Distributed Computing
Project Report

Yashi Rastogi (CS23RESCH11007)
Souvik Sarkar (CS23MTECH02001)

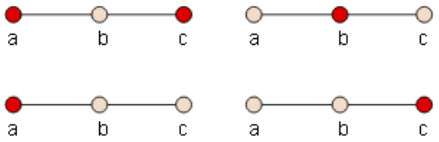
---

1. Introduction:

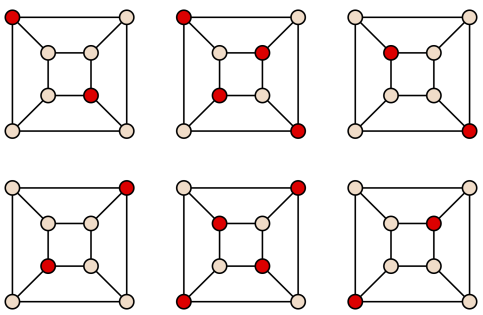
a. Problem Description:

In graph theory, a maximal independent set (MIS) or maximal stable set is an independent set that is not a subset of any other independent set. In other words, there is no vertex outside the independent set that may join it because it is maximal with respect to the independent set property.

For example, in the graph P3, a path with three vertices a, b, and c, and two edges ab and bc, the sets {b} and {a, c} are both maximally independent. The set {a} is independent, but is not maximal independent, because it is a subset of the larger independent set {a, c}. In this same graph, the maximal cliques are the sets {a, b} and {b, c}.



A graph may have many MISs of widely varying sizes; the largest, or possibly several equally large, MISs of a graph is called a maximum independent set. The graphs in which all maximal independent sets have the same size are called well-covered graphs.



b. Counting Maximal Independent Set:

The counting problem associated to maximal independent sets has been investigated in computational complexity theory. The problem asks, given an undirected graph, how many maximal independent sets it contains. This problem is NP-hard already when the input is restricted to be a bipartite graph.

c. BaseLine:

The "Vertex Coloring Problem" is a classic problem in graph theory where the goal is to assign colors to the vertices of an undirected graph in such a way that no two adjacent vertices share the same color, using the minimum number of colors possible. In this case, the constraint is that the total number of colors should not exceed d+1, where d is the maximum degree of the graph (the highest number of edges connected to any vertex).

An MPI (Message Passing Interface) program is a parallel computing model used for distributed memory systems, commonly employed in high-performance computing. MPI allows multiple processes running on different processors or nodes to communicate and synchronize with each other to solve a problem collectively.

An MPI program to find the optimal coloring of a given graph G would involve multiple processes working together to explore different possible colorings of the graph and communicate asynchronously to exchange information about their progress and any potential solutions found.

How an MPI program work:

- **Dividing the Graph**: The graph G would be divided among the MPI processes. Each process would be responsible for coloring a subset of the vertices.Coloring Process: Each process would independently attempt to color its assigned vertices according to the constraints (no adjacent vertices with the same color). Initially, each vertex might be assigned a unique color.
- **Communication:** Asynchronously, processes would communicate with each other to exchange information about their current coloring progress. They would share information about which vertices they have colored and what colors they have used.
- **Conflict Resolution:** When conflicts arise (i.e., adjacent vertices with the same color), processes might need to adjust their coloring strategy. This could involve reassigning colors to some vertices or requesting neighboring processes to change their coloring.
- **Optimization:** Processes would continuously work to minimize the total number of colors used while ensuring that the coloring remains valid.

- **Termination:** The processes would continue iterating and communicating until an optimal coloring is found or until a predefined termination condition is met (such as a maximum number of iterations).

i. Sequential Algorithm:

Given a Graph G(V,E), it is easy to find a single MIS using the following algorithm:

Step 1. Initialize I to an empty set.
Step 2. While V is not empty:
Step 3. Choose a node $v \in V$;
Step 4. Add v to the set I;
Step 5. Remove from V the node v and all its neighbours.
Step 6. Return I.

ii. Lubys's Algorithm:

Step 1. Initialize I to an empty set.
Step 2. While V is not empty:
Step 2a. Choose a random set of vertices $S \subseteq V$, by selecting each vertex v independently with probability 1/(2d(v)), where d is the degree of v (the number of neighbours of v).
Step 2b. For every edge in E, if both its endpoints are in the random set S, then remove from S the endpoint whose degree is lower (i.e. has fewer neighbours). Break ties arbitrarily, e.g. using a lexicographic order on the vertex names.
Step 2c. Add the set S to I.
Step 2d. Remove from V the set S and all the neighbours of nodes in S.
Step 3. Return I.

The following algorithm finds a MIS in time **O(log n)**.

iii. Ghaffari Algorithm:

Ghaffari algorithm guarantees for each node v that after $O(\log \Delta + \log 1/\varepsilon)$ rounds, with probability at least $1 - \varepsilon$, node v has terminated and it knows whether it is in the (computed) MIS or it has a neighbor in the (computed) MIS.
This algorithm guarantees that each node in a graph will determine,

The algorithm considers two scenarios for a node v: either it has a good chance of joining the MIS because it doesn't have many competing neighbors, or many of its neighbors are trying to join the MIS and each of them doesn't have too much competition, increasing the likelihood of one of them joining the MIS and consequently removing v.

To achieve this, the algorithm creates a deterministic dynamic that emphasizes these two scenarios as stable points and ensures that each node spends a significant amount of time in them, unless it's already been removed.

The MIS Algorithm of [13]: In each round, each node v gets *marked* with probability $p_t(v)$. If v is marked and none of its neighbors is marked, then v joins the MIS and gets removed along with its neighbors. The marking probabilities are set as follows: Initially $p_0(v) = 1/2$, and

$$p_{t+1}(v) = \begin{cases} p_t(v)/2, & \text{if } d_t(v) = \sum_{u \in N(v)} p_t(u) \geq 2 \\ \min\{2p_t(v), 1/2\}, & \text{if } d_t(v) = \sum_{u \in N(v)} p_t(u) < 2. \end{cases}$$

iv. Beeping Algorithm:

By using a fundamental observation, the Beeping MIS Algorithm streamlines communication. Rather than sending exact values between nodes to determine a node's degree, it suffices to determine whether a marked node exists among its neighbors. A marked neighbor is more likely to exist if the node has a large degree than a small degree; the opposite is true if the degree is small.

The algorithm lowers communication overhead by depending on the existence of marked neighbors as an indicator. Furthermore, a small degree of a node means that only a small number of its neighbors are probably marked, which reduces the computational cost.

This method requires less communication and accurately mimics the behavior of the algorithm by emphasizing local interactions. The algorithm's effectiveness lies in its ability to determine a node's degree based on whether or not it has marked neighbors, which minimizes the amount of communication and computation required.

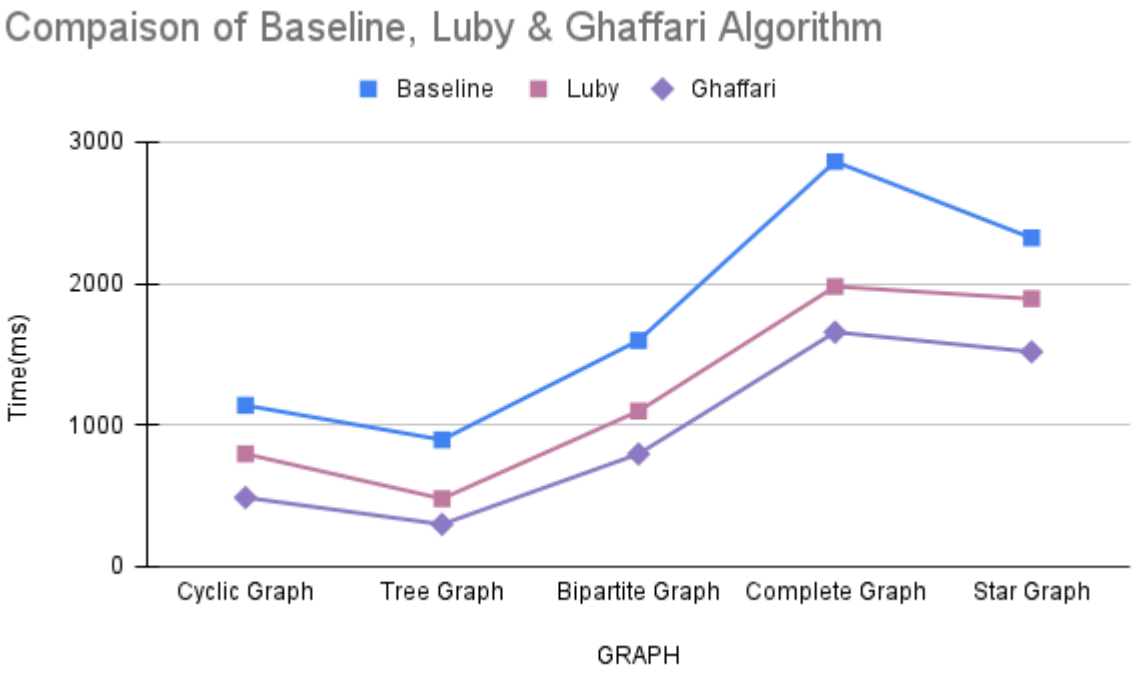
The Beeping MIS Algorithm: The algorithm works in iterations, each made of two rounds:

(R1) In the first round of iteration t, each node v *beeps* with probability $p_t(v)$ and remains silent otherwise. Initially $p_1(v) = 1/2$. If v beeps and all its neighbors are silent, then v joins the MIS. Node v updates its beeping probability for the next iteration as follows:

$$p_{t+1}(v) = \begin{cases} p_t(v)/2, & \text{if some neighbor of } v \text{ beeps,} \\ \min\{2p_t(v), \frac{1}{2}\}, & \text{otherwise.} \end{cases}$$

(R2) In the second round of iteration t, all nodes in the MIS beep. If a non-MIS node hears a beep, it learns that it has an MIS neighbor. MIS nodes and their neighbors get removed from the problem, for the next iterations.

2. Comparison Graph of Baseline , Luby's and Ghaffari Algorithm :



3 Observations

1. Tree graph: Trees are sparse graphs with no cycles, and their structure makes it easier to find independent sets efficiently. Each node in a tree can be part of the maximal independent set without conflicts, leading to a straightforward and efficient algorithm.

2. Cycle graph: Cycle graphs also have a relatively simple structure, consisting of a single cycle. While slightly more complex than trees, they are still sparse and lack the dense connectivity of complete graphs. Hence, finding a maximal independent set in a cycle graph tends to be faster than in denser graphs.

3. Bipartite graph: Bipartite graphs have two sets of nodes with no edges within the sets but only between them. While they are sparse, finding a maximal independent set in a bipartite graph may involve more complex algorithms than in tree or cycle graphs due to the need to traverse both sets and handle potential conflicts.

4. Star graph: Star graphs have one central node connected to all other nodes. While still sparse, the central node can pose challenges for finding a maximal independent set efficiently, especially if the algorithm's strategy involves expanding the set iteratively from a starting point.

5. Complete graph: Complete graphs have an edge between every pair of distinct nodes, making them highly dense. Finding a maximal independent set in a complete graph involves considering all possible combinations of nodes, leading to significantly higher complexity compared to sparser graphs like trees or cycles. Hence, it's generally the slowest among the listed graph types for this problem.