Time Table Information
and
Dynamic All-Pairs Shortest Paths

Christos D. Zaroliagis

Computer Technology Institute

&

Dept of Computer Eng & Informatics
University of Patras, Greece

- Timetabling: reduces to ≥ 1 *shortest path models*.

- In any shortest path model:

  - Underlying graph G models train changes/connections within/between stations.

  - G may vary with time due to: nonavailability of services at certain days, delays, introduction of new services, etc.

- Dynamic changes in G: how can timetable information be efficiently recomputed ?

$\Downarrow$

- *Dynamic all-pairs shortest paths*

Dynamic Algorithm: maintains a certain property (e.g., connectivity) of a graph that changes dynamically over time.

Goal: maintain the desired property without recomputing everything from scratch.

DA = Data Structure + Operations (Query, Update)

Update $\in$ {edge insertion, edge deletion, node insertion, node deletion}

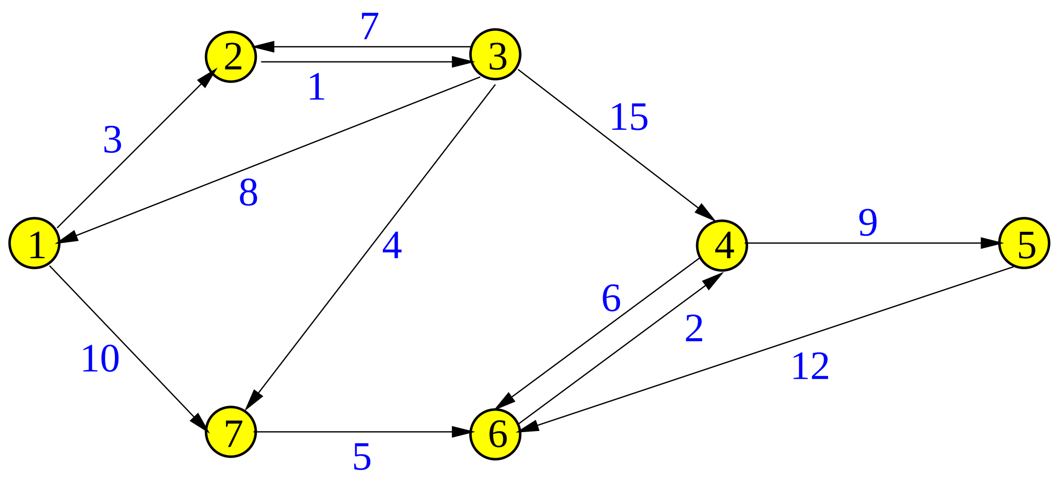
Dynamic Algorithm/Problem

- Partially dynamic:
  - Insertions only $\Rightarrow$ Incremental

    OR

  - Deletions only $\Rightarrow$ Decremental

- Fully dynamic: both insertions and deletions are allowed.



$$G = (V, E), \qquad wt : E \to \mathbb{R}; \qquad n = |V|,\ m = |E|$$

Shortest Path: find a path from u to v of minimum total weight.

(Shortest) Distance: weight of a shortest u-v path $[d(u, v)]$.

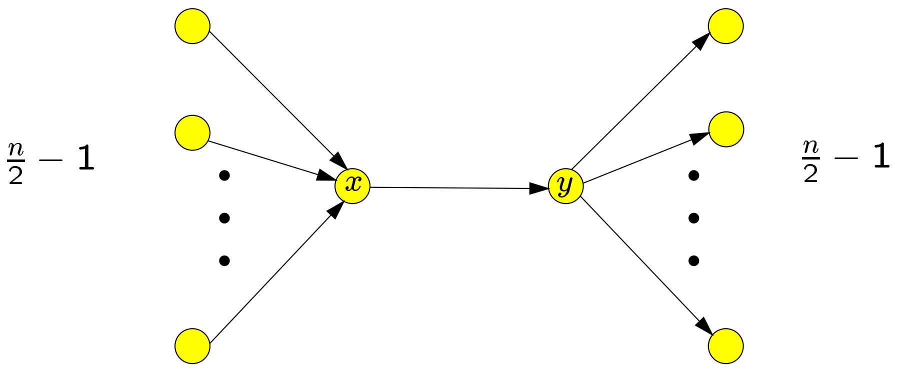
All-Pairs Shortest Paths (APSP): $\forall u, v \in G$, find a shortest u-v path.

Remainder of this Talk

- Dynamic APSP: why is difficult ?
- General digraphs
  - Best algorithms
  - Recent fully dynamic algorithm
- Special digraphs: much better algorithms
  - Planar digraphs
  - Small-Treewidth digraphs
- Conclusions

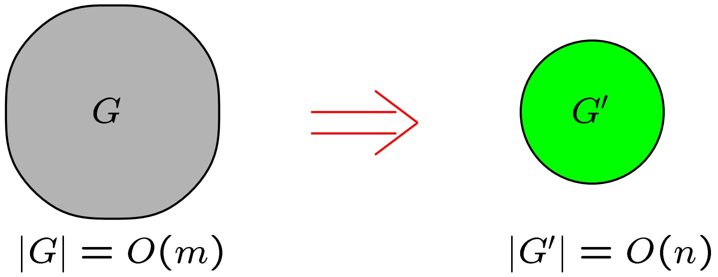
Dynamic APSP: Why is it difficult ?

- Change of an edge weight may modify many distances
  e.g., $\Omega(n^2)$



- Application of efficient techniques for undirected graphs is ruled out in the case of digraphs; e.g., sparsification

  [Eppstein,Galil,Italiano & Nissenzweig,92]



Sparse Certificate:

- usually easy to find in undirected graphs
- very difficult to find in digraphs

  [Khanna,Motwani & Wilson,97]

Best Bounds for General Digraphs

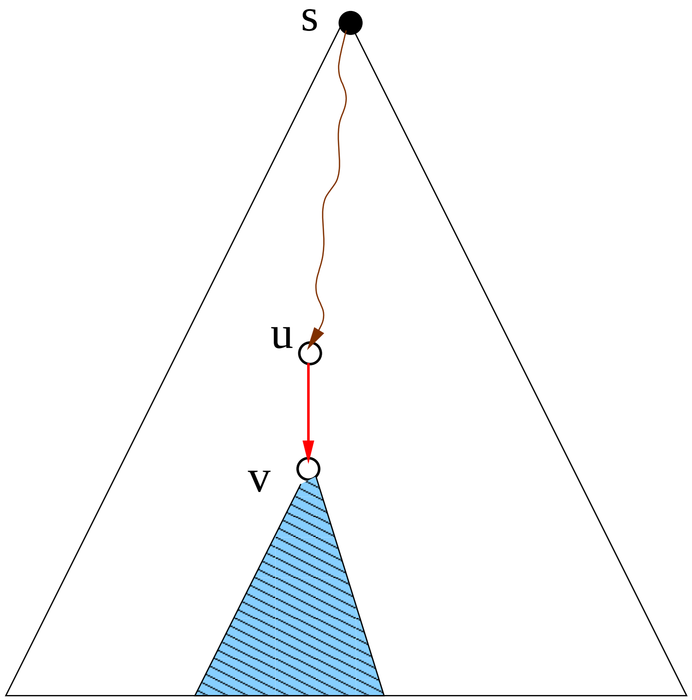
	[Even & Gazit,85]	[Ausiello et al,91]	[King,99]	[King,99]
Initial.	$n^2 \log n + nm$	n^2	$n^{2.5}\sqrt{C \log n}$ $+ nm$	$n^{2.5}\sqrt{C \log n}$ $+ nm$
Query	1	1	1	1
Insertion	n^2	$Cn \log(nC)$	—	$n^{2.5}\sqrt{C \log n}$
Deletion	$n^2 \log n + nm$	—	$n^2 C$	$n^{2.5}\sqrt{C \log n}$

Edge weights in $[1..C]$; amortized bounds

Decremental and Fully Dynamic APSP

- *Building block:* maintaining a single-source shortest path tree for distances up to Δ in a decremental environment.

Maintaining a SSSP tree T – main idea



- When (u, v) is deleted, look at the incoming edges of v. If the deletion can be repaired, then we are done.

- Otherwise, mimic Dijkstra's algorithm to reinsert to T the vertices in the subtree of v: a vertex is added to T if its distance from s is minimal among those not in T.

A vertex w can be

- *settled:* joined to s by tree edges
- *uncertain:* examined but not yet determined whether $d(s,w)$ has changed.

$$\ell(w) = \begin{cases} d(s,w) & \text{if } d(s,w) \le \Delta \\ \infty & \text{otherwise} \end{cases}$$

$$\textit{predlist}(w) = \{z : (z,w) \text{ is not a tree edge, } z \text{ is either settled or uncertain}\}$$

Delete (u, v)

1.Remove (u, v) from T;

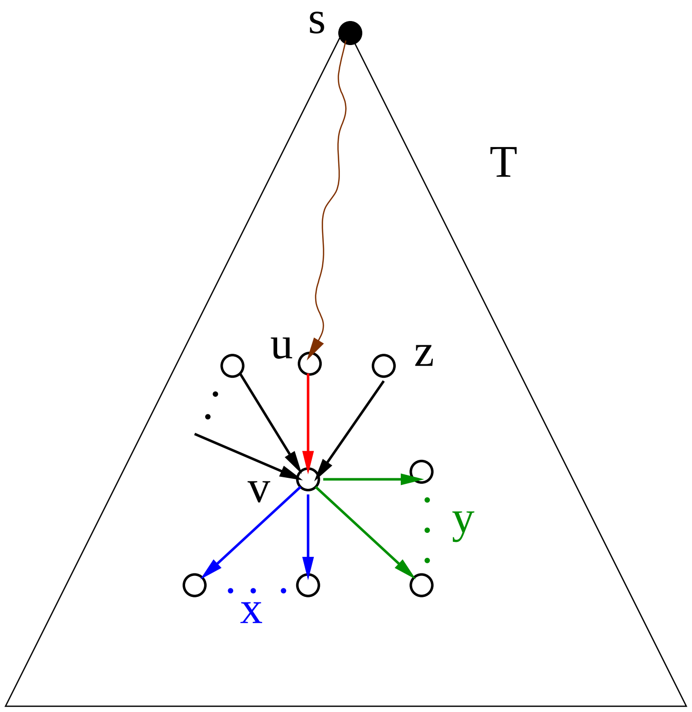
2.Find z minimizing $\ell(z) + wt(z, v) \equiv d(v)$;

3.**If** $d(v) = \ell(v)$ **then** add (z, v) to T,
and remove z from *predlist*(v);

If v is not uncertain **then** $\ell(v) = d(v)$
$\forall(v, w) \in E$, insert v into *predlist*(w);

Else **for** each non-tree edge (v, y) **do**
remove v from *predlist*(y);

for each tree edge (v, x) **do**
remove (v, x) from T and repeat;



Delete (u, v)

1. Remove (u, v) from T;

1a. Add v to a heap H with *key*$(v) = \ell(v)$;

Repeat until $H = \emptyset$:

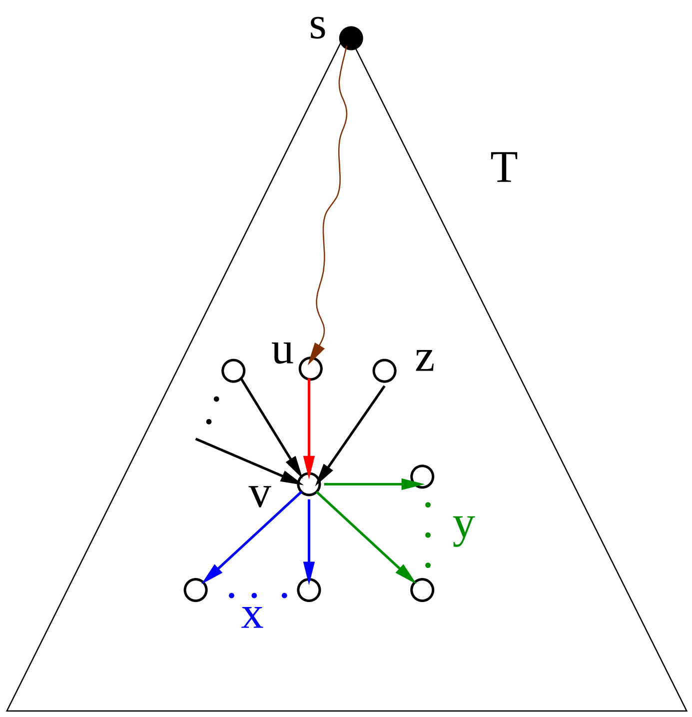
1b. $v =$ *del-min*(H);

2. Find z minimizing $\ell(z) + wt(z, v) \equiv d(v)$;

3. **If** $d(v) = \ell(v)$ **then** add (z, v) to T,
   and remove z from *predlist*(v);
   If v is not uncertain **then** $\ell(v) = d(v)$
   $\forall (v, w) \in E$, insert v into *predlist*(w);

   Else for each non-tree edge (v, y) **do**
   remove v from *predlist*(y),
   and add v to H with *key*$(v) = d(v)$;

   for each tree edge (v, x) **do**
   remove (v, x) from T,
   and add x to H with *key*$(x) = \ell(x)$;



Time complexity:

- Cost of a single deletion:

$$O\left(\left(\sum_v deg(v)\right) \cdot \log \Delta\right) = O(m \log \Delta)$$

- Over a sequence of deletions, $\ell(v)$ can increase no more than Δ times

$$\Downarrow$$

$$\text{Total cost} = \sum_v O\left(deg(v) \cdot \Delta \cdot \log \Delta\right) = O(m\Delta \log \Delta)$$

Improving the time complexity:

- Use *buckets* instead of heaps !

  predlist$(v) \to A[1..\Delta]$; *predlist*$(v)[i] = \{z : \ell(z) + wt(z,v) = i\}$

  Heap $H \to H'[1..\Delta]$; $H'[i] = \{\text{unsettled } w : \ell(w) = i\}$

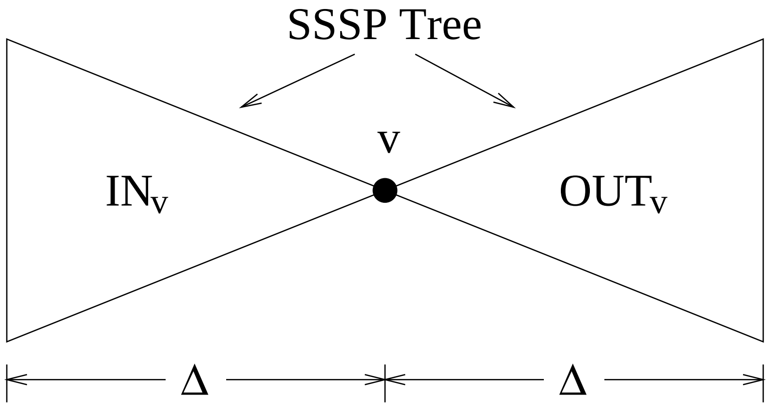
- Proceed ala Dial: examine all possible values $\ell(v) \leq L \leq \Delta$.

- $$\text{Total cost} = O\left(n\Delta + \Delta \sum_v deg(v)\right) = O(m\Delta)$$

Summing-up: Decremental APSP

- $\forall v \in V$, maintain a SSSP tree up to distances $\Delta = nC$.
- Total cost: $O(n \cdot m(nC)) = O(mn^2C)$
- For $\Omega(m)$ deletions $\Rightarrow$ amortized cost $= O(n^2C)$

Fully dynamic APSP – pairs up to distance Δ



- $\mathit{list}(u, w, k) = \{v : IN_v(u) + OUT_v(w) = k\}$
- $\mathit{count}(u, w, k) = |\mathit{list}(u, w, k)|$
- $D(u, w) = \begin{cases} \min\ k & \text{s.t. } \mathit{count}(u, w, k) > 0 \\ \infty & \text{otherwise} \end{cases}$

Initialization

1. $\forall v \in V$, maintain OUT_v.

2. Set *count*$(v,w,k) = \begin{cases} 1 & \text{if } OUT_v(w) = k \\ 0 & \text{otherwise} \end{cases}$

3. Initialize *list*(v,w,k) and $D(v,w)$ accordingly.

Total cost: $O(n(n\Delta + m) + n^2\Delta) = O(nm + n^2\Delta)$

Insert(E_v) [*inserts a set of edges incident to* v]

1. Delete IN_v and OUT_v, if exist.

2. Build and maintain new IN_v and OUT_v.

3. For each u, w, k, adjust *count*(u, w, k), *list*(u, w, k) and $D(u, w)$ accordingly.

Total cost:

$$\left.\begin{array}{rl} (1)\&(2): & O(m\Delta) \\ (3): & O(n^2\Delta) \text{ over all modifications} \end{array}\right\} \Rightarrow O(n^2\Delta)$$

Delete(E') [*deletes an arbitrary subset* E' *of edges*]

1. $\forall e \in E'$, delete it from all IN and OUT trees it is contained.

2. Each time a vertex u "moves down" in an OUT (IN) tree, adjust accordingly *count*(u, w, k), *list*(u, w, k), and $D(u, w)$ (*count*(w, u, k), *list*(w, u, k), and $D(w, u)$).

Total cost:

$$\left.\begin{array}{ll} (1): & O(nm\Delta) \\ (2): & O(n^2\Delta) \end{array}\right\} \Rightarrow O(nm\Delta)$$

Query(u, w)

- Distance: return $D(u, w)$.

- Shortest path: let v be a vertex in *list*$(u, w, D(u, w))$. Then, return the u-v path in IN_v and the v-w path in OUT_v.

Summing-up: fully dynamic APSP – distances up to Δ

▶ Total cost for R deletions, I insertions, Q queries:

$$O(nm\Delta + (R+I)n^2\Delta + Q)$$

▶ For a sequence of length $\Omega(m/n)$, we get an amortized cost per update:

$$O\left(\frac{nm\Delta + (m/n)n^2\Delta + (m/n)}{m/n}\right) = O(n^2\Delta)$$

Fully Dynamic APSP – arbitrarily long distances

- Determine a subset of vertices that “blocks” all shortest paths up to some distance.

- Find shortest path information between the vertices of the *blocker*, and between the rest of the vertices and the *blocker*.

Unweighted case

- Let F be a forest of IN_v and OUT_v trees of depth Δ, $v \in V$.
- A subset $S \subset V$ is a *blocker* for F if every directed path of length exactly Δ that starts (ends) at the root of an OUT (IN) tree contains a vertex in S which is distinct from the root.
- A blocker S for F of size $O((n/\Delta)\log n)$ can be constructed in $O(n^2 + n\Delta \log n)$ time.

Algorithm

1. $\forall v \in V$, construct and maintain a forest F of IN_v and OUT_v trees of depth Δ. $[O(n^2\Delta)]$

2. Run the *stitching procedure* :

   2.1 Find a blocker S for F. $[O(n^2 + n\Delta \log n)]$

   2.2 Let $G' = (S, E^s)$, $E^s = \{(x, y) : x, y \in S\}$, and $wt(x, y) = D(x, y)$. Run a static APSP algorithm on G' and let $SS(x, y)$ the shortest x-y distance in G'. $[O(|S|^3)]$

   2.3 $\forall v \in V, s \in S$ do: if $D(u, s) = \infty$, then $[O(n|S|^2)]$

   $$D(u, s) = \min_{s' \in S}\{D(u, s') + SS(s', s)\}$$

   2.4 $\forall u, w \in V$ do: if $D(u, w) = \infty$, then $[O(n^2|S|)]$

   $$D(u, w) = \min_{s \in S}\{D(u, s) + D(s, w)\}$$

Time complexity for updates:

$$\left.\begin{array}{ll} (1): & O(n^2\Delta) \\ (2): & O(n^2|S|) \end{array}\right\} \Rightarrow O(n^2|S|) = O(n^2(n\log n/\Delta)) = O(n^2\Delta)$$

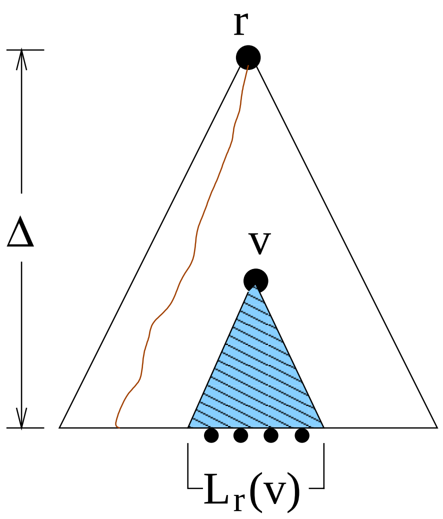
Hence, $\Delta = |S| = O(\sqrt{n\log n})$

$$\Downarrow$$

Amortized cost $= O(n^{2.5}\sqrt{\log n})$

Finding a blocker S

$L = \sum_r |L_r(r)|$; $score(v) = \sum_r |L_r(v)|$



Repeatedly find vertex of maximum score, add it to S, and remove it along with its subtrees from every tree in which it appears.

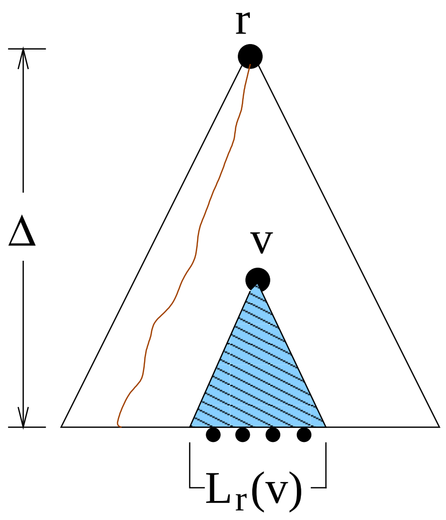
- Computing scores: traverse each tree in postorder; *score(parent)* $= \sum$ *score(child)*.

- Revising scores:

  $\forall w$ in v's subtree: $score(w) = score(w) - |L_r(w)|$

  $\forall$ ancestors x of v: $score(x) = score(x) - |L_r(v)|$

$|S| = \#$ iterations



- Progress depends on the # remaining leaves L': $L' = L - (\#$ paths containing $v)$

- n elements (vertices) and L subsets (root-to-leaf paths) each of size $\Delta \Rightarrow \exists$ some element (v) which belongs to $\geq L\Delta/n$ subsets (paths).

- $L' \leq L - \frac{L\Delta}{n} = L(1 - \frac{\Delta}{n})$.

- # remaining leaves after i iterations $\leq$

$$L\left(1 - \frac{\Delta}{n}\right)^i \leq Le^{-i\Delta/n} = 1 \Longleftrightarrow i = n \ln L/\Delta$$

Weighted Case

- Blocker now has size $O(Cn \log n / \Delta)$.
- Cost of stitching procedure is $O(Cn^3 \log n / \Delta)$ which has to match the $O(n^2 \Delta)$ term.
- Choosing $\Delta = |S| = O(\sqrt{nC \log n})$, we get

$$\text{Amortized cost} = O(n^{2.5} \sqrt{C \log n})$$

Can we do better in special cases?

e.g., train networks are almost planar

electrical networks $\approx$ series-parallel graphs

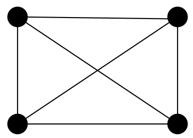
- Sparse digraphs: $m = O(n)$

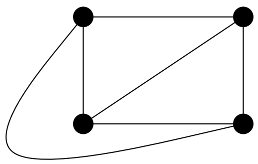
  - are encountered very often both in theory and in practice

  - have nice structural properties

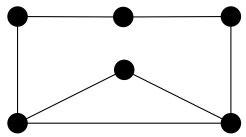
Sparse digraphs – three interesting cases

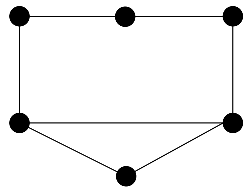
Planar





Outerplanar





Bounded Treewidth $\supset$ Series-Parallel $\supset$ Outerplanar

Best Bounds for Planar and Small-Treewidth Digraphs

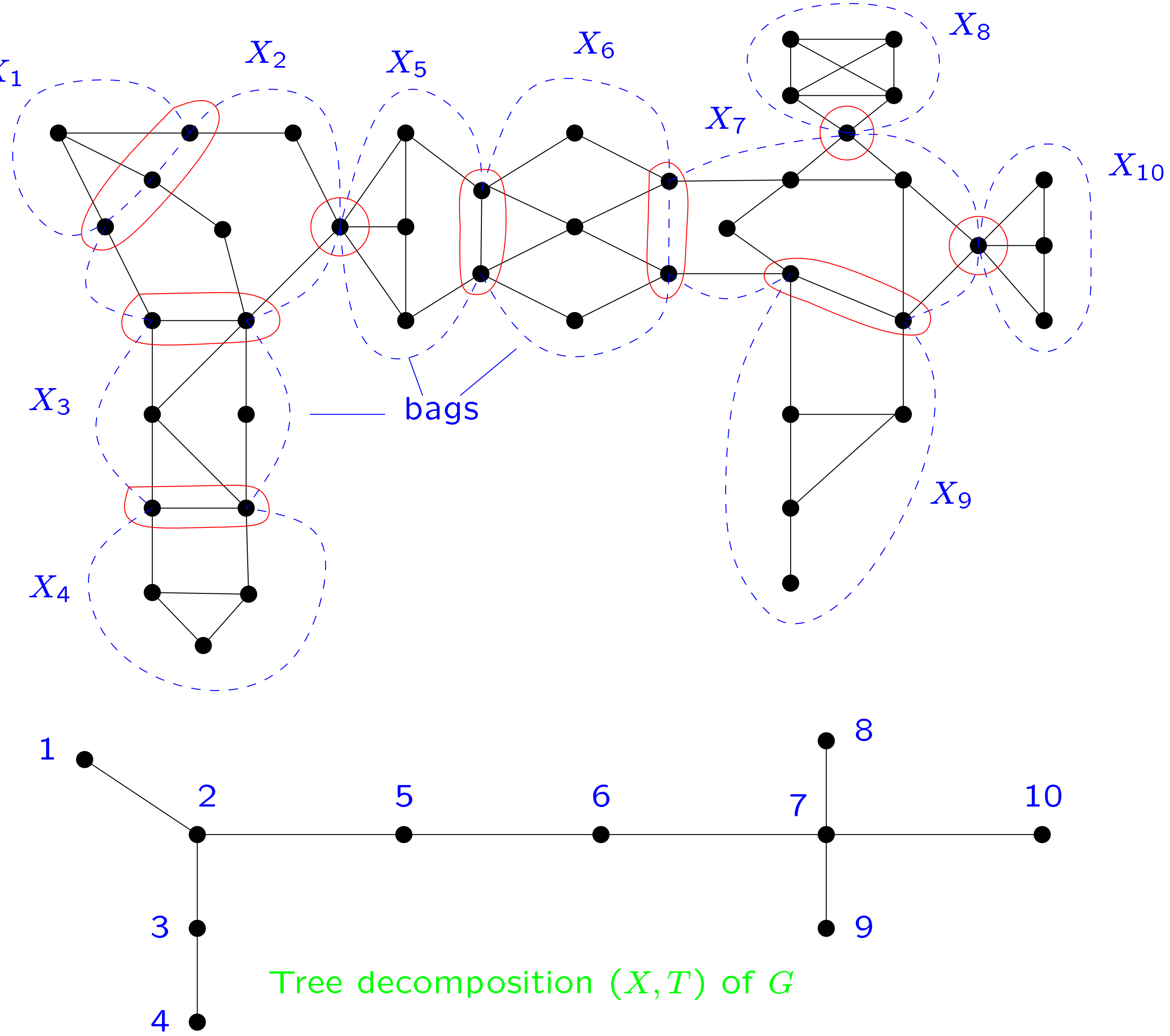
	Planar [Djidjev, Pantziou & Zaroliagis,00]	Small-Treewidth [Chaudhuri & Zaroliagis,95]	[Hagerup,97]
Initial.	$O(n)$	$O(n)$	$O(n \log^* n)$
Query	$O(\log n + q)$	$O(\alpha(n))$	$O(1)$
Insertion	$O(n)$	$O(n)$	$O(n \log^* n)$
Deletion & Cost update	$O(\log n)$	$O(n^\beta)$	$O(n^\beta)$

$q :=$ minimum number of faces that cover all vertices; $1 \leq q < n$

$0 < \beta < \frac{1}{2}$; $\quad \log^* n \leq 5, \quad \forall n \leq 2^{65536} \approx 10^{19730}$

$\alpha(n) \leq 6, \quad \forall n \leq 10^{10^{19720}}$

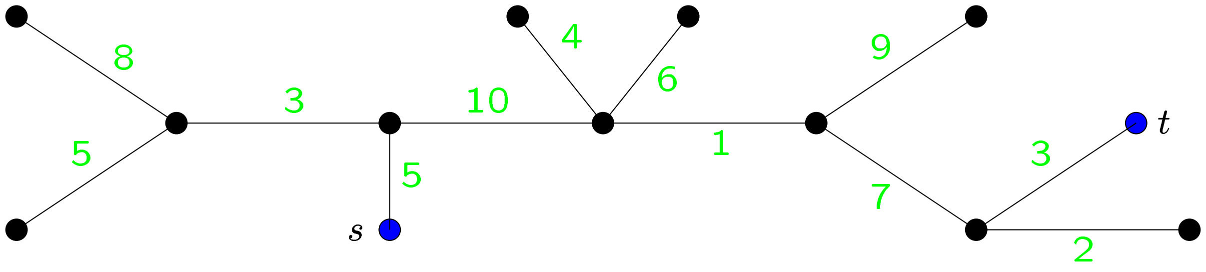
Small-Treewidth Digraphs



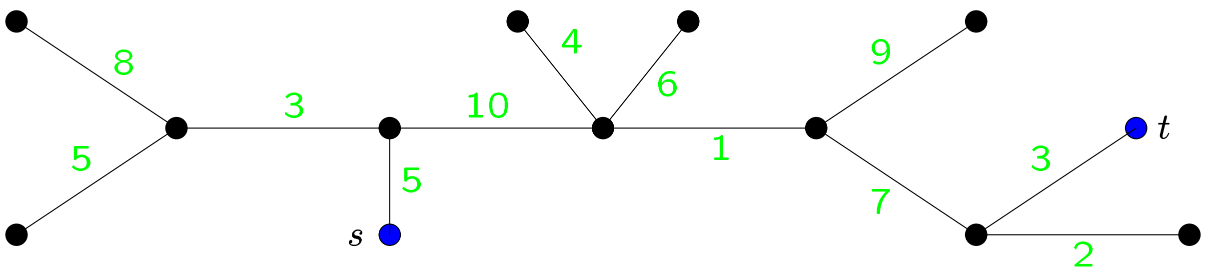
Tree decomposition (X,T) of G

Width := size of largest bag Treewidth := $\min_{\text{all } (X,T)}$ { width }

Vertices shared by 2 bags : Separator of G



- $d(s,t)$ = sum of edge-weights in the s-t path



- [Alon & Schieber 87, Chazelle 87]:

  → Given tree T with values on its nodes

  → associative operator $\oplus$ such that $a \oplus b$ is evaluated in $O(1)$ time

  → query(s,t): find the product of node values in the s-t path

  ⇒ After $O(n)$ (resp. $O(n \log^* n)$) preprocessing, a query can be answered in $O(\alpha(n))$ (resp. $O(1)$) time.

Main Idea

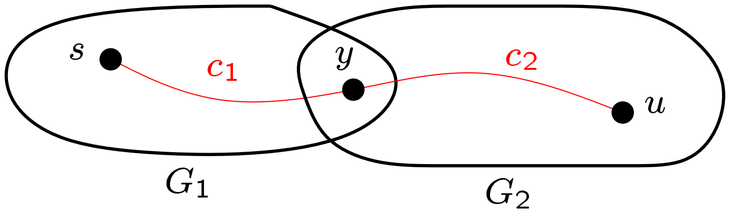
1. Work with the tree decomposition (X, T)

   (it can be found in $O(n)$ time [Bodlaender,93])

2. Define a certain value for each node of T

3. Define an associative operator on these values

4. Reduce the shortest path problem to computing products of these values along paths of T



(s, y, c_1) (y, u, c_2)

$$(s, u, c_1 + c_2)$$

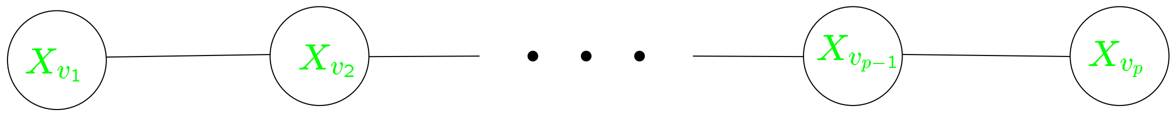
$$X, Y \subseteq V(G) \,:\; P(X, Y) = \{(x, y, d(x, y)) : x \in X, y \in Y\}$$

$X, Y \subseteq V(G)\ :\ P(X,Y) = \{(x, y, d(x,y)) : x \in X, y \in Y\}$

$P(X,Y) \oplus P(Z,W) \;=\; \{(x, w, d) : x \in X, w \in W, d=$ weight of a sh. x-w path including a vertex in $Y \cap Z\}$

Define: $\gamma(i) = P(X_i, X_i)$

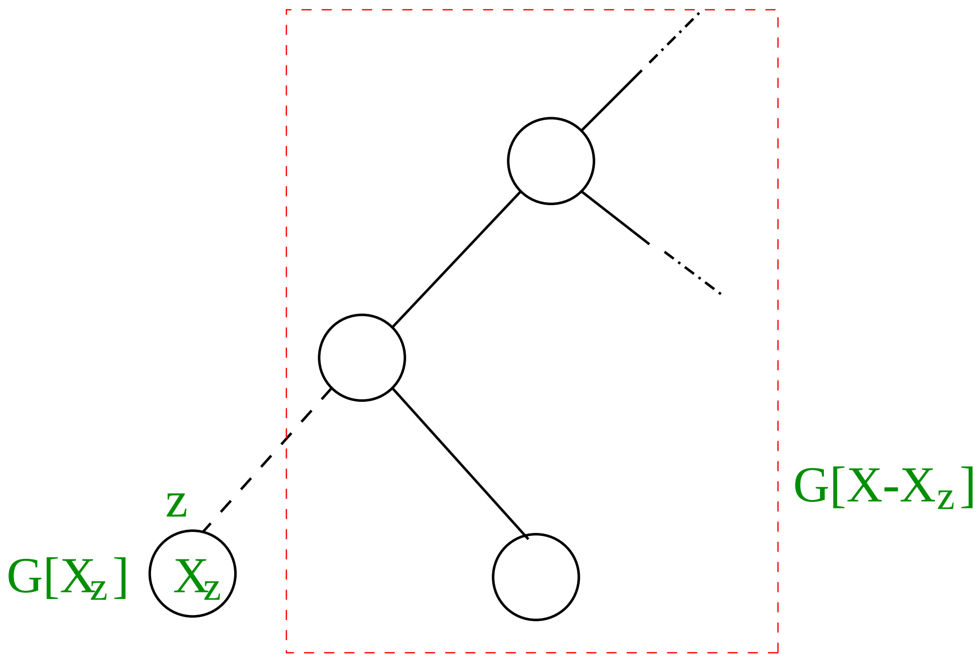
$\gamma(i) \oplus \gamma(j) = P(X_i, X_i) \oplus P(X_j, X_j) = \{(s, u, d) : s \in X_i, u \in X_j,$ $d =$ weight of a sh. s-u path through $y \in X_i \cap X_j\}$



Induction: $\gamma(v_1) \oplus \gamma(v_2) \oplus \cdots \oplus \gamma(v_p) = P(X_{v_1}, X_{v_p})$

How the values $\gamma(i)$ are computed ?

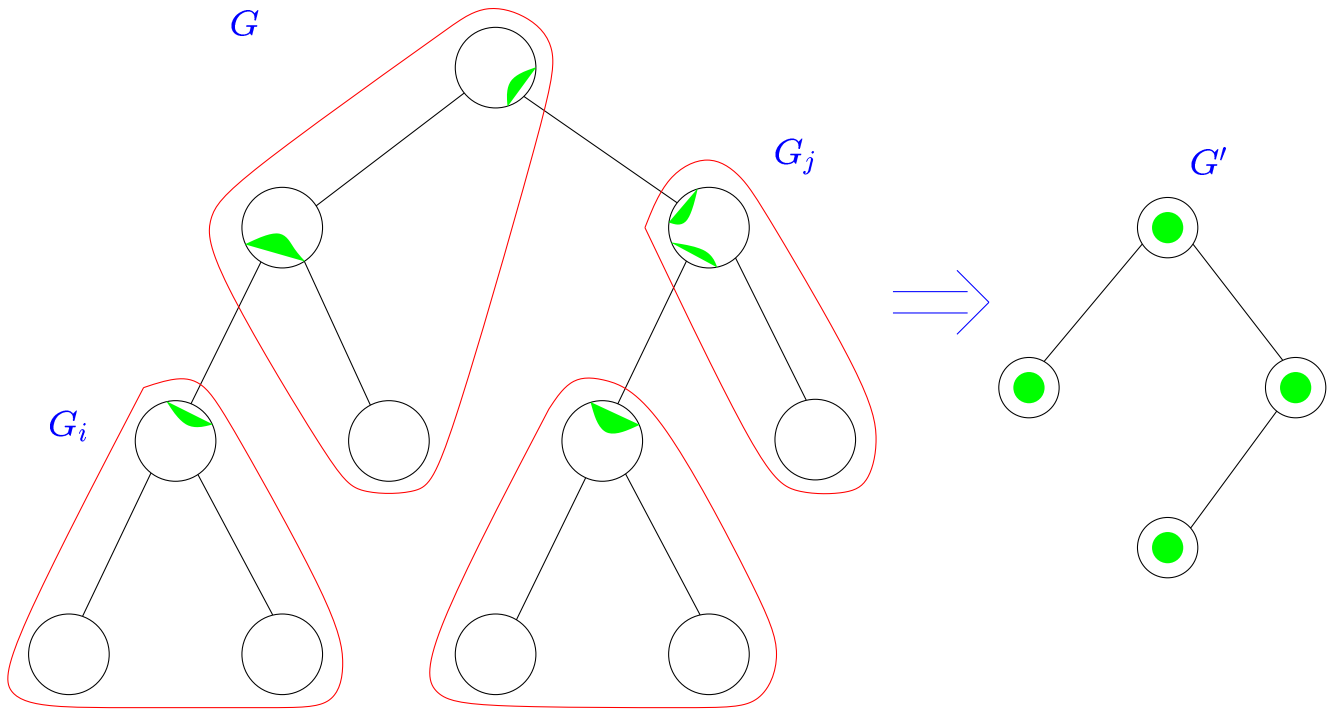
- Compute for each $u, v \in X_i$, $i \in T$, $d(u,v)$



1. Compute APSP in $G[X_z]$.
2. "Absorb" $G[X_z]$ into $G[X - X_z]$.
3. Recurse on $G[X - X_z]$.
4. Recompute APSP in $G[X_z]$ using the APSP info in $G[X - X_z]$.

Time: $O(t^3 n)$

Dynamization



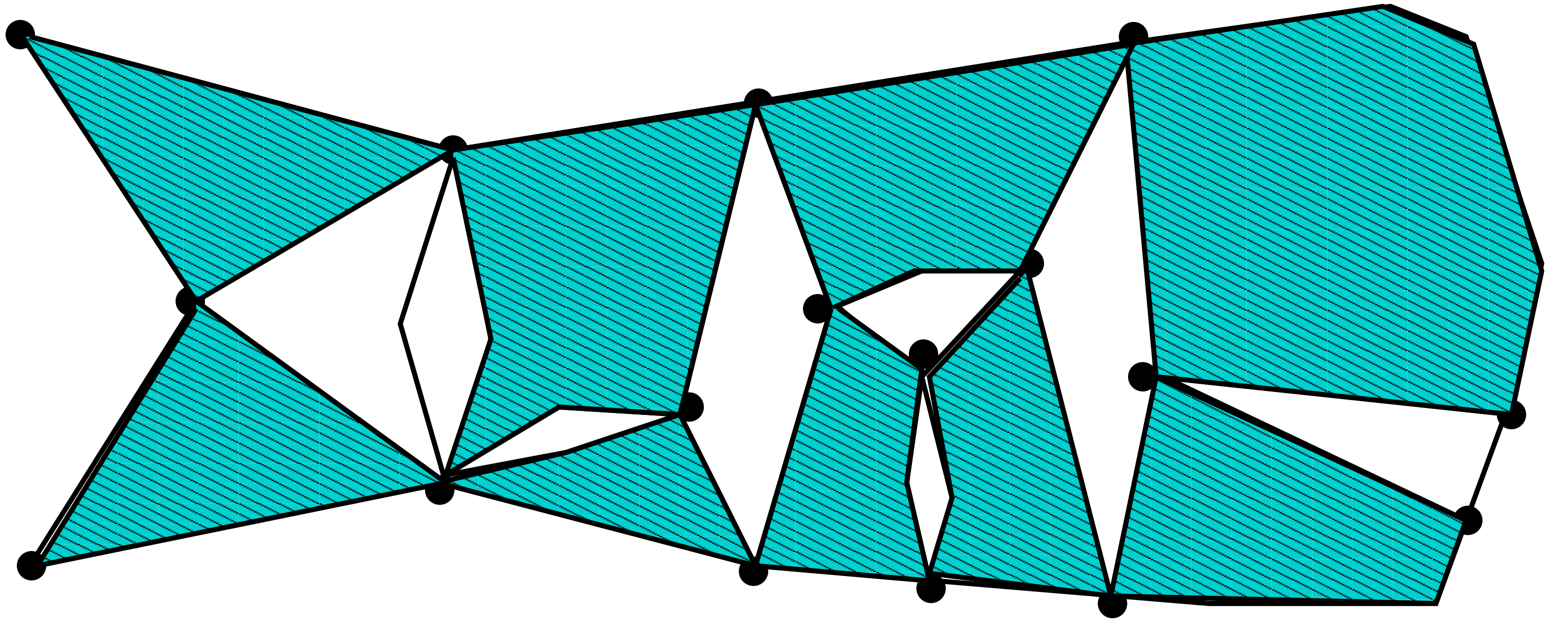
1. Divide G into $O(\sqrt{n})$ subgraphs G_i :
   $|C(G_i)| \leq 3t$, where $C(G_i) \equiv \bullet \leftarrow$ cut-set
   $|G_i| = O(\sqrt{n})$; $E(G_i) \cap E(G_j) = \emptyset$

2. Construct reduced graph G', $|G'| = O(\sqrt{n})$.

3. Construct data structures in every G_i and in G'.

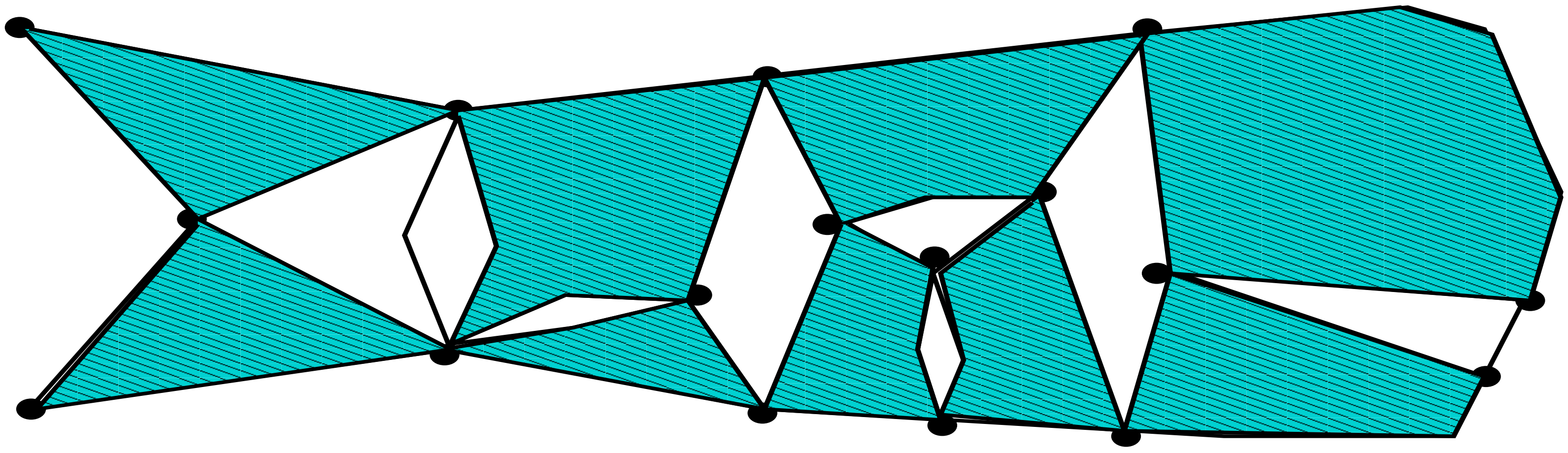
4. Recurse.

Planar Digraphs



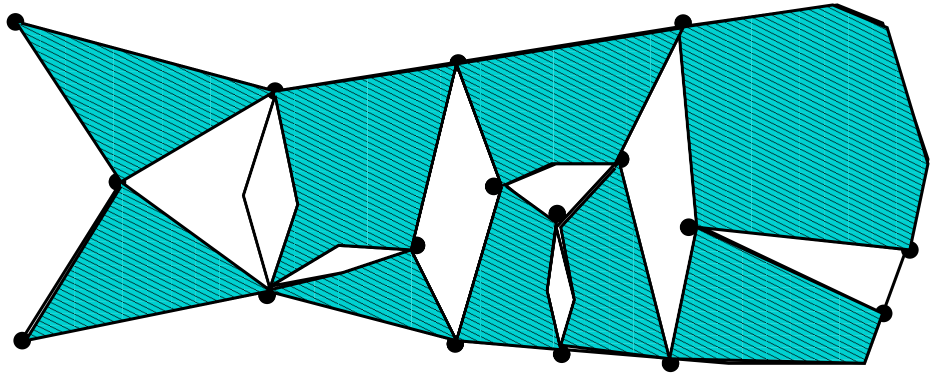
- Hammock decomposition [Frederickson,91]
  - Planar graph = union of *hammocks*
  - *Hammocks:* edge-disjoint, outerplanar graphs;
    $\leq$ 4 separator vertices (*attachments*)
  - q = number of hammocks

1. Compute the hammock decomposition.

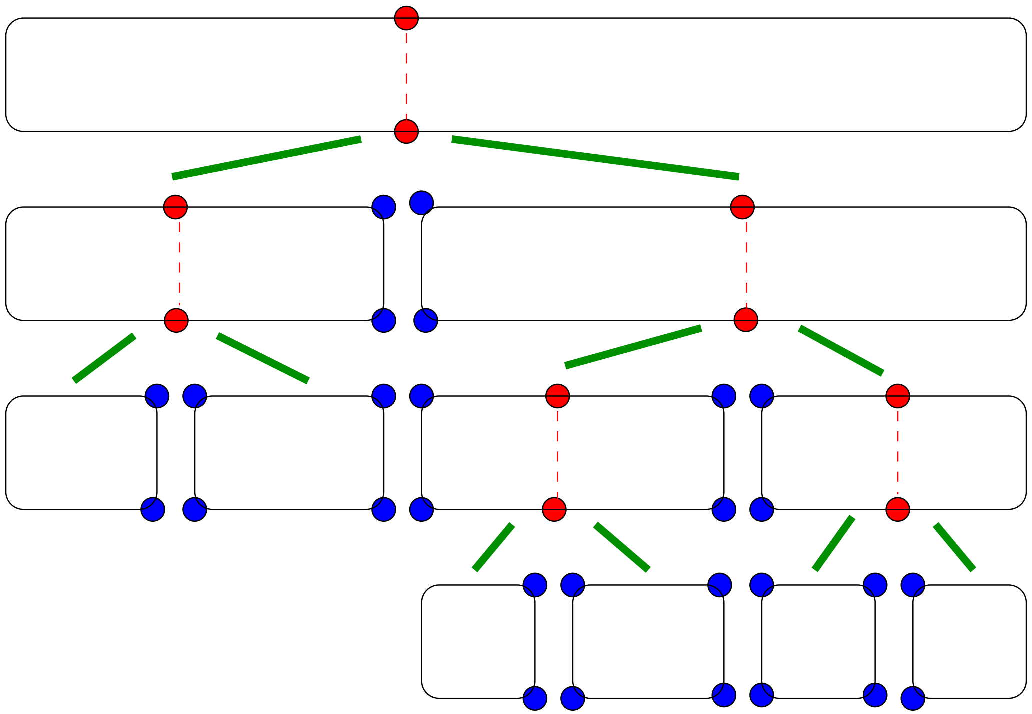


2. Maintain APSP in each hammock H.

3. Compress each H w.r.t. its separator vertices into an $O(1)$-sized outerplanar graph $\Longrightarrow$ planar G' of size $O(q)$.



Dynamic APSP in Outerplanar digraphs



- Recursive separator decomposition.
- Shortest path info between separator vertices is maintained locally and is encoded into *sparse certificates*.
- Query: distances between relevant separator vertices are computed and passed into the appropriate local subgraph.
- Update: each edge belongs to at most two leaf subgraphs.

Summing-up: Planar case

- Preprocessing $O(n)$

- Query $\left\{\begin{array}{r} \text{distances inside hammocks:} \quad O(\log n) \\ \text{distances among attachments:} \quad O(q) \qquad \end{array}\right\} O(\log n + q)$

- Update $\left\{\begin{array}{r} \text{in the hammock:} \quad O(\log n) \\ \text{in } G'\text{:} \quad O(1) \qquad \end{array}\right\} O(\log n)$

Conclusions

- Dynamic APSP
  - Difficult problem.
  - Existing algorithms: not satisfactory bounds.
  - General case *still* a challenge.
- Graphs with structure
  - Better algorithms exist for some interesting cases.
  - Further *good* results *?*