# Low-Port Tree Representations

Presented by: Shiri Chechik
Joint with: David Peleg

# Background

- Many distributed applications make use of predefined network representations:
  - Compact routing schemes
  - Informative labeling schemes for a variety of applications

# Background

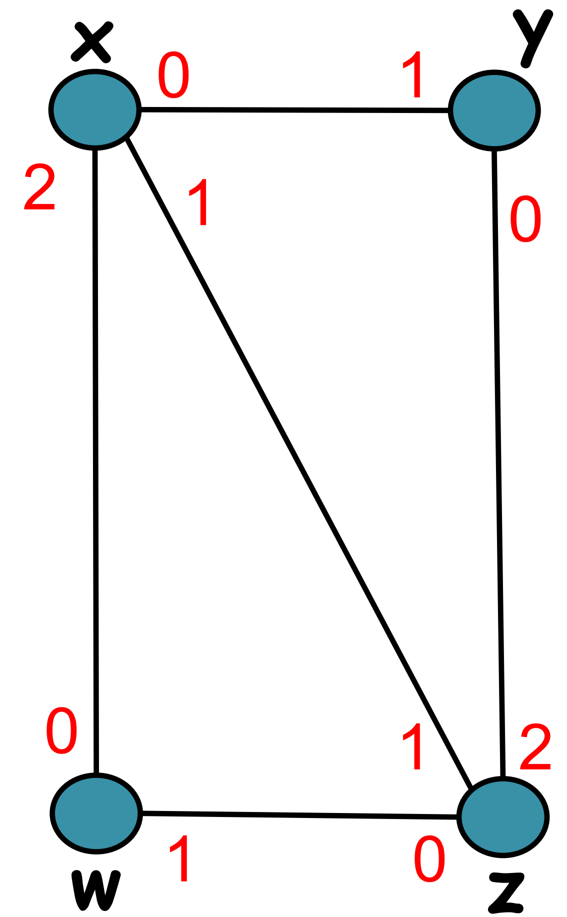
- An important special case of a predefined network representation is that of a spanning tree.
- Applications:
  - Broadcast
  - Convergecast
  - Graph exploration

# Background

- Desired property:
  compact storage
  (in terms of number of bits).
- Studied extensively for a variety of types of spanning trees.

# Port-Based Representations

- n-node graph G(V,E)
- Each node u has a pre-assigned distinct port number Port(u,v) (from the range {0, …,deg(u)-1}) for each edge (u,v)

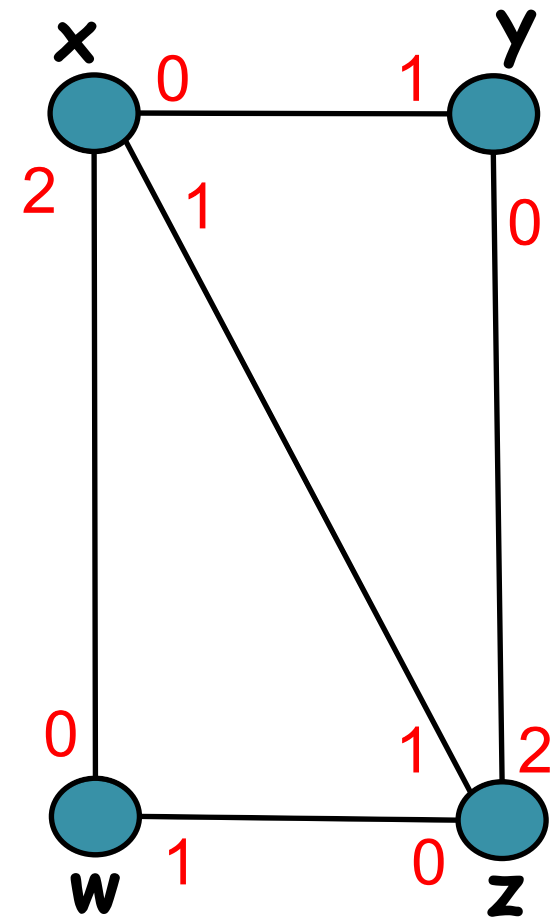


# Port-Based Representations

Note:

The port numbers are **not** necessarily **symmetric**;

it could be that $Port(u,v) \neq Port(v,u)$.



# Port-Based Representations

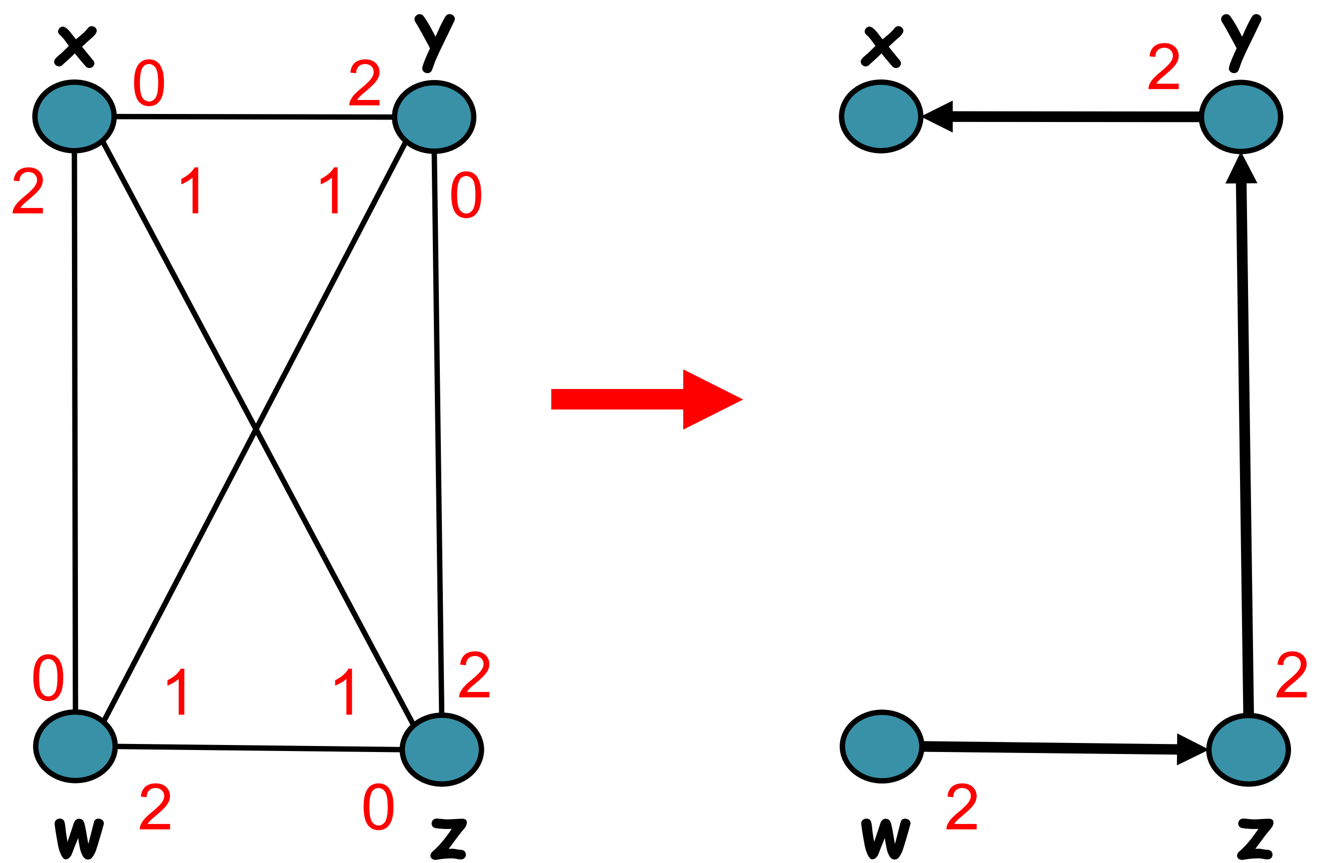
- The nodes are required to maintain a directed spanning tree T of G
- Each node is required to remember the port number leading to its parent in T.

# The problem

- The cost of a tree $T$ is the total number of bits stored by the nodes,

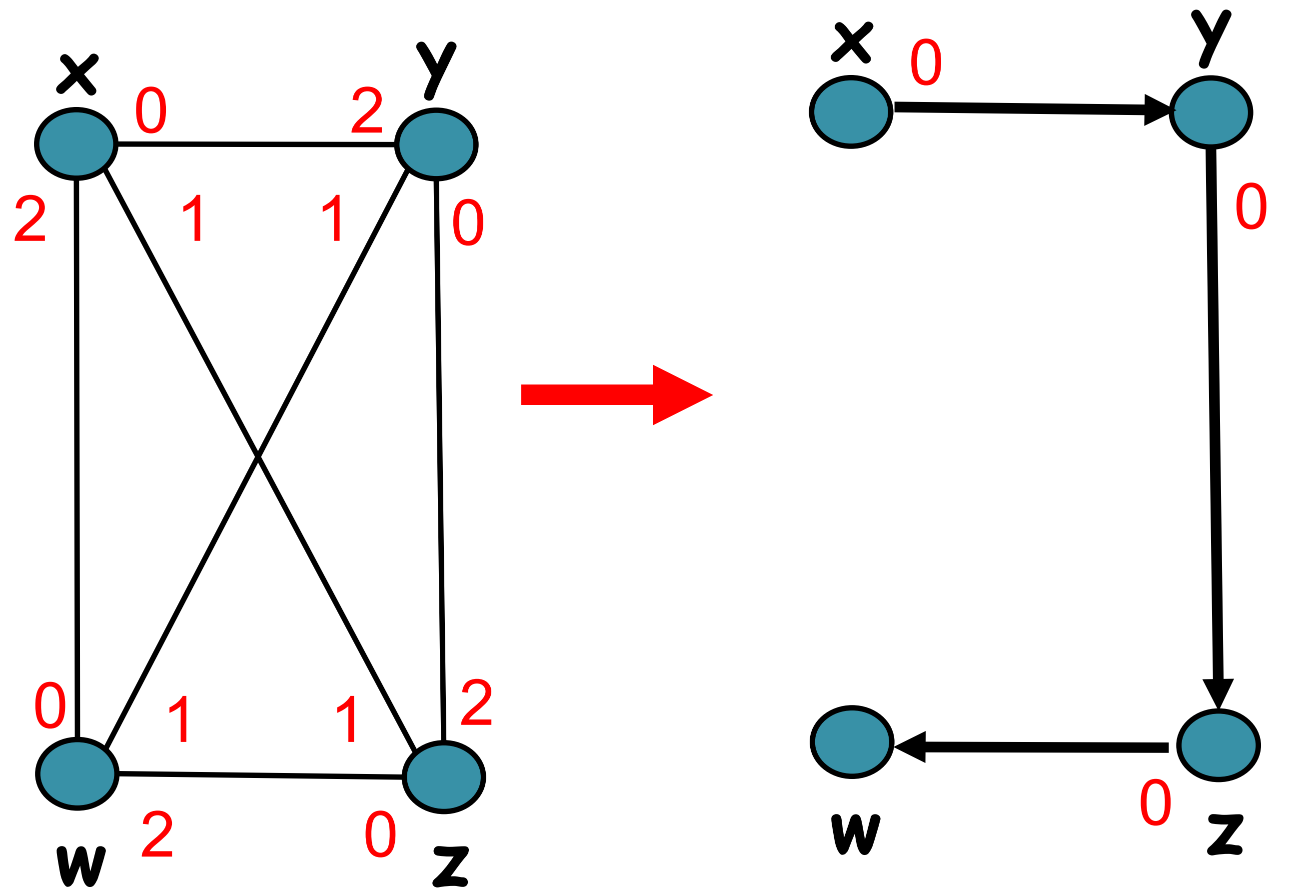
  $Cost(T,G) \approx \sum_v \log(Port(v,parent(v,T)))$
- $Cost(G) = \min_T \{Cost(T,G)\}$

# Illustration: Bad Tree




**Cost(T,G) = 6**

# Illustration: Good Tree




**Cost(T,G) = 3**

# Goals

- Algorithmic goal: given a graph $G(V,E)$ and port assignment, find a spanning tree $T$ of minimum cost.
- Combinatorial goal: establish tight bounds on the function $Cost(G)$

# Schemes for Port-Based Tree Representations

- The problem of compact port-based representations for spanning trees was introduced in [Cohen, Fraigniaud, Ilcinkas, Korman, Peleg, IWDC'05]
- Question raised: the existence of spanning trees with representations in which the average number of bits stored at each node is constant ($Cost(T,G) = O(n)$ )

# Known

Lemma [Cohen et al., IWDC'05]:

$\mathrm{Cost}(G) = O(n \log\log n)$ for every $n$-node graph $G$.

Special cases: $\mathrm{Cost}(G) = O(n)$

- for every complete $n$-node graph $G$
- for every $n$-node graph with a symmetric port labeling.

Conjecture [Cohen et al., IWDC'05]:

$\text{Cost}(G) = O(n)$ for every $n$-node $G$.

# New result

We confirm this conjecture, establishing a tight upper bound of $O(n)$ for an arbitrary n-node graph $G$ with an arbitrary port assignment.

# Trivial Upper Bound

- A trivial upper bound of $O(n \log n)$ can be derived by taking an arbitrary spanning tree of the graph $G$.

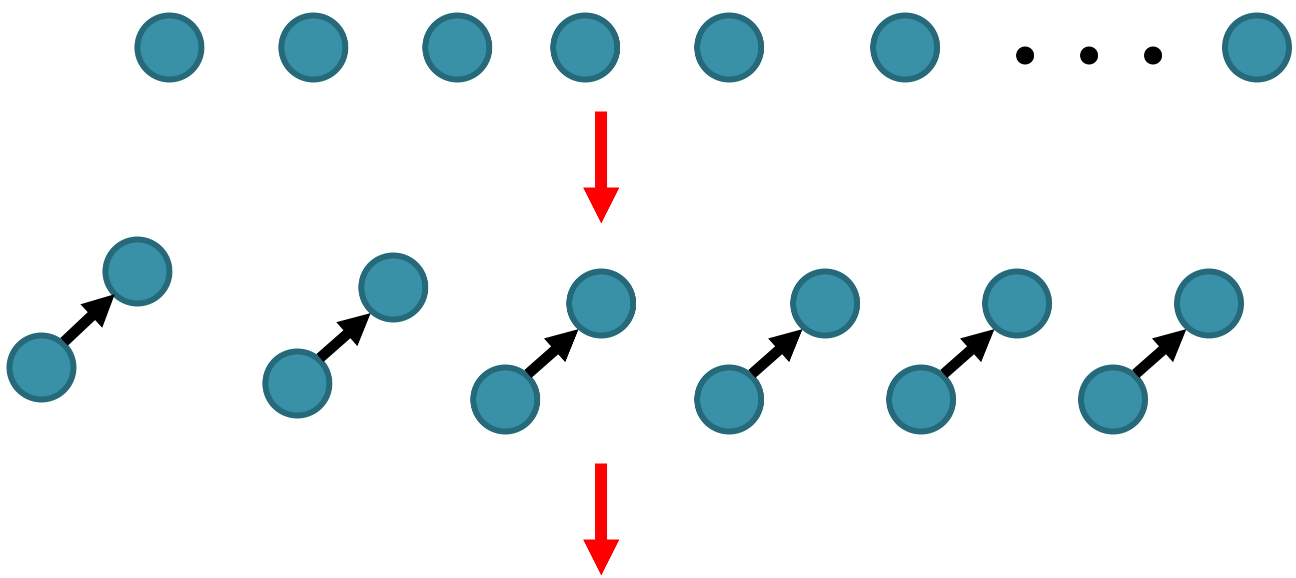
# Complete Graphs

## A tree construction algorithm

- The algorithm maintains a collection of rooted directed trees.
- Initially, each vertex forms a tree on its own.

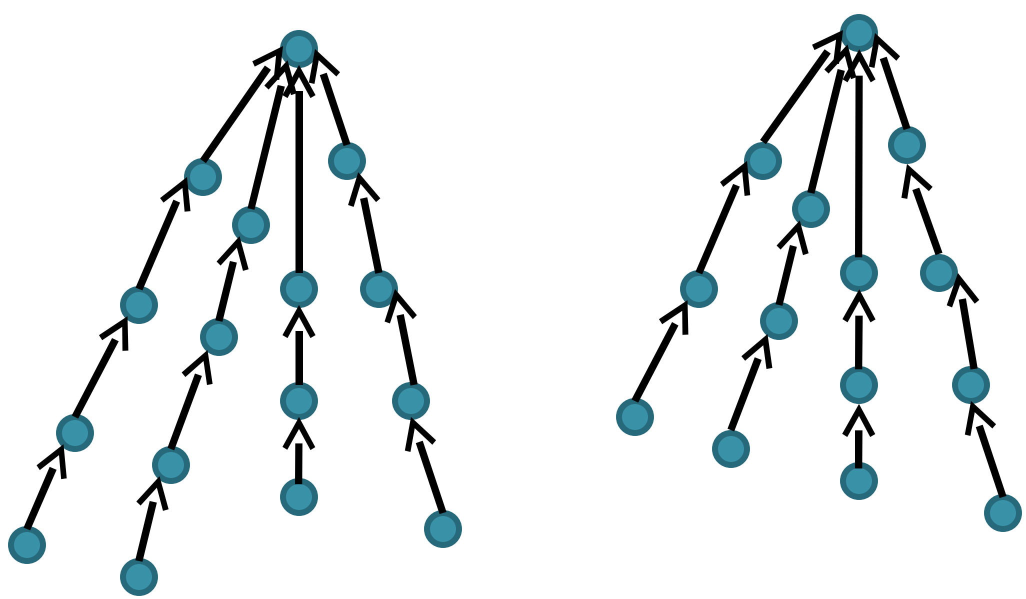
# Complete Graphs

The algorithm merges these trees into larger trees until it remains with a single tree spanning the entire graph.



# Complete Graphs

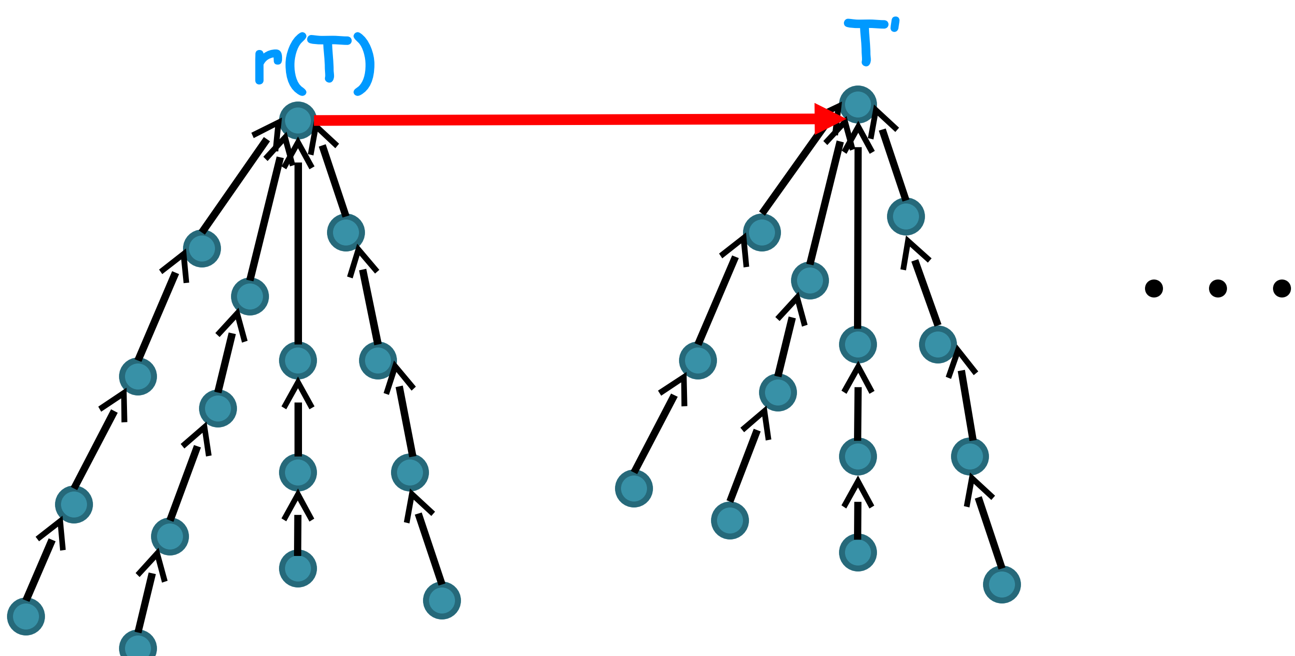
- The algorithm operates in **log n** phases
- Phase **k** handles all trees **T** such that **size(T)** < $2^k$



. . .

# Complete Graphs

- For each such tree T with root r(T), look at the set of outgoing edges that connect T to some other tree T' and select the edge e(T) of minimum cost.



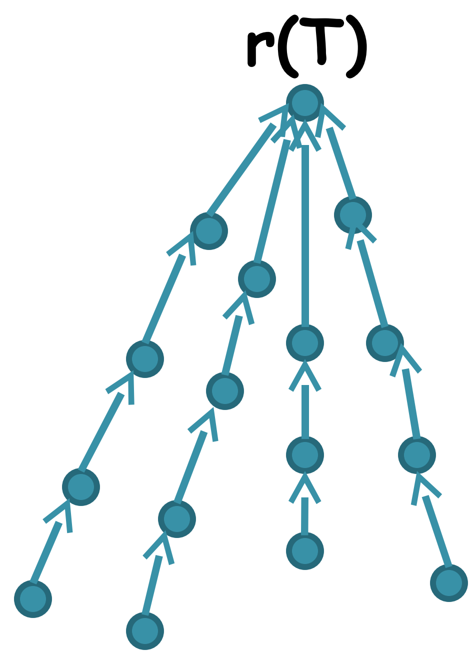
# Complete Graphs

**Lemma**

This process yields a tree of cost $O(n)$

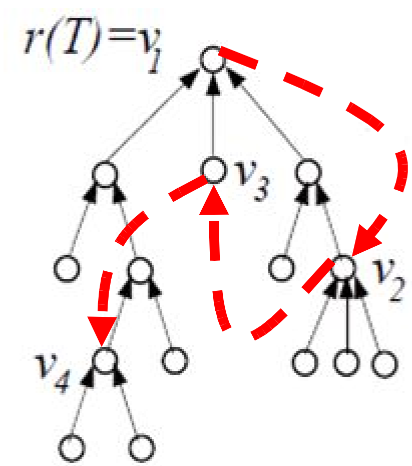
# Handling General Graphs

For an arbitrary graph, this process might fail once it encounters a tree T whose root has no outgoing edge to any node outside T



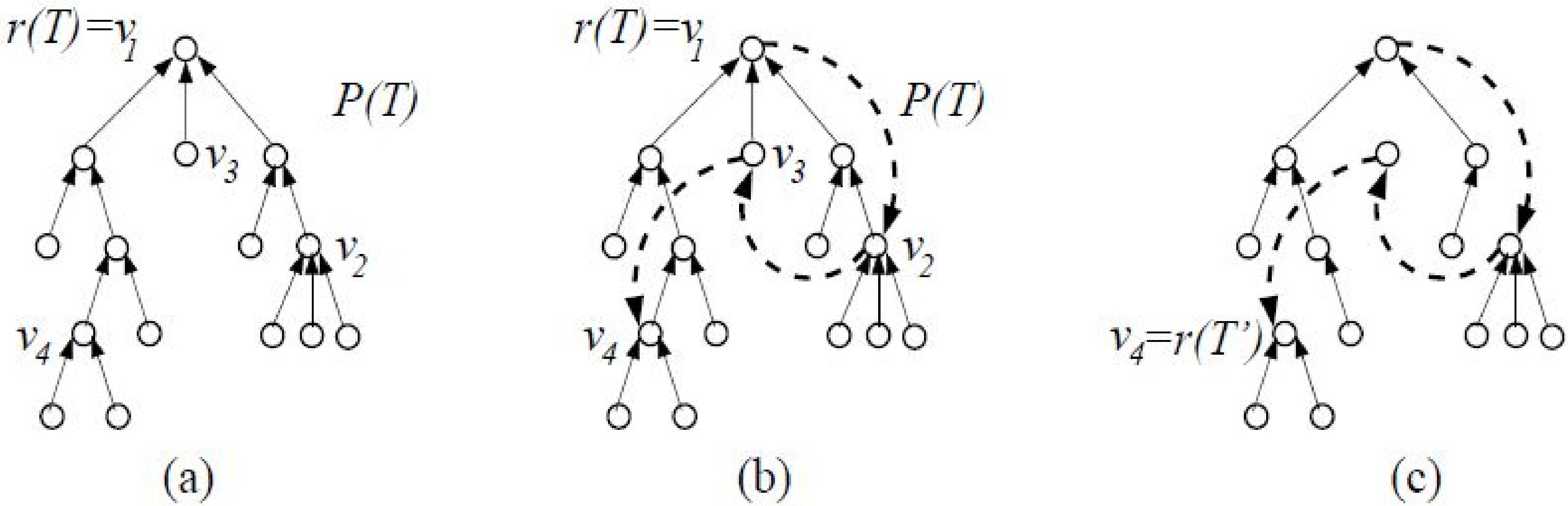
# Handling General Graphs

- For each directed sub-tree $T$, search for the shortest **reverse-path** $(v_1,\ldots,v_k)$ of $T$ (in hops):
  - $v_1 = r(T)$,
  - $v_1,\ldots,v_k \in V(T)$,
  - $(v_i, v_{i+1}) \in E$,
  - $v_k$ is an exit node, i.e., it has a neighbor outside $T$



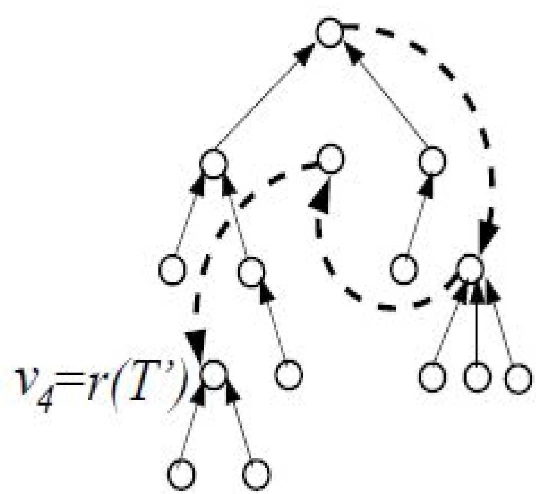
# Handling General Graphs

- Reverse the relevant edges on the path $(v_1, \ldots, v_k)$.



# Handling General Graphs

- Second, the algorithm looks at the set of exit edges of **r(T')**, and selects the exit edge **(r(T'), z)** of minimum **Port(r(T'), z)**.
- Merge **T'** with the tree containing **z** by adding the edge **(r(T'), z)**.

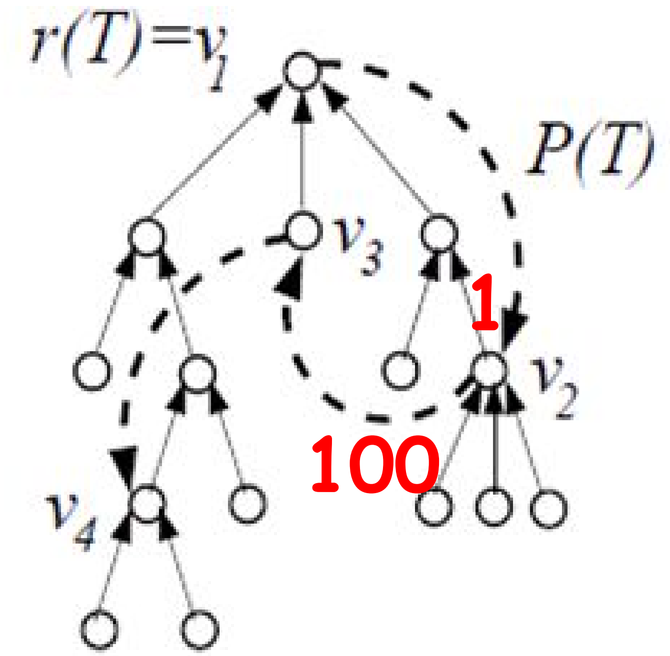


# Handling General Graphs

- By the analysis of Cohen et al., this algorithm yields a tree of cost **$O(n \log\log n)$**

# A tight Upper Bound – O(n)

- The reverse paths could be expensive.

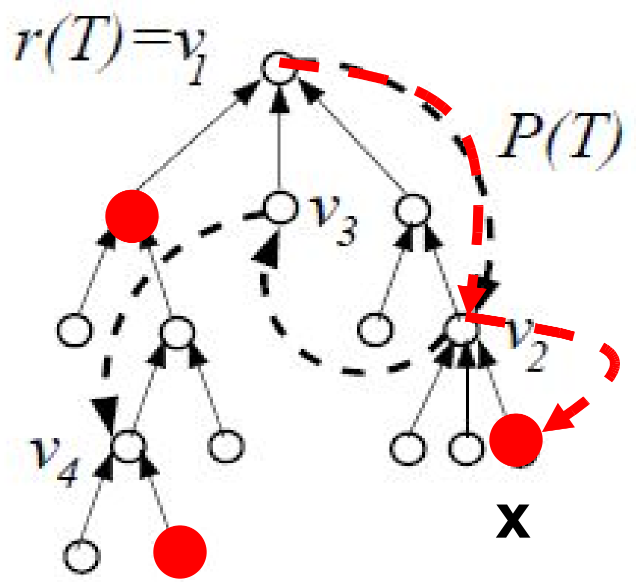


# A tight Upper Bound – O(n)

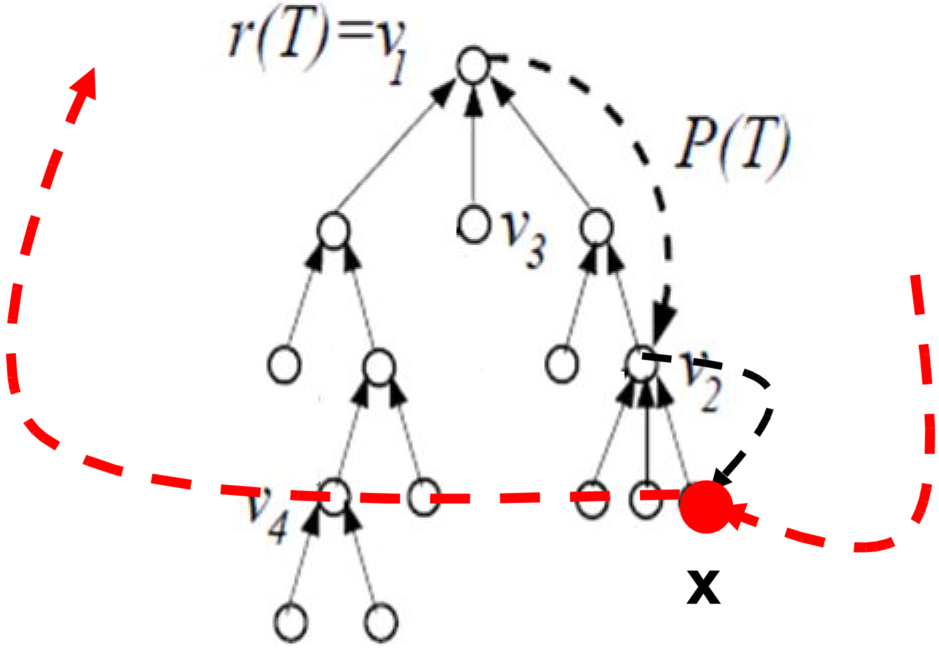
- The goal is to use reverse paths as "light" as possible.

# A tight Upper Bound – O(n)

- Consider all nodes that participate in some "lighter" reverse path of later iterations .
- Treat these nodes as nodes outside the tree T.



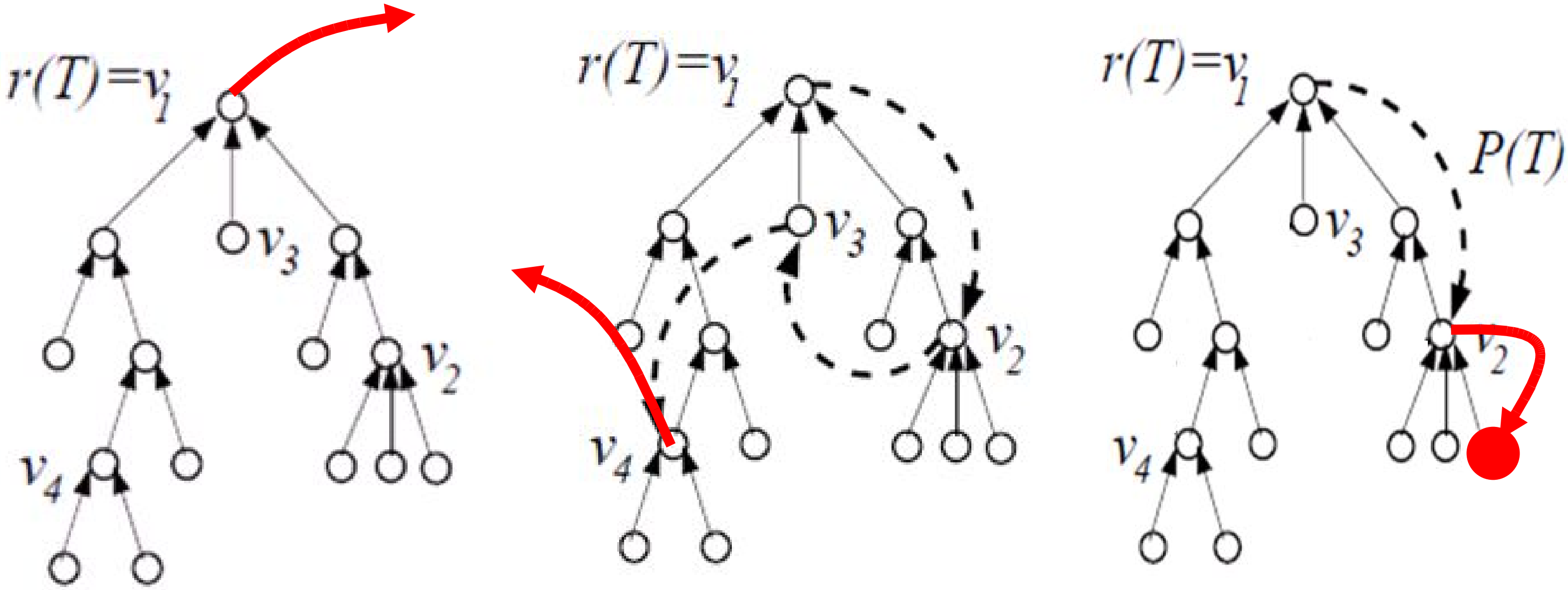
# A tight Upper Bound – O(n)



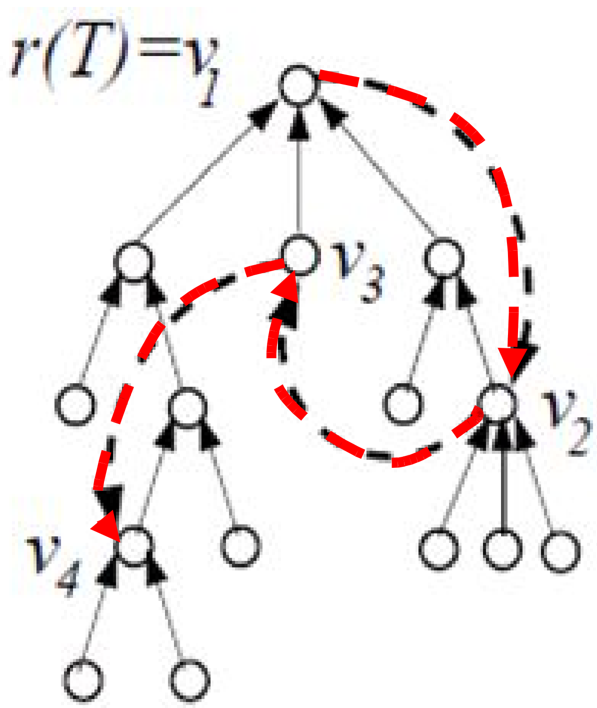
# A tight Upper Bound – O(n)

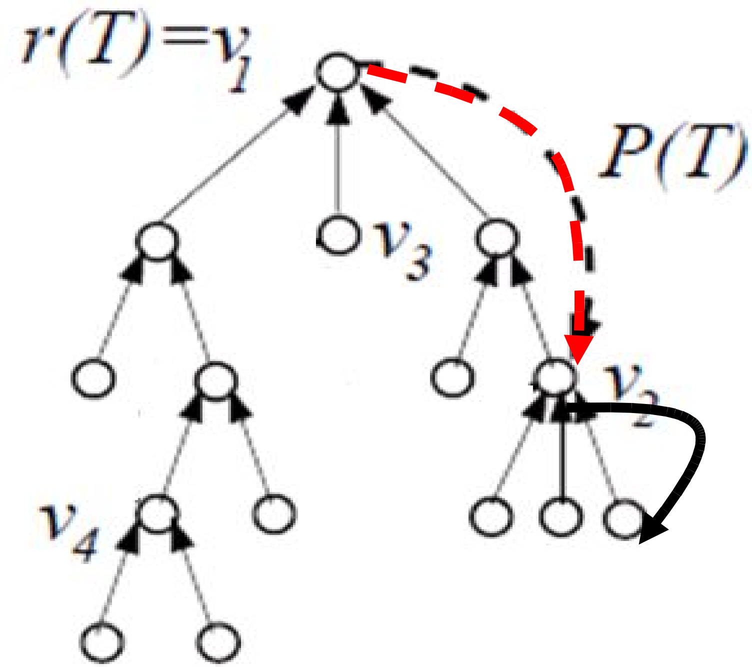
- Partition the edges of the final tree into two subsets, $E_{out}$ and $E_{rvrs}$.
- We bound separately the total cost of edges in each subset by $O(n)$.

# Out Edges



# Reverse Edges



# Reverse Paths- Cost

- Consider a reverse edge $(x,y)$. Let $c = Port(x,y)$.
- There are exactly $c$ neighbors $w$ of $x$ that are "cheaper" than $y$, i.e., such that $Port(x,w) < Port(x,y)$.
- **Charging rule:** If the algorithm selects the reverse edge $(x,y)$, then for each such cheaper node $w$, incur a charge of 1 for $w$.

# Reverse Paths- Cost

**Lemma 1**

- The lighter reverse paths are disjoint.

**Lemma 2**

- Every node **w** is being charged in only one lighter reverse path.

# Reverse Paths- Cost

**Lemma 3**

Consider a reverse path P where w is being charged.

Then there are at most three edges on P that charge w.

# A tight Upper Bound – O(n)

- A careful analysis of the construction with the lighter reverse paths shows that the cost of the tree is **O(n)**.

# Summary

- We show the construction of spanning trees in which the average number of bits stored at each node is constant.
- This is a tight upper bound.