Fast matrix multiplication and graph algorithms

Uri Zwick
Tel Aviv University

NHC Autumn School on Discrete Algorithms
Sunparea Seto, Seto, Aichi Nov. 15-17, 2006

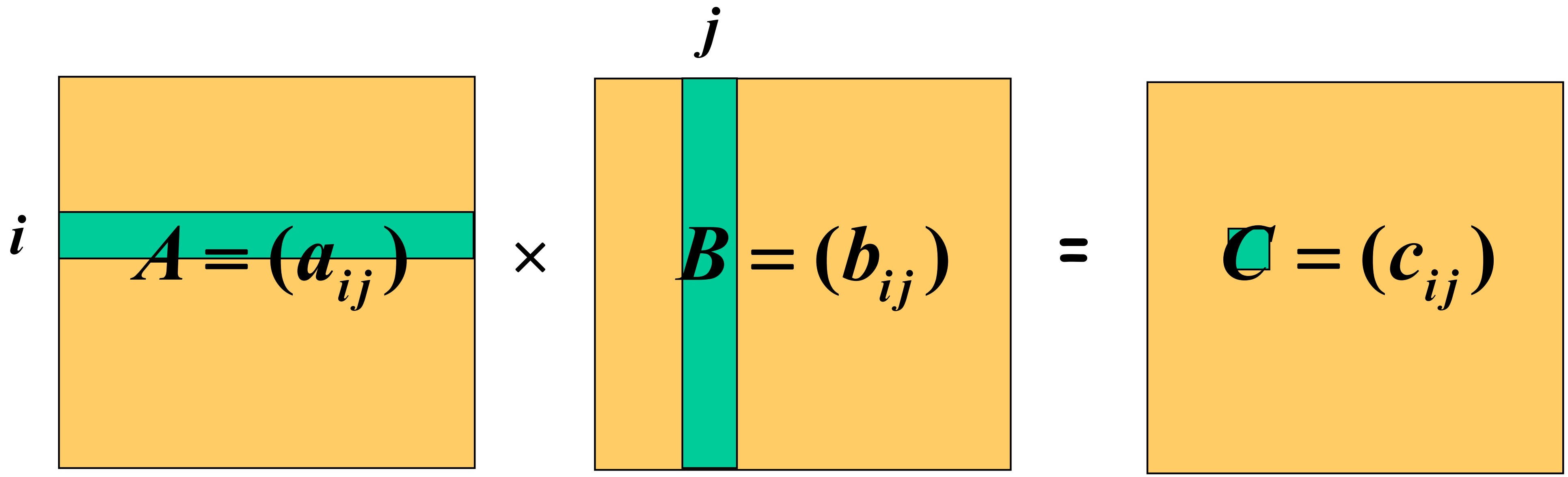
新世代の計算限界

Overview

- Short introduction to fast matrix multiplication
- Transitive closure
- Shortest paths in undirected graphs
- Shortest paths in directed graphs
- Perfect matchings

Short introduction to
Fast matrix multiplication

Algebraic Matrix Multiplication



$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Can be computed naively in $O(n^3)$ time.

Matrix multiplication algorithms

Complexity	**Authors**
n^3	(by definition)
$n^{2.81}$	**Strassen (1969)**
$n^{2.38}$	**Coppersmith, Winograd (1990)**

Conjecture/Open problem: $n^{2+o(1)}$???

Multiplying 2×2 matrices

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

8 multiplications
4 additions

$$T(n) = 8\, T(n/2) + O(n^2)$$

$$T(n) = O(n^{\log 8/\log 2}) = O(n^3)$$

Strassen's 2×2 algorithm

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Subtraction!

$$M_1 = (A$$
$$M_2 = (A_{21} + \quad B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

7 multiplications
18 additions/subtractions

Strassen's $n \times n$ algorithm

View each $n \times n$ matrix as a 2×2 matrix whose elements are $n/2 \times n/2$ matrices.

Apply the 2×2 algorithm recursively.

$$T(n) = 7\ T(n/2) + O(n^2)$$

$$T(n) = O(n^{\log 7/\log 2}) = O(n^{2.81})$$

Matrix multiplication algorithms

The $O(n^{2.81})$ bound of Strassen was improved by Pan, Bini-Capovani-Lotti-Romani, Schönhage and finally by Coppersmith and Winograd to $O(n^{2.38})$.

The algorithms are much more complicated…

We let $2 \leq \omega < 2.38$ be the exponent of matrix multiplication.

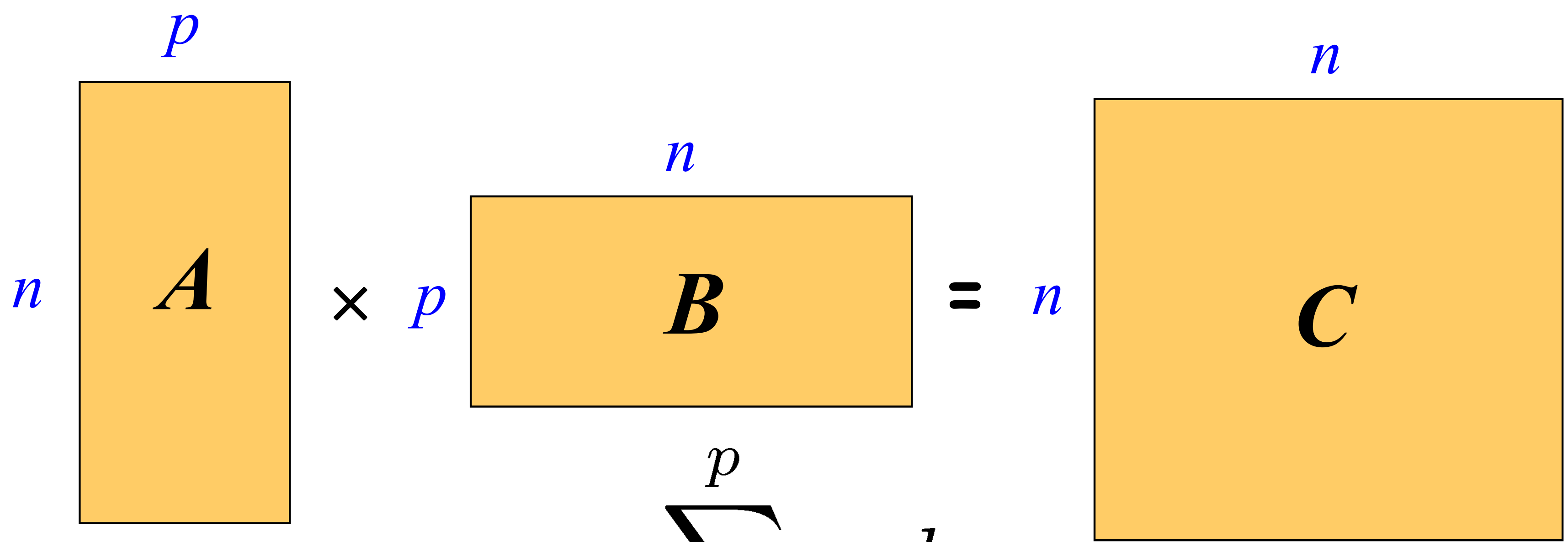
Gaussian elimination

The title of Strassen's 1969 paper is:
"Gaussian elimination is not optimal"

Other matrix operations that can
be performed in $O(n^{\omega})$ time:

- Computing determinants: $\det A$.
- Computing inverses: A^{-1}
- Computing **characteristic** polynomials

Rectangular Matrix multiplication



$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

Coppersmith (1997):

Complexity $\leq n^{1.85} p^{0.54} + n^{2+o(1)}$

For $p \leq n^{0.29}$, complexity $= n^{2+o(1)}$!!!

TRANSIVE CLOSURE

Transitive Closure

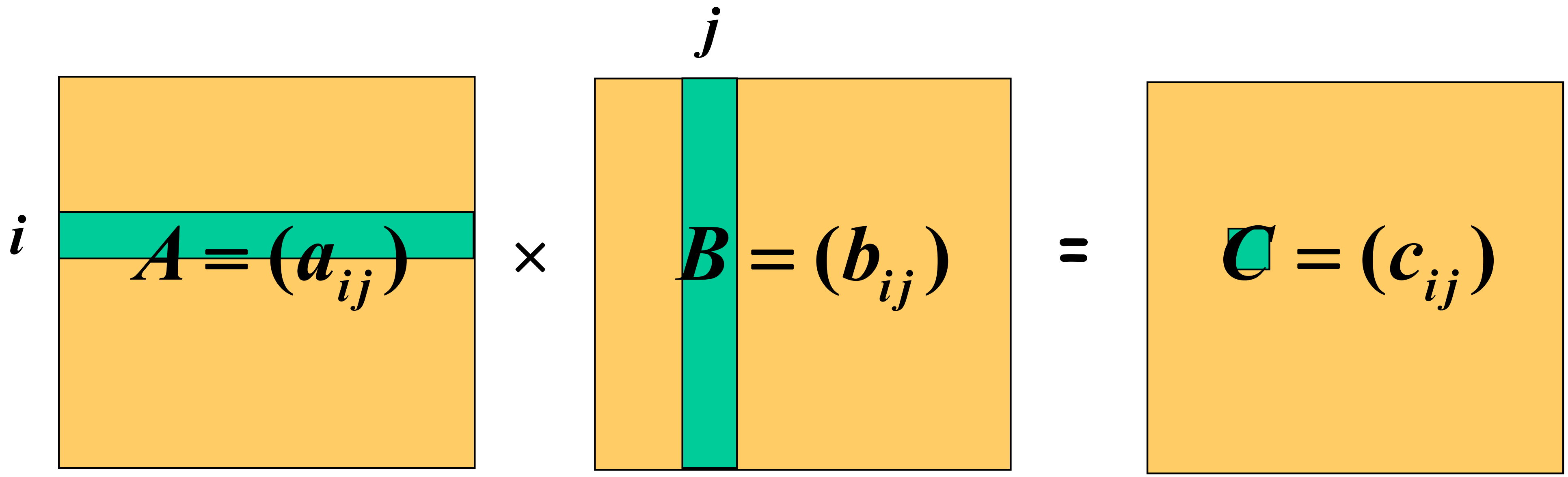
Let $G=(V,E)$ be a directed graph.

The transitive closure $G^*=(V,E^*)$ is the graph in which $(u,v)\in E^*$ iff there is a path from u to v.

Can be easily computed in $O(mn)$ time.

Can also be computed in $O(n^{\omega})$ time.

Boolean Matrix Multiplication



$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

Can be computed naively in O(n^3) time.

Algebraic Product

$$C = AB$$

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

$O(n^{2.38})$

algebraic operations

Boolean Product

$$C = A \cdot B$$

$$c_{ij} = \bigvee_k a_{ik} \wedge b_{kj}$$

?

Algebraic Product

$$C = A\,B$$

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

$O(n^{2.38})$

algebraic operations

Boolean Product

$$C = A \cdot B$$

$$c_{ij} = \bigvee_k a_{ik} \wedge b_{kj}$$

or ($\vee$) has no inverse!

Algebraic Product

$$C = AB$$

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

$O(n^{2.38})$ algebraic operations

Boolean Product

$$C = A \cdot B$$

$$c_{ij} = \bigvee_k a_{ik} \wedge b_{kj}$$

But, we can work over the **integers**!

Algebraic Product

$$C = A\,B$$

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

$\mathbf{O}(n^{2.38})$

algebraic operations

Boolean Product

$$C = A \cdot B$$

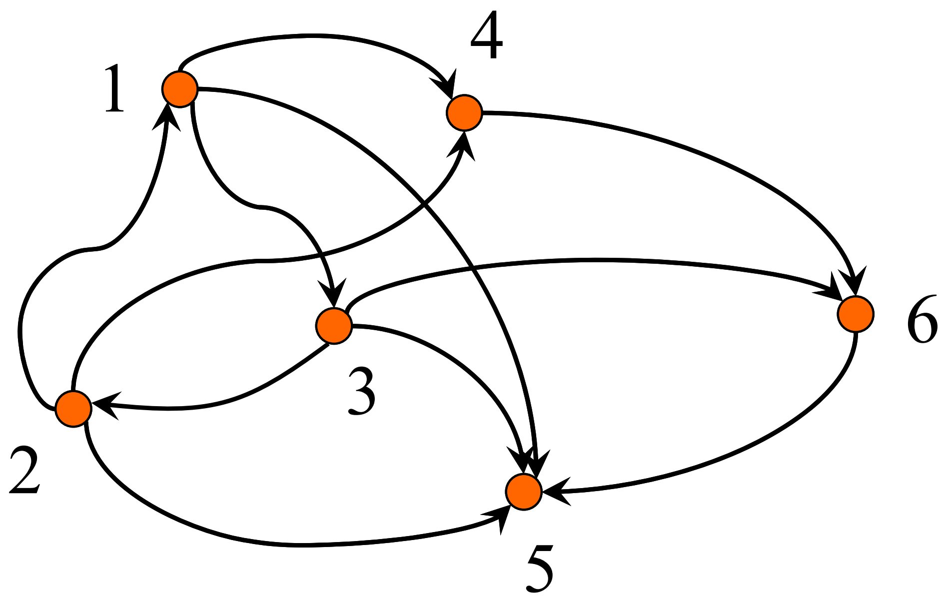
$$c_{ij} = \bigvee_k a_{ik} \wedge b_{kj}$$

$\mathbf{O}(n^{2.38})$

operations on $O(\log n)$ bit words

- Can you use Strassen's algorithm or the Coppersmith-Winograd algorithm to compute **Boolean** matrix multiplications?
- No, as these algebraic algorithms use **subtractions** and the Boolean-or ($\vee$) operation has no inverse!
- Still, we can run the algebraic algorithms over the integers and convert any non-zero result to 1.

Adjacency matrix of a directed graph



$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Exercise 0: If A is the adjacency matrix of a graph, then $(A^k)_{ij}=1$ iff there is a path of length k from i to j.

Transitive Closure using matrix multiplication

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

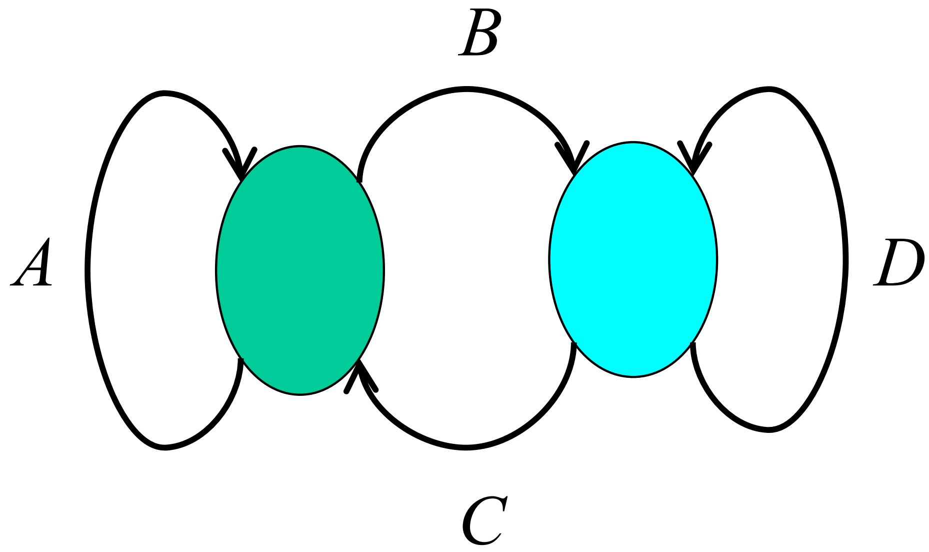
The transitive closure $G^*=(V,E^*)$ is the graph in which $(u,v)\in E^*$ iff there is a path from u to v.

If A is the adjacency matrix of G, then $(A\vee I)^{n-1}$ is the adjacency matrix of G^*.

The matrix $(A\vee I)^{n-1}$ can be computed by $\log n$ squaring operations in $O(n^\omega \log n)$ time.

It can also be computed in $O(n^\omega)$ time.

$$X = \begin{array}{|c|c|} \hline A & B \\ \hline C & D \\ \hline \end{array}$$



$$X^* = \begin{array}{|c|c|} \hline E & F \\ \hline G & H \\ \hline \end{array} = \begin{array}{|c|c|} \hline (A \vee BD^*C)^* & EBD^* \\ \hline D^*CE & D^* \vee GBD^* \\ \hline \end{array}$$

$$\mathbf{TC}(n) \le \mathbf{2}\ \mathbf{TC}(n/2) + \mathbf{6}\ \mathbf{BMM}(n/2) + \mathbf{O}(n^2)$$

Exercise 1: Give $O(n^{\omega})$ algorithms for findning, in a directed graph,

a) a triangle
b) a simple quadrangle
c) a simple cycle of length k.

Hints:

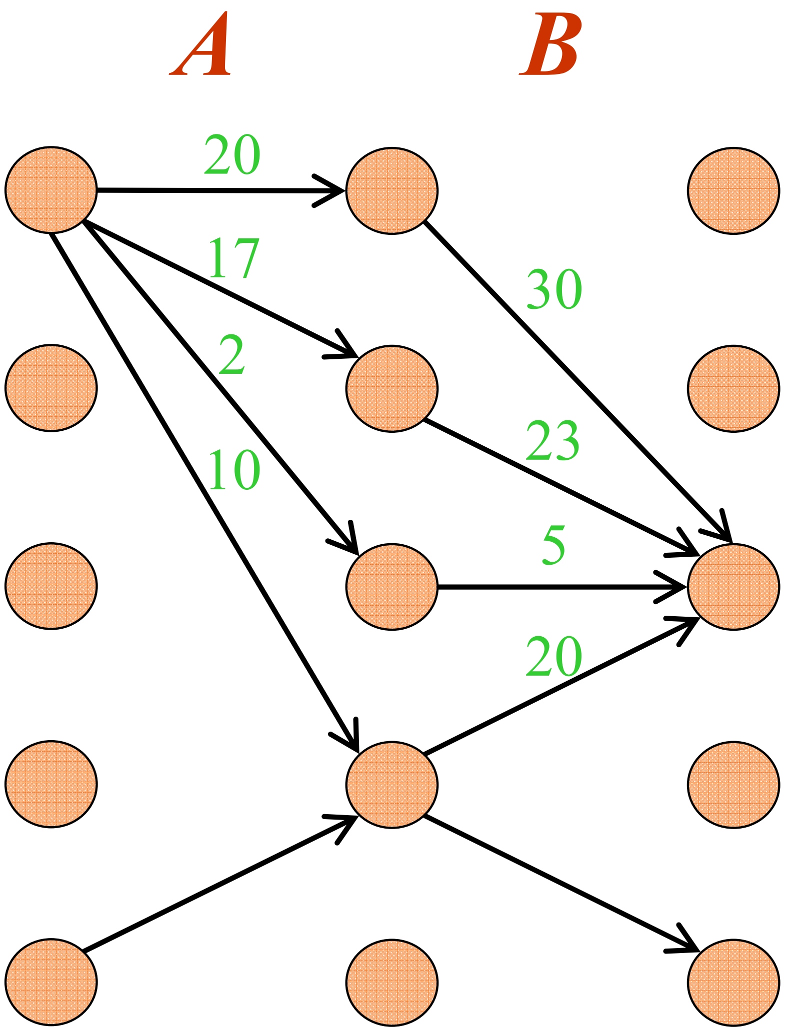
1. In an **acyclic** graph all paths are simple.
2. In c) running time may be **exponential** in k.
3. **Randomization** makes solution much easier.

SHORTEST PATHS

APSP – All-Pairs Shortest Paths

SSSP – Single-Source Shortest Paths

An interesting special case of the APSP problem



$$C = A * B$$

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}$$

Min-Plus product

Min-Plus Products

$$C = A * B$$

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}$$

$$\begin{pmatrix} -6 & -3 & -10 \\ 2 & 5 & -2 \\ -1 & -7 & -5 \end{pmatrix} = \begin{pmatrix} 1 & -3 & 7 \\ +\infty & 5 & +\infty \\ 8 & 2 & -5 \end{pmatrix} * \begin{pmatrix} 8 & +\infty & -4 \\ -3 & 0 & -7 \\ 5 & -2 & 1 \end{pmatrix}$$

Solving APSP by repeated squaring

If W is an n by n matrix containing the edge weights of a graph. Then W^n is the distance matrix.

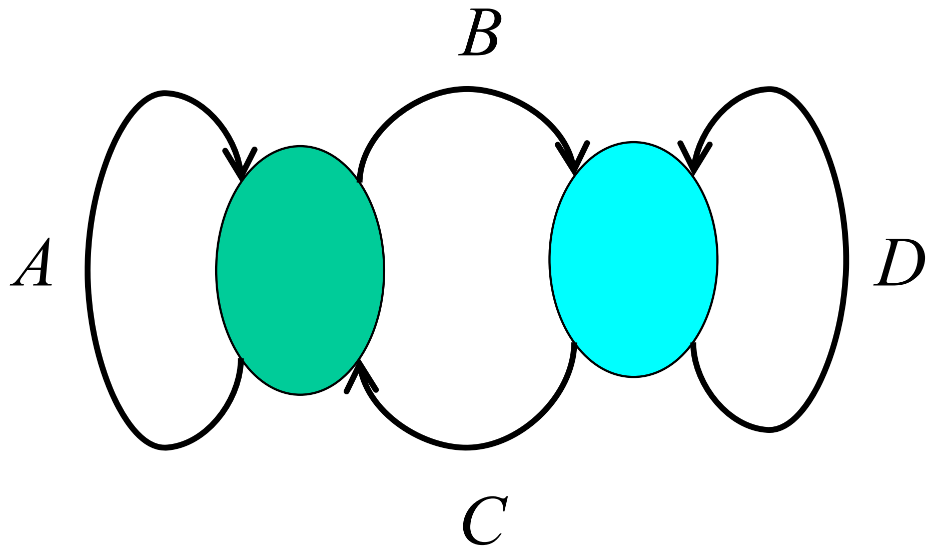
By induction, W^k gives the distances realized by paths that use at most k edges.

$$D \leftarrow W$$
$$\textbf{for } i \leftarrow 1 \textbf{ to } \lceil \log_2 n \rceil$$
$$\textbf{do } D \leftarrow D * D$$

Thus: $\mathrm{APSP}(n) \le \mathrm{MPP}(n) \log n$

Actually: $\mathrm{APSP}(n) = O(\mathrm{MPP}(n))$

$$X = \begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline\end{array}$$



$$X^* = \begin{array}{|c|c|}\hline E & F \\ \hline G & H \\ \hline\end{array} = \begin{array}{|c|c|}\hline (A \vee BD^*C)^* & EBD^* \\ \hline D^*CE & D^* \vee GBD^* \\ \hline\end{array}$$

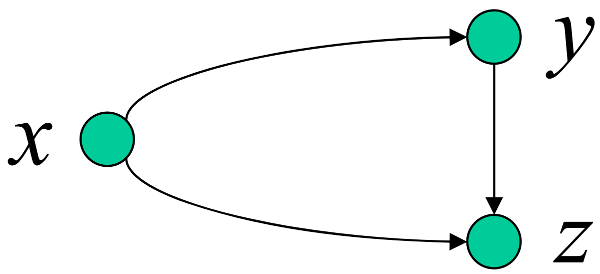
APSP(n) $\leq$ 2 APSP($n/2$) + 6 MPP($n/2$) + O(n^2)

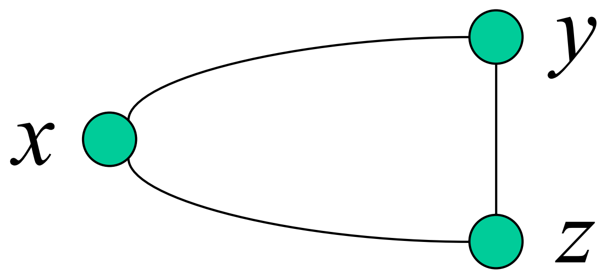
Algebraic Product	Min-Plus Product
$C = A \cdot B$	$C = A * B$
$c_{ij} = \sum_k a_{ik} b_{kj}$	$c_{ij} = \min_k \{ a_{ik} + b_{kj} \}$
$O(n^{2.38})$	min operation has no inverse!

?

UNWEIGHTED
UNDIRECTED
SHORTEST PATHS

Directed versus undirected graphs





$$\delta(x,z) \leq \delta(x,y) + \delta(y,z)$$

Triangle inequality

$$\delta(x,z) \leq \delta(x,y) + \delta(y,z)$$

$$\delta(x,y) \leq \delta(x,z) + \delta(z,y)$$

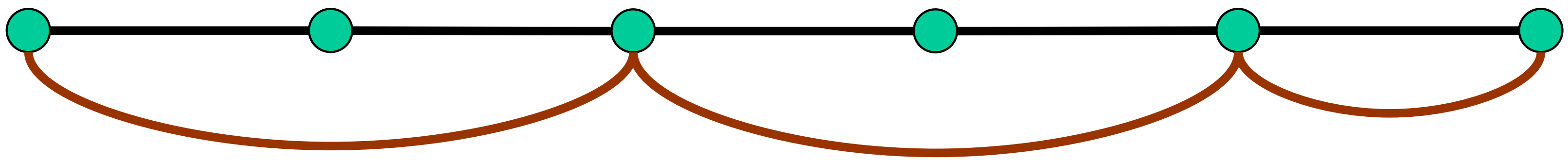
$$\delta(x,z) \geq \delta(x,y) - \delta(y,z)$$

Inverse triangle inequality

Distances in G and its square G^2

Let $G=(V,E)$. Then $G^2=(V,E^2)$, where $(u,v)\in E^2$ if and only if $(u,v)\in E$ or there exists $w\in V$ such that $(u,w),(w,v)\in E$

Let $\delta(u,v)$ be the distance from u to v in G.
Let $\delta^2(u,v)$ be the distance from u to v in G^2.



$\delta(u,v)=5$ $\delta^2(u,v)=3$

Distances in G and its square G^2 (cont.)

$$\delta^2(u,v) \le \lceil \delta(u,v)/2 \rceil$$

$$\delta(u,v) \le 2\delta^2(u,v)$$

Lemma: $\delta^2(u,v) = \lceil \delta(u,v)/2 \rceil$, for every $u,v \in V$.

Thus: $\delta(u,v) = 2\delta^2(u,v)$ or

$\delta(u,v) = 2\delta^2(u,v) - 1$

Distances in G and its square G^2 (cont.)

Lemma: If $\delta(u,v)=2\delta^2(u,v)$ then for every neighbor w of v we have $\delta^2(u,w) \geq \delta^2(u,v)$.

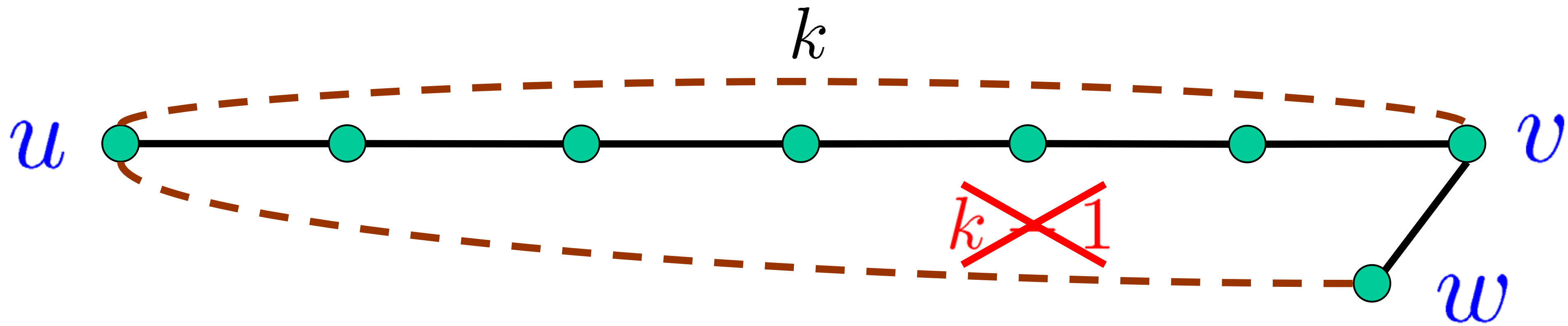
Lemma: If $\delta(u,v)=2\delta^2(u,v)-1$ then for every neighbor w of v we have $\delta^2(u,w) \leq \delta^2(u,v)$ and for at least one neighbor $\delta^2(u,w) < \delta^2(u,v)$.

Let A be the adjacency matrix of the G.
Let C be the distance matrix of G^2

$$\sum_{(v,w)\in E} c_{u,w} = \sum_{w} c_{u,w} a_{w,v} = (CA)_{u,v} \quad : \quad \deg(v)\, c_{u,v}$$

Even distances

Lemma: If $\delta(u,v)=2\delta^2(u,v)$ then for every neighbor w of v we have $\delta^2(u,w) \geq \delta^2(u,v)$.



Let A be the adjacency matrix of the G.
Let C be the distance matrix of G^2

$$\sum_{(v,w)\in E} c_{uw} = \sum_{w\in V} c_{uw} a_{wv} = (CA)_{uv} \geq \deg(v) c_{uv}$$

Odd distances

Lemma: If $\delta(u,v)=2\delta^2(u,v)-1$ then for every neighbor w of v we have $\delta^2(u,w) \le \delta^2(u,v)$ and for at least one neighbor $\delta^2(u,w) < \delta^2(u,v)$.

Exercise 2: Prove the lemma.

Let A be the adjacency matrix of the G.
Let C be the distance matrix of G^2

$$\sum_{(v,w)\in E} c_{uw} = \sum_{w\in V} c_{uw} a_{wv} = (CA)_{uv} < \deg(v) c_{uv}$$

Seidel's algorithm

1. If A is an all one matrix, then all distances are 1.
2. Compute A^2, the adjacency matrix of the squared graph.
3. Find, recursively, the distances in the squared graph.
4. Decide, using one integer matrix multiplication, for every two vertices u,v, whether their distance is **twice** the distance in the square, or **twice minus 1**.

Seide

Assume that A has 1's on the diagonal.

1. If A is an all one matrix, then all distances are 1.
2. Compute A^2, the adjacency matrix of the squared graph.
3. Find, recursively, the distances in the squared graph.
4. Decide, using one integer matrix multiplication, for every two vertices u,v, whether their distance is **twice** the distance in the square, or **twice minus 1**.

Boolean matrix multiplicaion

Integer matrix multiplicaion

Seidel's algorithm

1. If A is an all one matrix, then all distances are 1.
2. Compute A^2, the adjacency matrix of the squared graph.
3. Find, recursively, the distances in the squared graph.
4. Decide, using one integer matrix multiplication, for every two vertices u,v, whether their distance is **twice** the distance in the square, or **twice minus 1**.

Algorithm APD(A)
if $A=J$ then
  return $J-I$
else
  $C \leftarrow \mathrm{APD}(A^2)$
  $X \leftarrow CA$, $\mathrm{deg} \leftarrow Ae-1$
  $d_{ij} \leftarrow 2c_{ij} - [x_{ij} < c_{ij}\mathrm{deg}_j]$
  return D
end

Complexity:
$O(n^{\omega} \log n)$

Exercise 3: (*) Obtain a version of Seidel's algorithm that uses only **Boolean** matrix multiplications.

Hint: Look at distances also modulo 3.

Distances vs. Shortest Paths

We described an algorithm for computing all **distances**.

How do we get a representation of the **shortest paths**?

We need **witnesses** for the Boolean matrix multiplication.

Witnesses for Boolean Matrix Multiplication

$$C = AB$$

$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

A matrix W is a matrix of **witnesses** iff

$$\text{If } c_{ij} = 0 \text{ then } w_{ij} = 0$$

$$\text{If } c_{ij} = 1 \text{ then } w_{ij} = k \text{ where } a_{ik} = b_{kj} = 1$$

Can be computed naively in $O(n^3)$ time.

Can also be computed in $O(n^{\omega} \log n)$ time.

Exercise 4:

a) Obtain a deterministic $O(n^{\omega})$-time algorithm for finding **unique** witnesses.

b) Let $1 \leq d \leq n$ be an integer. Obtain a randomized $O(n^{\omega})$-time algorithm for finding witnesses for all positions that have between d and $2d$ witnesses.

c) Obtain an $O(n^{\omega} \log n)$-time algorithm for finding all witnesses.

Hint: In b) use **sampling**.

All-Pairs Shortest Paths
in graphs with small integer weights

Undirected graphs.
Edge weights in $\{0,1,\ldots M\}$

Running time	**Authors**
Mn^{ω}	[Shoshan-Zwick '99]

Improves results of
[Alon-Galil-Margalit '91] [Seidel '95]

DIRECTED

SHORTEST PATHS

Exercise 5: Obtain an $O(n^\omega \log n)$ time algorithm for computing the **diameter** of an unweighted directed graph.

PERFECT MATCHINGS

Using matrix multiplication to compute min-plus products

$$\begin{pmatrix} c_{11} & c_{12} & \\ c_{21} & c_{22} & \\ & & \ddots \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \\ a_{21} & a_{22} & \\ & & \ddots \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} & \\ b_{21} & b_{22} & \\ & & \ddots \end{pmatrix}$$

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}$$

$$\begin{pmatrix} c'_{11} & c'_{12} & \\ c'_{21} & c'_{22} & \\ & & \ddots \end{pmatrix} = \begin{pmatrix} x^{a_{11}} & x^{a_{12}} & \\ x^{a_{21}} & x^{a_{22}} & \\ & & \ddots \end{pmatrix} \times \begin{pmatrix} x^{b_{11}} & x^{b_{12}} & \\ x^{b_{21}} & x^{b_{22}} & \\ & & \ddots \end{pmatrix}$$

$$c'_{ij} = \sum_k x^{a_{ik}+b_{kj}} \qquad c_{ij} = \mathrm{first}(c'_{ij})$$

Using matrix multiplication to compute min-plus products

Assume: $0 \le a_{ij}, b_{ij} \le M$

$$\begin{pmatrix} c'_{11} & c'_{12} & \\ c'_{21} & c'_{22} & \\ & & \ddots \end{pmatrix} = \begin{pmatrix} x^{a_{11}} & x^{a_{12}} & \\ x^{a_{21}} & x^{a_{22}} & \\ & & \ddots \end{pmatrix} * \begin{pmatrix} x^{b_{11}} & x^{b_{12}} & \\ x^{b_{21}} & x^{b_{22}} & \\ & & \ddots \end{pmatrix}$$

n^{ω} polynomial products	$\times$	M operations per polynomial product	$=$	Mn^{ω} operations per max-plus product

Trying to implement the repeated squaring algorithm

$D \leftarrow W$
for $i \leftarrow 1$ to $\log_2 n$
do $D \leftarrow D*D$

Consider an easy case: all weights are 1.

After the i-th iteration, the finite elements in D are in the range $\{1,\ldots,2^i\}$.

The cost of the min-plus product is $2^i n^\omega$

The cost of the last product is $n^{\omega+1}$!!!

Sampled Repeated Squaring (Z '98)

$D \leftarrow W$

for $i \leftarrow 1$ to $\log_{3/2} n$ do

{

$\quad s \leftarrow (3/2)^{i+1}$

$\quad B \leftarrow \text{rand}(\, V , (9n \ln n)/s \,)$

$\quad D \leftarrow \min\{\, D , D[V,B]*D[B,V] \,\}$

}

Choose a subset of V of size $(9n \ln n)/s$

Select the **columns** of D whose indices are in B

Select the **rows** of D whose indices are in B

Sampled Repeated Squaring (Z '98)

$D \leftarrow W$

for $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**

{

$\quad s \leftarrow (3/2)^{i+1}$

$\quad B \leftarrow \mathrm{rand}(\, V, (9n \ln n)/s\,)$

$\quad D \leftarrow \min\{\, D\,,\, D[V,B] * D[B,V]\,\}$

}

With high probability, all distances are correct!

Sampled Repeated Squaring (Z '98)

$D \leftarrow W$

for $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**

{

$\quad s \leftarrow (3/2)^{i+1}$

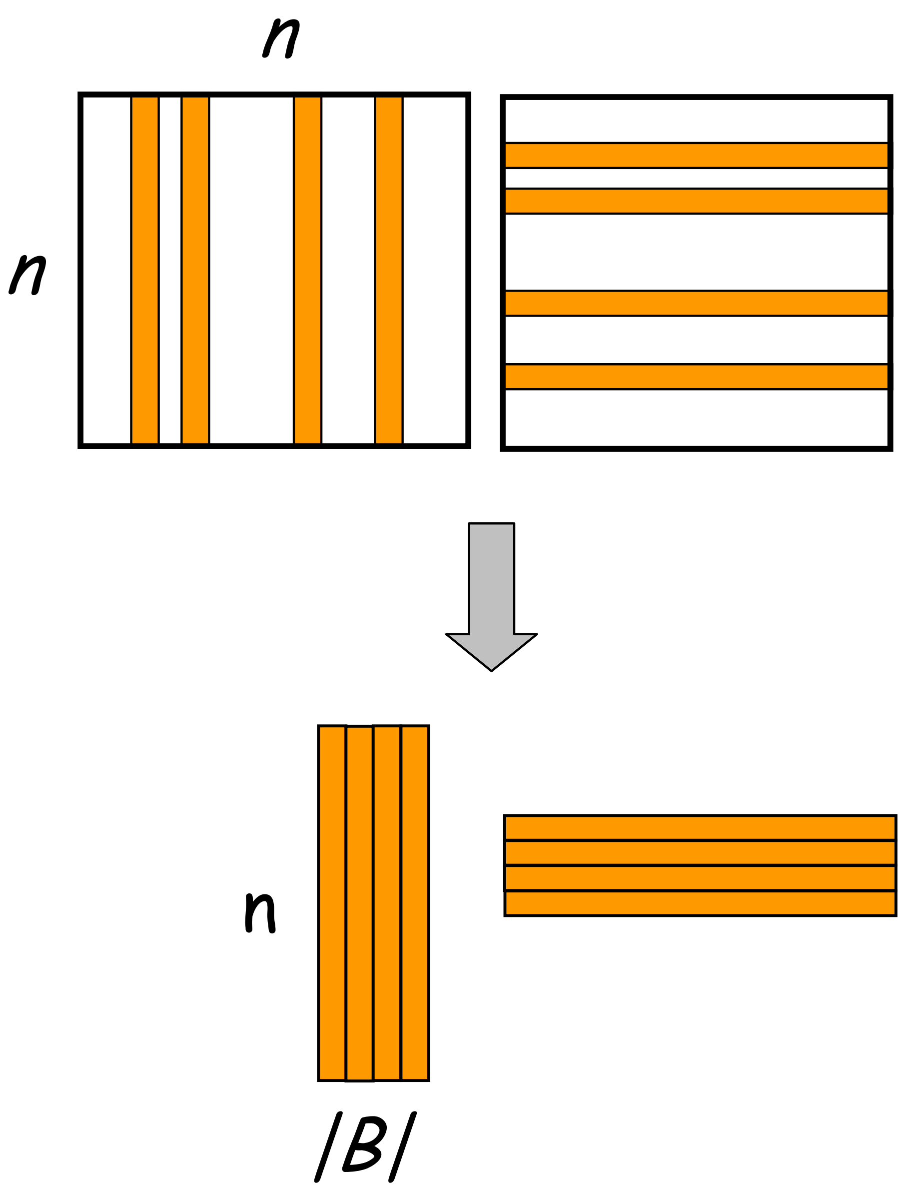
$\quad B \leftarrow \text{rand}(\ V\ , (9n \ln n)/s\ )$

$\quad D \leftarrow \min\{\ D\ ,\ D[V,B]*D[B,\mathrm{V}]\ \}$

}

The is also a slightly more complicated deterministic algorithm

Sampled Distance Products (Z '98)



In the i-th iteration, the set B is of size $n \ln n / s$, where $s = (3/2)^{i+1}$

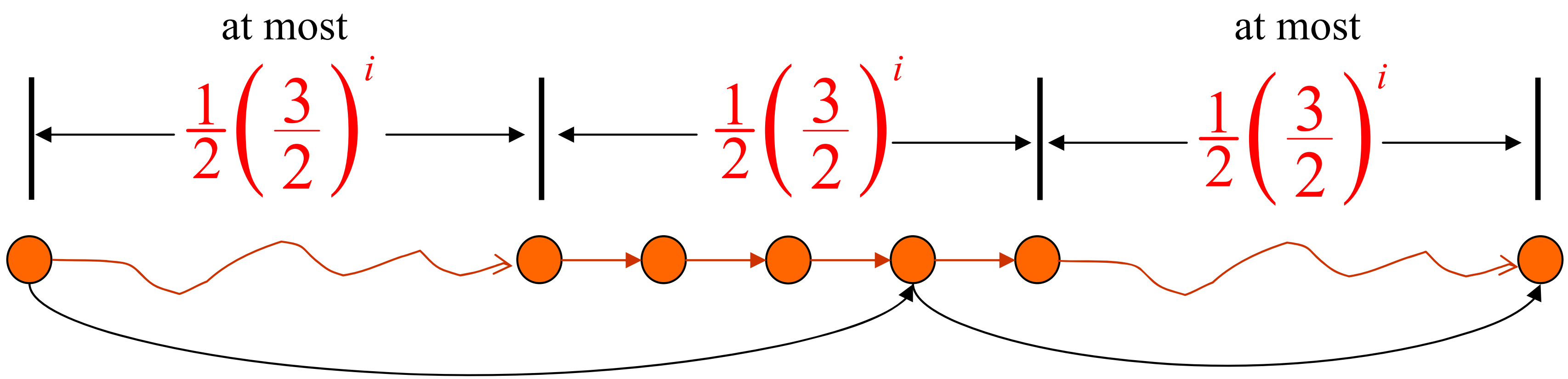
The matrices get smaller and smaller but the elements get larger and larger

Sampled Repeated Squaring - Correctness

```
D ← W
for i ←1 to log_{3/2} n do
{
    s ← (3/2)^{i+1}
    B ← rand(V,(9 ln n)/s)
    D ← min{ D , D[V,B]*D[B,V] }
}
```

Invariant: After the i-th iteration, distances that are attained using at most $(3/2)^i$ edges are correct.

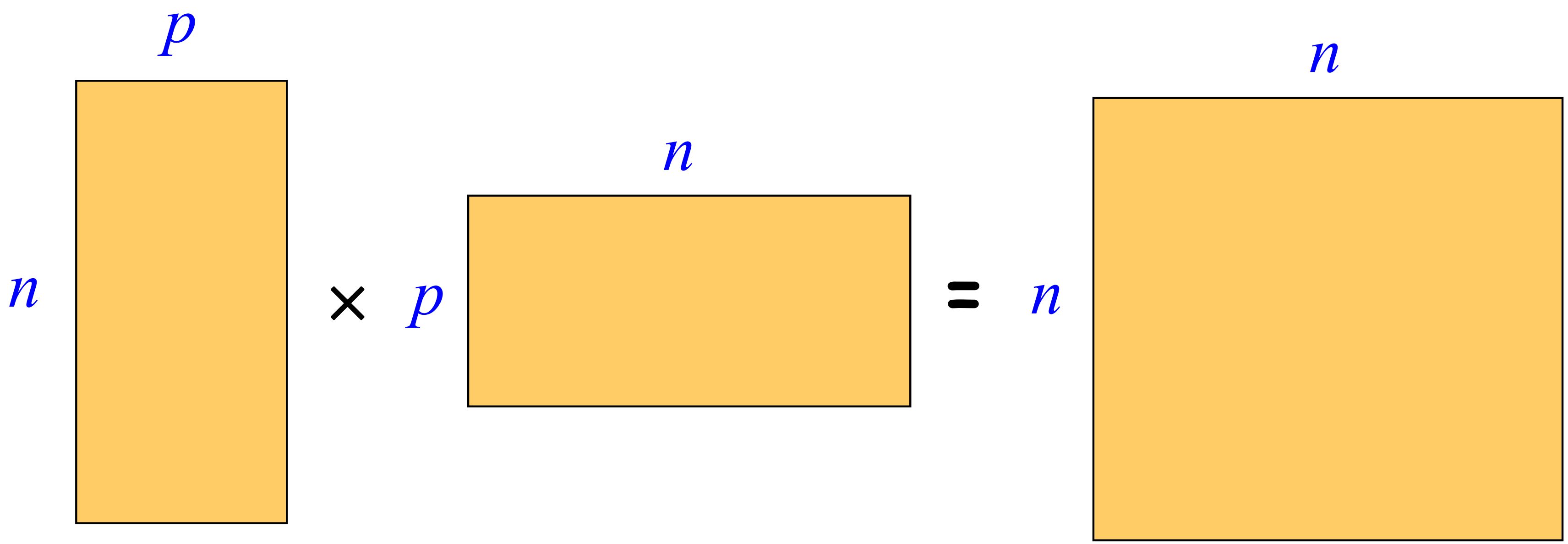
Consider a shortest path that uses at most $(3/2)^{i+1}$ edges



Let $s = (3/2)^{i+1}$

Failure probability : $\left(1-\frac{9\ln n}{s}\right)^{s/3} < n^{-3}$

Rectangular Matrix multiplication



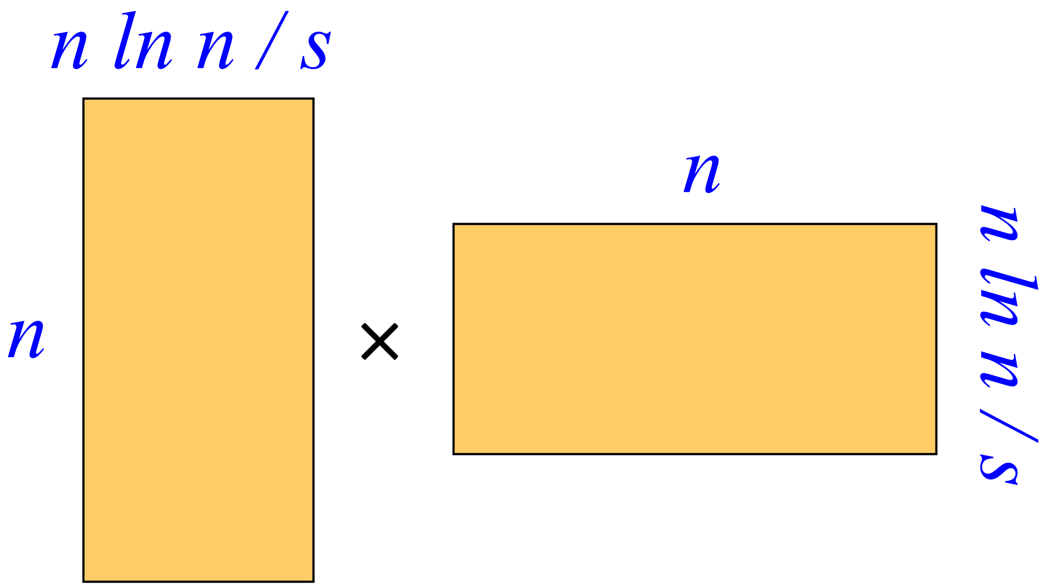
Naïve complexity: n^2p

[Coppersmith ’97]: $n^{1.85}p^{0.54}+n^{2+o(1)}$

For $p \le n^{0.29}$, complexity = $n^{2+o(1)}$!!!

Complexity of APSP algorithm

The i-th iteration:



$s = (3/2)^{i+1}$

The elements are of absolute value at most Ms

$$\min\left\{ Ms \cdot n^{1.85} \left(\frac{n}{s}\right)^{0.54}, \frac{n^3}{s} \right\} \le M^{0.68} n^{2.58}$$

Open problem:

Can APSP in directed graphs be solved in $O(n^{\omega})$ time?

Related result: [Yuster-Z'05]

A directed graphs can be processed in $O(n^{\omega})$ time so that any distance query can be answered in $O(n)$ time.

Corollary:

SSSP in directed graphs in $O(n^{\omega})$ time.

The preprocessing algorithm (YZ '05)

$D \leftarrow W$; $B \leftarrow V$

for $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**

{

$\quad s \leftarrow (3/2)^{i+1}$

$\quad B \leftarrow \mathrm{rand}(B, (9n \ln n)/s)$

$\quad D[V,B] \leftarrow \min\{D[V,B] \,,\, D[V,B] * D[B,B]\}$

$\quad D[B,V] \leftarrow \min\{D[B,V] \,,\, D[B,B] * D[B,V]\}$

}

The APSP algorithm

$D \leftarrow W$

for $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**

{

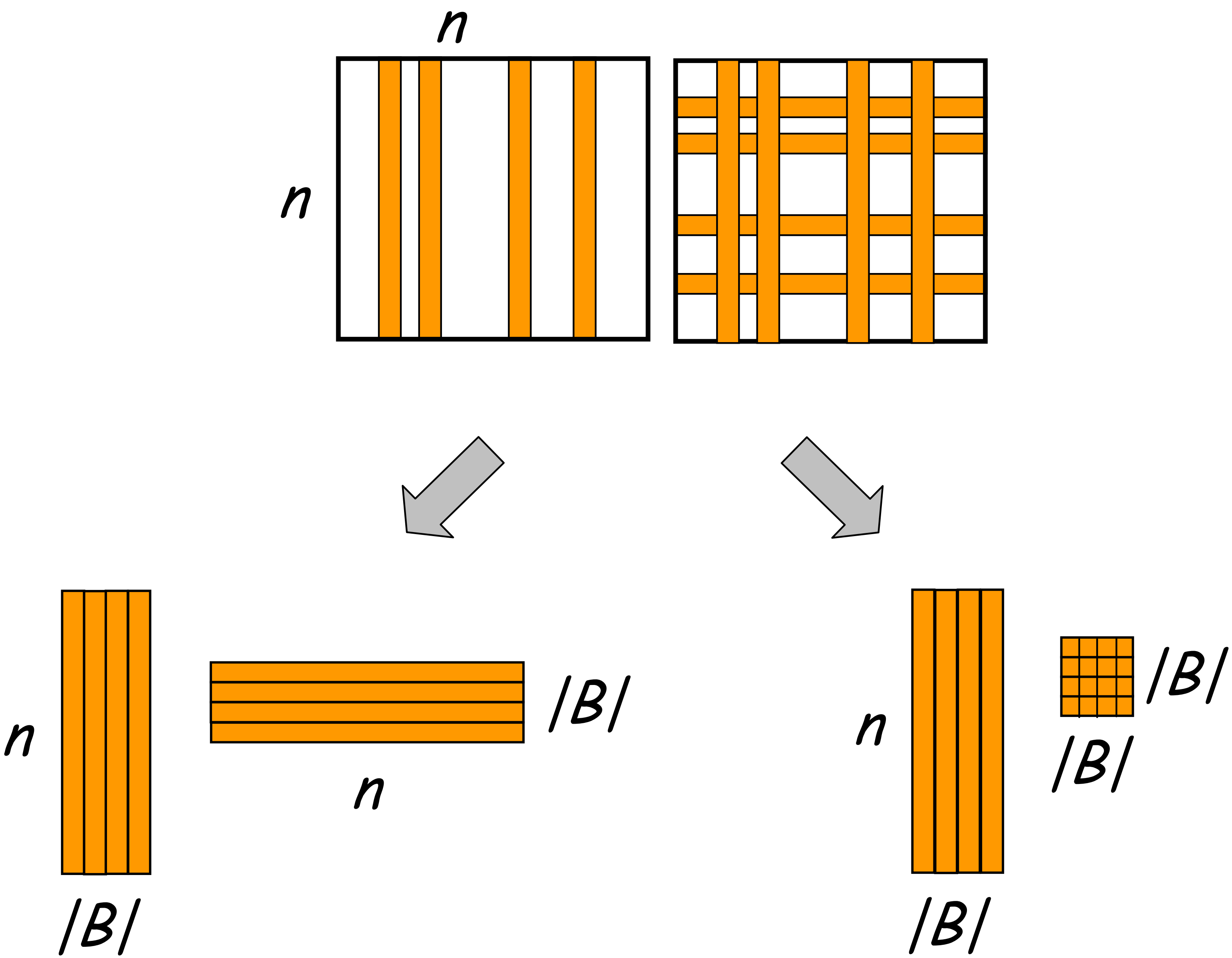
$s \leftarrow (3/2)^{i+1}$

$B \leftarrow \mathrm{rand}(V,(9n \ln n)/s)$

$D \leftarrow \min\{\, D \,,\, D[V,B] * D[B,V] \,\}$

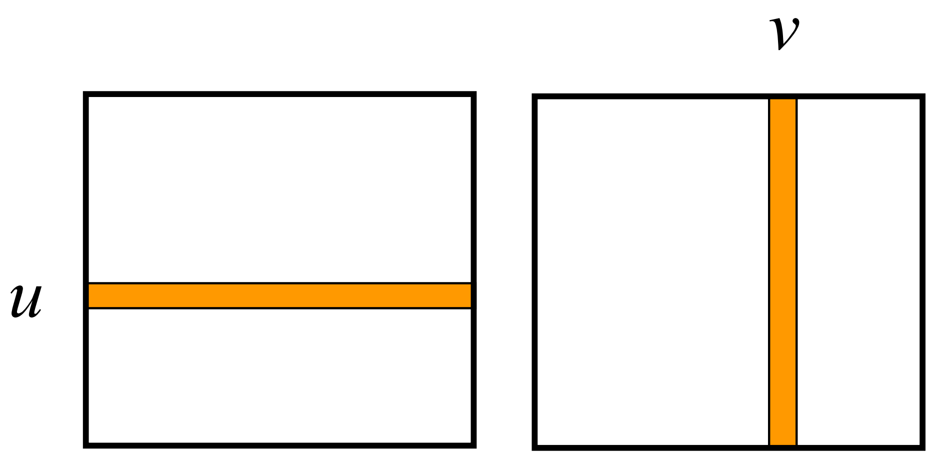
}

Twice Sampled Distance Products



The query answering algorithm

$$\boxed{\boldsymbol{\delta(u,v)} \leftarrow \; \boldsymbol{D[\{u\},V]*D[V,\{v\}]}}$$



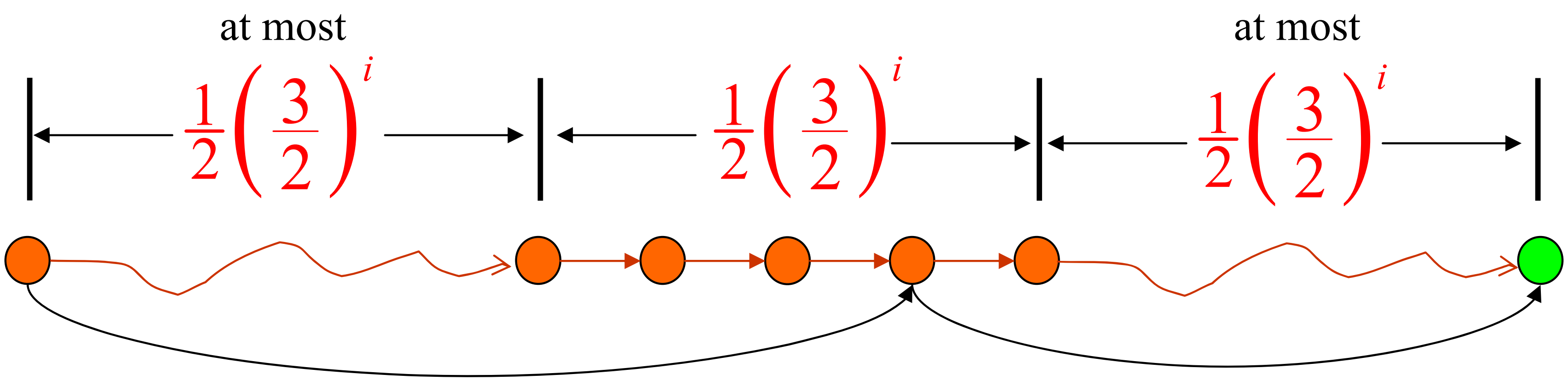
Query time: O(n)

The preprocessing algorithm: Correctness

Let B_i be the i-th sample. $B_1 \supseteq B_2 \supseteq B_3 \supseteq \ldots$

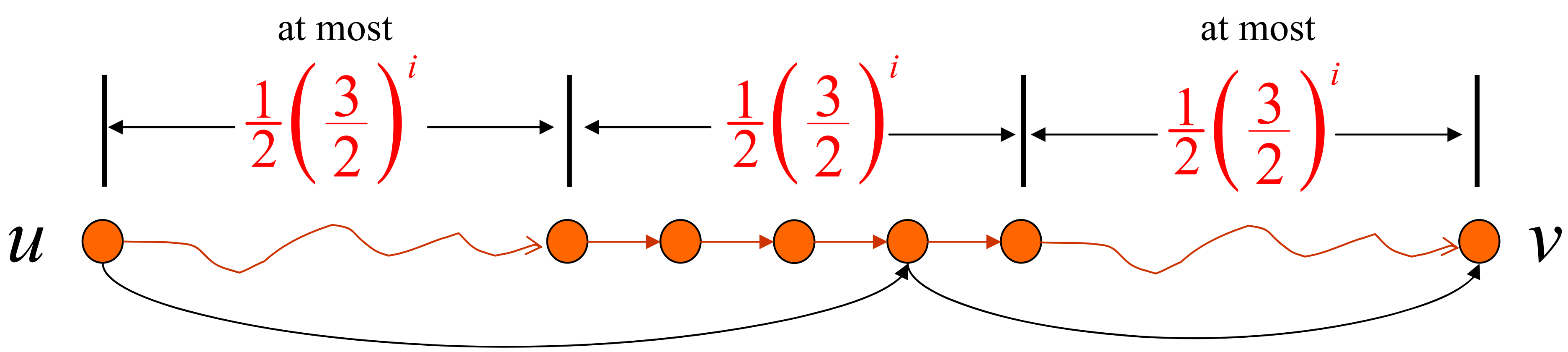
Invariant: After the i-th iteration, if $u \in B_i$ or $v \in B_i$ and there is a shortest path from u to v that uses at most $(3/2)^i$ edges, then $D(u,v)=\delta(u,v)$.

Consider a shortest path that uses at most $(3/2)^{i+1}$ edges



The query answering algorithm: Correctness

Suppose that the shortest path from u to v uses between $(3/2)^i$ and $(3/2)^{i+1}$ edges



All-Pairs Shortest Paths
in graphs with small integer weights

Directed graphs.
Edge weights in $\{-M,\ldots,0,\ldots M\}$

Running time	**Authors**
$M^{0.68}\, n^{2.58}$	[Zwick '98]

Improves results of
[Alon-Galil-Margalit '91] [Takaoka '98]

Answering distance queries

Directed graphs. Edge weights in $\{-M,\ldots,0,\ldots M\}$

Preprocessing time	**Query time**	**Authors**
$Mn^{2.38}$	n	[Yuster-Zwick '05]

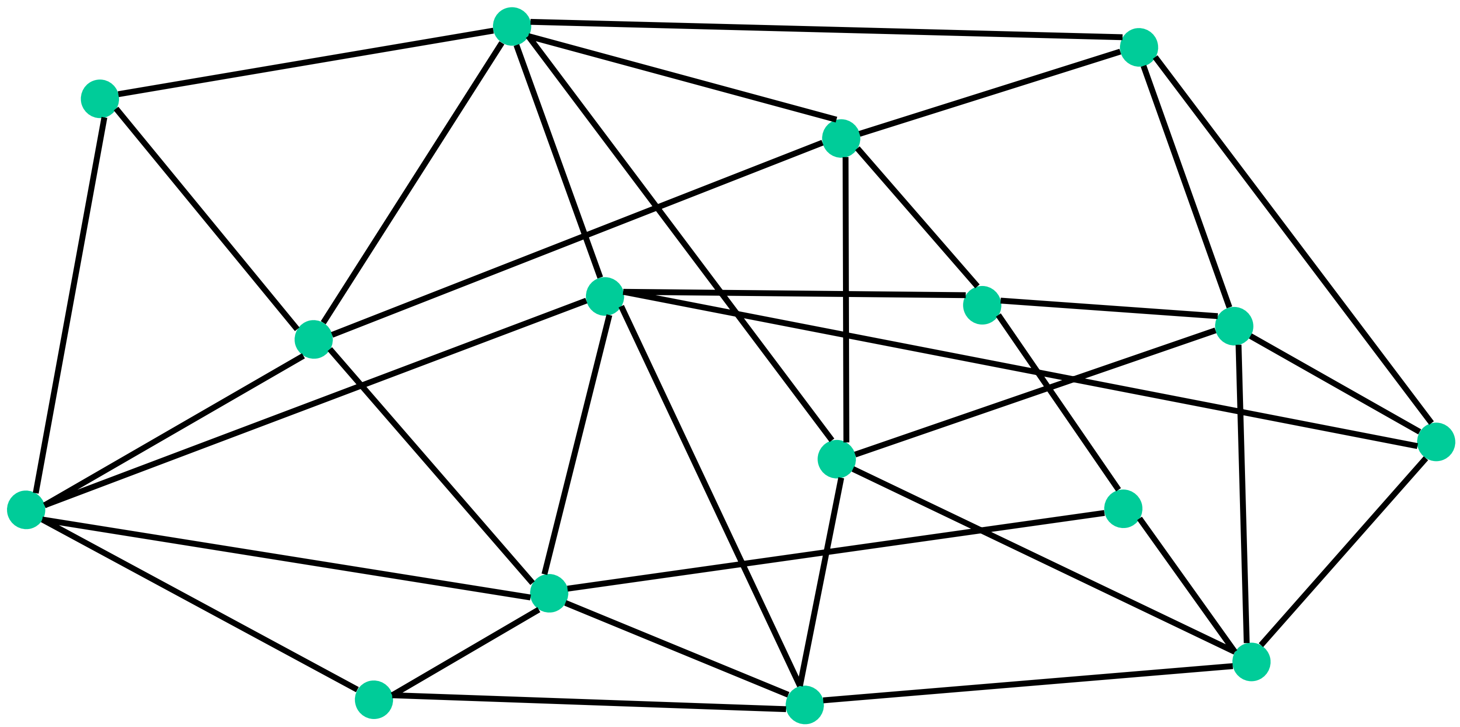
In particular, any $Mn^{1.38}$ distances can be computed in $Mn^{2.38}$ time.

For dense enough graphs with small enough edge weights, this improves on Goldberg's SSSP algorithm.

$Mn^{2.38}$ vs. $mn^{0.5}\log M$

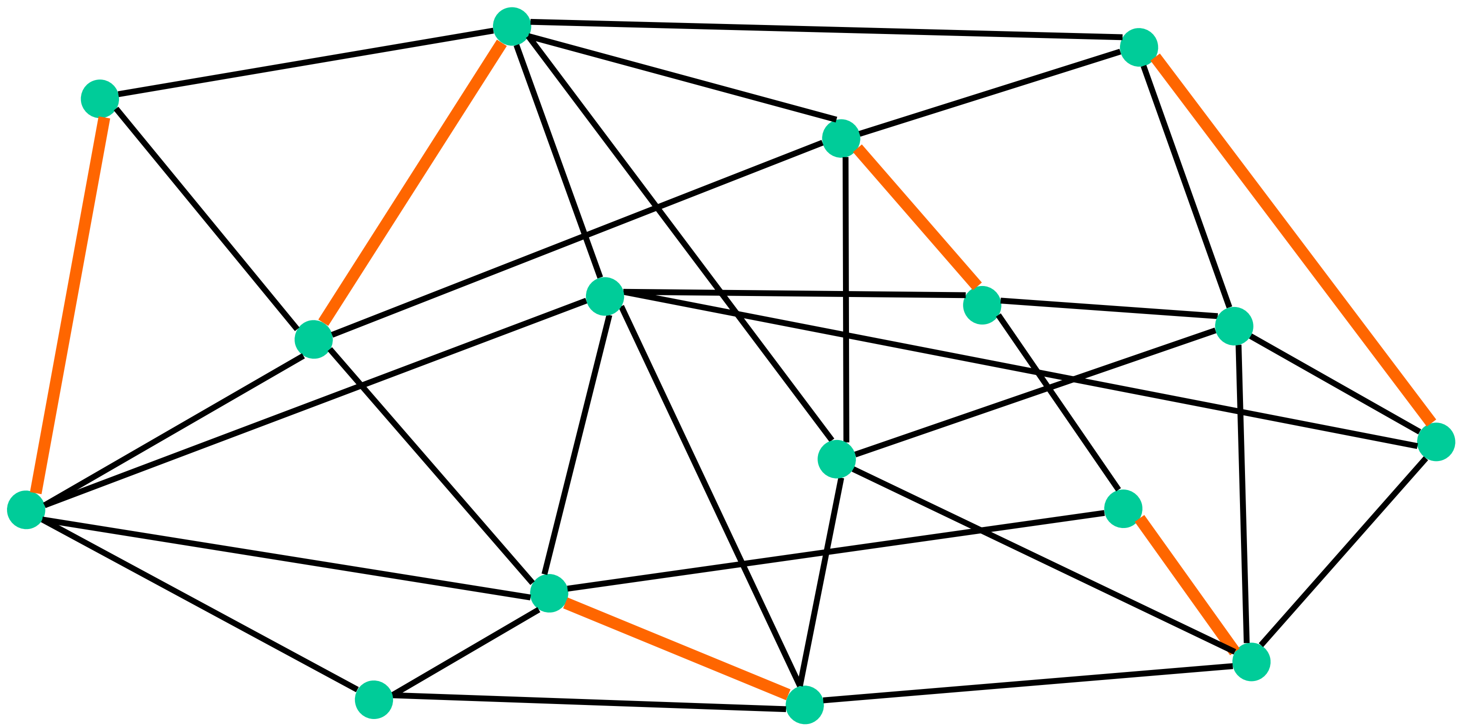
PERFECT MATCHINGS

Matchings



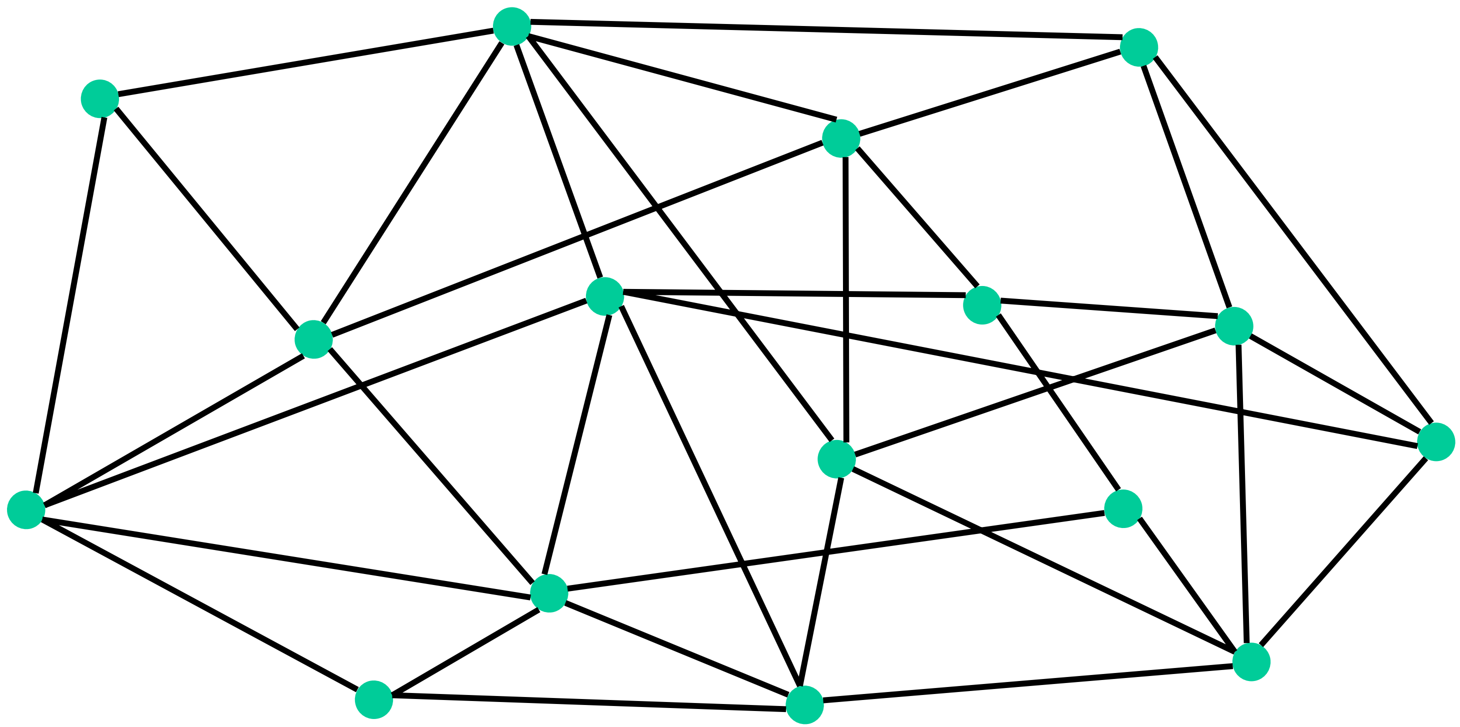
A matching is a subset of edges that do not touch one another.

Matchings



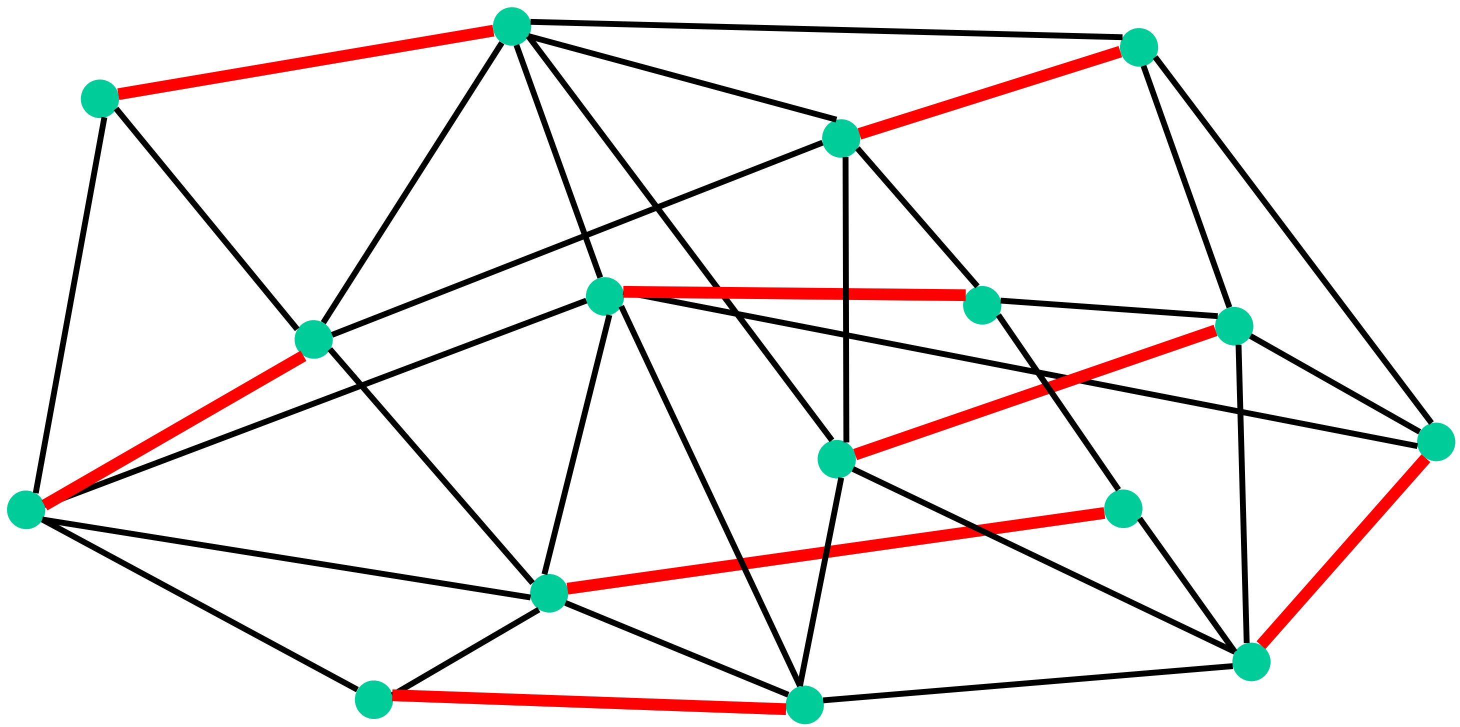
A matching is a subset of edges that do not touch one another.

Perfect Matchings



A matching is perfect if there are no unmatched vertices

Perfect Matchings



A matching is perfect if there are no unmatched vertices

Algorithms for finding perfect or maximum matchings

Combinatorial approach:

A matching M is a maximum matching iff it admits no augmenting paths

Algorithms for finding perfect or maximum matchings

Combinatorial approach:

A matching M is a maximum matching iff it admits no augmenting paths

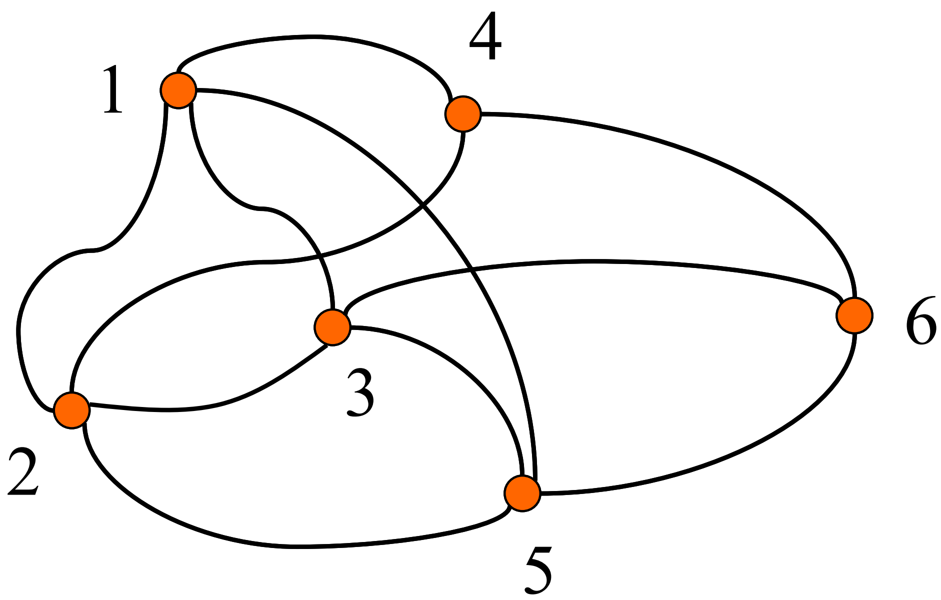
Combinatorial algorithms for finding perfect or maximum matchings

In bipartite graphs, augmenting paths can be found quite easily, and maximum matchings can be used using max flow techniques.

In non-bipartite the problem is much harder. (Edmonds' Blossom shrinking techniques)

Fastest running time (in both cases):
$O(mn^{1/2})$ [Hopcroft-Karp] [Micali-Vazirani]

Adjacency matrix of a undirected graph



$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

The adjacency matrix of an undirected graph is symmetric.

Matchings, Permanents, Determinants

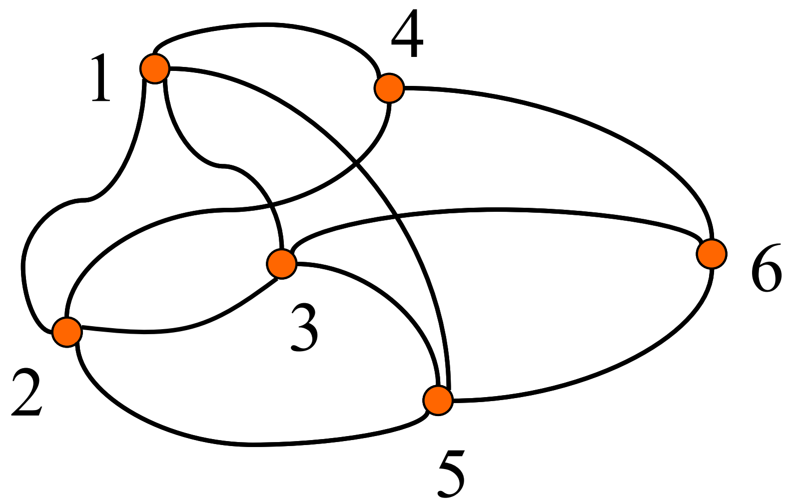
$$\det A = \sum_{\pi \in S_n} sign(\pi) \prod_{i=1}^{n} a_{i\pi(i)}$$

$$\text{per } A = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i\pi(i)}$$

Exercise 6: Show that if A is the adjacency matrix of a bipartite graph G, then per A is the number of perfect matchings in G.

Unfortunately computing the permanent is **#P-complete**…

Tutte's matrix
(Skew-symmetric symbolic adjacency matrix)

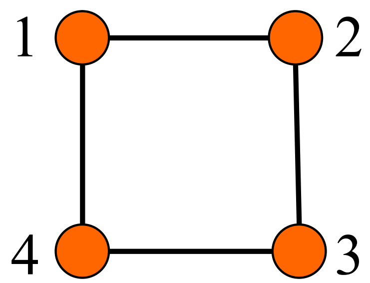


$$\begin{pmatrix} 0 & x_{12} & x_{13} & x_{14} & x_{15} & 0 \\ -x_{12} & 0 & x_{23} & x_{24} & x_{25} & 0 \\ -x_{13} & -x_{23} & 0 & 0 & x_{35} & x_{36} \\ -x_{14} & -x_{24} & 0 & 0 & 0 & x_{46} \\ -x_{15} & -x_{25} & -x_{35} & 0 & 0 & x_{56} \\ 0 & 0 & -x_{36} & -x_{46} & -x_{56} & 0 \end{pmatrix}$$

$$a_{ij} = \begin{cases} x_{ij} & \text{if } \{i,j\} \in E \text{ and } i < j, \\ -x_{ji} & \text{if } \{i,j\} \in E \text{ and } i > j, \\ 0 & \text{otherwise} \end{cases} \qquad A^T = -A$$

Tutte's theorem

Let ***G*=(*V*,*E*)** be a graph and let ***A*** be its Tutte matrix. Then, ***G*** has a perfect matching iff det ***A*** ≠0.



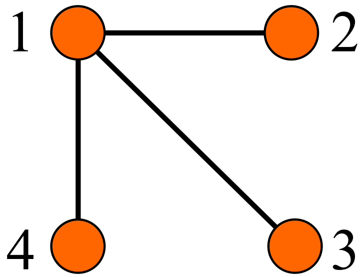
$$A = \begin{pmatrix} 0 & x_{12} & 0 & x_{14} \\ -x_{12} & 0 & x_{23} & 0 \\ 0 & -x_{23} & 0 & -x_{34} \\ -x_{14} & 0 & -x_{34} & 0 \end{pmatrix}$$

$$\det A \; = \; x_{12}^2 x_{34}^2 + x_{14}^2 x_{23}^2 + 2x_{12}x_{23}x_{34}x_{41} \; \neq \; 0$$

There are perfect matchings

Tutte's theorem

Let ***G*=(*V*,*E*)** be a graph and let ***A*** be its Tutte matrix. Then, ***G*** has a perfect matching iff det ***A*** ≠0.



$$A = \begin{pmatrix} 0 & x_{12} & x_{13} & x_{14} \\ -x_{12} & 0 & 0 & 0 \\ -x_{13} & 0 & 0 & 0 \\ -x_{14} & 0 & 0 & 0 \end{pmatrix}$$

$$\det A = 0$$

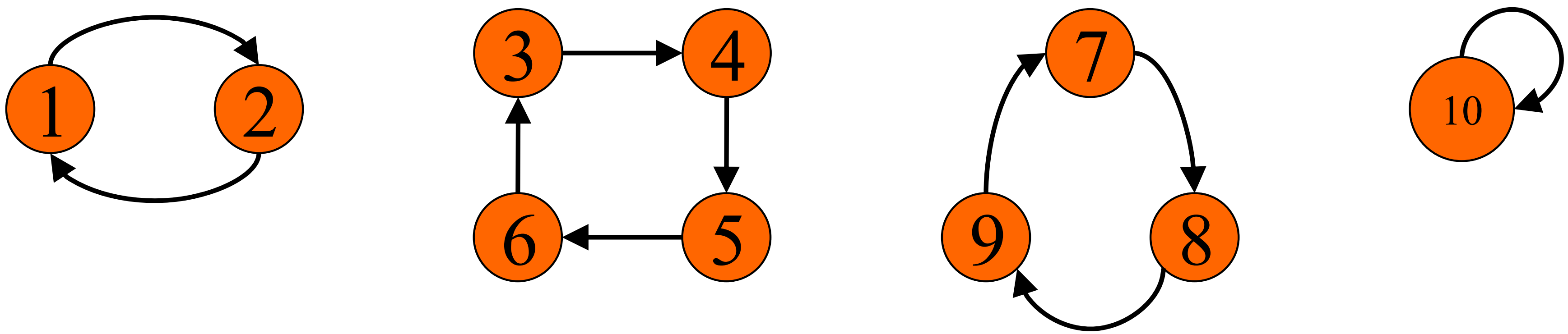
No perfect matchings

Proof of Tutte's theorem

$$\det A = \sum_{\pi \in S_n} sign(\pi) \prod_{i=1}^{n} a_{i\pi(i)}$$

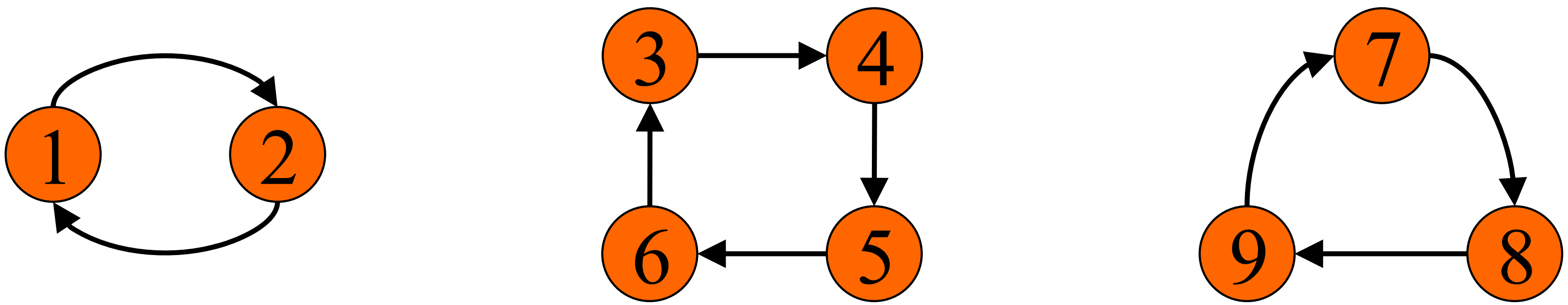
Every permutation $\pi \in S_n$ defines a cycle collection

$$\pi = (2\ 1\ 4\ 5\ 6\ 3\ 8\ 9\ 7\ 10)$$



Cycle covers

A permutation $\pi \in S_n$ for which $\{i, \pi(i)\} \in E$, for $1 \leq i \leq k$, defines a cycle cover of the graph.

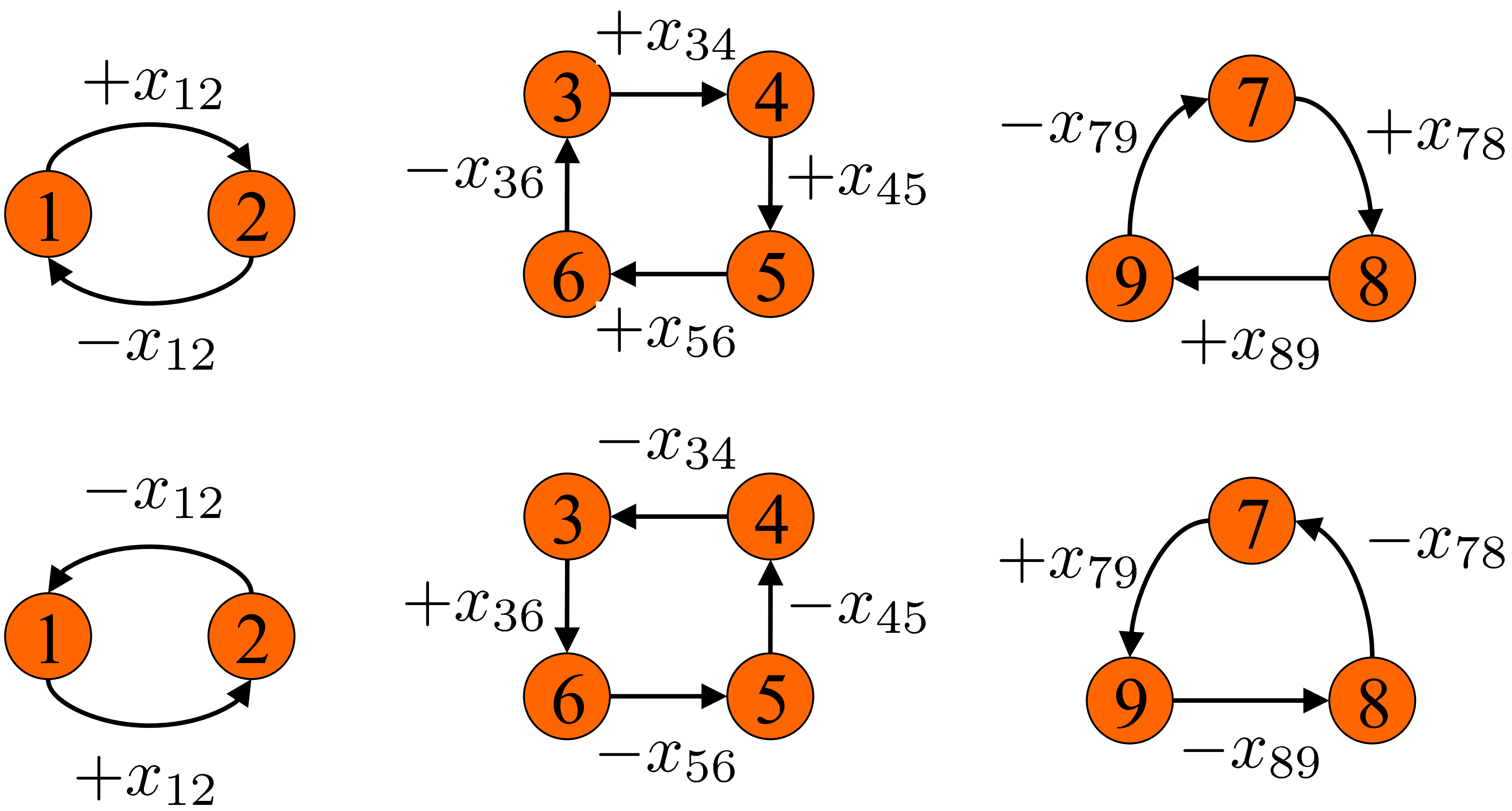


Exercise 7: If π' is obtained from π by reversing the direction of a cycle, then $sign(\pi') = sign(\pi)$.

$$\prod_{i=1}^{n} a_{i\pi'(i)} = \pm \prod_{i=1}^{n} a_{i\pi(i)}$$

Depending on the parity of the cycle!

Reversing Cycles



$$\prod_{i=1}^{n} a_{i\pi'(i)} = \pm \prod_{i=1}^{n} a_{i\pi(i)}$$

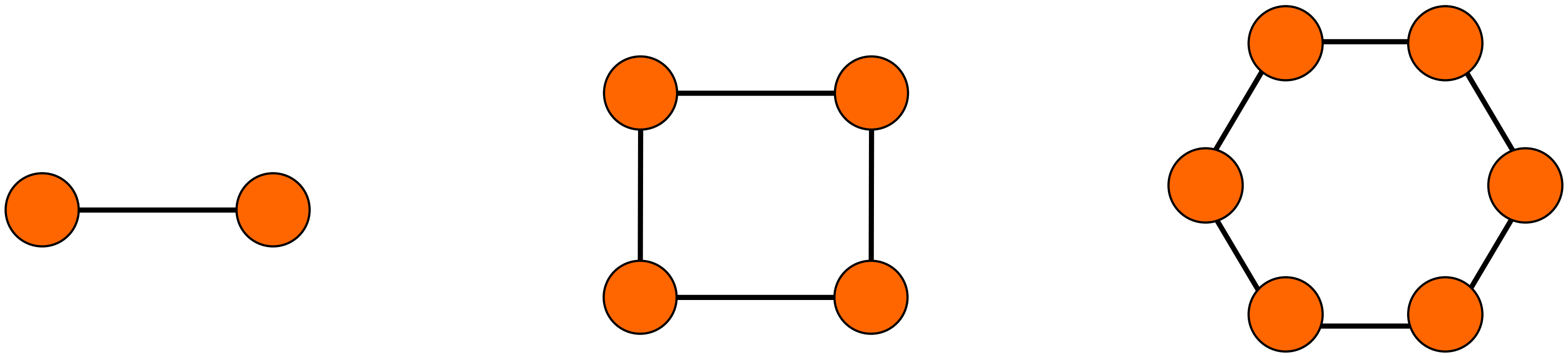
Depending on the parity of the cycle!

Proof of Tutte's theorem (cont.)

$$\det A = \sum_{\pi \in S_n} sign(\pi) \prod_{i=1}^{n} a_{i\pi(i)}$$

The permutations $\pi \in S_n$ that contain an **odd** cycle cancel each other! Thus we effectively sum only over **even cycle covers**.

A graph contains a perfect matching iff it contains an **even cycle covers**.



An algorithm for perfect matchings?

- Construct the Tutte matrix A.
- Compute $\det A$.
- If $\det A \neq 0$, say 'yes', otherwise 'no'.

Problem: $\det A$ is a symbolic expression that may be of exponential size!

Lovasz's solution: Replace each variable x_{ij} by a random element of Z_p, where $p = \Theta(n^2)$ is a prime number.

The Schwartz-Zippel lemma

Let $P(x_1,x_2,\dots,x_n)$ be a polynomial of degree d over a field F. Let $S \subseteq F$. If $P(x_1,x_2,\dots,x_n)\neq 0$ and $a_1,a_2,\dots,a_n$ are chosen randomly and independently from S, then

$$\Pr[\,P(a_1,a_2,\dots,a_n)=0\,] \;\le\; \frac{d}{|S|}$$

Proof by induction on n.

For $n=1$, follows from the fact that polynomial of degree d over a field has at most d roots

Lovasz's algorithm for existence of perfect matchings

- Construct the Tutte matrix $\boldsymbol{A}$.
- Replace each variable x_{ij} by a random element of Z_p, where p=O(n^2) is prime.
- Compute det $\boldsymbol{A}$.
- If det $\boldsymbol{A} \neq 0$, say 'yes', otherwise 'no'.

If algorithm says 'yes', then
the graph contains a perfect matching.

If the graph contains a perfect matching, then
the probability that the algorithm says 'no',
is at most O(1/n).

Finding perfect matchings

Rabin-Vazirani (1986): An edge $\{i,j\} \in E$ is contained in a perfect matching iff $(A^{-1})_{ij} \neq 0$.

Leads immediately to an $O(n^{\omega+1})$ algorithm: Find an allowed edge $\{i,j\} \in E$, delete it and it vertices from the graph, and recompute A^{-1}.

Mucha-Sankowski (2004): Recomputing A^{-1} from scratch is very wasteful. Running time can be reduced to $O(n^{\omega})$!

Harvey (2006): A simpler $O(n^{\omega})$ algorithm.

SUMMARY AND OPEN PROBLEMS

Open problems

- An $O(n^\omega)$ algorithm for the directed unweighted APSP problem?
- An $O(n^{3-\varepsilon})$ algorithm for the APSP problem with edge weights in $\{1,2,\ldots,n\}$?
- Deterministic $O(n^\omega)$ algorithm for maximum or perfect matcing?
- An $O(n^{2.5-\varepsilon})$ algorithm for weighted matching with edge weights in $\{1,2,\ldots,n\}$?
- Other applications of fast matrix multiplication?