Last lecture: showed the parallel LDD algorithm of MPX'13:

Computes a $(\beta, O(\frac{\log n}{\beta}))$-decomposition for $0 < \beta < 1$ in

$O(n+m)$ expected work, $O(\log^2 n/\beta)$ depth whp

$\Rightarrow$ βm edges cut in expectation

$\Rightarrow$ Strong diameter of $O(\frac{\log n}{\beta})$ whp.

This lecture: use LDD to solve other graph problems.

$\boxed{\text{Graph Connectivity}}$

Input: Undirected graph $G(V, E)$

Output: Connectivity labeling, L.

i.e. $L[u] = L[v]$ iff $\underbrace{u, v}_{\text{path between } u \text{ and } v}$ in the same component in G

• Sequentially, easy to solve in $O(n+m)$ work using graph search (BFS/DFS).

[Can we get a work-efficient, poly-log depth connectivity alg?]

Lot of work on this problem in early days of parallel algorithms.

Culminated in a randomized work-efficient algorithm with $O(\log n)$ depth on PRAM using $O((m+n)/\log(n))$ processors, i.e. optimal PRAM alg. But the algorithm is extremely complicated and impractical. $\Rightarrow$ not suitable to implement.

Several other work-efficient poly-log depth algorithms since then,

- Halperin/Zwick, Poon-Ramachandran, Cole-Klein-Tarjan
- some based on linear-work MST (will see this later).

Usually use sampling/filtering of edges for work-efficiency and need pretty complex arguments to prove work-efficiency.

This lecture: very simple LDD-based alg that is almost optimal.

LDD-Based Connectivity:

func $CC(G(V,E), \beta)$:

$L = LDD(G, \beta)$

$G'(V', E') = \text{Contract}(G, L)$ // contract all vertices in an LDD cluster into a single vertex; only keep inter-piece edges and remove any duplicates.

if $(|E'| = 0)$ then return L

else

$L' = CC(G', \beta)$ // recurse

$L'' = \{ L'[L[u]] \mid u \in V \}$ // propagate label information

Clear that the algorithm is correct.

Set β to a constant > 0.

- # edges decreases from m to βm in expectation and the rate of reduction is independent across iterations $\Rightarrow O(\log_{1/\beta} m)$ whp.
- Each recursive call is $O(\lg^2 n/\beta)$ depth, $O(m')$ work where m' = # inter-component edges

$\Rightarrow$ Depth $= O(\log_{1/\beta} m \log^2 n/\beta) = O(\log^3 n)$ whp

$\Rightarrow$ Work $= \sum_{i=0}^{\infty} \beta^i m = O(m)$ in expectation

Details:

Consider one level of the alg: Let m', n' be current m, n

LDD: $O(m'+n')$ work, $O(\log^2 n/\beta)$ depth whp

Contract:

- filtering out intra-cluster: $O(m')$ work, $O(\log n')$ depth
- removing duplicates: $O(m')$ expected work, $O(\log n')$ depth whp using semisort

Relabeling: $O(n')$ work, $O(\log n')$ depth

Overall: $O(m'+n')$ expected work, $O(\log^2 n/\beta)$ depth whp. (in a single level).

Is this algorithm practical? Yes!

⇒ Show / analyze code in GBBS.

Spanner: sparse subgraphs that approximate distances in the original graph.

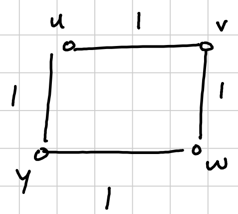
Let $G(V,E)$ be undirected, unweighted.

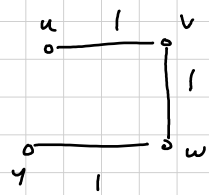
A subgraph H of G is a k-spanner if $\forall u,v \in V$

$$\mathrm{dist}_H(u,v) \le k \cdot \mathrm{dist}_G(u,v)$$

k is called the "stretch" factor.

Example: the mesh graph M_n, eg. M_4 below:

$G =$ 

$H =$ 

Stretch$(H) = 3$ since $\mathrm{dist}_H(u,y) = 3 \le \overset{k}{3} \cdot \mathrm{dist}_G(u,y) = 3$

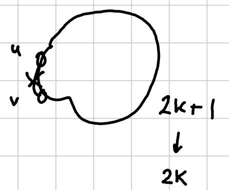
To show a subgraph $H(V', E')$ is a spanner, sufficient to show stretch bounds for every edge $e \in E \setminus E'$.

Thorup-Zwick: $\forall k \ge 1$ any undirected graph on n vertices admits $(2k-1)$-spanners with $O(n^{1+1/k})$ edges. This is essentially as good as possible. (conditional on EGC, below):

- $\gamma(n,k) =$ max # edges in n-vertex graph with girth $> k$
- Unweighted, undirected graphs of girth $> t+1$ do not have proper subgraphs that are t-spanners.

If one removes an edge in such a graph, the distance goes from $1 \to t'$, $t' > t$

⇒ spanner prop. not satisfied



$\text{dist}_H(u,v) \leq 2k \, \text{dist}_G(u,v)$

There is a $(4k+1)$-spanner with $O(n^{1+1/k})$ edges that we can construct in ___ work ___ depth.

Spanner $(G(V,E), k)$

$\beta \leftarrow \log\left(\frac{n}{2k}\right)$

$L \leftarrow LDD(G, \beta)$

$H \leftarrow \emptyset$

- Add all BFS tree edges used in the LDD algorithm to H.
- For each vertex $v \in V$ where v is a boundary vertex, add one edge from v to each adjacent cluster, to H.

return H.

Recall our lemma from last class:

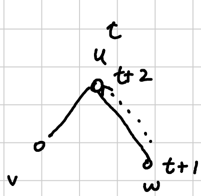
Lemma

- For any vertex or midpoint of an edge, the prob. that the smallest and k-th smallest value from $\{T_v + d(v,u) : v \in V\}$ differ by $\leq a$ is $\leq (a\beta)^{k-1}$

Lemma: In $LDD(G, \beta = \frac{\log n}{2k})$, any vertex intersects $O(n^{1/k})$ other clusters in expectation.

$B(v, 1)$

Consider $u \in V$.



$a=2$

$(2\beta)^{k-1} = \left(\frac{\log n}{k}\right)^{k-1}$

Need to set $a=2$. Plugging $\beta = \frac{\log n}{2k}$

Let $L =$ # intersecting clusters. We have $E[L] = \sum_{l=1}^{\infty} \Pr[L \geq l]$

$$E[L] = \sum_{l=1}^{\infty} (1 - e^{-2\beta})^{l-1} \leq$$

$$= \frac{1}{1-(1-e^{-2\beta})}$$

$$= e^{2\beta}$$

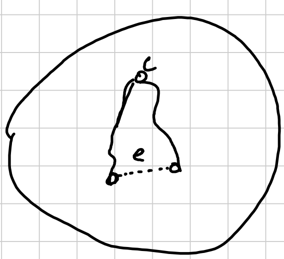
Plugging in $\beta = \frac{\log(n)}{2k}$; $e^{2\beta} = e^{\log(n)/k} = n^{1/k}$

$\Rightarrow$ number of overall intersecting clusters is $O(n^{1+1/k})$ in expectation and $O(n^{1+1/k})$ boundary edges added in expectation.

Stretch?

Consider an edge e. There are a few cases:

(1) e is internal to a cluster.



$\exists$ path through center of this cluster that has length $\leq 2r$ where r is radius of the cluster.

What is r? $= O\left(\frac{\log n}{\beta}\right) = O(k)$ w.h.p.

$\frac{\ln(n)}{\beta}$ in expectation $\Rightarrow$ r is $2k$ in expectation.

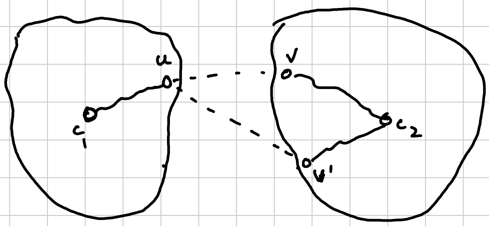
$\Rightarrow$ Stretch $=$ $4k$ in expectation.

$X_i = Exp(\beta)$

$E[X_i] = \frac{1}{\beta}$

Expected value of max $= \ln(n)$

(2) e is inter-cluster



- if (u,v) added as the single inter-cluster edge between c_1 and c_2, stretch is 1.

- Otherwise, $\exists$ some other (u,v') edge that's added.

$$\underset{1}{u \to v'} \underset{r}{\to c_2} \underset{r}{\to v}$$

This path has length $4k+1$ in expectation

$(r = 2k$ exp.$)$

Overall stretch is $4k+1$ in expectation.

Overall algorithm runs in $O(m+n)$ work, $O(\log^2 n/\beta)$ depth whp

$\qquad = O(k \log(n))$ depth whp.