I/O-Efficient Algorithms for Sparse Graphs

Laura Toma
Duke University

Norbert Zeh
Carleton University

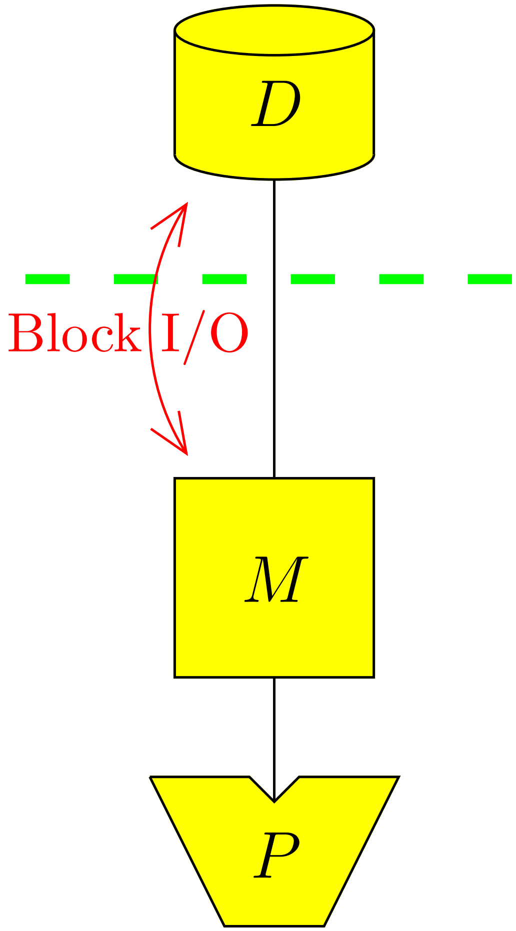
GI-Dagstuhl Seminar:
Algorithms for Memory Hierarchies
March 2002

External Graph Problems

- ⋆ Applications
  - Geographic Information Systems (GIS):
    - ∗ Terrain analysis: flow modeling, topographic indices
    - ∗ Routing (e.g. find optimal routes given US road network)
  - Web modeling
    - ∗ Web crawl of 200M nodes, 2000M links: shortest paths, (strongly) connected components, breadth/depth first search, diameter [BK00]
    - ∗ Search engines
- ⋆ Data resides on disk $\Longrightarrow$ I/O bottleneck

Parallel Disk Model (PDM)

[Vitter & Shriver]



M = # of vertices/edges that fit in memory

B = # of vertices/edges per disk block

D = # of disks

- ☆ I/O operation
- ☆ I/O complexity
- ☆ Basic bounds
  - $\text{scan}(E) = \frac{E}{B} \ll E$
  - $\text{sort}(E) = \Theta(\frac{E}{B} \log_{M/B} \frac{E}{B}) \ll E$

Upper & Lower Bounds

Upper bounds – deterministic, linear space

Problem	**General undirected graphs**	
CC, MST	$O\big(\mathrm{sort}(\|E\|) \cdot \log\log \frac{\|V\|B}{\|E\|}\big)$	[MR99, ABT01]
SSSP	$O\left(\|V\| + \frac{E}{B} \cdot \log \frac{\|V\|}{B}\right)$	[KS96]
DFS	$O\left(\|V\| + \frac{\|V\|}{M} \cdot \mathrm{scan}(E)\right)$	[CGG+95]
	$O\left((\|V\| + \mathrm{scan}(\|E\|)) \cdot \log_2 \|V\|\right)$	[KS96]
BFS	$O\left(\|V\| + \frac{\|E\|}{\|V\|} \cdot \mathrm{sort}(\|V\|)\right)$	[MR99]

Lower bounds: $\min\{V, \mathrm{sort}(|V|)\}$

Sparse Graphs

If $|E| = O(|V|)$:

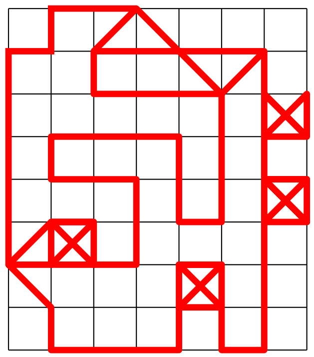
- ⋆ CC, MST: $O\big(\text{sort}(|V|) \cdot \log\log B\big)$
- ⋆ SSSP, BFS, DFS: $O(|V|)$

$G(V,E)$ **sparse** if $|E(H)| = O(|V(H)|)$, for any graph H which can be obtained from G by a series of edge contractions followed by removing duplicate edges.

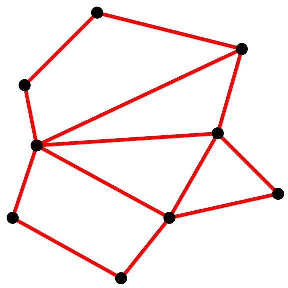
- ⋆ CC, MST in $O(\text{sort}(N))$ on sparse graphs
- ⋆ BFS, SSSP, DFS ? Open on sparse graphs
  - $O(\text{sort}(N))$ on planar graphs, grid graphs, outerplanar graphs, bounded treewidth graphs

Some Classes of Sparse Graphs

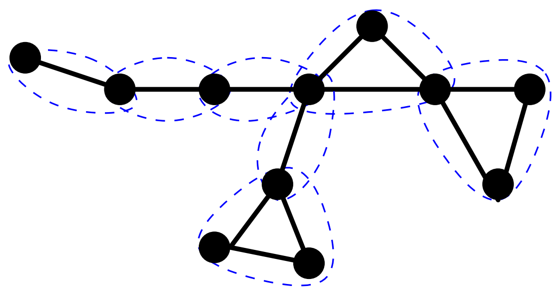
- ☆ Grid graphs



- ☆ Outerplanar graphs



- ☆ Bounded treewidth graphs
  - Tree decomposition: Partition of the edges into a set of subgraphs which "fit together in a tree-like way"



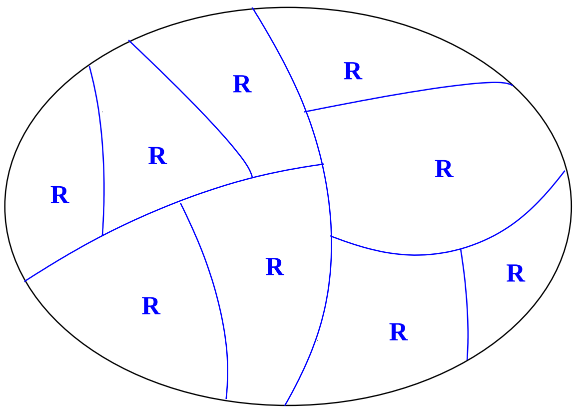
What do they have in common?

Small separators

An ϵ-separator of G is a set S of vertices whose removal disconnects G into subgraphs having at most ϵN vertices.

Planar graph separation

- ☆ [LT] 2 subgraphs with $\frac{2N}{3}$ vertices each and $O(\sqrt{N})$ separator vertices ($\frac{2}{3}$-separator)
- ☆ $\Longrightarrow$ (apply recursively) $O(\frac{N}{R})$ subgraphs with $O(R)$ vertices each and $O(\frac{N}{\sqrt{R}})$ separator vertices ($\frac{R}{N}$-separator)



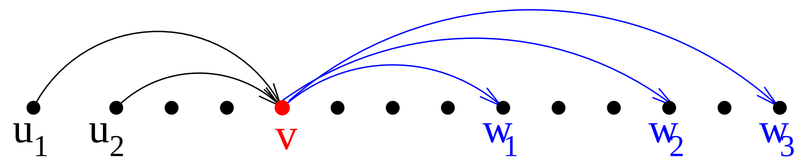
Planar graphs, grid graphs, outerplanar graphs, bounded treewidth graphs have small separators that can be computed efficiently in $O(\text{sort}(N))$ I/Os

Outline of the talk

- ⋆ Techniques
  - Graph contraction
  - Time forward processing
- ⋆ Connectivity problems (CC, MST, BCC, ear decomposition)
- ⋆ BFS and SSSP
- ⋆ DFS
- ⋆ Separators
- ⋆ Embedding and tree-decomposition

Time Forward Processing ([CGG+95], [A95])

Assume G is a DAG with vertices numbered in topological order. Compute for each vertex v a "value" based on the values of its in-neighbors u_i.



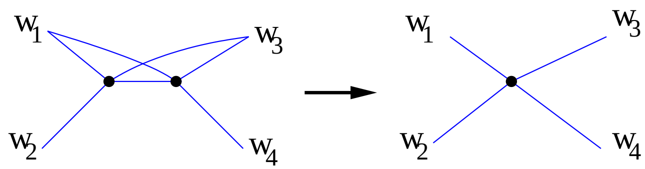
How to compute the value of u_i without spending one I/O?

- ☆ Priority queue: stores values of u_i with priority=v
- ☆ When processing v:
  - ExtractMin to find values of u_i
  - For each out-edge (v, w_j) insert value of v in pqueue with priority w_j (send forward in time)

$\Longrightarrow O(\text{sort}(E))$ I/Os

Graph Contraction

Edge contraction:



Graph contraction: Identify disjoint subgraphs and contract (to a point or to a smaller subgraph).

Goal: Reduce the size of the graph

$$G = G_0 \longrightarrow G_1 \cdots \longrightarrow G_k$$

- ☆ Solve the problem on G_k and derive the solution for $G_{k-1}, \ldots G_0$
- ☆ Typically a contraction step reduces the size of G by a constant fraction $\Longrightarrow O(\log V)$ contraction steps
- ☆ I/O-efficient graph contraction: usually stop after $O(\log B)$ steps ($V' = \frac{V}{B}$)

Connectivity Problems on Sparse Graphs ([CGG+95])

☆ CC, MST

- Graph contraction $G = G_0 \longrightarrow G_1 \longrightarrow \ldots$, where $V_i \leq \frac{V_{i-1}}{2}$
- Contraction on G_i takes $O(\mathrm{sort}(E_i))$ I/Os $\Longrightarrow \sum_i O(\mathrm{sort}(E_i))$
- Sparse graphs: $E_i = O(V_i) \Longrightarrow \sum_i O(\mathrm{sort}(E_i)) = O(\mathrm{sort}(E))$

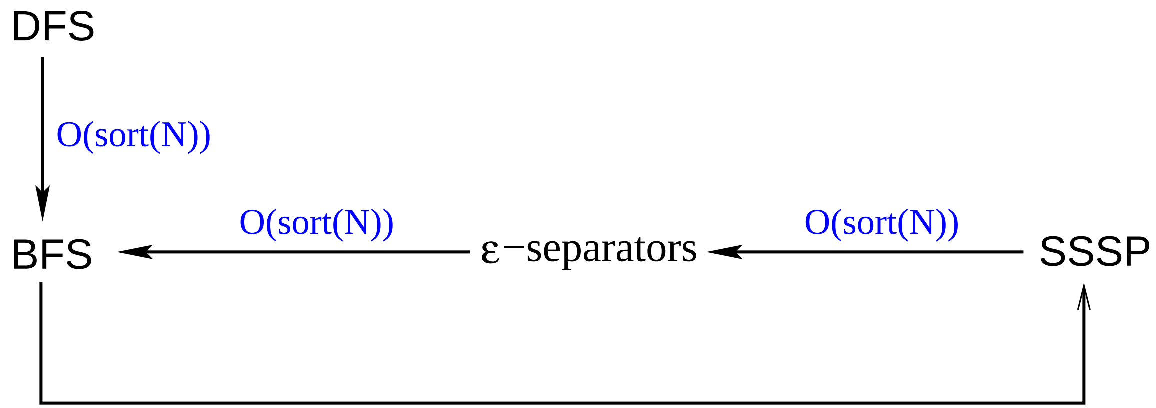
☆ Biconnected components, ear decomposition

- Based on PRAM algorithms [TV85,MSV86]
- Biconnected components: Reduces to computing a spanning tree, computing bottom-up labeling, and computing CC in a new graph G'
- Ear decomposition: Reduces to computing a spanning tree T, computing a BFS of T, and batched *lca* queries on T

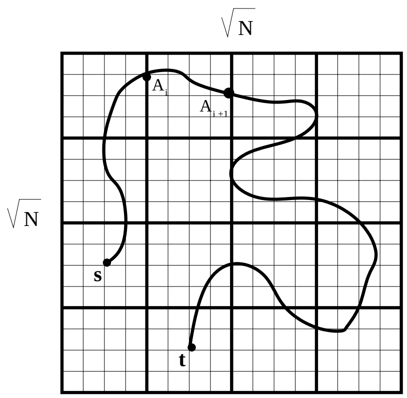
BFS and SSSP

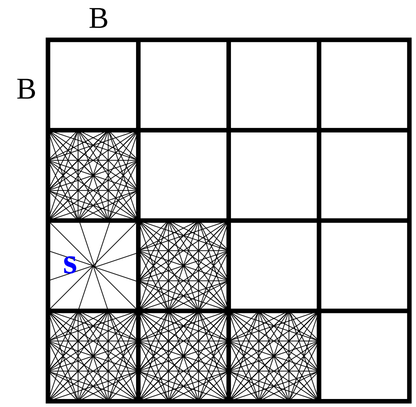
sparse	*planar*	*grid*	*outerplanar*	*bounded treewidth*
open	$O(\text{sort}(N))$	$O(\text{sort}(N))$	$O(\text{sort}(N))$	$O(\text{sort}(N))$

- ⋆ Existence of small separators
- ⋆ Planar graphs: $O(\text{sort}(N))$ reductions



SSSP on Grid Graphs





The path $A_i \longrightarrow A_{i+1}$ induced by $\delta(s,t)$ in subgrid σ is the shortest path between A_i and A_{i+1} in σ.

☆ Assume $M \geq B^2$

☆ Idea: Replace each $B \times B$ subgrid with a complete graph on the "boundary vertices":

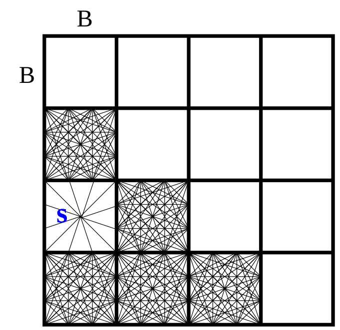
- Edge weight $\longleftrightarrow$ shortest path between the two boundary vertices **in the subgrid**

$\Longrightarrow$ reduced graph G^R: $\Theta(\frac{N}{B})$ vertices, $\Theta(N)$ edges

SSSP on Grid Graphs

Algorithm:

1. Compute SSSP in G^R from s to all boundary vertices
2. For any subgrid σ, for any $t \in \sigma$ then $\delta(s,t) = \min_{v \in Bnd(\sigma)}\{\delta(s,v) + \delta_\sigma(v,t)\}$

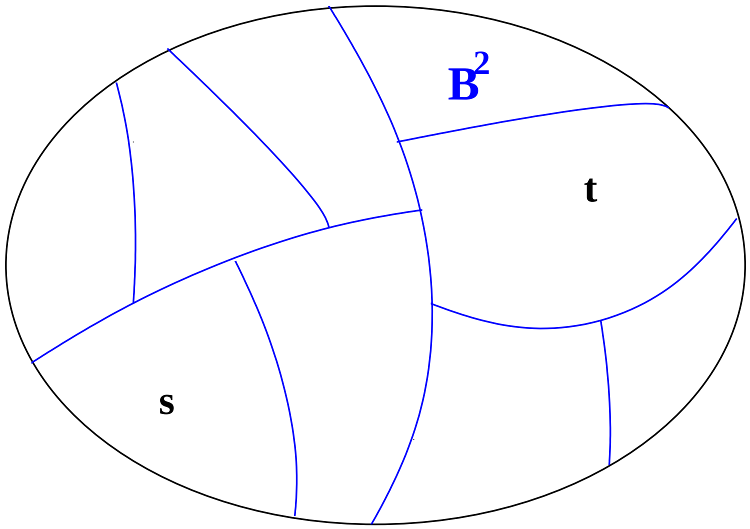


☆ Compute SSSP on G^R

- Use [KS96] $\Longrightarrow O(\frac{N}{B} + \frac{N}{B}\log_2 \frac{N}{B^2})$ I/Os
- Can be improved to $O(\text{sort}(N))$ I/Os
  - ∗ Dijkstra's algorithm, I/O-efficient priority queue
  - ∗ Boundaries of a subgrid can be "blocked" together: load them in $O(1)$ I/Os per vertex

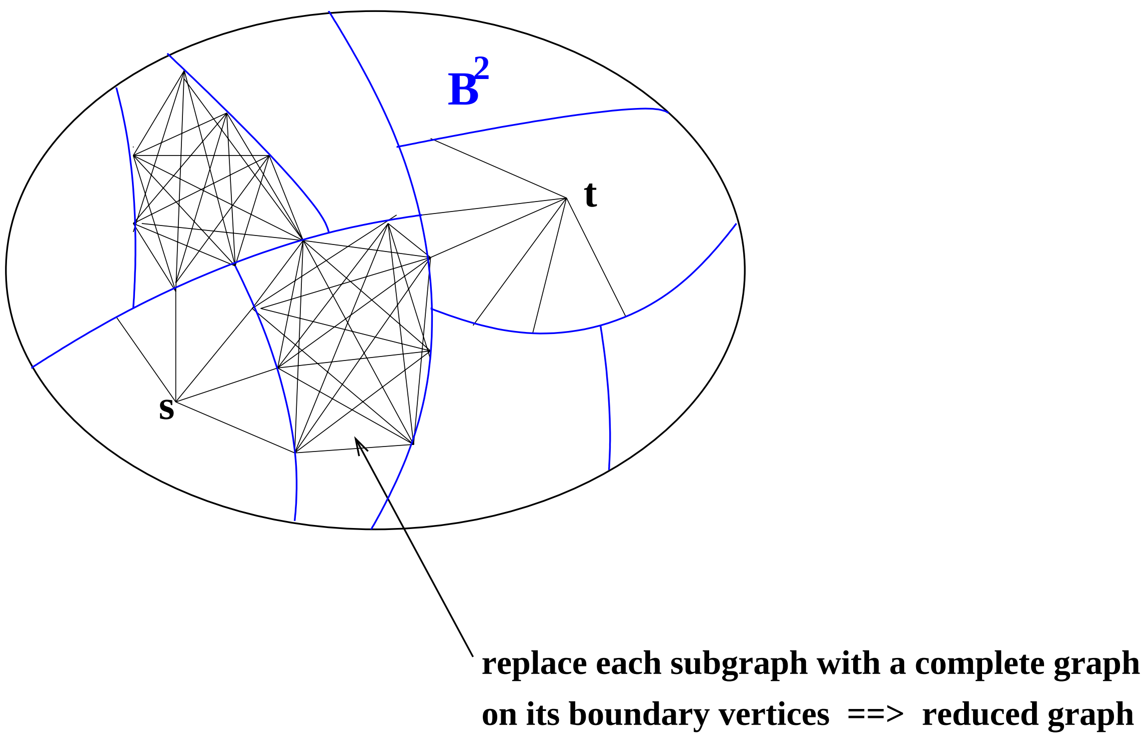
SSSP on Planar Graphs

- ⋆ Similar with grid graphs. Assume $M \geq B^2$, bounded degree.
- ⋆ Assume graph is separated
  - $O(\frac{N}{B^2})$ subgraphs, $O(B^2)$ vertices each, $S = O(\frac{N}{B})$ separator; each subgraph adjacent to $O(B)$ separators



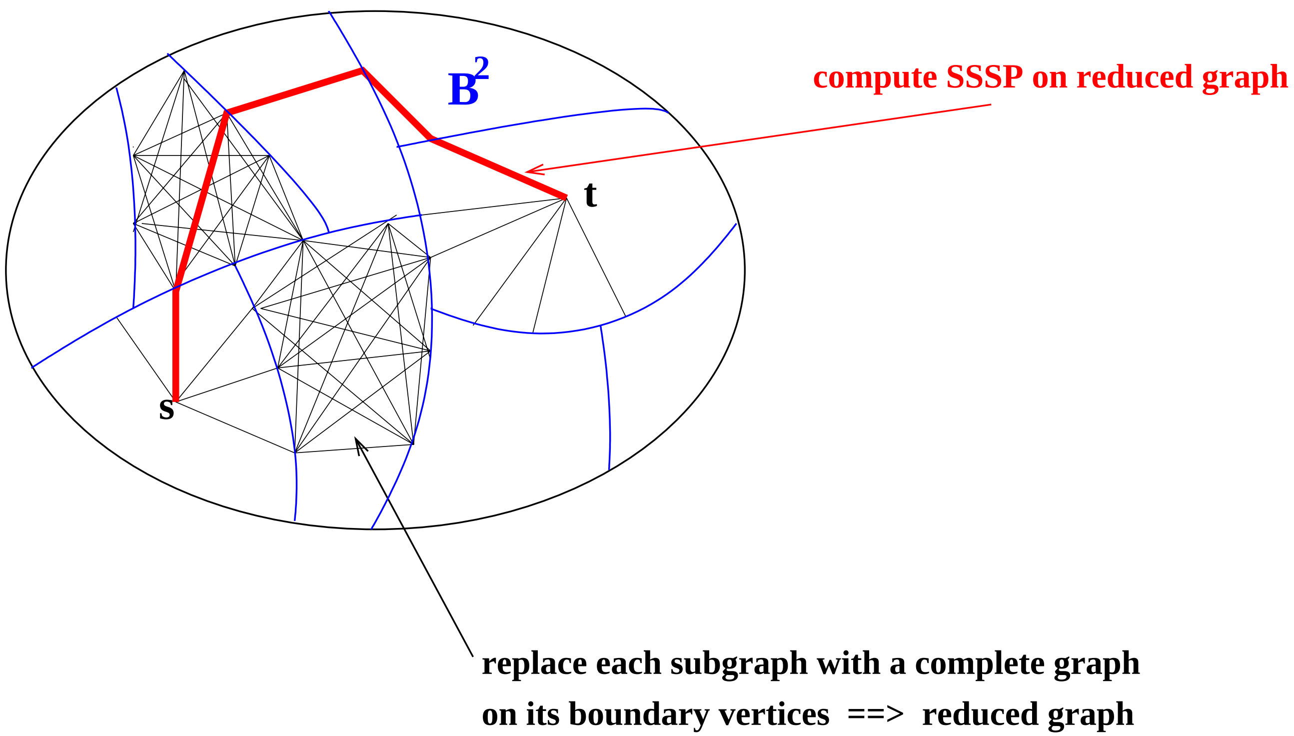
SSSP on Planar Graphs

- ⋆ Similar with grid graphs. Assume $M \geq B^2$, bounded degree.
- ⋆ Assume graph is separated
  - $O(\frac{N}{B^2})$ subgraphs, $O(B^2)$ vertices each, $S = O(\frac{N}{B})$ separator; each subgraph adjacent to $O(B)$ separators



SSSP on Planar Graphs

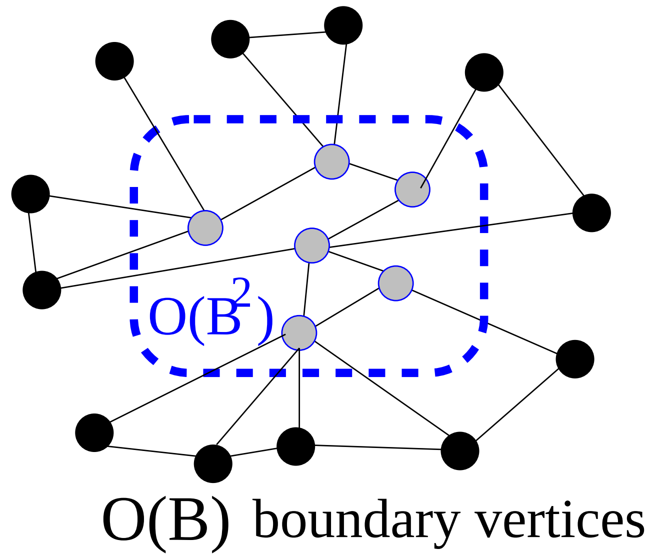
- ☆ Similar with grid graphs. Assume $M \geq B^2$, bounded degree.
- ☆ Assume graph is separated
  - $O(\frac{N}{B^2})$ subgraphs, $O(B^2)$ vertices each, $S = O(\frac{N}{B})$ separator; each subgraph adjacent to $O(B)$ separators



SSSP on Planar Graphs

Reduced graph G^R:

- ☆ $O(S) = O\left(\frac{N}{B}\right)$ vertices
- ☆ $O\left(\frac{N}{B^2} \cdot B^2\right) = O(N)$ edges



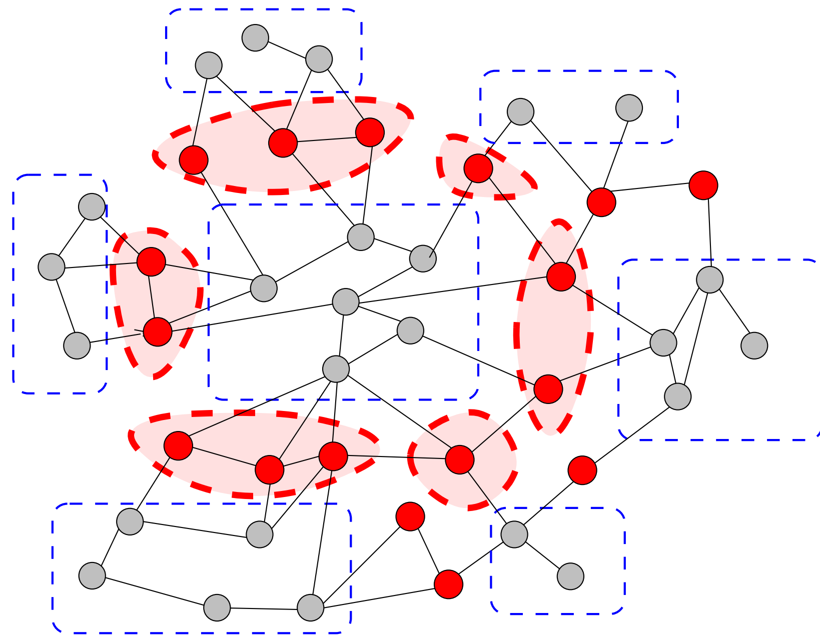
Compute SSSP on G^R

- ☆ Dijkstra's algorithm, I/O-efficient priority queue
- ☆ Keep list $L_S = \{\text{dist}(s, v), \forall v \in S\}$
- ☆ For each vertex, read from L_S its $O(B)$ adjacent boundary vertices

$\Longrightarrow O\left(\frac{N}{B} \cdot B\right) = O(N)$ I/Os (**assume bounded degree**)

SSSP on G^R in $O(\text{sort}(N))$ I/Os

★ $O\left(\frac{N}{B^2}\right)$ boundary sets!



★ Boundary set is $O(B) \Longrightarrow$ load in $O(1)$ I/O

★ Store L_S so that vertices in same boundary set are consecutive

★ Each boundary set is accessed once by its $O(B)$ adjacent vertices
$\Longrightarrow O\left(\frac{N}{B^2} \cdot B\right) = O\left(\frac{N}{B}\right)$ I/Os

Tree Decompositions of Graphs

A tree-decomposition of $G = (V, E)$ consists of

- ☆ A tree T
- ☆ A set $\mathcal{X}$ of sets of vertices of G, one for each node of T

Let X_i denote the vertex set corresponding to a node i of T. Then:

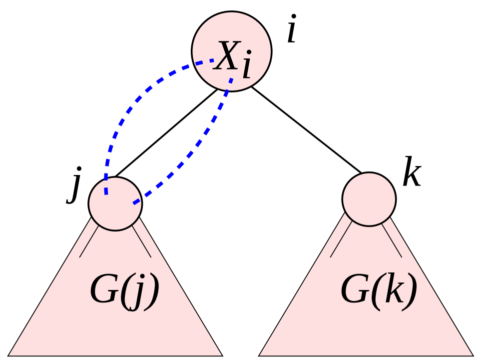
1. $\bigcup_{i \in T} X_i = V$
2. For every edge $(v, w) \in E$, there exists an $i \in I$, so that $v, w \in X_i$, and
3. For two nodes $i, k \in T$ and any node j on the path from i to k in T, $X_i \cap X_k \subseteq X_j$.

The *width* of tree-decomposition is $\max\{|X_i| - 1\}$.

The *treewidth* of a G is the minimum width over all possible tree decompositions of G.

Tree Decompositions

A tree decomposition $\longrightarrow$ Separator decomposition tree



Bounded treewidth graph $\longrightarrow |X_i| = O(1)\ \forall i \in T$

Notation:

- ☆ T_i is subtree of T rooted at i
- ☆ $V(i) = \bigcup_{j \in T_i} X_j$
- ☆ $G(i)$ is subgraph induced by $V(i)$

SSSP on Bounded-treewidth Graphs

General idea:

- ⋆ For each node $i \in T$, store APSP(X_i)
- ⋆ Dynamic programming

1. Find a tree-decomposition of width at most k
2. Bottom-up phase: For each node i in T, compute the shortest distance $d_i(u, v)$ in $G(i)$ between every $u, v \in X_i$ based on the solutions of the children of i.
3. Top-down phase: For each node i in T, compute the shortest distance $d(s, u)$ in G between s and every node $u \in X_i$ based on the solutions of the parent of i.

I/O-efficient: $O(\text{sort}(N))$ I/Os

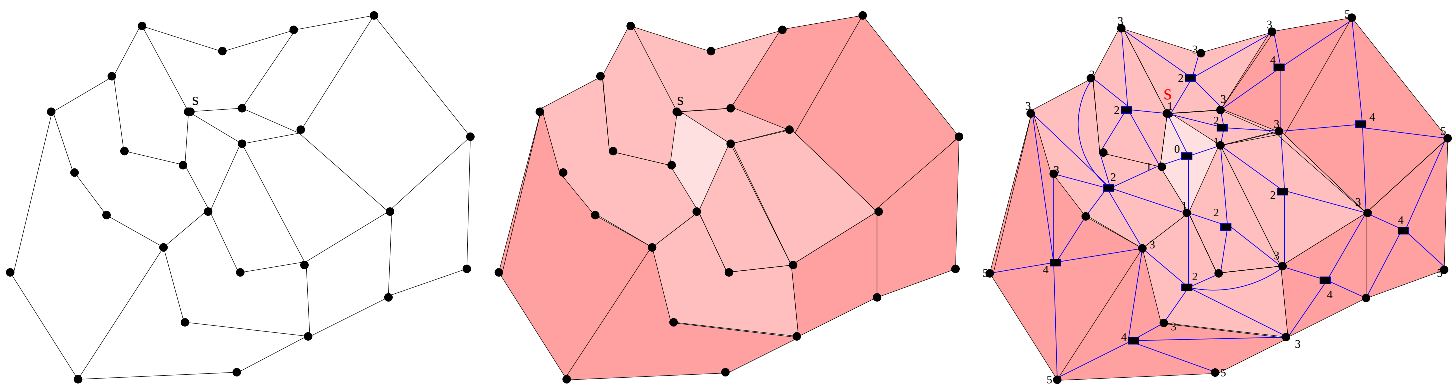
DFS Upper Bounds

sparse	*planar*	*grid*	*outerplanar*	*bounded treewidth*
open	$O(\text{sort}(N))$	open $(O\left(\frac{N}{\sqrt{B}}\right))$	$O(\text{sort}(N))$	open

- ☆ Some DFS tree (not the lexicographically ordered DFS)
- ☆ Planar graphs: reduction to BFS

A DFS to BFS Reduction on Planar Graphs

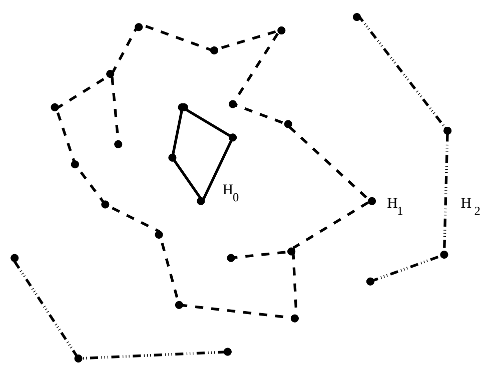
Idea: Partition the faces of G into levels around a source face containing s and grow DFS level-by-level.

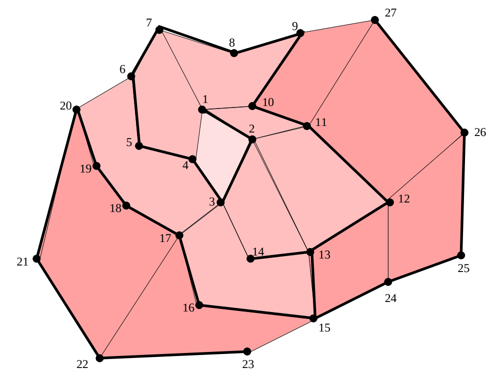


A DFS to BFS Reduction on Planar Graphs

- ☆ G_i = union of the boundaries of faces at level $\leq i$
- ☆ T_i = DFS tree of G_i
- ☆ $H_i = G_i \setminus G_{i-1}$

Compute a spanning forest of H_i and attach it onto T_{i-1}.





Lemma: The bicomps of H_i are the boundary cycles of G_i.

Lemma: A spanning tree is a DFS tree if and only if it does not have cross edges.

A DFS to BFS Reduction on Planar Graphs

I/O-analysis

☆ Compute CC of H_i': $O(\text{sort}(|H_i|))$ I/Os

☆ Compute DFS of H_i'

- compute bicomps, bicomp-cut-point tree, tree DFS: $O(\text{sort}(|H_i|))$ I/Os

☆ Find deepest node in T_{i-1} which connects to H_i'

$\Longrightarrow$ total $O(\text{sort}(N))$ I/Os