**Notes:**

**Problem :**

**Fault-tolerant transmission** and **secure message distribution** are the challenges in designing interconnected networks in a reliable communication network. Providing disjoint paths between each pair of vertices is the practical way to satisfy the above requirements through multi-path routing (See Figure 3). Thereby raising a need for multiple independent spanning trees. A recursive solution to this approach was proposed but it couldn’t be parallelized.

**Goal of this paper:**

To develop a parallelizable algorithm for MISTs(Multiple Independent Spanning Trees) construction on Bubble Sort Networks.

**Preliminaries:**

1. **What is a Spanning Tree:**A spanning tree is a subgraph of a graph that:
2. Includes all the vertices of the original graph
3. Is a tree (i.e., it has no cycles and is connected)

**Why are spanning trees useful?**

* Minimizes communication cost (e.g. in networks)
* Backbones for broadcasting or routing

1. **What are Multiple Independent Spanning Trees:**

Given a connected graph G, MISTs are a set of spanning trees such that:

* Each Tree​ is a valid spanning tree of G
* The trees are independent in structure, usually:
  + **Edge-disjoint**: No two trees share the same edge
  + **Vertex-disjoint paths**: From a specific root node to every other node, the paths taken in each tree are all disjoint except at the endpoints
* If a graph is **k-connected** (K vertices need to be removed to make the graph disconnect), then we can build K ISTs.

**Why is this useful?**

**1. Fault tolerance in networks**

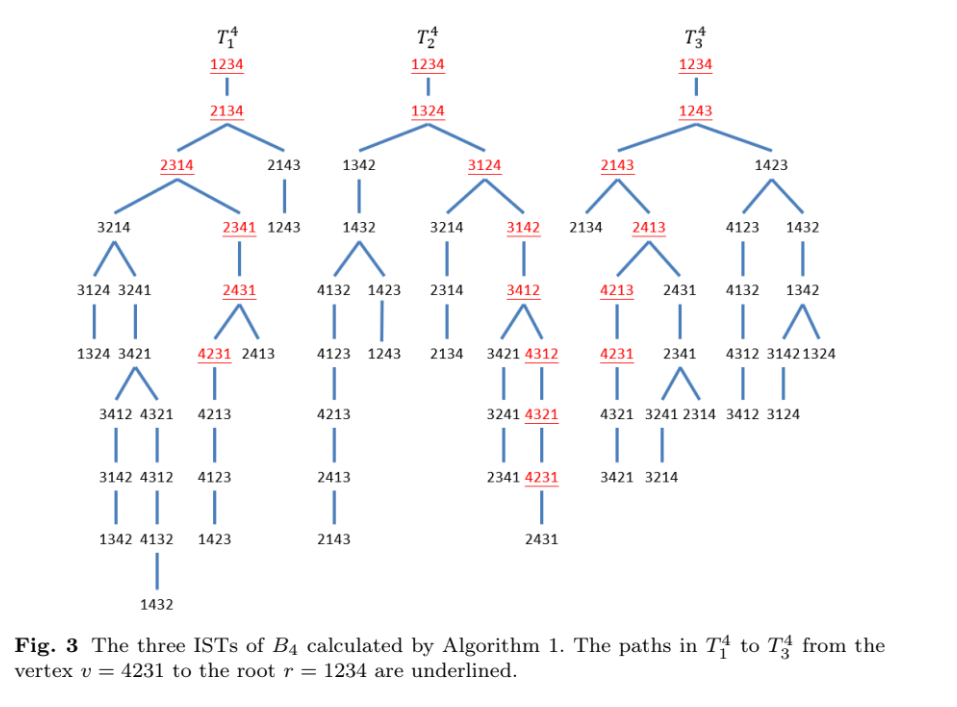
If one tree fails (e.g., due to an edge or node going down), other trees can still carry information without interference.

**2. Parallel routing**

You can send different messages through different trees to avoid congestion or ensure delivery.

**3. Security & resilience**

Messages can be split across trees and recombined, which helps prevent eavesdropping or data loss.



(Multiple Independent Spanning Trees)

1. **What are Bubble Sort Networks:**Consider E = {1,2,…,n}. A bubble sort network of **dimension n** has:
   1. **Vertices**: all permutations of n elements → total of n! nodes
   2. **Edges**: each edge connects two permutations that differ by a **single adjacent swap** (like bubble sort would do)

So the structure is:

* **Vertices = permutations**
* **Edges = adjacent swaps**

This forms a graph known as a **Cayley graph**

**Connectivity:** n-1 and its **Diameter:** n(n-1)/2

**🔁 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**

123 — 132

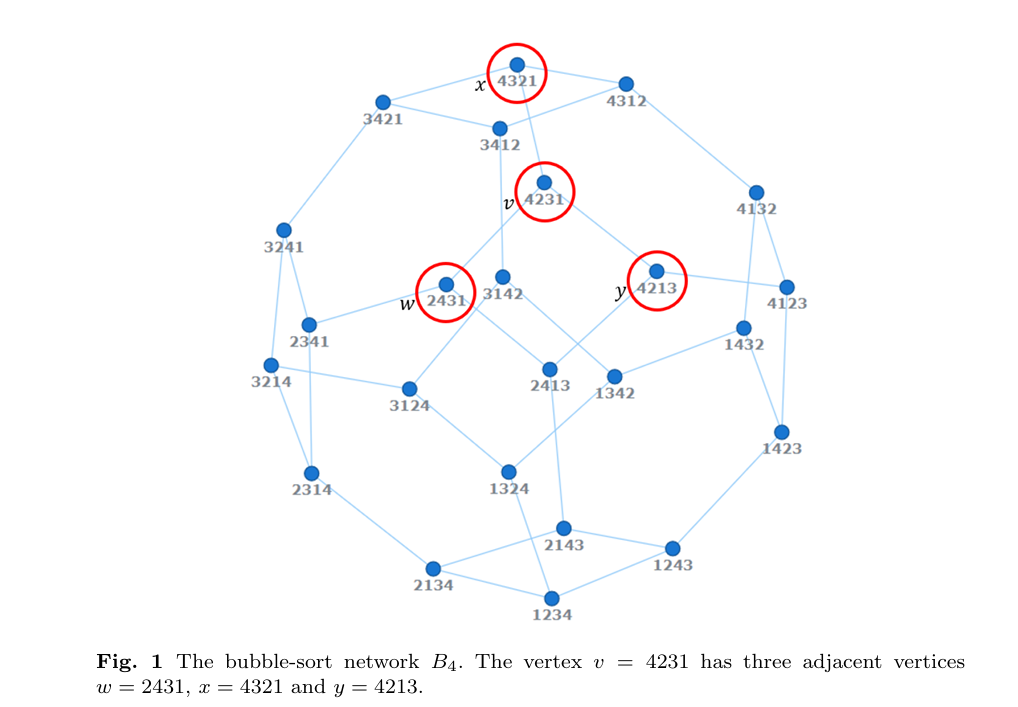
| |

213 — 231

| |

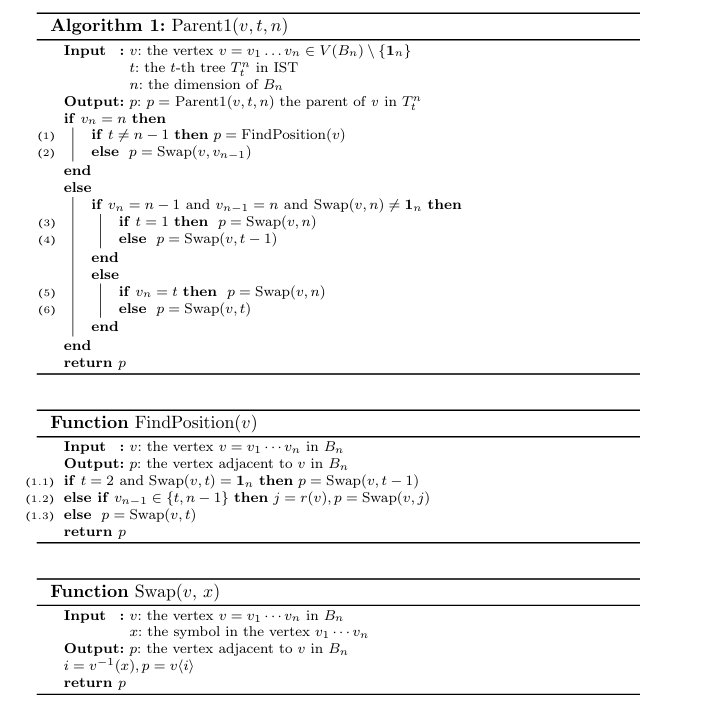
312 — 321

Each edge is an adjacent transposition (like how bubble sort makes swaps).



(Bubble Sort Network of dimension 4)

**Why Used:**Because the topology is regular and known it is used to test routing algorithms, broadcasting, and network resilience.

**Algorithm Proposed in the research paper:  
**

Where Parent(v,t,n) returns the parent of the vertex **v** in the **t-th** tree of the bubble sort network **B** of dimension **n.** We can iterate this for every vertex to construct the ISTs.

**Output of Algorithm for n=4:**A table of mathematical equations

AI-generated content may be incorrect.

**Results:  
Time complexity:** O(n · n!) of our algorithm is asymptotically optimal, where **n is the dimension of the Bubble sort Network** and **n! is the number of vertices of the network**.  
Every vertex’s parent calculation can be parallelized. So the algorithm is fully parallelizable.

**Parallelization Strategy:**

1. **Using METIS tool (No Need):**

**<There is no need for using METIS as every vertex’s parent calculation is entirely a local process (Embarrassingly Parallelizable Problem) so no communication occurs between machines which we could have minimized >**

**Don’t include the lower part in slides. Just read it:**

METIS is a tool that takes a large graph and partitions it into smaller subgraphs (or "parts") while minimizing the number of edges between them.

In parallel graph algorithms, **performance suffers when too many edges cross between different MPI processes**. These cross-process edges require:

* Message passing between processes
* Synchronization delays
* Complex data management

So, to make your parallel algorithm efficient:

* You want each MPI process to get a “chunk” of the graph that’s **dense inside** but has **few connections outside**
* This minimizes inter-process communication (a major bottleneck in MPI)

METIS helps by:

* **Partitioning the graph intelligently** so as to minimize inter-process communication and synchronization

1. **Using MPICH (inter-node parallelism):**

MPICH will be used to distribute the Spanning Tree Construction among multiple machines. Master process will share the range of trees that each process(machine) has to construct and will later receive the constructed trees.

1. **Using OpenMP (intra-node parallelism):**

Each machine will use OpenMP for parallelizing the parent computation for each vertex in the assigned portion of tree using the algorithm provided in the research paper. Since the problem is embarrassingly parallel no such communication or synchronization will be needed. Only the loop iterations will be distributed among multiple threads.