Fully Dynamic Maximal Independent Set with Sublinear Update Time

Sepehr Assadi

University of Pennsylvania

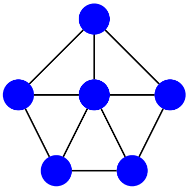
Joint work with:

Krzysztof Onak, Baruch Schieber, and Shay Solomon

IBM Research

The Maximal Independent Set Problem

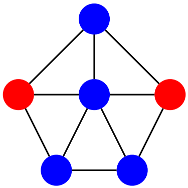
In a graph $G(V, E)$:



Maximal Independent Set (MIS): A maximal collection of vertices $\mathcal{M}$ such that no pair of vertices are adjacent.

The Maximal Independent Set Problem

In a graph $G(V, E)$:



Maximal Independent Set (MIS): A maximal collection of vertices $\mathcal{M}$ such that no pair of vertices are adjacent.

The Maximal Independent Set Problem

Finding an MIS is a fundamental problem with numerous applications.

Closely related to a plethora of other basic problems, such as vertex cover, matching, vertex coloring, and edge coloring.

Has been studied extensively in various settings, in particular parallel and distributed algorithms.

- Initiated by seminal works of [ABI86, Lub86, Lin87].

Maintaining an MIS in Dynamic Graphs

Maintaining an MIS in dynamically changing graphs on the other hand has not received much attention.

Maintaining an MIS in Dynamic Graphs

Maintaining an MIS in dynamically changing graphs on the other hand has not received much attention.

Censor-Hillel, Haramaty, and Karnin [CHK16]: randomized algorithm in distributed dynamic networks.

Maintaining an MIS in Dynamic Graphs

Maintaining an MIS in dynamically changing graphs on the other hand has not received much attention.

Censor-Hillel, Haramaty, and Karnin [CHK16]: randomized algorithm in distributed dynamic networks.

Maintaining an MIS in (sequential) dynamic graphs setting was left open by [CHK16].

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.
- Maintain a solution to a combinatorial problem, say an MIS, after every update.

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.
- Maintain a solution to a combinatorial problem, say an MIS, after every update.
- Can we do better than simply recomputing the solution from scratch after every update?

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.
- Maintain a solution to a combinatorial problem, say an MIS, after every update.
- Can we do better than simply recomputing the solution from scratch after every update?

Dynamic graphs appear in lots of applications.

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.
- Maintain a solution to a combinatorial problem, say an MIS, after every update.
- Can we do better than simply recomputing the solution from scratch after every update?

Dynamic graphs appear in lots of applications.

Numerous problems have been studied in dynamic graphs:

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.
- Maintain a solution to a combinatorial problem, say an MIS, after every update.
- Can we do better than simply recomputing the solution from scratch after every update?

Dynamic graphs appear in lots of applications.

Numerous problems have been studied in dynamic graphs:

Connectivity, spanning tree, maximal matching, approximate matching and vertex cover, shortest path, graph coloring, . . .

Dynamic Graphs Setting

- We have a graph $G(V, E)$ that undergoes edge insertions and deletions.
- Maintain a solution to a combinatorial problem, say an MIS, after every update.
- Can we do better than simply recomputing the solution from scratch after every update?

Dynamic graphs appear in lots of applications.

Numerous problems have been studied in dynamic graphs:

Connectivity, spanning tree, maximal matching, approximate matching and vertex cover, shortest path, graph coloring, ...

Dynamic MIS Problem

Not much is known for maintaining an MIS in dynamic graphs.

Dynamic MIS Problem

Not much is known for maintaining an MIS in dynamic graphs.

- $O(m)$ update time by recomputing an MIS after every update.

Dynamic MIS Problem

Not much is known for maintaining an MIS in dynamic graphs.

- $O(m)$ update time by recomputing an MIS after every update.
- Is there something better we could do?

Warm-Up: A Simple Algorithm

Δ: an upper bound on the maximum degree of the graph at all times.

Can we at least achieve $O(\Delta)$ update time?

Warm-Up: A Simple Algorithm

Δ: an upper bound on the maximum degree of the graph at all times.

Can we at least achieve $O(\Delta)$ update time?

For many problems of similar nature, say maximal matching or $(\Delta + 1)$-vertex coloring, this is a trivial task.

Warm-Up: A Simple Algorithm

Δ: an upper bound on the maximum degree of the graph at all times.

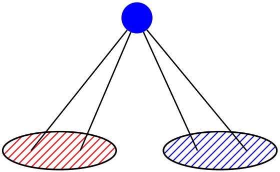
Can we at least achieve $O(\Delta)$ update time?

For many problems of similar nature, say maximal matching or $(\Delta + 1)$-vertex coloring, this is a trivial task.

Let us examine a natural strategy for maintaining an MIS $\mathcal{M}$ in $O(\Delta)$ update time.

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

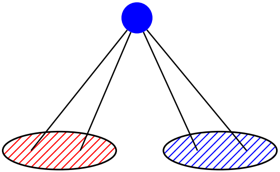
Invariant: Every vertex knows its neighbors in $\mathcal{M}$.



Warm-Up: An $O(\Delta)$ Update Time Algorithm?

Invariant: Every vertex knows its neighbors in $\mathcal{M}$.

So any vertex in $O(1)$ time can decide to join or leave $\mathcal{M}$.



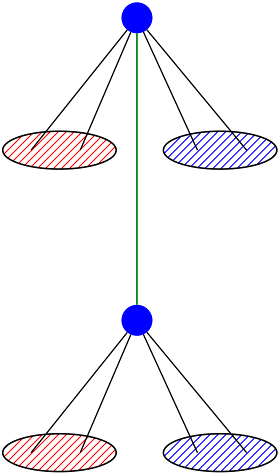
Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

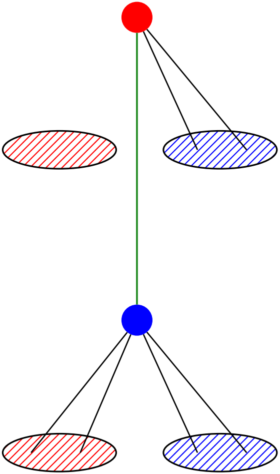
If $u, v \notin \mathcal{M}$ nothing to do.



Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

If $u \in \mathcal{M}$ and $v \notin \mathcal{M}$ (or vice versa):

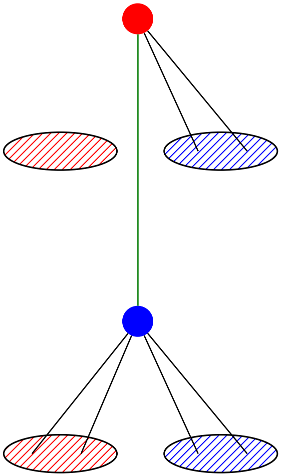


Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

If $u \in \mathcal{M}$ and $v \notin \mathcal{M}$ (or vice versa):

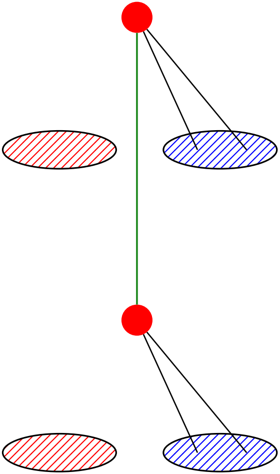
- Edge insertion: update list of $\mathcal{M}$-neighbors of v.
- Edge deletion: v checks if it can now join $\mathcal{M}$ or not. Inform all its neighbors in $O(\Delta)$ time if it joins.



Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

If $u, v \in \mathcal{M}$:

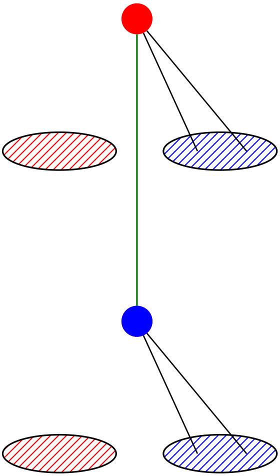


Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

If $u, v \in \mathcal{M}$:

- Edge insertion: One of them, say v, needs to leave $\mathcal{M}$. All neighbors of v can now potentially join $\mathcal{M}$.



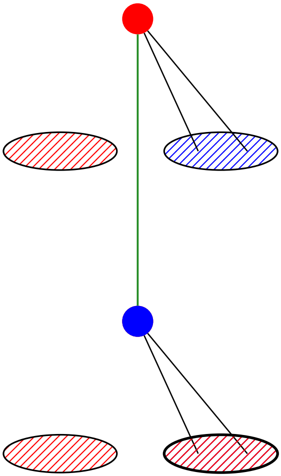
Warm-Up: An $O(\Delta)$ Update Time Algorithm?

How to maintain the invariant after an edge (u, v) is updated?

If $u, v \in \mathcal{M}$:

- Edge insertion: One of them, say v, needs to leave $\mathcal{M}$. All neighbors of v can now potentially join $\mathcal{M}$.

  This is problematic: $\Omega(\Delta)$ vertices potentially need to inform their $\Omega(\Delta)$ neighbors if they joined $\mathcal{M}$!



Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Are we doomed then?

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Are we doomed then? Not really!

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Are we doomed then? Not really!

The algorithm has a very simple yet useful property:

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Are we doomed then? Not really!

The algorithm has a very simple yet useful property:

After a single update:

- Many vertices can potentially join $\mathcal{M}$.
- Only one vertex can ever leave $\mathcal{M}$.

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Are we doomed then? Not really!

The algorithm has a very simple yet useful property:

After a single update:

- Many vertices can potentially join $\mathcal{M}$.
- Only one vertex can ever leave $\mathcal{M}$.

How to use this?

- Charge a vertex removed from $\mathcal{M}$ with $O(\Delta)$ budget/time.
- Use it for both removing it now as well as (potentially) inserting it later.

Warm-Up: An $O(\Delta)$ Update Time Algorithm?

$\Omega(\Delta^2)$ update time in the worst case.

Are we doomed then? Not really!

The algorithm has a very simple yet useful property:

After a single update:

- Many vertices can potentially join $\mathcal{M}$.
- Only one vertex can ever leave $\mathcal{M}$.

How to use this?

- Charge a vertex removed from $\mathcal{M}$ with $O(\Delta)$ budget/time.
- Use it for both removing it now as well as (potentially) inserting it later.

Amortized update time is only $O(\Delta)$!

Motivating Question

We saw a very simple $O(\Delta)$ amortized update time algorithm for dynamic MIS.

Motivating Question

We saw a very simple $O(\Delta)$ amortized update time algorithm for dynamic MIS.

Still does not improve upon naive re-computation algorithm for sparse graphs with $\Delta = \Theta(m)$.

Motivating Question

We saw a very simple $O(\Delta)$ amortized update time algorithm for dynamic MIS.

Still does not improve upon naive re-computation algorithm for sparse graphs with $\Delta = \Theta(m)$.

Motivating Question. Can we maintain an MIS in a dynamic graph in sublinear update time, i.e., $o(m)$ time?

Motivating Question

We saw a very simple $O(\Delta)$ amortized update time algorithm for dynamic MIS.

Still does not improve upon naive re-computation algorithm for sparse graphs with $\Delta = \Theta(m)$.

Motivating Question. Can we maintain an MIS in a dynamic graph in sublinear update time, i.e., $o(m)$ time? Yes!

Our Main Result

We prove that

A maximal independent set can be maintained deterministically in $O(m^{3/4})$ amortized update time, where m denotes the dynamic number of edges.

Our Main Result

We prove that

> A maximal independent set can be maintained deterministically in $O(m^{3/4})$ amortized update time, where m denotes the dynamic number of edges.

First improvement over the $O(m)$ update time for all values of m.

Our Main Result

We prove that

> A maximal independent set can be maintained deterministically in $O(m^{3/4})$ amortized update time, where m denotes the dynamic number of edges.

First improvement over the $O(m)$ update time for all values of m.

Can also be implemented in distributed dynamic networks, strengthening the result of [CHK16].

Deterministic vs Randomized Dynamic Algorithms

- A significant distinction between deterministic and randomized algorithms in the dynamic setting.

Deterministic vs Randomized Dynamic Algorithms

- A significant distinction between deterministic and randomized algorithms in the dynamic setting.
- Main reason: assumption of a non-adaptive oblivious adversary by randomized algorithms.

Deterministic vs Randomized Dynamic Algorithms

- A significant distinction between deterministic and randomized algorithms in the dynamic setting.
- Main reason: assumption of a non-adaptive oblivious adversary by randomized algorithms.
  - Adversary has to fix the sequence of updates beforehand and cannot adapt to decisions of the dynamic algorithm.

Deterministic vs Randomized Dynamic Algorithms

- A significant distinction between deterministic and randomized algorithms in the dynamic setting.
- Main reason: assumption of a non-adaptive oblivious adversary by randomized algorithms.
  - Adversary has to fix the sequence of updates beforehand and cannot adapt to decisions of the dynamic algorithm.
  - This can render randomized algorithms entirely unusable in certain scenarios.

Deterministic vs Randomized Dynamic Algorithms

- A significant distinction between deterministic and randomized algorithms in the dynamic setting.
- Main reason: assumption of a non-adaptive oblivious adversary by randomized algorithms.
  - Adversary has to fix the sequence of updates beforehand and cannot adapt to decisions of the dynamic algorithm.
  - This can render randomized algorithms entirely unusable in certain scenarios.
- A huge gap between the update time of best deterministic vs randomized algorithms for dynamic problems:

Deterministic vs Randomized Dynamic Algorithms

- A significant distinction between deterministic and randomized algorithms in the dynamic setting.
- Main reason: assumption of a non-adaptive oblivious adversary by randomized algorithms.
  - Adversary has to fix the sequence of updates beforehand and cannot adapt to decisions of the dynamic algorithm.
  - This can render randomized algorithms entirely unusable in certain scenarios.
- A huge gap between the update time of best deterministic vs randomized algorithms for dynamic problems:
  - Maximal Matching: $O(\sqrt{m})$ for deterministic [NS13] vs $O(1)$ for randomized [Sol16].
  - $(\Delta + 1)$-Vertex Coloring: No non-trivial deterministic algorithm vs $O(\log \Delta)$ for randomized [BCHN18].

An $O(m^{3/4})$ Amortized Update Time Dynamic Algorithm for MIS

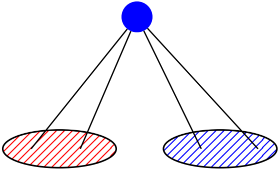
High Level Idea

Part I: Maintaining a local knowledge of the graph for each vertex.

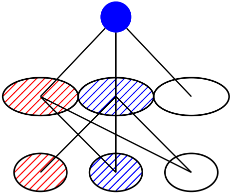
High Level Idea

Part I: Maintaining a local knowledge of the graph for each vertex.

$O(\Delta)$ time algorithm:



$O(m^{3/4})$ time algorithm:

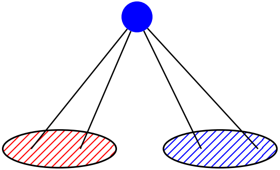


High Level Idea

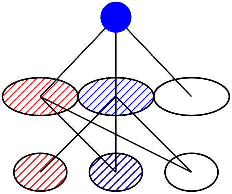
Part I: Maintaining a local knowledge of the graph for each vertex.

- Each vertex knows some information about its neighbors that are in $\mathcal{M}$.

$O(\Delta)$ time algorithm:



$O(m^{3/4})$ time algorithm:

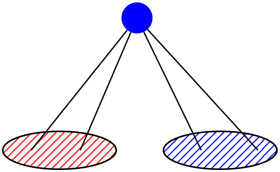


High Level Idea

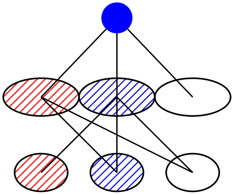
Part I: Maintaining a local knowledge of the graph for each vertex.

- Each vertex knows some information about its neighbors that are in $\mathcal{M}$.
- This information maybe inconsistent across vertices as we cannot afford to update Δ neighbors of each vertex per update.

$O(\Delta)$ time algorithm:



$O(m^{3/4})$ time algorithm:

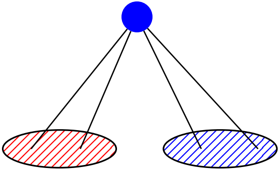


High Level Idea

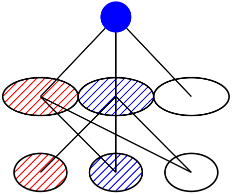
Part I: Maintaining a local knowledge of the graph for each vertex.

- Each vertex knows some information about its neighbors that are in $\mathcal{M}$.
- This information maybe inconsistent across vertices as we cannot afford to update Δ neighbors of each vertex per update.
- To compensate, we also maintain some information about 2-hop neighbors of vertices.

$O(\Delta)$ time algorithm:



$O(m^{3/4})$ time algorithm:



Our Algorithm: High Level Plan

Part II: Maintaining the MIS using the local and inconsistent information of vertices.

Our Algorithm: High Level Plan

Part II: Maintaining the MIS using the local and inconsistent information of vertices.

Main challenge:

- Based on their partial information, some vertices may join $\mathcal{M}$.
- They may however have some neighbors already in $\mathcal{M}$.
- As such we may need to delete some vertices in the current $\mathcal{M}$ and process them recursively again.

Maintaining the Partial Information

Vertices are partitioned into four sets based on their degrees:

V_{High}: $deg \geq m^{3/4}$

$V_{\text{Med-High}}$: $m^{3/4} > deg \geq m^{1/2}$

$V_{\text{Med-Low}}$: $m^{1/2} > deg \geq m^{1/4}$

V_{Low}: remaining vertices

Maintaining the Partial Information

Vertices are partitioned into four sets based on their degrees:

V_{High}: $deg \geq m^{3/4}$

$V_{\text{Med-High}}$: $m^{3/4} > deg \geq m^{1/2}$

$V_{\text{Med-Low}}$: $m^{1/2} > deg \geq m^{1/4}$

V_{Low}: remaining vertices

Why this helps?

Maintaining the Partial Information

Vertices are partitioned into four sets based on their degrees:

V_{High}: $deg \geq m^{3/4}$

$V_{\text{Med-High}}$: $m^{3/4} > deg \geq m^{1/2}$

$V_{\text{Med-Low}}$: $m^{1/2} > deg \geq m^{1/4}$

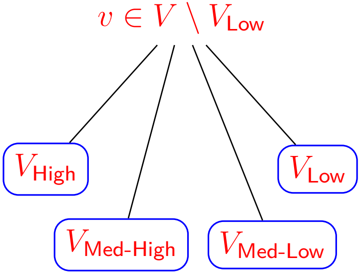
V_{Low}: remaining vertices

Why this helps?

Maintaining the Partial Information

$\mathcal{M}$-neighbor information:

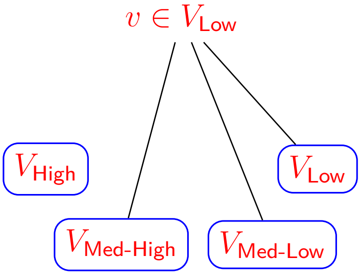
- All vertices except for V_{Low} know all their $\mathcal{M}$-neighbors.



Maintaining the Partial Information

$\mathcal{M}$-neighbor information:

- All vertices except for V_{Low} know all their $\mathcal{M}$-neighbors.
- Vertices in V_{Low} know all their neighbors in $\mathcal{M}$ except for the ones in V_{High}.



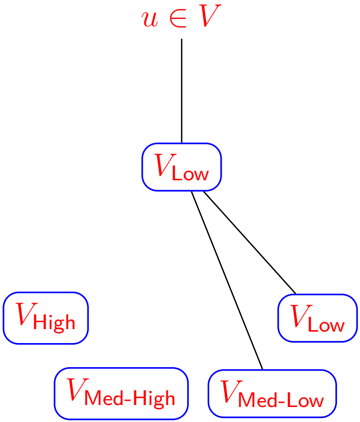
Maintaining the Partial Information

$\mathcal{M}$-neighbor information:

- All vertices except for V_{Low} know all their $\mathcal{M}$-neighbors.
- Vertices in V_{Low} know all their neighbors in $\mathcal{M}$ except for the ones in V_{High}.

$\mathcal{M}$-2-hop-neighbor information:

- Any vertex u for any one of its neighbors v in V_{Low} knows all neighbors of v that are in $V_{\text{Low}} \cup V_{\text{Med-Low}}$ and are in $\mathcal{M}$.



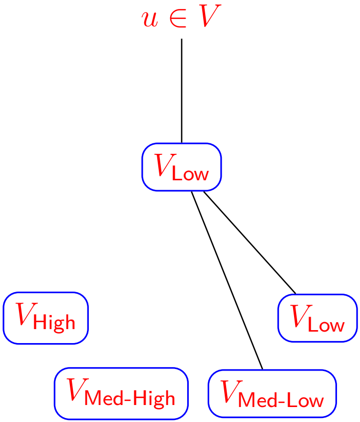
Maintaining the Partial Information

$\mathcal{M}$-neighbor information:

- All vertices except for V_{Low} know all their $\mathcal{M}$-neighbors.
- Vertices in V_{Low} know all their neighbors in $\mathcal{M}$ except for the ones in V_{High}.

$\mathcal{M}$-2-hop-neighbor information:

- Any vertex u for any one of its neighbors v in V_{Low} knows all neighbors of v that are in $V_{\text{Low}} \cup V_{\text{Med-Low}}$ and are in $\mathcal{M}$.



Lemma. One can maintain this information in $O(m^{3/4})$ time per each update to the graph or $\mathcal{M}$.

The Update Algorithm

Unlike the $O(\Delta)$-time algorithm, multiple vertices may be deleted from $\mathcal{M}$.

The Update Algorithm

Unlike the $O(\Delta)$-time algorithm, multiple vertices may be deleted from $\mathcal{M}$.

Invariant (Main Invariant)

After every update:

- *If only a single vertex leaves $\mathcal{M}$, then there is no restriction on the number of vertices joining $\mathcal{M}$ (which could be zero).*

The Update Algorithm

Unlike the $O(\Delta)$-time algorithm, multiple vertices may be deleted from $\mathcal{M}$.

Invariant (Main Invariant)

After every update:

- *If only a single vertex leaves $\mathcal{M}$, then there is no restriction on the number of vertices joining $\mathcal{M}$ (which could be zero).*
- *However, if $k > 1$ vertices leave $\mathcal{M}$, then at least $2k$ vertices would join $\mathcal{M}$.*

The Update Algorithm

Unlike the $O(\Delta)$-time algorithm, multiple vertices may be deleted from $\mathcal{M}$.

Invariant (Main Invariant)

After every update:

- *If only a single vertex leaves $\mathcal{M}$, then there is no restriction on the number of vertices joining $\mathcal{M}$ (which could be zero).*
- *However, if $k > 1$ vertices leave $\mathcal{M}$, then at least $2k$ vertices would join $\mathcal{M}$.*

The time spent per each update to $\mathcal{M}$ is $O(m^{3/4})$.

The Update Algorithm

Unlike the $O(\Delta)$-time algorithm, multiple vertices may be deleted from $\mathcal{M}$.

Invariant (Main Invariant)

After every update:

- *If only a single vertex leaves $\mathcal{M}$, then there is no restriction on the number of vertices joining $\mathcal{M}$ (which could be zero).*
- *However, if $k > 1$ vertices leave $\mathcal{M}$, then at least $2k$ vertices would join $\mathcal{M}$.*

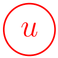
The time spent per each update to $\mathcal{M}$ is $O(m^{3/4})$.

This is enough to obtain the $O(m^{3/4})$ amortized update time.

Processing Updates

The main challenging update: adversary inserts an edge (u, v) between $u, v \in \mathcal{M}$.

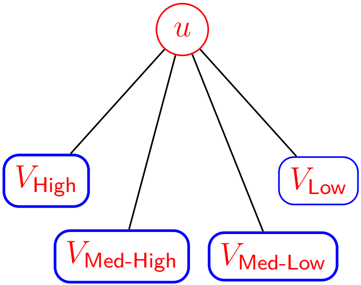
- We delete u from $\mathcal{M}$.



Processing Updates

The main challenging update: adversary inserts an edge (u, v) between $u, v \in \mathcal{M}$.

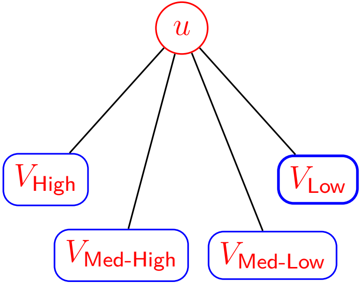
- We delete u from $\mathcal{M}$.
- Neighbors of u not in V_{Low}:
  - Each one knows if it can join $\mathcal{M}$.
  - Any one that joins $\mathcal{M}$ needs $O(m^{3/4})$ time to update partial information of its neighborhood.



Processing Updates

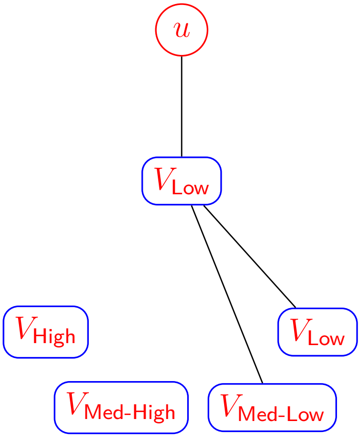
The main challenging update: adversary inserts an edge (u, v) between $u, v \in \mathcal{M}$.

- We delete u from $\mathcal{M}$.
- Neighbors of u not in V_{Low}:
  - Each one knows if it can join $\mathcal{M}$.
  - Any one that joins $\mathcal{M}$ needs $O(m^{3/4})$ time to update partial information of its neighborhood.
- Neighbors of u inside V_{Low}:
  - Their local information is not enough to determine whether they should join $\mathcal{M}$ or not.



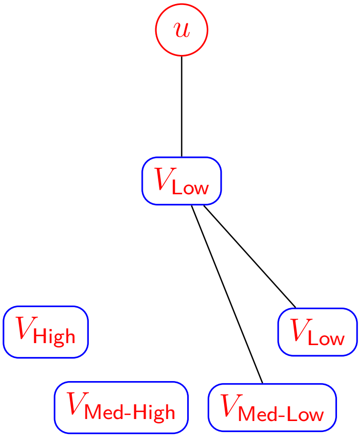
Handling Low-Degree Neighbors of u

- u knows all its neighbors that do not have a neighbor in $\mathcal{M} \cap (V_{\text{Low}} \cup V_{\text{Med-Low}})$.



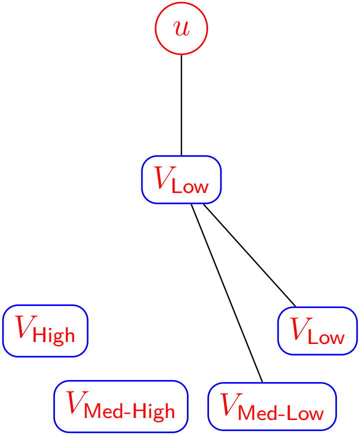
Handling Low-Degree Neighbors of u

- u knows all its neighbors that do not have a neighbor in $\mathcal{M} \cap (V_{\text{Low}} \cup V_{\text{Med-Low}})$.
- Let A be the set of such neighbors.



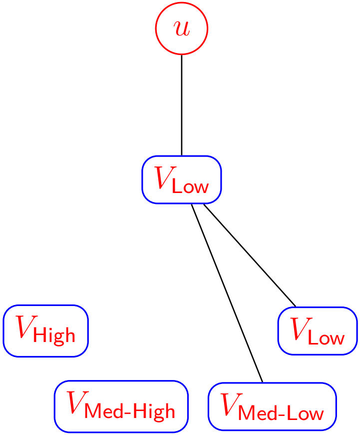
Handling Low-Degree Neighbors of u

- u knows all its neighbors that do not have a neighbor in $\mathcal{M} \cap (V_{\text{Low}} \cup V_{\text{Med-Low}})$.
- Let A be the set of such neighbors.
- We process the update based on whether A is large or not.



Handling Low-Degree Neighbors of u

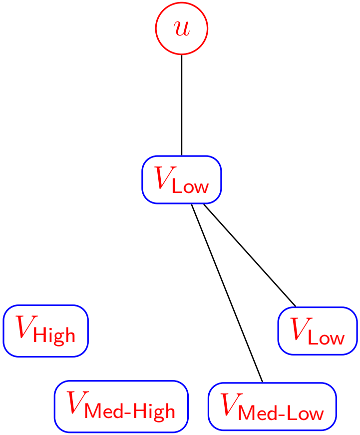
Let us assume A is very large i.e., has $\omega(m^{3/4})$ vertices:



Handling Low-Degree Neighbors of u

Let us assume A is very large i.e., has $\omega(m^{3/4})$ vertices:

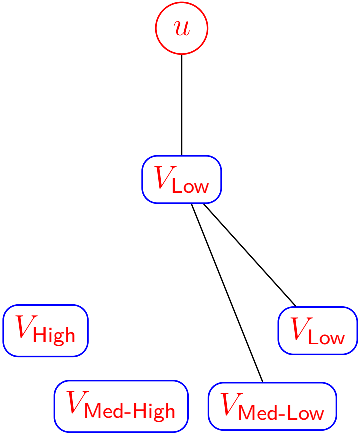
1. Greedily insert these vertices to $\mathcal{M}$, only check for consistency between the inserted vertices.



Handling Low-Degree Neighbors of u

Let us assume A is very large i.e., has $\omega(m^{3/4})$ vertices:

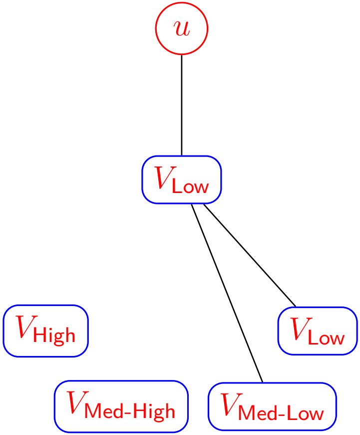
1. Greedily insert these vertices to $\mathcal{M}$, only check for consistency between the inserted vertices.
   - How many inserted? $\omega(m^{1/2})$.



Handling Low-Degree Neighbors of u

Let us assume A is very large i.e., has $\omega(m^{3/4})$ vertices:

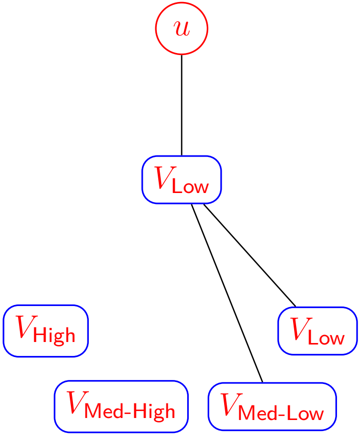
1. Greedily insert these vertices to $\mathcal{M}$, only check for consistency between the inserted vertices.
   - How many inserted? $\omega(m^{1/2})$.
2. $\mathcal{M}$ may become infeasible as there would be some neighboring vertices inside it.



Handling Low-Degree Neighbors of u

Let us assume A is very large i.e., has $\omega(m^{3/4})$ vertices:

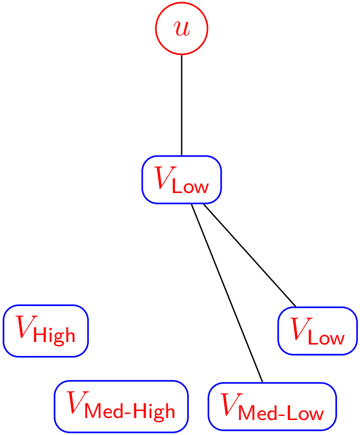
1. Greedily insert these vertices to $\mathcal{M}$, only check for consistency between the inserted vertices.
   - How many inserted? $\omega(m^{1/2})$.
2. $\mathcal{M}$ may become infeasible as there would be some neighboring vertices inside it.
3. Simply remove these vertices from $\mathcal{M}$. Recursively process these vertices.



Handling Low-Degree Neighbors of u

Let us assume A is very large i.e., has $\omega(m^{3/4})$ vertices:

1. Greedily insert these vertices to $\mathcal{M}$, only check for consistency between the inserted vertices.
   - How many inserted? $\omega(m^{1/2})$.
2. $\mathcal{M}$ may become infeasible as there would be some neighboring vertices inside it.
3. Simply remove these vertices from $\mathcal{M}$. Recursively process these vertices.
   - How many deleted? $O(m^{1/2})$.



Wrap-Up

- We spend $O(m^{3/4})$ time per update for maintaining the local information.
- We spend $O(m^{3/4})$ time for any vertex inserted to or deleted from $\mathcal{M}$.
- By main invariant, we can charge each vertex deleted from $\mathcal{M}$ with $O(m^{3/4})$ time, to be used when this vertex is inserted back later (if ever).

Wrap-Up

- We spend $O(m^{3/4})$ time per update for maintaining the local information.
- We spend $O(m^{3/4})$ time for any vertex inserted to or deleted from $\mathcal{M}$.
- By main invariant, we can charge each vertex deleted from $\mathcal{M}$ with $O(m^{3/4})$ time, to be used when this vertex is inserted back later (if ever).

MIS can be maintained deterministically with $\min\left\{O(m^{3/4}), O(\Delta)\right\}$ amortized update time in dynamic graphs.

Recent Developments

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.
- Du and Zhang [DZ18]: arXiv, April 2018

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.
- Du and Zhang [DZ18]: arXiv, April 2018
  - Deterministic $\widetilde{O}(m^{2/3})$ amortized update time.

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.
- Du and Zhang [DZ18]: arXiv, April 2018
  - Deterministic $\widetilde{O}(m^{2/3})$ amortized update time.
  - Randomized $\widetilde{O}(\sqrt{m})$ expected amortized update time.

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.
- Du and Zhang [DZ18]: arXiv, April 2018
  - Deterministic $\widetilde{O}(m^{2/3})$ amortized update time.
  - Randomized $\widetilde{O}(\sqrt{m})$ expected amortized update time.
- Assadi, Onak, Schieber, and Solomon [AOSS18b]: arXiv, June 2018

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.
- Du and Zhang [DZ18]: arXiv, April 2018
  - Deterministic $\widetilde{O}(m^{2/3})$ amortized update time.
  - Randomized $\widetilde{O}(\sqrt{m})$ expected amortized update time.
- Assadi, Onak, Schieber, and Solomon [AOSS18b]: arXiv, June 2018
  - Randomized $\widetilde{O}(\sqrt{n})$ expected amortized update time.

Recent Developments

- Onak, Schieber, Solomon, and Wein [OSSW18]: ICALP 2018
  - Deterministic $O(\log^2 n)$ amortized update time for bounded arboricity graphs.
- Gupta and Khan [GK18]: arXiv, April 2018
  - Deterministic $O(m^{2/3})$ amortized update time.
  - Further results for incremental and decremental settings.
- Du and Zhang [DZ18]: arXiv, April 2018
  - Deterministic $\widetilde{O}(m^{2/3})$ amortized update time.
  - Randomized $\widetilde{O}(\sqrt{m})$ expected amortized update time.
- Assadi, Onak, Schieber, and Solomon [AOSS18b]: arXiv, June 2018
  - Randomized $\widetilde{O}(\sqrt{n})$ expected amortized update time.
  - Randomized $\widetilde{O}(m^{1/3})$ expected amortized update time.

Conclusion

- We gave a deterministic $O(m^{3/4})$ amortized update time algorithm for maintaining an MIS in dynamic graphs.
- Our algorithm can also be implemented in dynamic distributed networks.

Conclusion

- We gave a deterministic $O(m^{3/4})$ amortized update time algorithm for maintaining an MIS in dynamic graphs.
- Our algorithm can also be implemented in dynamic distributed networks.

Open questions:

- Faster dynamic algorithms for MIS?
  - Best deterministic algorithm: $O(m^{2/3})$ time [GK18].
  - Best randomized algorithm: $\min\left\{\widetilde{O}(\sqrt{n}), \widetilde{O}(m^{1/3})\right\}$ time [AOSS18b].

Conclusion

- We gave a deterministic $O(m^{3/4})$ amortized update time algorithm for maintaining an MIS in dynamic graphs.
- Our algorithm can also be implemented in dynamic distributed networks.

Open questions:

- Faster dynamic algorithms for MIS?
  - Best deterministic algorithm: $O(m^{2/3})$ time [GK18].
  - Best randomized algorithm: $\min\left\{\widetilde{O}(\sqrt{n}), \widetilde{O}(m^{1/3})\right\}$ time [AOSS18b].
- Better deterministic dynamic algorithms for other "maximal-type" problems?
  - Example: $o(\sqrt{m})$ time algorithm for maximal matching?

Conclusion

- We gave a deterministic $O(m^{3/4})$ amortized update time algorithm for maintaining an MIS in dynamic graphs.
- Our algorithm can also be implemented in dynamic distributed networks.

Open questions:

- Faster dynamic algorithms for MIS?
  - Best deterministic algorithm: $O(m^{2/3})$ time [GK18].
  - Best randomized algorithm: $\min\left\{\widetilde{O}(\sqrt{n}), \widetilde{O}(m^{1/3})\right\}$ time [AOSS18b].
- Better deterministic dynamic algorithms for other "maximal-type" problems?
  - Example: $o(\sqrt{m})$ time algorithm for maximal matching?

Thank you!

Noga Alon, László Babai, and Alon Itai.
A fast and simple randomized parallel algorithm for the maximal independent set problem.
J. Algorithms, 7(4):567–583, 1986.

Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai.
Dynamic algorithms for graph coloring.
In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20, 2018.

Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin.
Optimal dynamic distributed MIS.
In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016.

Yuhao Du and Hengjie Zhang.

Improved algorithms for fully dynamic maximal independent set.
CoRR, abs/1804.08908, 2018.

Manoj Gupta and Shahbaz Khan.
Simple dynamic algorithms for maximal independent set and other problems.
CoRR, abs/1804.01823, 2018.

Nathan Linial.
Distributive graph algorithms-global solutions from local data.
In *Proceedings of the 28th IEEE Annual Symposium on Foundations of Computer Science, FOCS 1987, Los Angeles, CA, USA, October 27-29, 1987*, pages 331–335, 1987.

Michael Luby.
A simple parallel algorithm for the maximal independent set problem.
SIAM J. Comput., 15(4):1036–1053, 1986.

Ofer Neiman and Shay Solomon.

Simple deterministic algorithms for fully dynamic maximal matching.
In *Proceedings of the 45th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.

Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein.
Fully dynamic mis in uniformly sparse graphs.
In *To appear in ICALP'18*, 2018.

S. Solomon.
Fully dynamic maximal matching in constant update time.
In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2016, New Brunswick, NJ, USA, October 9-11, 2016*, pages 325–334, 2016.