Greedy Sequential Maximal Independent Set and Matching are Parallel on Average

Julian Shun

Joint work with

Guy Blelloch and Jeremy Fineman

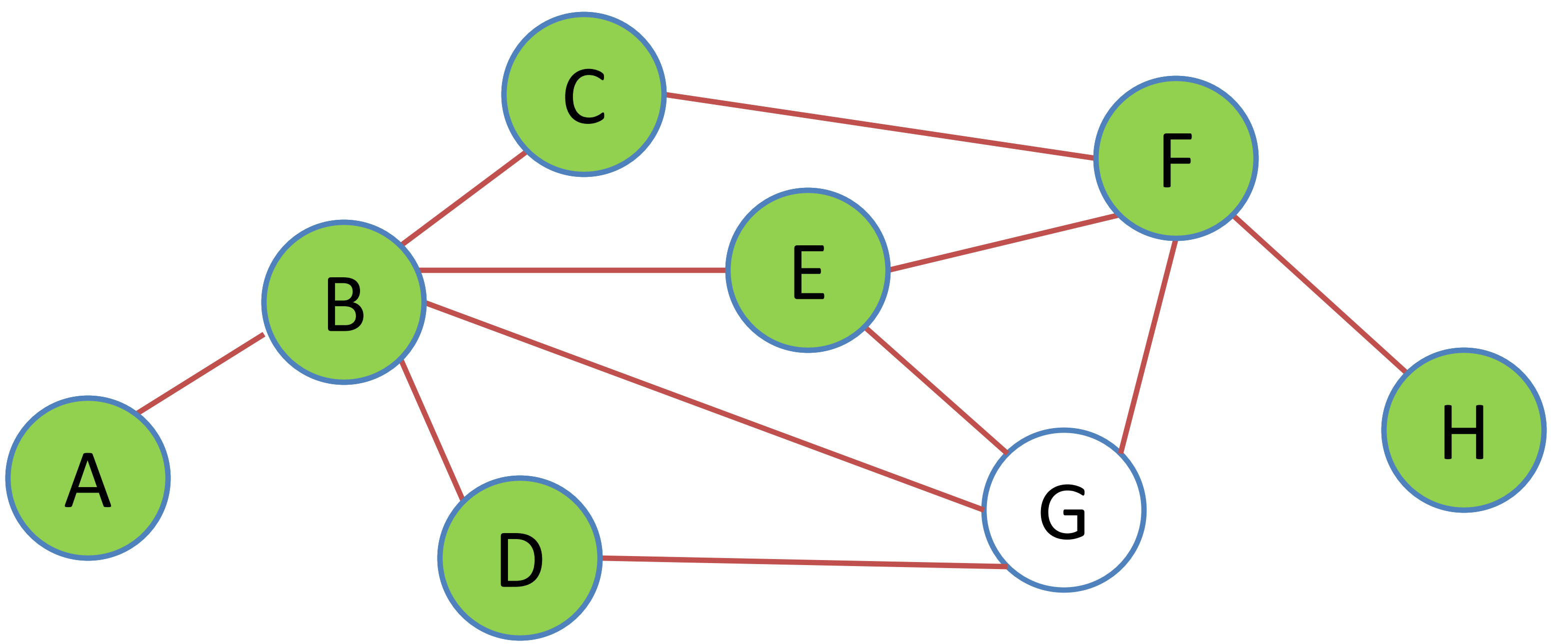
(Paper in SPAA 2012)

Outline

- Introduction
  - Definitions and sequential algorithm for Maximal Independent Set
- Luby’s Algorithm
- Parallel Greedy algorithm
- Analysis of Parallel Greedy algorithm
- Experiments

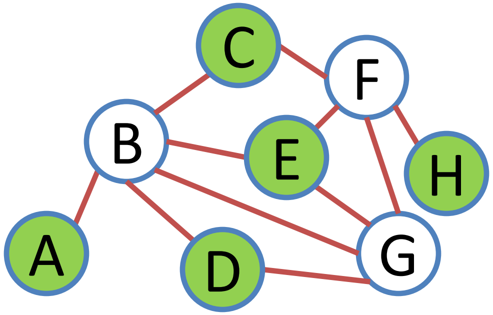
Maximal Independent Set (MIS)

- Undirected graph G = (V, E)
- Return a subset $U \subseteq V$ such that
  1. $U \bigcap N(U) = \emptyset$ (Independent set)
  2. $\forall v \in V \setminus U, \quad N(v) \bigcap U \neq \emptyset$ (maximal)



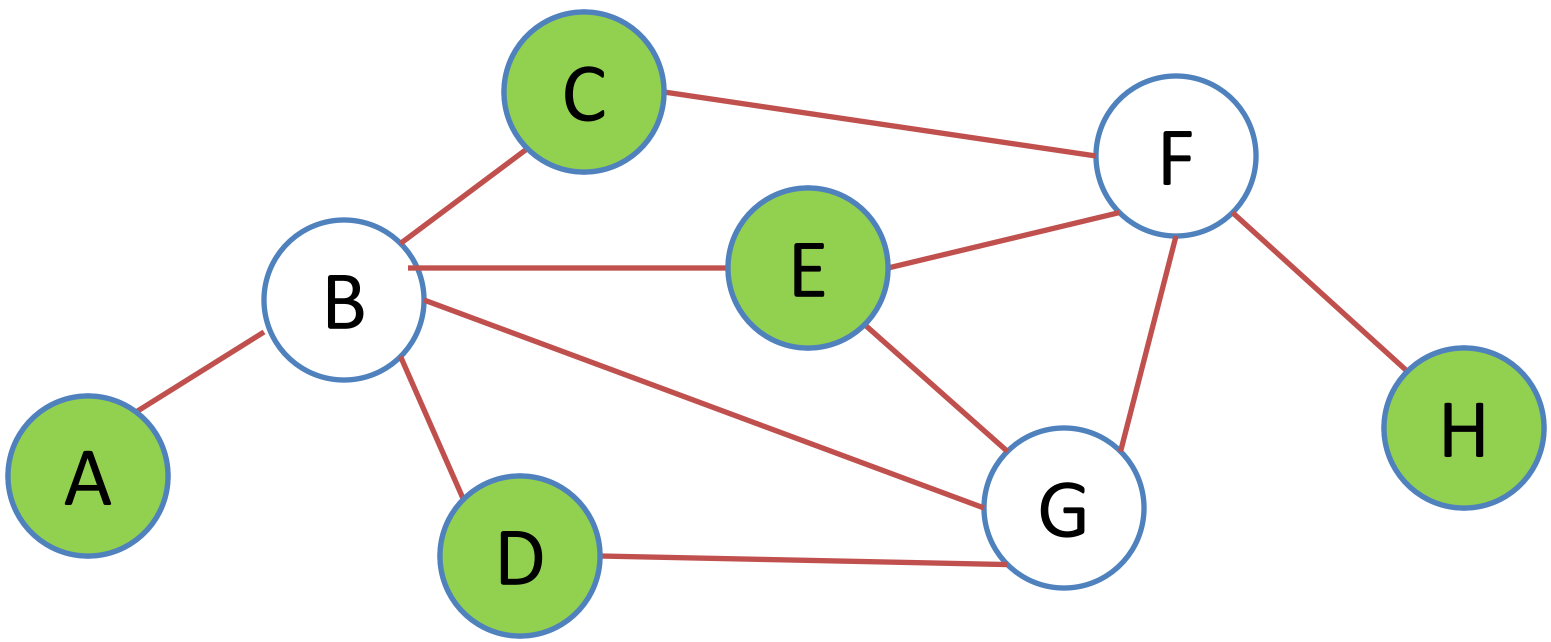
Motivation

- Why do we care about maximal independent sets (MIS)?
  - Used as a subroutine in many parallel algorithms to identify "independent" parts of graph that can be processed simultaneously
  - Map/graph coloring, scheduling, computational biology, distributed computing etc.



Sequential greedy algorithm

- MIS corresponding sequentially processing the vertices in order
  - This has been called the <u>lexicographically first ordering</u>

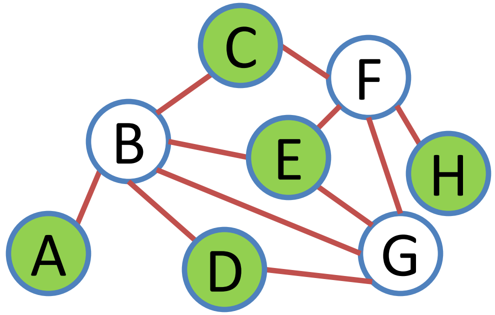


Outline

- Introduction
  - Definitions and sequential algorithm for Maximal Independent Set
- Luby's Algorithm
- Parallel Greedy algorithm
- Analysis of Parallel Greedy algorithm
- Experiments

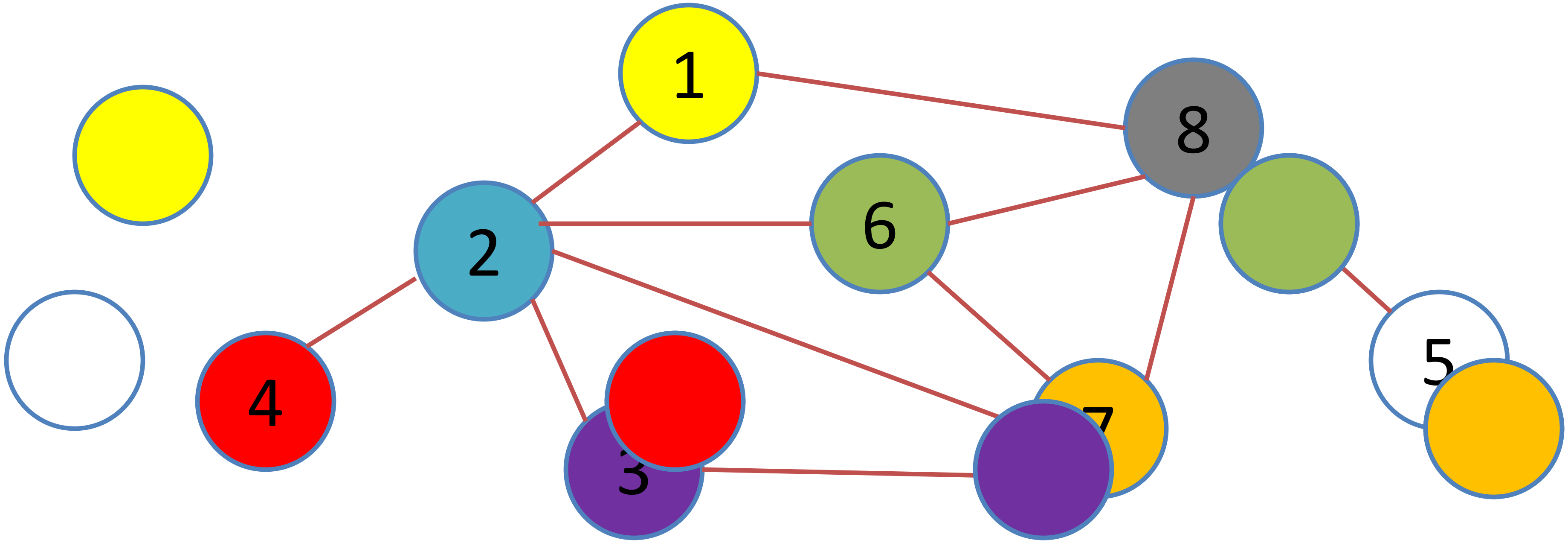
Luby's algorithm

- Each round:
  - assign random priorities to all vertices
  - vertices with a priority greater than all of its neighbors' priorities join the MIS
  - remove vertices in MIS and all of their neighbors
- Repeat this process until no vertices remain



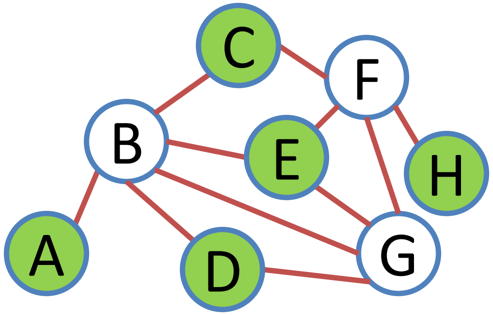
Luby's algorithm

- Each round:
  - assign random priorities to all vertices
  - vertices with a priority higher than all of its neighbors' priorities join the MIS *(smaller number → higher priority!)*
  - remove vertices in MIS and all of their neighbors
- Repeat this process until no vertices remain

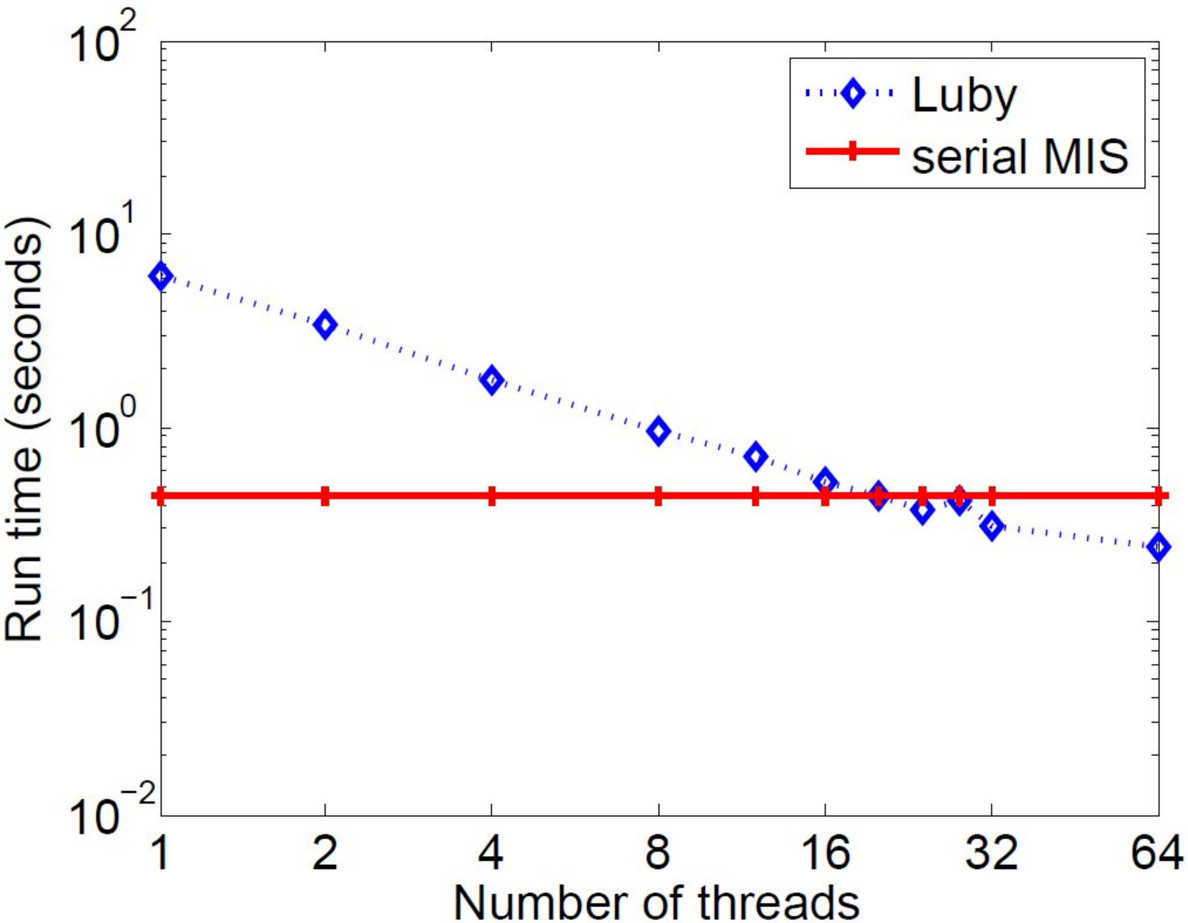


Luby's algorithm

- Requires O(log m) depth and O(m log m) work
- Can be made to run in O($\log^2$ m) depth and O(m) work
  - Pack vertices/edges each iteration

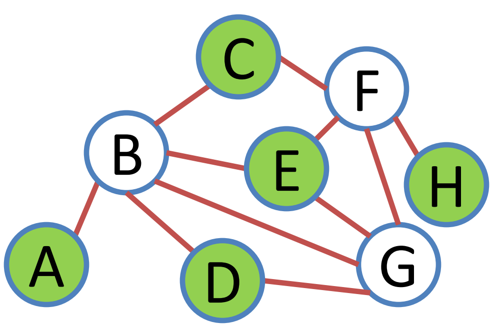


Luby vs. Sequential greedy



(a) Running time vs. number of threads on a **random graph** ($n = 10^7, m = 5 \times 10^7$) in log-log scale

- Since sequential greedy implementation is so simple, it is hard for a parallel implementation to beat it
- Luby wins after 16 threads
- Note: Luby does not return the same answer as sequential

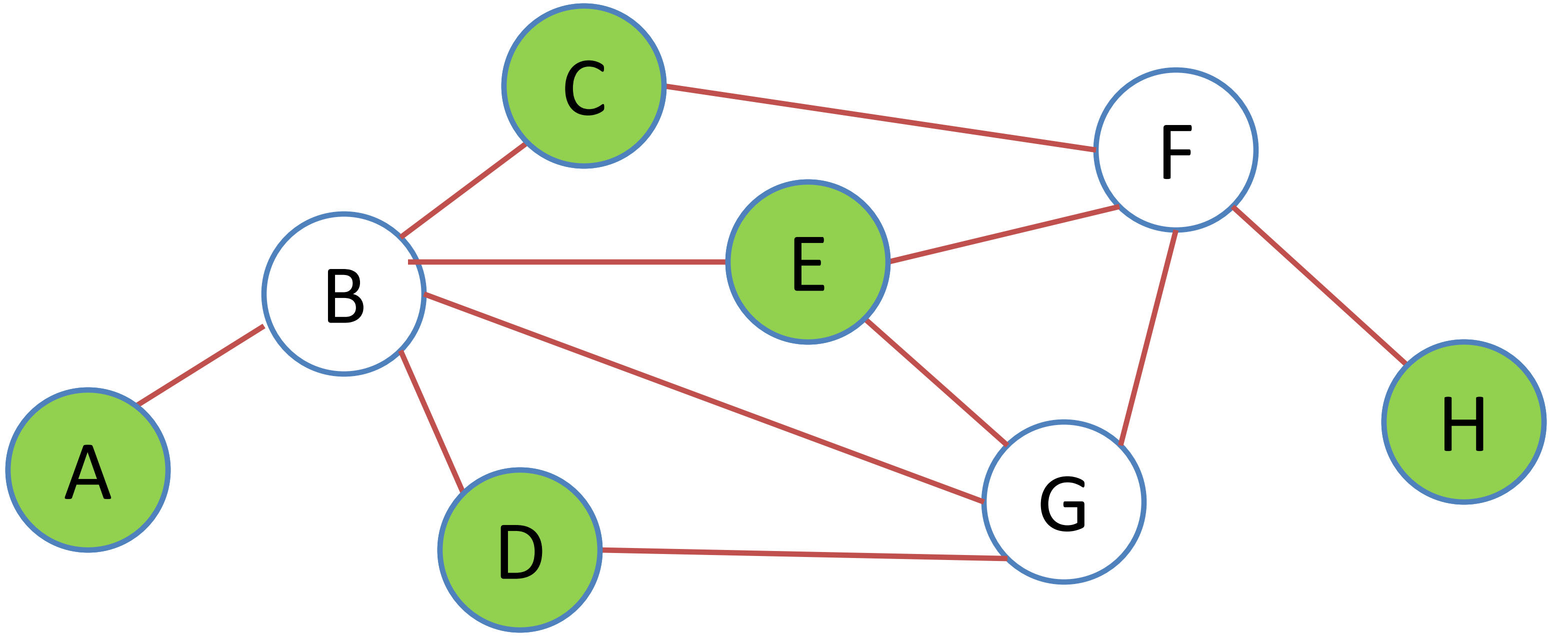


Outline

- Introduction
  - Definitions and sequential algorithm for Maximal Independent Set
- Luby’s Algorithm
- Parallel Greedy algorithm
- Analysis of Parallel Greedy algorithm
- Experiments

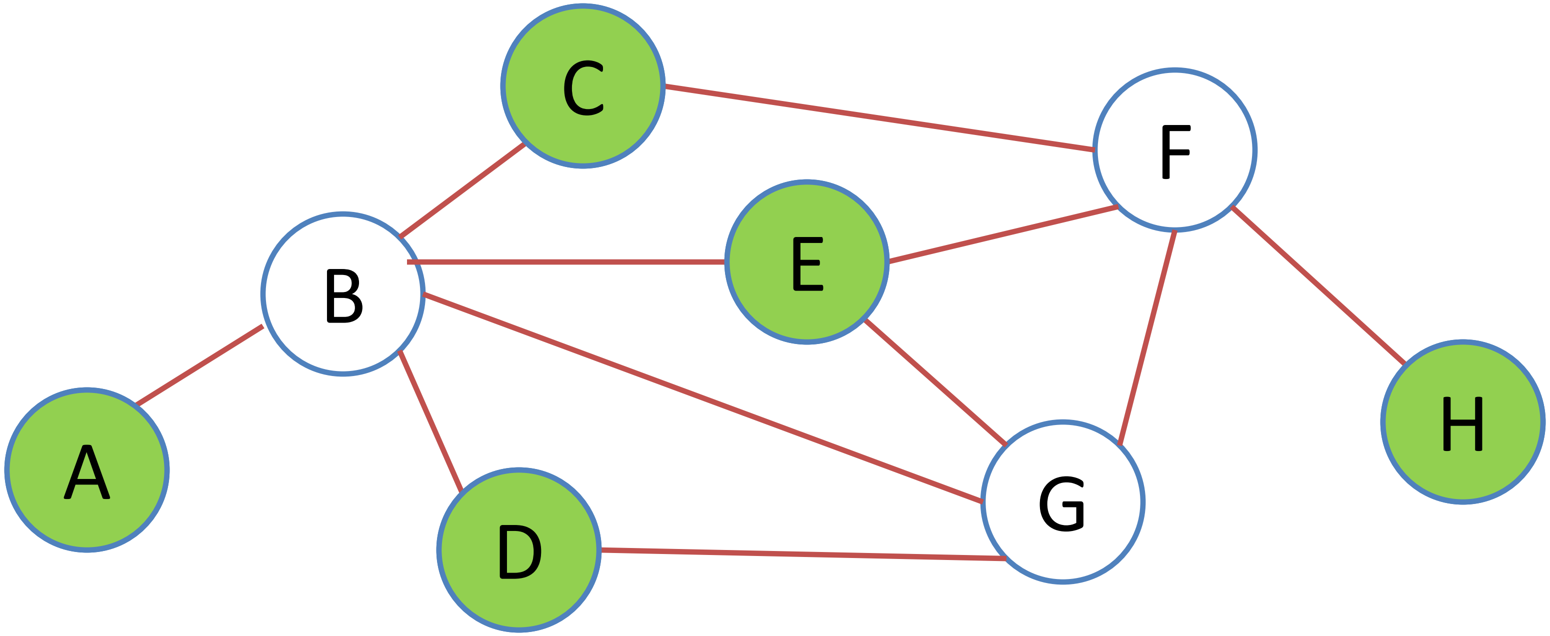
Sequential greedy algorithm

- MIS corresponding sequentially processing the vertices in order
  - This has been called the lexicographically first ordering



“Sequential” greedy algorithm

- Note that some vertices may be processed in parallel



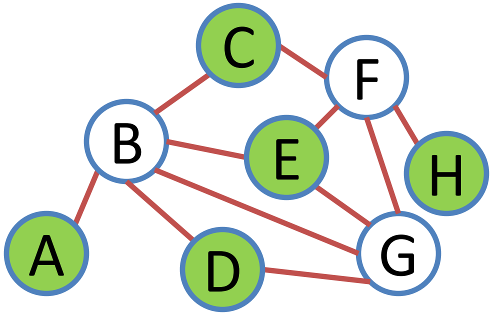
Parallel-Greedy vs. Luby's algorithm

Parallel-Greedy

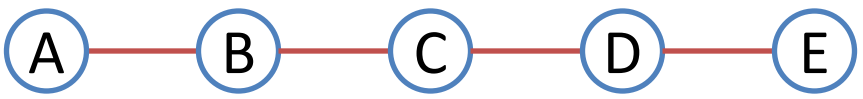
- Randomly order the vertices
- While vertices remain:
  - **In parallel:** vertices with higher priority than all of their neighbors join the MIS
  - Remove vertices in MIS and all of their neighbors from the graph

How many iterations does Parallel-Greedy take?

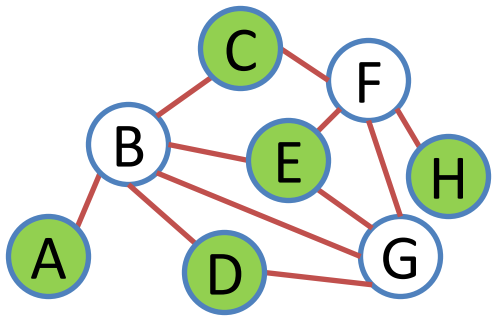
Luby



Parallel-Greedy

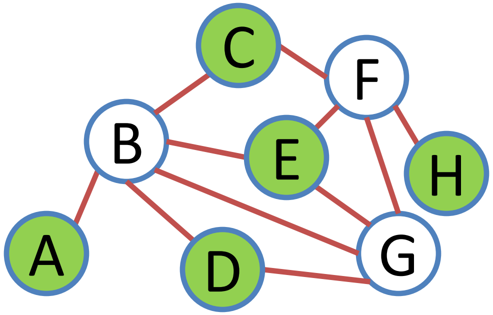
- How many iterations does this algorithm take?
  - For an arbitrary ordering, it could take O(n) iterations
    - Example: 

  - What about for a random ordering?
    - This talk: we show that the number of iterations is polylogarithmic



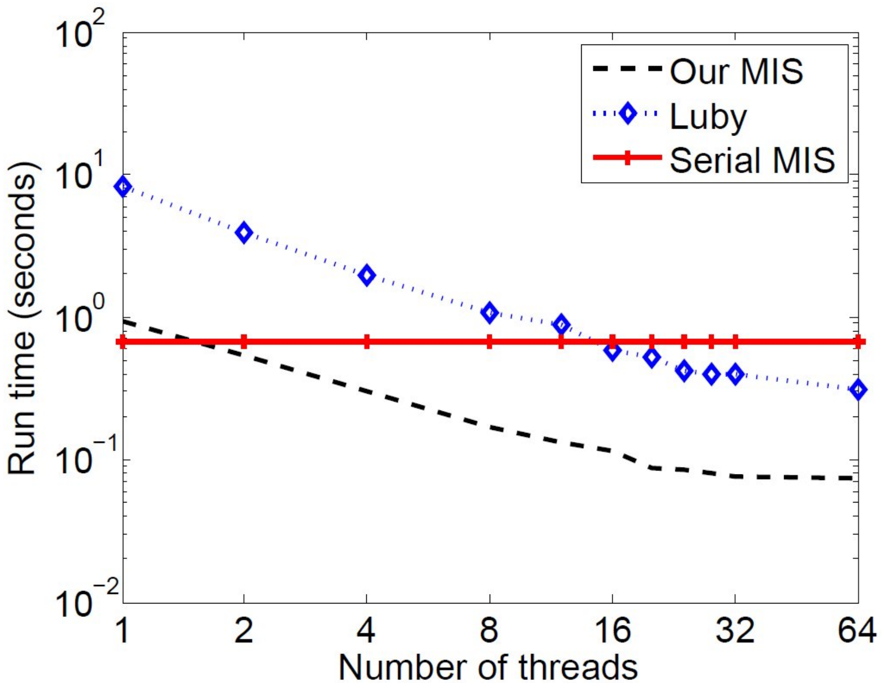
Related Work for Parallel-Greedy

- For **arbitrary graphs** and **arbitrary orderings**, this problem was proved to be P-complete (Cook '85)
- For uniform **random graphs**, this problem was shown to have polylog depth (Coppersmith et al. '89 showed a depth of $O(\log^2 n)$; Calkin and Frieze '90 improved the depth to $O(\log n)$)
- This talk: for **arbitrary graphs** and **random orderings**, this problem has $O(\log^2 n)$ depth
- Depth recently improved to $\Theta(\log n)$ [Fischer and Noever, SODA 2018]

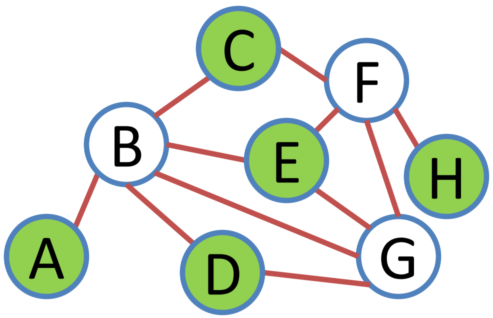


Practical Benefits

- Performance, fast runtime (by using prefixes)
- Guarantees the same result as the sequential algorithm's output every time
  - Such determinism allows for ease of debugging, verification of correctness, reasoning about code, etc.



(b) Running time vs. number of threads on a **rMat graph** ($n = 2^{24}, m = 5 \times 10^7$) in log-log scale



Outline

- Introduction
  - Definitions and sequential algorithm for Maximal Independent Set
- Luby’s Algorithm
- Parallel Greedy algorithm
- Analysis of Parallel Greedy algorithm
- Experiments

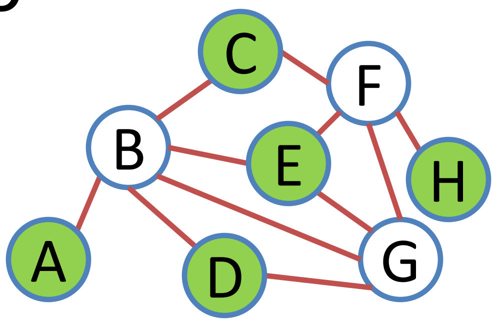
Analysis

Luby's MIS Algorithm		Parallel-Greedy Algorithm	
Work	**Depth**	**Work**	**Depth**
$O(m \log m)$	$O(\log m)$	$O(m \log^2 m)$	$O(\log^2 m)$
$O(m)$	$O(\log^2 m)$	$O(m)$	$O(\log^3 m)$

- Luby's analysis relies on the iterations being independent since ordering is regenerated per iteration
- For Parallel-Greedy, ordering is generated just once!
  - Requires other analysis techniques

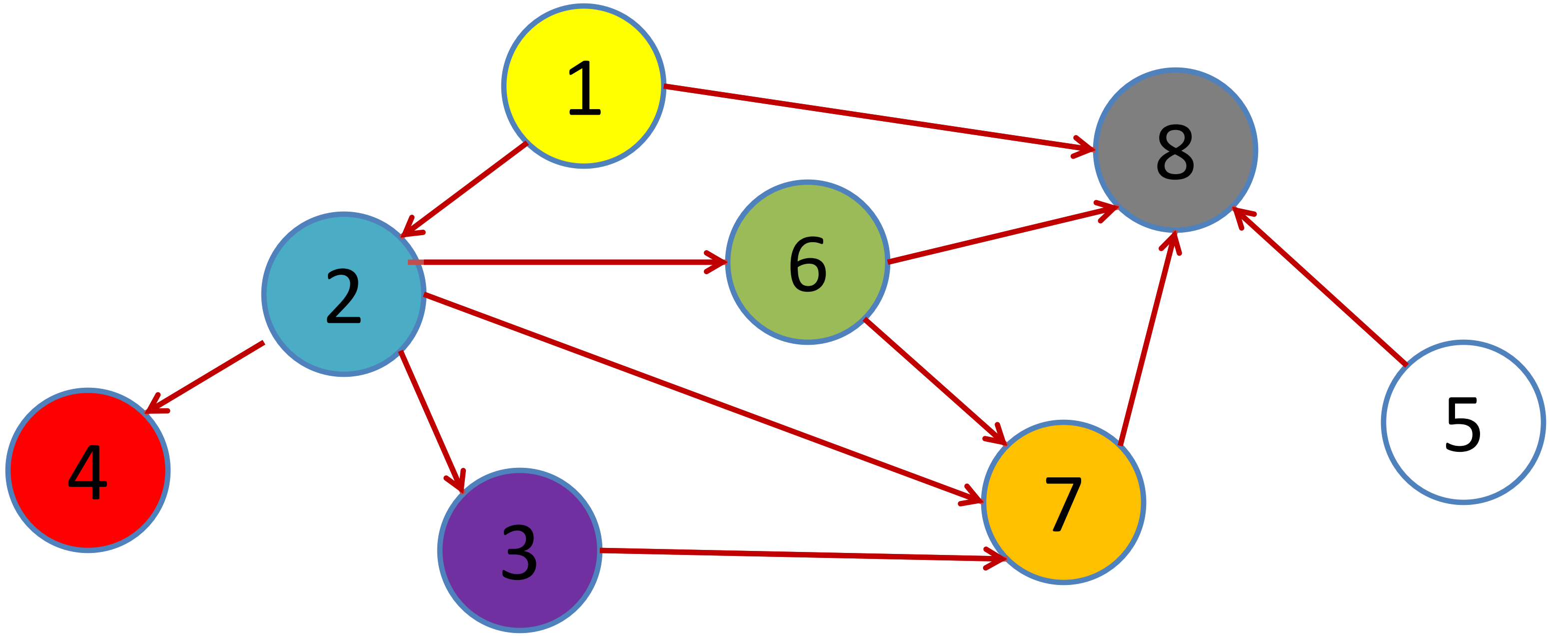
Priority-Directed Acyclic Graph (pDAG)

- For some set of vertices V, the priority-DAG is the vertex-induced subgraph of V, where each edge is directed from its higher priority endpoint to its lower priority endpoint
- The dependence length of a pDAG is the number of steps of Parallel-Greedy required to process the graph to completion
  - This is also the depth of a call to Parallel-Greedy

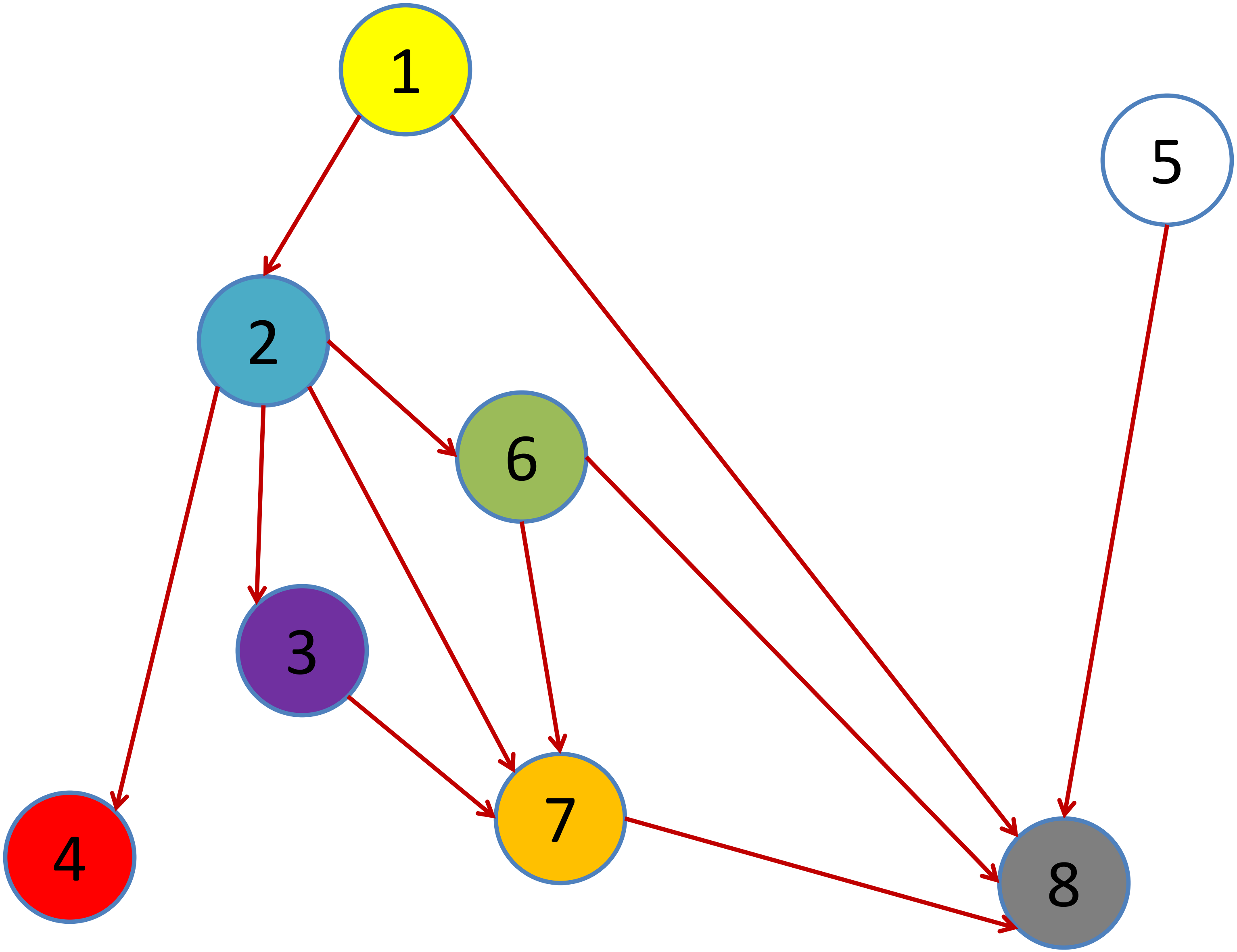


Priority-Directed Acyclic Graph (pDAG)

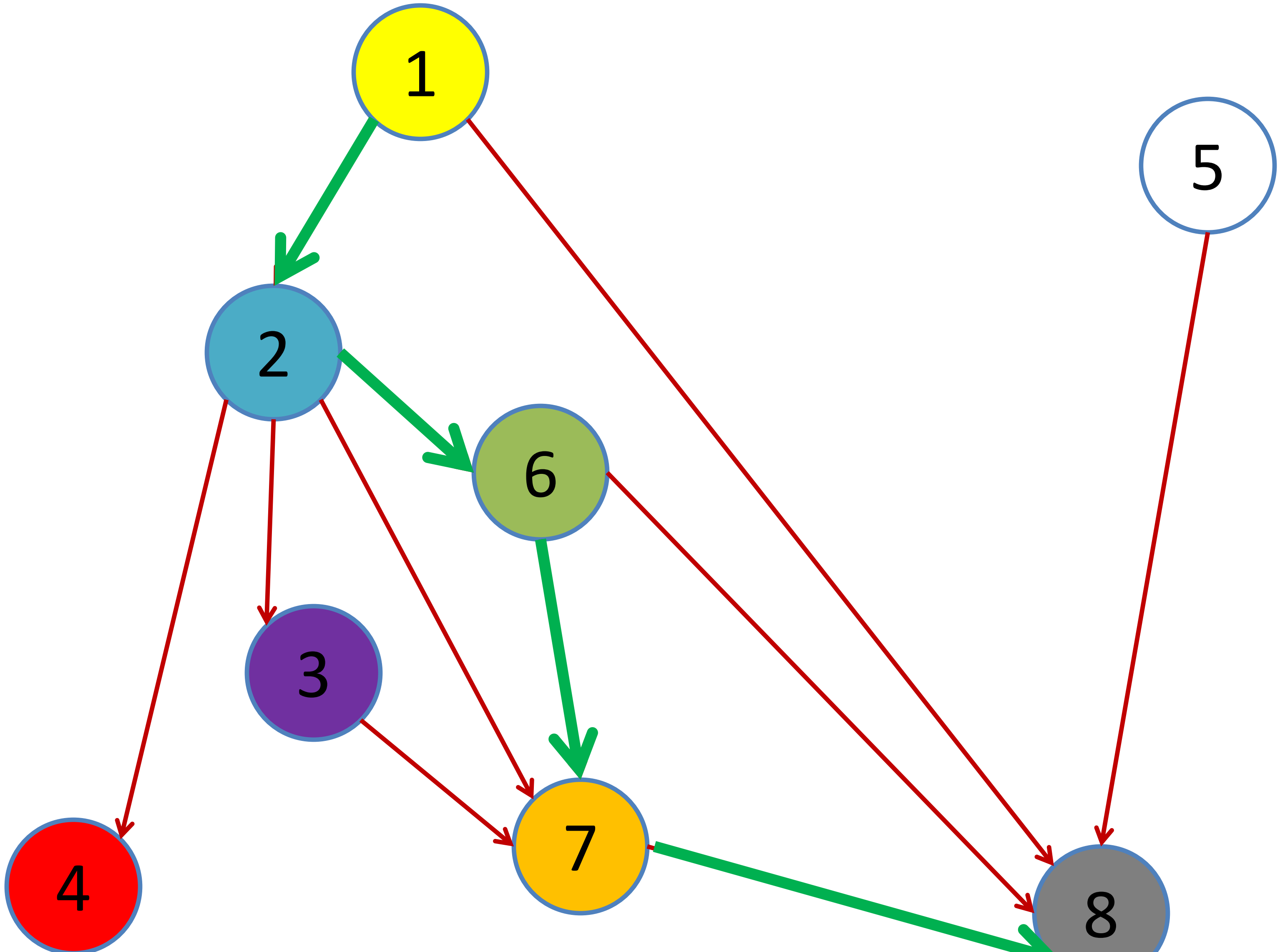
- Another way to view parallel algorithm is to repeat the following until no vertices remain:
  - Put all “roots” in MIS, remove roots, all neighbors of roots, and any incident edges



Priority-Directed Acyclic Graph (pDAG)



Priority-Directed Acyclic Graph (pDAG)

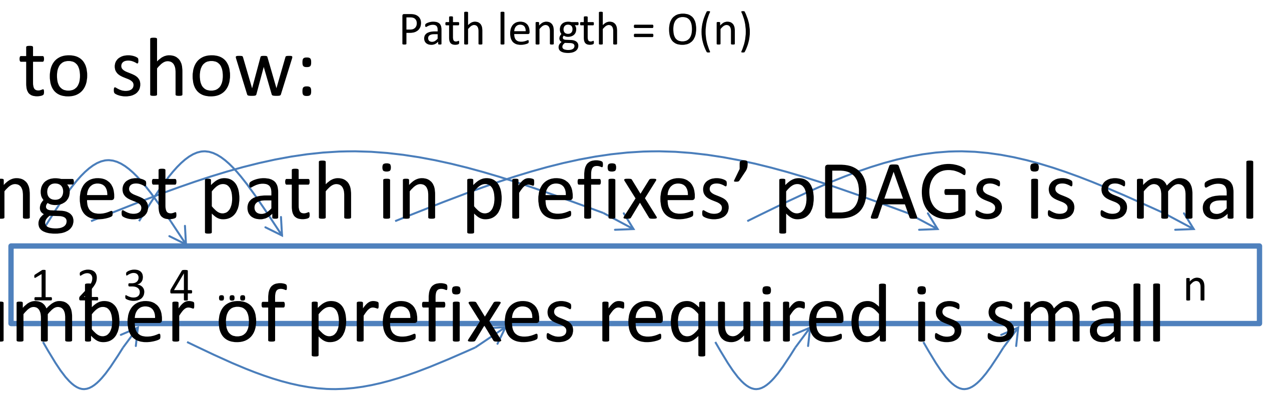


Priority-Directed Acyclic Graph (pDAG)

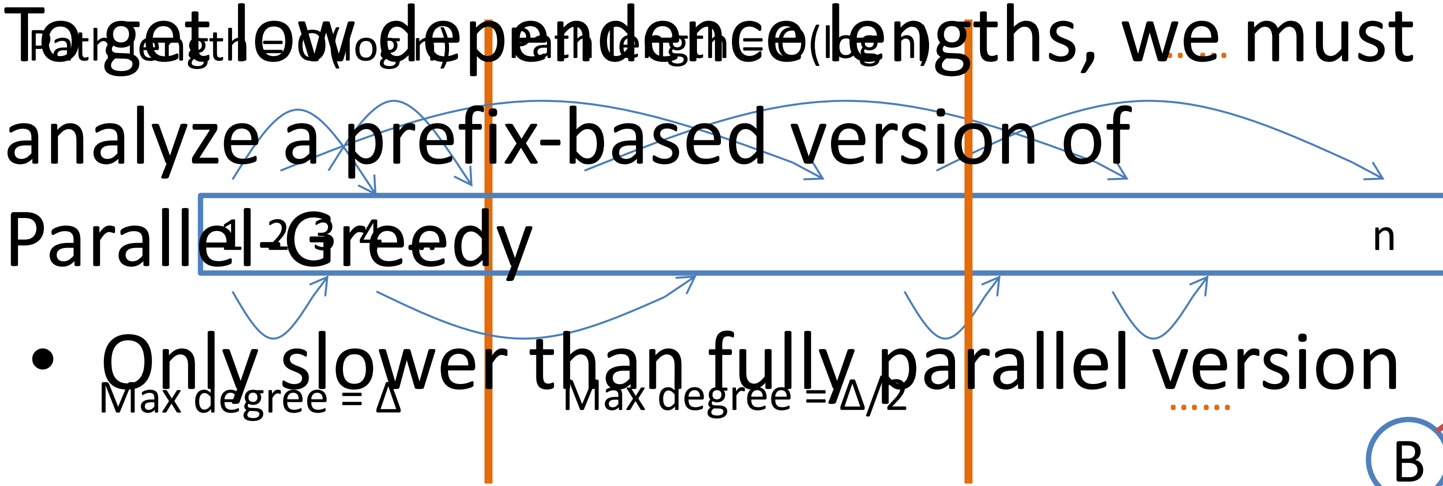
- The dependence length is upper bounded by the longest directed path in the pDAG, but could be much less
  - Ex: A complete graph has a directed path of length O(n) but the dependence length is O(1)

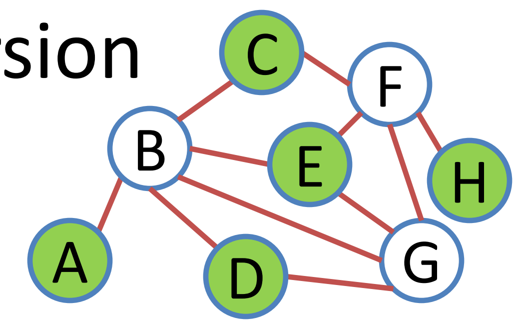
Prefix-based MIS algorithm

- Need to show:
  - Longest path in prefixes' pDAGs is small
  - Number of prefixes required is small



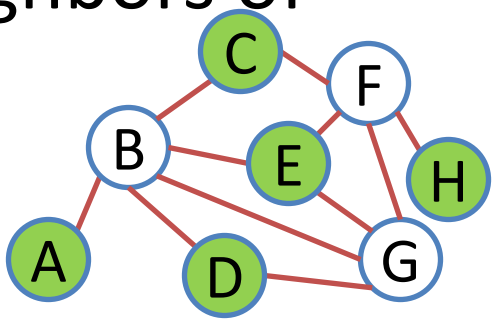
- To get low dependence lengths, we must analyze a prefix-based version of Parallel-Greedy
  - Only slower than fully parallel version





Prefix-based MIS algorithm

- Randomly order the vertices
- While vertices remain:
  - Choose a prefix parameter δ (a fraction)
  - Take the δn highest priority vertices in prefix
  - Run Parallel-Greedy until completion on induced subgraph of prefix vertices
  - Remove prefix vertices and neighbors of MIS from graph



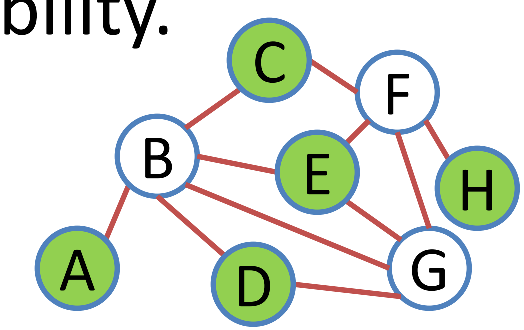
Number of rounds is small

- Randomly order the vertices
- While vertices remain:
  - Choose a <u>prefix</u> parameter δ
  - Take the δ|V| highest priority vertices in prefix
  - Run Parallel-Greedy until completion on induced subgraph of prefix vertices
  - Remove prefix vertices and neighbors of MIS from graph

- We only need to pick prefixes for log Δ rounds, where Δ is the maximum degree in original graph
- *Theorem:* Set $\delta = O(2^i \log(n)/\Delta)$ for the i'th round (Δ = max degree in original graph). Then all remaining vertices after the i'th round have degree at most $\Delta/2^i$ with high probability.
- *Proof:* Consider sequential process of randomly picking a vertex, adding it to MIS and removing its neighbors.
  - Probability a vertex of degree ≥ d is still around after δn steps is at most

$$\left(1-\frac{d}{n}\right)^{\delta n} < e^{-c \ln n} = \frac{1}{n^c}$$

  - Take union bound over all vertices



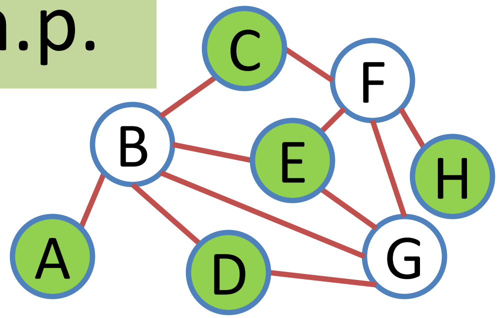
pDAG of each prefix is shallow

- *Theorem:* For a δ-prefix where $\delta = O(2^i \log(n)/\Delta)$, longest path in pDAG is length $O(\log n)$ w.h.p.
- *Proof (sketch):*
  - Number of possible k-length paths is at most d^k.
  - Probability of the path existing entirely in the prefix is δ^k.
  - Probability that the path is directed is $1/k!$
  - Union bound: $n\left(\frac{d^k \delta^k}{k!}\right) \le n\left(\frac{ed\delta}{k}\right)^k = n\left(\frac{e\log n}{k}\right)^k = \frac{1}{n^c}$
  - Plug in $\delta = O(2^i \log(n)/\Delta)$ and $d = \Delta/2^i$ from before and $k = O(\log n)$ yields high probability.

Prefix-based MIS algorithm

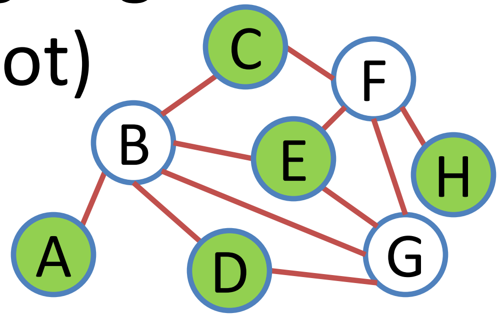
- We showed that the dependence length of the prefix's pDAG is small O(log n) w.h.p.
- We also showed that the number of prefixes taken until all vertices are removed is small O(log Δ) w.h.p.
- Hence the depth of the whole algorithm is small

$O(\log n) \times O(\log \Delta) = O(\log^2 n)$ w.h.p.



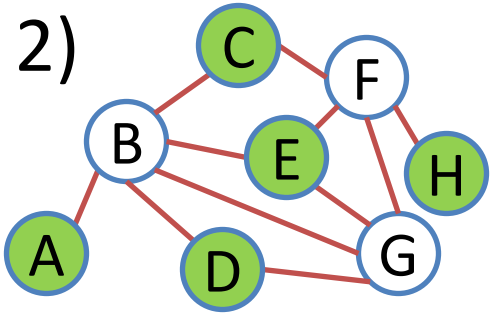
Achieving linear work

- Straightforward implementation will require O(m) work per layer of each pDAG, giving $O(m \log^2 n)$ total work
- Linear-work implementation: For each pDAG, keep an array of roots
  - Each vertex has incoming edges in an array, and a pointer initially to the start of the array
  - In any round if a vertex has an edge deleted, it checks whether all of its incoming edges are deleted (and if so it becomes a root)



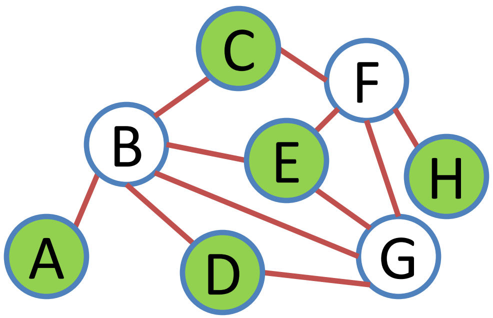
Achieving linear work

- 1) For each pDAG, keep an array of roots
- 2) Checking
  - In any round if a vertex has an edge deleted, it marks all deleted edges, and checks whether all of its incoming edges are deleted
  - We examine edges in powers of 2: first examine one parent, then two, then four…
  - If we see an incoming edge not deleted, we stop and charge the work of checking to all previous edges (within a factor of 2)



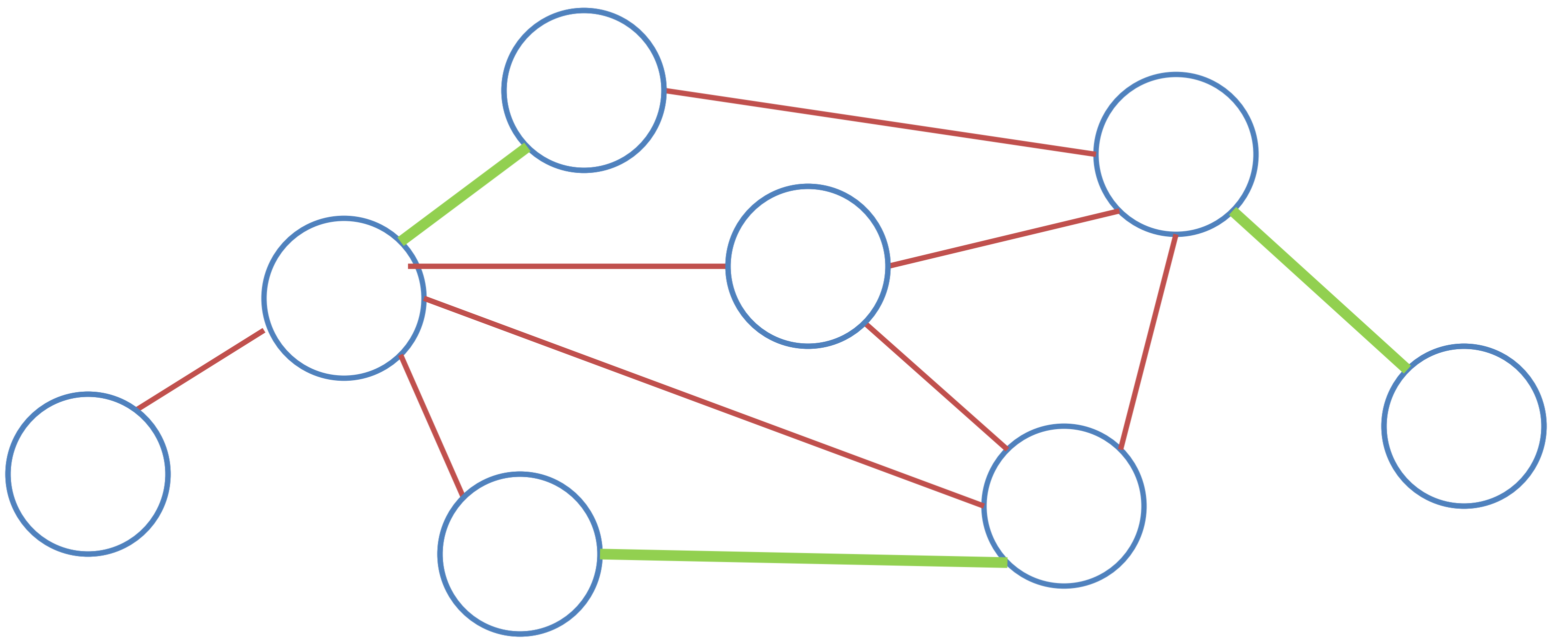
Achieving linear work

- 3) When a vertex is added to MIS it deletes all neighbors and checks all neighbors' neighbors, adding them to array of roots if necessary
  - Eliminate duplicates by using concurrent writes and packing
- Total cost of checks is O(m). Checking and packing requires O(log m) additional depth.
- This gives O(m) work and O($\log^3$ m) depth overall



Maximal Matching (MM)

- Given an undirected graph G = (V, E), return a subset $E' \subseteq E$ such that no edges in E' share an endpoint and all edges in $E \setminus E'$ have a neighboring edge in E'



Maximal Matching

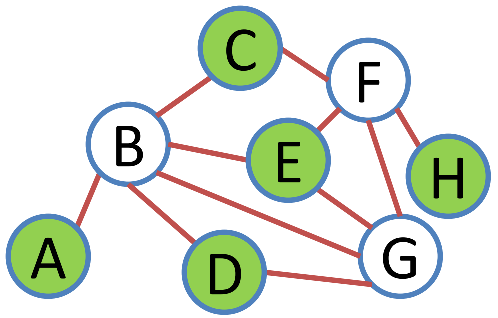
- By using same analysis as MIS, implicitly processing the line graph, we get a depth of $O(\log^2 m)$ w.h.p.
  - Line graph G': vertices in G' correspond to edges in G, and an edge exists between two vertices in G' if and only if the corresponding edges in G share an endpoint
- We can achieve linear work with an extra factor of $O(\log m)$ in the depth.
- This gives $O(m)$ work and $O(\log^3 m)$ depth overall.

Outline

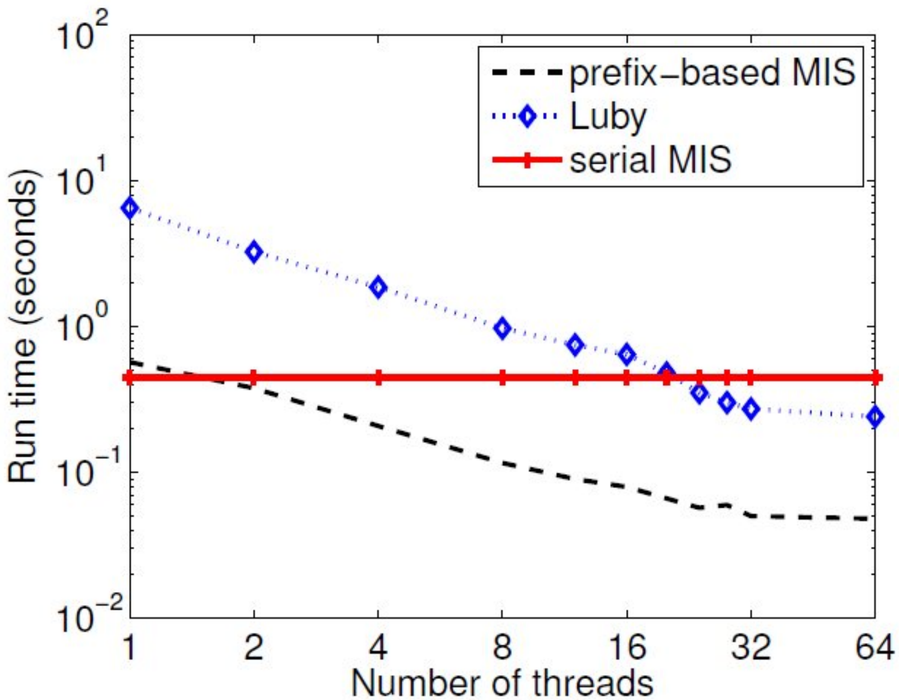
- Introduction
  - Definitions and sequential algorithm for Maximal Independent Set
- Luby's Algorithm
- Parallel Greedy algorithm
- Analysis of Parallel Greedy algorithm
- Experiments

Implementations

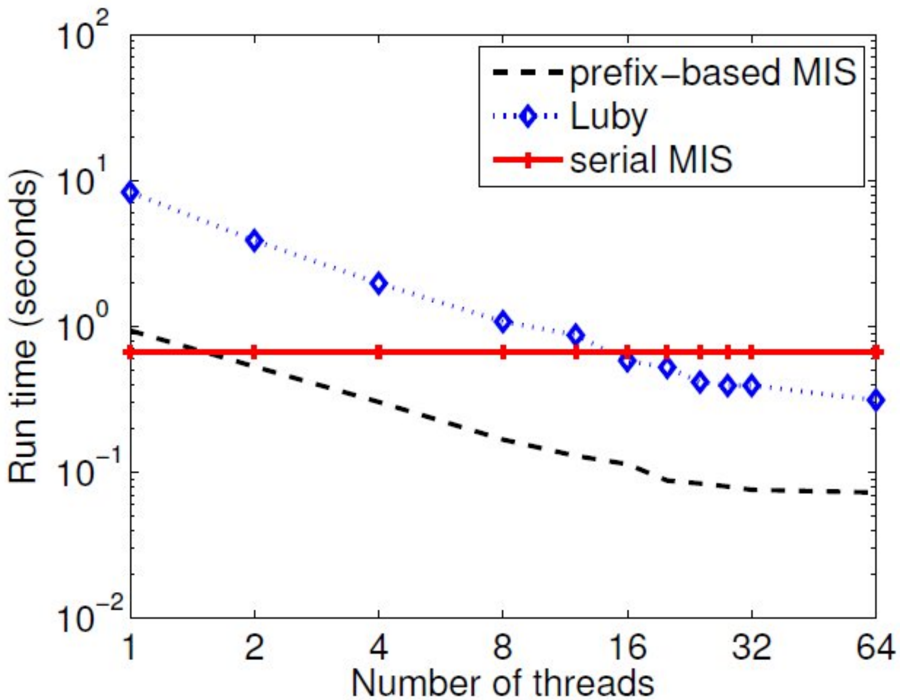
- Implemented using a fixed prefix size
  - Motivated theoretically
  - Reduces redundant work and improves running time
  - Technique of using prefixes also applied to other deterministic algorithms [Blelloch, Fineman, Gibbons, Shun, PPoPP 2012]



Experiments (MIS)

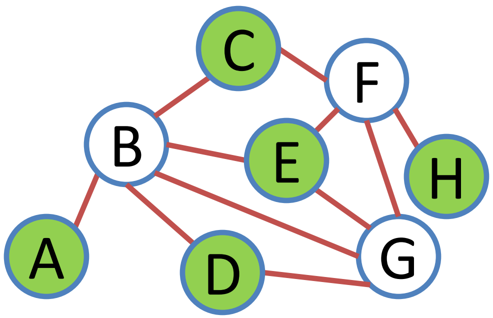


(a) Running time vs. number of threads on a **random graph** ($n = 10^7, m = 5 \times 10^7$) in log-log scale

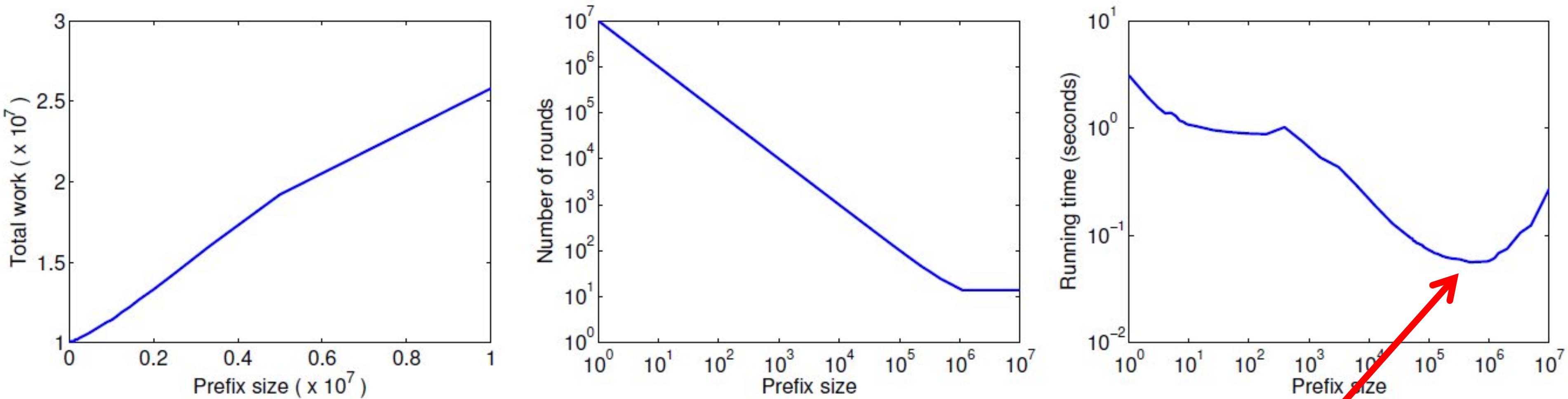


(b) Running time vs. number of threads on a **rMat graph** ($n = 2^{24}, m = 5 \times 10^7$) in log-log scale

- 32-core Intel Nehalem with hyperthreading
- Used an "optimal" prefix size
- prefix-based MIS 3x to 8x faster than Luby's MIS



Experiments (MIS)

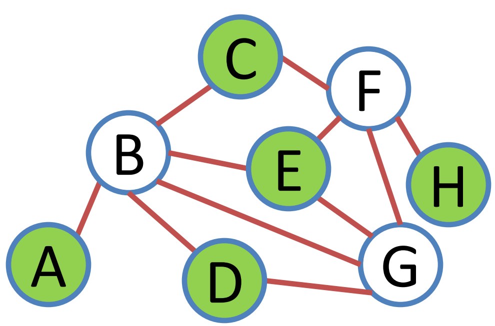


(d) Total work done vs. prefix size on a **rMat graph** ($n = 2^{24}, m = 5 \times 10^7$)

(e) Number of rounds vs. prefix size on a **rMat graph** ($n = 2^{24}, m = 5 \times 10^7$) in log-log scale

(f) Running time (32 processors) vs. prefix size on a **rMat graph** ($n = 2^{24}, m = 5 \times 10^7$) in log-log scale

- Work increases with larger prefix size (more redundant work)
- Number of rounds decreases with larger prefix size (more parallelism)
- There is some optimal prefix size which results in the lowest running time



Conclusions

- Sequential greedy MIS algorithm on arbitrary graphs for random orderings is actually parallel
- With some modification we obtain similar results for greedy maximal matching
- Has practical implications such as giving faster implementations and guaranteeing determinism (same solution as sequential)