Parallel Graph Algorithms: Connectivity and SSSP

Ian Chen

University of Illinois Urbana-Champaign

Problem Statements

- Connectivity: given an (undirected) graph $G = (V, E)$:

Problem Statements

- Connectivity: given an (undirected) graph $G = (V, E)$:
  - How many connected components does the graph have?

Problem Statements

- Connectivity: given an (undirected) graph $G = (V, E)$:
  - How many connected components does the graph have?
  - What is the size of the largest connected component?

Problem Statements

- Connectivity: given an (undirected) graph $G = (V, E)$:
  - How many connected components does the graph have?
  - What is the size of the largest connected component?
  - WLOG, assume no isolated vertices

Problem Statements

- Connectivity: given an (undirected) graph $G = (V, E)$:
  - How many connected components does the graph have?
  - What is the size of the largest connected component?
  - WLOG, assume no isolated vertices
- Single source shortest paths: given a (weighted, undirected) graph $G = (V, E)$ and source vertex $s \in V$:

Problem Statements

- Connectivity: given an (undirected) graph $G = (V, E)$:
  - How many connected components does the graph have?
  - What is the size of the largest connected component?
  - WLOG, assume no isolated vertices
- Single source shortest paths: given a (weighted, undirected) graph $G = (V, E)$ and source vertex $s \in V$:
  - What is the (length of) shortest path from $s \to v$ for $v \in V$?

Parallel Models

Graph Connectivity

Single Source Shortest Paths

Parallel Models

Graph Connectivity

Single Source Shortest Paths

Parallel Model

- Random-Access Machine (RAM) model:

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:
  - P processors, typically $P = \text{poly}(n)$

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:
  - P processors, typically $P = \text{poly}(n)$
  - Concurrent Read Concurrent Write (CRCW)

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:
  - P processors, typically $P = \text{poly}(n)$
  - Concurrent Read Concurrent Write (CRCW)
- Work: number of elementary operations

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:
  - P processors, typically $P = \text{poly}(n)$
  - Concurrent Read Concurrent Write (CRCW)
- Work: number of elementary operations
- Depth: max sequence of dependent operations

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:
  - P processors, typically $P = \text{poly}(n)$
  - Concurrent Read Concurrent Write (CRCW)
- Work: number of elementary operations
- Depth: max sequence of dependent operations
- Brent's Theorem: PRAM algorithm takes $O(\frac{W}{P} + D)$ time

Parallel Model

- Random-Access Machine (RAM) model:
  - Standard model of manipulating words
- Parallel RAM (PRAM) model:
  - P processors, typically $P = \text{poly}(n)$
  - Concurrent Read Concurrent Write (CRCW)
- Work: number of elementary operations
- Depth: max sequence of dependent operations
- Brent's Theorem: PRAM algorithm takes $O(\frac{W}{P} + D)$ time
- $P^* = \frac{W}{D}$ maximum parallelism of an algorithm

Concurrency Primitives

- map: apply a function f onto each element of an array A

[1] Gu et al. “A Top-Down Parallel Semisort”. 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth

[1] Gu et al. “A Top-Down Parallel Semisort”. 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth
- scan: given binary operation $\oplus$ (associative with identity), return new array $B_i = \bigoplus_{i \leq j} A_i$

[1] Gu et al. "A Top-Down Parallel Semisort". 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth
- scan: given binary operation $\oplus$ (associative with identity), return new array $B_i = \bigoplus_{i \leq j} A_i$
  - $O(n)$ work, $O(\log n)$ depth

[1] Gu et al. "A Top-Down Parallel Semisort". 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth
- scan: given binary operation $\oplus$ (associative with identity), return new array $B_i = \bigoplus_{i \leq j} A_i$
  - $O(n)$ work, $O(\log n)$ depth
- filter: apply predicate f onto each element of an array, and return new array with only true flags

[1] Gu et al. "A Top-Down Parallel Semisort". 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth
- scan: given binary operation $\oplus$ (associative with identity), return new array $B_i = \bigoplus_{i \leq j} A_i$
  - $O(n)$ work, $O(\log n)$ depth
- filter: apply predicate f onto each element of an array, and return new array with only true flags
  - $O(n)$ work, $O(\log n)$ depth

[1] Gu et al. "A Top-Down Parallel Semisort". 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth
- scan: given binary operation $\oplus$ (associative with identity), return new array $B_i = \bigoplus_{i \leq j} A_i$
  - $O(n)$ work, $O(\log n)$ depth
- filter: apply predicate f onto each element of an array, and return new array with only true flags
  - $O(n)$ work, $O(\log n)$ depth
- group: given equality test f, reorder elements such that all equal elements are contiguous

[1] Gu et al. "A Top-Down Parallel Semisort". 2015.

Concurrency Primitives

- map: apply a function f onto each element of an array A
  - $O(n)$ work, $O(1)$ depth
- scan: given binary operation $\oplus$ (associative with identity), return new array $B_i = \bigoplus_{i \leq j} A_i$
  - $O(n)$ work, $O(\log n)$ depth
- filter: apply predicate f onto each element of an array, and return new array with only true flags
  - $O(n)$ work, $O(\log n)$ depth
- group: given equality test f, reorder elements such that all equal elements are contiguous
  - $O(n)$ work, $O(\log n)$ depth[1]

[1] Gu et al. "A Top-Down Parallel Semisort". 2015.

Parallel Models

Graph Connectivity

Single Source Shortest Paths

Parallel BFS

ParallelBFS(undirected graph $G = (V, E)$, source vertex $s \in V$):
  $X \leftarrow \{ s \}$ active set
  $F \leftarrow N(X)$ frontier
  while $|F| > 0$:
    $X \leftarrow X \cup F$; $F \leftarrow N(X)$

- Parallel BFS finds connected component of a source

Parallel BFS

ParallelBFS(undirected graph $G = (V, E)$, source vertex $s \in V$):
  $X \leftarrow \{ s \}$ active set
  $F \leftarrow N(X)$ frontier
  while $|F| > 0$:
    $X \leftarrow X \cup F$; $F \leftarrow N(X)$

- Parallel BFS finds connected component of a source
- Iterate until visit every vertex in the graph

Parallel BFS

ParallelBFS(undirected graph $G = (V, E)$, source vertex $s \in V$):
  $X \leftarrow \{ s \}$ active set
  $F \leftarrow N(X)$ frontier
  while $|F| > 0$:
    $X \leftarrow X \cup F$; $F \leftarrow N(X)$

- Parallel BFS finds connected component of a source
- Iterate until visit every vertex in the graph
- $O(\Delta \log n)$ time where Δ sum of diameters of each component

Random Mating

- Generate (asymmetric) stars with coin flips (tails and heads)

Random Mating

- Generate (asymmetric) stars with coin flips (tails and heads)
- A vertex is contracted if it is a tail and some adjacent node is head

Random Mating

- Generate (asymmetric) stars with coin flips (tails and heads)
- A vertex is contracted if it is a tail and some adjacent node is head
  - $P(v \text{ contracted}) = \frac{1}{2} \times (1 - 2^{-\deg(v)}) \geq \frac{1}{4}$

Random Mating

- Generate (asymmetric) stars with coin flips (tails and heads)
- A vertex is contracted if it is a tail and some adjacent node is head
  - $P(v \text{ contracted}) = \frac{1}{2} \times (1 - 2^{-\deg(v)}) \geq \frac{1}{4}$
- $O(\log n)$ iterations $\implies$ $O(m \log n)$ work and $O(\log^2 n)$ depth

Random Mating

- Generate (asymmetric) stars with coin flips (tails and heads)
- A vertex is contracted if it is a tail and some adjacent node is head
  - $P(v \text{ contracted}) = \frac{1}{2} \times (1 - 2^{-\deg(v)}) \geq \frac{1}{4}$
- $O(\log n)$ iterations $\implies$ $O(m \log n)$ work and $O(\log^2 n)$ depth

CountCcRandomMating($G = (V, E)$):
  if $E = \emptyset$, return $|V|$
  map(coinflip, V)
  $E' \leftarrow$ group edges from $T \to H$; reduce to at most one element
  Contract all edges in E'

Connectivity via Low Diameter Decompositions

- BFS is work efficient but can be sequential on large-diameter graphs

[2] Shun, Dhulipala, and G. Blelloch. "A simple and practical linear-work parallel algorithm for connectivity". 2014.

Connectivity via Low Diameter Decompositions

- BFS is work efficient but can be sequential on large-diameter graphs
- Contractions are depth efficient but cost additional work

[2] Shun, Dhulipala, and G. Blelloch. "A simple and practical linear-work parallel algorithm for connectivity". 2014.

Connectivity via Low Diameter Decompositions

- BFS is work efficient but can be sequential on large-diameter graphs
- Contractions are depth efficient but cost additional work
- Let $\textsc{Ldd}(G) \subset G$ cover $O(m)$ edges and has "low" diameter.

[2] Shun, Dhulipala, and G. Blelloch. "A simple and practical linear-work parallel algorithm for connectivity". 2014.

Connectivity via Low Diameter Decompositions

- BFS is work efficient but can be sequential on large-diameter graphs
- Contractions are depth efficient but cost additional work
- Let $\textsc{Ldd}(G) \subset G$ cover $O(m)$ edges and has "low" diameter.
- Contracting componentsd of LDD leads to efficient algorithm[2]

$\textsc{Cc}(G = (V, E))$:
  if $E' = \emptyset$, return V
  $L \leftarrow \textsc{CcBfs}(\textsc{Ldd}(G))$
  $L' \leftarrow \textsc{Cc}(\textsc{Contract}(\mathrm{G},\ \mathrm{L}))$
  Return $\textsc{Relabel}(L, L')$

[2] Shun, Dhulipala, and G. Blelloch. "A simple and practical linear-work parallel algorithm for connectivity". 2014.

Random Shift LDD

- Start a BFS from multiple sources

Random Shift LDD

- Start a BFS from multiple sources
- Partition nodes based on nearest source

Random Shift LDD

- Start a BFS from multiple sources
- Partition nodes based on nearest source
- u promotes itself to a seed at time T_u, if not visited

Random Shift LDD

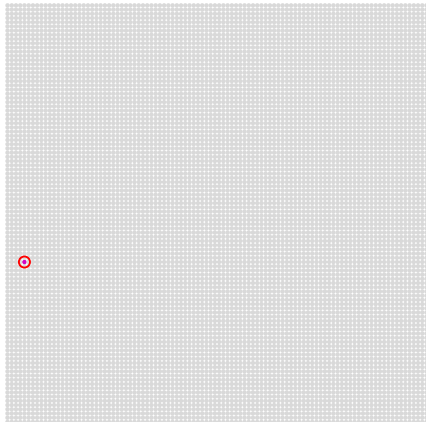


Figure: **Low Diameter Decomposition with Random Shifts**: 100×100 grid graph. Colors represent clusters (gray unclustered). Using $\lambda = 25$.

Random Shift LDD

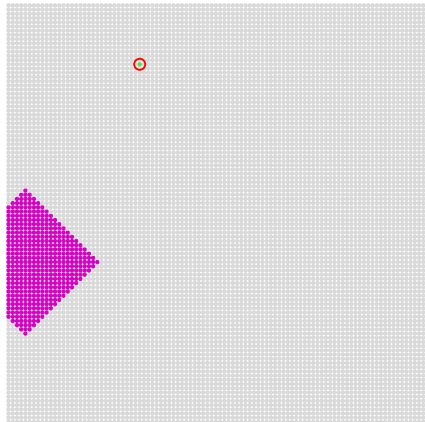


Figure: **Low Diameter Decomposition with Random Shifts**: 100×100 grid graph. Colors represent clusters (gray unclustered). Using $\lambda = 25$.

Random Shift LDD

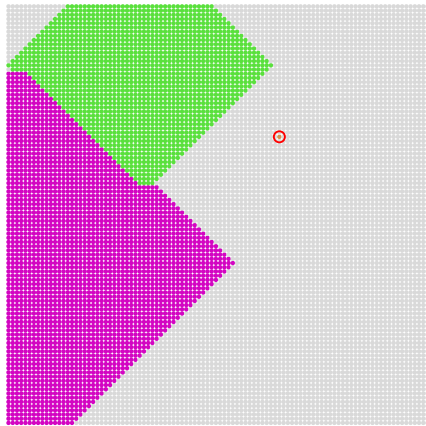


Figure: **Low Diameter Decomposition with Random Shifts**: 100×100 grid graph. Colors represent clusters (gray unclustered). Using $\lambda = 25$.

Random Shift LDD

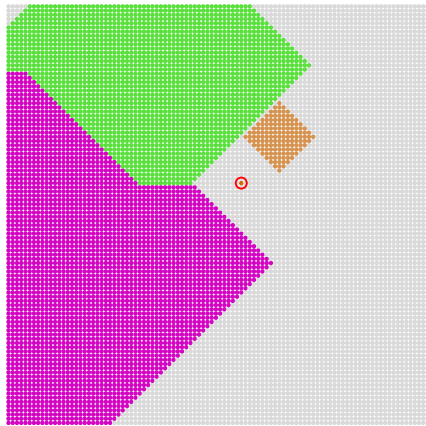


Figure: **Low Diameter Decomposition with Random Shifts**: 100×100 grid graph. Colors represent clusters (gray unclustered). Using $\lambda = 25$.

Random Shift LDD

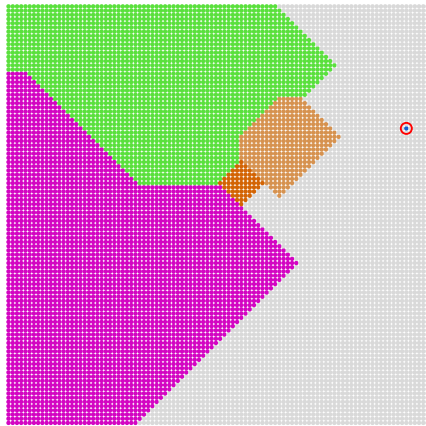


Figure: **Low Diameter Decomposition with Random Shifts**: 100×100 grid graph. Colors represent clusters (gray unclustered). Using $\lambda = 25$.

Random Shift LDD

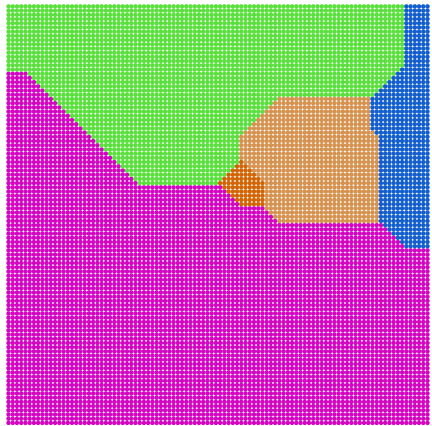


Figure: **Low Diameter Decomposition with Random Shifts**: 100×100 grid graph. Colors represent clusters (gray unclustered). Using $\lambda = 25$.

Random Shift LDD

- What conditions do we want on T_u?

[3] Miller, Peng, and Xu. "Parallel graph decompositions using random shifts". 2013.

Random Shift LDD

- What conditions do we want on T_u?
  - T_u and $T_u^{(k)}$ are independent

[3] Miller, Peng, and Xu. “Parallel graph decompositions using random shifts”. 2013.

Random Shift LDD

- What conditions do we want on T_u?
  - T_u and $T_u^{(k)}$ are independent
  - $T_u \leq O(\log n)$ with high probability, for low diameter

[3] Miller, Peng, and Xu. "Parallel graph decompositions using random shifts". 2013.

Random Shift LDD

- What conditions do we want on T_u?
  - T_u and $T_u^{(k)}$ are independent
  - $T_u \leq O(\log n)$ with high probability, for low diameter
  - $T_u^{(2)} - T_u^{(1)} > T_u^{(3)} - T_u^{(2)} > \ldots$, for low "contention"

[3] Miller, Peng, and Xu. "Parallel graph decompositions using random shifts". 2013.

Random Shift LDD

- What conditions do we want on T_u?
  - T_u and $T_u^{(k)}$ are independent
  - $T_u \leq O(\log n)$ with high probability, for low diameter
  - $T_u^{(2)} - T_u^{(1)} > T_u^{(3)} - T_u^{(2)} > \ldots$, for low "contention"
- $T_u \sim C - \text{Exp}(\lambda = \frac{1}{2})$ for some constant C

[3] Miller, Peng, and Xu. "Parallel graph decompositions using random shifts". 2013.

Random Shift LDD

- What conditions do we want on T_u?
  - T_u and $T_u^{(k)}$ are independent
  - $T_u \leq O(\log n)$ with high probability, for low diameter
  - $T_u^{(2)} - T_u^{(1)} > T_u^{(3)} - T_u^{(2)} > \ldots$, for low "contention"
- $T_u \sim C - \text{Exp}(\lambda = \frac{1}{2})$ for some constant C
- Find components via nearest neighbor $d_\delta(u, v) = T_u + d(u, v)$[3]

LDD($G = (V, E)$):
  Form G' with additional vertex s
  Add edges su with weight $-\delta_u$ for each $u \in V$
  Return BFS(G')

[3] Miller, Peng, and Xu. "Parallel graph decompositions using random shifts". 2013.

Recall: Exponential Distribution

- Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Exp}(p)$, $X^{(k)} = X_{i_k}$ be the k-th order statistic:

Recall: Exponential Distribution

- Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Exp}(p)$, $X^{(k)} = X_{i_k}$ be the k-th order statistic:
- $P(X_i > t) = e^{-\lambda t}$, $E(X_i) = \frac{1}{\lambda} = 2$

Recall: Exponential Distribution

- Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Exp}(p)$, $X^{(k)} = X_{i_k}$ be the k-th order statistic:
- $P(X_i > t) = e^{-\lambda t}$, $E(X_i) = \frac{1}{\lambda} = 2$
- $P(X_i > s + t \mid X_i > s) = P(X_i > t) = e^{-\beta t}$

Recall: Exponential Distribution

- Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Exp}(p)$, $X^{(k)} = X_{i_k}$ be the k-th order statistic:
- $P(X_i > t) = e^{-\lambda t}$, $E(X_i) = \frac{1}{\lambda} = 2$
- $P(X_i > s + t \mid X_i > s) = P(X_i > t) = e^{-\beta t}$
- $X^{(k+1)} - X^{(k)} \sim \text{Exp}((n-k-1)\lambda)$

$$\begin{aligned} X_{i_{k+1}} - X_{i_k} &= P(X_{i_{k+1}}, \ldots, X_{i_n} > t + X_{i_k} \mid X_{i_{k+1}}, \ldots, X_{i_n} > X_{i_k}) \\ &= P(X_{i_{k+1}}, \ldots, X_{i_n} \geq t) \\ &= (e^{-\lambda t})^{n-k-1} = e^{-((n-k-1)\lambda)t} \end{aligned}$$

Analysis I

Lemma

Let $[u] = \arg\min_{v \in V} d_\delta(v, u)$*. If* $[u] = [v]$*, then* $[u] = [w]$ *for all* w *on shortest path (in* G*) between* u *and* v*.*

Proof.

- Let $u_0, u_1, \ldots, u_\ell$ be shortest path, $u_0 = u$ and $u_\ell = v$
- Induct on u_i from $i = \ell$ to $i = 0$
- For $i < \ell, u' \in V$, then $d_\delta(u, u_i) \leq d_\delta(u', u_i)$, since

$$d_\delta(u, u_i) + 1 \leq d_\delta(u, u_{i+1}) \leq d_\delta(u', u_{i+1}) \leq d_\delta(u', u_i) + 1$$

- With probability 1, there are no ties, so equality only if $u = u'$

□

Analysis II

Lemma

The diameter of each component is at most $O(\frac{\log n}{\lambda})$ *w.h.p.*

Proof.

- $\Delta = \max_{u \in V} \min_{v \in V} d_\delta(v, u)$
- Fix any u, and let $w = \arg\min_{v \in V} d_\delta(v, u)$
- $d(w, u) = d_\delta(w, u) + \delta_w \leq d_\delta(u, u) + \delta_w = -\delta_u + \delta_w \leq \delta_w$
- $$P(\delta_w > 2\frac{\log n}{\lambda}) = \exp\left\{(-\lambda) \cdot 2\frac{\log n}{\lambda}\right\} = \frac{1}{n^2}$$
- Union bound over all vertices, then $P(\Delta > 2\frac{\log n}{\lambda}) \leq \frac{1}{n}$

□

Analysis III

Lemma

The expected number of edges uv with $[u] \neq [v]$ is at most λm.

Proof.

- Fix an edge uv. We will show $P([u] \neq [v]) \leq \lambda$.
- WLOG, say $d_\delta([u], u) \leq d_\delta([v], v)$. Then, $d_\delta([u], u) \leq d_\delta([v], u) + 1$
- Let $w \in V$ be the second min of $d_\delta(w, u)$: $d_\delta(w, u) \leq d_\delta([v], u) + 1$
- Then, $Y = -d(w, u) + \delta_w$ and $X = -d([u], u) + \delta_{[u]}$ are the max and 2nd max of shifted geometric distribution
- $$P(Y - X \leq 1) = P(X^n_{(n)} - X^n_{(n-1)} \leq 1) = 1 - e^{-\lambda} \leq \lambda$$

□

Analysis IV

Theorem

There exists an algorithm that solves connected components with $O(n+m)$ work and $O(\log^3 n)$ span with high probability.

Proof.

- Find LDD with $\Delta = 2\log n$ and $m/2$ edges w.h.p. ($\lambda = \frac{1}{2}$)
- BFS takes $O(\Delta \log n) = O(\log^2 n)$ depth and $O(n+m)$ work
- Work W and depth S recurrence (w.h.p.):

$$W(n,m) = O(n+m) + W(n,m') \leq O(n+m) + W(n,m/2) = O(n+m)$$

$$S(n,m) = O(\log^2 n) + S(n,m') \leq O(\log^2 n) + S(n,m/2) = O(\log^3 n)$$

□

Parallel Models

Graph Connectivity

Single Source Shortest Paths

Bellman-Ford

- Edge relaxation: $d(v) \leq \hat{d}_k(v) \leq \hat{d}_{k-1}(u) + w(u, v)$

Bellman-Ford

- Edge relaxation: $d(v) \leq \hat{d}_k(v) \leq \hat{d}_{k-1}(u) + w(u, v)$
- Nodes are settled in level-order of shortest paths tree T_s

Bellman-Ford

- Edge relaxation: $d(v) \leq \hat{d}_k(v) \leq \hat{d}_{k-1}(u) + w(u, v)$
- Nodes are settled in level-order of shortest paths tree T_s
- $O(hm)$ work, $O(h \log n)$ depth, where h is height of T_s

BellmanFord(undirected graph $G = (V, E)$, source vertex $s \in V$):
  $\hat{d}(\cdot) \leftarrow +\infty$, $\hat{d}(s) \leftarrow 0$
  for k from 0 to ...
    for $v_{jk} \in V$:
      $\hat{d}_k(\cdot) \leftarrow \min(\hat{d}_k(\cdot), \hat{d}_k(v_{jk}) + w(v_{jk}, \cdot))$
  return $\hat{d}_k(\cdot)$

Dijkstras

- Edge relaxation[4]: $d(v) \leq \hat{d}(v) \leq \hat{d}(u) + w(u, v)$

[4]We assume that all weights are at least 1.

Dijkstras

- Edge relaxation[4]: $d(v) \leq \hat{d}(v) \leq \hat{d}(u) + w(u, v)$
- Nodes are "settled" ($\hat{d}(v) = d(v)$) in distance-order

[4]We assume that all weights are at least 1.

Dijkstras

- Edge relaxation[4]: $d(v) \leq \hat{d}(v) \leq \hat{d}(u) + w(u, v)$
- Nodes are "settled" ($\hat{d}(v) = d(v)$) in distance-order
- $O(m \log n)$ work, $O(n \log n)$ depth

DIJKSTRAS(undirected graph $G = (V, E)$, source vertex $s \in V$):
$\hat{d}(\cdot) \leftarrow +\infty$, $\hat{d}_0(s) \leftarrow 0$
for k from 0 to $n - 1$:
$v_k \leftarrow \arg\min_{v \text{ not yet seen}} \hat{d}(v)$
$\hat{d}(\cdot) = \min(\hat{d}(\cdot), \hat{d}(v_k) + w(v_k, \cdot))$
return $\hat{d}(\cdot)$

[4]We assume that all weights are at least 1.

Focusing on depth

- In worst case, $h = n - 1$ (consider a linked list)

Focusing on depth

- In worst case, $h = n - 1$ (consider a linked list)
- What are some graphs where the height is much smaller?

Focusing on depth

- In worst case, $h = n - 1$ (consider a linked list)
- What are some graphs where the height is much smaller?
  - Cliques

Focusing on depth

- In worst case, $h = n - 1$ (consider a linked list)
- What are some graphs where the height is much smaller?
  - Cliques
  - Stars

Focusing on depth

- Barbarasi-Alberts (preferential-attachment) networks

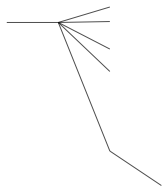


Figure: **Growth of Barbarasi-Alberts network**: BA($n = 8, m = 1$)

Focusing on depth

- Barbarasi-Alberts (preferential-attachment) networks

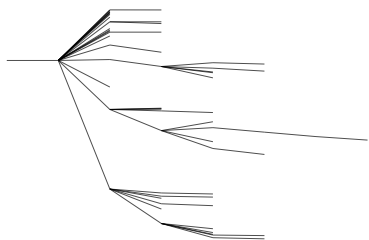


Figure: **Growth of Barbarasi-Alberts network**: BA($n = 64, m = 1$)

Focusing on depth

- Barbarasi-Alberts (preferential-attachment) networks

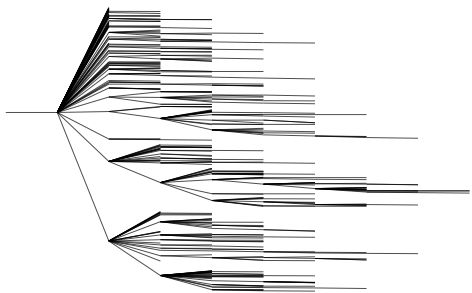


Figure: **Growth of Barbarasi-Alberts network**: BA($n = 512, m = 1$)

Focusing on depth

- Barbarasi-Alberts (preferential-attachment) networks

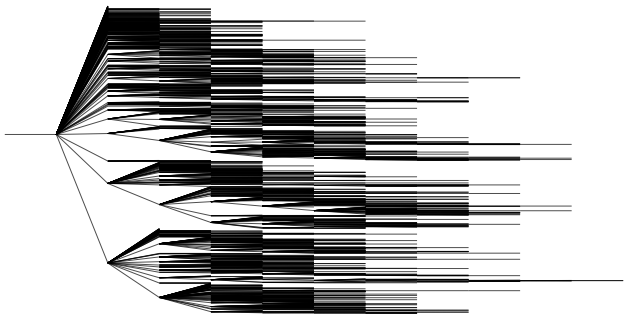


Figure: **Growth of Barbarasi-Alberts network**: $\mathrm{BA}(n = 4096, m = 1)$

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

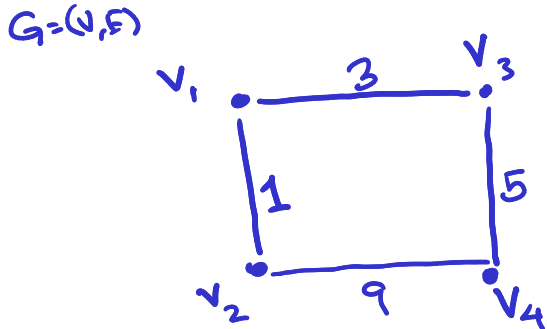


Figure: **Converting** $(1, 2)$ **graph**: Input graph with $n = m = 4$

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

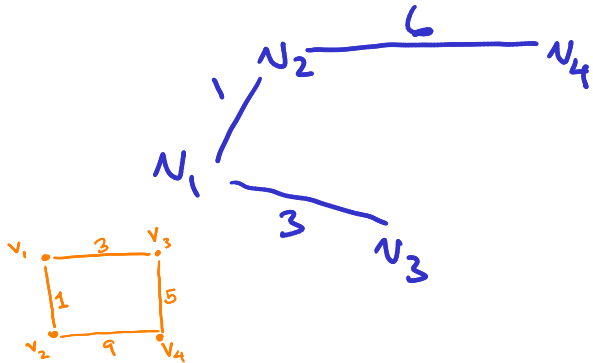


Figure: **Converting** $(1, 2)$ **graph**: Shortest path tree of v_1

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

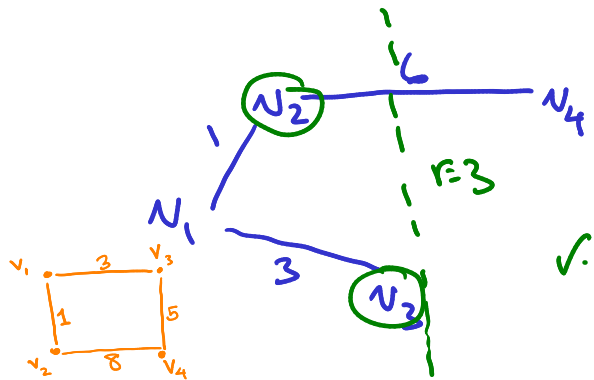


Figure: **Converting** $(1, 2)$ **graph**: v_1 adjacent to ρ closest nodes

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

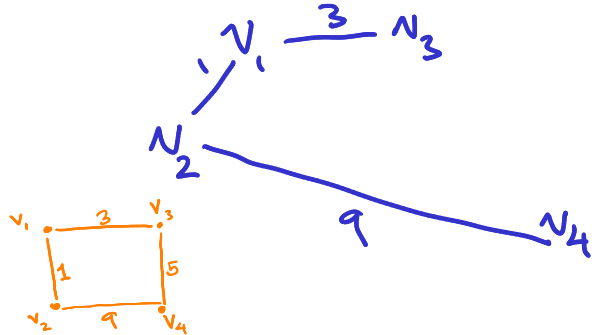


Figure: **Converting** $(1, 2)$ **graph**: Shortest path tree of v_2

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

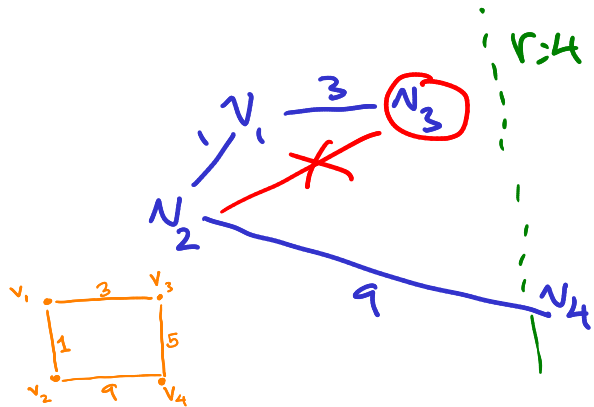


Figure: **Converting** $(1, 2)$ **graph**: v_2 not adjacent to ρ nearest nodes

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

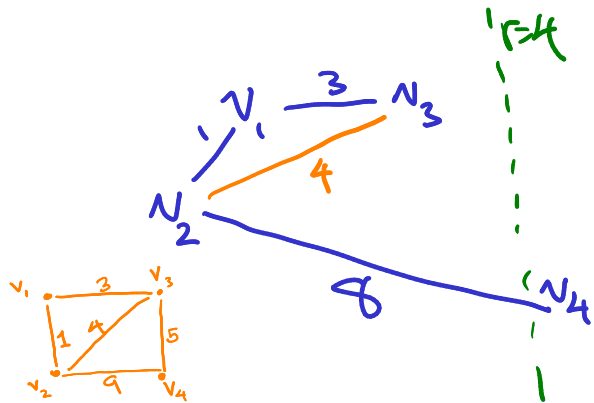


Figure: **Converting** $(1, 2)$ **graph**: Shortcut v_2v_3 with weight 4

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

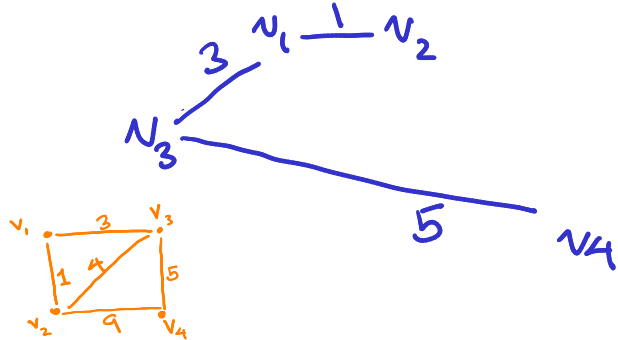


Figure: **Converting** $(1, 2)$ **graph**: Shortest path tree of v_3

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

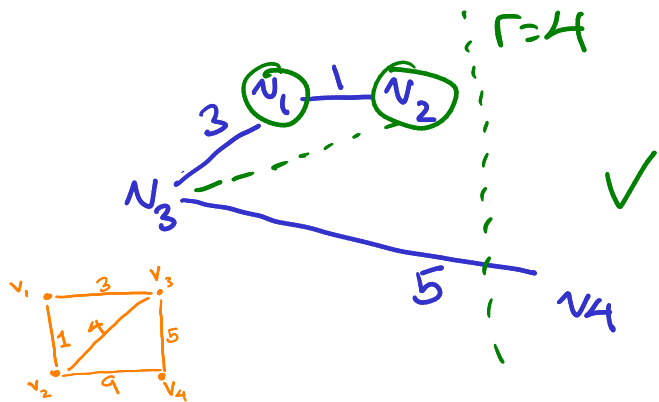


Figure: **Converting** $(1, 2)$ **graph**: v_3 is adjacent to ρ closest nodes

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

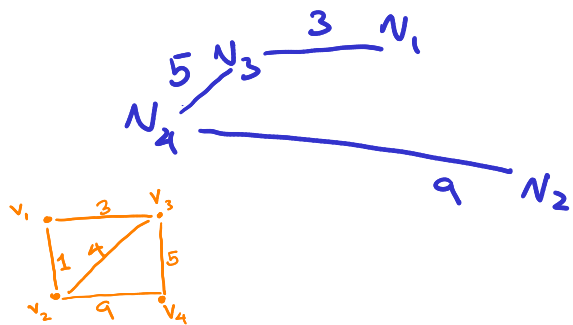


Figure: **Converting** $(1, 2)$ **graph**: Shortest path tree of v_4

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes

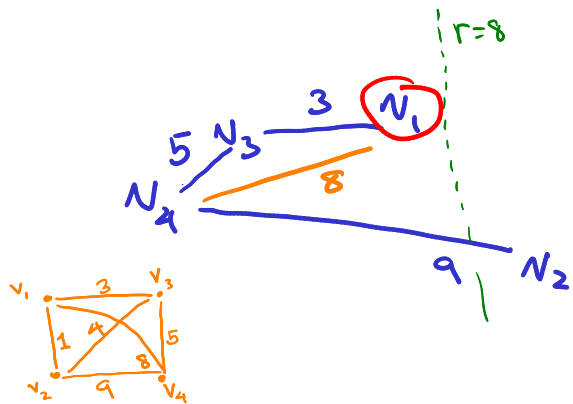


Figure: **Converting** $(1, 2)$ **graph**: Shortcut v_4v_1 with weight 8

$(1, \rho)$ graphs

- G is $(1, \rho)$ if every vertex is adjacent to its ρ closest nodes
- Every G is a $(1, 1)$ graph
- Can check whether a graph is $(1, \rho)$ in $O(\rho \log \rho)$ depth
- Can convert a graph to $(1, \rho)$ in $O(\rho \log \rho)$ depth[5]

[5] G. E. Blelloch et al. “Parallel Shortest Paths Using Radius Stepping”. 2016.

Bounding depth on $(1, \rho)$ graphs

Theorem

If G is $(1, \rho)$ graph, Bellman-Ford runs in depth[6] $\tilde{O}(\frac{n}{\rho} \log L)$

Lemma

After any iteration i, it takes at most $1 + \log_2 \rho L$ iterations of Bellman-Ford to settle ρ additional nodes.

[6] $\tilde{O}(\cdot)$ hides poly-log (in n) factors

Bounding depth on $(1, \rho)$ graphs

Theorem

If G is $(1, \rho)$ graph, Bellman-Ford runs in depth[6] $\tilde{O}(\frac{n}{\rho} \log L)$

Lemma

After any iteration i, it takes at most $1 + \log_2 \rho L$ iterations of Bellman-Ford to settle ρ additional nodes.

Proof.

[6] $\tilde{O}(\cdot)$ hides poly-log (in n) factors

Bounding depth on $(1, \rho)$ graphs

Theorem

If G is $(1, \rho)$ graph, Bellman-Ford runs in depth[6] $\tilde{O}(\frac{n}{\rho} \log L)$

Lemma

After any iteration i, it takes at most $1 + \log_2 \rho L$ iterations of Bellman-Ford to settle ρ additional nodes.

Proof.

- Each iteration of Bellman-Ford can be parallelized in $O(\log n)$ depth

[6] $\tilde{O}(\cdot)$ hides poly-log (in n) factors

Bounding depth on $(1, \rho)$ graphs

Theorem

If G is $(1, \rho)$ graph, Bellman-Ford runs in depth[6] $\tilde{O}(\frac{n}{\rho} \log L)$

Lemma

After any iteration i, it takes at most $1 + \log_2 \rho L$ iterations of Bellman-Ford to settle ρ additional nodes.

Proof.

- Each iteration of Bellman-Ford can be parallelized in $O(\log n)$ depth
- It takes $\left\lceil \frac{n}{\rho} \right\rceil (1 + \log_2 \rho L)$ iterations to settle all n nodes

[6] $\tilde{O}(\cdot)$ hides poly-log (in n) factors

Bounding depth on $(1, \rho)$ graphs

Theorem

If G is $(1, \rho)$ graph, Bellman-Ford runs in depth[6] $\tilde{O}(\frac{n}{\rho} \log L)$

Lemma

After any iteration i, it takes at most $1 + \log_2 \rho L$ iterations of Bellman-Ford to settle ρ additional nodes.

Proof.

- Each iteration of Bellman-Ford can be parallelized in $O(\log n)$ depth
- It takes $\left\lceil \frac{n}{\rho} \right\rceil (1 + \log_2 \rho L)$ iterations to settle all n nodes
- Therefore, Bellman-Ford has depth $O(\frac{n}{\rho} \log \rho L \log n)$

□

[6] $\tilde{O}(\cdot)$ hides poly-log (in n) factors

Depth on $(1, \rho)$ graphs

Lemma

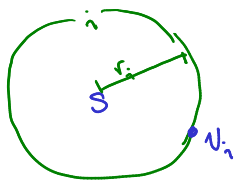
After any iteration i, it takes at most $1 + \log_2 \rho L$ iterations of Bellman-Ford to settle ρ additional nodes.

Proof.

- Let v_j be furthest settled node from s in iteration j, $r_j = d(v_j)$
- Denote $r(v)$ as the ρ-smallest distance for node v
- If $r(v) < d_j - d_i$, then we find ρ new nodes by $j + 1$
- If $r(v) \geq d_j - d_i$, then $d_{j+1} - d_j \geq d_j - d_i$
- Since $r(v) \leq \rho L$, then case 2 happens at most $\log_2 \rho L$ times

□

Depth on $(1, \rho)$ graphs I



Radius Stepping

RadiusStepping($(1,\rho)$ graph $G=(V,E)$, source vertex $s \in V$):

$\hat{d}(\cdot) \leftarrow +\infty$, $\hat{d}(s) \leftarrow 0$

$S_0 \leftarrow \{ s \}$

for k from 1 to ...

$r_k \leftarrow \min_{v \in V \setminus S_{k-1}} \{ \hat{d}(v) + r(v) \}$

for $v_{jk} \in V \setminus S_{k-1}$ where $\hat{d}(v_{jk}) \leq r_k$

$\hat{d}(\cdot) \leftarrow \min(\hat{d}(\cdot), \hat{d}(v_{jk}) + w(v_{jk}, \cdot))$

$S_k \leftarrow \{ v \in V \mid \hat{d}(v) \leq r_k \}$; $k \leftarrow k+1$

return $\hat{d}(\cdot)$

Work-Depth Tradeoffs

- $\tilde{O}(n\rho^2 + (m+n)\rho)$ work and $\tilde{O}(\rho + \frac{n}{\rho} \log L)$ depth

Work-Depth Tradeoffs

- $\tilde{O}(n\rho^2 + (m+n)\rho)$ work and $\tilde{O}(\rho + \frac{n}{\rho}\log L)$ depth
  - $O(n\rho^2)$ work and $O(\rho \log \rho)$ depth to preprocess

Work-Depth Tradeoffs

- $\tilde{O}(n\rho^2 + (m+n)\rho)$ work and $\tilde{O}(\rho + \frac{n}{\rho}\log L)$ depth
  - $O(n\rho^2)$ work and $O(\rho \log \rho)$ depth to preprocess
  - $O((m+n)\rho)$ work and $O(\frac{n}{\rho}\log \rho L \log n)$ depth for Bellman-Ford

Work-Depth Tradeoffs

- $\tilde{O}(n\rho^2 + (m+n)\rho)$ work and $\tilde{O}(\rho + \frac{n}{\rho}\log L)$ depth
  - $O(n\rho^2)$ work and $O(\rho \log \rho)$ depth to preprocess
  - $O((m+n)\rho)$ work and $O(\frac{n}{\rho}\log \rho L \log n)$ depth for Bellman-Ford
- $\rho = \sqrt{n}$, then $O(\sqrt{n})$ depth, but only efficient on very dense graphs

Work-Depth Tradeoffs

- $\tilde{O}(n\rho^2 + (m+n)\rho)$ work and $\tilde{O}(\rho + \frac{n}{\rho}\log L)$ depth
  - $O(n\rho^2)$ work and $O(\rho \log \rho)$ depth to preprocess
  - $O((m+n)\rho)$ work and $O(\frac{n}{\rho}\log \rho L \log n)$ depth for Bellman-Ford
- $\rho = \sqrt{n}$, then $O(\sqrt{n})$ depth, but only efficient on very dense graphs
- $\rho = \text{polylog}(n)$, then $o(n)$ depth and nearly work efficient

SOTA

Algorithm	Work	Span
Dijkstra	$\tilde{O}(m)$	$\tilde{O}(n)$
Bellman-Ford	$\tilde{O}(k_n m)$	$\tilde{O}(k_n)$
Δ^*-Stepping	$\tilde{O}(k_n m)$	$\tilde{O}\left(\frac{k_n(\Delta+L)}{\Delta}\right)$
Radius-Stepping	$\tilde{O}(k_\rho m)$	$\tilde{O}\left(\frac{k_\rho n}{\rho}\log L\right)$
ρ-Stepping	$\tilde{O}(k_n m)$	$\tilde{O}\left(\frac{k_\rho n}{\rho}\right)$

Table: Adapted from Table 2 of Dong et al. (2021). k_n is the height of shortest path tree. k_ρ is chosen such that G is a (k_ρ, ρ) graph. Δ is a positive constant.

References I

- Blelloch, Guy E. et al. (July 2016). “Parallel Shortest Paths Using Radius Stepping”. In: *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA ’16. ACM, pp. 443–454. DOI: 10.1145/2935764.2935765.
- Dong, Xiaojun et al. (July 2021). “Efficient Stepping Algorithms and Implementations for Parallel Shortest Paths”. In: *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA ’21. ACM, pp. 184–197. DOI: 10.1145/3409964.3461782.
- Gu, Yan et al. (June 2015). “A Top-Down Parallel Semisort”. In: *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*. SPAA ’15. ACM, pp. 24–34. DOI: 10.1145/2755573.2755597.

References II

- Miller, Gary L., Richard Peng, and Shen Chen Xu (July 2013). "Parallel graph decompositions using random shifts". In: *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. SPAA '13. ACM, pp. 196–203. DOI: 10.1145/2486159.2486180.
- Shun, Julian, Laxman Dhulipala, and Guy Blelloch (June 2014). "A simple and practical linear-work parallel algorithm for connectivity". In: *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*. SPAA '14. ACM, pp. 143–153. DOI: 10.1145/2612669.2612692.