Lec 12–??: Counting Triangles

COSC–254 – March 6-??, 2019
Outline

*Graphs*: basic definitions

*Triangles*: what and why
Outline

Graphs: basic definitions

Triangles: what and why

Approximate counting of triangles in edge data streams
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*

*Approximate* counting of triangles in a static graph
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*

*Approximate* counting of triangles in a static graph

Counting triangles in *MapReduce*
Feedback time!

Outline

*Graphs*: basic definitions

*Triangles*: what and why
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*

*Approximate* counting of triangles in a static graph
Outline

*Graphs*: basic definitions

*Triangles*: what and why

*Approximate* counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*

*Approximate* counting of triangles in a static graph

Counting triangles in *MapReduce*
Many natural phenomena can be represented as *networks of interactions* between elements:

- social networks (off- and on-line)
- the Web
- the Internet
- sets of proteins
- roads
- phone calls
- products
- companies
- …
Graphs

Google, Facebook, Twitter, Amazon, Netflix: built *because, over, for, a graph*
Graphs

Google, Facebook, Twitter, Amazon, Netflix: built because, over, for, a graph

Graphs are beautiful mathematical objects, with many interesting properties.

Algorithms
Combinatorics
Randomness
Network science
Spectral analysis

Prof. Fan Chung, UCSD
A graph $G$ is a pair $(V, E)$ where

$V$ is a set of *vertices* (or *nodes*)

E.g., all computers on the Internet, every account on Instagram, all proteins in the body…
Graphs

A graph $G$ is a pair $(V, E)$ where

$V$ is a set of vertices (or nodes)

E.g., all computers on the Internet, every account on Instagram, all proteins in the body…

$E$ is a set of edges, which are pair of vertices:

Edge $(u, v)$, with $u, v \in V$.

Represents interaction between $u$ and $v$ (e.g., connection, likes, friendships, …)
A graph $G$ is a pair $(V, E)$ where

$V$ is a set of vertices (or nodes)

E.g., all computers on the Internet, every account on Instagram, all proteins in the body...

$E$ is a set of edges, which are pair of vertices:

Edge $(u, v)$, with $u, v \in V$.

Represents interaction between $u$ and $v$ (e.g., connection, likes, friendships, ...)

$V = \{1, 2, 3, 4, 5\}$

$E = \{(1, 2), (1, 3), (1, 5), (2, 5), (3, 4), (3, 5)\}$
Graphs

A graph $G$ is a pair $(V, E)$ where

$V$ is a set of vertices (or nodes)

E.g., all computers on the Internet, every account on Instagram, all proteins in the body...

$E$ is a set of edges, which are pair of vertices:

Edge $(u, v)$, with $u, v \in V$.

Represents interaction between $u$ and $v$ (e.g., connection, likes, friendships, …)

$V = \{1, 2, 3, 4, 5\}$

$E = \{(1, 2), (1, 3), (1, 5), (2, 5), (3, 4), (3, 5)\}$
Graphs

\[ V = \{1, 2, 3, 4, 5\} \]
\[ E = \{(1, 2), (1, 3), (1, 5), (2, 5), (3, 4), (3, 5)\} \]

\( E \subseteq V \times V \), but is \((u, v)\) the same edge as \((v, u)\)?
Graphs

\[ V = \{1, 2, 3, 4, 5\}\]
\[ E = \{(1, 2), (1, 3), (1, 5), (2, 5), (3, 4), (3, 5)\}\]

*Undirected* graph (edges are *symmetric* interactions)

E.g., Facebook friendships

\[ E \subseteq V \times V, \text{ but is } (u, v) \text{ the same edge as } (v, u) ? \]

It depends: the graph \( G \) can be *directed* or *undirected*

\[ V = \{1, 2, 3, 4, 5\}\]
\[ E = \{(1, 2), (2, 1), (1, 3), (1, 5), (2, 5), (3, 4), (3, 5)\}\]

*Directed* graph (edges are *asymmetric* interactions)

E.g., Instagram/Twitter follows
What is the shortest path from vertex $a$ to vertex $b$?
What is the *shortest path* from vertex $a$ to vertex $b$?

What is the *minimum number of edges* to remove to split the graph in two parts? (*min-cut* problem)
What is the *shortest path* from vertex $a$ to vertex $b$?

What is the *minimum number of edges* to remove to split the graph in two parts? (*min-cut* problem)

What are the *important vertices*? (PageRank, centrality measures)
(Few) Important problems on graphs

What is the *shortest path* from vertex $a$ to vertex $b$?

What is the *minimum number of edges* to remove to split the graph in two parts? *(min-cut problem)*

What are the *important vertices*? (PageRank, centrality measures)

Do graphs representing computer networks have *different properties* than graphs representing road networks?
(Few) Important problems on graphs

What is the *shortest path* from vertex $a$ to vertex $b$?

What is the *minimum number of edges* to remove to split the graph in two parts? (*min-cut* problem)

What are the *important vertices*? (*PageRank*, centrality measures)

Do graphs representing computer networks have *different properties* than graphs representing road networks?

Can we find *communities* in a social network?
Outline

✓ Graphs: basic definitions

Triangles: what and why

Approximate counting of triangles in edge data streams

Exact counting of triangles in a static graph

Approximate counting of triangles in a static graph

Counting triangles in MapReduce
Section outline

Triangles: definition and why counting them
Section outline

Triangles: definition and why counting them

Edge streams: insertions, deletions, and computational resources
Section outline

Triangles: definition and why counting them

Edge streams: insertions, deletions, and computational resources

TRIÈST: approximate counting of triangles in edge streams, with *reservoir sampling*
Section outline

Triangles: definition and why counting them

Edge streams: insertions, deletions, and computational resources

TRIÈST: approximate counting of triangles in edge streams, with reservoir sampling

TRIÈST-IMPR: how a small change in the algorithm makes a big difference
What are triangles?

Let $G = (V, E)$ be an undirected graph.

**Triangle**: a set of *three edges* forming a cycle;

**Global triangle count** $\Delta_G$: the no. of triangles in $G$;

**Local triangle count** $\Delta_v$ for $v \in V$: the no. of triangles that $v$ “belongs” to;
What are triangles?

Let $G = (V, E)$ be an undirected graph.

**Triangle**: a set of *three edges* forming a cycle; E.g., $\{(1, 3), (3, 5), (5, 1)\}$

**Global triangle count** $\Delta_G$: the no. of triangles in $G$;

**Local triangle count** $\Delta_v$ for $v \in V$: the no. of triangles that $v$ “belongs” to;
What are triangles?

Let $G = (V, E)$ be an undirected graph.

**Triangle**: a set of *three edges* forming a cycle; E.g., $\{(1,3), (3,5), (5,1)\}$

**Global triangle count** $\Delta_G$: the no. of triangles in $G$; E.g., $\Delta_G = 3$;

**Local triangle count** $\Delta_v$ for $v \in V$: the no. of triangles that $v$ “belongs” to;
What are triangles?

Let $G = (V, E)$ be an undirected graph.

**Triangle:** a set of *three edges* forming a cycle; E.g., $\{(1, 3), (3, 5), (5, 1)\}$

**Global triangle count** $\Delta_G$: the no. of triangles in $G$; E.g., $\Delta_G = 3$;

**Local triangle count** $\Delta_v$ for $v \in V$: the no. of triangles that $v$ “belongs” to; E.g., $\Delta_1 = 2$, $\Delta_5 = 3$, $\Delta_6 = 0$, ...
Why and how should we count triangles?

APPLICATIONS: community/spam/event detection, link prediction/recommendation, prototype for more complex patterns, node classification, …
Why and how should we count triangles?

Applications: community/spam/event detection, link prediction/recommendation, prototype for more complex patterns, node classification, …

Algorithms

• *Exact* algorithm
  Takes time $\Theta \left( \frac{|E|^3}{2} \right) = \Theta(|V|^3)$;

• *Approximation* algorithms based on fixed probability edge sampling

• *MapReduce* algorithms;

They assume that the graph is *static* (does not change over time)
Why and how should we count triangles?

**Applications:** community/spam/event detection, link prediction/recommendation, prototype for more complex patterns, node classification, …

**Algorithms**

- *Exact* algorithm
  
  Takes time $\Theta \left( |E|^{3/2} \right) = \Theta(|V|^3)$;

- *Approximation* algorithms based on fixed probability edge sampling

- *MapReduce* algorithms;

They assume that the graph is *static* (does not change over time)

In many settings **$G$ changes over time**

  Edges and vertices are added/removed (e.g., friendships, phone calls)
What if the graph changes over time?

*Dynamic graph:* at each *time step* $t$, one and only one edge is *inserted or deleted*;

**Goal:** Keep track of $\Delta G$ and $\Delta v$, $v \in V$, at each time step;
What if the graph changes over time?

**Dynamic graph**: at each *time step* $t$, one and only one edge is *inserted or deleted*;

**Goal**: Keep track of $\Delta_G$ and $\Delta_v$, $v \in V$, at each time step;

**Questions**:  
- How do we *represent the graph evolution* over time? 
- What *computational resources* can we use?  
- What *operations* can we do?

edge stream model
Section outline

✓ Triangles: definition and why counting them

Edge streams: insertions, deletions, and computational resources

TRIÈST: approximate counting of triangles in edge streams, with reservoir sampling

TRIÈST-IMPR: how a small change in the algorithm makes a big difference
Edge streams

Let’s answer these questions

• How do we *represent the graph evolution* over time?
• What *computational resources* can we use?
• What *operations* can we do?
What are edge streams?

At each time step, a new edge update (insertion or deletion) is on the stream:

<table>
<thead>
<tr>
<th>Time</th>
<th>…</th>
<th>( t )</th>
<th>( t + 1 )</th>
<th>( t + 2 )</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream</td>
<td>…</td>
<td>(+, (1, 3))</td>
<td>(−, (3, 2))</td>
<td>(+, (1, 5))</td>
<td>…</td>
</tr>
</tbody>
</table>

\( G^{(t)} = (V^{(t)}, E^{(t)}) \): graph induced by the edges inserted and not deleted up to time \( t \).
What are edge streams?

At each time step, a new edge update (insertion or deletion) is on the stream:

<table>
<thead>
<tr>
<th>Time</th>
<th>...</th>
<th>( t )</th>
<th>( t + 1 )</th>
<th>( t + 2 )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream</td>
<td>...</td>
<td>+, ((1,3))</td>
<td>−, ((3,2))</td>
<td>+, ((1,5))</td>
<td>...</td>
</tr>
</tbody>
</table>

\( G^{(t)} = (V^{(t)}, E^{(t)}) \): graph \textit{induced} by the edges \textit{inserted and not deleted} up to time \( t \).

Example: Graph \( G^{(t-1)} \):

![Graph](image)
What are edge streams?

At each time step, a new *edge update* (*insertion* or *deletion*) is on the stream:

<table>
<thead>
<tr>
<th>Time</th>
<th>...</th>
<th>$t$</th>
<th>$t + 1$</th>
<th>$t + 2$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream</td>
<td>...</td>
<td>$+$, (1, 3)</td>
<td>$-$, (3, 2)</td>
<td>$+$, (1, 5)</td>
<td>...</td>
</tr>
</tbody>
</table>

$G^{(t)} = (V^{(t)}, E^{(t)})$: graph *induced* by the edges *inserted and not deleted* up to time $t$.

**Example:** Graph $G^{(t)}$:

![Graph with nodes 0, 1, 2, 3, 4 connected by edges](image)
What are edge streams?

At each time step, a new edge update (insertion or deletion) is on the stream:

| Time  | ... | $t$ | $t+1$ | $t+2$ | ...
|-------|-----|-----|-------|-------|-----
| Stream| ... | +, (1,3) | −, (3,2) | +, (1,5) | ...

$G^{(t)} = (V^{(t)}, E^{(t)})$: graph induced by the edges inserted and not deleted up to time $t$.

**Example:** Graph $G^{(t+1)}$:

```
0 -- 1 -- 4
|    |
3 -- 2
```
What are edge streams?

At each time step, a new **edge update** (insertion or deletion) is on the stream:

<table>
<thead>
<tr>
<th>Time</th>
<th>...</th>
<th>( t )</th>
<th>( t + 1 )</th>
<th>( t + 2 )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream</td>
<td>...</td>
<td>+, (1, 3)</td>
<td>−, (3, 2)</td>
<td>+, (1, 5)</td>
<td>...</td>
</tr>
</tbody>
</table>

\( G^{(t)} = (V^{(t)}, E^{(t)}) \): graph **induced** by the edges **inserted and not deleted** up to time \( t \).

**Example:** Graph \( G^{(t+2)} \):

![Graph](image)

The global and local triangle counts (may) change after each update;

**Our goal:** at each time \( t \), give estimates of \( \Delta_{G^{(t)}} \) and \( \Delta_v, v \in V^{(t)} \).
What resources do we have?

Our goal: at each time $t$, give an estimate of $\Delta G(t)$ and $\Delta v$, $v \in V(t)$. 
What resources do we have?

**Our goal:** *at each time* $t$, give an *estimate* of $\Delta_{G(t)}$ and $\Delta_v$, $v \in V(t)$.

“**Big Data**” problem: the *graph grows faster than the available storage*;

The stream is *infinite*, but we have *limited memory*:
- A space $S$ to store up to $M$ edges;
- Up to $M + 1$ counters/variables to use for estimating $\Delta_{G(t)}$ and $\Delta_{v(t)}$, $v \in V(t)$.
What operations can we do?

We have $S$ (to store up to $M$ edges), and $M + 1$ counters

At time $t$ we can:
- delete edges from $S$; and
What operations can we do?

We have $S$ (to store up to $M$ edges), and $M + 1$ counters

At time $t$ we can:

- delete edges from $S$; and
- insert $e_t$ into $S$ if there is room; and
What operations can we do?

We have $S$ (to store up to $M$ edges), and $M + 1$ counters

At time $t$ we can:
- delete edges from $S$; and
- insert $e_t$ into $S$ if there is room; and
- update the counters using information from $S$;
What operations can we do?

We have $S$ (to store up to $M$ edges), and $M + 1$ counters

At time $t$ we can:
- delete edges from $S$; and
- insert $e_t$ into $S$ if there is room; and
- update the counters using information from $S$;

The stream is *fast*, and we must act fast:
At time $t$, we must obtain estimations of $\Delta_{G(t)}$ and $\Delta_v$, $v \in V(t)$, in time $O(M)$. 
Section outline

- Triangles: definition and why counting them

- Edge streams: insertions, deletions, and computational resources

TRIÈST: approximate counting of triangles in edge streams, with *reservoir sampling*

TRIÈST-IMPR: how a small change in the algorithm makes a big difference
What is TRIÈST?

(The local dialect name of Trieste, a city in the North-East of Italy, next to Slovenia.)

TRIÈST (TRIangles ESTimation):

A suite of 3 algorithms for approximate triangle counting from edge streams:
What is TRIÈST?

(The local dialect name of Trieste, a city in the North-East of Italy, next to Slovenia.)

TRIÈST (TRIangles ESTimation):

A suite of 3 algorithms for approximate triangle counting from edge streams:

- TRIÈST-BASE: baseline algorithm for insertion-only streams;
- TRIÈST-IMPR: improved algorithm for insertion-only streams with
- (TRIÈST-FD: algorithm for fully-dynamic streams (insertions and deletions.))
What is TRIÈST?

(The local dialect name of *Trieste*, a city in the North-East of Italy, next to Slovenia.)

TRIÈST (*TRI*angles *EST*imation):

* A suite of 3 algorithms for approximate triangle counting from edge streams:
  * TRIÈST-BASE: *baseline* algorithm for *insertion-only streams*;
  * TRIÈST-IMPR: *improved* algorithm for insertion-only streams with
  * (TRIÈST-FD: algorithm for *fully-dynamic streams* (insertions and deletions.))

All three algorithms offer *unbiased estimates* of the local and global triangle counts;
What is the general idea behind TRIÈST?

Let’s focus on TRIÈST-BASE (i.e., *insertion-only* streams) and on $\Delta_G$;

The edges in $S$ at time $t$ induce a graph $G_S = (V_S, S)$;
What is the general idea behind TRIÈST?

Let’s focus on TRIÈST-BASE (i.e., *insertion-only* streams) and on $\Delta G$;

The edges in $S$ at time $t$ induce a graph $G_S = (V_S, S)$;

**Idea:** *Estimate* $\Delta G(t)$ *using* $\Delta G_S$, the number of triangles in $G_S$;

The estimate for $\Delta G(t)$ is obtained by *dividing* $\Delta G_S$ *by a weight* $\pi_t$ (stay tuned!)

TRIÈST-BASE uses the counters to keep the *exact value* of $\Delta G_S$ (and $\Delta v_S, v \in V_S$);

TRIÈST-BASE uses *reservoir sampling* to store a random sample of $E(t)$ in $S$;

Maintaining the exact value for $\Delta G_S$ after each update to $S$ is *fast*;
How does TRIÈST-BASE work?

Let $D$ be the counter to keep the value of $\Delta G_S$

At any time $t \leq M$, deterministically insert $e_t$ into $S$, and increase $D$ by the no. of triangles in $G_S$ involving $e_t$;

At any $t > M$, flip a coin with tail-bias $M/t$;

If the outcome is head, do nothing;

If the outcome is tail:

1) Choose an edge in $S$ u.a.r. and replace it with $e_t$

2) Decrease $D$ by the no. of triangles involving the removed edge;

3) Increase $D$ by the no. of triangles involving $e_t$;
**Example**

**Memory:** \( M = 8 \); **Time:** end of \( t - 1 \);

**Graph** \( G_S = (V_S, S) \):

![Graph](image)

**Global triangle count** \( D = \Delta_{G_S} : 3 \)
Example

**Memory:** \( M = 8; \) **Time:** \( t; \)

**Edge on the stream:** \((2, 5);\)

**Coin bias:** \( M/t; \) **Coin flip outcome:** tail;

**Graph** \( G_S = (V_S, S): \)

![Graph](image)

**Global triangle count** \( D = \Delta_{G_S}: 3\)
Example

Memory: $M = 8$; Time: $t$;

Edge on the stream: $(2, 5)$;

Coin bias: $M/t$; Coin flip outcome: tail;

Actions: 1) Remove an edge in $G_S$ at random (e.g., $(0, 1)$); 2) Add $(2, 5)$ to $G_S$.
3) Update $D$;

Graph $G_S = (V_S, S)$:

![Graph diagram]

Global triangle count $D = \Delta_{G_S} : 3$
Example

Memory: $M = 8$; Time: $t$;

Edge on the stream: $(2, 5)$;

Coin bias: $M/t$; Coin flip outcome: tail;

Actions: 1) **Remove** an edge in $G_S$ at random (e.g., $(0, 1)$); 2) **Add** $(2, 5)$ to $G_S$.
   3) **Update** $D$;

Graph $G_S = (V_S, S)$:

![Diagram of graph $G_S$](image)

**Global triangle count** $D = \Delta_{G_S}: 3$
Example

**Memory:** $M = 8$; **Time:** $t$;

**Edge on the stream:** $(2, 5)$;

**Coin bias:** $M/t$; **Coin flip outcome:** tail;

**Actions:**
1) *Remove* an edge in $G_S$ at random (e.g., $(0, 1)$);
2) *Add* $(2, 5)$ to $G_S$.
3) *Update* $D$;

**Graph** $G_S = (V_S, S)$:

![Graph GS](image)

**Global triangle count** $D = \Delta_{G_S}$: 3
Example

Memory: \( M = 8 \); Time: \( t \);

Edge on the stream: \( (2, 5) \);

Coin bias: \( M/t \); Coin flip outcome: tail;

Actions: 1) Remove an edge in \( G_S \) at random (e.g., \( (0, 1) \)); 2) Add \( (2, 5) \) to \( G_S \).
3) Update \( D \);

Graph \( G_S = (V_S, S) \):

![Graph](image)

Global triangle count \( D = \Delta_{G_S} \): \( 3 - 1 + 1 = 3 \)
Example

Memory: \( M = 8 \); Time: \( t + 1 \);

Edge on the stream: \( (2, 4) \);

Coin bias: \( M/(t + 1) \); Coin flip outcome:

Actions:

Graph \( G_S = (V_S, S) \):

Global triangle count \( D = \Delta_{G_S} : 3 \)
Example

**Memory:** \( M = 8 \); **Time:** \( t + 1 \);

**Edge on the stream:** \((2, 4)\);

**Coin bias:** \( M / (t + 1) \); **Coin flip outcome:** head;

**Actions:** Do *nothing*;

**Graph** \( G_S = (V_S, S) \):

![Graph Diagram]

**Global triangle count** \( D = \Delta_{G_S} \): 3
How does TRIÈST-BASE estimate the number of triangles?

For $t < M$, the estimate is $D$.

It is an “exact estimate”: $D = \Delta_{GS} = \Delta_{G(t)}$
How does TRIÈST-BASE estimate the number of triangles?

For \( t < M \), the estimate is \( D \).

It is an “exact estimate”: \( D = \Delta_{GS} = \Delta_{G(t)} \)

For \( t \geq M \), let

\[
\pi_t = \frac{\binom{t-3}{M-3}}{\binom{t}{M}}.
\]

The estimate is \( D/\pi_t \), i.e., \( \Delta_{GS}/\pi_t \);
How does TRIÈST-BASE estimate the number of triangles?

For $t < M$, the estimate is $D$.

It is an “exact estimate”: $D = \Delta_G = \Delta_{G(t)}$

For $t \geq M$, let

$$\pi_t = \binom{t-3}{M-3} \binom{M}{t}$$

The estimate is $D/\pi_t$, i.e., $\Delta_G / \pi_t$;

We divide by $\pi_t$ to obtain unbiased estimates:

$$\mathbb{E} \left[ \frac{\Delta_G}{\pi_t} \right] = \Delta_{G(t)}$$

Unbiased estimates are good: TRIÈST does not make systematic errors.
How do we show that the estimations are unbiased?

**Theorem**

For any time \( t \geq M \), it holds

\[
E \left[ \frac{\Delta G_S}{\pi_t} \right] = \Delta G^{(t)}.
\]

We need the following lemma.

**Lemma**

For all \( t \geq M \), the set \( S^{(t)} \subseteq E^{(t)} \) is chosen uniformly at random among all subsets \( A \) of \( E^{(t)} \) of size \( |A| = M \):

\[
\Pr \left( S^{(t)} = A \right) = \frac{1}{\binom{t}{M}}, \text{ for all } A \subseteq E^{(t)}, |A| = M.
\]

We have seen the proof in class, under a different form (reservoir sampling)!
What is the probability of a triangle to be in $S$?

**Lemma**

The set $S(t) \subseteq E^{(t)}$ is chosen uniformly at random among all subsets $A$ of $E^{(t)}$ of size $|A| = M$:

$$\Pr \left( S(t) = A \right) = \frac{1}{\binom{t}{M}}, \text{ for all } A \subseteq E^{(t)}, |A| = M.$$  

**Corollary**

The probability that a triangle $(a, b, c)$ of $G^{(t)}$ is in $G_S$ at time $t$ is

$$\pi_t = \frac{\binom{t-3}{M-3}}{\binom{t}{M}}.$$
What is the probability of a triangle to be in $S$?

**Lemma**

The set $S^{(t)} \subseteq E^{(t)}$ is chosen uniformly at random among all subsets $A$ of $E^{(t)}$ of size $|A| = M$:

$$\Pr(S^{(t)} = A) = \frac{1}{\binom{t}{M}}, \text{ for all } A \subseteq E^{(t)}, |A| = M.$$

**Corollary**

The probability that a triangle $(a, b, c)$ of $G^{(t)}$ is in $G_S$ at time $t$ is

$$\pi_t = \frac{\binom{t-3}{M-3}}{\binom{t}{M}},$$

because $\binom{t-3}{M-3}$ is the number of $M$-subsets of $E^{(t)}$ containing $(a, b, c)$. 

27
Can we now prove the unbiasedness?

We want to show that, for any time \( t \geq M \), it holds

\[
\mathbb{E} \left[ \frac{\Delta G_S}{\pi_t} \right] = \Delta G(t) .
\]
Can we now prove the unbiasedness?

We want to show that, for any time \( t \geq M \), it holds

\[
\mathbb{E} \left[ \frac{\Delta G_S}{\pi_t} \right] = \Delta G(t) .
\]

For each triangle \( Z = (a, b, c) \) in \( G(t) \), let

\[
X_Z = \begin{cases} 
1 & \text{if } Z \text{ is in } G_S \text{ at time } t \\
0 & \text{otherwise.}
\end{cases}
\]

Then we can write

\[
\frac{\Delta G_S}{\pi_t} = \frac{1}{\pi_t} \sum_{Z \in G(t)} X_Z ,
\]
Can we now prove the unbiasedness?

We want to show that, for any time $t \geq M$, it holds

$$\mathbb{E} \left[ \frac{\Delta G_S}{\pi_t} \right] = \Delta G(t) \cdot$$

For each triangle $Z = (a, b, c)$ in $G(t)$, let

$$X_Z = \begin{cases} 
1 & \text{if } Z \text{ is in } G_S \text{ at time } t \\
0 & \text{otherwise.} 
\end{cases}$$

Then we can write

$$\frac{\Delta G_S}{\pi_t} = \frac{1}{\pi_t} \sum_{Z \in G(t)} X_Z,$$

and we have

$$\mathbb{E} \left[ \frac{\Delta G_S}{\pi_t} \right] = \mathbb{E} \left[ \frac{1}{\pi_t} \sum_{Z \in G(t)} X_Z \right].$$
Can we now prove the unbiasedness?

We want to show that, for any time \( t \geq M \), it holds

\[
\mathbb{E} \left[ \frac{\Delta G_S}{\pi_t} \right] = \Delta_{G(t)} .
\]

For each triangle \( Z = (a, b, c) \) in \( G(t) \), let

\[
X_Z = \begin{cases} 
1 & \text{if } Z \text{ is in } G_S \text{ at time } t \\
0 & \text{otherwise}.
\end{cases}
\]

Then we can write

\[
\frac{\Delta G_S}{\pi_t} = \frac{1}{\pi_t} \sum_{Z \in G(t)} X_Z ,
\]

and we have

\[
\mathbb{E} \left[ \frac{\Delta G_S}{\pi_t} \right] = \mathbb{E} \left[ \frac{1}{\pi_t} \sum_{Z \in G(t)} X_Z \right] = (\text{linearity of expectation}) = \frac{1}{\pi_t} \sum_{Z \in G(t)} \mathbb{E}[X_Z] .
\]
From the Corollary we have

\[ \Pr(X_Z = 1) = \Pr(Z \in G_S) = \pi_t \quad \text{and} \quad \Pr(X_Z = 0) = 1 - \pi_t . \]

Thus,

\[ \mathbb{E}[X_Z] = 1 \cdot \Pr(X_Z = 1) + 0 \cdot \Pr(X_Z = 0) = \pi_t . \]
From the Corollary we have

\[ \Pr(X_Z = 1) = \Pr(Z \in G_S) = \pi_t \quad \text{and} \quad \Pr(X_Z = 0) = 1 - \pi_t . \]

Thus,

\[ \mathbb{E}[X_Z] = 1 \cdot \Pr(X_Z = 1) + 0 \cdot \Pr(X_Z = 0) = \pi_t . \]

So,

\[ \mathbb{E}\left[ \frac{\Delta_{G_S}}{\pi_t} \right] = \frac{1}{\pi_t} \sum_{Z \in G(t)} \mathbb{E}[X_Z] = \frac{1}{\pi_t} \sum_{Z \in G(t)} \pi_t = \sum_{Z \in G(t)} 1 = \Delta_{G(t)} . \]
Section outline

✓ Triangles: definition and why counting them

Edge streams: insertions, deletions, and computational resources

✓ TRIÈST: approximate counting of triangles in edge streams, with *reservoir sampling*

TRIÈST-IMPR: how a small change in the algorithm makes a big difference
What is wrong with TRIÈST-BASE?

1) -BASE uses the *exact* value of $\Delta G_S$ at time $t$ to estimate $\Delta G(t)$;

$\Delta G_S$ may decrease...while $\Delta G(t') \geq \Delta G(t)$ for any $t' > t$!
What is wrong with TRIÈST-BASE?

1) -BASE uses the *exact* value of $\Delta G_S$ at time $t$ to estimate $\Delta G(t)$;

$\Delta G_S$ may decrease... while $\Delta G(t') \geq \Delta G(t)$ for any $t' > t$!

Solution: *never decrease the estimate*, i.e., use $G_S$ only to identify *new* triangles;
What is wrong with TRIÈST-BASE?

2) -BASE only counts a triangle if *all three* edges are in \( S \)… but if *two* edges are in \( S \), and the *third one is on the stream* right now, we may *infer* that the triangle exists, so we should *count it*;

**Solution:** *first increment* the counter \( D \), *then decide* whether to insert the edge into \( S \);
When edge $e_t$ is on the stream:

1. Count the number $g$ of triangles in the graph induced by $S \cup \{e_t\}$
When edge $e_t$ is on the stream:

1. Count the number $g$ of triangles in the graph induced by $S \cup \{e_t\}$

2. Let $\eta_t = \frac{(t-1)(t-2)}{M(M-1)}$, and increment $D$ by $g\eta_t$ (i.e., $D \leftarrow D + g\eta$)
When edge $e_t$ is on the stream:

1. Count the number $g$ of triangles in the graph induced by $S \cup \{e_t\}$
2. Let $\eta_t = \frac{(t-1)(t-2)}{M(M-1)}$, and increment $D$ by $g\eta_t$ (i.e., $D \leftarrow D + g\eta_t$)
3. Decide whether to update $S$ with $e_t$
When edge $e_t$ is on the stream:

1. Count the number $g$ of triangles in the graph induced by $S \cup \{e_t\}$
2. Let $\eta_t = \frac{(t-1)(t-2)}{M(M-1)}$, and increment $D$ by $g\eta_t$ (i.e., $D \leftarrow D + g\eta$)
3. Decide whether to update $S$ with $e_t$

-IMPR estimation of $\Delta_{G(t)}$ is $D$
When edge $e_t$ is on the stream:

1. Count the number $g$ of triangles in the graph induced by $S \cup \{e_t\}$
2. Let $\eta_t = \frac{(t-1)(t-2)}{M(M-1)}$, and increment $D$ by $g\eta_t$ (i.e., $D \leftarrow D + g\eta$)
3. Decide whether to update $S$ with $e_t$

-IMPR estimation of $\Delta_{G(t)}$ is $D$

In -BASE:

3 came before 1

The estimation was $D/\pi_t$ (in -IMPR, there’s no $\pi_t$, and $\eta_t$’s role is different)
When edge $e_t$ is on the stream:

1. Count the number $g$ of triangles in the graph induced by $S \cup \{e_t\}$

2. Let $\eta_t = \frac{(t-1)(t-2)}{M(M-1)}$, and increment $D$ by $g\eta_t$ (i.e., $D \leftarrow D + g\eta$)

3. Decide whether to update $S$ with $e_t$

-IMPR estimation of $\Delta_{G(t)}$ is $D$

In -BASE:

3 came before 1

The estimation was $D/\pi_t$ (in -IMPR, there’s no $\pi_t$, and $\eta_t$’s role is different) These small changes make -IMPR many times more accurate than -BASE

The analysis is much more complicated, but worth the effort!
Example

Memory: \( M = 8 \); Time: end of \( t - 1 \);

Graph \( G_S = (V_S, S) \):

Global triangle count \( D : x \)
**Example**

**Memory:** $M = 8$; **Time:** $t$;

**Edge on the stream:** $(2, 5)$;

**Actions:**

**Graph** $G_S = (V_S, S)$:

![Graph Diagram]

**Global triangle count** $D$: $x$
Example

Memory: $M = 8$; Time: $t$;

Edge on the stream: $(2, 5)$;

Actions: 1) Compute $g (=1)$; 2) Increment $D$;

Graph $G_S = (V_S, S)$:

Global triangle count $D: x + \eta_t$
Example

**Memory:** $M = 8$; **Time:** $t$;

**Edge on the stream:** $(2, 5)$;

**Actions:** 1) Compute $g$ (=1); 2) Increment $D$; 3) Flip the coin;

**Coin bias:** $M/t$; **Coin flip outcome:** tail;

**Graph** $G_S = (V_S, S)$:

![Graph](image)

**Global triangle count** $D$: $x + \eta_t$
Example

**Memory:** $M = 8$; **Time:** $t$;

**Edge on the stream:** $(2, 5)$;

**Actions:** 1) Compute $g (=1)$; 2) Increment $D$; 3) Flip the coin;
   4) *Remove* an edge in $G_S$ at random (e.g., $(0, 1)$); 5) *Add* $(2, 5)$ to $G_S$.

**Coin bias:** $M/t$; **Coin flip outcome:** *tail*;

**Graph** $G_S = (V_S, S)$:

![Graph GS]

**Global triangle count** $D$: $x + \eta_t$
**Example**

**Memory:** $M = 8$; **Time:** $t$;

**Edge on the stream:** $(2, 5)$;

**Actions:**
1) Compute $g$ (=1);
2) Increment $D$;
3) Flip the coin;
4) **Remove** an edge in $G_S$ at random (e.g., $(0, 1)$); 
5) **Add** $(2, 5)$ to $G_S$.

**Coin bias:** $M/t$; **Coin flip outcome:** tail;

**Graph** $G_S = (V_S, S)$:

![Graph Diagram]

**Global triangle count** $D$: $x + \eta_t$
Example

Memory: \( M = 8 \); Time: \( t + 1 \);

Edge on the stream: \((2, 4)\);

Actions:

Coin bias: Coin flip outcome:

Graph \( G_S = (V_S, S) \):

Global triangle count \( D: x + \eta_t \)
Example

**Memory:** $M = 8$; **Time:** $t + 1$;

**Edge on the stream:** $(2, 4)$;

**Actions:**

**Coin bias:** **Coin flip outcome:**

**Graph** $G_S = (V_S, S)$:

*Global triangle count* $D: x + \eta_t$
Example

Memory: $M = 8$; Time: $t + 1$;

Edge on the stream: $(2, 4)$;

Actions:
   1) Compute $g (-2)$; 2) Increment $D$;

Coin bias: Coin flip outcome:

Graph $G_S = (V_S, S)$:

Global triangle count $D$: $x + \eta_t + 2\eta_{t+1}$
Example

Memory: $M = 8$; Time: $t + 1$;

Edge on the stream: $(2, 4)$;

Actions:
1) Compute $g (-2)$; 2) Increment $D$; 3) Flip the coin;

Coin bias: $M/(t + 1)$; Coin flip outcome: head;

Graph $G_S = (V_S, S)$:

Global triangle count $D$: $x + \eta_t + 2\eta_{t+1}$
**Example**

**Memory:** $M = 8$; **Time:** $t + 1$;

**Edge on the stream:** $(2, 4)$;

**Actions:**
1) Compute $g = 2$; 2) Increment $D$; 3) Flip the coin; 4) Do *nothing* else;

**Coin bias:** $M/(t + 1)$; **Coin flip outcome:** head;

**Graph** $G_S = (V_S, S)$:

![Graph](image)

**Global triangle count** $D$: $x + \eta_t + 2\eta_{t+1}$
Outline

✓ **Graphs**: basic definitions

✓ **Triangles**: what and why

✓ **Approximate** counting of triangles in *edge data streams*

*Exact* counting of triangles in a *static graph*

*Approximate* counting of triangles in a static graph

Counting triangles in *MapReduce*
Naïve algorithm taking time $O(|V|^3)$
Naïve algorithm taking time $O(|V|^3)$

“Smart” algorithm taking time $O(|E|^{3/2})$
Naïve algorithm taking time $O(|V|^3)$

“Smart” algorithm taking time $O(|E|^{3/2})$

Sampling-based approximation algorithm
Assumptions and Notation

The vertices are numbered arbitrarily from 1 to $n = |V|$

There are $m$ edges in the graph, i.e., $|E| = m$

Hash table to check, given $u, v \in V$, whether $(u, v) \in E$ in (expected) constant time

Building this table takes time $O(m)$
Naïve algorithm

A triangle involves three vertices (a *triplet*).

**Idea:** For *every triplet* of vertices, check if they form a triangle.

**Input:** a graph $G = (V, E)$  
**Output:** $\Delta_G$, the number of triangles in $G$. 

Naïve algorithm

```latex
\begin{align*}
q & \leftarrow 0 \\
\text{For each node } h & \text{ from } 1 \text{ to } n; \text{ do} \\
\quad & \text{For each node } i \text{ from } 1 \text{ to } n; \text{ do} \\
\quad & \quad \text{For each node } j \text{ from } 1 \text{ to } n; \text{ do} \\
\quad & \quad \quad \text{If } (h, i) \in E \text{ AND } (i, j) \in E \text{ AND } (j, h) \in E; \text{ then} \\
\quad & \quad \quad \quad d \leftarrow q + 1 \\
\text{Return } q/6 \\
\end{align*}
```

This algorithm takes time $O(|V|^3)$ (using the hash table to check for edges).
Naïve algorithm

A triangle involves three vertices (a \textit{triplet}).

\textbf{Idea:} For \textit{every triplet} of vertices, check if they form a triangle.

\textbf{Input:} a graph \( G = (V, E) \) \quad \textbf{Output:} \( \Delta_G \), the number of triangles in \( G \).

\( q \leftarrow 0 \)

For each node \( h \) from 1 to \( n \); do
  For each node \( i \) from 1 to \( n \); do
    For each node \( j \) from 1 to \( n \); do
      If \((h, i) \in E \) AND \((i, j) \in E \) AND \((j, h) \in E \); then \( d \leftarrow q + 1 \)

Return \( q/6 \)

Divide by 6 because we are checking each triplet six times.

This algorithm takes time \( O(|V|^3) \) (using the hash table to check for edges).
Naïve algorithm

A triangle involves three vertices (a \textit{triplet}).

\textbf{Idea:} For \textit{every triplet} of vertices, check if they form a triangle.

\textbf{Input:} a graph $G = (V, E)$ \hspace{1cm} \textbf{Output:} $\Delta_G$, the number of triangles in $G$.

$q \leftarrow 0$

For each node $h$ from 1 to $n$; do
  For each node $i$ from 1 to $n$; do
    For each node $j$ from 1 to $n$; do
      If $(h, i) \in E$ AND $(i, j) \in E$ AND $(j, h) \in E$; then $d \leftarrow q + 1$

Return $q/6$

Divide by 6 because we are \textit{checking each triplet six times}
Naïve algorithm

A triangle involves three vertices (a \textit{triplet}).

\textbf{IDEA:} For \textit{every triplet} of vertices, check if they form a triangle.

\textbf{INPUT:} a graph $G = (V, E)$ \hspace{1cm} \textbf{OUTPUT:} $\Delta_G$, the number of triangles in $G$

$q \leftarrow 0$

For each node $h$ from 1 to $n$; do

\hspace{1cm} For each node $i$ from 1 to $n$; do

\hspace{2cm} For each node $j$ from 1 to $n$; do

\hspace{3cm} If $(h, i) \in E$ AND $(i, j) \in E$ AND $(j, h) \in E$; then $d \leftarrow q + 1$

Return $q/6$

Divide by 6 because we are \textit{checking each triplet six times}

This algorithm takes time $O(|V|^3)$ (using the hash table to check for edges)
Can we be a little less naïve?

$q \leftarrow 0$

For each node $h$ from 1 to $n$; do
  
  For each node $i$ from 1 to $n$; do
    
    For each node $j$ from 1 to $n$; do
      
      If $(h, i) \in E$ AND $(i, j) \in E$ AND $(j, h) \in E$; then $q \leftarrow q + 1$

Return $q/6$

How can we avoid checking each triplet 6 times?
Can we be a little less naïve?

\[ q \leftarrow 0 \]
For each node \( h \) from 1 to \( n \); do
   For each node \( i \) from 1 to \( n \); do
      For each node \( j \) from 1 to \( n \); do
         If \( (h, i) \in E \) AND \( (i, j) \in E \) AND \( (j, h) \in E \); then \( q \leftarrow q + 1 \)
Return \( q/6 \)

How can we avoid checking each triplet 6 times?
   
   By ordering the triplets: check \( (h, i, j) \) only if \( h < i < j \).
Can we be a little less naïve?

\[
q \leftarrow 0
\]
For each node \( h \) from 1 to \( n \); do
  For each node \( i \) from 1 to \( n \); do
    For each node \( j \) from 1 to \( n \); do
      If \((h, i) \in E \) AND \((i, j) \in E \) AND \((j, h) \in E \); then \( q \leftarrow q + 1 \)
Return \( q/6 \)

How can we avoid checking each triplet 6 times?
  By \textit{ordering the triplets}: check \((h, i, j)\) only if \( h < i < j \).

\[
q \leftarrow 0
\]
For each node \( h \) from 1 to \( n \); do
  For each node \( i \) from \( h + 1 \) to \( n \); do
    For each node \( j \) from \( i + 1 \) to \( n \); do
      If \((h, i) \in E \) AND \((i, j) \in E \) AND \((j, h) \in E \); then \( q \leftarrow q + 1 \)
Return \( q \)
Moving code around could be important

\[ q \leftarrow 0 \]

For each node \( h \) from 1 to \( n \); do
  For each node \( i \) from \( h + 1 \) to \( n \); do
    For each node \( j \) from \( i + 1 \) to \( n \); do
      If \((h, i) \in E \) AND \((i, j) \in E \) AND \((j, h) \in E \); then \( q \leftarrow q + 1 \)

Return \( q \)

For every pair \((h, i)\), we are checking \textit{all triplets} \((h, i, j)\). Can we avoid it?
Moving code around could be important

\[ q \leftarrow 0 \]

For each node \( h \) from 1 to \( n \); do
  For each node \( i \) from \( h + 1 \) to \( n \); do
    For each node \( j \) from \( i + 1 \) to \( n \); do
      If \((h, i) \in E \text{ AND } (i, j) \in E \text{ AND } (j, h) \in E\); then \( q \leftarrow q + 1 \)

Return \( q \)

For every pair \((h, i)\), we are checking \textit{all triplets} \((h, i, j)\). Can we avoid it?

\[ q \leftarrow 0 \]

For each node \( h \) from 1 to \( n \); do
  For each node \( i \) from \( h + 1 \) to \( n \); do
    If \((h, i) \in E\)
      For each node \( j \) from \( i + 1 \) to \( n \); do
        If \((i, j) \in E \text{ AND } (j, h) \in E\); then \( q \leftarrow q + 1 \)

Return \( q \)
Takeaway message

All these algorithms take time $O(|V|^3)$

In practice they take wildly different times
   (try: they are trivial to implement)
All these algorithms take time $O(|V|^3)$

In practice they take wildly different times
   (try: they are trivial to implement)

When analyzing massive data, *algorithm engineering* becomes important.
All these algorithms take time $O(|V|^3)$

In practice they take wildly different times
(try: they are trivial to implement)

When analyzing massive data, *algorithm engineering* becomes important.

*Don’t believe the pseudocode* of an algorithm
It conveys the *main ideas*, and is sufficient for the *worst-case* time complexity
Refining the ideas and implementing the algorithm *efficiently* could be quite different
Naïve algorithm taking time $O(|V|^3)$

“Smart” algorithm taking time $O(|E|^{3/2})$

Sampling-based approximation algorithm
Vertex degree

For any vertex \( v \), the *degree of* \( v \), denoted as \( d(v) \) is the number of edges that have \( v \) as one of their *endpoints*.

\[
d(v) = |\{(v, z) \in E\}|
\]

**Example**

\[d(5) = 3, \ d(4) = 1, \ldots\]
Vertex degree

For any vertex $v$, the *degree of $v$*, denoted as $d(v)$ is the number of edges that have $v$ as one of their *endpoints*.

$$d(v) = |\{(v, z) \in E\}|$$

**Example**

$$d(5) = 3, \ d(4) = 1, \ldots$$

Fact: $\sum_{v \in V} d(v) =$
Vertex degree

For any vertex $v$, the *degree of $v$*, denoted as $d(v)$ is the number of edges that have $v$ as one of their *endpoints*.

$$d(v) = |\{(v, z) \in E\}|$$

**Example**

![Graph Diagram]

$d(5) = 3$, $d(4) = 1$, ...

Fact: $\sum_{v \in V} d(v) = 2m$
An order to the nodes

We say that $v \prec u$ if and only if either
\[ d(v) < d(u), \text{ or } \]
\[ d(v) = d(u) \text{ AND } v < u. \]

**Example**

$4 \prec 3$, $1 \prec 5$
A vertex $v$ is a heavy hitter if $d(v) = \sqrt{m}$.
A vertex $v$ is a heavy hitter if $d(v) = \sqrt{m}$

How many heavy hitters can there be, at most?
A vertex $v$ is a **heavy hitter** if $d(v) = \sqrt{m}$

How many heavy hitters can there be, at most? At most $\min\{2\sqrt{m}, n\}$
A vertex $v$ is a *heavy hitter* if $d(v) = \sqrt{m}$

How many heavy hitters can there be, at most? At most $\min\{2\sqrt{m}, n\}$

The algorithm will first count triangles whose vertices are all heavy-hitters, and then count the other triangles.
Algorithm

1. Pre-processing
   1.1. Degree counting (to find heavy hitters)
   1.2. Neighbor index construction
Algorithm

1. Pre-processing
   1.1. Degree counting (to find heavy hitters)
   1.2. Neighbor index construction

2. Find heavy-hitter triangles
Algorithm

1. Pre-processing
   1.1. Degree counting (to find heavy hitters)
   1.2. Neighbor index construction

2. Find heavy-hitter triangles

3. Find other triangles
Preprocessing

1.1. Compute $d(v)$ for each $v \in V$ and find heavy-hitters

1.2 Build a *hash table* that, given $v \in V$, gives us the *neighbors* of $v$

(neighbors of $v$: $\{u \in V : (u, v) \in E\}$)
Preprocessing

1.1. Compute $d(v)$ for each $v \in V$ and find heavy-hitters

1.2 Build a hash table that, given $v \in V$, gives us the neighbors of $v$

(neighbors of $v$: $\{u \in V : (u, v) \in E\}$)

How:

- $d_v \leftarrow 0$ for each $v \in V$  // degree counters
- $H \leftarrow$ empty set  // set of the heavy-hitters
- $Z \leftarrow$ empty hash table  // $Z[u]$ will be the set of neighbors of $u$
Preprocessing

1.1. Compute $d(v)$ for each $v \in V$ and find heavy-hitters

1.2 Build a hash table that, given $v \in V$, gives us the neighbors of $v$

(neighbors of $v$: $\{u \in V : (u,v) \in E\}$)

How:

\begin{align*}
d_v &\leftarrow 0 \text{ for each } v \in V \quad \text{// degree counters} \\
H &\leftarrow \text{empty set} \quad \text{// set of the heavy-hitters} \\
Z &\leftarrow \text{empty hash table} \quad \text{// } Z[u] \text{ will be the set of neighbors of } u
\end{align*}

For each edge $(u,v) \in V$, do

\begin{align*}
d_u &\leftarrow d_u + 1, \quad d_v \leftarrow d_v + 1 \quad \text{// Increment degree counters for } u \text{ and } v
\end{align*}
Preprocessing

1.1. Compute $d(v)$ for each $v \in V$ and find heavy-hitters

1.2 Build a hash table that, given $v \in V$, gives us the neighbors of $v$

(neighbors of $v$: $\{u \in V : (u, v) \in E\}$)

How:

- $d_v \leftarrow 0$ for each $v \in V$ // degree counters
- $H \leftarrow$ empty set // set of the heavy-hitters
- $Z \leftarrow$ empty hash table // $Z[u]$ will be the set of neighbors of $u$

For each edge $(u, v) \in V$, do

- $d_u \leftarrow d_u + 1$, $d_v \leftarrow d_v + 1$ // Increment degree counters for $u$ and $v$
- If $d_u = \sqrt{m}$, then $H \leftarrow H \cup \{u\}$ // $u$ is a heavy-hitter, so add it to $H$
- If $d_v = \sqrt{m}$, then $H \leftarrow H \cup \{v\}$ // $v$ is a heavy-hitter, so add it to $H$
1.1. Compute $d(v)$ for each $v \in V$ and find heavy-hitters.

1.2 Build a hash table that, given $v \in V$, gives us the neighbors of $v$ (neighbors of $v$: $\{u \in V : (u, v) \in E\}$)

How:

- $d_v \leftarrow 0$ for each $v \in V$  // degree counters
- $H \leftarrow$ empty set  // set of the heavy-hitters
- $Z \leftarrow$ empty hash table  // $Z[u]$ will be the set of neighbors of $u$

For each edge $(u, v) \in V$, do

- $d_u \leftarrow d_u + 1, d_v \leftarrow d_v + 1$  // Increment degree counters for $u$ and $v$
- If $d_u = \sqrt{m}$, then $H \leftarrow H \cup \{u\}$  // $u$ is a heavy-hitter, so add it to $H$
- If $d_v = \sqrt{m}$, then $H \leftarrow H \cup \{v\}$  // $v$ is a heavy-hitter, so add it to $H$
- $Z[u] \leftarrow Z[u] \cup \{v\}, Z[v] \leftarrow Z[v] \cup \{u\}$  // Add $u$ and $v$ to their
  // respective sets of neighbors

Time: $O(m)$: we iterate over the $m$ edges, and each iteration takes constant time.
Preprocessing

1.1. Compute $d(v)$ for each $v \in V$ and find heavy-hitters
1.2 Build a hash table that, given $v \in V$, gives us the neighbors of $v$
(neighbors of $v$: $\{u \in V : (u, v) \in E\}$)

How:

\begin{align*}
  &d_v \leftarrow 0 \text{ for each } v \in V \quad \text{// degree counters} \\
  &H \leftarrow \text{empty set} \quad \text{// set of the heavy-hitters} \\
  &Z \leftarrow \text{empty hash table} \quad \text{// $Z[u]$ will be the set of neighbors of } u \\
  &\text{For each edge } (u, v) \in V, \text{ do} \\
  &\quad d_u \leftarrow d_u + 1, d_v \leftarrow d_v + 1 \quad \text{// Increment degree counters for } u \text{ and } v \\
  &\quad \text{If } d_u = \sqrt{m}, \text{ then } H \leftarrow H \cup \{u\} \quad \text{// } u \text{ is a heavy-hitter, so add it to } H \\
  &\quad \text{If } d_v = \sqrt{m}, \text{ then } H \leftarrow H \cup \{v\} \quad \text{// } v \text{ is a heavy-hitter, so add it to } H \\
  &\quad Z[u] \leftarrow Z[u] \cup \{v\}, Z[v] \leftarrow Z[v] \cup \{u\} \quad \text{// Add } u \text{ and } v \text{ to their} \\
  &\quad \quad \quad \text{// respective sets of neighbors} \\
\end{align*}

Time: $O(m)$: we iterate over the $m$ edges, and each iteration takes constant time.
Finding heavy-hitter triangles

Heavy-hitter triangle: a triangle with three heavy-hitter vertices

There are at most $O(\sqrt{m})$ heavy-hitter vertices, so at most $O(m^{3/2})$ heavy-hitter triangles.

Using $H$ and the naïve algorithm we can find all heavy-hitter triangles in time $O(m^{3/2})$.

This approach is essentially optimal: we need time $O(x)$ to count $O(x)$ objects!
Finding heavy-hitter triangles

Heavy-hitter triangle: a triangle with three heavy-hitter vertices

How many heavy-hitter triangles there are at most?

There are at most $O(\sqrt{m})$ heavy-hitter vertices, so at most $O(m^{3/2})$ heavy-hitter triangles.

Using $H$ and the naïve algorithm we can find all heavy-hitter triangles in time $O(m^{3/2})$.

This approach is essentially optimal: we need time $O(x)$ to count $O(x)$ objects!
Finding heavy-hitter triangles

Heavy-hitter triangle: a triangle with *three heavy-hitter vertices*

How many heavy-hitter triangles there are at most?

There are at most $O(\sqrt{m})$ heavy-hitter vertices,
so *at most* $O(m^{3/2})$ *heavy-hitter triangles*
Finding heavy-hitter triangles

Heavy-hitter triangle: a triangle with *three heavy-hitter vertices*

How many heavy-hitter triangles there are at most?

There are at most $O(\sqrt{m})$ heavy-hitter vertices, so *at most $O(m^{3/2})$ heavy-hitter triangles*

Using $H$ and the naïve algorithm we can find all heavy-hitter triangles in time $O(m^{3/2})$.

This approach is essentially optimal: we need time $O(x)$ to count $O(x)$ objects!
Finding the other triangles

\[ q \leftarrow 0 \quad \text{// to count the triangles} \]

For each edge \( e = (v_1, v_2) \), do

\[ \text{If } v_1 \in H \text{ AND } v_2 \in H, \text{ then CONTINUE} \quad \text{// Skip heavy-hitter edges} \]
Finding the other triangles

\[ q \leftarrow 0 \] \quad // \text{to count the triangles}

For each edge \( e = (v_1, v_2) \), do

- If \( v_1 \in H \) AND \( v_2 \in H \), then CONTINUE \quad // \text{Skip heavy-hitter edges}

\[ q \leftarrow q + 1 \] \quad // \text{Found a triangle so increment counter}
Finding the other triangles

\[ q \leftarrow 0 \quad \text{// to count the triangles} \]

For each edge \( e = (v_1, v_2) \), do

- If \( v_1 \in H \) AND \( v_2 \in H \) then CONTINUE \quad \text{// Skip heavy-hitter edges}
- // Assume \( v_1 < v_2 \), which implies \( v_1 \notin H \)

\[ \{u_1, \ldots, u_k\} \leftarrow Z[v_1] \quad \text{// Neighbors of } v_1. \quad k < \sqrt{m} \text{ because } v_1 \notin H \]
Finding the other triangles

\[ q \leftarrow 0 \quad \text{// to count the triangles} \]

For each edge \( e = (v_1, v_2) \), do

If \( v_1 \in H \text{ AND } v_2 \in H \), then CONTINUE \text{// Skip heavy-hitter edges}

// Assume \( v_1 \prec v_2 \), which implies \( v_1 \not\in H \)

\[ \{u_1, \ldots, u_k\} \leftarrow Z[v_1] \quad \text{// Neighbors of } v_1. \quad k < \sqrt{m} \text{ because } v_1 \not\in H \]

For each \( u_i \), do \text{// Iterate over the neighbors of } v_1

If \( u_i \prec v_2 \), then CONTINUE \text{// To ensure a triplet is examined once}
Finding the other triangles

\( q \leftarrow 0 \) // to count the triangles

For each edge \( e = (v_1, v_2) \), do

If \( v_1 \in H \) AND \( v_2 \in H \), then CONTINUE // Skip heavy-hitter edges

// Assume \( v_1 \prec v_2 \), which implies \( v_1 \notin H \)

\( \{u_1, \ldots, u_k\} \leftarrow Z[v_1] \) // Neighbors of \( v_1 \). \( k < \sqrt{m} \) because \( v_1 \notin H \)

For each \( u_i \), do // Iterate over the neighbors of \( v_1 \)

If \( u_i \prec v_2 \), then CONTINUE // To ensure a triplet is examined once

If \( (u_i, v_2) \in E \), then \( q \leftarrow q + 1 \) // Found a triangle so increment counter
Finding the other triangles

\[ q \leftarrow 0 \quad \text{// to count the triangles} \]

For each edge \( e = (v_1, v_2) \), do

\[
\text{If } v_1 \in H \text{ AND } v_2 \in H, \text{ then CONTINUE } \quad \text{// Skip heavy-hitter edges} \\
\text{// Assume } v_1 \prec v_2, \text{ which implies } v_1 \not\in H
\]

\[
\{u_1, \ldots, u_k\} \leftarrow Z[v_1] \quad \text{// Neighbors of } v_1. \quad k < \sqrt{m} \text{ because } v_1 \not\in H
\]

For each \( u_i \), do

\[
\text{If } u_i \prec v_2, \text{ then CONTINUE } \quad \text{// To ensure a triplet is examined once} \\
\text{If } (u_i, v_2) \in E, \text{ then } q \leftarrow q + 1 \quad \text{// Found a triangle so increment counter}
\]

The running time is dominated by the two nested for-loops:

For each iteration of the \textit{external} for-loop, the \textit{internal} for-loop does \( O(\sqrt{m}) \) iterations, each taking \( O(1) \) time.
Finding the other triangles

\( q \leftarrow 0 \)  // to count the triangles

For each edge \( e = (v_1, v_2) \), do

If \( v_1 \in H \) AND \( v_2 \in H \), then CONTINUE  // Skip heavy-hitter edges

// Assume \( v_1 \prec v_2 \), which implies \( v_1 \notin H \)
\( \{u_1, \ldots, u_k\} \leftarrow Z[v_1] \)  // Neighbors of \( v_1 \). \( k < \sqrt{m} \) because \( v_1 \notin H \)

For each \( u_i \), do  // Iterate over the neighbors of \( v_1 \)

If \( u_i \prec v_2 \), then CONTINUE  // To ensure a triplet is examined once

If \( (u_i, v_2) \in E \), then \( q \leftarrow q + 1 \)  // Found a triangle so increment counter

The running time is dominated by the two nested for-loops:

For each iteration of the external for-loop, the internal for-loop does \( O(\sqrt{m}) \) iterations, each taking \( O(1) \) time.

The external for-loop does \( O(m) \) iterations.
Finding the other triangles

\[
q \leftarrow 0 \quad \text{// to count the triangles}
\]

For each edge \( e = (v_1, v_2) \), do

\[
\text{If } v_1 \in H \text{ AND } v_2 \in H, \text{ then CONTINUE} \quad \text{// Skip heavy-hitter edges}
\]

\[
\text{// Assume } v_1 \prec v_2, \text{ which implies } v_1 \notin H
\]

\[
\{u_1, \ldots, u_k\} \leftarrow Z[v_1] \quad \text{// Neighbors of } v_1. \quad k < \sqrt{m} \text{ because } v_1 \notin H
\]

For each \( u_i \), do \( \text{// Iterate over the neighbors of } v_1 \)

\[
\text{If } u_i \prec v_2, \text{ then CONTINUE} \quad \text{// To ensure a triplet is examined } \text{once}
\]

\[
\text{If } (u_i, v_2) \in E, \text{ then } q \leftarrow q + 1 \quad \text{// Found a triangle so increment counter}
\]

The running time is dominated by the two nested for-loops:

For each iteration of the \textit{external} for-loop, the \textit{internal} for-loop does \( O(\sqrt{m}) \) iterations, each taking \( O(1) \) time.

The external for-loop does \( O(m) \) iterations.

Runtime: \( O(m \cdot \sqrt{m}) = O(m^{3/2}) \)
Runtime recap

Pre-processing takes time $O(m)$

Finding the heavy-hitter triangles takes time $O(m^{3/2})$

Finding the other triangles takes time $O(m^{3/2})$

Total time: $O(m^{3/2})$
Naïve algorithm taking time $O(|V|^3)$

“Smart” algorithm taking time $O(|E|^{3/2})$

Sampling-based approximation algorithm
Sampling-based approximation algorithm

Idea: similar to streaming TRIÈST algorithm

Sample edges independently with probability $p$. 
Sampling-based approximation algorithm

**Idea:** similar to streaming TRIÈST algorithm

Sample edges independently with probability $p$.

Let $E_S$ be the collection of sampled edges.

Consider the graph $G_S = (V_S, E_S)$, where $V_S$ is the set of vertices that are extremes of at least one edge in $E_S$.

Use $\Delta_{G_S}/p^3$ as *estimate* for $\Delta_G$. 
Use $\Delta_{GS}/p^3$ as *estimate* for $\Delta_G$.

How large should $p$ be to have a good estimate?
Sampling-based approximation algorithm

Use $\Delta_{GS}/p^3$ as \textit{estimate} for $\Delta_G$.

How large should $p$ be to have a good estimate?

Depends on the maximum number $T$ of triangles in $G$ that \textit{share an edge}. 
Sampling-based approximation algorithm

Use $\Delta_{G_S}/p^3$ as estimate for $\Delta_G$.

How large should $p$ be to have a good estimate?

Depends on the maximum number $T$ of triangles in $G$ that share an edge.

If we don’t sample an edge shared by many triangles, our estimate is very bad.
Sampling-based approximation algorithm

Use $\Delta_{G_S}/p^3$ as *estimate* for $\Delta_G$.

How large should $p$ be to have a good estimate?

Depends on the maximum number $T$ of triangles in $G$ that *share an edge*.

If we don’t sample an edge shared by many triangles, our estimate is very bad.

This information is a bit disappointing: we do not know $T$ in advance!
Sampling-based approximation algorithm

Use $\Delta G_S / p^3$ as estimate for $\Delta G$.

How large should $p$ be to have a good estimate?

Depends on the maximum number $T$ of triangles in $G$ that share an edge.

If we don’t sample an edge shared by many triangles, our estimate is very bad.

This information is a bit disappointing: we do not know $T$ in advance!

Typical situation when analyzing algorithms on graphs:
    - the analysis depends on unknown values of properties of the graph.
Outline

✓ **Graphs**: basic definitions

✓ **Triangles**: what and why

✓ *Approximate* counting of triangles in *edge data streams*

✓ *Exact* counting of triangles in a *static graph*

✓ *Approximate* counting of triangles in a static graph

Counting triangles in *MapReduce*
Section Outline

(Natural Equi-)Joins in MapReduce

Multi-way Joins in MapReduce

Self-Joins

Triangle counting in MapReduce with a Multi-way Self-Join
Relations, Tuples, and Attributes

Fundamental concepts in databases.

A *relation* is a table containing rows of data (*tuples*).
The columns of the table are called *attributes*. The set of attributes is called the *schema*.

Example: A relation called ScoresHW

<table>
<thead>
<tr>
<th>Student</th>
<th>HW01</th>
<th>HW02</th>
<th>HW03</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>AE</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

We denote a relation $R$ with attributes $A_1, \ldots, A_k$ as $R(A_1, \ldots, A_k)$:

ScoresHW(Student,HW01,HW02,HW03)

We denote the value of a tuple $r$ in an attribute $A$ as $r.A$: MR.HW01=15.
A \textit{(natural equi-)join} is a fundamental operation in databases. It involves two relations $R$ and $S$ with \textit{at least one} attribute in common: E.g., $R(A, B)$ and $S(A, C)$. The result of the join is a relation $J(A, B, C)$.
(Natural Equi-)Joins

A *(natural equi-)*join is a fundamental operation in databases.

It involves two relations $R$ and $S$ with *at least one* attribute in common: E.g., $R(A, B)$ and $S(A, C)$.

The result of the join is a relation $J(A, B, C)$

Computing the join $J = R \bowtie S$:

$J = \emptyset$ // to store the result of the join

For each tuple $r \in R$

For each tuple $s \in S$

If $r.A == s.A$ then add the tuple $(r.A, r.B, s.C)$ to $J$. 


**Example of a join**

<table>
<thead>
<tr>
<th>Student</th>
<th>HW01</th>
<th>HW02</th>
<th>HW03</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>AE</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>45</td>
</tr>
<tr>
<td>AE</td>
<td>60</td>
</tr>
<tr>
<td>ZZ</td>
<td>55</td>
</tr>
</tbody>
</table>
Example of a join

ScoresHW

<table>
<thead>
<tr>
<th>Student</th>
<th>HW01</th>
<th>HW02</th>
<th>HW03</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>AE</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

ScoresFin

<table>
<thead>
<tr>
<th>Student</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>45</td>
</tr>
<tr>
<td>AE</td>
<td>60</td>
</tr>
<tr>
<td>ZZ</td>
<td>55</td>
</tr>
</tbody>
</table>

ScoresHW \(\bowtie\) ScoresFin:

<table>
<thead>
<tr>
<th>Student</th>
<th>HW01</th>
<th>HW02</th>
<th>HW03</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
<td>45</td>
</tr>
<tr>
<td>AE</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
<td>55</td>
</tr>
</tbody>
</table>
Join in MapReduce

We want to compute $R(A, B) \bowtie S(A, C)$.

Input: a tuple $t$ of $R$ is a pair $(R, (t.A, t.B))$, and similarly for $S$.
Output: a pair $(\ast, (t.A, t.B, s.C))$ for each $t \in R, s \in S$ s.t. $t.A == s.A$
Join in MapReduce

We want to compute \( R(A, B) \bowtie S(A, C) \).

Input: a tuple \( t \) of \( R \) is a pair \((R, (t.A, t.B))\), and similarly for \( S \).
Output: a pair \((\ast, (t.A, t.B, s.C))\) for each \( t \in R, s \in S \) s.t. \( t.A == s.A \)

map: Map \((X, (a, z))\) to \((a, (X, z))\)
Join in MapReduce

We want to compute $R(A, B) \bowtie S(A, C)$.

Input: a tuple $t$ of $R$ is a pair $(R, (t.A, t.B))$, and similarly for $S$.

Output: a pair $(\ast, (t.A, t.B, s.C))$ for each $t \in R$, $s \in S$ s.t. $t.A == s.A$

map: $\text{Map} \ (X, (a, z))$ to $(a, (X, z))$

reduce: Take $(a, ((R, b_1), \ldots, (R, b_\ell), (S, c_1), \ldots, (S, c_k)))$

and output $(\ast, (a, b_i, c_j))$ for each $1 \leq i \leq \ell$ and $1 \leq j \leq k$. 
Section Outline

✓ (Natural Equi-)Joins in MapReduce

✓ Multi-way Joins in MapReduce

Self-Joins

Triangle counting in MapReduce with a Multi-way Self-Join
Multi-way joins

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

We could join \( R \) with \( S \) first, and then join the result with \( T \).

It will require \textit{two} MapReduce jobs. We can do it in a single job.
Multi-way joins in MapReduce

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

Choose two values \( q_B \) and \( q_C \) of buckets

Pick two hash functions \( h_B \) and \( h_C \) from the domains of \( B \) and \( C \) respectively to \( q_B \) and \( q_C \) buckets respectively.

There will be \( q_B q_C \) reducers, each associated to a pair \((i, j)\) of buckets.

Reducer \((i, j)\) receives all tuples \( R(u, v), S(v, w), T(w, z) \) s.t. \( h_B(v) = i \) and \( h_C(w) = j \)
Multi-way joins in MapReduce

\( R(A, B) \bowtie S(B, C) \bowtie T(C, D) \)

Choose two values \( q_B \) and \( q_C \) of buckets

Pick two hash functions \( h_B \) and \( h_C \) from the domains of \( B \) and \( C \) respectively to \( q_B \) and \( q_C \) buckets respectively.

There will be \( q_B q_C \) reducers, each associated to a pair \((i, j)\) of buckets.
Reducer \((i, j)\) receives all tuples \( R(u, v), S(v, w), T(w, z) \) s.t. \( h_B(v) = i \) and \( h_C(w) = j \)

Map:
send tuple \( t \) to all reducers where it may find tuples from other relations to join with. \( t \) must have attribute \( B \) or \( C \) or both, so we can determine which reducers to send \( t \) to. Tuples of \( R \) and of \( T \) will be sent to multiple reducers.
Multi-way joins in MapReduce

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

Choose two values \( q_B \) and \( q_C \) of \textit{buckets}

Pick \textit{two} hash functions \( h_B \) and \( h_C \) from the domains of \( B \) and \( C \) respectively to \( q_B \) and \( q_C \) buckets respectively.

There will be \( q_B q_C \) reducers, each associated to a pair \((i, j)\) of buckets.

Reducer \((i, j)\) receives all tuples \( R(u, v), S(v, w), T(w, z) \) s.t. \( h_B(v) = i \) and \( h_C(w) = j \)

Map:
send tuple \( t \) to \textit{all} reducers where it may find tuples \textit{from other relations} to join with.
\( t \) \textit{must} have attribute \( B \) or \( C \) or both, so we can determine which reducers to send \( t \) to.
Tuples of \( R \) and of \( T \) will be sent to \textit{multiple} reducers.

Reduce: compute the joined tuples using the received tuples.
Multi-way joins in MapReduce

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

Reducer \((i, j)\) receives all tuples \(R(u, v), S(v, w), T(w, z)\) s.t. \(h_B(v) = i\) and \(h_C(w) = j\)
Multi-way joins in MapReduce

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

Reducer \((i, j)\) receives all tuples \(R(u, v), S(v, w), T(w, z)\) s.t. \(h_B(v) = i\) and \(h_C(w) = j\)

Input: tuple \((X, (u, v))\), where \(X\) is \(R\), \(S\), or \(T\), and \(u, v\) are the values in the attributes.
Multi-way joins in MapReduce

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

Reducer \((i, j)\) receives all tuples \(R(u, v), S(v, w), T(w, z)\) s.t. \(h_B(v) = i\) and \(h_C(w) = j\)

Input: tuple \((X, (u, v))\), where \(X\) is \(R\), \(S\), or \(T\), and \(u, v\) are the values in the attributes.

map:

Map \((S, (v, w))\) to \(((h_B(v), h_C(w)), (S, (v, w)))\)
Multi-way joins in MapReduce

\[ R(A, B) \Join S(B, C) \Join T(C, D) \]

Reducer \((i, j)\) receives all tuples \(R(u, v), S(v, w), T(w, z)\) s.t. \(h_B(v) = i\) and \(h_C(w) = j\)

Input: tuple \((X, (u, v))\), where \(X\) is \(R\), \(S\), or \(T\), and \(u, v\) are the values in the attributes.

**map:**

Map \((S, (v, w))\) to \(((h_B(v), h_C(w)), (S, (v, w))))\)

Map \((R, (u, v))\) to \(((h_B(v), i), (R, (u, v)))\), for each \(i = 1, 2, \ldots, q_C\).
Multi-way joins in MapReduce

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, D) \]

Reducer \((i, j)\) receives all tuples \(R(u, v), S(v, w), T(w, z)\) s.t. \(h_B(v) = i\) and \(h_C(w) = j\)

Input: tuple \((X, (u, v))\), where \(X \) is \(R, S,\) or \(T\), and \(u, v\) are the values in the attributes.

map:
- Map \((S, (v, w))\) to \(((h_B(v), h_C(w)), (S, (v, w))))\)
- Map \((R, (u, v))\) to \((h_B(v), i), (R, (u, v)))\), for each \(i = 1, 2, \ldots, q_C\).
- Map \((T, (w, z))\) to \((i, h_C(w)), (T, (w, z)))\), for each \(i = 1, 2, \ldots, q_B\).
Reduce for key \((i, j)\) receives a list of pairs of the form:

\[(R, (u, v)) \text{ s.t. } h_B(v) = i\]
\[(S, (v, w)) \text{ s.t. } h_B(v) = i \text{ and } h_C(w) = j\]
\[(T, (w, z)) \text{ s.t. } h_C(w) = j\]
Multi-way joins in MapReduce

reduce for key \((i, j)\) receives a list of pairs of the form:

\[
\begin{align*}
(R, (u, v)) \text{ s.t. } h_B(v) &= i \\
(S, (v, w)) \text{ s.t. } h_B(v) &= i \text{ and } h_C(w) = j \\
(T, (w, z)) \text{ s.t. } h_C(w) &= j
\end{align*}
\]

Because of potential \textit{collisions}, the reducer cannot blindly output all triplets made of a tuple from \(R\), one from \(S\) and from \(T\).

E.g.: there may be two values \(z_1\) and \(z_2\) such that \(h_B(z_1) = h_B(z_2) = i\).
Multi-way joins in MapReduce

reduce for key \((i, j)\) receives a list of pairs of the form:

\((R, (u, v))\) s.t. \(h_B(v) = i\)
\((S, (v, w))\) s.t. \(h_B(v) = i\) and \(h_C(w) = j\)
\((T, (w, z))\) s.t. \(h_C(w) = j\)

Because of potential collisions, the reducer cannot blindly output all triplets made of a tuple from \(R\), one from \(S\) and from \(T\).

E.g.: there may be two values \(z_1\) and \(z_2\) such that \(h_B(z_1) = h_B(z_2) = i\).

The reducer:

• first joins the collection of pairs with first element \(R\) with the collection of pairs with first element \(S\); and then
• joins the result with the collection of pairs with first element \(T\).
Section Outline

✓ (Natural Equi-)Joins in MapReduce

✓ Multi-way Joins in MapReduce

Self-Joins

Triangle counting in MapReduce with a Multi-way Self-Join
Self-join

Often, we want to join a relation $R$ with itself.

Example: Join Scores with itself to find students with the same grade in HW01.

We have to imagine that $R$ is two relations with different schemas, except for the attribute we want to join on:

$\text{ScoresHW}_1(\text{Students}_1, \text{HW01}, \text{HW02}_1, \text{HW03}_1)$
$\text{ScoresHW}_2(\text{Students}_2, \text{HW01}, \text{HW02}_2, \text{HW03}_2)$
Example of a self-join

ScoresHW:

<table>
<thead>
<tr>
<th>Student</th>
<th>HW01</th>
<th>HW02</th>
<th>HW03</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>AE</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>
Example of a self-join

ScoresHW:

<table>
<thead>
<tr>
<th>Student</th>
<th>HW01</th>
<th>HW02</th>
<th>HW03</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>AE</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

ScoresHW(...,HW01,...) ⊳ ScoresHW(...,HW01,...)

<table>
<thead>
<tr>
<th>Student₁</th>
<th>HW01</th>
<th>HW02₁</th>
<th>HW03₁</th>
<th>Student₂</th>
<th>HW02₂</th>
<th>HW03₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
<td>ZZ</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>MR</td>
<td>15</td>
<td>23</td>
<td>12</td>
<td>MR</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
<td>MR</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>ZZ</td>
<td>15</td>
<td>30</td>
<td>15</td>
<td>ZZ</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>
Self-join on different attributes

We can also do a self-join on different attributes. Best explained with example.

Let $G = (V, E)$ be a directed graph (edge $(u, w)$ not the same as edge $(w, u)$). Consider the relation $R(U, W)$ w/ 1 tuple for each $(u, w) \in E$. 
Self-join on different attributes

We can also do a self-join on different attributes. Best explained with example.

Let $G = (V, E)$ be a directed graph (edge $(u, w)$ not the same as edge $(w, u)$).
Consider the relation $R(U, W)$ w/ 1 tuple for each $(u, w) \in E$.

We want to find paths of length 2 in $G$,
  i.e., triplets $(u, w, z)$ such that $(u, w) \in E$ and $(w, z) \in E$
Self-join on different attributes

We can also do a self-join on different attributes. Best explained with example.

Let \( G = (V, E) \) be a directed graph (edge \((u, w)\) not the same as edge \((w, u)\)). Consider the relation \( R(U, W) \) w/ 1 tuple for each \((u, w) \in E\).

We want to find paths of length 2 in \( G \),
\[ i.e., \text{triplets } (u, w, z) \text{ such that } (u, w) \in E \text{ and } (w, z) \in E \]

Every tuple \( t \) of \( R \) to be joined with every tuple \( s \) of \( R \) if and only if \( t.W = s.U \).

We denote this operation as \( R(U, W) \Join R(W, Z) \)
Section Outline

✓ (Natural Equi-)Joins in MapReduce

✓ Multi-way Joins in MapReduce

✓ Self-Joins

Triangle counting in MapReduce with a Multi-way Self-Join
Triangle counting in MapReduce

Undirected graph $G$. Vertices are numbered $1, 2, \ldots, n$.

We have a relation $E$ for the edges: a tuple $t = (u, v)$ is in $E$ iff there is an edge between $u$ and $v$ and $u < v$. 
Triangle counting in MapReduce

Undirected graph $G$. Vertices are numbered $1, 2, \ldots , n$.

We have a relation $E$ for the edges: a tuple $t = (u, v)$ is in $E$ iff there is an edge between $u$ and $v$ and $u < v$. The set of triangles in $G$ can be expressed as the triplets in the natural multi-way self-join:

$$E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z)$$

Each triangle $(v_1, v_2, v_3)$ appears once:

If $v_1 < v_2 < v_3$, the triangle is obtained when $X = v_1, Y = v_2$, and $Z = v_3$. 
Triangle counting in MapReduce

The multi-way *self*-join

\[ E(X, Y) \Join E(X, Z) \Join E(Y, Z) \]

is not the same as the multi-way join

\[ R(A, B) \Join S(B, C) \Join T(C, D) \]

The self-join has an additional constraint on the equality of attributes between the relations on the sides.

The MapReduce algorithm will work a little different.
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

Single hash function \( h \) which maps to nodes \( q \) buckets

\( q^3 \) reducers, each associated to a triplet \((x, y, z)\) of buckets.
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

Single hash function \( h \) which maps to nodes \( q \) buckets

\( q^3 \) reducers, each associated to a triplet \((x, y, z)\) of buckets.

map must consider each tuple \((u, v)\) for each of the three roles it plays, and send it to the appropriate reducers in the appropriate form.
Triangle counting in MapReduce

\[ E(X, Y) \Join E(X, Z) \Join E(Y, Z) \]

**map:** input is \((u, v)\)

1) Consider \((u, v)\) as a tuple in the first relation:

Output \([(h(u), h(v), z), (1, (u, v))]\) for each \(z = 1, \ldots q\)

   - **key** \([h(u), h(v), z]\)
   - **value** \((1, (u, v))\)
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

**map**: input is \((u, v)\)

1) Consider \((u, v)\) as a tuple in the first relation:
   
   Output \(((h(u), h(v), z), (1, (u, v)))\) for each \(z = 1, \ldots q\)

2) Consider \((u, v)\) as a tuple in the second relation:
   
   Output \(((h(u), y, h(v), (2, (u, v)))\) for each \(y = 1, \ldots q\)
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

**map:** input is \((u, v)\)

1) Consider \((u, v)\) as a tuple in the first relation:
   Output \(((h(u), h(v), z), (1, (u, v)))\) for each \(z = 1, \ldots, q\)

2) Consider \((u, v)\) as a tuple in the second relation:
   Output \(((h(u), y, h(v), (2, (u, v)))\) for each \(y = 1, \ldots, q\)

3) Consider \((u, v)\) as a tuple in the third relation:
   Output \((x, h(u), h(v), (3, (u, v)))\) for each \(x = 1, \ldots, q\)
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

The reducer with key \((x, y, z)\) receives a list of triplets in the form:

1. \((1, (u, v))\) s.t. \(h(u) = x\) and \(h(v) = y\).
2. \((2, (w, k))\) s.t. \(h(w) = x\) and \(h(k) = z\).
3. \((3, (r, s))\) s.t. \(h(r) = y\) and \(h(s) = z\).
The reducer with key \((x, y, z)\) receives a list of triplets in the form:

(1, \((u, v)\)) s.t. \(h(u) = x\) and \(h(v) = y\).

(2, \((w, k)\)) s.t. \(h(w) = x\) and \(h(k) = z\).

(3, \((r, s)\)) s.t. \(h(r) = y\) and \(h(s) = z\).

Because of potential collisions, the reducer cannot blindly output all triplets made of a pair with first element 1, a pair with first element 2, and a pair with first element 3.

E.g.: there may be two values \(a_1\) and \(a_2\) such that \(h(a_1) = h(a_2) = x\).
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

The reducer with key \((x, y, z)\) receives a list of triplets in the form:

1. \((1, (u, v))\) s.t. \(h(u) = x\) and \(h(v) = y\).
2. \((2, (w, k))\) s.t. \(h(w) = x\) and \(h(k) = z\).
3. \((3, (r, s))\) s.t. \(h(r) = y\) and \(h(s) = z\).

Because of potential collisions, the reducer cannot blindly output all triplets made of a pair with first element 1, a pair with first element 2, and a pair with first element 3.

E.g.: there may be two values \(a_1\) and \(a_2\) such that \(h(a_1) = h(a_2) = x\).

The reducer:

• first joins the collection of pairs with first element 1 with the collection of pairs with first element 1, checking if \(u == w\); and then
Triangle counting in MapReduce

\[ E(X, Y) \bowtie E(X, Z) \bowtie E(Y, Z) \]

The reducer with key \((x, y, z)\) receives a list of triplets in the form:

1. \((1, (u, v))\) s.t. \(h(u) = x\) and \(h(v) = y\).
2. \((2, (w, k))\) s.t. \(h(w) = x\) and \(h(k) = z\).
3. \((3, (r, s))\) s.t. \(h(r) = y\) and \(h(s) = z\).

Because of potential collisions, the reducer cannot blindly output all triplets made of a pair with first element \(1\), a pair with first element \(2\), and a pair with first element \(3\).

E.g.: there may be two values \(a_1\) and \(a_2\) such that \(h(a_1) = h(a_2) = x\).

The reducer:
- first joins the collection of pairs with first element \(1\) with the collection of pairs with first element \(1\), checking if \(u == w\); and then
- joins the result with the collection of pairs with first element \(3\), checking if \(v == r\) \(\AND\) \(k == s\).
Section Outline

✓ (Natural Equi-)Joins in MapReduce

✓ Multi-way Joins in MapReduce

✓ Self-Joins

✓ Triangle counting in MapReduce with a Multi-way Self-Join
Outline

✓ **Graphs**: basic definitions

✓ **Triangles**: what and why

✓ **Approximate** counting of triangles in *edge data streams*

✓ **Exact** counting of triangles in a *static graph*

✓ **Approximate** counting of triangles in a static graph

✓ Counting triangles in *MapReduce*