Outline

1. Algebraic matrix multiplication
   a. Strassen’s algorithm
   b. Rectangular matrix multiplication

2. Boolean matrix multiplication
   a. Simple reduction to integer matrix multiplication
   b. Computing the transitive closure of a graph.

3. Min-Plus matrix multiplication
   a. Equivalence to the APSP problem
   b. Expensive reduction to algebraic products
   c. Fredman’s trick
4. APSP in undirected graphs
   a. An $O(n^{2.38})$ algorithm for unweighted graphs (Seidel)
   b. An $O(Mn^{2.38})$ algorithm for weighted graphs (Shoshan-Zwick)

5. APSP in directed graphs
   1. An $O(M^{0.68}n^{2.58})$ algorithm (Zwick)
   2. An $O(Mn^{2.38})$ preprocessing / $O(n)$ query answering algorithm (Yuster-Zwick)
   3. An $O(n^{2.38}\log M) (1+\varepsilon)$-approximation algorithm

6. Summary and open problems
Short introduction to Fast matrix multiplication
Algebraic Matrix Multiplication

$A = (a_{ij}) \times B = (b_{ij}) = C = (c_{ij})$

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

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

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3$</td>
<td>—</td>
</tr>
<tr>
<td>$n^{2.81}$</td>
<td>Strassen (1969)</td>
</tr>
<tr>
<td>$n^{2.38}$</td>
<td>Coppersmith, Winograd (1990)</td>
</tr>
</tbody>
</table>

Conjecture/Open problem: $n^{2+o(1)}$
Multiplying $2 \times 2$ 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{align*}
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{align*}

8 multiplications
4 additions

Works over any ring!
Multiplying $n \times n$ matrices

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

\[
C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad \text{8 multiplications}
\]

\[
C_{12} = A_{11}B_{12} + A_{12}B_{22} \quad \text{4 additions}
\]

\[
C_{21} = A_{21}B_{11} + A_{22}B_{21}
\]

\[
C_{22} = A_{21}B_{12} + A_{22}B_{22}
\]

\[
T(n) = 8 \ T(n/2) + O(n^2)
\]

\[
T(n) = O(n^{\log_8{8}/\log_2{2}}) = O(n^3)
\]
Strassen’s $2 \times 2$ algorithm

\[
\begin{align*}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22} \\
M_1 &= (A_{11} + A_{12})B_{22} \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{align*}
\]

7 multiplications

18 additions/subtractions

Works over any ring!
“Strassen Symmetry”
(by Mike Paterson)
Strassen’s $n \times n$ algorithm

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

Apply the $2 \times 2$ 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…

New group theoretic approach [Cohn-Umans ‘03] [Cohn-Kleinberg-szegedy-Umans ‘05]

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

Many believe that $\omega=2+o(1)$. 
Determinants / Inverses

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
Matrix Multiplication
Determinants / Inverses

What is it good for?

Transitive closure
Shortest Paths
Perfect/Maximum matchings
Dynamic transitive closure
k-vertex connectivity
Counting spanning trees
Rectangular Matrix multiplication

$n \times p = n$

Naïve complexity: $n^2p$

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

For $p \leq n^{0.29}$, complexity = $n^{2+o(1)}$
BOOLEAN MATRIX MULTIPLICATION and TRANSITIVE CLOSURE
Boolean Matrix Multiplication

\[ A = (a_{ij}) \times B = (b_{ij}) = C = (c_{ij}) \]

\[ c_{ij} = \bigvee_{k=1}^{n} a_{ik} \land b_{kj} \]

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

\[ C = A \cdot B \]

\[ c_{ij} = \sum_{k} a_{ik} b_{kj} \]

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

Boolean Product

\[ C = A \cdot B \]

\[ c_{ij} = \bigvee_{k} a_{ik} \land b_{kj} \]

\( O(n^{2.38}) \) operations on \( \text{O}(\log n) \) bit words

But, we can work over the integers!

(modulo \( n+1 \))
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.
Adjacency matrix of a directed graph

\[ A \]

\[ \begin{pmatrix}
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{pmatrix} \]

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.

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

The matrix $(A \lor I)^{n-1}$ 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.
\[ X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \]

\[ X^* = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \]

\[ \text{TC}(n) \leq 2 \text{TC}(n/2) + 6 \text{BMM}(n/2) + O(n^2) \]
Exercise 1: Give $O(n^\omega)$ algorithms for finding, 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.
MIN-PLUS MATRIX MULTIPLICATION and ALL-PAIRS SHORTEST PATHS (APSP)
An interesting special case of the APSP problem

Min-Plus product

$$C = A \ast B$$

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

Min-Plus product
Min-Plus Products

\[ C = A \ast B \]

\[ c_{i,j} = \min_k \{ a_{i,k} + b_{k,j} \} \]

\[
\begin{pmatrix}
-6 & -3 & -10 \\
2 & 5 & -2 \\
-1 & -7 & -5
\end{pmatrix}
= \begin{pmatrix}
1 & -3 & 7 \\
+\infty & 5 & +\infty \\
8 & 2 & -5
\end{pmatrix}
\ast
\begin{pmatrix}
8 & +\infty & -4 \\
-3 & 0 & -7 \\
5 & -2 & 1
\end{pmatrix}
\]
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.

\[
D \leftarrow W \\
\text{for } i \leftarrow 1 \text{ to } \lceil \log_2 n \rceil \\
\text{do } D \leftarrow D \times D
\]

Thus: $\text{APSP}(n) \leq \text{MPP}(n) \log n$

Actually: $\text{APSP}(n) = O(\text{MPP}(n))$
$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$X^* = \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} (A \lor BD^*C)^* & EBD^* \\ D^*CE & D^* \lor GBD^* \end{bmatrix}$$

$$\text{APSP}(n) \leq 2 \text{APSP}(n/2) + 6 \text{MPP}(n/2) + O(n^2)$$
Algebraic Product

\[ C = A \cdot B \]

\[ c_{ij} = \sum_k a_{ik} b_{kj} \]

\[ O(n^{2.38}) \]

Min-Plus Product

\[ C = A \ast B \]

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

min operation has no inverse!
Fredman’s trick

The **min-plus** product of two $n \times n$ matrices can be deduced after only $O(n^{2.5})$ additions and comparisons.

It is not known how to implement the algorithm in $O(n^{2.5})$ time.
Algebraic Decision Trees

\[ a_{17} - a_{19} \leq b_{92} - b_{72} \]

\[ n^{2.5} \]

\[ \begin{align*}
c_{11} &= a_{17} + b_7 \\
c_{12} &= a_{14} + b_4 \\
& \vdots
\end{align*} \]

\[ \begin{align*}
c_{11} &= a_{13} + b_3 \\
c_{12} &= a_{15} + b_5 \\
& \vdots
\end{align*} \]

\[ \begin{align*}
c_{11} &= a_{18} + b_8 \\
c_{12} &= a_{16} + b_6 \\
& \vdots
\end{align*} \]

\[ \begin{align*}
c_{11} &= a_{12} + b_2 \\
c_{12} &= a_{13} + b_3 \\
& \vdots
\end{align*} \]
Breaking a square product into several rectangular products

\[ A \times B = \min_i A_i \times B_i \]

\[ \text{MPP}(n) \leq (n/m) \left( \text{MPP}(n,m,n) + n^2 \right) \]
Fredman’s trick

Naïve calculation requires $n^2m$ operations

Fredman observed that the result can be inferred after performing only $O(nm^2)$ operations

$$a_{ir} + b_{rj} \leq a_{is} + b_{sj}$$
$$a_{ir} - a_{is} \leq b_{sj} - b_{rj}$$
Fredman’s trick (cont.)

\[ a_{ir} + b_{rj} \leq a_{is} + b_{sj} \iff a_{ir} - a_{is} \leq b_{sj} - b_{rj} \]

- **Generate** all the differences \( a_{ir} - a_{is} \) and \( b_{sj} - b_{rj} \).
- **Sort** them using \( O(nm^2) \) comparisons. (Non-trivial!)
- **Merge** the two sorted lists using \( O(nm^2) \) comparisons.

The ordering of the elements in the sorted list determines the result of the min-plus product.
# All-Pairs Shortest Paths

in directed graphs with “real” edge weights

<table>
<thead>
<tr>
<th>Running time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n^3)</td>
<td>[Floyd ’62] [Warshall ’62]</td>
</tr>
<tr>
<td>(n^3 (\log \log n / \log n)^{1/3})</td>
<td>[Fredman ’76]</td>
</tr>
<tr>
<td>(n^3 (\log \log n / \log n)^{1/2})</td>
<td>[Takaoka ’92]</td>
</tr>
<tr>
<td>(n^3 / (\log n)^{1/2})</td>
<td>[Dobosiewicz ’90]</td>
</tr>
<tr>
<td>(n^3 (\log \log n / \log n)^{5/7})</td>
<td>[Han ’04]</td>
</tr>
<tr>
<td>(n^3 \log \log n / \log n)</td>
<td>[Takaoka ’04]</td>
</tr>
<tr>
<td>(n^3 (\log \log n)^{1/2} / \log n)</td>
<td>[Zwick ’04]</td>
</tr>
<tr>
<td>(n^3 / \log n)</td>
<td>[Chan ’05]</td>
</tr>
<tr>
<td>(n^3 (\log \log n / \log n)^{5/4})</td>
<td>[Han ’06]</td>
</tr>
<tr>
<td>(n^3 (\log \log n)^3 / (\log n)^2)</td>
<td>[Chan ’07]</td>
</tr>
</tbody>
</table>
PERFECT MATCHINGS
A matching is a subset of edges that do not touch one another.
A matching is a subset of edges that do not touch one another.
A matching is perfect if there are no unmatched vertices.
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]
The adjacency matrix of a undirected graph is symmetric.
Matchings, Permanents, Determinants

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

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

**Exercise 2:** Show that if \( A \) is the adjacency matrix of a bipartite graph \( G \), then \( \text{per}(A) \) is the number of perfect matchings in \( G \).

Unfortunately computing the permanent is \( \#P\)-complete…
Tutte’s matrix
(Skew-symmetric symbolic adjacency matrix)

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

\[
a_{ij} = \begin{cases} 
  x_{ij} & \text{if } \{i, j\} \in E \text{ and } i < j, \\
  -x_{ji} & \text{if } \{i, j\} \in E \text{ and } i > j, \\
  0 & \text{otherwise}
\end{cases}
\]

\[A^T = -A\]
Tutte’s theorem

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

There are perfect matchings

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

$$\det A = x_{12}^2 x_{34}^2 + x_{14}^2 x_{23}^2 + 2x_{12}x_{23}x_{34}x_{41} \neq 0$$
Tutte’s theorem

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

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

$\det A = 0$

No perfect matchings
Proof of Tutte’s theorem

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

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 3: If \( \pi' \) is obtained from \( \pi \) by reversing the direction of a cycle, then \( \text{sign}(\pi') = \text{sign}(\pi) \).

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

Depending on the parity of the cycle!
Reversing Cycles

Depending on the parity of the cycle!

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

Depending on the parity of the cycle!
Proof of Tutte’s theorem (cont.)

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

The permutations \( \pi \in S_n \) that contain an odd cycle cancel each other!

We effectively sum only over even cycle covers.

A graph contains a perfect matching iff it contains an even cycle cover.
Proof of Tutte’s theorem (cont.)

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

Perfect Matching $\rightarrow$ Even cycle cover
Proof of Tutte’s theorem (cont.)

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

Even cycle cover $\Rightarrow$ Perfect matching
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 \( \mathbb{Z}_p \), where \( p = \Theta(n^2) \) is a prime number.
The Schwartz-Zippel lemma

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

\[
\Pr[ P(a_1,a_2,\ldots,a_n) = 0 ] \leq \frac{d}{|S|}
\]

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 $A$.
- Replace each variable $x_{ij}$ by a random element of $\mathbb{Z}_p$, where $p=O(n^2)$ is prime.
- Compute $\det A$.
- If $\det A \neq 0$, say ‘yes’, otherwise ‘no’.

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

If algorithm says ‘yes’, then the graph contains a perfect matching.
Parallel algorithms

Determinants can be computed very quickly in parallel

\[ \text{DET} \in \text{NC}^2 \]

Perfect matchings can be detected very quickly in parallel (using randomization)

\[ \text{PERFECT-MATCH} \in \text{RNC}^2 \]

Open problem:

??? \[ \text{PERFECT-MATCH} \in \text{NC} ??? \]
Finding perfect matchings

Self Reducibility

Delete an edge and check whether there is still a perfect matching

Needs $O(n^2)$ determinant computations

Running time $O(n^{\omega+2})$

Fairly slow…

Not parallelizable!
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 its 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.
Adjoint and Cramer’s rule

\[(\text{adj}(A))_{ij} = (-1)^{i+j} \det(A^{j,i}) = \det_{j}\]

A with the j-th row and i-th column deleted

Cramer’s rule: \[A^{-1} = \frac{\text{adj}(A)}{\det(A)}\]
Finding perfect matchings

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

\[(\text{adj}(A))_{ij} = \left(-1\right)^{i+j} \det(A^{j,i}) = \det_j \]

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

Still not parallelizable
Finding unique minimum weight perfect matchings

Suppose that edge \( \{i,j\} \in E \) has integer weight \( w_{ij} \).

Suppose that there is a unique minimum weight perfect matching \( M \) of total weight \( W \).

Replace \( x_{ij} \) by \( 2^{w_{ij}} \).

Then, \( 2^{2W} \mid \det(A) \) but \( 2^{2W+1} \nmid \det(A) \).

Furthermore, \( \{i,j\} \in M \) iff \( \frac{2^{w_{ij}} \det(A^{ij})}{2^{2W}} \) is odd.
Isolating lemma
[Mulmuley-Vazirani-Vazirani (1987)]

Suppose that $G$ has a perfect matching

Assign each edge $\{i,j\} \in E$

a random integer weight $w_{ij} \in [1, 2m]$

With probability of at least $\frac{1}{2}$, the minimum weight perfect matching of $G$ is unique

Lemma holds for general collections of sets, not just perfect matchings
Proof of Isolating lemma
[Mulmuley-Vazirani-Vazirani (1987)]

An edge $\{i,j\}$ is ambivalent if there is a minimum weight perfect matching that contains it and another that does not.

Suppose that weights were assigned to all edges except for $\{i,j\}$.

Let $a_{ij}$ be the largest weight for which $\{i,j\}$ participates in some minimum weight perfect matchings.

If $w_{ij} < a_{ij}$, then $\{i,j\}$ participates in all minimum weight perfect matchings.

The probability that $\{i,j\}$ is ambivalent is at most $1/(2m)!$.
Finding perfect matchings
[Mulmuley-Vazirani-Vazirani (1987)]

Choose random weights in \([1,2m]\)
Compute determinant and adjoint
Read of a perfect matching (w.h.p.)

Is using \(m\)-bit integers cheating?
Not if we are willing to pay for it!
Complexity is \(O(mn^\omega) \leq O(n^{\omega+2})\)

Finding perfect matchings in \(\text{RNC}^2\)

Improves an \(\text{RNC}^3\) algorithm by
[Karp-Upfal-Wigderson (1986)]
Multiplying two $N$-bit numbers

``School method’’

$N^2$

[Schönhage-Strassen (1971)]

$N \log N \log \log N$

[Fürer (2007)]

[De-Kurur-Saha-Saptharishi (2008)]

$N \log N 2^{O(\log^* N)}$

For our purposes…  $\tilde{O}(N)$
Finding perfect matchings

We are not over yet…

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

[Harvey (2006)]
A simpler $O(n^\omega)$ algorithm.
Using matrix multiplication to compute min-plus products

\[
\begin{pmatrix}
    c_{11} & c_{12} \\
    c_{21} & c_{22} \\
    O & O
\end{pmatrix}
= \begin{pmatrix}
    a_{11} & a_{12} \\
    a_{21} & a_{22} \\
    O & O
\end{pmatrix}
\times \begin{pmatrix}
    b_{11} & b_{12} \\
    b_{21} & b_{22} \\
    O & O
\end{pmatrix}
\]

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

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

\[c'_{ij} = \sum_k x^{a_{ik}+b_{kj}}\quad c_{ij} = \text{first}(c'_{ij})\]
Using matrix multiplication to compute min-plus products

Assume: \( 0 \leq a_{ij}, b_{ij} \leq M \)

\[
\begin{pmatrix}
c'_{11} & c'_{12} \\
c'_{21} & c'_{22}
\end{pmatrix}
\begin{pmatrix}
x^{a_{11}} & x^{a_{12}} \\
x^{a_{21}} & x^{a_{22}}
\end{pmatrix}
\begin{pmatrix}
x^{b_{11}} & x^{b_{12}} \\
x^{b_{21}} & x^{b_{22}}
\end{pmatrix}
= \begin{pmatrix}
i & i \\
i & i
\end{pmatrix}
\]

\( n^\omega \) polynomial products \( \times \) \( M \) operations per polynomial product \( \times \) \( Mn^\omega \) operations per max-plus product
SHORTEST PATHS

APSP – All-Pairs Shortest Paths
SSSP – Single-Source Shortest Paths
UNWEIGHTED
UNDIRECTED
SHORTEST PATHS
4. APSP in undirected graphs
   a. An $O(n^{2.38})$ algorithm for unweighted graphs (Seidel)
   b. An $O(Mn^{2.38})$ algorithm for weighted graphs (Shoshan-Zwick)

5. APSP in directed graphs
   1. An $O(M^{0.68}n^{2.58})$ algorithm (Zwick)
   2. An $O(Mn^{2.38})$ preprocessing / $O(n)$ query answering algorithm (Yuster-Zwick)
   3. An $O(n^{2.38}\log M)$ $(1+\varepsilon)$-approximation algorithm

6. Summary and open problems
Directed versus undirected graphs

\[ \delta(x,z) \leq \delta(x,y) + \delta(y,z) \]

Triangle inequality

\[ \delta(x,y) \leq \delta(x,z) + \delta(z,y) \]

\[ \delta(x,z) \geq \delta(x,y) - \delta(y,z) \]

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

**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 $G^2$

$$\sum_{(v,w) \in E} c_{uw} = \sum_{w \in V} c_{uw} a_{wv} = (CA)_{uv} \geq \text{deg}(v)c_{uv}$$
**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 $\delta^2(u,w) \leq \delta^2(u,v)$ and for at least one neighbor $\delta^2(u,w) < \delta^2(u,v)$.

Exercise 4: Prove the lemma.

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

$$\sum_{(v,w)\in E} c_{uw} = \sum_{w \in V} c_{uw}a_{vw} = (CA)_{uv} < \deg(v)c_{uv}$$
Seidel's algorithm

Algorithm APD($A$)

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.

Assume that $A$ has 1’s on the diagonal.

Boolean matrix multiplication

Return $J$

else

$C \leftarrow APD(A^2)$

$X \leftarrow CA$, $deg \leftarrow Ae-1$

$deg_j$ = $deg_{ij}$

$X = C = A$ (diag. 1’s)

Integer matrix multiplication

Complexity: $O(n^{\omega \log n})$
Exercise 5: (*) Obtain a version of Seidel’s algorithm that uses only \textbf{Boolean} matrix multiplications.

\textbf{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

\[ C = AB \]

\[ c_{ij} = \bigvee_{k=1}^{n} a_{ik} \land b_{kj} \]

A matrix \( W \) is a matrix of \textit{witnesses} iff

If \( c_{ij} = 0 \) then \( w_{ij} = 0 \)
If \( c_{ij} = 1 \) then \( w_{ij} = k \) 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 6:

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 \( \{0, 1, \ldots, M\} \)

<table>
<thead>
<tr>
<th>Running time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Mn^\omega )</td>
<td>[Shoshan-Zwick ’99]</td>
</tr>
</tbody>
</table>

Improves results of
[Alon-Galil-Margalit ’91] [Seidel ’95]
DIRECTED
SHORTEST PATHS
Exercise 7:

Obtain an $O(n^{o\log n})$ time algorithm for computing the **diameter** of an unweighted directed graph.
Using matrix multiplication to compute min-plus products

\[
\begin{pmatrix}
c_{11} & c_{12} \\
c_{21} & c_{22}
\end{pmatrix}
\begin{pmatrix}
O
\end{pmatrix}
= \begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{pmatrix}
\begin{pmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
c'_{11} & c'_{12} \\
c'_{21} & c'_{22}
\end{pmatrix}
\begin{pmatrix}
O
\end{pmatrix}
= \begin{pmatrix}
x^{a_{11}} & x^{a_{12}} \\
x^{a_{21}} & x^{a_{22}}
\end{pmatrix}
\begin{pmatrix}
x^{b_{11}} & x^{b_{12}} \\
x^{b_{21}} & x^{b_{22}}
\end{pmatrix}
\]

\[
c'_{ij} = \sum_k x^{a_{ik} + b_{kj}}
\]

\[
c_{ij} = \text{first}(c'_{ij})
\]
Using matrix multiplication to compute min-plus products

Assume: \(0 \leq a_{ij}, b_{ij} \leq M\)

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

\(n^\omega\) polynomial products \(\times\) \(M\) operations per polynomial product \(\times\) \(Mn^\omega\) operations per max-plus product
Trying to implement the repeated squaring algorithm

\[ D \leftarrow W \]
\[ \text{for } i \leftarrow 1 \text{ to } \log_2 n \]
\[ \text{do } D \leftarrow D \times D \]

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
\{ 
  \[ s \leftarrow (3/2)^{i+1} \]
  \[ B \leftarrow \text{rand}(V, (9n \ln n)/s) \]
  \[ D \leftarrow \min\{ D, D[V,B]*D[B,V] \} \]
\}

Choose a subset of \( V \) of size \( \approx n/s \)

Select the columns of \( D \) whose indices are in \( B \)

Select the rows of \( D \) whose indices are in \( B \)

The is also a slightly more complicated deterministic algorithm with high probability, all distances are correct!
Sampled Distance Products (Z ’98)

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

The matrices get smaller and smaller but the elements get larger and larger.
Sampled Repeated Squaring - Correctness

**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

$\begin{array}{ccc}
n & \times & p \\
p & & n \\
= & & n
\end{array}$

Naïve complexity: $n^2p$

[Coppersmith (1997)] [Huang-Pan (1998)]

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

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

\[ n \times n^{0.29} \times n^{0.29} = n \]

[Coppersmith (1997)]

\[ n \times n^{0.29} \text{ by } n^{0.29} \times n \]

\[ n^{2 + o(1)} \text{ operations!} \]

\[ \alpha = 0.29 \ldots \]
Rectangular Matrix multiplication

\[ p \times p = n \]

[ Huang-Pan (1998) ]

Break into \( q \times q^\alpha \) and \( q^\alpha \times q \) sub-matrices

\[
q = \left( \frac{n}{p} \right)^{\frac{1}{1-\alpha}} \\
\left( \frac{n}{q} \right)^{\omega} \cdot q^2 \\
\approx n^{\omega - \frac{\omega-2}{1-\alpha}} \cdot p^{\frac{\omega-2}{1-\alpha}} \\
\approx n^{1.85} p^{0.54}
\]
Complexity of APSP algorithm

The \( i \)-th iteration:

\[
\begin{align*}
\text{Naïve matrix multiplication} & \quad n \times n \\
\text{"Fast" matrix multiplication} & \quad n \times n/s \\
\min \{ MS \cdot n^{1.85} \left( \frac{n}{s} \right)^{0.54}, \frac{n^3}{s} \} & \leq M^{0.68} n^{2.58}
\end{align*}
\]

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

The elements are of absolute value at most \( Ms \)
All-Pairs Shortest Paths
in graphs with small integer weights

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

<table>
<thead>
<tr>
<th>Running time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Mn^{2.38}$</td>
<td>[Shoshan-Zwick ’99]</td>
</tr>
</tbody>
</table>

Improves results of
[Alon-Galil-Margalit ’91] [Seidel ’95]
All-Pairs Shortest Paths
in graphs with small integer weights

Directed graphs.
Edge weights in \(-M, \ldots, 0, \ldots, M\)

<table>
<thead>
<tr>
<th>Running time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M^{0.68} n^{2.58}$</td>
<td>[Zwick ’98]</td>
</tr>
</tbody>
</table>

Improves results of
[Alon-Galil-Margalit ’91] [Takaoka ’98]
Open problem:
Can APSP in directed graphs be solved in $O(n^\omega)$ time?

[Yuster-Z (2005)]
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.

Also obtained, using a different technique, by Sankowski (2005)
The preprocessing algorithm (YZ ’05)

\[ D \leftarrow W \; ; \; B \leftarrow V \]
for \( i \leftarrow 1 \) to \( \log_{3/2} n \) do
\{
\begin{align*}
  s & \leftarrow (3/2)^{i+1} \\
  B & \leftarrow \text{rand}(B,(9n \ln n)/s) \\
  D[V,B] & \leftarrow \min\{ D[V,B] , D[V,B]*D[B,B] \} \\
  D[B,V] & \leftarrow \min\{ D[B,V] , D[B,B]*D[B,V] \}
\end{align*}
\}
The APSP algorithm

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

Twice Sampled Distance Products
The query answering algorithm

\[ \delta(u,v) \leftarrow D[\{u\}, V] \ast D[V, \{v\}] \]

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.
1. Algebraic matrix multiplication
   a. Strassen’s algorithm
   b. Rectangular matrix multiplication

2. Min-Plus matrix multiplication
   a. Equivalence to the APSP problem
   b. Expensive reduction to algebraic products
   c. Fredman’s trick

3. APSP in undirected graphs
   a. An $O(n^{2.38})$ algorithm for unweighted graphs (Seidel)
   b. An $O(Mn^{2.38})$ algorithm for weighted graphs (Shoshan-Zwick)

4. APSP in directed graphs
   1. An $O(M^{0.68}n^{2.58})$ algorithm (Zwick)
   2. An $O(Mn^{2.38})$ preprocessing / $O(n)$ query answering alg. (Yuster-Z)
   3. An $O(n^{2.38}\log M)$ $(1+\varepsilon)$-approximation algorithm

5. Summary and open problems
Approximate min-plus products

Obvious idea: scaling

\[ \text{SCALE}(A, M, R): \quad a'_{ij} \left\{ \begin{array}{ll}
Ra_{ij} / M & , \quad \text{if } 0 \leq a_{ij} \leq M \\
+\infty & , \quad \text{otherwise}
\end{array} \right. \]

\[
\text{APX-MPP}(A, B, M, R) : \\
A' \leftarrow \text{SCALE}(A, M, R) \\
B' \leftarrow \text{SCALE}(B, M, R) \\
\text{return MPP}(A', B') 
\]

Complexity is \( Rn^{2.38} \), instead of \( Mn^{2.38} \), but small values can be greatly distorted.
Addaptive Scaling

**APX-MPP**\((A,B,M,R)\):

\[
C' \leftarrow \infty \\
\text{for } r \leftarrow \log_2 R \text{ to } \log_2 M \text{ do} \\
\quad A' \leftarrow \text{SCALE}(A,2^r,R) \\
\quad B' \leftarrow \text{SCALE}(B,2^r,R) \\
\quad C' \leftarrow \min\{C', \text{MPP}(A',B')\} \\
\text{end}
\]

Complexity is \(Rn^{2.38} \log M\)

Stretch at most \(1+4/R\)
1. **Algebraic matrix multiplication**  
   a. Strassen’s algorithm  
   b. Rectangular matrix multiplication  

2. **Min-Plus matrix multiplication**  
   a. Equivalence to the APSP problem  
   b. Expensive reduction to algebraic products  
   c. Fredman’s trick  

3. **APSP in undirected graphs**  
   a. An $O(n^{2.38})$ algorithm for unweighted graphs (Seidel)  
   b. An $O(Mn^{2.38})$ algorithm for weighted graphs (Shoshan-Zwick)  

4. **APSP in directed graphs**  
   1. An $O(M^{0.68}n^{2.58})$ algorithm (Zwick)  
   2. An $O(Mn^{2.38})$ preprocessing / $O(n)$ query answering alg. (Yuster-Z)  
   3. An $O(n^{2.38}\log M)$ (1+$\varepsilon$)-approximation algorithm  

5. **Summary and open problems**
Answering distance queries

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

<table>
<thead>
<tr>
<th>Preprocessing time</th>
<th>Query time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Mn^{2.38}$</td>
<td>$n$</td>
<td>[Yuster-Zwick ’05]</td>
</tr>
</tbody>
</table>

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. $mn^{0.5}\log M$
Approximate All-Pairs Shortest Paths
in graphs with non-negative integer weights

**Directed** graphs.
Edge weights in \( \{0, 1, \ldots, M\} \)

\((1+\varepsilon)\)-approximate distances

<table>
<thead>
<tr>
<th>Running time</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (n^{2.38} \log M)/\varepsilon )</td>
<td>[Zwick ’98]</td>
</tr>
</tbody>
</table>
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 $\{1,2,\ldots,n\}$?

An $O(n^{2.5-\varepsilon})$ algorithm for the SSSP problem with edge weights in $\{-1,0,1,2,\ldots,n\}$?
DYNAMIC TRANSITIVE CLOSURE
Dynamic transitive closure

- **Edge-Update**($e$) – add/remove an edge $e$
- **Vertex-Update**($v$) – add/remove edges touching $v$.
- **Query**($u,v$) – is there are directed path from $u$ to $v$?

[Sankowski ’04]

<table>
<thead>
<tr>
<th></th>
<th>Edge-Update</th>
<th>Vertex-Update</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(improving [Demetrescu-Italiano ’00], [Roditty ’03])
Inserting/Deleting and edge

May change $\Omega(n^2)$ entries of the transitive closure matrix
Symbolic Adjacency matrix

\[
\begin{pmatrix}
1 & 0 & x_{13} & x_{14} & x_{15} & 0 \\
 x_{21} & 1 & 0 & x_{24} & x_{25} & 0 \\
 0 & x_{32} & 1 & 0 & x_{35} & x_{36} \\
 0 & 0 & 0 & 1 & 0 & x_{46} \\
 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & x_{56} & 1
\end{pmatrix}
\]

\[\text{det}(A) \neq 0\]
Reachability via adjoint

[Sankowski ’04]

Let $A$ be the symbolic adjacency matrix of $G$. (With 1s on the diagonal.)

There is a directed path from $i$ to $j$ in $G$ iff

$$(\text{adj}(A))_{ij} \neq 0$$
Reachability via adjoint (example)

Is there a path from 1 to 5?

\[
\begin{pmatrix}
0 & 0 & x_{13} & x_{14} & x_{15} & 0 \\
0 & 1 & 0 & x_{24} & x_{25} & 0 \\
0 & x_{32} & 1 & 0 & x_{35} & x_{36} \\
0 & 0 & 0 & 1 & 0 & x_{46} \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & x_{65} & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & x_{13} & x_{14} & x_{15} & 0 \\
x_{21} & 1 & 0 & x_{24} & x_{25} & 0 \\
0 & x_{32} & 1 & 0 & x_{35} & x_{36} \\
0 & 0 & 0 & 1 & 0 & x_{46} \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & x_{65} & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
-x_{15} \\
-x_{13} x_{32} x_{25} \\
+x_{13} x_{35} \\
-x_{13} x_{36} x_{56} \\
-x_{14} x_{46} x_{65} \\
-x_{13} x_{32} x_{24} x_{46} x_{65}
\end{pmatrix}
= \]

\[
-\frac{1}{x_{13} x_{32} x_{24} x_{46} x_{65}}
\]
Dynamic transitive closure

• **Edge-Update** \((e)\) – add/remove an edge \(e\)
• **Vertex-Update** \((v)\) – add/remove edges touching \(v\).
• **Query** \((u,v)\) – is there are directed path from \(u\) to \(v\)?

Dynamic matrix inverse

• **Entry-Update** \((i,j,x)\) – Add \(x\) to \(A_{ij}\)
• **Row-Update** \((i,v)\) – Add \(v\) to the \(i\)-th row of \(A\)
• **Column-Update** \((j,u)\) – Add \(u\) to the \(j\)-th column of \(A\)
• **Query** \((i,j)\) – return \((A^{-1})_{ij}\)
Sherman-Morrison formula

\[ (A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \]

Inverse of a rank one correction
is a rank one correction of the inverse

Inverse updated in \(O(n^2)\) time
$O(n^2)$ update / $O(1)$ query algorithm

[Sankowski ’04]

Let $p \approx n^3$ be a prime number
Assign random values $a_{ij} \in F_p$ to the variables $x_{ij}$
Maintain $A^{-1}$ over $F_p$

Edge-Update $\rightarrow$ Entry-Update
Vertex-Update $\rightarrow$ Row-Update + Column-Update

Perform updates using the Sherman-Morrison formula

Small error probability
(by the Schwartz-Zippel lemma)
Lazy updates

Consider single entry updates

\[ A_k = A_{k-1} + a_k u_k v_k \]
\[ a_k = \pm a_{i_k, j_k} \quad u_k = e_{i_k} \quad v_k = e_{j_k}^T \]

\[ A_k^{-1} = A_{k-1}^{-1} + \alpha_k u_k' v_k' \]
\[ \alpha_k = 1 + a_k v_k A_{k-1}^{-1} u_k = 1 + a_k (A_{k-1}^{-1})_{j_k, i_k} \]

\[ u_k' = A_{k-1}^{-1} u_k = (A_{k-1}^{-1})^{*, i_k} \]
\[ v_k' = v_k A_{k-1}^{-1} = (A_{k-1}^{-1})_{j_k, *} \]
\[ A_k^{-1} = A_0^{-1} + \sum_{i=1}^{k} \alpha_i u_i' v_i' \]
Lazy updates (cont.)

\[ A_k^{-1} = A_0^{-1} + \sum_{i=1}^{k} \alpha_i u'_i v'_i \]

Do not maintain \( A_k^{-1} \) explicitly!
Maintain \( \alpha_i, u'_i, v'_i, i = 1, 2, \ldots, k \)

Querying \( (A_k^{-1})_{r,c} \) \(-\( O(k) \) time

Computing \( \alpha_k, u'_k, v'_k \) \(-\( O(nk) \) time

Queries and updates get more and more expensive!
Lazy updates (cont.)

\[ A_k^{-1} = A_0^{-1} + \sum_{i=1}^{k} \alpha_i u'_i v'_i \]

Query time – \( O(k) \)
Update time – \( O(nk) \)

Compute \( A_k^{-1} \) explicitly after each \( K \) updates

Time required – \( O(M(n, K, n)) \) time
Amortized update time – \( O(nK + M(n, K, n)/K) \)
Update time minimized when \( K \approx n^{0.575} \)

Can be made worst-case
**Even Lazier updates**

\[
A_k^{-1} = A_0^{-1} + \sum_{i=1}^{k} \alpha_i u'_i v'_i
\]

After \( \ell \) updates in positions \((r_1, c_1), (r_2, c_2), \ldots, (r_\ell, c_\ell)\)

maintain:
\[
\alpha_i, (u'_i)_{c_j}, (v'_i)_{r_j}, \text{ for } 1 \leq i, j \leq \ell
\]

Query time \( \sim O(k^2) \)

Update time \( \sim O(k^2) \)

After \( K \), explicitly update \( A_k^{-1} \)
Dynamic transitive closure

- **Edge-Update**($e$) – add/remove an edge $e$
- **Vertex-Update**($v$) – add/remove edges touching $v$.
- **Query**($u,v$) – is there are directed path from $u$ to $v$?

[Sankowski ’04]

<table>
<thead>
<tr>
<th></th>
<th>$n^2$</th>
<th>$n^{1.575}$</th>
<th>$n^{1.495}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge-Update</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Vertex-Update</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>$1$</td>
<td>$n^{0.575}$</td>
<td>$n^{1.495}$</td>
</tr>
</tbody>
</table>

(improving [Demetrescu-Italiano ’00], [Roditty ’03])
Finding triangles in $O(m^{2\omega}/(\omega+1))$ time
[Alon-Yuster-Z (1997)]

Let $\Delta$ be a parameter. $\Delta = m^{(\omega-1)/(\omega+1)}$

High degree vertices: vertices of degree $\geq \Delta$.
Low degree vertices: vertices of degree $< \Delta$.

There are at most $2m/\Delta$ high degree vertices
Finding longer simple cycles

A graph $G$ contains a $C_k$ iff $\text{Tr}(A^k) \neq 0$?

We want simple cycles!
Color coding [AYZ ’95]

Assign each vertex \( v \) a random number \( c(v) \) from 
\( \{0,1,...,k-1\} \).

Remove all edges \( (u,v) \) for which \( c(v) \neq c(u) + 1 \) \( (\text{mod } k) \).

All cycles of length \( k \) in the graph are now simple.

If a graph contains a \( C_k \) then with a probability of at
least \( k^{-k} \) it still contains a \( C_k \) after this process.

An improved version works with probability \( 2^{-O(k)} \).

Can be derandomized at a logarithmic cost.
Sherman-Morrison-Woodbury formula

\[(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^TA^{-1}U)^{-1}V^TA^{-1}\]

Inverse of a rank $k$ correction
is a rank $k$ correction of the inverse
Can be computed in $O(M(n,k,n))$ time.