Distributed Systems

Monsoon 2021

Lecture 24

International Institute of Information Technology

Hyderabad, India

Quick Recap

- Started studying distributed algorithms, mostly on graphs.
- Looked at problems such as
  - BFS
  - Spanning Tree (Asynchronous)
  - Shortest Paths.
  - Primitives such as Convergecast and Broadcast
  - MST

Maximal Independent Sets (MIS)

- For a graph (V, E), an independent set of nodes U, where $U \subseteq V$, is such that for each i and j in U, $(i, j) \notin E$.
- An independent set U is a maximal independent set if no proper superset of U is an independent set.
- A graph may have multiple MIS; perhaps of varying sizes.
- The largest sized independent set is the maximum independent set.
  - Application: wireless broadcast - allocation of frequency bands (mutex)
  - NP-complete

Maximal Independent Sets (MIS)

- For a graph (V, E), an independent set of nodes U, where $U \subseteq V$, is such that for each i and j in U, $(i, j) \notin E$.
- An independent set U is a maximal independent set if no proper superset of U is an independent set.
- Has been a very popular problem with ongoing research in distributed algorithms.

Maximal Independent Sets (MIS)

- Has been a very popular problem a problem with ongoing research in distributed algorithms.
- A greedy sequential algorithm is very easy to design.

Greedy Sequential Algorithm

```
Algorithm GreedyMIS(G)
Begin
   U = {}, S = V
   While (S is not empty) do
       Pick a vertex v in S and add v to U
       Remove v and all neighbors of v in S
   End-While
   Return U
End.
```

Maximal Independent Sets (MIS)

- Has been a very popular problem a problem with ongoing research in distributed algorithms.
- A greedy sequential algorithm is very easy to design.
- But such an algorithm can take a long time in the distributed setting.
- Need quick way to set apart neighbors so that no two neighbors join the MIS together.
- Problem is called as symmetry breaking.

Maximal Independent Sets (MIS)

- Need quick way to set apart neighbors so that no two neighbors join the MIS together.
- Problem is called as symmetry breaking.
- In the deterministic setting, it is shown that any algorithm will require nearly n^{ε} rounds.
- Randomization however is very helpful.

Luby's Algorithm for MIS

- The algorithm designed by Luby more than three decades ago works as follows.
- Every node maintains a state among {`active`, `inactive`}.
- Every active node v picks a random number 0 or 1 with probability ½ d(v).
  - 1 means it is interested in joining the MIS.
- Each such node then shares its choice with all its active neighbors.
- A node withdraws from the contest if it finds another neighbor of high degree that also chose 1.
- Such a withdrawing node still remains `active` for now.

Luby's Algorithm for MIS

- Every node maintains a state among {`active`, `inactive`}.
- Every active node picks a random number 0 or 1.
- Each such node then shares its choice with all its active neighbors.
- A node withdraws from the contest if it finds another neighbor of high degree that also chose 1. Such a withdrawing node still remains `active` for now.
- If an active node that chose 1 did not withdraw, then the node joins the MIS. Becomes `inactive`.
  - Indicates all its active neighbors of its decision to join MIS.
- Active nodes that have some neighbor joining an MIS turn `inactive`.

Luby's Algorithm for MIS

- It is not immediately clear as to how this randomization helps.
- It can be shown with some analysis that the algorithm requires O(log n) rounds with high probability.
- The above is shown by arguing that in each round, a fraction of the edges can be made redundant.
  - An edge is redundant if either of its end points are `inactive`.
  - This indicates that in O(log n) rounds, all edges are redundant.
- The analysis follows….

Analysis

- To carry out the analysis, define:
  - A vertex v is good if more than 1/3rd of its neighbors have degree less than d(v).
  - A vertex v is bad if at least 2/3rd of its neighbors have degree at least d(v).
  - An edge e is good if at least one endpoint of e is good.
  - An edge e is bad if both its endpoints are bad.

Analysis – Sketch

- Good vertices have enough low degree neighbors.
- The probability that a node chooses to join the MIS is inversely proportional to its degree.
  - Low degree is good!
- So at least one such low degree neighbor is in S with good probability.
  - Helps delete good vertices.
  - This in turn helps delete good edges.
- If we can show that there are enough good edges, then suffices if a fraction of them are deleted.

Analysis – Claim 1

- For every good vertex v with $d(v) > 0$, the probability that some neighbor w of v gets marked is at least $1 - e^{-1/6}$
- Proof: Pr(v is marked) = $\frac{1}{2d(v)}$.
- Since v is good, at least $d(v)/3$ neighbors have degree at most $d(v)$. Let w be such a neighbor.
- Pr(w is marked) = $\frac{1}{2d(w)} \geq \frac{1}{2d(v)}$.
- Pr(no low degree neighbor of v is marked) $\leq$

$$\left(1 - \frac{1}{2d(v)}\right)^{d(v)/3} = e^{-1/6}.$$

- We did not consider the actions of the high degree neighbors in the above calculation.

Analysis – Claim 2

- If a vertex w is marked, then Pr(w in S) $\geq$ ½.
- Proof: If w is marked, then w is not in S only if some high degree neighbor of w is also marked.
- Each such high degree neighbors of w is marked with probability at most ½d(w).
- Number of such high-degree neighbors $\leq$ d(w).

$$
\begin{aligned}
\Pr(w \text{ in } S \mid w \text{ is marked}) &= 1 - \Pr(w \text{ not in } S \mid w \text{ is marked}) \\
&= 1 - \Pr(\exists u \in N(w),\ d(u) \geq d(w),\ u \text{ marked}) \\
&\geq 1 - |u \in N(w),\ d(u) \geq d(w)|\ \tfrac{1}{2d(w)} \\
&\geq 1 - |u \in N(w)|\ \tfrac{1}{2d(w)} = 1 - d(w).\ \tfrac{1}{2d(w)} \\
&= \tfrac{1}{2}.
\end{aligned}
$$

Analysis – Claim 3

- Let v be a good vertex. Then,

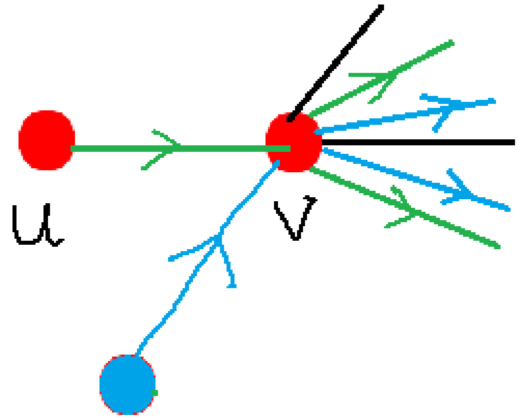
  $$\Pr(v \text{ is deleted}) \geq (1 - e^{-1/6})/2.$$

- Proof: Combine Claims 1 and 2.

Analysis – Claim 4

- At least half the edges are good.
- Proof: For every bad edge e, associate a pair of edges via a function $f:E_B \to \binom{E}{2}$ such that for any two distinct bad edges e_1 and e_2, $f(e_1) \cap f(e_2) = \emptyset$.
- Completes the proof since only |E|/2 such pairs exist.
- The function f defined as follows.

Analysis – Claim 4



- For each edge (u, v) ∈ E, orient it towards the vertex of higher degree.
- Consider a bad edge e = (u,v) oriented towards v.
- Since v is bad, the out-degree of v is at least twice its in-degree.
- So, there exists a way to pick a pair of edges for every bad edge.

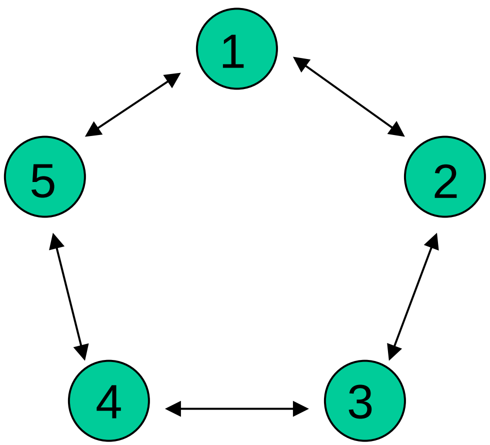
Putting Everything Together

- In each iteration, it is expected that a constant fraction of edges are deleted.
  - Half the edges are good, and a good edge is deleted with probability at least $(1 - e^{-1/6})/2$.
- So, on expectation, only O(log m) = O(log n) iterations suffice.
- Can also show that with high probability O(log n) iterations suffice.

Leader Election

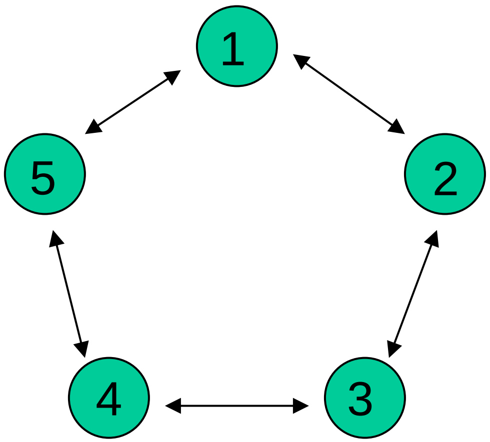
- Another problem that is studied in the distributed setting is leader election.
- The problem is defined as follows.
- Let G be a (directed) graph (V, E). One of the nodes of V has to be designated as the leader.
- Several applications
  - We saw an application to the 2-phase commit algorithm also.

Leader Election



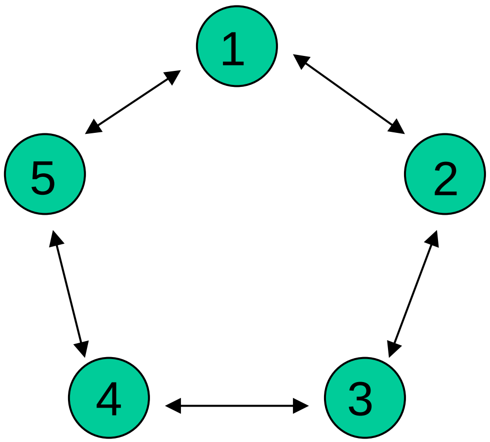
- As a simple example, consider the case where G is a (bidirectional) directed ring.
- Processes can distinguish clockwise from counterclockwise neighbor.

Leader Election



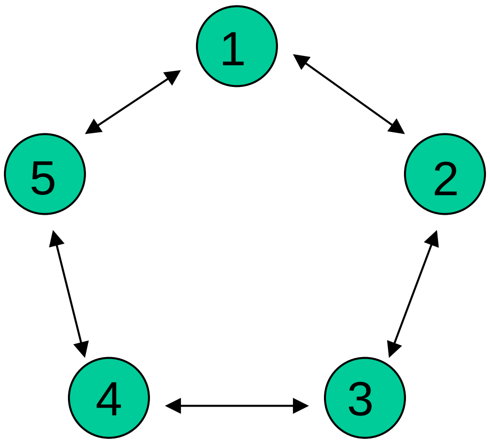
- General idea:
  - Every process passes its UID around the ring
  - Upon reception, compare to own UID
    - If larger: Pass to next process
    - If smaller: Discard
    - If equal: Declare self as leader

Leader Election



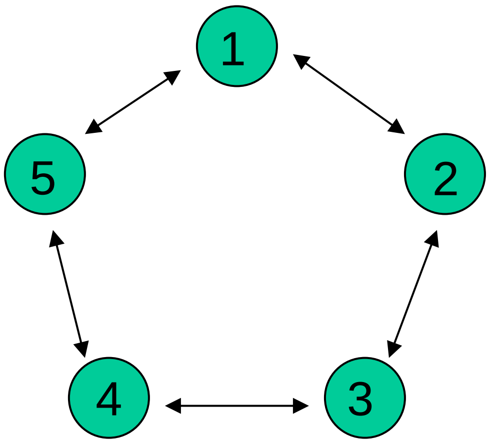
- Proof by induction on r
  - After r rounds, the node r away from the P_{max} will be sending UID_{max}
  - When $r = n$, UID_{max} will have traveled around the ring

Leader Election



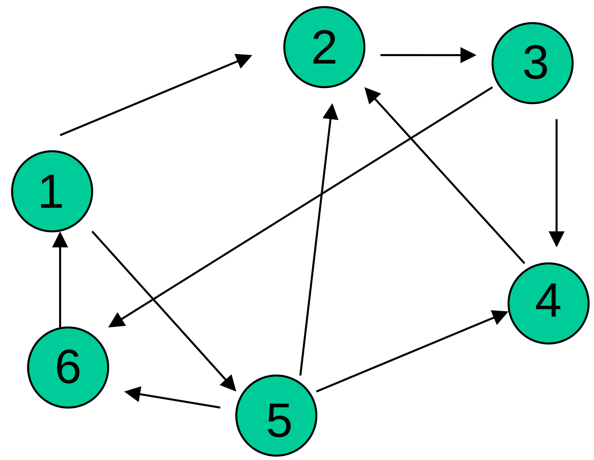
- An improvement is given by Hirshberg and Sinclair
- Achieves O(n log n) communication complexity, O(n) time complexity

Leader Election



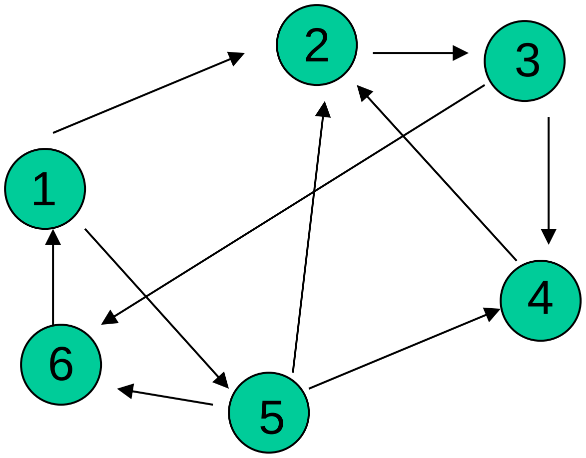
- General idea:
  - Operates in phases
  - During phase *p*, send UID in both directions for a distance of 2^p
    - Processing of received UID same as earlier
    - If its UID comes back, continue to next round
    - If it receives its outbound UID, declare self as leader

General Graphs



- For general graphs, one of the popular algorithms is the FloodMax algorithm.
- General idea:
  - Each process remembers the largest UID it has seen
  - For each round, propagate UID_{max} on all outgoing links
  - After *diam* rounds, if UID_{max} is own UID, declare self as leader

General Graphs



- Communication complexity is *diam* · |*E*|
  - |*E*| is the number of outgoing edges