Constructing Large Matchings via Query Access to a Maximal Matching Oracle

FSTTCS 2020

Lidiya Khalidah binti Khalil and Christian Konrad

University of Bristol
Let $G = (V, E)$ be a graph
Let $G = (V, E)$ be a graph.

- A *matching* $M \subseteq E$ is a subset of non-adjacent edges.

![Matching Example]

A matching $M$ is maximal if $M \cup \{e\}$ is not a matching, for every $e \in E \setminus M$.

$M$ is maximum if for every other matching $M' \subseteq E$:

$|M'| \geq |M|$.

Property: $|\text{maximal matching}| \geq \frac{1}{2} |\text{maximum matching}|$.
Matchings

Let $G = (V, E)$ be a graph

- A *matching* $M \subseteq E$ is a subset of non-adjacent edges

\[ \begin{align*}
\text{matching} & \quad \text{not a matching} \\
\begin{tikzpicture}
  \node (a) at (0,0) [circle,fill,inner sep=2pt] {};
  \node (b) at (1,0) [circle,fill,inner sep=2pt] {};
  \node (c) at (2,0) [circle,fill,inner sep=2pt] {};
  \node (d) at (3,0) [circle,fill,inner sep=2pt] {};
  \draw (a) -- (b);
  \draw (c) -- (d);
\end{tikzpicture} & \quad \begin{tikzpicture}
  \node (a) at (0,0) [circle,fill,inner sep=2pt] {};
  \node (b) at (1,0) [circle,fill,inner sep=2pt] {};
  \node (c) at (2,0) [circle,fill,inner sep=2pt] {};
  \node (d) at (3,0) [circle,fill,inner sep=2pt] {};
  \draw (a) -- (b);
  \draw (c) -- (d);
\end{tikzpicture}
\end{align*} \]

- $M$ is *maximal* if $M \cup \{e\}$ is not a matching, for every $e \in E \setminus M$

\[ \begin{align*}
\text{maximal} & \quad \text{not maximal} \\
\begin{tikzpicture}
  \node (a) at (0,0) [circle,fill,inner sep=2pt] {};
  \node (b) at (1,0) [circle,fill,inner sep=2pt] {};
  \node (c) at (2,0) [circle,fill,inner sep=2pt] {};
  \node (d) at (3,0) [circle,fill,inner sep=2pt] {};
  \draw (a) -- (b);
  \draw (c) -- (d);
\end{tikzpicture} & \quad \begin{tikzpicture}
  \node (a) at (0,0) [circle,fill,inner sep=2pt] {};
  \node (b) at (1,0) [circle,fill,inner sep=2pt] {};
  \node (c) at (2,0) [circle,fill,inner sep=2pt] {};
  \node (d) at (3,0) [circle,fill,inner sep=2pt] {};
  \draw (a) -- (b);
  \draw (c) -- (d);
\end{tikzpicture}
\end{align*} \]
Let $G = (V, E)$ be a graph

- A matching $M \subseteq E$ is a subset of non-adjacent edges

![Matching and Not a Matching](image1)

- $M$ is maximal if $M \cup \{e\}$ is not a matching, for every $e \in E \setminus M$

![Maximal and Not Maximal](image2)

- $M^*$ is maximum if for every other matching $M \subseteq E$: $|M^*| \geq |M|$
Matchings

Let $G = (V, E)$ be a graph

- A matching $M \subseteq E$ is a subset of non-adjacent edges

  ![Diagram of matching and non-matching edges]

  - matching
  - not a matching

- $M$ is maximal if $M \cup \{e\}$ is not a matching, for every $e \in E \setminus M$

  ![Diagram of maximal and not maximal edges]

  - maximal
  - not maximal

- $M^*$ is maximum if for every other matching $M \subseteq E$: $|M^*| \geq |M|$  

  ![Diagram of maximum edge set]

  - maximum

**Property:** $|\text{maximal matching}| \geq \frac{1}{2} |\text{maximum matching}|$
Computing Maximal Matchings is often easy

**Goal:** Maximum Matching approximation (better than 1/2-approx.)
Computing Maximal Matchings is often easy

**Goal:** Maximum Matching approximation (better than \(1/2\)-approx.)

**In many computational models...** (e.g. streaming, distributed models)
- computing maximal matchings is easy
- computing maximum matching approximations is more difficult
Computing Maximal Matchings is often easy

**Goal:** Maximum Matching approximation (better than 1/2-approx.)

In many computational models... (e.g. streaming, distributed models)

- computing maximal matchings is easy
- computing maximum matching approximations is more difficult

**Edge-arrival Streaming Model:**

```
... 1 2 3 4 5 6 7 ...
```

Lidiya Khalidah binti Khalil and Christian Konrad

Large Matchings via a Maximal Matching Oracle 3 / 17
Computing Maximal Matchings is often easy

**Goal:** Maximum Matching approximation (better than 1/2-approx.)

**In many computational models...** (e.g. streaming, distributed models)
- computing maximal matchings is easy
- computing maximum matching approximations is more difficult

**Edge-arrival Streaming Model:**
- Input stream: Sequence of edges of input graph $G = (V, E)$ with $n = |V|$ in arbitrary order
  
  $S = e_2e_1e_4e_3$

---

Lidiya Khalidah binti Khalil and Christian Konrad
Large Matchings via a Maximal Matching Oracle
Computing Maximal Matchings is often easy

**Goal:** Maximum Matching approximation (better than 1/2-approx.)

**In many computational models...** (e.g. streaming, distributed models)
- computing maximal matchings is easy
- computing maximum matching approximations is more difficult

**Edge-arrival Streaming Model:**
- Input stream: Sequence of edges of input graph $G = (V, E)$ with $n = |V|$ in arbitrary order
  
  $$S = e_2e_1e_4e_3$$

- Goal: Few passes algorithms with small space
Computing Maximal Matchings is often easy

**Goal:** Maximum Matching approximation (better than 1/2-approx.)

In many computational models... (e.g. streaming, distributed models)
- computing maximal matchings is easy
- computing maximum matching approximations is more difficult

**Edge-arrival Streaming Model:**
- Input stream: Sequence of edges of input graph $G = (V, E)$ with $n = |V|$ in arbitrary order

$$S = e_2 e_1 e_4 e_3$$

- Goal: Few passes algorithms with small space
- Streaming Maximal Matching Algorithm: Insert current edge into initially empty matching if possible ($\text{GREEDY}$), using space $\tilde{O}(n)$
# State of the Art Streaming Matching Algorithms

<table>
<thead>
<tr>
<th># passes</th>
<th>Approximation</th>
<th>det/rand</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bipartite Graphs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>det</td>
<td>Greedy, folklore</td>
</tr>
<tr>
<td>2</td>
<td>$2 - \sqrt{2} \approx 0.5857$</td>
<td>rand</td>
<td>Konrad ’18</td>
</tr>
<tr>
<td>3</td>
<td>0.6067</td>
<td>rand</td>
<td>Konrad ’18</td>
</tr>
<tr>
<td>$O(\frac{1}{\epsilon^2})$</td>
<td>$1 - \epsilon$</td>
<td>det</td>
<td>Assadi, Liu, Tarjan ’21</td>
</tr>
<tr>
<td><strong>General Graphs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>det</td>
<td>Greedy, folklore</td>
</tr>
<tr>
<td>2</td>
<td>0.53125</td>
<td>det</td>
<td>Kale and Tirodkar ’17</td>
</tr>
<tr>
<td>$\frac{1}{\epsilon} O(\frac{1}{\epsilon})$</td>
<td>$1 - \epsilon$</td>
<td>det</td>
<td>Tirodkar ’18</td>
</tr>
</tbody>
</table>

Most of these algorithms (including previous works) solely run Greedy in carefully selected subgraphs in each pass, thereby collecting edges and outputting the largest matching among the edges stored.

How large a matching can we compute if we solely invoke Greedy in each pass?
Most of these algorithms (including previous works) solely run \textsc{Greedy} in carefully selected subgraphs in each pass, thereby collecting edges and outputting the largest matching among the edges stored.
<table>
<thead>
<tr>
<th># passes</th>
<th>Approximation</th>
<th>det/rand</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bipartite Graphs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>det</td>
<td>Greedy, folklore</td>
</tr>
<tr>
<td>2</td>
<td>$2 - \sqrt{2} \approx 0.5857$</td>
<td>rand</td>
<td>Konrad ’18</td>
</tr>
<tr>
<td>3</td>
<td>0.6067</td>
<td>rand</td>
<td>Konrad ’18</td>
</tr>
<tr>
<td>$O\left(\frac{1}{\epsilon^2}\right)$</td>
<td>$1 - \epsilon$</td>
<td>det</td>
<td>Assadi, Liu, Tarjan ’21</td>
</tr>
<tr>
<td><strong>General Graphs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>det</td>
<td>Greedy, folklore</td>
</tr>
<tr>
<td>2</td>
<td>0.53125</td>
<td>det</td>
<td>Kale and Tirodkar ’17</td>
</tr>
<tr>
<td>$\frac{1}{\epsilon}O\left(\frac{1}{\epsilon}\right)$</td>
<td>$1 - \epsilon$</td>
<td>det</td>
<td>Tirodkar ’18</td>
</tr>
</tbody>
</table>

Most of these algorithms (including previous works) solely run Greedy in carefully selected subgraphs in each pass, thereby collecting edges and outputting the largest matching among the edges stored.

**How large a matching can we compute if we solely invoke Greedy in each pass?**
Maximal Matching Oracle

Matching Game:

Player Oracle query($V_i$) response: maximal matching in $G[V_i]$

Player and oracle play $r$ rounds of a "matching game"

In each round $r$:

1. Player queries a subset of vertices $V_i \subseteq V$
2. Oracle returns maximal matching $M_i$ in induced subgraph $G[V_i]$

Player outputs largest matching in $\bigcup_{1 \leq i \leq r} M_i$

Research Question:

What is the trade-off between the number of rounds and the approximation ratio?
Matching Game:

- Player and oracle play $r$ rounds of a “matching game”
- In each round $r$:
  1. Player queries a subset of vertices $V_i \subseteq V$
  2. Oracle returns maximal matching $M_i$ in induced subgraph $G[V_i]$
- Player outputs largest matching in $\bigcup_{1 \leq i \leq r} M_i$
Maximal Matching Oracle

Matching Game:

query($V_i$)

response: maximal matching in $G[V_i]$

- Player and oracle play $r$ rounds of a “matching game”
- In each round $r$:
  1. Player queries a subset of vertices $V_i \subseteq V$
  2. Oracle returns maximal matching $M_i$ in induced subgraph $G[V_i]$
- Player outputs largest matching in $\bigcup_{1 \leq i \leq r} M_i$

Research Question: What is the trade-off between the number of rounds and the approximation ratio?
Matching Game - Upper Bounds

Upper Bounds for Bipartite Graphs:

Oracle

Player

query($V_i$)

maximal matching in $G[V_i]$
Upper Bounds for Bipartite Graphs:

- **1 round:** $\frac{1}{2}$-approximation
  - $\text{query}(V)$ yields maximal matching in input graph

Upper Bounds for General Graphs:

- (except 1 round $\frac{1}{2}$-approx.)
Matching Game - Upper Bounds

**Upper Bounds for Bipartite Graphs:**

- **1 round:** $\frac{1}{2}$-approximation
  - query($V$) yields maximal matching in input graph
- **2 rounds:** $\geq \frac{1}{2}$-approximation
Matching Game - Upper Bounds

Upper Bounds for Bipartite Graphs:

- **1 round:** $\frac{1}{2}$-approximation
  query($V$) yields maximal matching in input graph

- **2 rounds:** $\geq \frac{1}{2}$-approximation

- **3 rounds:** $\frac{3}{5}$-approximation
  3-pass streaming algorithm analysed by Kale and Tirodkar ’17
Upper Bounds for Bipartite Graphs:

- **1 round**: $\frac{1}{2}$-approximation
  
  query($V$) yields maximal matching in input graph

- **2 rounds**: $\geq \frac{1}{2}$-approximation

- **3 rounds**: $3/5$-approximation
  
  3-pass streaming algorithm analysed by Kale and Tirodkar ’17

- **$\Theta(\frac{1}{\epsilon^6})$ rounds**: $(1 - \epsilon)$-approximation
  
  Streaming algorithm that runs in $\Theta(\frac{1}{\epsilon^5})$ passes by Eggert et al. ’12 can be adapted to the model
Matching Game - Upper Bounds

Upper Bounds for Bipartite Graphs:

- **1 round**: $\frac{1}{2}$-approximation
  query($V$) yields maximal matching in input graph

- **2 rounds**: $\geq \frac{1}{2}$-approximation

- **3 rounds**: $3/5$-approximation
  3-pass streaming algorithm analysed by Kale and Tirodkar ’17

- $\Theta\left(\frac{1}{\epsilon^6}\right)$ rounds: $(1 - \epsilon)$-approximation
  Streaming algorithm that runs in $\Theta\left(\frac{1}{\epsilon^5}\right)$ passes by Eggert et al. ’12
  can be adapted to the model

Upper Bounds for General Graphs:
Upper Bounds for Bipartite Graphs:

- **1 round:** $\frac{1}{2}$-approximation
  query($V$) yields maximal matching in input graph
- **2 rounds:** $\geq \frac{1}{2}$-approximation
- **3 rounds:** 3/5-approximation
  3-pass streaming algorithm analysed by Kale and Tirodkar '17
- **$\Theta(\frac{1}{\epsilon^6})$ rounds:** $(1 - \epsilon)$-approximation
  Streaming algorithm that runs in $\Theta(\frac{1}{\epsilon^5})$ passes by Eggert et al. '12 can be adapted to the model

Upper Bounds for General Graphs: $X$ (except 1 round $\frac{1}{2}$-approx.)
Our Results

We give the following Lower Bound Results:

Bipartite Graphs
- 1 round: 1\frac{1}{2}-approximation is optimal
- 2 rounds: \geq 1\frac{1}{2}-approximation: 1\frac{1}{2} is best possible
- 3 rounds: 3\frac{5}{6}-approximation is optimal

\Theta\left(\frac{1}{\epsilon^6}\right) rounds (1 - \epsilon)-approximation:

General Graphs
- (1 round) 1/2-approximation

Ω\left(\frac{n}{\epsilon}\right) rounds are needed for an approximation ratio 1/2 + \epsilon, for any \epsilon > 0.

Outline:
1. \Omega\left(\frac{1}{\epsilon^6}\right) rounds are needed for a (1 - \epsilon)-approximation
2. \Omega\left(\frac{n}{\epsilon}\right) rounds are needed for a (1/2 + \epsilon)-approximation in general graphs, \epsilon > 0
3. 0.
Our Results

We give the following Lower Bound Results:

Bipartite Graphs

- 1 round $\frac{1}{2}$-approximation:

- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible

- 3 rounds $\frac{3}{5}$-approximation: optimal

General Graphs

- (1 round $\frac{1}{2}$-approximation)

- $\Omega(n)$ rounds are needed for an approximation ratio $\frac{1}{2} + \epsilon$, for any $\epsilon > 0$
We give the following Lower Bound Results:

**Bipartite Graphs**
- 1 round $\frac{1}{2}$-approximation: optimal ✓

**General Graphs**
- (1 round $\frac{1}{2}$-approximation)
- $\Omega(n)$ rounds are needed for an approximation ratio $\frac{1}{2} + \epsilon$, for any $\epsilon > 0$. 

Outline:
1. $\Omega(\frac{1}{\epsilon})$ rounds are needed for a $(1 - \epsilon)$-approximation
2. $\Omega(n)$ rounds needed for $(\frac{1}{2} + \epsilon)$-approx. in general graphs, $\epsilon > 0$
3. $\theta$. $6$-approximation lower bound for 3 rounds (main technical result)
Our Results

We give the following Lower Bound Results:

**Bipartite Graphs**

- 1 round \( \frac{1}{2} \)-approximation: optimal ✓
- 2 rounds \( \geq \frac{1}{2} \)-approximation:
Our Results

We give the following Lower Bound Results:

Bipartite Graphs
- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓

General Graphs
- $(1$ round $\frac{1}{2}$-approximation)
- $\Omega(n)$ rounds are needed for an approximation ratio $\frac{1}{2} + \epsilon$, for any $\epsilon > 0$
Our Results

We give the following Lower Bound Results:

**Bipartite Graphs**

- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation:
Our Results

We give the following Lower Bound Results:

Bipartite Graphs

- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation: optimal ✓
Our Results

We give the following Lower Bound Results:

Bipartite Graphs

- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation: optimal ✓
- $\Theta\left(\frac{1}{\epsilon^6}\right)$ rounds $(1 - \epsilon)$-approximation:
We give the following Lower Bound Results:

**Bipartite Graphs**

- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation: optimal ✓
- $\Theta(\frac{1}{\epsilon^6})$ rounds $(1 - \epsilon)$-approximation: $\Omega(\frac{1}{\epsilon})$ rounds are needed for a $(1 - \epsilon)$-approximation
Our Results

We give the following Lower Bound Results:

Bipartite Graphs
- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation: optimal ✓
- $\Theta\left(\frac{1}{\epsilon^6}\right)$ rounds $(1 - \epsilon)$-approximation: $\Omega\left(\frac{1}{\epsilon}\right)$ rounds are needed for a $(1 - \epsilon)$-approximation

General Graphs (1 round 1/2-approximation)
Our Results

We give the following Lower Bound Results:

**Bipartite Graphs**
- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation: optimal ✓
- $\Theta\left(\frac{1}{\epsilon^6}\right)$ rounds $(1 - \epsilon)$-approximation: $\Omega\left(\frac{1}{\epsilon}\right)$ rounds are needed for a $(1 - \epsilon)$-approximation

**General Graphs** (1 round 1/2-approximation)
$\Omega(n)$ rounds are needed for an approximation ratio $\frac{1}{2} + \epsilon$, for any $\epsilon > 0$
Our Results

We give the following Lower Bound Results:

**Bipartite Graphs**
- 1 round $\frac{1}{2}$-approximation: optimal ✓
- 2 rounds $\geq \frac{1}{2}$-approximation: $\frac{1}{2}$ is best possible ✓
- 3 rounds $\frac{3}{5}$-approximation: optimal ✓
- $\Theta(\frac{1}{\epsilon^6})$ rounds $(1 - \epsilon)$-approximation: $\Omega(\frac{1}{\epsilon})$ rounds are needed for a $(1 - \epsilon)$-approximation

**General Graphs** (1 round $\frac{1}{2}$-approximation)
$\Omega(n)$ rounds are needed for an approximation ratio $\frac{1}{2} + \epsilon$, for any $\epsilon > 0$

**Outline:**
1. $\Omega(\frac{1}{\epsilon})$ rounds are needed for a $(1 - \epsilon)$-approximation
2. $\Omega(n)$ rounds needed for $(\frac{1}{2} + \epsilon)$-approx. in general graphs, $\epsilon > 0$
3. 0.6-approximation lower bound for 3 rounds (main technical result)
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.
Theorem. Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

Proof. Consider semi-complete graph on $2c$ vertices

![Semi-complete graph diagram]
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$

![Diagram of a semi-complete graph with perfect matching](image_url)
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$
- Sub-optimal matching constitutes at best a $\frac{c - 1}{c} = (1 - \frac{1}{c})$-approximation
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$
- Sub-optimal matching constitutes at best a $\frac{c-1}{c} = (1 - \frac{1}{c})$-approximation
- **Insight:** Any query gives at most one edge from $M^*$
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$
- Sub-optimal matching constitutes at best a $\frac{c-1}{c} = (1 - \frac{1}{c})$-approximation
- **Insight:** Any query gives at most one edge from $M^*$
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$
- Sub-optimal matching constitutes at best a $(1 - \frac{1}{c})$-approximation
- **Insight:** Any query gives at most one edge from $M^*$
- Hence, to achieve a $(1 - \epsilon)$-approximation, for $\epsilon = \frac{1}{c+1}$, $c = \frac{1}{\epsilon} - 1$ queries are needed
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$
- Sub-optimal matching constitutes at best a $\frac{c-1}{c} = (1 - \frac{1}{c})$-approximation

**Insight:** Any query gives at most one edge from $M^*$
- Hence, to achieve a $(1 - \epsilon)$-approximation, for $\epsilon = \frac{1}{c+1}$, $c = \frac{1}{\epsilon} - 1$ queries are needed
- Use multiple disjoint gadgets for arbitrary $n$
**Theorem.** Any query algorithm with approximation factor $1 - \epsilon$ requires at least $\frac{1}{\epsilon} - 1$ queries, even in bipartite graphs.

**Proof.** Consider semi-complete graph on $2c$ vertices

- Unique perfect matching $M^*$ of size $c$
- Sub-optimal matching constitutes at best a $\frac{c-1}{c} = (1 - \frac{1}{c})$-approximation
- **Insight:** Any query gives at most one edge from $M^*$
- Hence, to achieve a $(1 - \epsilon)$-approximation, for $\epsilon = \frac{1}{c+1}$, $c = \frac{1}{\epsilon} - 1$ queries are needed
- Use multiple disjoint gadgets for arbitrary $n$
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$. 
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

```
// Bomb graph on n vertices
```

"outside" edges $=$ perfect matching
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

- “outside” edges = perfect matching
- “inside” edge: blocks two optimal edges
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

- "outside" edges = perfect matching
- "inside" edge: blocks two optimal edges
- $\left(\frac{1}{2} + \frac{r}{n}\right)$-approx: $r$ "outside" edges
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

- “outside” edges = perfect matching
- “inside” edge: blocks two optimal edges
- $(\frac{1}{2} + \frac{r}{n})$-approx: $r$ “outside” edges
- **Insight:** $\leq 1$ “outside” edges per query
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

- “outside” edges = perfect matching
- “inside” edge: blocks two optimal edges
- $(\frac{1}{2} + \frac{r}{n})$-approx: $r$ “outside” edges
- **Insight:** \( \leq 1 \) “outside” edges per query
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

- "outside" edges = perfect matching
- "inside" edge: blocks two optimal edges
- $(\frac{1}{2} + \frac{r}{n})$-approx: $r$ "outside" edges
- **Insight:** $\leq 1$ "outside" edges per query
- $r$ queries needed
**Theorem.** Any $r$-round query algorithm for general graphs has an approximation of at most $\frac{1}{2} + \frac{r}{n}$.

**Proof.** Consider a bomb graph on $n$ vertices

- “outside” edges = perfect matching
- “inside” edge: blocks two optimal edges
- $(\frac{1}{2} + \frac{r}{n})$-approx: $r$ “outside” edges
- **Insight:** $\leq 1$ “outside” edges per query
- $r$ queries needed
Deterministic / Randomized Query Algorithms:
- Lower bounds on previous slides hold even if the input graph is known by the player
- They also hold for randomized query algorithms

Lower Bound for 3 Rounds on Bipartite Graphs:
- More subtle argument
- Oracle builds graph that depends on the queries
- Lower bound therefore only holds for deterministic algorithms
Three Round Query Algorithm for Bipartite Graphs

Algorithm (input graph $G = (A, B, E)$)
Three Round Query Algorithm for Bipartite Graphs

Algorithm (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow \text{endpoints of path of length two in } M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. return largest matching using edges $M \cup M_L \cup M_R$
Algorithm (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow \text{endpoints of path of length two in } M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. return largest matching using edges $M \cup M_L \cup M_R$

Input graph $G = (A, B, E)$ with perfect matching $M^*$
**Algorithm** (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow$ endpoints of path of length two in $M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. **return** largest matching using edges $M \cup M_L \cup M_R$

1st query: Matching $M$
Three Round Query Algorithm for Bipartite Graphs

**Algorithm** (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow$ endpoints of path of length two in $M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. **return** largest matching using edges $M \cup M_L \cup M_R$

Subgraph $G[A(M) \cup \overline{M(B)}]$
Algorithm (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow$ endpoints of path of length two in $M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. return largest matching using edges $M \cup M_L \cup M_R$

2nd query: Matching $M_L$
Three Round Query Algorithm for Bipartite Graphs

**Algorithm** (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow$ endpoints of path of length two in $M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. **return** largest matching using edges $M \cup M_L \cup M_R$

$B'$ and Subgraph $G[A(M) \cup B']$
Algorithm (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow$ endpoints of path of length two in $M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup \overline{M(A)})$
5. return largest matching using edges $M \cup M_L \cup M_R$
**Algorithm** (input graph $G = (A, B, E)$)

1. $M \leftarrow \text{query}(A \cup B)$
2. $M_L \leftarrow \text{query}(M(A) \cup \overline{M(B)})$
3. $B' \subseteq B(M) \leftarrow \text{endpoints of path of length two in } M \cup M_L$
4. $M_R \leftarrow \text{query}(B' \cup M(A))$
5. **return** largest matching using edges $M \cup M_L \cup M_R$

Largest matching in $M \cup M_L \cup M_R$ (M augmented with $M_L \cup M_R$)
Analysis:
Three Round Query Algorithm for Bipartite Graphs (2)

Analysis: $\frac{3}{5}$-approximation algorithm [Kale and Tirodkar, ’17]
Three Round Query Algorithm for Bipartite Graphs (2)

Analysis: \( \frac{3}{5} \)-approximation algorithm [Kale and Tirodkar, ’17]

Worst-case Example:
Three Round Query Algorithm for Bipartite Graphs (2)

**Analysis:** \(\frac{3}{5}\)-approximation algorithm [Kale and Tirodkar, ’17]

**Worst-case Example:**

\[
\begin{align*}
M & \leftarrow \text{query}(A \cup B) \\
M_L & \leftarrow \text{query}(M(A) \cup \overline{M(B)}) \\
B' & \subseteq B(M) \leftarrow \text{endpoints of path of length two in } M \cup M_L \\
M_R & \leftarrow \text{query}(B' \cup \overline{M(A)})
\end{align*}
\]
**Strategy:** Bound “knowledge” about input graph after each query ("structure graph"); ensure perfect matching can be added.
**Strategy:** Bound “knowledge” about input graph after each query (“structure graph”); ensure perfect matching can be added

**First Query:**
- Oracle commits to structure below and returns subset of edges $M$ (no edges between $A_{out}$ and $B_{out}$)
- A perfect matching (blue edges) can be added, which implies that approximation factor is $3/5$ at best after first query
**Second Query:**

- Information can be bounded by structure below - grey edges indicate that edges are not present in output graph.
- Again, perfect matching can be added, which implies that approximation factor is $3/5$ at best after second query.

![Diagram](image)
Third Query:

- Structure cannot easily be captured using a single “structure graph“
- Instead, case distinctions with cleverly grouping cases together
Third Query:
- Structure cannot easily be captured using a single “structure graph“
- Instead, case distinctions with cleverly grouping cases together

Example Case: Query includes \{b_1, b_2, b_3\}
Third Query:
- Structure cannot easily be captured using a single “structure graph“
- Instead, case distinctions with cleverly grouping cases together

Example Case: Query includes \{b_1, b_2, b_3\}
Third Query:
- Structure cannot easily be captured using a single “structure graph”
- Instead, case distinctions with cleverly grouping cases together

Example Case: Query includes \{b_1, b_2, b_3\}

Key Technique: Structural properties that allow eliminating cases
Open Problems and Outlook

Open Problems:

- Can we compute a Maximum Matching in $o(n^2)$ rounds?
- Can we prove that $\Omega(1/\epsilon^2)$ rounds are required for computing a $(1-\epsilon)$-approximation?

Outlook:

Extensions: Edge queries instead of vertex queries
Randomization?
Open Problems and Outlook

**Open Problems:**

- Can we compute a Maximum Matching in $o(n^2)$ rounds?

**Outlook:**

- Extensions: Edge queries instead of vertex queries
- Randomization?
Open Problems and Outlook

Open Problems:

- Can we compute a Maximum Matching in $o(n^2)$ rounds?
- Can we prove that $\Omega(1/\epsilon^2)$ rounds are required for computing a $(1 - \epsilon)$-approximation?

Outlook:

- Extensions: Edge queries instead of vertex queries
- Randomization?
Open Problems and Outlook

Open Problems:
- Can we compute a Maximum Matching in $o(n^2)$ rounds?
- Can we prove that $\Omega(1/\epsilon^2)$ rounds are required for computing a $(1 - \epsilon)$-approximation?

Outlook:

Extensions: Edge queries instead of vertex queries
Randomization?
Open Problems and Outlook

Open Problems:

- Can we compute a Maximum Matching in $o(n^2)$ rounds?
- Can we prove that $\Omega(1/\epsilon^2)$ rounds are required for computing a $(1 - \epsilon)$-approximation?

Outlook:

- Extensions: Edge queries instead of vertex queries
Open Problems and Outlook

Open Problems:
- Can we compute a Maximum Matching in $o(n^2)$ rounds?
- Can we prove that $\Omega(1/\epsilon^2)$ rounds are required for computing a $(1 - \epsilon)$-approximation?

Outlook:
- Extensions: Edge queries instead of vertex queries
- Randomization?
Thank you for your attention.