Faster Algorithms for All Pairs Non-decreasing Paths Problem

Ran Duan, Ce Jin and **Hongxun Wu**

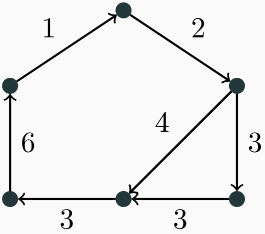
Institute for Interdisciplinary Information Sciences, Tsinghua University

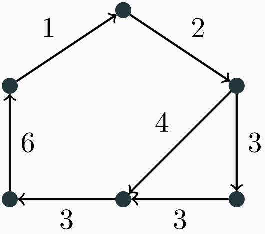
Background

APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

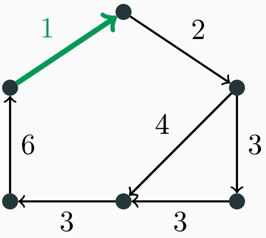


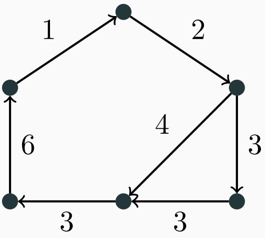


APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

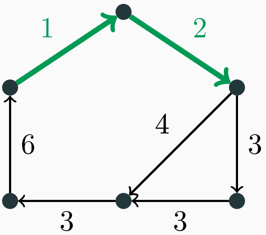


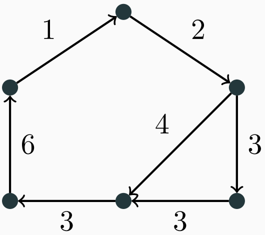


APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

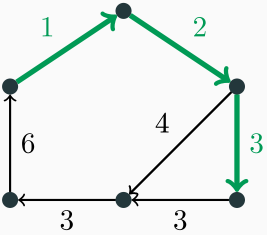


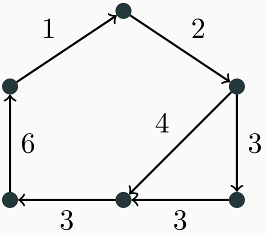


APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

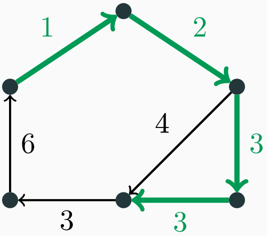


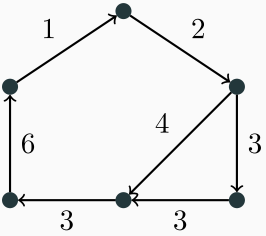


APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

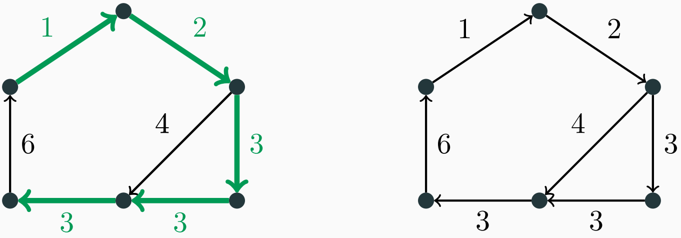




APNP

Nondecreasing Path

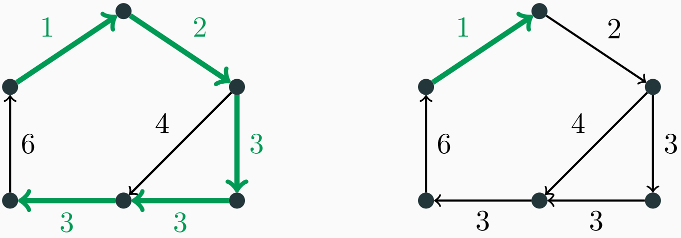
In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.



APNP

Nondecreasing Path

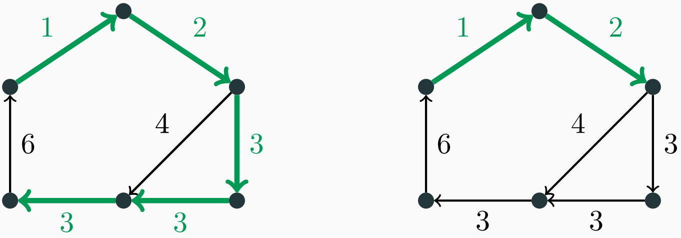
In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.



APNP

Nondecreasing Path

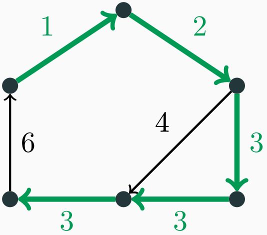
In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

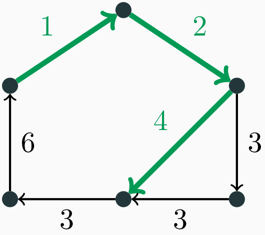


APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.

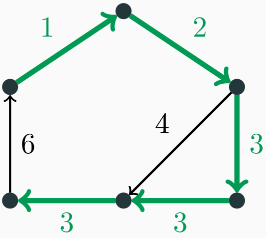


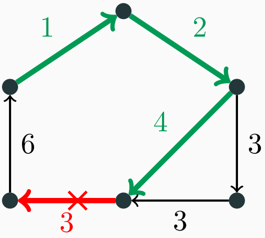


APNP

Nondecreasing Path

In a directed edge-weighted graph G, nondecreasing path $e_1, e_2, \cdots e_n$ is a path with nondecreasing edge-weights $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}) \leq w(e_n)$.





APNP

Nondecreasing Path

Define weight of a path to be the weight of its last edge.

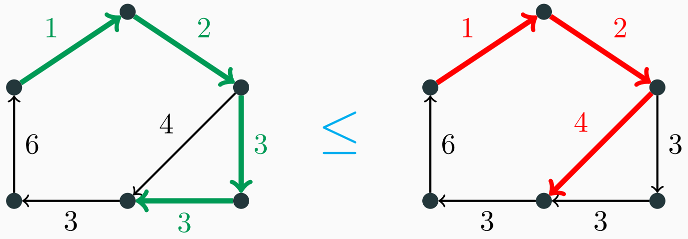
We want this weight to be as small as possible.

APNP

Nondecreasing Path

Define weight of a path to be the weight of its last edge.

We want this weight to be as small as possible.



APNP

Single Source Nondecresing Path (SSNP)

Single source nondecreasing path asks the following problem:

What is the minimum nondecreasing path from s to t?

All Pair Nondecresing Path (APNP)

All pair nondecreasing path asks the following problem **for every pair of vertices s and t**:

What is the minimum nondecreasing path from s to t?

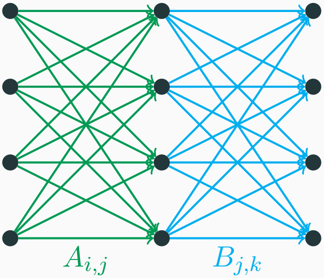
$(min, \leq)$-product

$(min, \leq)$-product

Let A, B be two $n \times n$ matrices, their $(min, \leq)$-product C is

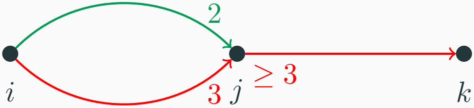
$$C_{i,k} = \min_{k}\{B_{j,k} | A_{i,j} \leq B_{j,k}\}$$

Two level APNP Instance.



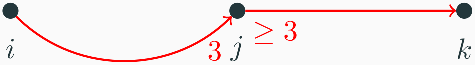
Simple Observation

- Optimal prefix: If we switch the prefix from i to j to the minimum nondecreasing path, it is still a nondecreasing path.



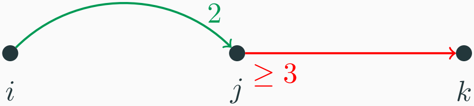
Simple Observation

- Optimal prefix: If we switch the prefix from i to j to the minimum nondecreasing path, it is still a nondecreasing path.



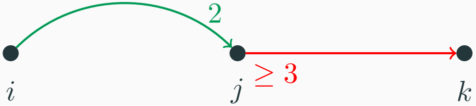
Simple Observation

- Optimal prefix: If we switch the prefix from i to j to the minimum nondecreasing path, it is still a nondecreasing path.



Simple Observation

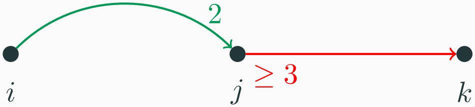
- Optimal prefix: If we switch the prefix from i to j to the minimum nondecreasing path, it is still a nondecreasing path.



- Since the prefix of an optimal path is still an optimal path. We can successively extend those optimal path by one edge to find all optimal paths.

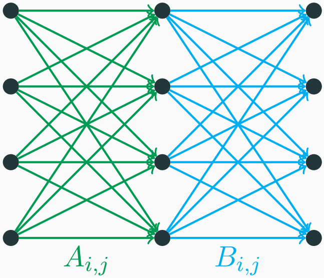
Simple Observation

- Optimal prefix: If we switch the prefix from i to j to the minimum nondecreasing path, it is still a nondecreasing path.



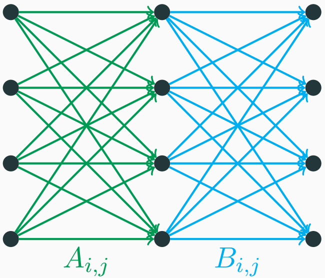
- Since the prefix of an optimal path is still an optimal path. We can successively extend those optimal path by one edge to find all optimal paths.
- Namely, one can compute $n-1$ many $(min, \leq)$-products to solve APNP problem.

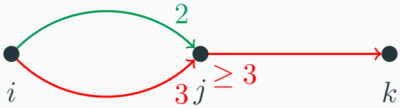
Simple Observation



- $(min, \leq)$-product is simply two level APNP problem.

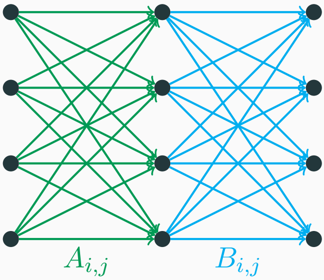
Simple Observation

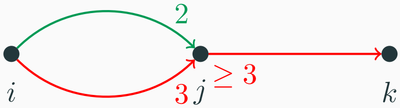




- $(min, \leq)$-product is simply two level APNP problem.
- APNP can be solve by $n-1$ successive $(min, \leq)$-products.
  - But it is not associative, we cannot directly reduce it to $\log(n)$ $(min, \leq)$-products.

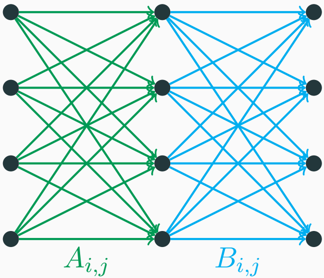
Simple Observation

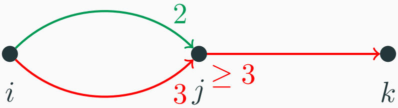




- $(min, \leq)$-product is simply two level APNP problem.
- APNP can be solve by $n-1$ successive $(min, \leq)$-products.
  - But it is not associative, we cannot directly reduce it to $\log(n)$ $(min, \leq)$-products.
- Can we solve APNP as fast as $(min, \leq)$-product?

Simple Observation



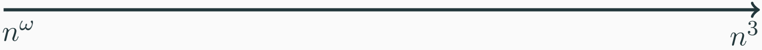


- $(min, \leq)$-product is simply two level APNP problem.
- APNP can be solve by $n-1$ successive $(min, \leq)$-products.
  - But it is not associative, we cannot directly reduce it to $\log(n)$ $(min, \leq)$-products.
- Can we solve APNP as fast as $(min, \leq)$-product? Yes!

Previous Works & Our Result

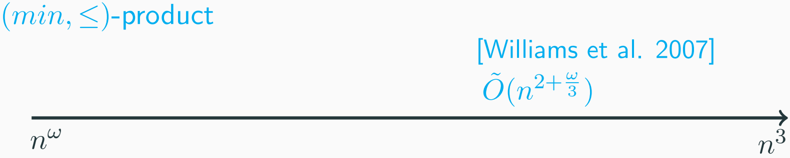
Previous Works

$(min, \leq)$-product



- Here $\omega < 2.373$ is the exponent of the complexity of fast matrix multiplication. Namely, multiplication of two $n \times n$ matrices takes $\Theta(n^\omega)$ time.

Previous Works



$(min, \leq)$-product

[Williams et al. 2007]

$\tilde{O}(n^{2+\frac{\omega}{3}})$

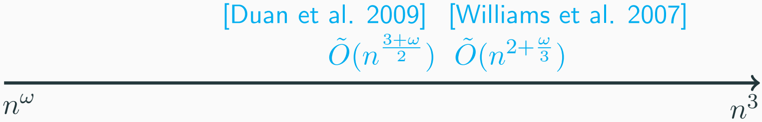
n^{ω}

n^{3}

- Here $\omega < 2.373$ is the exponent of the complexity of fast matrix multiplication. Namely, multiplication of two $n \times n$ matrices takes $\Theta(n^{\omega})$ time.

Previous Works

$(min, \leq)$-product



[Duan et al. 2009] $\tilde{O}(n^{\frac{3+\omega}{2}})$

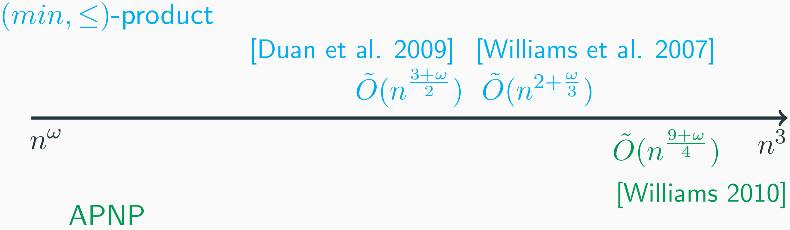
[Williams et al. 2007] $\tilde{O}(n^{2+\frac{\omega}{3}})$

n^ω

n^3

- Here $\omega < 2.373$ is the exponent of the complexity of fast matrix multiplication. Namely, multiplication of two $n \times n$ matrices takes $\Theta(n^\omega)$ time.

Previous Works



$(min, \leq)$-product

[Duan et al. 2009] [Williams et al. 2007]

$\tilde{O}(n^{\frac{3+\omega}{2}})$ $\tilde{O}(n^{2+\frac{\omega}{3}})$

n^ω

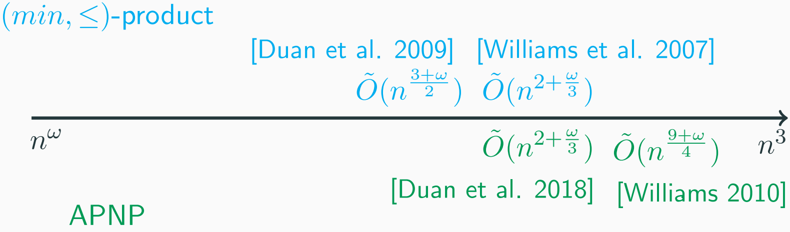
$\tilde{O}(n^{\frac{9+\omega}{4}})$ n^3

[Williams 2010]

APNP

- Here $\omega < 2.373$ is the exponent of the complexity of fast matrix multiplication. Namely, multiplication of two $n \times n$ matrices takes $\Theta(n^\omega)$ time.

Previous Works



$(min, \leq)$-product

[Duan et al. 2009] $\tilde{O}(n^{\frac{3+\omega}{2}})$

[Williams et al. 2007] $\tilde{O}(n^{2+\frac{\omega}{3}})$

n^ω

$\tilde{O}(n^{2+\frac{\omega}{3}})$ [Duan et al. 2018]

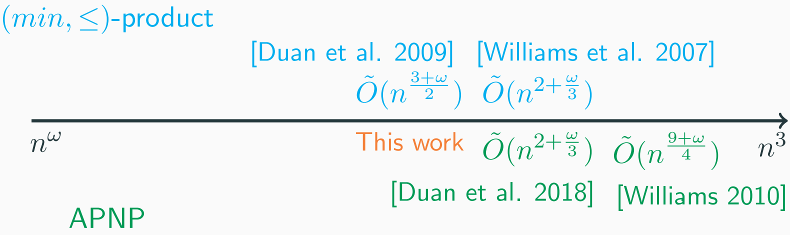
$\tilde{O}(n^{\frac{9+\omega}{4}})$ [Williams 2010]

n^3

APNP

- Here $\omega < 2.373$ is the exponent of the complexity of fast matrix multiplication. Namely, multiplication of two $n \times n$ matrices takes $\Theta(n^\omega)$ time.

Previous Works



$(min, \leq)$-product

[Duan et al. 2009] $\tilde{O}(n^{\frac{3+\omega}{2}})$

[Williams et al. 2007] $\tilde{O}(n^{2+\frac{\omega}{3}})$

n^ω

This work

$\tilde{O}(n^{2+\frac{\omega}{3}})$ [Duan et al. 2018]

$\tilde{O}(n^{\frac{9+\omega}{4}})$ [Williams 2010]

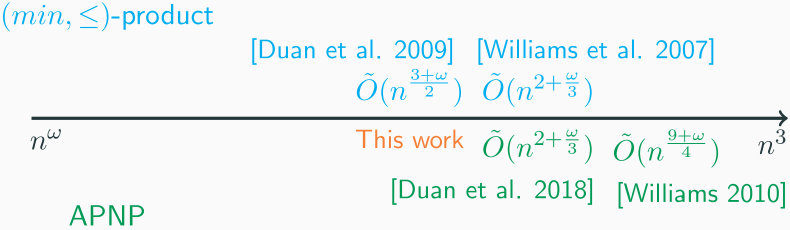
n^3

APNP

Theorem 1

The all pairs non-decreasing paths (APNP) problem on directed simple graphs can be solved in $\tilde{O}(n^{\frac{3+\omega}{2}})$ time.

Previous Works



$(min, \leq)$-product

[Duan et al. 2009] $\tilde{O}(n^{\frac{3+\omega}{2}})$

[Williams et al. 2007] $\tilde{O}(n^{2+\frac{\omega}{3}})$

n^ω

This work

$\tilde{O}(n^{2+\frac{\omega}{3}})$ [Duan et al. 2018]

$\tilde{O}(n^{\frac{9+\omega}{4}})$ [Williams 2010]

n^3

APNP

Theorem 1

The all pairs non-decreasing paths (APNP) problem on directed simple graphs can be solved in $\tilde{O}(n^{\frac{3+\omega}{2}})$ time.

Theorem 2

The all pairs non-decreasing paths (APNP) problem on undirected simple graphs can be solved in $\tilde{O}(n^2)$ time.

Our algorithm for APNP on directed simple graphs

Hign Level Idea

- Modified simplest Dijkstra algorithm

Hign Level Idea

- Modified simplest Dijkstra algorithm
  - Take high degree vertex and low degree vertex differently

Hign Level Idea

- Modified simplest Dijkstra algorithm
    - Take high degree vertex and low degree vertex differently
- Novel divide and conquer approach

Hign Level Idea

- Modified simplest Dijkstra algorithm
    - Take high degree vertex and low degree vertex differently
- Novel divide and conquer approach
    - As graph gets sparser, the matrices get smaller.

Hign Level Idea

- Modified simplest Dijkstra algorithm
  - Take high degree vertex and low degree vertex differently
- Novel divide and conquer approach
  - As graph gets sparser, the matrices get smaller.
  - Though still n - 1 matrix multiplications, most of them are small.

Hign Level Idea

- Modified simplest Dijkstra algorithm
  - Take high degree vertex and low degree vertex differently
- Novel divide and conquer approach
  - As graph gets sparser, the matrices get smaller.
  - Though still n - 1 matrix multiplications, most of them are small.
- Simplest divide and conquer approach still has high complexity

Hign Level Idea

- Modified simplest Dijkstra algorithm
    - Take high degree vertex and low degree vertex differently
- Novel divide and conquer approach
    - As graph gets sparser, the matrices get smaller.
    - Though still n - 1 matrix multiplications, most of them are small.
- Simplest divide and conquer approach still has high complexity
    - Because of the existence of “high-low edges”.

Hign Level Idea

- Modified simplest Dijkstra algorithm
    - Take high degree vertex and low degree vertex differently
- Novel divide and conquer approach
    - As graph gets sparser, the matrices get smaller.
    - Though still n - 1 matrix multiplications, most of them are small.
- Simplest divide and conquer approach still has high complexity
    - Because of the existence of "high-low edges".
    - So we design an oracle which helps us handle them.

High Degree and Low Degree

- Classify vertices according to their degrees.
    - t is a parameter to be determined later.
    - Low degree : $\leq n^{1-t}$ edges.
    - High degree : $> n^{1-t}$ edges.

High Degree and Low Degree

- Classify vertices according to their degrees.
  - t is a parameter to be determined later.
  - Low degree : $\leq n^{1-t}$ edges.
  - High degree : $> n^{1-t}$ edges.
- Edges are classified into three types according to the degree of their end points.
  - Low edges: low $\rightarrow$ high/low
  - High-low edges: high $\rightarrow$ low
  - High-high edges: high $\rightarrow$ high

Dijkstra Search

Algorithm 1 Dijkstra Search for APNP

```
for minimum unvisited nondecreasing path $i \to j$ do
    for each edge $(j, k)$ s.t. $w(i \to j) \leq w(j, k)$ do
        Relax edge $(j, k)$ by $d(i \to k) \leftarrow \min(d(i \to k), w(j, k))$
    end for
end for
```

- The simplest $O(n^3)$ algorithm for APNP is a Dijkstra Search which always visit the minimum unvisited nondecreasing path

Dijkstra Search

Algorithm 1 Dijkstra Search for APNP

```
for minimum unvisited nondecreasing path $i \to j$ do
    for each edge $(j, k)$ s.t. $w(i \to j) \le w(j, k)$ do
        Relax edge $(j, k)$ by $d(i \to k) \leftarrow \min(d(i \to k), w(j, k))$
    end for
end for
```

- The simplest $O(n^3)$ algorithm for APNP is a Dijkstra Search which always visit the minimum unvisited nondecreasing path
- This procedure is very friendly to low degree vertices.

Dijkstra Search

Algorithm 1 Dijkstra Search for APNP

```
for minimum unvisited nondecreasing path $i \to j$ do
    for each edge $(j, k)$ s.t. $w(i \to j) \leq w(j, k)$ do
        Relax edge $(j, k)$ by $d(i \to k) \leftarrow \min(d(i \to k), w(j, k))$
    end for
end for
```

- Basic idea:
  - For low degree vertices, enumerate all outgoing edges (j, k) is efficient enough.
  - For high degree vertices, the graph gets sparse after recursion, there are only bounded number of them. We use fast matrix multiplication to relax edges associated with high degree vertices.

Divide and Conquer

Algorithm 2 Divide and Conquer

```
function SOLVE($G$)
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```

- Let's analyze it to see what the main challenge is.

Divide and Conquer

Algorithm 2 Divide and Conquer

```
function SOLVE($G$)
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```

- Let's analyze it to see what the main challenge is.
- The first step of our algorithm is to sort all edges in G. Divide it into two disjoint subgraphs. All edge weights in $G_{[0]}$ is smaller than edges weights in $G_{[1]}$.

Relax High-degree edges

Algorithm 3 Divide and Conquer

```
function $\textsc{Solve}(G)$
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```

- Relaxation of edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$ is exactly one step of $(min, \leq)$-product.
- Each time, the paths are extended by one edge.

Divide and Conquer

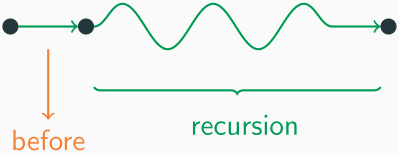
Algorithm 4 Divide and Conquer

```
function SOLVE($G$)
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```

Divide and Conquer

Algorithm 4 Divide and Conquer

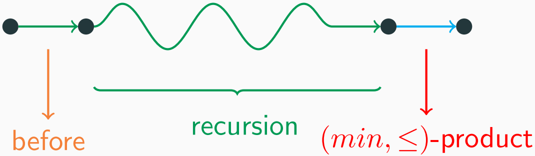
```
function SOLVE($G$)
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```



Divide and Conquer

Algorithm 4 Divide and Conquer

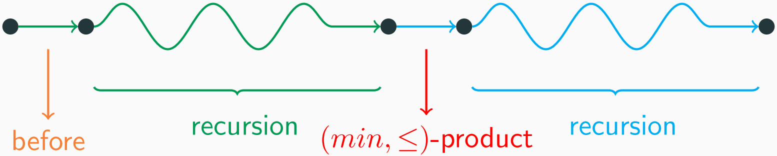
```
function SOLVE($G$)
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```



Divide and Conquer

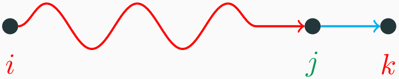
Algorithm 4 Divide and Conquer

```
function Solve($G$)
    Divide Graph $G$ into $G_{[0]}$ and $G_{[1]}$ according to edge weight
    Solve($G_{[0]}$)
    Relax high-low edges and high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```



Main Challenge : Complexity

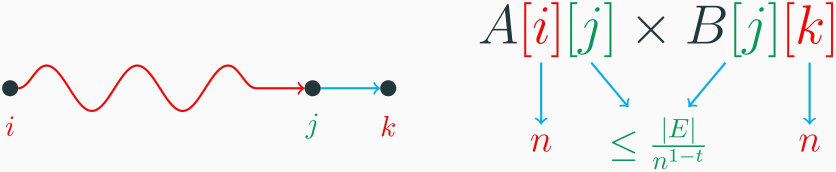
- Why this procedure won't work?
  - We need to handle high-low edges and high-high edges at same time with matrix mutliplication.



$$A[\underset{\substack{\downarrow\\ n}}{i}][\underset{\substack{\searrow\\ \le \frac{|E|}{n^{1-t}}}}{j}] \times B[\underset{\swarrow}{j}][\underset{\substack{\downarrow\\ n}}{k}]$$

Main Challenge : Complexity

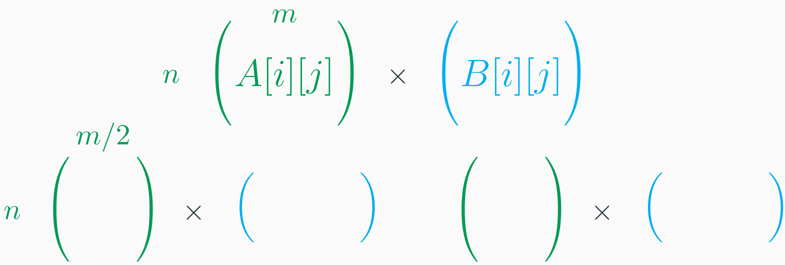
- Why this procedure won't work?
  - We need to handle high-low edges and high-high edges at same time with matrix mutliplication.
  - Each level of recursion the second dimension of matrix mutliplication is divided by 2.



$$A[i][j] \times B[j][k]$$

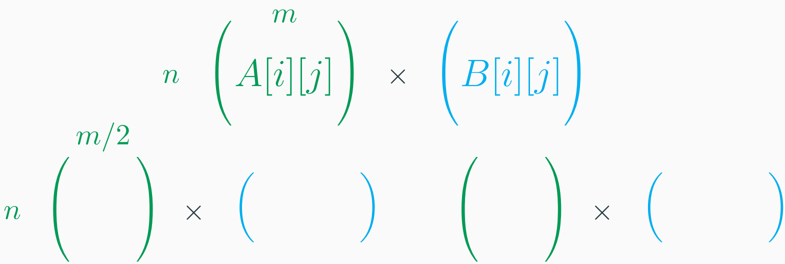
$$n \qquad \leq \frac{|E|}{n^{1-t}} \qquad n$$

Main Challenge : Complexity



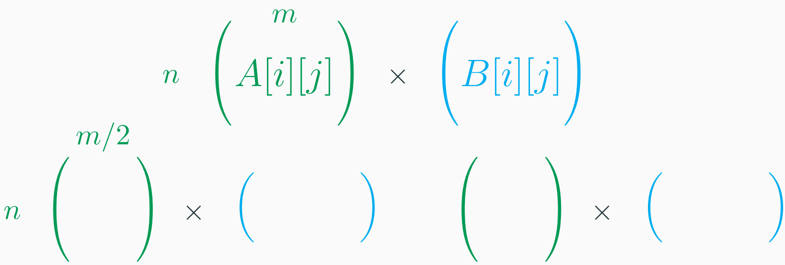
- The second dimension is divided by 2. For bruteforce $\Theta(n^3)$ matrix multiplication, the complexity of matrix multiplication is divide by 2 as well.

Main Challenge : Complexity



- The second dimension is divided by 2. For bruteforce $\Theta(n^3)$ matrix multiplication, the complexity of matrix multiplication is divide by 2 as well.
- But for $\Theta(n^\omega)$ fast square matrix mutliplication, the complexity is divided by some constant less than 2.

Main Challenge : Complexity



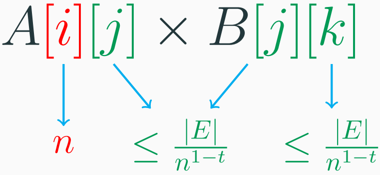
- The second dimension is divided by 2. For bruteforce $\Theta(n^3)$ matrix multiplication, the complexity of matrix multiplication is divide by 2 as well.
- But for $\Theta(n^\omega)$ fast square matrix mutliplication, the complexity is divided by some constant less than 2.
- Difficulty: The number of subproblems grows faster!

New idea for high-low edges

- We came up with a new technique for high-low edges.
- Thus in each layer of recursion we only have to care about high-high edges.

New idea for high-low edges

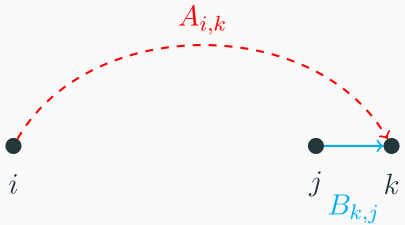
- We came up with a new technique for high-low edges.
- Thus in each layer of recursion we only have to care about high-high edges.
- Both dimensions are divided by 2 in recursion now.



$$A[i][j] \times B[j][k]$$

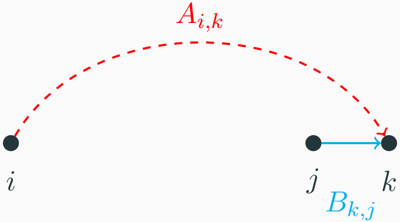
$$n \qquad \le \frac{|E|}{n^{1-t}} \qquad \le \frac{|E|}{n^{1-t}}$$

New idea for High-low edges



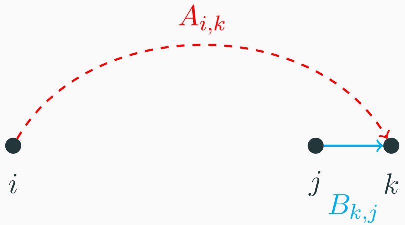
- If there is an optimal nondecreasing path $i \to k$ with a high-low edge as its last edge, we can enumerate all in-coming edges of k to find it.

New idea for High-low edges



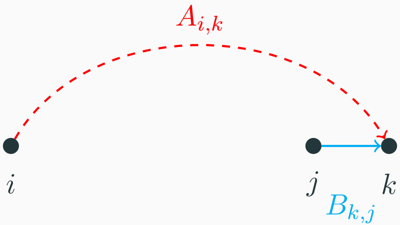
- We need an oracle to “predict” the existence of such path $i \to k$.

New idea for High-low edges



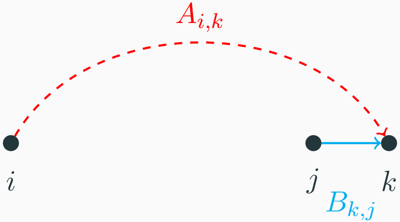
- We need an oracle to "predict" the existence of such path $i \to k$.
    - $A_{i,k} = 1$ if we haven't found path from i to k.

New idea for High-low edges



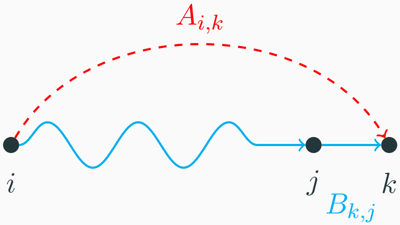
- We need an oracle to "predict" the existence of such path $i \to k$.
  - $A_{i,k} = 1$ if we haven't found path from i to k.
  - $B_{k,j} = 1$ if there is an edge (j, k).

New idea for High-low edges



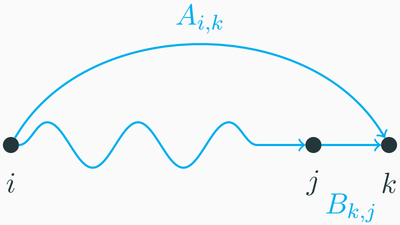
- We need an oracle to “predict” the existence of such path $i \to k$.
    - $A_{i,k} = 1$ if we haven’t found path from i to k.
    - $B_{k,j} = 1$ if there is an edge (j, k).
    - We compute $C_{i,j} = \sum_k A_{i,k} B_{k,j}$

New idea for High-low edges



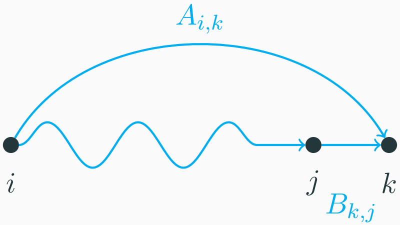
- When we visit path $i \to j$ and $C_{i,j} > 0$, we then enumerate all outgoing edges of j to update path $i \to k$.

New idea for High-low edges



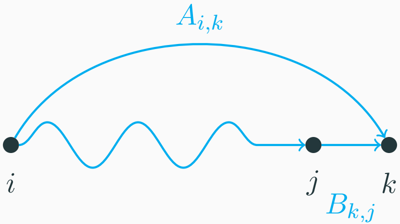
- When we visit path $i \to j$ and $C_{i,j} > 0$, we then enumerate all outgoing edges of j to update path $i \to k$.

New idea for High-low edges



- When we visit path $i \to j$ and $C_{i,j} > 0$, we then enumerate all outgoing edges of j to update path $i \to k$.
- What if j has high degree ?

New idea for High-low edges



- When we visit path $i \to j$ and $C_{i,j} > 0$, we then enumerate all outgoing edges of j to update path $i \to k$.
- After we find a nondecreasing path $i \to k$, we enumerate incoming edges (j', k) of k for two purposes:
  - Find the optimal nondecreasing path $i \to k$.
  - Decrease $C_{i,j'}$ by one, so we won't enumerate for the same path $i \to k$ twice.

New idea for High-low edges

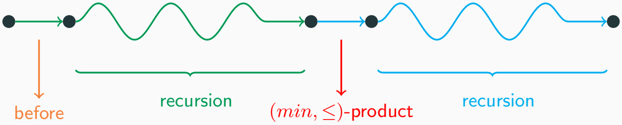
- What is the point of this technique if it still needs matrix multiplication?

New idea for High-low edges

- What is the point of this technique if it still needs matrix multiplication?
    - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.

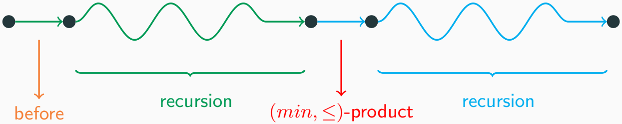
New idea for High-low edges

- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



New idea for High-low edges

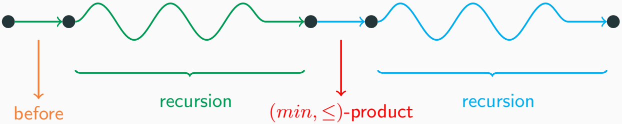
- What is the point of this technique if it still needs matrix multiplication?
    - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
        - High-high edges



        - Low edges / High-low edges

New idea for High-low edges

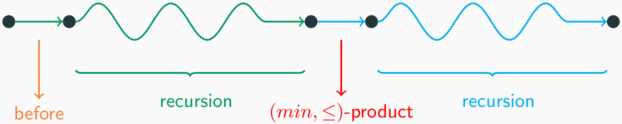
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

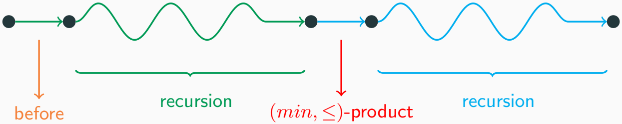
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

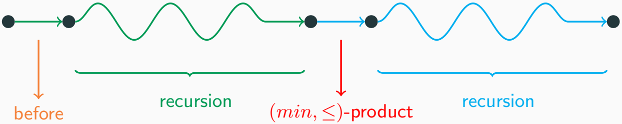
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

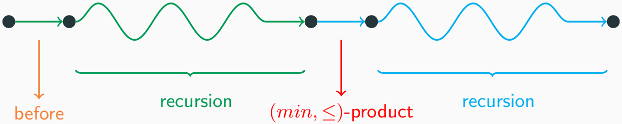
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is “dynamic” and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

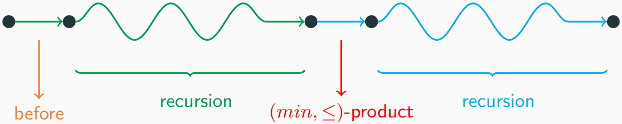
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is “dynamic” and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

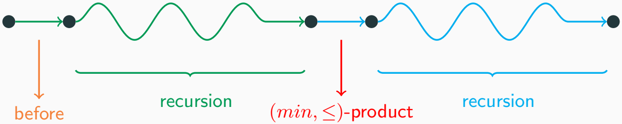
- What is the point of this technique if it still needs matrix multiplication?
    - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
        - High-high edges



        - Low edges / High-low edges

New idea for High-low edges

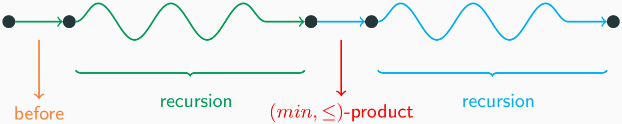
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

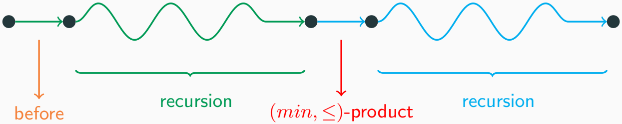
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

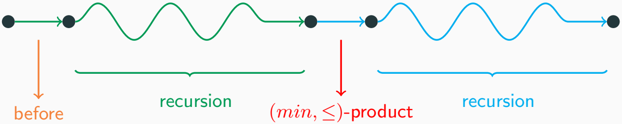
- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges



    - Low edges / High-low edges

New idea for High-low edges

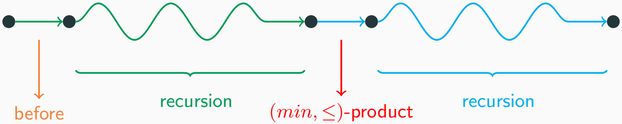
- What is the point of this technique if it still needs matrix multiplication?
    - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is “dynamic” and friendly to sequential updates.
        - High-high edges



        - Low edges / High-low edges

New idea for High-low edges

- What is the point of this technique if it still needs matrix multiplication?
  - This technique can relax all high-low edges without recursion! Unlike $(min, \leq)$-product, it is "dynamic" and friendly to sequential updates.
    - High-high edges

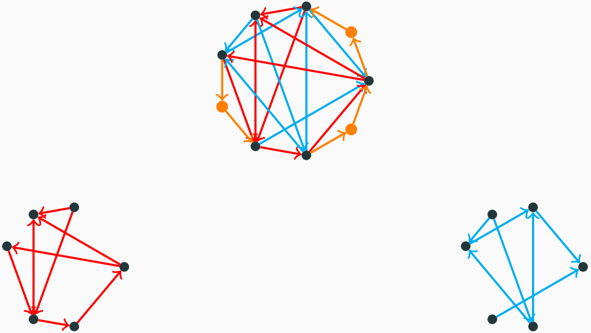


    - Low edges / High-low edges

  - In each layer of recursion, since low edges and high-low edges are already handled, we only keep those high degree vertices to next layer!

Divide and Conquer

We only divide the induced subgraph of high degree vertices.



- As the graph is getting sparser, the nubmer of vertices decrease. The third dimension of matrix mutliplication also decreasing now!

Our algorithm

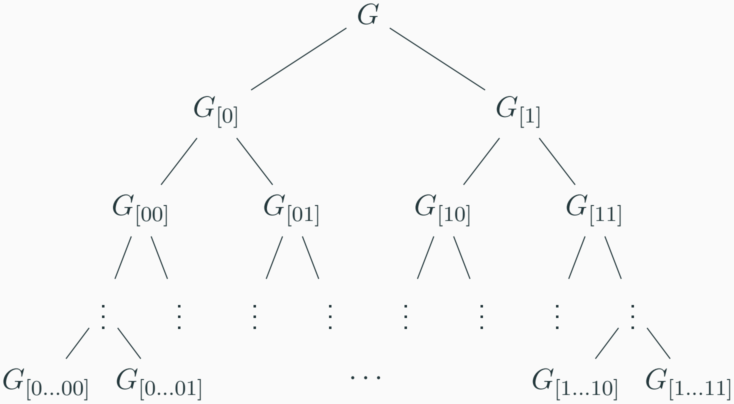
Algorithm 5 Divide and Conquer

```
function Solve($G$)
    Run the matrix multiplication for high-low edges
    Divide the induced graph of high vertices into $G_{[0]}, G_{[1]}$
    Solve($G_{[0]}$)
    Relax high-high edges in $G_{[1]}$ w.r.t. paths ends in $G_{[0]}$
    Solve($G_{[1]}$)
end function
```

- We relax low edges and high-low edges when we visit path $i \to j$.
- So they are relaxed at the leaves of the recursion.

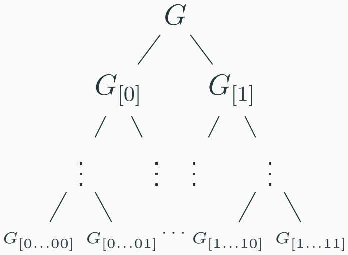
Our algorithm

- The recursion tree looks like following:



- When we reach a leaf, we "visit" the path of that weight.
- It is still a Dijkstra Search.

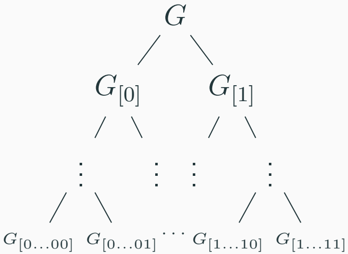
Time Complexity



#edges	**#high vertices**	**Complexity**
n^2	n	n^ω
$n^2/2$	n	$2n^\omega$
$\vdots$	$\vdots$	$\vdots$
n^{2-t}	n	$n^{t+\omega}$
$n^{2-t}/2$	$n/2$	$< n^{t+\omega}$
$\vdots$	$\vdots$	$\vdots$

- Enumeration takes $O(n^{3-t})$ time, since each pair of vertices is responsible for $O(n^{1-t})$ enumeration.

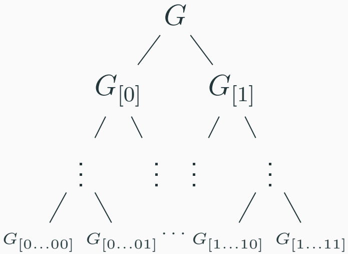
Time Complexity



#edges	**#high vertices**	**Complexity**
n^2	n	n^ω
$n^2/2$	n	$2n^\omega$
$\vdots$	$\vdots$	$\vdots$
n^{2-t}	n	$n^{t+\omega}$
$n^{2-t}/2$	$n/2$	$< n^{t+\omega}$
$\vdots$	$\vdots$	$\vdots$

- Enumeration takes $O(n^{3-t})$ time, since each pair of vertices is responsible for $O(n^{1-t})$ enumeration.
- When the number of edges is less than n^{2-t}, the number of high vertices starts decrease linearly.

Time Complexity



#edges	**#high vertices**	**Complexity**
n^2	n	n^ω
$n^2/2$	n	$2n^\omega$
$\vdots$	$\vdots$	$\vdots$
n^{2-t}	n	$n^{t+\omega}$
$n^{2-t}/2$	$n/2$	$< n^{t+\omega}$
$\vdots$	$\vdots$	$\vdots$

- Enumeration takes $O(n^{3-t})$ time, since each pair of vertices is responsible for $O(n^{1-t})$ enumeration.
- When the number of edges is less than n^{2-t}, the number of high vertices starts decrease linearly.
  - So the maximum complexity of matrix mutliplication for each layer is $O(n^{t+\omega})$

Conclusion

Conclusion & Open problems

- APNP algorithm in $\tilde{O}(n^{\frac{3+\omega}{2}})$ time.

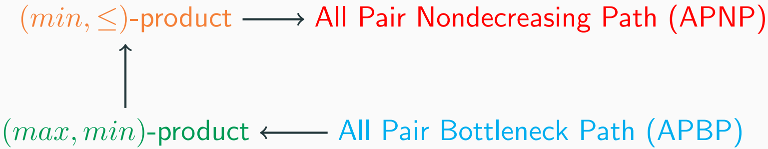
Conclusion & Open problems

- APNP algorithm in $\tilde{O}(n^{\frac{3+\omega}{2}})$ time.
- All these problem now have best running algorithm in time $\tilde{O}(n^{\frac{3+\omega}{2}})$.

$$\begin{tikzcd}
(min, \leq)\text{-product} \arrow[r] & \text{All Pair Nondecreasing Path (APNP)} \\
(max, min)\text{-product} \arrow[u] & \text{All Pair Bottleneck Path (APBP)} \arrow[l]
\end{tikzcd}$$

Conclusion & Open problems

- APNP algorithm in $\tilde{O}(n^{\frac{3+\omega}{2}})$ time.
- All these problem now have best running algorithm in time $\tilde{O}(n^{\frac{3+\omega}{2}})$.



$(min, \leq)$-product ⟶ All Pair Nondecreasing Path (APNP)

↑

(max, min)-product ⟵ All Pair Bottleneck Path (APBP)

- Is there faster algoirthm for these problems? Can we show some lower bounds for these porblems?

Q & A

Questions?

Thank you!