A Parallel Algorithm for MISTs in Bubble-Sort Networks

Presented by:

M. Huzaifa (22i-1220)

M. Sarmad (22i-0941)

M. Umar Hassan (22i-0942)

Selected Paper:

Title: A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks

Drive Link:

https://drive.google.com/file/d/1leYyetxK3SqK8absssBw3QsNsD-T_Xwn_/view?usp=sharing

Problem Statement

Challenge

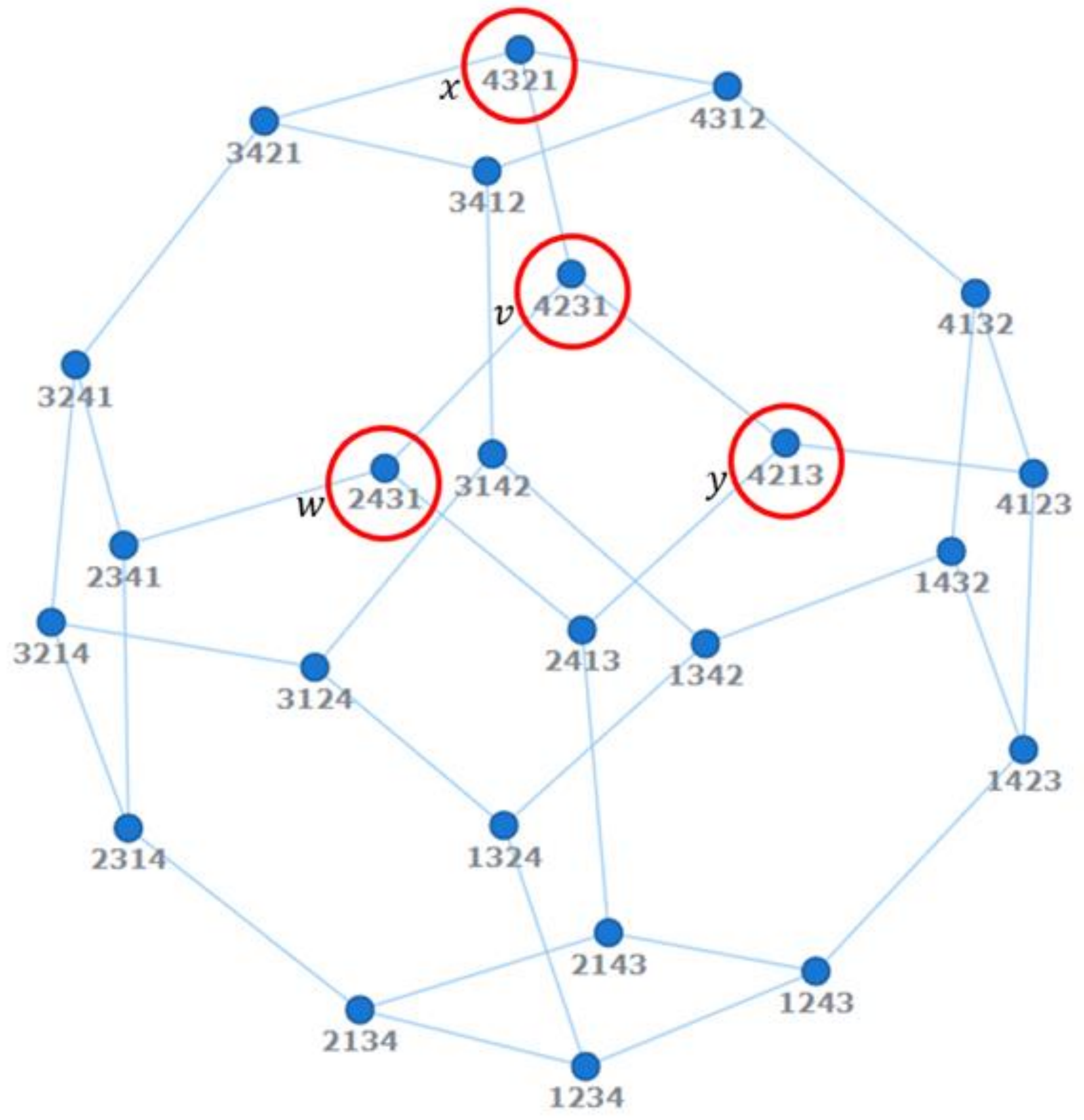
Design fault-tolerant, secure networks with disjoint paths.

Solution

Requires Multi-path routing and Multiple Independent Spanning Trees (MISTs).

Limitations

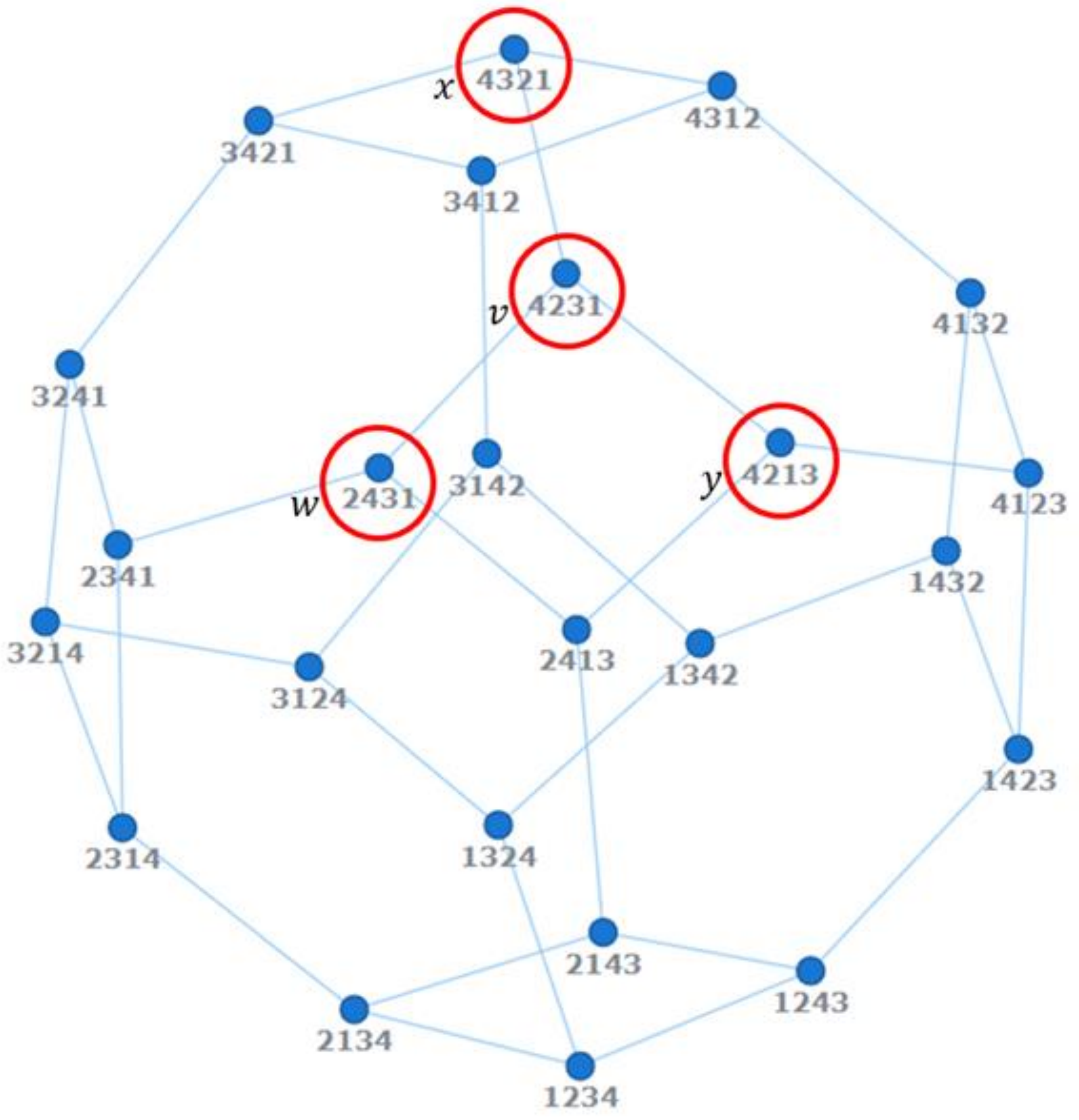
Existing recursive MIST algorithms in B_n lack parallelization.



Paper Goal

- **Develop a parallelizable algorithm for constructing Multiple Independent Spanning Trees (MISTs).**

- **Proposed solution specifically designed for Bubble Sort Networks.**



Terminology: Spanning Trees

Definition

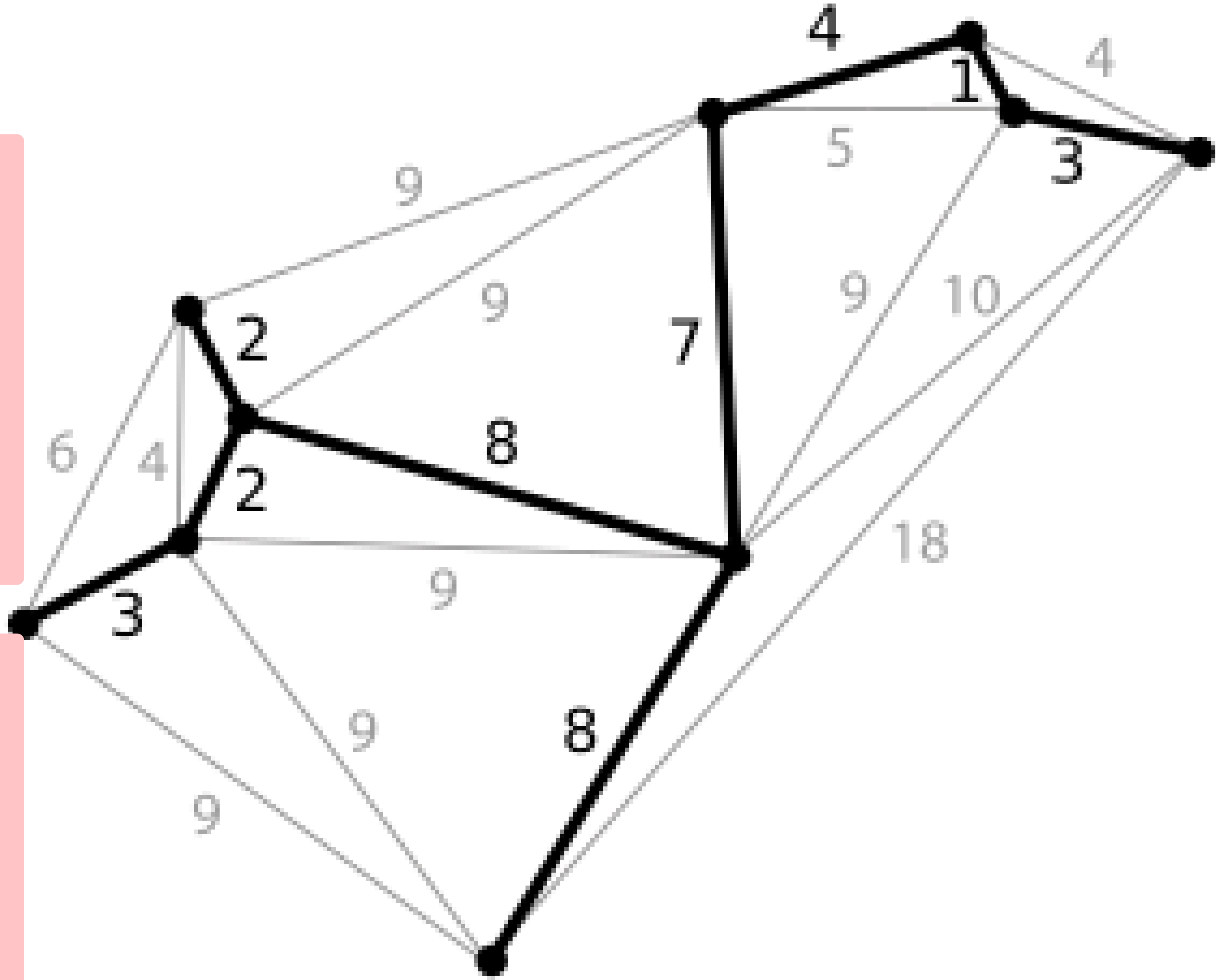
A spanning tree connects all nodes with no cycles.

Properties

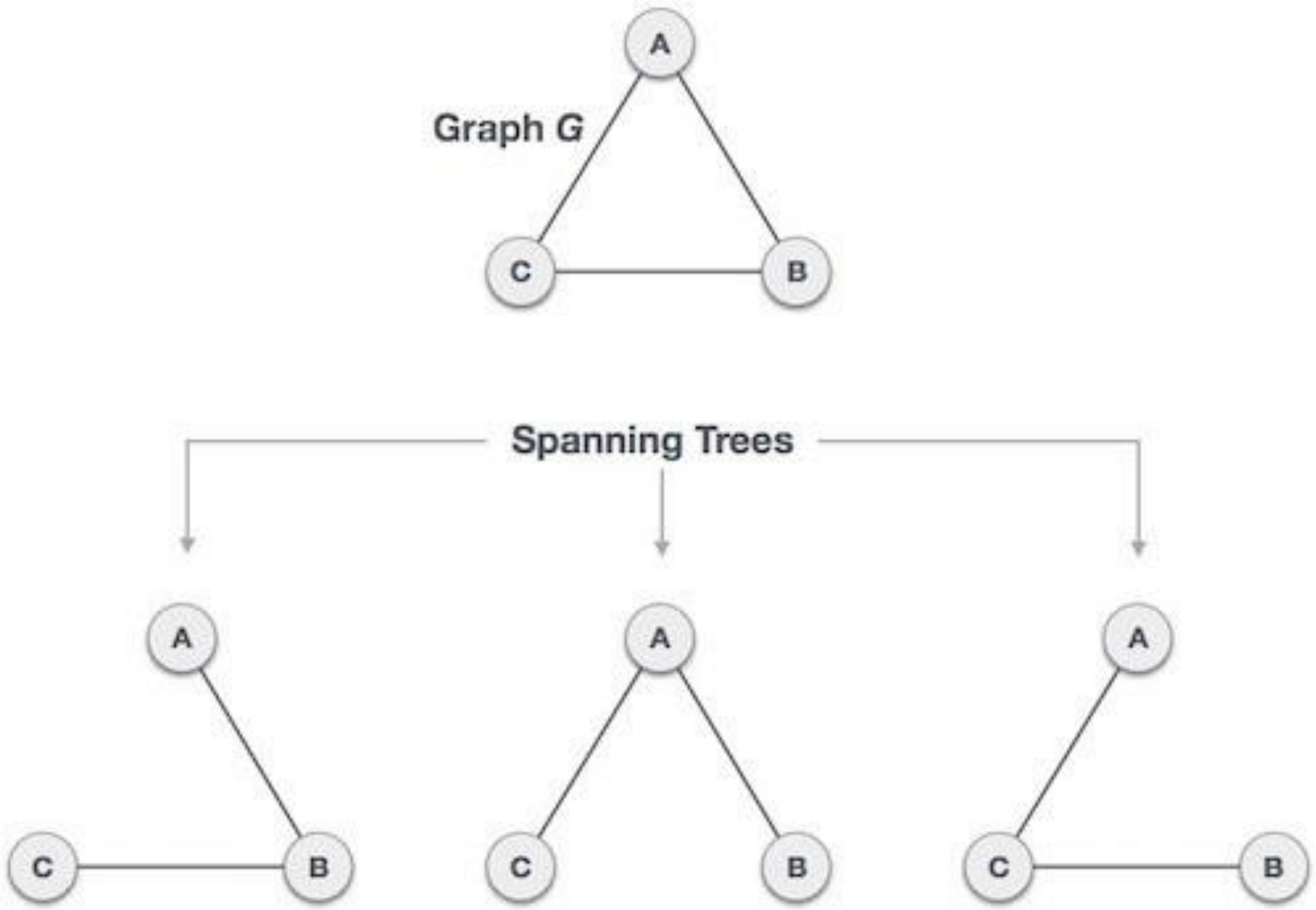
- Includes all vertices of graph
- No cycles, minimal edges

Uses

Routing backbones, broadcast trees, minimize communication cost.



Terminology: Spanning Trees



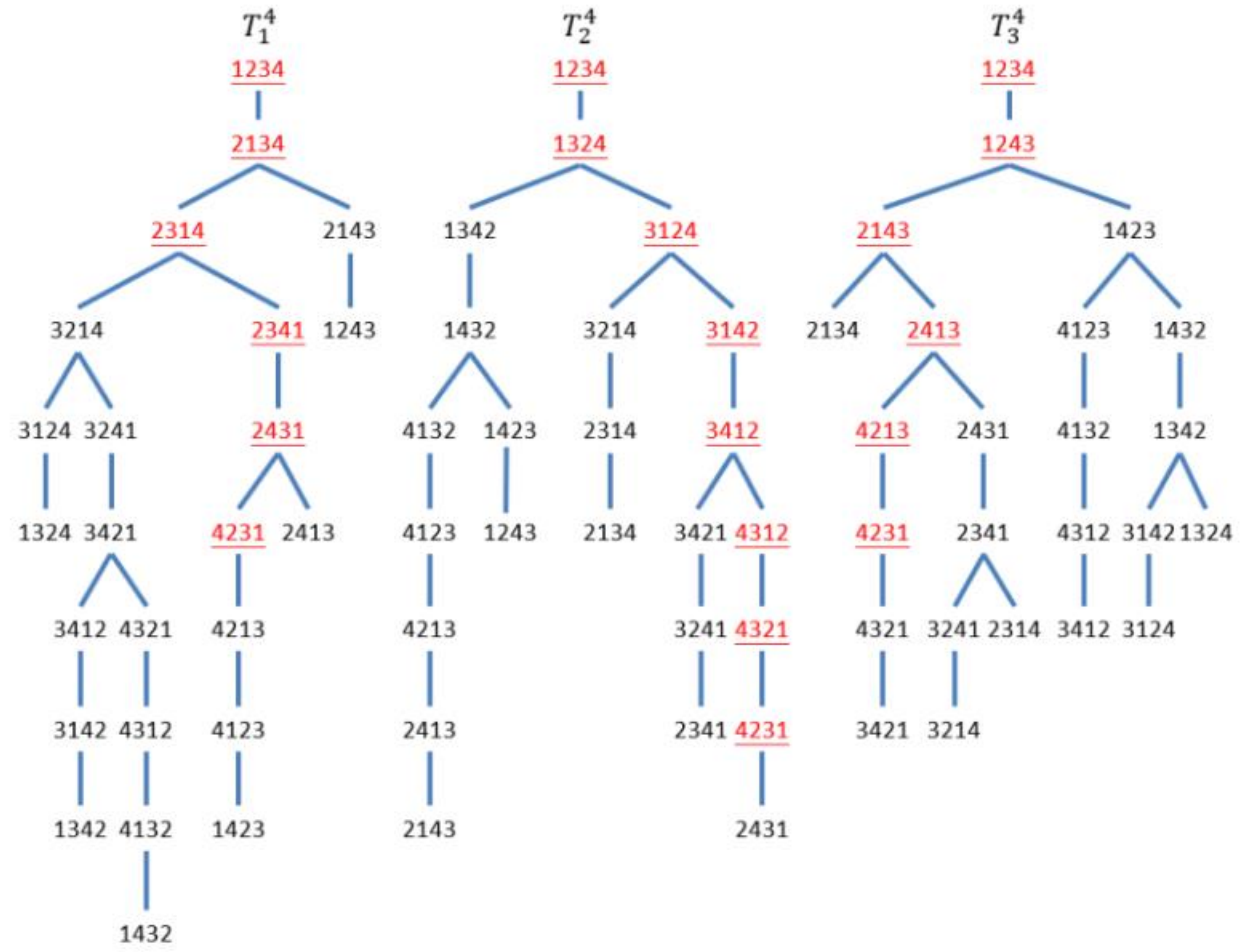
Terminology: Multiple Independent Spanning Trees (MISTs)

Definition

set of independent spanning trees constructed from the same connected graph G.

Properties

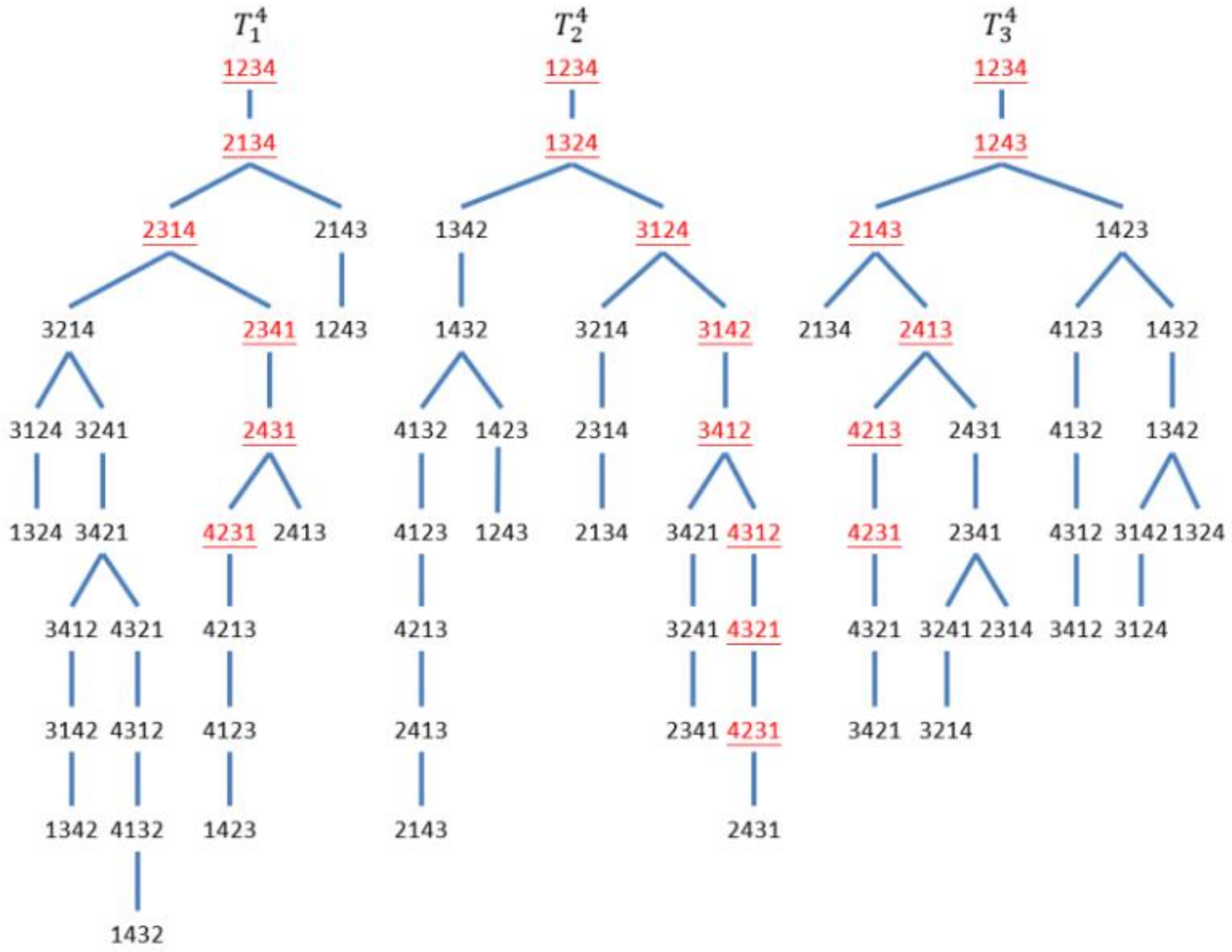
- Valid spanning trees of graph
- Edge-disjoint or vertex-disjoint paths
- Up to k trees in a k-connected graph



Terminology: Multiple Independent Spanning Trees (MISTs)

Uses

- **Fault Tolerance:** If a node or edge fails in one tree, others can still function — crucial in network reliability.
- **Parallel Routing:** Allows load balancing and reduced congestion by routing data over multiple trees simultaneously.
- **Security & Resilience:** Splitting messages across trees improves confidentiality and resistance to data loss.



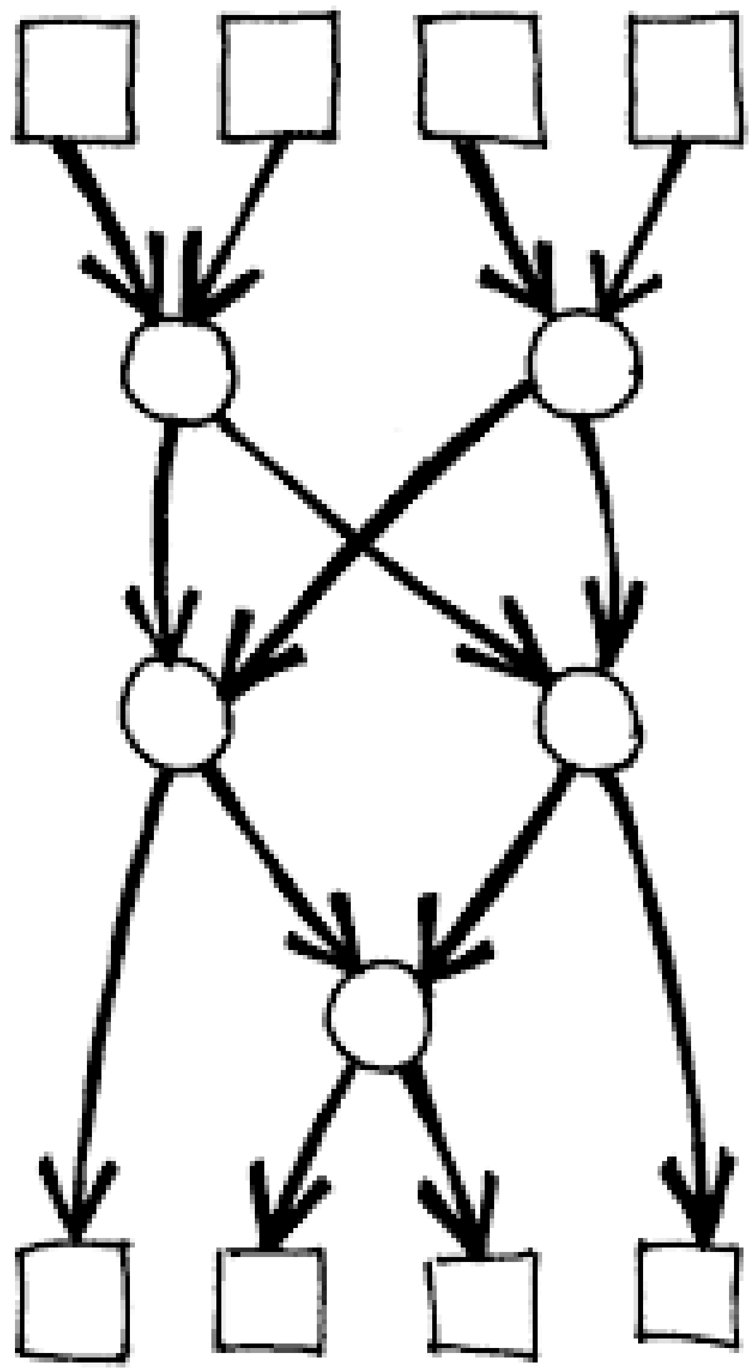
Terminology: Bubble-Sort Network (B_n)

Definition

A graph where nodes are all permutations of n elements, and edges represent adjacent swaps (like bubble sort).

Properties

- Vertices: All n! permutations
- Edges: Adjacent transpositions
- Structure: Cayley graph
- Connectivity: $n-1$
- Diameter: $n(n-1)/2$



Uses

- Test routing algorithms
- Simulate broadcasting
- Study network resilience

Terminology: Bubble-Sort Network (B_n)

Example: Bubble sort graph of dimension 3

- Permutations:

  {123,132,213,231,312,321}
- Edges:

  Each node is connected to others by swapping adjacent elements
- Each edge is an adjacent transposition (like how bubble sort makes swaps).

```
123 — 132
 |       |
213 — 231
 |       |
312 — 321
```

Algorithm Overview

1

Core Function

Parent(v, t, n) computes parent of vertex v in tree T_n^t.

2

Preprocessing

- Compute v^{-1} (inverse permutation)
- Find r(v): first misplaced symbol position

3

Decision Rules

Swap based on conditions of t, rightmost incorrect symbol, or default position v t.

4

Parallelism

Each vertex's parent computed independently, suitable for parallelization.

Algorithm 1: Parent1(v, t, n)

Input : v: the vertex $v = v_1 \dots v_n \in V(B_n) \setminus \{\mathbf{1}_n\}$
t: the t-th tree T_t^n in IST
n: the dimension of B_n
Output: p: $p =$ Parent1(v, t, n) the parent of v in T_t^n
if $v_n = n$ **then**
(1) **if** $t \neq n-1$ **then** $p =$ FindPosition(v)
(2) **else** $p =$ Swap(v, v_{n-1})
end
else
if $v_n = n-1$ and $v_{n-1} = n$ and Swap$(v, n) \neq \mathbf{1}_n$ **then**
(3) **if** $t = 1$ **then** $p =$ Swap(v, n)
(4) **else** $p =$ Swap$(v, t-1)$
end
else
(5) **if** $v_n = t$ **then** $p =$ Swap(v, n)
(6) **else** $p =$ Swap(v, t)
end
end
return p

Function FindPosition(v)

Input : v: the vertex $v = v_1 \cdots v_n$ in B_n
Output: p: the vertex adjacent to v in B_n
(1.1) **if** $t = 2$ and Swap$(v, t) = \mathbf{1}_n$ **then** $p =$ Swap$(v, t-1)$
(1.2) **else if** $v_{n-1} \in \{t, n-1\}$ **then** $j = r(v), p =$ Swap(v, j)
(1.3) **else** $p =$ Swap(v, t)
return p

Function Swap(v, x)

Input : v: the vertex $v = v_1 \cdots v_n$ in B_n
x: the symbol in the vertex $v_1 \cdots v_n$
Output: p: the vertex adjacent to v in B_n
$i = v^{-1}(x), p = v\langle i \rangle$
return p

Results and Complexity

Time Complexity Analysis:

- Per Vertex: O(n) time to compute parent
- Total: O(n × n!) for all vertices in B_n

Parallelism:

Fully parallelizable — each vertex's parent computed independently

Upper Bound:

Asymptotically optimal for Bubble Sort Network of size n!

Table 1 The parent of every vertex $v \in V(B_4) \setminus \{\mathbf{1}_4\}$ in T_t^4 for $t \in \{1, 2, 3\}$ calculated b
Algorithm 1

v	t	v_4	Rule	p	v	t	v_4	Rule	p
1234	-	-	-	-	3124	1	4	(1.3)	3214
						2		(1.2)	1324
						3		(2)	3142
1243	1	3	(6)	2143	3142	1	2	(6)	3412
	2		(6)	1423		2		(5)	3124
	3		(5)	1234		3		(6)	1342
1324	1	4	(1.3)	3124	3214	1	4	(1.2)	2314
	2		(1.2)	1234		2		(1.3)	3124
	3		(2)	1342		3		(2)	3241
1342	1	2	(6)	3142	3241	1	1	(5)	3214
	2		(5)	1324		2		(6)	3421
	3		(6)	1432		3		(6)	2341
1423	1	3	(6)	4123	3412	1	2	(6)	3421
	2		(6)	1432		2		(5)	3142
	3		(5)	1243		3		(6)	4312
1432	1	2	(6)	4132	3421	1	1	(5)	3241
	2		(5)	1342		2		(6)	3412
	3		(6)	1423		3		(6)	4321
2134	1	4	(1.2)	1234	4123	1	3	(6)	4213
	2		(1.1)	2314		2		(6)	4132
	3		(2)	2143		3		(5)	1423
2143	1	3	(3)	2134	4132	1	2	(6)	4312
	2		(4)	2413		2		(5)	1432
	3		(4)	1243		3		(6)	4123
2314	1	4	(1.2)	2134	4213	1	3	(6)	4231
	2		(1.3)	3214		2		(6)	4123
	3		(2)	2341		3		(5)	2413
2341	1	1	(5)	2314	4231	1	1	(5)	2431
	2		(6)	3241		2		(6)	4321
	3		(6)	2431		3		(6)	4213
2413	1	3	(6)	2431	4312	1	2	(6)	4321
	2		(6)	4213		2		(5)	3412
	3		(5)	2143		3		(6)	4132
2431	1	1	(5)	2341	4321	1	1	(5)	3421
	2		(6)	4231		2		(6)	4312
	3		(6)	2413		3		(6)	4231

Proposed Solution

No Need for METIS

- Parent computation is fully local, so graph partitioning is unnecessary.
- Embarrassingly Parallel Problem (No communication needed between machines)

MPICH – Inter-node Parallelism

- Distributes Spanning Tree Construction across machines
- Master assigns tree ranges to each machine
- Workers return constructed subtrees to master

Proposed Solution

OpenMP – Intra-node Parallelism

- Used within each machine for vertex-level parallelism
- Parent computation parallelized using OpenMP threads
- No synchronization needed due to independence of iterations

Thank You