usually $O(n^3)$ $O(n^{2.37})$
    ↳ $O(k^3 \cdot n)$
  - solved maximum matching (was $O(\sqrt{n} \cdot |E|)$) in $O(k^4 \cdot n \cdot \log^2 n)$
- k = tree-or depth [Iwata et al. 2017] * 8 days old *
  - Negative Cycle Detection was $O(n \cdot |E|)$, now $O(k \cdot (|E| + n \log n))$
    ↳ four other problems, maximum matching
- k = clique-width [Coudert et al 2017]
  - triangle-counting $O(M(n))$ in $O(k^2 \cdot (|E| + n))$
    ↳ "complexity of matrix mult" $O(n^{2.37})$ currently

Tree-width of a rectangle?



convert to a graph:

(1) Square: each non-zero entry gives a directed arc. (treat as adjacency matrix).

(2) Each row is a node in partition A
Each column ... in partition B
} make bipartite graph.

How did Fomin et al use tree-width of these graphs to do a $O(k^3 n)$ linear system solve $Ax = b$?

↓

Answer: Use tree-decomposition of the graph associated to A to perform "intelligent" row operations in Gaussian Elimination.

↓

"Perfect Elim Ordering"

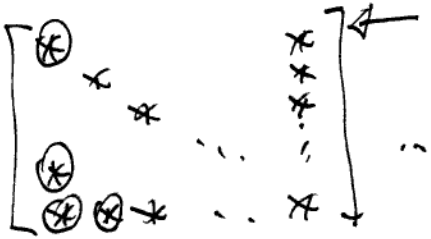
Example of "Perfect Elim Ordering" on matrices

$A =$ 

where "*" = non-zero and space = 0.

"Arrow matrix".

If we did Gauss Elim using first row, we would cause mass fill-in.

If we permute to  .. If we do G. Elim starting with top-row then $O(n)$ operations (!) (use top row to eliminate just the last rows, then the second row to eliminate just the last row, etc)

In Gauss Elim, Perfect Elimination ordering means an order in which you pivot on rows so that elimination does not cause any fill-in.

This turns out to be related to the "elimination ordering" we've discussed for graphs used in constructing a tree decomposition!

Problems

1. Devise an algorithm to solve APSP in sub-cubic time, parameterized by K = path-width (or tree-width, if you prefer)

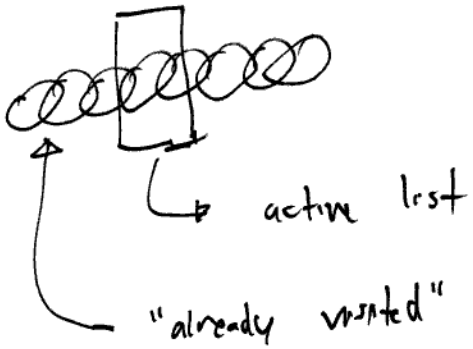
   - aim for $O(K n^2)$, but $O(K^2 \cdot n^2)$ is fine.
   - assume you already have a path (tree) decomposition width K.
   - can assume the decomposition is nice.

- A decomposition has "bags" $X_i \subseteq V$
  - every $v \in V$ is in at least one X_i
  - every edge $e_{uv} \in E$ has $u, v \in X_i$
  - the set of bags X_i containing any fixed $v \in V$ is a connected sub-graph of the tree (path)
  - any bag is a cutset

"Nice" decomposition: each bag

Problem ideas

(1) DP :
- maintain "active list" of nodes as you iterate over bags
- update shortest distances from "already visited" nodes to nodes in the "active list"



(2) Design $O(kn)$ BFS DFS (probably)

Then apply to each node.

Potential "proof review"

Pairwise-Neighborhood-Intersection (PNI): compute $|N(v_i) \cap N(v_j)|$ for all $v_i, v_j \in V$

on a graph with tree-width k in time $O(k \cdot n^2)$

- assume you are given a width k tree decomposition.
  - you can assume it is nice
- assume you have a dictionary that gives each node's neighborhood as a set, e.g., neighb_dict[v] = $\{u_1, \ldots, u_\ell\}$