Boolean Matrix Multiplication

Given Definition: Boolean Matrix Multiplication (BMM)

Input: Two Boolean matrices $A, B \in \{0,1\}^{n \times n}$

Task: Compute Boolean matrix C defined by

$$C_{ij} = \bigvee_{k=1}^{n} A_{ik} \wedge B_{k,j}$$

(Matrix Product in Boolean semiring)

("or" $\bigvee$, "and" $\wedge$)

($\exists k$ s.t. $A[i,k]=1$ and $B[k,j]=1$?)

Algorithms:

- $O(n^3)$ (naive)
- $O(n^3/\log^2 n)$ [Arlazarov et al. '70] "4 Russians"
- ⋮
- $O\left(\frac{n^3 \cdot \text{poly}(\log\log n)}{\log^4 n}\right)$ [Yu '15]

} "combinatorial"

- Reduction to matrix multiplication on real numbers: "non-combinatorial" "algebraic"

  $O(n^{2.81})$ [Strassen '69]

  ⋮

  $O(n^{2.38})$ [Coppersmith, Winograd '87]

  $O(n^{2.37})$ [Le Gall '14]

$$C_{ij} = \bigvee_{k=1}^{n} A_{ik} \wedge A_{kj} \quad \text{vs.} \quad \hat{C}_{ij} = \sum_{k=1}^{n} \underbrace{A_{ik} \cdot A_{kj}}_{= A_{ik} \wedge A_{kj}}$$

Observation: $C_{ij} = \begin{cases} 1 & \text{if } \hat{C}_{ij} \geq 1 \\ 0 & \text{o.w.} \end{cases}$

$$C_{ij} \neq 0 \iff \hat{C}_{ij} \neq 0$$

Remark: While the subcubic matrix multiplication alg for real numbers above are theoretically fast, they are often considered impractical because of large "hidden" constants in $O()$-notation. Algorithms not relying on fast matrix mult. are often called "combinatorial", but this term has no precise definition.

Def: ω is the infimum over all β s.t. matrix multiplication has an $O(n^\beta)$-time algorithm. $\Rightarrow$ MM is in time $O(n^{\omega+\varepsilon})$ for every $\varepsilon > 0$; Naively: $O(n^\omega)$ time

Strassen's Algorithm

Key Insight: Using fewer multiplications to multiply two 2×2 matrices A and B

Naive:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}}_{A} \times \underbrace{\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}}_{B} = \underbrace{\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}}_{C}$$

$c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21}$

$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22}$

$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21}$

$c_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22}$

8 multiplications

recursive scheme:

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

$\Rightarrow T(n) \leq O(n^3)$

Strassen's Improvement:

$m_1 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22})$

$m_2 = (a_{21} + a_{22}) \cdot b_{11}$

$m_3 = a_{11} \cdot (b_{12} - b_{22})$

$m_4 = a_{22} \cdot (b_{21} - b_{11})$

$m_5 = (a_{11} + a_{12}) \cdot b_{22}$

$m_6 = (a_{21} - a_{11}) \cdot (b_{11} + b_{12})$

$m_7 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22})$

$c_{11} = m_1 + m_4 - m_5 + m_7$

$c_{12} = m_3 + m_5$

$c_{21} = m_2 + m_4$

$c_{22} = m_1 - m_2 + m_3 + m_6$

7 multiplications

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

$\Rightarrow T(n) \leq O(n^{\log_2 7}) = O(n^{2.8074})$

Remark: algorithm works for any ring (e.g. $(\mathbb{R}, +, \cdot)$), but not for any semi-ring (e.g. $(\mathbb{R} \cup \{\infty\}, \min, +)$)

Reason: algorithm exploits taking the inverse w.r.t. addition; not guaranteed to exist in semi-ring.

This is what makes the algorithm "non-combinatorial".

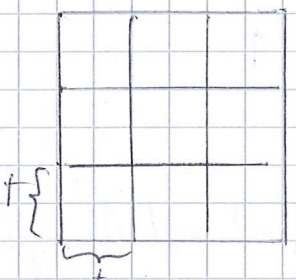
(e.g. in BMM problem we can be truly subcubic by shifting to a richer algebraic structure than Boolean semi-ring)

Four Soviets Algorithm [Arlazarov, Dinic, Kronrod, Faddeev '70]

Computes $C = A \times B$ (Boolean) in time $O(n^3/\log n)$

Idea: 1. Preprocess A and construct lookup table
2. Speed up naive algorithm using lookup table

① Divide A into blocks of size $t \times t$ (where $t = 0.5 \log n$)



number of blocks: $(\frac{n}{t})^2 = \frac{n^2}{\log^2 n}$

For each of the $(\frac{n}{t})^2$ blocks X:

- Precompute product $X \cdot v$ for every vector $v \in \{0,1\}^t$

t inner products: time $O(t^2)$ per v

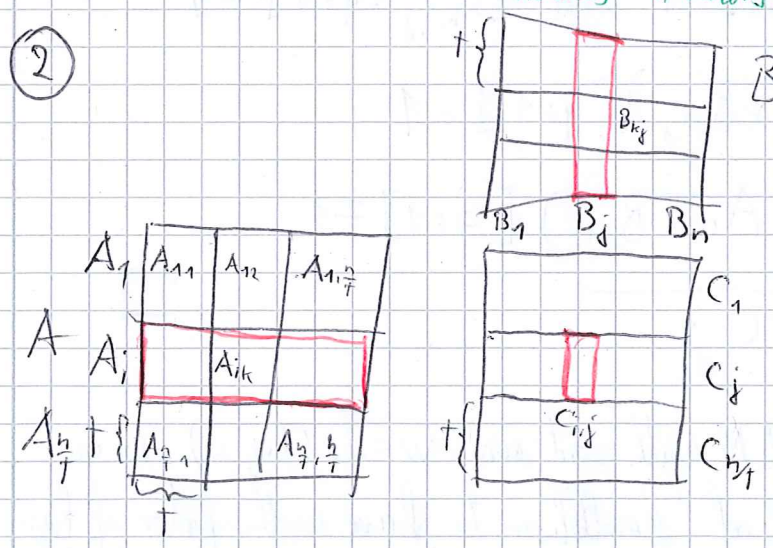
- Store each result in lookup table

→ Given X, v: retrieve $X \cdot v$ in time $O(t)$

Total preprocessing time: $O((\frac{n}{t})^2 \cdot 2^t \cdot t^2) = n^2 2^t = n^{2.5} = O(\frac{n^3}{\log n})$

#blocks #vectors inner product

② Compute C in time $O(n \cdot \frac{n}{t} \cdot \frac{n}{t} \cdot t)$



$= O(\frac{n^3}{t}) = O(\frac{n^3}{\log n})$

$$C_{ij} = A_{i\,\ast} \times B_{\ast j} = A_{i1} \times B_{1j} \vee \dots \vee A_{i,\frac{n}{t}} \times B_{\frac{n}{t},j}$$

$$= \bigvee_{k=1}^{n/t} \underbrace{A_{ik} \times B_{kj}}_{\text{Time } O(t)} \quad \text{(Table lookup)}$$

Correctness: Lemma:

$$C_{i,j}[q+1] = (A \times B)[t \cdot i + q, j] \quad \forall \begin{matrix} 1 \le i \le \frac{n}{t} \\ 1 \le j \le n \\ 0 \le q \le t-1 \end{matrix}$$

Proof: If $C_{i,j}[q+1] = 1$:

$$\Rightarrow \left(\bigvee (A_{i,k} \times B_{k,j})\right)[q+1] = 1$$

$$\Rightarrow \exists k^* \text{ s.t. } (A_{i,k^*} \times B_{k^*,j})[q+1] = 1$$

$$\Rightarrow A_{i,k^*}[q+1] \times B_{k^*,j} = 1$$

A_{i,k^*}: row of

$$\Rightarrow \exists k' : A_{i,k^*}[q+1, k'] = 1 \text{ and } B_{k^*,j}[k'] = 1$$

$$\Rightarrow A[t \cdot i + q, t \cdot k^* + k'] = 1 \text{ and } B[t \cdot k^* + k', j] = 1$$

$$\Rightarrow (A \times B)[t \cdot i + q, j] = 1$$

If $(A \times B)[t \cdot i + q, j] = 1$:

$\Rightarrow \exists k^*$ s.t.

$$A[t \cdot i + q, k^*] = 1 \text{ and } B[k^*, j] = 1$$

We can write $k' = k^* \cdot t + q'$ where $1 \le k^* \le \frac{n}{t}$, $0 \le q' < t$

Consider $A_{i,k^*} \times B_{k^*,j}$

$$A_{i,k^*}[q+1, q'+1] = 1 \text{ and } B_{k^*,j}[q'+1] = 1$$

$$\Rightarrow (A_{i,k^*} \times B_{k^*,j})[q+1] = 1$$

$$\Rightarrow \underbrace{\left(\bigvee A_{i,k} \times B_{k,j}\right)}_{= C_{i,j}}[q+1] = 1$$

□

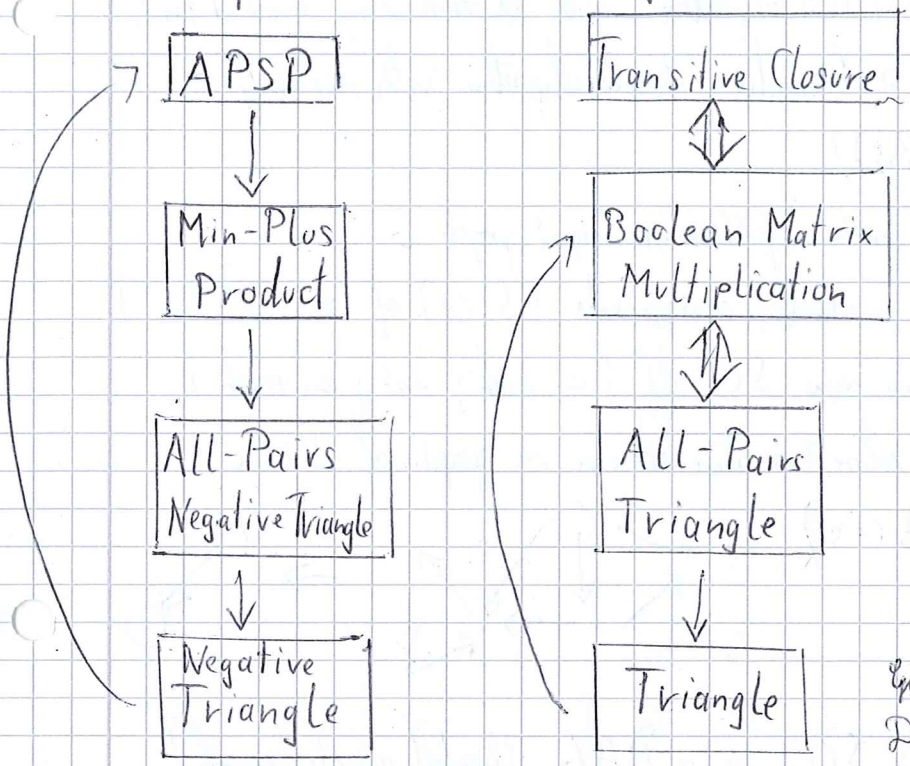
Remark: On a Word RAM with word size $w = \Omega(\log n)$, we can exploit "bit-level" parallelism to shave another factor of $\log n$.

$O(\log n)$ { □

$\underbrace{\square}_{O(\log n)}$ □ $O(1)$ operations to compute inner product (bit-wise AND)

Subcubic Equivalences

(for "Combinatorial" Algorithms)



given directed graph G, compute, for every pair of nodes u and v if there is a path from u to v (TC (matrix))

Given unweighted undirected graph G with $V=[n]$
Decide $\forall i \in I, j \in J$, is there a triangle (i,j,k) for some $k \in K$

Given unweighted, undirected G. Does it contain a triangle

All of these reductions are "combinatorial"
$\Rightarrow$ If one of these problems has a "combinatorial" algorithm with running time $O(n^{3-\varepsilon})$, then all of them have such algorithms

Example: Triangle $\longrightarrow$ Boolean Matrix ~~Mult~~ Multiplication

1 ~~Compute Boolen Product~~ $C := A$

Let A be adjacency matrix of G; i.e. $A[i,j] = \begin{cases} 1 \text{ if } \exists \text{ edge } \{i,j\} \\ 0 \text{ otherwise} \end{cases}$

Algorithm:

1. Compute $C := A \times A$

$$C_{ij} = \bigvee_{k=1}^{n} A_{i,k} \wedge A_{k,j}$$

1 oracle call

2. Compute $\bigvee_{i,j} A_{ij} \wedge C_{ij}$ $\quad O(n^2)$

$$= \bigvee_{i,j,k} A_{ij} \wedge A_{ik} \wedge A_{kj}$$

$= 1$ iff there is a triangle

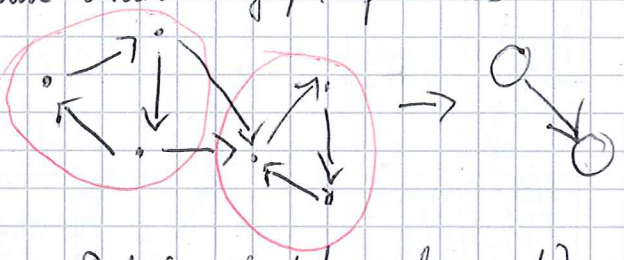
($\bigvee$ amounts to $\exists$)

From Transitive Closure to Boolean Matrix Multiplication

Theorem: If there is a BMM algorithm with running time $T(n)$, then there is a transitive closure algorithm with running time $T'(n) = O(T(n))$

Proof: The algorithm works as follows (for input graph G)

(1) Compute strongly connected components (SCCs) of G $O(n^2)$
- Nodes i, j are in same SCC iff i can reach j and j can reach i
- It suffices to solve transitive closure on graph of SCCs (contracting SCCs)



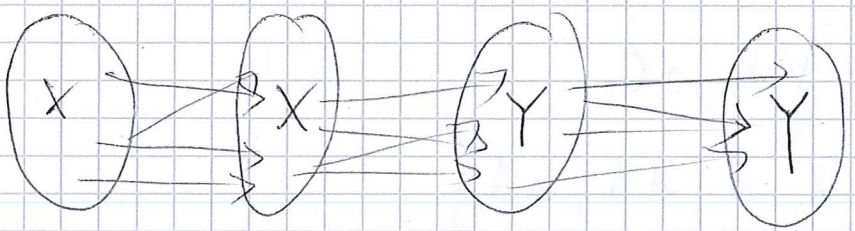
- Graph of SCCs is a DAG (directed acyclic graph)

(2) Compute topological order $\prec$ on DAG D $O(n^2)$
edge (i,j) in D $\Rightarrow i \prec j$

(3) Recursive approach
X = First $\frac{n}{2}$ nodes of D in topological order
Y = Last $\frac{n}{2}$ nodes of D in topological order
M: $\frac{n}{2} \times \frac{n}{2}$ matrix s.t. $M[i,j] = \begin{cases} 1 & \text{if } \exists \text{ edge from } i\text{-th node of } X \text{ to } j\text{-th node of } Y \text{ in } D \\ 0 & \text{otherwise} \end{cases}$

Observation: $TC(D) = TC(D[X]) \times M \times TC(D[Y])$

TC (matrix in DAG D) — subgraph of D induced by X — Boolean matrix multiplication



Running Time: $T'(n) = 2T'\left(\frac{n}{2}\right) + O(T(n))$
$= O(T(n))$ (Master Theorem)