An Improved Combinatorial Algorithm for Boolean Matrix Multiplication

Huacheng Yu

Stanford University

June 8, 2016

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$\begin{pmatrix} 1 & 0 & \cdots & 1 \\ 0 & 1 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & \cdots & 1 \\ 1 & 1 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} =$$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$\begin{pmatrix} 1 & 0 & \cdots & 1 \\ 0 & 1 & \cdots & 0 \\ a_{i,1} & a_{i,2} & \cdots & a_{i,n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & b_{1,j} & \cdots & 1 \\ 1 & b_{2,j} & \cdots & 0 \\ 1 & b_{3,j} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_{n,j} & \cdots & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 0 \\ 1 & c_{i,j} & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

No harder than Integer MM:

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

No harder than Integer MM:

- Strassen [1969]: $O(n^{2.81})$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

No harder than Integer MM:

- Strassen [1969]: $O(n^{2.81})$
- Coppersmith-Winograd [1990]: $O(n^{2.376})$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

No harder than Integer MM:

- Strassen [1969]: $O(n^{2.81})$
- Coppersmith-Winograd [1990]: $O(n^{2.376})$
- Vassilevska Williams [2012], Le Gall [2014]: $O(n^{2.373})$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

No harder than Integer MM:

- Strassen [1969]: $O(n^{2.81})$
- Coppersmith-Winograd [1990]: $O(n^{2.376})$
- Vassilevska Williams [2012], Le Gall [2014]: $O(n^{2.373})$

Algebraic algorithms!

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Combinatorial algorithms!

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Combinatorial algorithms!

- Four Russians [1970]: $O(n^3/\log^2 n)$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Combinatorial algorithms!

- Four Russians [1970]: $O(n^3/\log^2 n)$
- Bansal-Williams [2009]: $\hat{O}(n^3/\log^{2.25} n)$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Combinatorial algorithms!

- Four Russians [1970]: $O(n^3/\log^2 n)$
- Bansal-Williams [2009]: $\hat{O}(n^3/\log^{2.25} n)$
- Chan [2015]: $\hat{O}(n^3/\log^3 n)$

Introduction

Boolean Matrix Multiplication (Boolean MM or BMM):

$$c_{i,j} = \bigvee_{k=1}^{n} (a_{i,k} \wedge b_{k,j})$$

Combinatorial algorithms!

- Four Russians [1970]: $O(n^3/\log^2 n)$
- Bansal-Williams [2009]: $\hat{O}(n^3/\log^{2.25} n)$
- Chan [2015]: $\hat{O}(n^3/\log^3 n)$
- This paper [2015]: $\hat{O}(n^3/\log^4 n)$

Algebraic vs Combinatorial

On Boolean MM:

Algebraic Algorithms	Combinatorial Algorithms
Asymptotically faster	Asymptotically slower

Algebraic vs Combinatorial

On Boolean MM:

Algebraic Algorithms	Combinatorial Algorithms
Asymptotically faster	Asymptotically slower
Difficult to implement slow in practice	Very simple fast in practice

Algebraic vs Combinatorial

On Boolean MM:

Algebraic Algorithms	Combinatorial Algorithms
Asymptotically faster	Asymptotically slower
Difficult to implement slow in practice	Very simple fast in practice
Require similar algebraic structure to generalize	Generalizable in a different way

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$
- solve sequence alignment in $o(n^2)$

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$
- solve sequence alignment in $o(n^2)$
- multiply $n \times \hat{O}(\log^3 n)$ and $\hat{O}(\log^3 n) \times n$ Boolean matrices in $O(n^2)$ time

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$
- solve sequence alignment in $o(n^2)$
- multiply $n \times \hat{O}(\log^3 n)$ and $\hat{O}(\log^3 n) \times n$ Boolean matrices in $O(n^2)$ time
- multiply $n \times n$ and $n \times \hat{O}(\log^3 n)$ Boolean matrices in $O(n^2)$ time

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$
- solve sequence alignment in $o(n^2)$
- multiply $n \times \hat{O}(\log^3 n)$ and $\hat{O}(\log^3 n) \times n$ Boolean matrices in $O(n^2)$ time
- multiply $n \times n$ and $n \times \hat{O}(\log^3 n)$ Boolean matrices in $O(n^2)$ time
- preprocess a Boolean matrix A, answer queries Ax in $n^2/\log^2 n$

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$
- solve sequence alignment in $o(n^2)$
- multiply $n \times \hat{O}(\log^3 n)$ and $\hat{O}(\log^3 n) \times n$ Boolean matrices in $O(n^2)$ time
- multiply $n \times n$ and $n \times \hat{O}(\log^3 n)$ Boolean matrices in $O(n^2)$ time
- preprocess a Boolean matrix A, answer queries Ax in $n^2/\log^2 n$
  - Larsen-Williams [2016]: $n^2/2^{\Omega(\sqrt{\log n})}$ (by a half-algebraic algorithm)

Combinatorial Algorithms

Combinatorial algorithms can:

- solve edit distance in $o(n^2)$
- solve sequence alignment in $o(n^2)$
- multiply $n \times \hat{O}(\log^3 n)$ and $\hat{O}(\log^3 n) \times n$ Boolean matrices in $O(n^2)$ time
- multiply $n \times n$ and $n \times \hat{O}(\log^3 n)$ Boolean matrices in $O(n^2)$ time
- preprocess a Boolean matrix A, answer queries Ax in $n^2/\log^2 n$
  - Larsen-Williams [2016]: $n^2/2^{\Omega(\sqrt{\log n})}$ (by a half-algebraic algorithm) (randomized, heavy preprocessing or amortized time)

Triangle Detection

Triangle detection: *given an n-node graph, does it contain a triangle (3-clique or 3-cycle)?*

Triangle Detection

Triangle detection: *given an n-node graph, does it contain a triangle (3-clique or 3-cycle)?*

Vassilevska Williams and Williams [2010] proved:

$$\text{Triangle detection} \xleftrightarrow{\text{“sub-cubic”}} \text{Boolean MM}$$

Triangle Detection

Triangle detection: *given an n-node graph, does it contain a triangle (3-clique or 3-cycle)?*

Vassilevska Williams and Williams [2010] proved:

$$\text{Triangle detection} \xleftrightarrow{\text{“sub-cubic”}} \text{Boolean MM}$$

$$O(n^3/g(n)) \xrightarrow{\text{combinatorial}} O(n^3/g(n^{1/3}))$$

Main Result

Our main theorem:

Theorem

Given a graph G *on* n *vertices*

Main Result

Our main theorem:

Theorem

Given a graph G on n vertices, we can detect if there is a triangle in G using a combinatorial algorithm in $\hat{O}(n^3/\log^4 n)$ time.

Main Result

Our main theorem:

Theorem

Given a graph G on n vertices, we can detect if there is a triangle in G using a combinatorial algorithm in $\hat{O}(n^3/\log^4 n)$ time.

A general framework for triangle detection:

Main Result

Our main theorem:

Theorem

Given a graph G on n vertices, we can detect if there is a triangle in G using a combinatorial algorithm in $\hat{O}(n^3/\log^4 n)$ time.

A general framework for triangle detection:

"Theorem"

If there is an algorithm that takes a graph and can "efficiently" find and solve triangle detection on a "large" subgraph

Main Result

Our main theorem:

Theorem

Given a graph G *on* n *vertices, we can detect if there is a triangle in* G *using a combinatorial algorithm in* $\hat{O}(n^3/\log^4 n)$ *time.*

A general framework for triangle detection:

"Theorem"

If there is an algorithm that takes a graph and can "efficiently" find and solve triangle detection on a "large" subgraph, then triangle detection is "easy" in general.

Triangle Detection Algorithm

Preliminaries

Wish to detect if a graph G contains a triangle.

Preliminaries

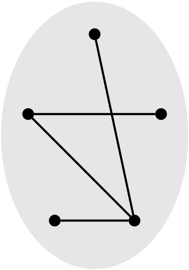
Wish to detect if a graph G contains a triangle.

Observation: can assume G is tripartite.

Preliminaries

Wish to detect if a graph G contains a triangle.

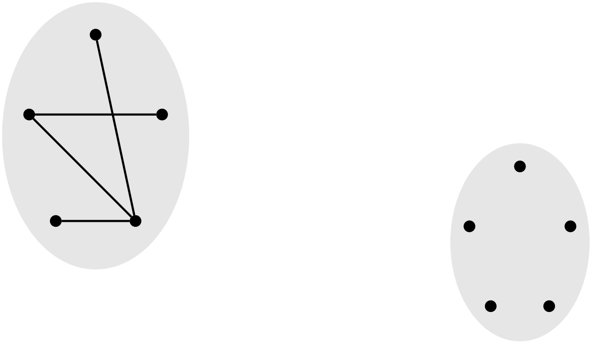
Observation: can assume G is tripartite.



Preliminaries

Wish to detect if a graph G contains a triangle.

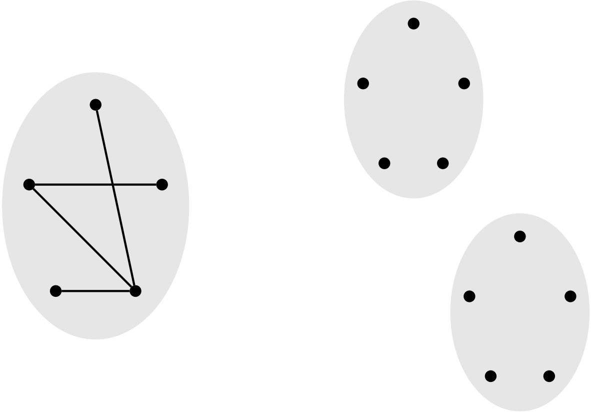
Observation: can assume G is tripartite.



Preliminaries

Wish to detect if a graph G contains a triangle.

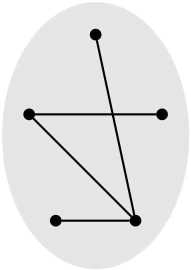
Observation: can assume G is tripartite.

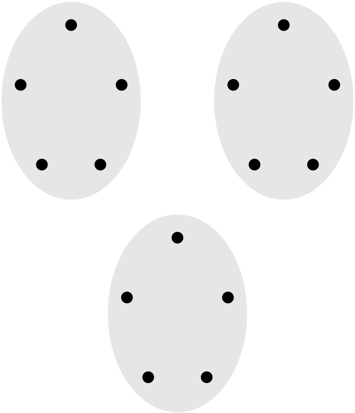


Preliminaries

Wish to detect if a graph G contains a triangle.

Observation: can assume G is tripartite.

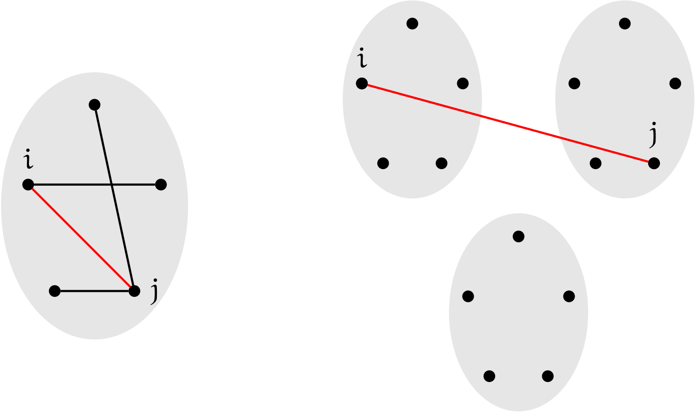




Preliminaries

Wish to detect if a graph G contains a triangle.

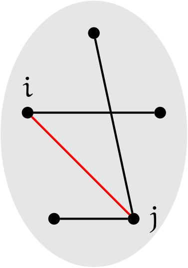
Observation: can assume G is tripartite.

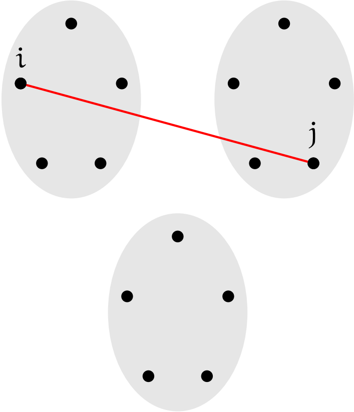


Preliminaries

Wish to detect if a tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle.

Observation: can assume G is tripartite.

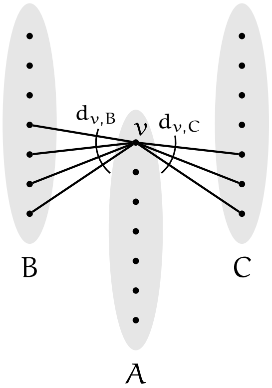




Preliminaries

Wish to detect if a tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle.

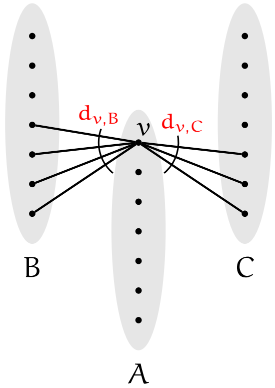
- One naïve approach:
  for $v \in A$, check edge between its neighbours



Preliminaries

Wish to detect if a tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle.

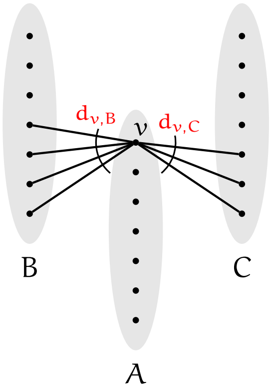
- One naïve approach:
  for $v \in A$, check edge between its neighbours
- spend $d_{v,B} d_{v,C}$ time



Preliminaries

Wish to detect if a tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle.

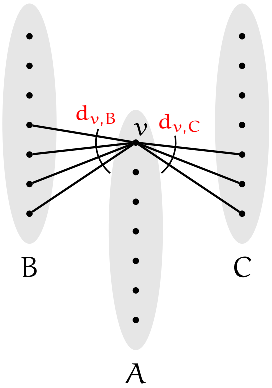
- One naïve approach:
  for $v \in A$, check edge between its neighbours
- spend $d_{v,B} d_{v,C}$ time
- fast if $d_{v,B} d_{v,C}$ small on average



Preliminaries

Wish to detect if a tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle.

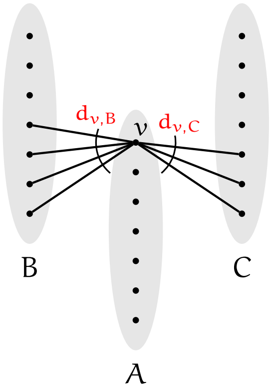
- One naïve approach:
  for $v \in A$, check edge between its neighbours
- spend $d_{v,B}\, d_{v,C}$ time
- fast if $d_{v,B}\, d_{v,C}$ small on average
- otherwise can find large "non-edge area" between B and C



Preliminaries

Wish to detect if a tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle.

- One naïve approach:
  for $v \in A$, check edge between its neighbours
- spend $d_{v,B} d_{v,C}$ time
- fast if $d_{v,B} d_{v,C}$ small on average
- otherwise can find large "non-edge area" between B and C
- **recursion!** (also used in Chan's algorithm)



The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

The Sparse Case

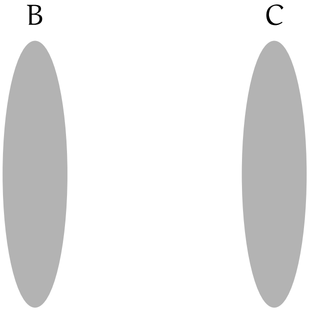
The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

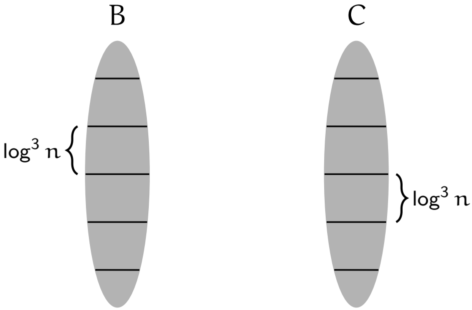
Solvable in time $\hat{O}(n^3/\log^4 n)$.



The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

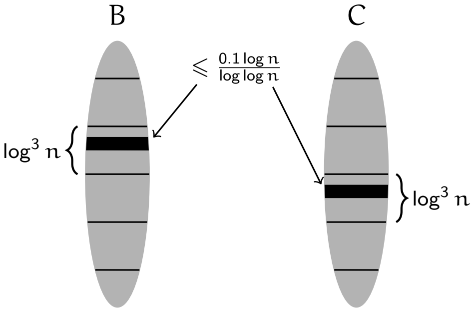
Solvable in time $\hat{O}(n^3/\log^4 n)$.



The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

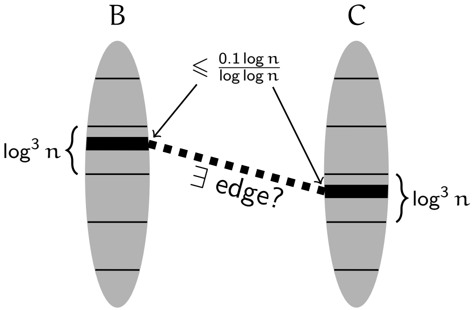
Solvable in time $\hat{O}(n^3/\log^4 n)$.



The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

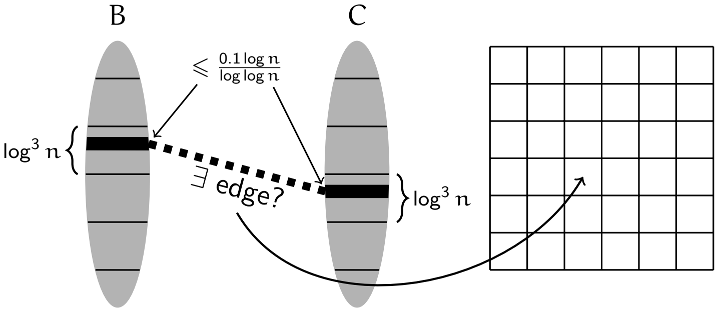
Solvable in time $\hat{O}(n^3/\log^4 n)$.



The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

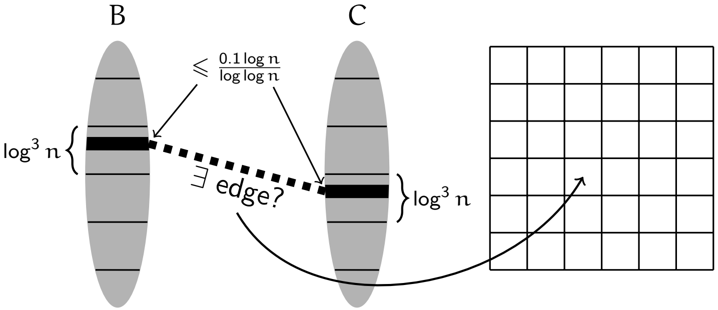
Solvable in time $\hat{O}(n^3/\log^4 n)$.



The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.



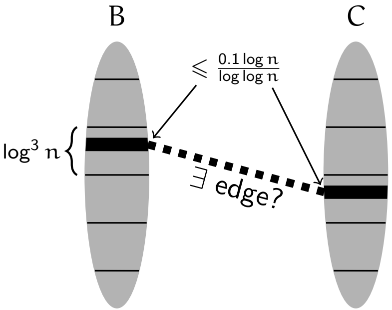
Preprocessing: $n^2(\log^3 n)^{0.2 \log n / \log\log n} = O(n^{2.6})$.

The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

Preprocessing: $O(n^{2.6})$.



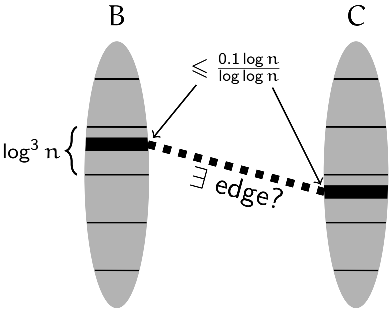
The Sparse Case

The sparse case: $d_{v,B}d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

Preprocessing: $O(n^{2.6})$.

- for $v \in A$, partition its neighbourhood into small subsets within each block



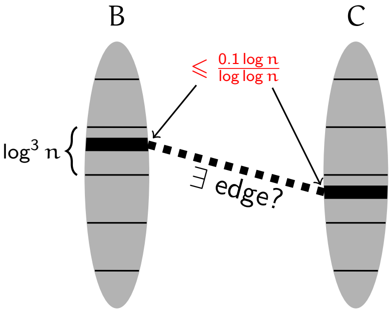
The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

Preprocessing: $O(n^{2.6})$.

- for $v \in A$, partition its neighbourhood into small subsets within each block



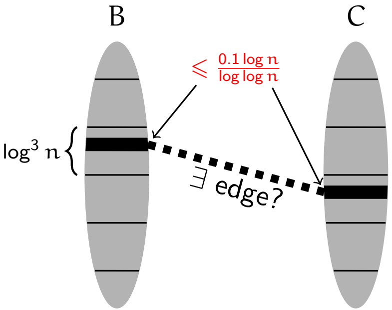
The Sparse Case

The sparse case: $d_{v,B}\, d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

Preprocessing: $O(n^{2.6})$.

- for $v \in A$, partition its neighbourhood into small subsets within each block
- check each pair of small subsets by lookup table



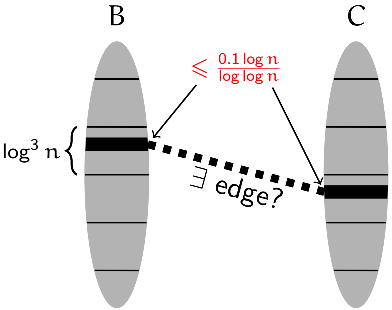
The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

Preprocessing: $O(n^{2.6})$.

- for $v \in A$, partition its neighbourhood into small subsets within each block
- check each pair of small subsets by lookup table
- $\hat{O}(d_{v,B}/\log n + n/\log^3 n)$, $\hat{O}(d_{v,C}/\log n + n/\log^3 n)$ small subsets respectively



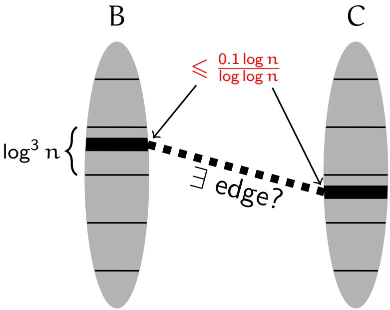
The Sparse Case

The sparse case: $d_{v,B} d_{v,C} \leqslant \hat{O}(n^2/\log^2 n)$ for every $v \in A$.

Solvable in time $\hat{O}(n^3/\log^4 n)$.

Preprocessing: $O(n^{2.6})$.

- for $v \in A$, partition its neighbourhood into small subsets within each block
- check each pair of small subsets by lookup table
- $\hat{O}(d_{v,B}/\log n + n/\log^3 n)$, $\hat{O}(d_{v,C}/\log n + n/\log^3 n)$ small subsets respectively



Spend $\hat{O}(n^3/\log^4 n)$ in total.

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0:

Step 1:

Step 2:

Step 3:

Step 4:

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1:

Step 2:

Step 3:

Step 4:

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2:

Step 3:

Step 4:

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3:

Step 4:

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4:

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

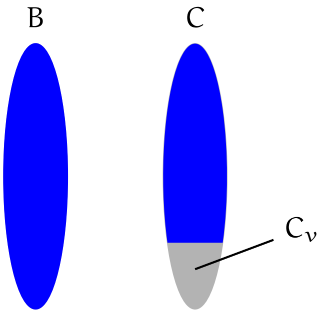
Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

The Algorithm

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

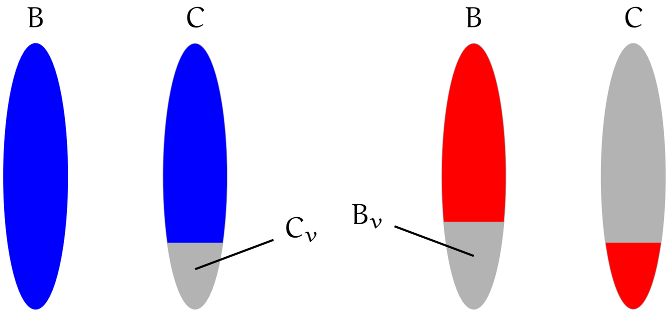
The Algorithm

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.



The Algorithm

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.



The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B}d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B}d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log \log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

Correctness: straightforward.

Analysis of the Running Time

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

Analysis of the Running Time

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

Analysis of the Running Time

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

Analysis of the Running Time

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.
$O(n(|B| + |C|))$

Analysis of the Running Time

- Small graph case:

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.
$O(n(|B| + |C|))$

Analysis of the Running Time

- Small graph case:

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

- Large graph case:

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.
$O(n(|B| + |C|))$

Analysis of the Running Time

- Small graph case:

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

- Large graph case:

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.
$O(n(|B| + |C|))$

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$, then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$, else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$. $O(n(|B| + |C|))$

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B}\,d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.
$O(n(|B| + |C|))$

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B|+|C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair
- dense case: charge to $B_v \times C_v$ evenly. $O(\sqrt{n}\log^2 n)$ per pair

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair
- dense case: charge to $B_v \times C_v$ evenly. $O(\sqrt{n}\log^2 n)$ per pair

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair
- dense case: charge to $B_v \times C_v$ evenly. $O(\sqrt{n}\log^2 n)$ per pair

Every pair is charged at most once!

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair
- dense case: charge to $B_v \times C_v$ evenly. $O(\sqrt{n}\log^2 n)$ per pair

Every pair is charged at most once!

- only charge to pairs not going into the same recursive branch.

Large Graph Case

Large graph case ($|B|, |C| \geqslant \sqrt{n}$):

- sparse case algorithm. $\hat{O}(n|B||C|/\log^4 n)$
- check degrees, generate inputs for recursion. $O(n(|B| + |C|))$
- check $B_v \times C_v$. $O(|B_v||C_v|)$

Charge the running time to pairs in $B \times C$:

- sparse case: charge to $B \times C$ evenly. $\hat{O}(n/\log^4 n)$ per pair
- dense case: charge to $B_v \times C_v$ evenly. $O(\sqrt{n}\log^2 n)$ per pair

Every pair is charged at most once!

- only charge to pairs not going into the same recursive branch.

Time spent on large graph case: $\hat{O}(n^3/\log^4 n)$.

Analysis of the Running Time

- Small graph case:

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

- Large graph case:

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm; $\hat{O}(n|B||C|/\log^4 n)$

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; let B_v [resp. C_v] be v's neighbourhood in B [resp. C]; $O(n(|B| + |C|))$

Step 3: Check all pairs in $B_v \times C_v$ for an edge; $O(|B_v||C_v|)$

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.
$O(n(|B| + |C|))$

Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log \log n)^2))$

Small Graph Case

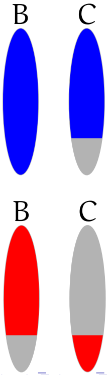
Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B}d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

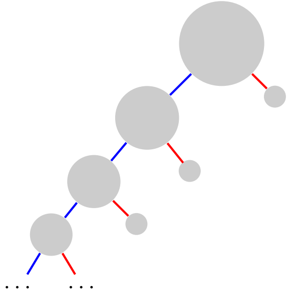
Let v: $d_{v,B}\, d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

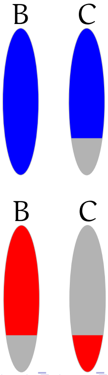


Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B}\, d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n / (\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

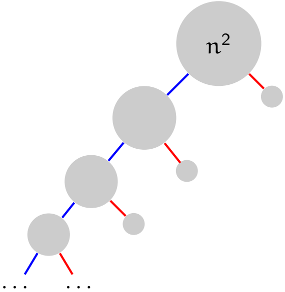


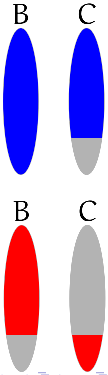


Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B}\, d_{v,C} \geqslant |B||C|/\Delta^2$. ($\Delta = \Theta(\log n/(\log\log n)^2)$)
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

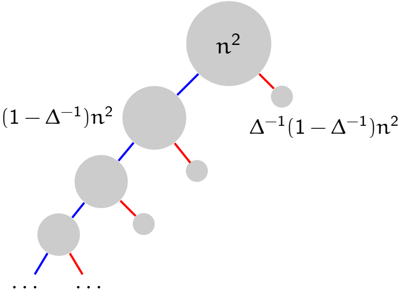


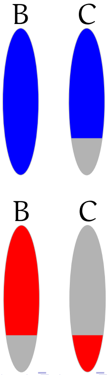


Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

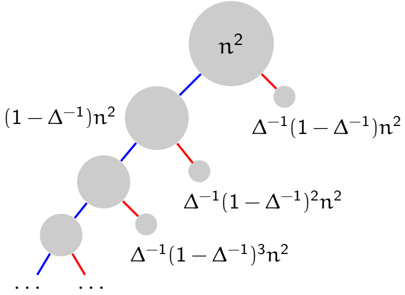


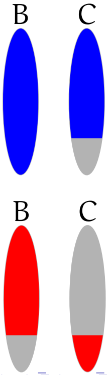


Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B}\, d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

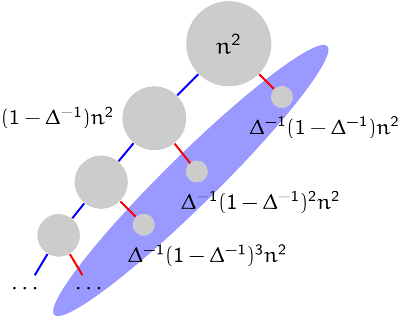


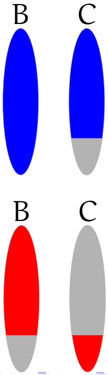


Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B}\, d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

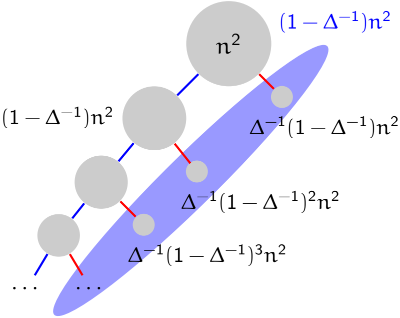


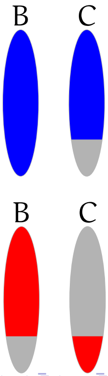


Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B}\, d_{v,C} \geqslant |B||C|/\Delta^2$. ($\Delta = \Theta(\log n / (\log\log n)^2)$)
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

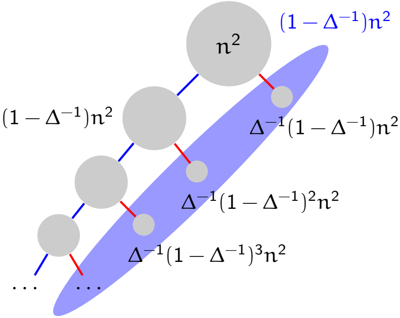




Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$. ($\Delta = \Theta(\log n/(\log\log n)^2)$)
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.

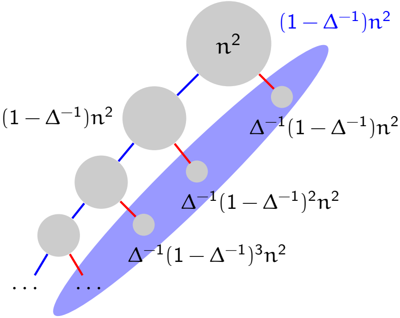


$\Omega(\Delta \log\log n)$ right steps: $O(n^3/\log^{10} n)$

Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$

Let v: $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.



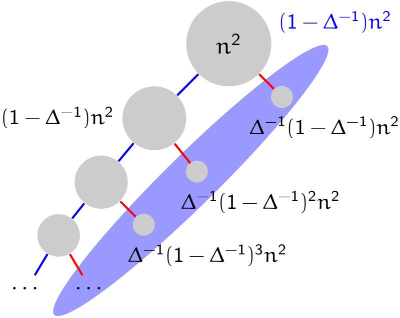
$\Omega(\Delta \log\log n)$ right steps: $O(n^3/\log^{10} n)$

$\#$ nodes with $O(\Delta \log\log n)$ right steps: $n^{0.4}$

Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$ $\leqslant O(n^{2.5})$

Let v: $d_{v,B}d_{v,C} \geqslant |B||C|/\Delta^2$. $(\Delta = \Theta(\log n/(\log\log n)^2))$
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.



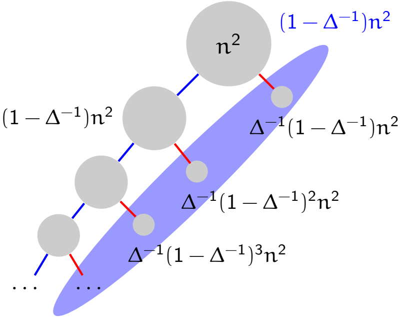
$\Omega(\Delta\log\log n)$ right steps: $O(n^3/\log^{10} n)$

nodes with $O(\Delta\log\log n)$ right steps: $n^{0.4}$

Small Graph Case

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search; $O(n|B||C|)$ $\leqslant O(n^{2.5})$

Let v: $d_{v,B}\,d_{v,C} \geqslant |B||C|/\Delta^2$. ($\Delta = \Theta(\log n/(\log\log n)^2)$)
For simplicity, assume $d_{v,B} = |B|/\Delta$, $d_{v,C} = |C|/\Delta$.



$\Omega(\Delta \log\log n)$ right steps: $O(n^3/\log^{10} n)$

nodes with $O(\Delta\log\log n)$ right steps: $n^{0.4}$

$O(n^3/\log^{10} n)$ time.

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; Let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

Correctness: straightforward.

The Algorithm

Given an n-node tripartite graph, fix $\Delta = \frac{\log n}{100(\log\log n)^2}$.

Step 0: If $|B| < \sqrt{n}$ or $|C| < \sqrt{n}$, use exhaustive search;

Step 1: If for every $v \in A$, $d_{v,B} d_{v,C} \leqslant |B||C|/\Delta^2$, use the sparse case algorithm;

Step 2: Otherwise, find a $v \in A$ such that $d_{v,B} d_{v,C} \geqslant |B||C|/\Delta^2$; Let B_v [resp. C_v] be v's neighbourhood in B [resp. C];

Step 3: Check all pairs in $B_v \times C_v$ for an edge;

Step 4: If $|B_v|/|B| > |C_v|/|C|$,
then recurse on $(A, B, C \setminus C_v)$ and $(A, B \setminus B_v, C_v)$,
else recurse on $(A, B \setminus B_v, C)$ and $(A, B_v, C \setminus C_v)$.

Correctness: straightforward.

Running time: $\hat{O}(n^3/\log^4 n)$.

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM
- A general framework for triangle detection

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM
- A general framework for triangle detection
- Open problem:

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM
- A general framework for triangle detection
- Open problem:
  - Triangle detection on tripartite graphs with vertex set sizes $n, n, \hat{O}(\log^4 n)$ in $O(n^2)$ time?

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM
- A general framework for triangle detection
- Open problem:
  - Triangle detection on tripartite graphs with vertex set sizes $n, n, \hat{O}(\log^4 n)$ in $O(n^2)$ time?
  - Multiplying $n \times n$ and $n \times \hat{O}(\log^4 n)$ Boolean matrices in $O(n^2)$

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM
- A general framework for triangle detection
- Open problem:
  - Triangle detection on tripartite graphs with vertex set sizes $n, n, \hat{O}(\log^4 n)$ in $O(n^2)$ time?
  - Multiplying $n \times n$ and $n \times \hat{O}(\log^4 n)$ Boolean matrices in $O(n^2)$
  - Current record: $\hat{O}(\log^3 n)$ by Chan.

Conclusion

- $\hat{O}(n^3/\log^4 n)$ time algorithm for triangle detection and Boolean MM
- A general framework for triangle detection
- Open problem:
  - Triangle detection on tripartite graphs with vertex set sizes $n, n, \hat{O}(\log^4 n)$ in $O(n^2)$ time?
  - Multiplying $n \times n$ and $n \times \hat{O}(\log^4 n)$ Boolean matrices in $O(n^2)$
  - Current record: $\hat{O}(\log^3 n)$ by Chan.

Thanks!