Using the Primal-Dual Technique in Dynamic Graph Algorithms

Monika Henzinger
University of Vienna

joint work with Sayan Bhattacharya and Giuseppe Italiano
SODA‘15 and ICALP‘16
Static Algorithm

Optimization Problem

Considerations: (1) Running Time
(2) Approximation Ratio
What is a Dynamic Algorithm?

Optimization Problem

Input $I + \Delta I$ \hspace{1cm} Algorithm \hspace{1cm} Output $O + \Delta O$

Considerations: (1) Running Time

(2) Approximation Ratio

Do we need to recompute the output from scratch?
Fully Dynamic *Maximum Matching* Algorithm

Maximum matching $M$

**Input**: $I + \Delta I$

**Algorithm**

**Output**: $O + \Delta O$

**Operations:**
- Initialize()
- Insert-Edge($u, v$): Insert $(u,v)$
- Delete-Edge($u,v$): Delete $(u,v)$
Matching Version

Maximum matching $M$

Input $I + \Delta I$ → Algorithm → Output $O + \Delta O$

Operations:
- Initialize(): Return $M$
- Insert-Edge($u$, $v$): Insert ($u,v$) and output the change in $M$
- Delete-Edge($u,v$): Delete ($u,v$) and output the change in $M$
Simpler: \textit{Value} Version

Maximum matching $M$

\begin{itemize}
  \item Initialize()
  \item Insert-Edge\((u, v)\): Insert \((u,v)\)
  \item Delete-Edge\((u,v)\): Delete \((u,v)\)
  \item Query\(): Return the size of the maximum matching
\end{itemize}
**Fully dynamic maximum matching**

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Update/ query time</th>
<th>Version</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n^\omega)$, $\omega &lt; 2.376$</td>
<td>$O(n^{1.495})/O(1)$</td>
<td>value</td>
<td>Sankowski ‘07</td>
</tr>
<tr>
<td>$\text{poly}(n)$</td>
<td>Not possible: $o(m^{2-\varepsilon})$ and $o(m^{1-\varepsilon})$</td>
<td>value</td>
<td>Abboud, Vassilesvka Williams ‘14, H, Krinninger, Nanongkai, Saranurak ‘15</td>
</tr>
<tr>
<td>Conditional on Triangle Detection, 3SUM, BMM, OMy</td>
<td></td>
<td></td>
<td>Kopelowitz, Pettie, Porat ‘15</td>
</tr>
</tbody>
</table>

Can an *approximate* matching be maintained in polylog update time?

$O(m)/O(1)$ matching

naive
Fully Dynamic $k$-Approximate Matching Algorithm

Operations:
- Initialize()
- Insert-Edge($u, v$)
- Delete-Edge($u, v$)
- Query(): Return a value $\alpha$ s. t. $\alpha \geq \frac{\text{size of max matching}}{k}$
## Approximate matching

<table>
<thead>
<tr>
<th>Approx. ratio</th>
<th>Update time</th>
<th>Amortized?</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(1))</td>
<td>(O(\log^2 n))</td>
<td>yes</td>
<td>Onak, Rubinfeld ‘10</td>
</tr>
<tr>
<td>2</td>
<td>(O(\log n))</td>
<td>yes</td>
<td>Baswana, Gupta, Sen ‘11</td>
</tr>
<tr>
<td>1.5</td>
<td>(O(\sqrt{m}))</td>
<td>no</td>
<td>Neiman, Solomon ‘13</td>
</tr>
<tr>
<td>1 + (\epsilon)</td>
<td>(O(\sqrt{m}/\epsilon))</td>
<td>no</td>
<td>Gupta, Peng ‘13</td>
</tr>
<tr>
<td>3 + (\epsilon)</td>
<td>(O(\log n /\epsilon^2))</td>
<td>yes</td>
<td>Bhattacharya, H, Italiano ‘15</td>
</tr>
<tr>
<td>3/2 + (\epsilon)</td>
<td>(O(n^{1/4}/\epsilon^2))</td>
<td>yes</td>
<td>Bernstein, Stein ‘15, Bernstein ‘16</td>
</tr>
<tr>
<td>2 + (\epsilon)</td>
<td>(O(\text{poly log } n /\epsilon^2))</td>
<td>yes</td>
<td>Bhattacharya, H, Nanongkai ‘16</td>
</tr>
</tbody>
</table>
Approximate matching

0 Conditional lower bounds only for related problem:
  0 Maintain matching with no augmenting paths of size of size at most 5 (or 7): With polynomial preprocessing time for update cannot be \( o(m^{1/2-\varepsilon}) \) (Conditional on Triangle Detection, 3SUM, BMM, OMv)
Why is it difficult?

Suppose we want to maintain a maximal matching (2-approx.).

\[ \text{degree}(v) = \Theta(n). \]

\( (u, v) \in M. \)
Why is it difficult?

Suppose we want to maintain a maximal matching (2-approx.).

\[ \text{degree}(v) = \Theta(n). \]
Why is it difficult?

Suppose we want to maintain a maximal matching (2-approx.).

\[ \text{degree}(v) = \Theta(n). \]

How to find a new mate for \( v \)?
Why is it difficult?

Suppose we want to maintain a maximal matching (2-approx.).

degree($v$) = $\Theta(n)$.

How to find a new mate for $v$?

Takes $\Theta(n)$ time!
New technique

Primal-dual based method [Bhattacharya, H, Italiano SODA‘15, ICALP‘15]:
- Fully dynamic approximate vertex cover
- Fully dynamic approximate matching and b-matching
- Fully dynamic approximate set cover
Our idea: Fractional matching

\[ \deg(v) = \Theta(n) \]
Our idea: Fractional matching

Initially, set $w(u, v) = 1/n \quad \forall u \in N(v)$.

$W(v) := \sum_{u \in N(v)} w(u, v) = 1$

which equals the size of the integral matching

$\deg(v) = \Theta(n)$
Our idea: Fractional matching

\[ \deg(v) = \Theta(n) \]

Initially, set \( w(u, v) = 1/n \ \forall u \in N(v). \)

During next \( n/2 \) edge deletions:

\[ W(v) := \sum_{u \in N(v)} w(u, v) \geq \frac{1}{2} \]
Our idea: Fractional matching

\[ \text{deg}(v) = \Theta(n) \]

Initially, set \( w(u, v) = 1/n \ \forall u \in N(v) \).

During next \( n/2 \) edge deletions:

\[ W(v) := \sum_{u \in N(v)} w(u, v) \geq \frac{1}{2} \]

After \( n/2 \) deletions reassign weight:

Time \( \theta(n) \)
Our idea: Fractional matching

Initially, set $w(u, v) = 1/n \ \forall u \in N(v)$.

During next $n/2$ edge deletions:

$$W(v) := \sum_{u \in N(v)} w(u, v) \geq \frac{1}{2}$$

After $n/2$ deletions reassign weight:
Time $\theta(n)$

Amortize over previous $n/2$ deletions
A static primal-dual algorithm

Max. fractional matching

Max. \[ \sum_{(u,v) \in E} w(u, v) \]
\[ \sum_{v: (u,v) \in E} w(u, v) \leq 1 \quad \forall u \in V. \]

Min. fractional vertex cover

Min. \[ \sum_{v \in V} y(v) \]
\[ y(u) + y(v) \geq 1 \quad \forall (u, v) \in E. \]
A static primal-dual algorithm

<table>
<thead>
<tr>
<th>Max. fractional matching</th>
<th>Min. fractional vertex cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. [\sum_{(u,v) \in E} w(u, v)]</td>
<td>Min. [\sum_{v \in V} y(v)]</td>
</tr>
<tr>
<td>[\sum_{v: (u,v) \in E} w(u, v) \leq 1 \ \forall u \in V.]</td>
<td>[y(u) + y(v) \geq 1 \ \forall (u,v) \in E.]</td>
</tr>
</tbody>
</table>

Standard primal-dual approach [Kuhn 1955, Dantzig, Ford, Fulkerson 1956, ...]:

- Start with feasible primal solution and infeasible dual solution
- **Primal growth process**: increase all primary variables at same rate
- Whenever primal constraint becomes tight, stop growth of all primary variables involved in constraint and update accordingly dual variable
A static primal-dual algorithm

Max. fractional matching

Max. \( \sum_{(u,v) \in E} w(u, v) \)

\( \sum_{v:(u,v) \in E} w(u, v) \leq 1 \ \forall u \in V. \)

Min. fractional vertex cover

Min. \( \sum_{v \in V} y(v) \)

\( y(u) + y(v) \geq 1 \ \forall (u, v) \in E. \)

New ideas:

• Discretization of standard process to get a hierarchical graph partition with \( O(\log n) \) levels as follows:
  • Level in partition is set of dual variables whose corresponding primal constraint became approximately tight at same instant
• Maintain hierarchical partition and corresponding primal-dual solution lazily
A static algorithm

Hierarchical partition of $V$ into subsets $S_L, \ldots, S_0, S_{-1}$ such that $S = V \setminus S_{-1}$ is a vertex cover

Weights $\{w(e)\}, e \in E$ form fractional matching

<table>
<thead>
<tr>
<th>$M$</th>
<th>${w(e)}$</th>
<th>OPT</th>
<th>$S$</th>
</tr>
</thead>
</table>

$3/2$ \hspace{1cm} $2 + \varepsilon$
A static algorithm

- Hierarchical partition of $V$ into subsets $S_L, \ldots, S_0, S_{-1}$ such that $S = V \setminus S_{-1}$ is a vertex cover
- Weights $\{w(e)\}$, $e \in E$ form fractional matching

$\frac{3}{2} \cdot (2 + \varepsilon) = (3 + \varepsilon)$-approximate max. matching

In this talk: $\frac{3}{2} \cdot 4 = 6$-approximate max. matching
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

\[ \text{weight of a node} \]

\[ \text{weight of an edge} \]

Set \( w(e) = 1/2^L \) for all \( e \in E \).
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \quad \text{weight of a node} \]

\[ \text{weight of an edge} \]

Set \( w(e) = 1/2^L \) for all \( e \in E. \)

\[ W(v) \in [0, 1] \]
A static algorithm

$G(S)$: subgraph of $G$ induced by $S \subseteq V.$

$L = \log n.$

\[
W(v) = \sum_{u \in N(v)} w(u, v)
\]

weight of a node

weight of an edge

\[
W(v) \in [0, 1]
\]

Set $w(e) = \frac{1}{2^L}$ for all $e \in E.$

Partition $V$ into two parts: $S_L$ and $T_L.$
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

\[ \text{weight of a node} \]

\[ \text{weight of an edge} \]

\[ S_L \]

\[ W(v) \in [1/2, 1] \]

\[ T_L \]

\[ W(v) \in [0, 1/2] \]

Set \( w(e) = 1/2^L \) for all \( e \in E \).
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

weight of a node

weight of an edge

Set \( w(e) = \frac{1}{2^L} \) for all \( e \in E \).

Set \( w(e) = 2 \cdot w(e) \) for all \( e \in G(T_L) \).
A static algorithm

\( G(S) \) : subgraph of \( G \) induced by \( S \subseteq V \).

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

Weight of a node

Weight of an edge

Set \( w(e) = 1/2^L \) for all \( e \in E \).

\( W(v) \in [1/2, 1] \)

Set \( w(e) = 2 \cdot w(e) \) for all \( e \in G(T_L) \).

\( W(v) \in [0, 1] \)
A static algorithm

\( G(S) \): subgraph of \( G \) induced by \( S \subseteq V \).

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

Weight of a node

Weight of an edge

\[ W(v) \in [1/2, 1] \]

Set \( w(e) = 1/2^L \) for all \( e \in E \).

\[ W(v) \in [0, 1] \]

Set \( w(e) = 2 \cdot w(e) \) for all \( e \in G(T_L) \).

Partition \( T_L \) into two parts: \( S_{L-1} \) and \( T_{L-1} \).
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

- weight of a node
- weight of an edge

\[ W(v) \in [1/2, 1] \]

- \( S_L \)

- \( S_{L-1} \)

- \( T_{L-1} \)

Set \( w(e) = \frac{1}{2^L} \) for all \( e \in E \).

Set \( w(e) = 2 \cdot w(e) \) for all \( e \in G(T_L) \).

\[ W(v) \in [0, 1/2] \]
**A static algorithm**

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

weight of a node

weight of an edge

\[ \text{Set } w(e) = 1/2^L \text{ for all } e \in E. \]

\[ W(v) \in [1/2, 1] \]

\[ \text{Set } w(e) = 2 \cdot w(e) \text{ for all } e \in G(T_L). \]

\[ W(v) \in [1/2, 1] \]

\[ \text{Set } w(e) = 2 \cdot w(e) \text{ for all } e \in G(T_{L-1}). \]

\[ W(v) \in [0, 1/2] \]
**A static algorithm**

$G(S)$: subgraph of $G$ induced by $S \subseteq V.$

$L = \log n.$

$W(v) = \sum_{u \in N(v)} w(u, v)$

weight of a node

weight of an edge

---

Set $w(e) = 1/2^L$ for all $e \in E.$

Set $w(e) = 2 \cdot w(e)$ for all $e \in G(T_L).$

Set $w(e) = 2 \cdot w(e)$ for all $e \in G(T_{L-1}).$
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

weight of a node

weight of an edge

\[ S_L \]

\[ W(v) \in [1/2, 1] \]

Set \( w(e) = 1/2^L \) for all \( e \in E \).

\[ S_{L-1} \]

\[ W(v) \in [1/2, 1] \]

Set \( w(e) = 2 \cdot w(e) \) for all \( e \in G(T_L) \).

\[ T_{L-1} \]

\[ W(v) \in [0, 1] \]

Set \( w(e) = 2 \cdot w(e) \) for all \( e \in G(T_{L-1}) \).

\text{REPEAT till we reach level 0!}
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

weight of a node

weight of an edge

Set \( w(e) = 1/2^L \) for all \( e \in E \).

\[ S_L \]

\[ W(v) \in [1/2, 1] \]

\[ S_0 \]

\[ W(v) \in [1/2, 1] \]

\[ T_0 = S_{-1} \]

\[ W(v) \in [0, 1/2] \]

\[ \Rightarrow w(e) = 1 \text{ for all } e \in G(T_1) \]

\[ \Rightarrow w(e) = 1 \text{ for all } e \in G(S_0 \cup T_0) \]
A static algorithm

\[ G(S) : \text{subgraph of } G \text{ induced by } S \subseteq V. \]

\[ L = \log n. \]

\[ W(v) = \sum_{u \in N(v)} w(u, v) \]

weight of a node

weight of an edge

Set \( w(e) = 1/2^L \) for all \( e \in E. \)

Set \( w(e) = 1 \) for all \( e \in G(T_1) \)

\[ \Rightarrow w(e) = 1 \text{ for all } e \in G(S_0 \cup T_0) \]

There is no edge in \( G(S_{-1}). \)
A static algorithm

There is no edge in $G(S_{-1})_{47}$
A static algorithm

\[ W(v) \in [1/2, 1] \]

There is no edge in \( G(S_{-1}) \)
A static algorithm

\[ W(v) \in [1/2, 1] \]

\[ W(v) \in [0, 1/2] \]

There is no edge in \( G(S_{-1}) \)
A static algorithm

\[ w(e) = 1/2^i \]
\[ W(v) \in [1/2, 1] \]

\[ W(v) \in [0, 1/2] \]

There is no edge in \( G(S_{-1})_{50} \)
A static algorithm

The set $S = S_0 \cup \cdots S_L$ is a vertex cover of $G$.

There is no edge in $G(S_{-1})_{51}$.
A static algorithm

The set $S = S_0 \cup \cdots \cup S_L$ is a vertex cover of $G$.

The weights $\{w(e)\}$ give a fractional matching in $G$.

$w(e) = 1/2^i$  \quad  $W(v) \in [1/2, 1]$  \quad  $W(v) \in [0, 1/2]$
A static algorithm

The set $S = S_0 \cup \cdots S_L$ is a vertex cover of $G$.

The weights $\{w(e)\}$ give a fractional matching in $G$.

Claim: $S$ is a 4 approx. minimum vertex cover in $G$.

$w(e) = 1/2^i$

$W(v) \in [1/2, 1]$

$W(v) \in [0, 1/2]$
A static algorithm

The set $S = S_0 \cup \cdots \cup S_L$ is a vertex cover of $G$.

The weights $\{w(e)\}$ give a fractional matching in $G$.

Claim: $S$ is a 4 approx. minimum vertex cover in $G$.

Proof of the claim:

$W(v) \in [1/2, 1]$ 

$|S| \leq \sum_{v \in S} 2 \cdot W(v)$

$W(v) \in [0, 1/2]$
A static algorithm

The set $S = S_0 \cup \cdots S_L$ is a vertex cover of $G$.

The weights $\{w(e)\}$ give a fractional matching in $G$.

Claim: $S$ is a 4 approx. minimum vertex cover in $G$.

proof of the claim:

$W(v) \in [1/2, 1]$ \hspace{1cm} $|S| \leq \sum_{v \in S} 2 \cdot W(v)$

$\leq 2 \cdot \sum_{(u,v) \in E} 2 \cdot w(u, v)$

$W(v) \in [0, 1/2]$
A static algorithm

The set $S = S_0 \cup \cdots S_L$ is a vertex cover of $G$.

The weights $\{w(e)\}$ give a fractional matching in $G$.

Claim: $S$ is a 4 approx. minimum vertex cover in $G$.

Proof of the claim:

$W(v) \in [1/2, 1]$

$|S| \leq \sum_{v \in S} 2 \cdot W(v)$

$\leq 2 \cdot \sum_{(u,v) \in E} 2 \cdot w(u,v)$

$= 4 \cdot |M|$

where $M$ is the fractional matching
A static algorithm

The set $S = S_0 \cup \cdots S_L$ is a vertex cover of $G$.

The weights $\{w(e)\}$ give a fractional matching in $G$.

Claim: $S$ is a 4 approx. minimum vertex cover in $G$.

proof of the claim:

$W(v) \in [1/2, 1]$

$|S| \leq \sum_{v \in S} 2 \cdot W(v)$

$\leq 2 \cdot \sum_{(u,v) \in E} 2 \cdot w(u,v)$

$= 4 \cdot |M|$  \hspace{1cm} \text{Apply LP-duality!}$

$W(v) \in [0, 1/2]$
Making it dynamic

\[ W(v) \in [1/2, 1] \]

\[ W(v) \in [0, 1/2] \]
Making it dynamic

A node $v$ is *clean* if $W(v)$ is within the prescribed range, and *dirty* otherwise.

$W(v) \in [1/2, 1]$

$W(v) \in [0, 1/2]$
Making it dynamic

A node $v$ is *clean* if $W(v)$ is within the prescribed range, and *dirty* otherwise.

Initially, (a) the graph is empty, and

(b) every node is at level $-1$, and hence clean.

$W(v) \in [1/2, 1]$

$W(v) \in [0, 1/2]$
Making it dynamic

A node \( v \) is *clean* if \( W(v) \) is within the prescribed range, and *dirty* otherwise.

Initially, (a) the graph is empty, and

(b) every node is at level \( -1 \), and hence clean.

\[ W(v) \in [1/2, 1] \]

Suppose that every node is clean before an edge insertion/deletion in \( G \).

The current edge insertion/deletion may make some of the nodes dirty. How to fix this?

\[ W(v) \in [0, 1/2] \]
Making it dynamic

A node $v$ is *clean* if $W(v)$ is within the prescribed range, and *dirty* otherwise.

\[
W(v) \in [1/2, 1]
\]

**FIX-DIRTY-NODES**

**WHILE** there is a dirty node $x$

If $W(x)$ is too large, then $\ell(x) \leftarrow \ell(x) + 1$. 

\[
W(v) \in [0, 1/2]
\]
Making it dynamic

A node $v$ is *clean* if $W(v)$ is within the prescribed range, and *dirty* otherwise.

Fix-Dirty-Nodes

$W(v) \in [1/2, 1]$

Edge weight changes

$w(e) = 1/2^i$

$w(e) = 1/2^{\ell(x)}$

**WHILE** there is a dirty node $x$

If $W(x)$ is too large, then $\ell(x) \leftarrow \ell(x) + 1$. 

$W(v) \in [0, 1/2]$
Making it dynamic

A node \( v \) is \textit{clean} if \( W(v) \) is within the prescribed range, and \textit{dirty} otherwise.

\[
w(e) = 1/2^i\quad W(v) \in [1/2, 1]
\]

\[
\text{Fix-Dirty-Nodes}
\]

\[
\text{While there is a dirty node } x
\]

\[
\text{If } W(x) \text{ is too large, then } \ell(x) \leftarrow \ell(x) + 1.
\]

\[
\text{Else } \ell(x) \leftarrow \ell(x) - 1.
\]
A node $v$ is *clean* if $W(v)$ is within the prescribed range, and *dirty* otherwise.

If and when the WHILE loop terminates, every node is clean.

But how to bound the amortized update time?

Potential function!

$\text{W}$-DIRTY-NODES

WHILE there is a dirty node $x$

If $W(x)$ is too large, then $\ell(x) \leftarrow \ell(x) + 1$.

Else $\ell(x) \leftarrow \ell(x) - 1$. 

\[ W(v) \in [0, 1/2] \]
Summary

We have a deterministic primal-dual algorithm for dynamic matching and vertex cover.

Update time: $O(\log n/\epsilon^2)$.

Approximation ratio: $3 + \epsilon$ (for matching)

$2 + \epsilon$ (for vertex cover)

Can we maintain a $O(1)$ approx. integral matching?
Deterministic rounding
(Bhattacharya, H, Nanongkai, STOC `16)

So far: fractional matching = value version
  (3 + \epsilon)-approximate in $O(\log n)$ amortized update time

Now: integral matching = matching version
  (2 + \epsilon)-approximate integral matching in $O(\text{polylog } n)$ amortized update time
Deterministic rounding

The support \( X(w) = \{e, e \in E, w(e) > 0\} \).

Main idea:
- Turn \( w \) into \( w' \) such that \( \forall v, W(v) = W'(v) \) and \( \deg_{X(w'_i)}(v) = O(\log n) \).

Let \( G' = (V, X(w')) \):
- Size of fractional matching \( w' \) in \( G' = \) size of fractional matching \( w \) in \( G \)
- Maximum degree in \( G' \) is \( O(\log n) \)
Deterministic rounding

Main idea:
0 Let $G' = (V, X(w'))$:
   0 Size of fractional matching $w'$ in $G' = $ size of fractional matching $w$ in $G$
   0 Maximum degree in $G'$ is $O(\log n)$
0 Theorem: Given a fraction matching $w$ there exists an integral matching $M$ s.t.
   $$|M| \geq \frac{3}{2} \sum_{e \in E} w(e)$$
   ➢ Integral matching $M'$ in $G'$: $|M'| \geq \frac{3}{2} \sum_{e \in E} w'(e) = \frac{3}{2} \sum_{e \in E} w(e)$
   ➢ $M'$ is a $\frac{3}{2} (2 + \varepsilon) = (3 + \varepsilon')$-approximate integral matching of $G$
   ➢ Improved analysis gives $(2 + \varepsilon)$
Deterministic rounding

Implementation of main idea:

1. Repeatedly take an Euler tour of $G$, delete every alternate edge and double the weight on the remaining edges until every node has degree $O(\log n)$
   - Weight of node does not change, but degree halves

2. Use Gupta-Peng’s $(1 + \varepsilon)$-approximate algorithm on $G'$: update time $O(\text{max degree}) = O(\log n)$

- Can be maintained dynamically in polylog time
Open Problems

1. Better than 2-approximation in polylog update time

2. Lower bounds on the update time for approximate matching

3. Apply primal-dual approach to other dynamic graph problems
Thank you!