Fully Dynamic Maximal Matching in $O(\log n)$ update time

Manoj Gupta, IIT Delhi
Joint work with Sandeep Sen, IIT Delhi and Surender Baswana, IIT Kanpur

2nd Annual Mysore Park Workshop in Theoretical Computer Science
Outline

1. **Introduction**
   - The Problem
   - A Simple Algorithm

2. **$\sqrt{n}$ algorithm**
   - The overview of the approach
   - Overview of the analysis
   - Algorithm

3. **From $\sqrt{n}$ to $\log n$**
   - Speeding up the algorithm

4. **Open Problem**
A matching in a graph is a set of edges $M$ such that no two edges in $M$ share a common endpoint.
A matching in a graph is a set of edges $M$ such that no two edges in $M$ share a common endpoint.

Matched vertex
A matching in a graph is a set of edges \( M \) such that no two edges in \( M \) share a common endpoint.

- Matched vertex
- Free vertex
A matching is maximal if for each vertex $v$:
- $v$ is matched or
- $v$ does not have a free neighbor.
A matching is maximal if for each vertex $v$:
- $v$ is matched or
- $v$ does not have a free neighbor.

**Problem**
Maintain maximal matching in a dynamic graph
A matching is maximal if for each vertex $v$:
- $v$ is matched or
- $v$ does not have a free neighbor.

**Problem**
Maintain maximal matching in a dynamic graph

**Expectation from the algorithm**
Update time should be $\text{polylog}(n)$
Previous Work

- Ivkovic and Lloyd (1994) - $O((n + m)^{0.7072})$
- Onak and Rubinfeld (2010) gave a $c$-approximation of maximum matching in $O(\log^2 n)$ update time.
A Naive Approach

- **Insertion of an edge**
  
  - Insertion = $O(1)$
  - Deletion = $O(n)$

- **Deletion of an edge**
  
  - Do Nothing
A Naive Approach

Insertion of an edge

\[ f \rightarrow m \rightarrow \text{Do Nothing} \]

Deletion of an edge

\[ m \rightarrow m \]
A Simple Algorithm

A Naive Approach

**Insertion of an edge**

- Insertion = $O(1)$
- Deletion = $O(n)$

**Deletion of an edge**

- Search neighborhood of both vertices for free vertex
A Simple Algorithm

A Naive Approach

### Insertion of an edge
- Insertion = $O(1)$

### Deletion of an edge
- Deletion = $O(n)$
The difficulty

- Deletion of a matched edge
- Handling high degree vertex
A Simple Algorithm

The difficulty

- Deletion of a matched edge
- Handling high degree vertex

Possible ways to solve

- Make sure that high degree vertex are always matched
- Make sure that a matched edge is deleted rarely
Partition the vertices into two buckets (level 1 and 0) such that most of the vertices have high "degree" when they come to level 1.

The partition is dynamic and the vertices may move from level 1 and level 0.

Maintain the following invariant:
- The vertex at level 1 are always matched.
- The vertex at level 0 has degree $< \sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.
The overview of the approach

- Partition the vertices into two buckets (level 1 and 0) such that most of the vertices have high "degree" when they come to level 1.
- The partition is dynamic and the vertices may move from level 1 and level 0.
- Maintain the following invariant:
  - The vertex at level 1 are always matched.
  - The vertex at level 0 has degree $< \sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

![Diagram showing partition of vertices into level 1 and level 0 with matching edges indicated.](image-url)
The overview of the approach

**Notion of ownership**

Each edge present in the graph will be owned by one or both of its end points as follows:

- If both the end points are at level 0, then it is owned by both the endpoints.
- If only one endpoint is at level 1, then it owns the edge.
- If both the end points are at the same level, we can break the tie arbitrarily.
Notion of ownership

Each edge present in the graph will be owned by one or both of its end points as follows:

- If both the end points are at level 0, then it is owned by both the endpoints.
- If only one endpoint is at level 1, then it owns the edge.
- If both the end points are at the same level, we can break the tie arbitrarily.
Notion of ownership

Each edge present in the graph will be owned by one or both of its end points as follows:

- If both the end points are at level 0, then it is owned by both the endpoints.
- If only one endpoint is at level 1, then it owns the edge.
- If both the end points are at the same level, we can break the tie arbitrarily.
Notion of ownership

Each edge present in the graph will be owned by one or both of its end points as follows:

- If both the end points are at level 0, then it is owned by both the endpoints.
- If only one endpoint is at level 1, then it owns the edge.
- If both the end points are at the same level, we can break the tie arbitrarily.
**Notion of matched epoch**

Epoch of \((u, v)\) is the maximal continuous time period for which it remains in the matching.

| \(t\)     | \(t + 1\) | \(t + 2\) | \(t + 3\) | \(t' - 1\) | \(t'\)            |
|-----------|-----------|-----------|-----------|-----------|-------------------|
| \((u, v)\) matched |          | Add\((u, w)\) | Del\((v, z)\) |          | \((u, v)\) removed from matching |
Algorithm

Epoch of Level 0

- Inv2: The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

Start of the epoch

But... what if $u$ has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?
Inv2: The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

**Start of the epoch**

But...what if $u$ has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

**End of the epoch**

$m$ to $m$
**Epoch of Level 0**

- **Inv2:** The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

**Start of the epoch**

But...what if $u$ has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

**End of the epoch**

Search only the edges whose other endpoint are at level 0.
Epoch of Level 0

Inv2: The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

**Start of the epoch**

- But... what if $u$ has more than $\sqrt{n}$ edges in $G[V_0]$ after edge insertion?

**End of the epoch**

- Start = $O(1)$
- End = $O(\sqrt{n})$

Search only the edges whose other endpoint are at level 0.
**Algorithm**

**Epoch at level 1: Start**

- **Inv2:** The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

---

Invariant 2 does not hold for vertex $u$.

---

![Graph](attachment:graph.png)
Algorithm

**Epoch at level 1: Start**

- Inv2: The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

- $\rightarrow$ Vertex $u$ moves to level 1
- $\rightarrow$ Make $u$ the *owner* of all its edges at level 0
- $\rightarrow$ Find a random edge from the owned edges, say $(u, v)$
Epoch at level 1: Start

Inv2: The vertex at level 0 has degree $\sqrt{n}$ in $G[V_0]$ and each free vertex at this level has all its neighbors matched.

$\rightarrow$ Add $(u, v)$ to $M$

$\rightarrow$ Move $v$ to level 1

$\rightarrow$ Make $v$ the owner of all its adjacent edges

Start of an epoch at level 1 = $O(\sqrt{n})$
From $\sqrt{n}$ to log $n$

Algorithm

Epoch at level 1: End

Edge $(u, v)$ is deleted
→ Give up the ownership of the edges at level 1

→ If $u$ is still the owner of $\geq \sqrt{n}$ edges

the the procedure is same as in the previous slide
Algorithm

Epoch at level 1: End

→ Else $u$ moves to level 1 and starts level 0 epoch there

→ But the degree of vertex $a$ in $G[V_0]$ increases by 1 and may move to level 1

→ All such vertices move up and start a epoch at level 1

End of an epoch at level 1 = $O(n)$
### Algorithm

| Epochs | Start | End   | Total cost | Total number of Epochs | Total computation cost |
|--------|-------|-------|------------|------------------------|------------------------|
| Level 0| $O(1)$| $O(\sqrt{n})$ | $O(\sqrt{n})$ | $T$                    | $O(T \sqrt{n})$        |
| Level 1| $O(\sqrt{n})$ | $O(n)$ | $O(n)$     |                        |                        |
### Algorithm

| Epochs | Start | End   | Total cost | Total number of Epochs | Total computation cost |
|--------|-------|-------|------------|------------------------|------------------------|
| Level 0| $O(1)$| $O(\sqrt{n})$ | $O(\sqrt{n})$ | $T$                     | $O(T \sqrt{n})$        |
| Level 1| $O(\sqrt{n})$ | $O(n)$ | $O(n)$     | $O(T/\sqrt{n})$        | $O(T \sqrt{n})$        |

The algorithm has $O(\sqrt{n})$ update time.
### Algorithm

|                    | Start  | End    | Total cost | Total number ofEpochs | Total computation cost |
|--------------------|--------|--------|------------|------------------------|------------------------|
| **Level 0**        | \(O(1)\) | \(O(\sqrt{n})\) | \(O(\sqrt{n})\) | \(T\)                  | \(O(T \sqrt{n})\)     |
| **Level 1**        | \(O(\sqrt{n})\) | \(O(n)\) | \(O(n)\) | \(O(T/\sqrt{n})\)     | \(O(T \sqrt{n})\)     |

The algorithm has \(O(\sqrt{n})\) update time.
In the two level algorithm, we define a threshold $\alpha(n)$ for a vertex to move from level 0 to level 1

- The update time at level 0 is $O(\alpha(n))$
- The update time at level 1 is $O(n/\alpha(n))$
- Both the update time are same when $\alpha(n) = \sqrt{n}$

Speeding up the algorithm

- Try to minimize the gap between the number of edges a vertex can own in an epoch and the number of edges it owned at the moment it created the epoch
- This ratio is $\sqrt{n}$ in 2-level algorithm
An overview of the $\log n$-level algorithm

- Maintain $\log n$ levels
- When a vertex creates an epoch at level $i$, it would own at least $2^i$ edges, and during the epoch it will be allowed to own at most $2^{i+1}$ edges
- The ratio is a constant
- In implementing these ideas, an extra factor of $O(\log n)$ comes up due to the $\log n$ level hierarchy
Speeding up the algorithm

Example

The number of edges $v$ can own if it rises to level 2 = 2

The number of edges $v$ can own if it rises to level 3 = 4

- $v$ -1 0 1 2 3
- 0 1 2 3
Example

The number of edges $v$ can own if it rises to level 2 = $2 < 2^2$
**Example**

The number of edges \( v \) can own if it rises to level 3 = 4 < 2^3
Open Problem

- There exists an algorithm for maximal matching in $O(\log n)$ update time but is there a algorithm which maintains $c$ – approximation of maximum matching where $c < 2$

- Is there any combinatorial algorithm which maintains maximum matching in $o(m)$ time
Questions?