Let A and B be two $n \times n$ matrices over some field F (for the most part we will work with rationals $\mathbb{Q}$ and reals $\mathbb{R}$ and complex numbers $\mathbb{C}$ or finite fields $\mathbb{Z}_p$ for prime p).

The matrix product $C = AB$ is defined as producing a $n \times n$ matrix C where

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{ik}$$

It is easy to see that the definition allows one to compute C in $O(n^3)$ arithmetic operations.

Surprisingly [Strassen in 1969] showed that one can do better. He developed a $n^{2.81}$ time algorithm using a clever divide and conquer strategy. First consider the following obvious divide and conquer

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

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

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

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

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

8 multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices
plus 4 additions of $\frac{n}{2} \times \frac{n}{2}$ matrices

We get a recurrence of the form

$$T(n) = 8\,T\left(\frac{n}{2}\right) + \Theta(n^2)$$

which leads to $T(n) = \Theta(n^3)$.

However Strassen was able to reduce the number of multiplications to 7 using "Subtraction"

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

where $M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$M_2 = (A_{21} + A_{22}) B_{11}$

$M_3 = A_{11}(B_{12} - B_{22})$

$M_4 = A_{22}(B_{21} - B_{11})$

$M_5 = (A_{11} + A_{12}) B_{22}$

$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$

$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

$$T(n) = 7\,T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$\simeq T(n) = O(n^{\log_2 7}) = O(n^{2.81}).$$

Several subsequent improvements led to Coppersmith and Winograd's algorithm that runs in time $O(n^{2.376})$ in 1990.

This stood for a long time until some recent improvements due to Stothers (2011), Williams (2011) and Le Gall (2014) with current time $n^{2.37287}$.

"Big Open Problem": Can matrix multiplication be done in $n^{2+o(1)}$ time?

Important connections to tensors, symmetry, groups etc. See work of Cohn, Umans, Williams, Alman and others on approaches to finding improved upper bounds and lower bounds.

ω (omega) is the smallest possible exponent for matrix multiplication.

$$2 \leq \omega < 2.37287$$

Strassen's 1969 paper not only showed that matrix multiplication can be done in n^{ω} time but

also many other matrix computations

- computing inverse A^{-1} of $n \times n$ matrix A
- $\det(A)$
- solving $Ax = b$
- computing characteristic polynomial $\det(A - \lambda I)$.
- computing rank (A) and corresponding submatrix.

See link to Zwicker slides for some additional information on how some of the above are accomplished.

In combinatorial applications we encounter two other forms of matrix multiplication.

Boolean matrix multiplication

Given $A, B \in \{0,1\}^{n \times n}$ we define $C = A \circ B$ as

$$C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$$

In other words $C_{ij} = 1$ iff $\exists$ an index k s.t. both A_{ik} and B_{kj} are 1.

It is easy to see that BMM can be reduced to standard integer multiplication. Is BMM easier? Not known.

Min-plus product: given A & B define $A * B$ as

$$C_{ij} = \min_{k=1}^{n} (A_{ik} + B_{kj})$$

It is not clear how to do $A * B$ using standard products but one can achieve some results via complicated reductions.

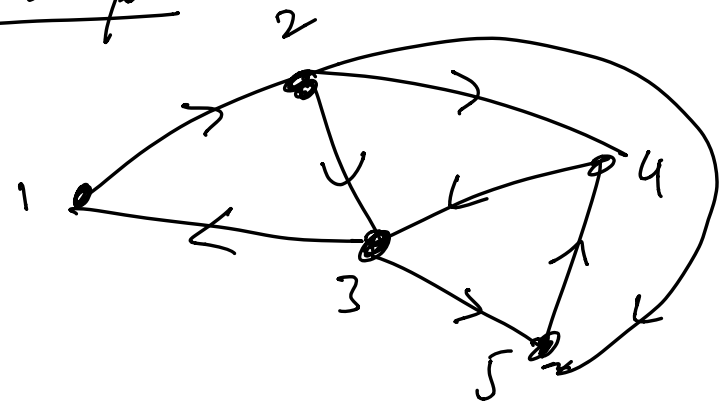
Combinatorial Applications

We aim to give a short sampling of some simple and surprising applications of fast matrix multiplication. There are many other applications which you can find in references.

There are several matrices that one can associate with a graph but perhaps the most common and natural one is the adjacency matrix commonly denoted by A_G which we will

shorten to A when G is clear.
For a directed simple graph G on n vertices A is a $n \times n$ matrix with $A_{ij} = 1$ if (i,j) is an edge and $A_{ij} = 0$ otherwise. Note that A is not necessarily symmetric.
For an undirected graph we define A by bidirecting each edge and taking the adjacency matrix of the resulting directed graph which is the same as setting $A_{ij} = A_{ji} = 1$ if $ij \in E$.

Example



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Let G be a directed graph and A be its adjacency matrix.

Question: What does A^2 represent?

$$A^2_{ij} = \sum_{k=1}^{n} A_{ik} A_{kj}$$

A^2_{ij} = # of walks from i to j of length 2.

Lemma: A^k_{ij} is # of walks from i to j in G.

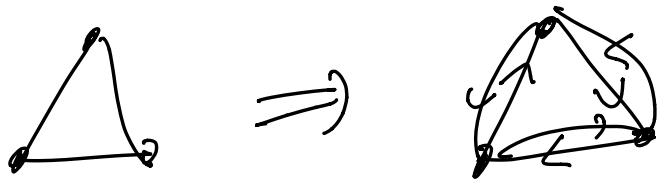
Recall a walk in a directed graph is a sequence of vertices $v_0, v_1, \ldots, v_k$ where $(v_h, v_{h+1}) \in E$ for $0 \le h < k$. Note vertices can repeat unlike a path.

One can prove above lemma by induction on k.

A^k_{ii} counts # of closed walks starting and ending at i

of length k.

For $k=3$ a closed walk in a graph without self loops is a cycle of length 3. In in an undirected graph suppose we want to count # of triangles.



If we bidirected G to get $\vec{G}$ then $A^3_{\vec{G}}$ will allow us to count $\sum_i A^3_{\vec{G}}(i,i)$ which is 6 times # of triangles in G.

Theorem: The # of triangles in a n vertex undirected graph and also the # of triangles incident to each vertex can be computed in $O(n^\omega)$ time.

Interestingly Δ-counting is quite useful in social network analysis. We refer the reader to [see Tim Roughgarden's notes]

For sparse graphs an algorithm with run time $O(m^{3/2})$ is known where m is # of edges.

Boolean Matrix Multiplication & TC

A basic question in graphs is reachability. That is, given G and nodes s, t can s reach t. Can be solved in linear time via standard graph search methods.

The transitive closure of a directed graph $G = (V, E)$ is a graph $G^* = (V, E^*)$ where $(i, j) \in E^*$ iff i can reach j in G.

In other words G^* encodes all-pairs reachability. $A_{G^*}(i, j) = 1$ iff i can reach j in G.

Q: How fast can one compute
A^* given A?

Easy solution is to compute
reachability from each vertex
which takes $O(mn)$ time total.

Can we do better?

Consider A and add a
self loop
to each vertex.
This corresponds to setting $A_{ii} = 1$
for each i.
Now what does A^2 mean?

A^2_{ij} counts length 2 walks in the graph with self loops.

Thus $A^2_{ij} > 0$ iff i can reach j with at most 2 edges.

In fact we can do boolean matrix multiplication.

We can write the process of adding self-loops as $(A \vee I)$ where I is the identity matrix.

Lemma: There is a path from i to j in G iff $(A \vee I)^{n-1} = 1$ where multiplication is boolean.

Above is easy to prove. Since
i can reach j iff there is
a path/walk with at most $n-1$
edges.

We can compute $(A \vee I)^{n-1}$
using $O(\log n)$ matrix multiplications
by repeated squaring.

$B = A \vee I$

for $i = 1$ to $\lceil \log n \rceil$

$B = B \circ B$ (boolean matrix)

Output B.

So total time is $O(n^\omega \log n)$.

One can improve the run time and get rid of the log factor.

Open Problem: Is there a combinatorial algorithm that runs in $O(n^{3-\delta})$ time for any fixed $\delta > 0$?

Matrix multiplication can also be used for dynamic maintenance of transitive closure. See ref.

APSP and Matrix Multiplication

Given a graph with non-negative edge weights APSP is the problem of computing all pairs shortest paths. Using Dijkstra's shortest path algorithm n times one can achieve $O(nm + n^2 \log n)$ time. For dense graphs this is $O(n^3)$. Can we do better?

It is not clear for general integer weights but in some special cases one can indeed do this via matrix multiplication

The complexity of APSP is an important question in fine-grained complexity. See survey by Williams.

Seidel's algorithm for undirected unit weight graphs.

Theorem [Seidel] Given a simple undirected graph with unit edge lengths APSP can be solved in $O(n^\omega \log n)$ time.

Defn: Let $G = (V, E)$ be an undirected simple graph. The square of G, denoted by G^2, is the graph $G^2 = (V, E')$ where $ij \in E'$ iff $ij \in E$ or $\exists k$ s.t. $ik, kj \in E$.

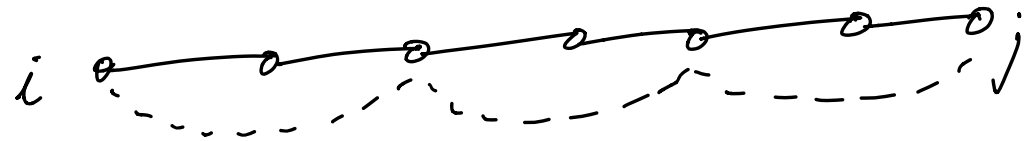
We have already seen that G^2 can be computed in $O(n^\omega)$ time by boolean matrix multiplication.

Idea: can we compute APSP in G by computing APSP in G^2?

For this we need to understand some shortest path properties in

G and G^2.

Claim: If $d_G(i,j)$ is even then $d_{G^2}(i,j) = \frac{1}{2} d_G(i,j)$.



From figure it is clear that

$$d_{G^2}(i,j) \leq \frac{1}{2} d_G(i,j)$$

It is also easy to observe that

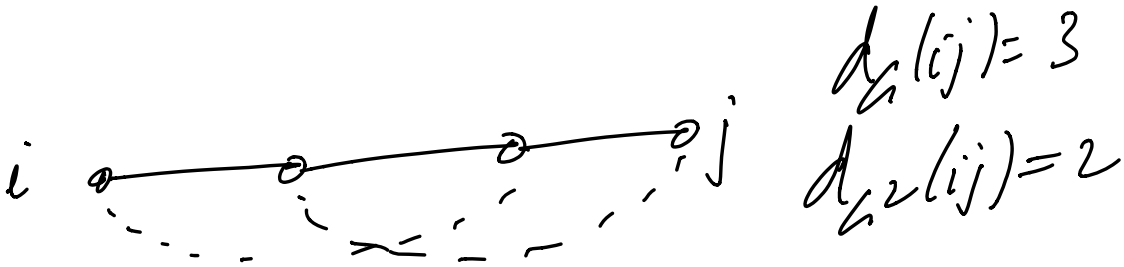
$$d_G(i,j) \leq 2\, d_{G^2}(i,j)$$

for every pair. Hence

$\frac{1}{2} d_G(i,j) = d_{G^2}(i,j)$ if $d_G(i,j)$ is even.

Claim: If $d_G(ij)$ is odd and $d_{G^2}(ij) > 1$ then

$$d_{G^2}(ij) \leq \left\lceil \frac{1}{2} d_G(ij) \right\rceil .$$



Therefore: $d_G(ij) = 2 d_{G^2}(ij)$

or $d_G(ij) = 2 d_{G^2}(ij) - 1$

and if we can figure out which of the cases we are in then we can compute d_G from d_{G^2}.

Lemma: Suppose $d_G(ij)$ is even then for every neighbor $k \neq i$
$d_{G^2}(kj) \geq d_{G^2}(ij)$

Lemma: Suppose $d_G(ij)$ is odd and (≥ 3) then for every neighbor $k \neq i$
$d_{G^2}(kj) \leq d_{G^2}(ij)$ and
$\exists$ at least one neighbor k'
s.t. $d_{G^2}(k'j) < d_{G^2}(ij)$

Let $N(i)$ be the set of neighbors of i.

Corollary: $d_{G^2}(i,j) = 2d_{G^2}(i,j) - 1$

if $\sum_{k \in N(i)} d_{G^2}(k,j) < \deg(i)\, d_{G^2}(i,j)$

and otherwise $d_{G^2}(i,j) = 2d_{G^2}(i,j)$

How do we compute $\sum_{k \in N(i)} d_{G^2}(k,j)$

for all i and j?

Let B be the distance matrix of G^2. Note B is not a boolean matrix.

Then $(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$

$$= \sum_{k \in N(i)} B_{kj}$$

Thus we can check for each ij if $\deg(i) B_{ij} > (AB)_{ij}$ and conclude whether $d_A(ij) = 2 d_{A^2}(ij)$ or $2 d_{A^2}(ij) - 1$.

We note that Seidel's algorithm requires one boolean matrix multiplication and one integer multiplication in each recursive call.

We leave it as an exercise to finalize details and conclude that the total exec time is $O(n^\omega \log n)$.

Exercise: Can one use only Boolean matrix multiplication?

Many questions and related results. See Zwick's excellent slides on topic.

APSP and min plus product

Why min plus product?

Captures APSP

Let W be matrix with edge weights W_{ij} is weight of edge (i,j)
$= \infty$ or large number if no edge

Lemma: W^k where product is min-plus captures distances using at most k edges.

Hence by repeated squaring one can get APSP in $O(MPP(n) \log n)$ time. With some code in

$O(MPP(n))$ time.

Exponentiation to implement MPP.

Let α be some parameter

Want $A * B$ min plus product

Instead do $A' \cdot B'$ where

$A'_{ij} = \alpha^{A_{ij}}$ $\quad B'_{ij} = \alpha^{B_{ij}}$

Let $C = A'B'$

$$C_{ij} = \sum_{k=1}^{n} \alpha^{A_{ik} + B_{kj}}$$

For sufficiently large α can deduce

from $c_{ij} \min_{k=1}^{n} (A_{ik} + B_{kj})$

but numbers become big.

# of bits needed to exploit above when edge weights are not large.