max planck institut
informatik

Complexity Theory of Polynomial-Time Problems

Lecture 1: Introduction, Easy Examples

Karl Bringmann and Sebastian Krinninger

Audience

no formal requirements, but:

NP-hardness,

satisfiability problem,

how to multiply two matrices,

dynamic programming,

all-pairs-shortest-path problem,

Dijkstra‘s algorithm

… if you (vaguely) remember at least half of these things,
then you should be able to follow the course

NP-hardness

suppose we want to solve some problem

fastest algorithm that we can come up with: $O(2^n)$

should we search further for an **efficient** algorithm?

prove NP-hardness!

an efficient algorithm would show P=NP

we may assume that **no efficient algorithm exists**

relax the problem: approximation algorithms, fixed parameter tractability, average case, heuristics…

What about polynomial time?

suppose we want to solve another problem

we come up with an $O(n^2)$ algorithm → **polynomial time** = efficient

big data: input is too large for $O(n^2)$ algorithm

should we search for faster algorithms?

P vs NP is too coarse

we need **fine-grained complexity**
to study **limits of big data**

Conditional Lower Bounds

which **barriers** prevent subquadratic algorithms?

consider a well-studied problem P:

best-known running time $O(n^c)$

conjecture that it has no $O(n^{c-\varepsilon})$ algorithm for any $\varepsilon > 0$

$\Downarrow$

relate another problem Q to P via a **reduction**

→ *conditional lower bound*

Hard problems

SAT: given a formula in conj. normal form on n variables

is it satisfiable?

conjecture: no $O(2^{(1-\varepsilon)n})$ algorithm (SETH)

OV: given n vectors in $\{0,1\}^d$ (for small d)

are any two orthogonal?

conjecture: no $O(n^{2-\varepsilon})$ algorithm

APSP: given a weighted graph with n vertices

compute the distance between any pair of vertices

conjecture: no $O(n^{3-\varepsilon})$ algorithm

3SUM: given n integers

do any three sum to 0?

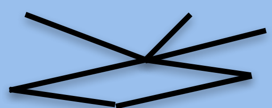
conjecture: no $O(n^{2-\varepsilon})$ algorithm

Relations = Reductions

transfer hardness of one problem to another one by reductions

problem P

instance I



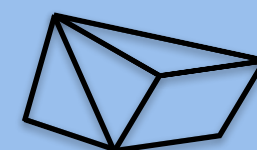
size n

reduction

$r(n)$ time

problem Q

instance J



size $s(n)$

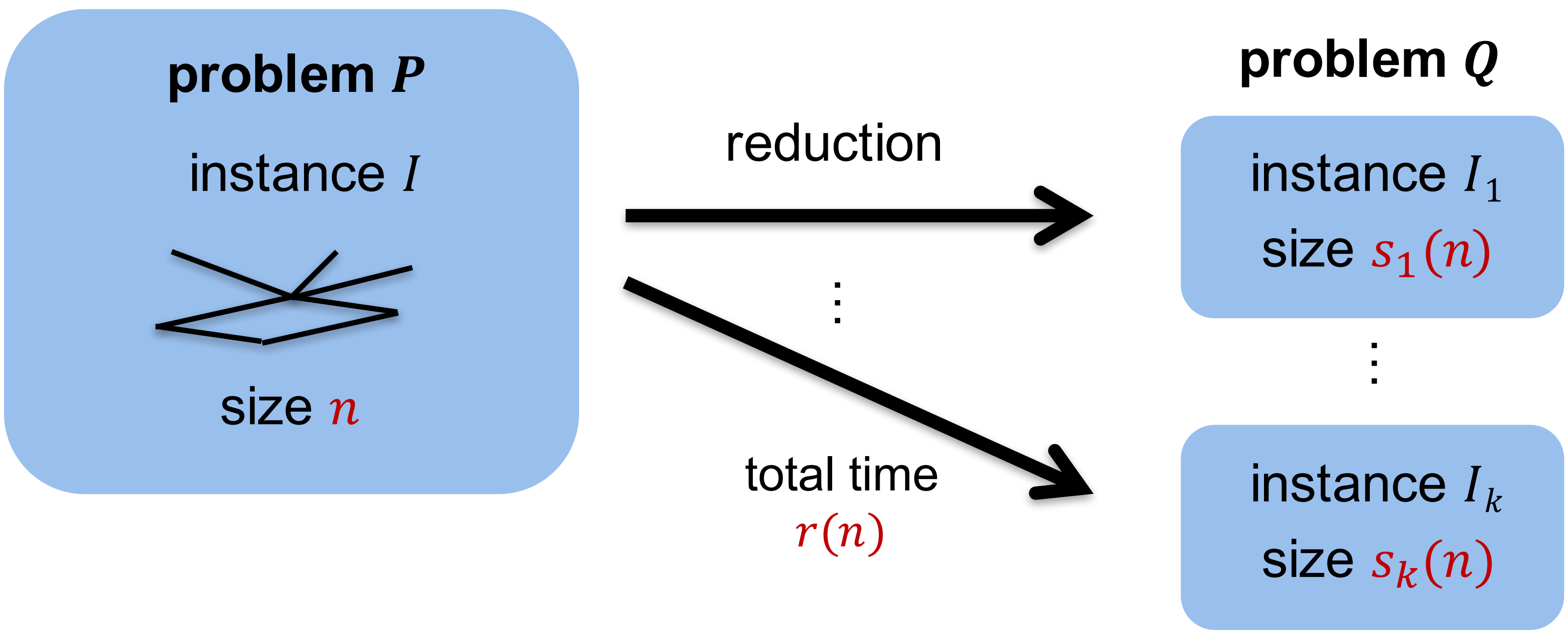
I is a ‘yes’-instance $\Longleftrightarrow$ J is a ‘yes’-instance

$t(n)$ algorithm for Q implies a $r(n) + t(s(n))$ algorithm for P

if P has no $r(n) + t(s(n))$ algorithm then Q has no $t(n)$ algorithm

Relations = Reductions

transfer hardness of one problem to another one by reductions



$t(n)$ algorithm for Q implies a $r(n) + \sum_{i=1}^{k} t(s_i(n))$ algorithm for P

max planck institut
informatik

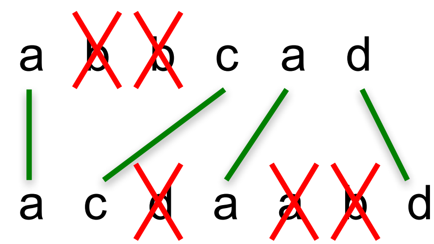
Showcase Results

longest common subseq. $O(n^2)$ SETH-hard $n^{2-\varepsilon}$

edit distance, longest palindromic subsequence, Fréchet distance...

[B.,Künnemann'15, Abboud,Backurs,V-Williams'15]

given two strings x, y of length n, compute the **longest string** z that is a **subsequence** of both x and y



Showcase Results

longest common subseq. $O(n^2)$ SETH-hard $n^{2-\varepsilon}$

edit distance, longest palindromic subsequence, Fréchet distance...

[B.,Künnemann'15, Abboud,Backurs,V-Williams'15]

we can stop searching for faster algorithms!

in this sense, conditional lower bounds replace NP-hardness

$O(n^{2-\varepsilon})$ algorithms are unlikely to exist

improvements are at least as hard as a **breakthrough for SAT**

Showcase Results

longest common subseq. $O(n^2)$ SETH-hard $n^{2-\varepsilon}$

edit distance, longest palindromic subsequence, Fréchet distance...

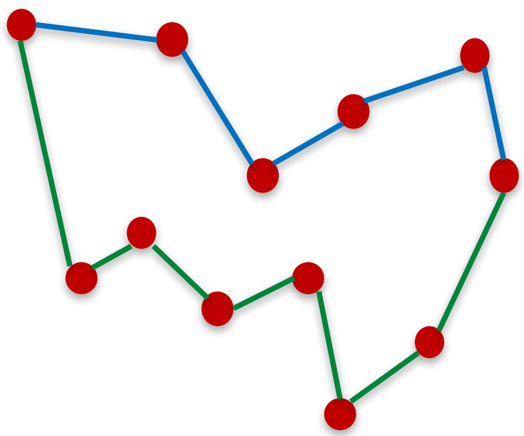
[B.,Künnemann'15, Abboud,Backurs,V-Williams'15]

bitonic TSP $O(n^2)$ $O(n \log^4 n)$

longest increasing subsequence, matrix chain multiplication...

[de Berg,Buchin,Jansen,Woeginger'16]

given n points in the plane, compute a **minimum tour connecting all points** among all tours consisting of **two x-monotone parts**



Showcase Results

longest common subseq.	$O(n^2)$	SETH-hard $n^{2-\varepsilon}$
edit distance, longest palindromic subsequence, Fréchet distance...		[B.,Künnemann'15, Abboud,Backurs,V-Williams'15]
bitonic TSP	$O(n^2)$	$O(n \log^4 n)$
longest increasing subsequence, matrix chain multiplication...		[de Berg,Buchin,Jansen,Woeginger'16]
maximum submatrix	$O(n^3)$	APSP−hard $n^{3-\varepsilon}$
minimum weight triangle, graph centrality measures...		[Backurs,Dikkala,Tzamos'16]

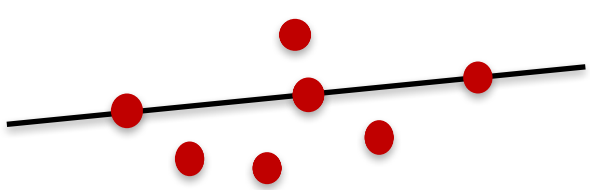
given matrix A over $\mathbb{Z}$, **choose a submatrix** (consisting of consecutive rows and columns of A) **maximizing the sum of all entries**

-3	2	-2	0
-2	5	7	-2
1	3	-1	1
3	-2	0	0

Showcase Results

longest common subseq.	$O(n^2)$	SETH-hard $n^{2-\varepsilon}$
edit distance, longest palindromic subsequence, Fréchet distance...		[B.,Künnemann'15, Abboud,Backurs,V-Williams'15]
bitonic TSP	$O(n^2)$	$O(n \log^4 n)$
longest increasing subsequence, matrix chain multiplication...		[de Berg,Buchin,Jansen,Woeginger'16]
maximum submatrix	$O(n^3)$	APSP−hard $n^{3-\varepsilon}$
minimum weight triangle, graph centrality measures...		[Backurs,Dikkala,Tzamos'16]
colinearity	$O(n^2)$	3SUM−hard $n^{2-\varepsilon}$
motion planning, polygon containment...		[Gajentaan,Overmars'95]

given n points in the plane,
are any three of them on a line?



max planck institut
informatik

Showcase Results

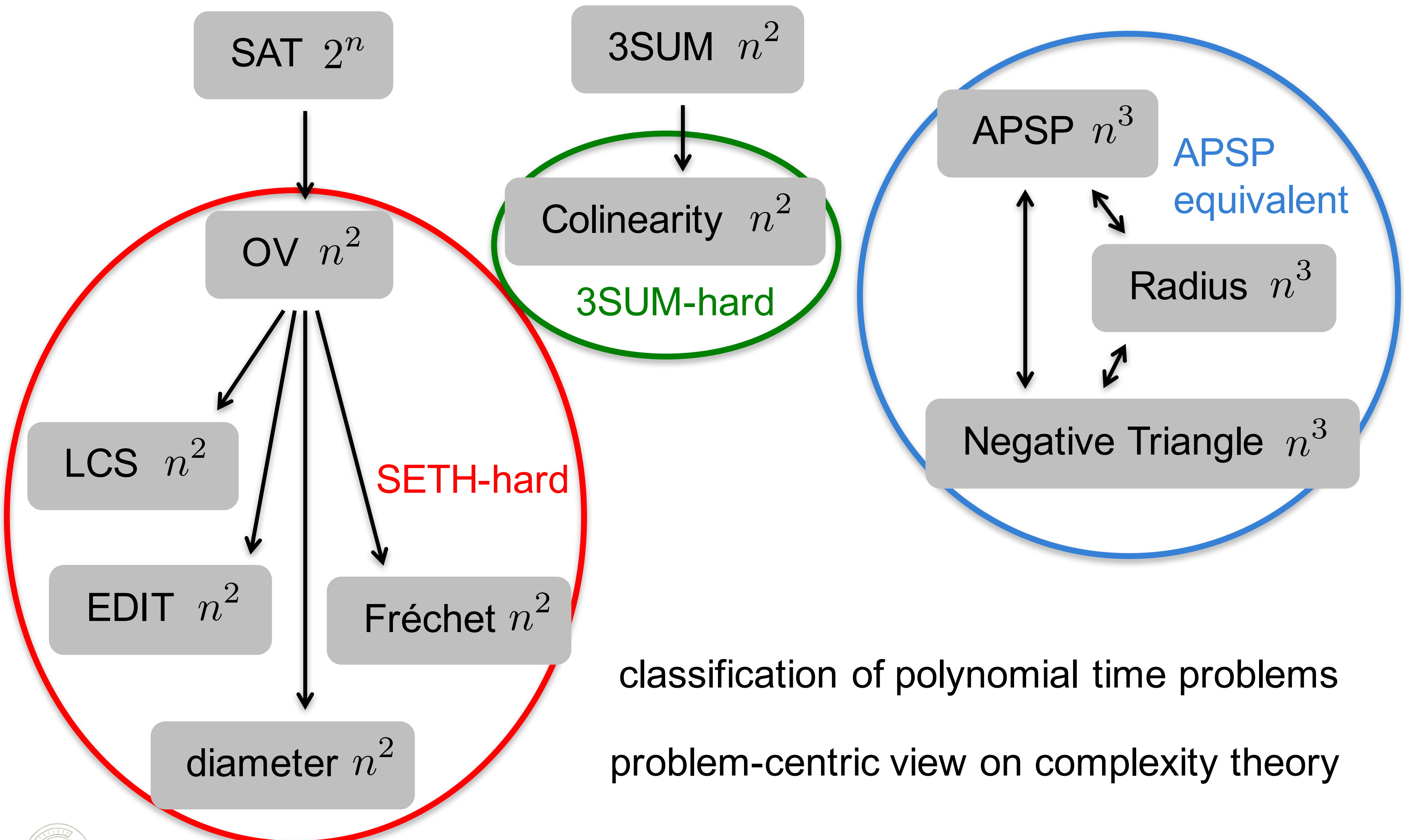
longest common subseq.	$O(n^2)$	SETH-hard $n^{2-\varepsilon}$
edit distance, longest palindromic subsequence, Fréchet distance...		[B.,Künnemann'15, Abboud,Backurs,V-Williams'15]
bitonic TSP	$O(n^2)$	$O(n \log^4 n)$
longest increasing subsequence, matrix chain multiplication...		[de Berg,Buchin,Jansen,Woeginger'16]
maximum submatrix	$O(n^3)$	APSP−hard $n^{3-\varepsilon}$
minimum weight triangle, graph centrality measures...		[Backurs,Dikkala,Tzamos'16]
colinearity	$O(n^2)$	3SUM−hard $n^{2-\varepsilon}$
motion planning, polygon containment...		[Gajentaan,Overmars'95]

Open: optimal binary search tree $O(n^2)$

knapsack $O(nW)$

many more...

Complexity Inside P



classification of polynomial time problems

problem-centric view on complexity theory

Machine Model

complexity theory is (to some extent) independent of the machine model – but only up to polynomial factors

we have to fix a machine model!

Turing Machine:

any (single-tape) Turing machine takes time $\Omega(n^2)$ for recognizing **palindromes**

abbcaacbba

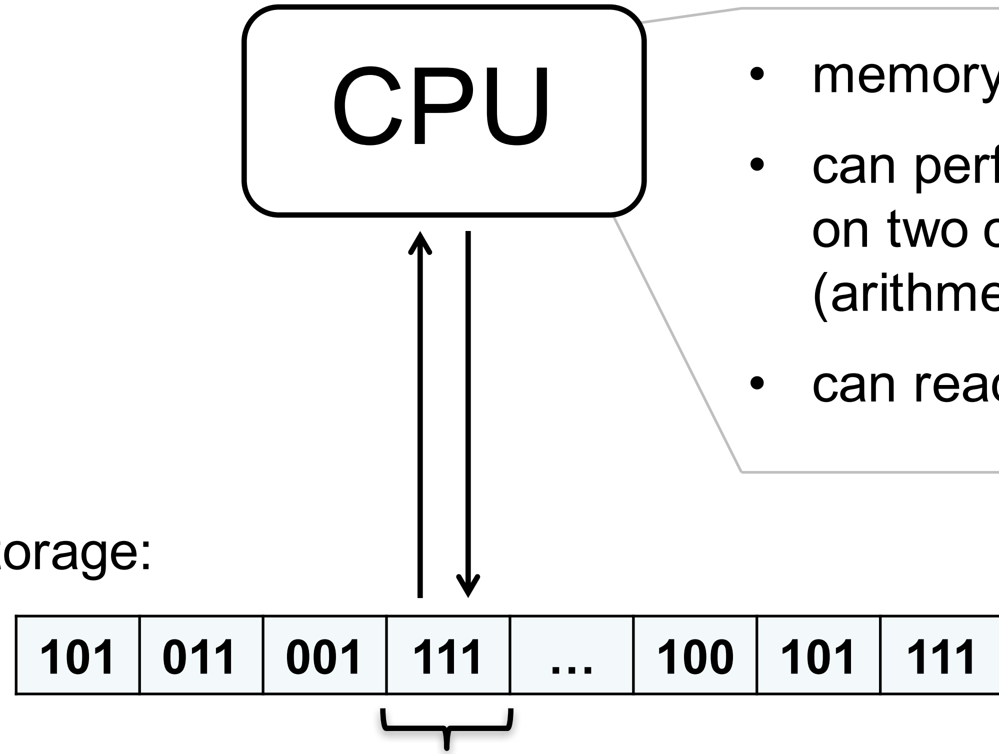
this does not apply to real computers!

Machine Model

Random Access Machine (RAM):

- memory of O(1) cells
- can perform all reasonable operations on two cells in O(1) time (arithmetic + logical operations…)
- can read/write any cell in O(1) time

storage:



cell / word

consisting of $\Theta(\log n)$ bits

can recognize **palindromes** in time $O(n)$

the details do not matter!

More Discussion

What about unconditional lower bounds?

no tools available beyond $\Omega(n \log n)$

What if the hypotheses are wrong?

NP-hardness was in the same situation 40 years ago

relations between problems will stay

suggests ways to attack a problem + which problems to attack

Conditional Lower Bounds …

… allow to classify polynomial time problems

… are an analogue of NP-hardness

yield good reasons to stop searching for faster algorithms

should belong to the basic toolbox of theoretical computer scientists

… allow to search for new algorithms with better focus

improve SAT before longest common subsequence…

non-matching lower bounds suggest better algorithms

… motivate new algorithms

relax the problem and study approximation algorithms,
parameterized running time, …

Content of the Course

we study **core problems** SAT, OV, APSP, 3SUM, and others

conditional lower bounds: from each of these hypotheses

algorithms: learn fastest known algorithms for core problems

fine-grained complexity is a young field of research

we will see many open problems & possibilities for BA/MA-theses

II. An Example for OV-hardness

Orthogonal Vectors Hypothesis

Input: Sets $A, B \subseteq \{0,1\}^d$ of size n

Task: Decide whether there are $a \in A, b \in B$ such that $a \perp b$

$$\Leftrightarrow \sum_{i=1}^{d} a_i \cdot b_i = 0$$

$$\Leftrightarrow \text{for all } 1 \le i \le d\text{: } a_i = 0 \text{ or } b_i = 0$$

$A = \{(1,1,1), (1,1,0), (1,0,1), (0,0,1)\}$

$B = \{(0,1,0), (0,1,1), (1,0,1), (1,1,1)\}$

trivial $O(n^2 d)$ algorithm

best known algorithm: $O(n^{2-1/O(\log c)})$ where $d = c \log n$ [Lecture 03]

OV-Hypothesis: no $O(n^{2-\varepsilon}\text{poly}(d))$ algorithm for any $\varepsilon > 0$

„OV has no $O(n^{2-\varepsilon})$ algorithm, even if $d = \text{polylog}\, n$"

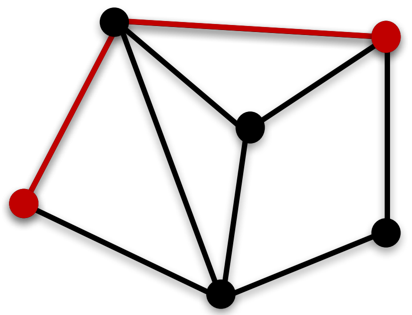
max planck institut informatik

Graph Diameter Problem

Input: An unweighted graph $G = (V, E)$

Task: Compute the largest distance between any pair of vertices

$$= \max_{u,v \in V} d_G(u, v)$$



diameter 2

Easy algorithm:

Single-source-shortest-paths:

Dijkstra's algorithm: $O(m + n \log n)$

All-pairs-shortest-paths:

Dijkstra from every node: $O\big(n(m + n \log n)\big) \leq O(n\, m \log n)$

from this information we can compute the diameter in time $O(n^2)$

OV-Hardness Result

OV

sets $A, B \subseteq \{0,1\}^d$

of size n

reduction

→

time $O(dn)$

diameter

graph G

$O(n)$ vertices

$O(dn)$ edges

$O(n^{2-\varepsilon}\text{poly}(d))$ algorithm $\Leftarrow$ $O((nm)^{1-\varepsilon})$ algorithm

Thm: Diameter has no $O((nm)^{1-\varepsilon})$ algorithm unless the OV-Hypothesis fails. [Roditty,V-Williams'13]

Proof

OV

sets $A, B \subseteq \{0,1\}^d$

of size n

reduction

time $O(dn)$

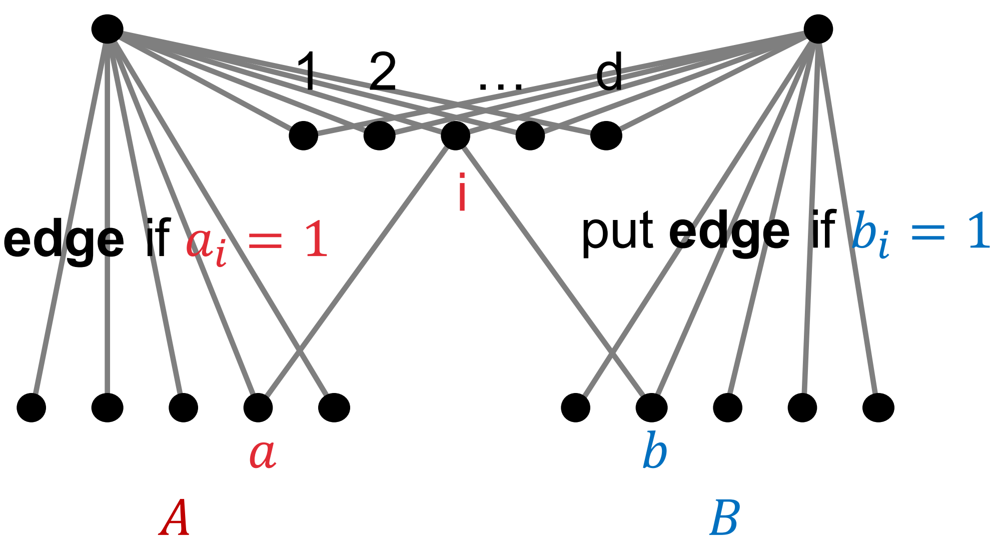
diameter

graph G

$O(n)$ vertices

$O(dn)$ edges

Proof: can assume: every vector has at least one ‚1‘



$d(a, b) = 2 \Leftrightarrow$
a, b not orthogonal

diameter = 3 $\Leftrightarrow$
there exists an
orthogonal pair

Proof

OV

sets $A, B \subseteq \{0,1\}^d$

of size n

reduction → time $O(dn)$

diameter

graph G

$O(n)$ vertices

$O(dn)$ edges

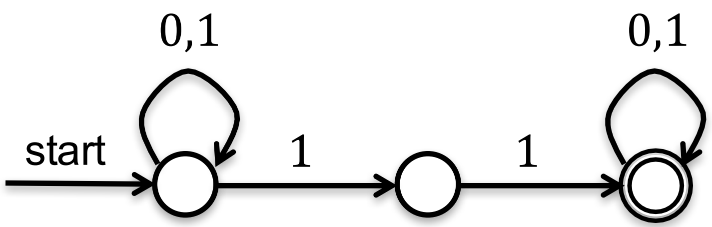
Remark: Even deciding whether the diameter is ≤ 2 or ≥ 3 has no $O((nm)^{1-\varepsilon})$ algorithm unless OVH fails.

There is no $(3/2 - \varepsilon)$-approximation for the diameter in time $O((nm)^{1-\varepsilon})$ unless OVH fails.

III. Another Example for OV-hardness

NFA Acceptance Problem

nondeterministic finite automaton G accepts input string s if there is a walk in G from starting state to some accepting state, labelled with s



string: 01011010

dynamic programming algorithm in time $O(|s||G|)$:

$$T[i] := \textit{set of states reachable via walks labelled with } s[1..i]$$

$$T[0] := \{\text{starting state}\}$$

$$T[i] := \{v \mid \exists u \in T[i-1] \text{ and } \exists \text{ transition } u \rightarrow v \text{ labelled } s[i]\}$$

OV-Hardness Result

OV

sets $A, B \subseteq \{0,1\}^d$

of size n

reduction

time $O(dn)$

NFA acceptance

grammar G, string s

$|G| = O(dn)$

$|s| = O(dn)$

$O(n^{2-\varepsilon}\text{poly}(d))$ algorithm $\Leftarrow$ $O((|s|\ |G|)^{1-\varepsilon})$ algorithm

Thm: NFA acceptance has no $O((|s|\ |G|)^{1-\varepsilon})$ algorithm unless OVH fails. [Impagliazzo]

max planck institut informatik

Proof

OV

sets $A, B \subseteq \{0,1\}^d$

of size n

reduction → time $O(dn)$

NFA acceptance

grammar G, string s

$|G| = O(dn)$

$|s| = O(dn)$

Proof:

fix some $a \in A$:

in string s:

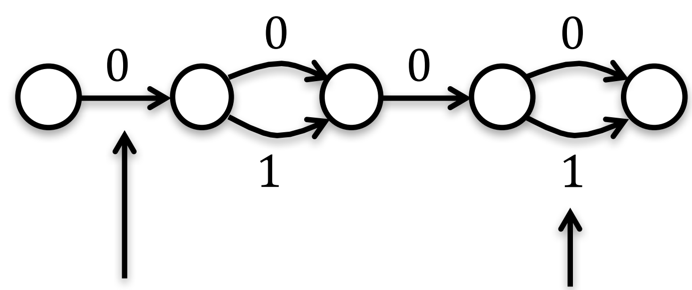
0011

$\uparrow$

$= a_1 a_2 \ldots a_d$

fix some $b \in B$:

in NFA G:



if $b_i = 1$

if $b_i = 0$

Proof

OV

sets $A, B \subseteq \{0,1\}^d$

of size n

reduction → time $O(dn)$

NFA acceptance

grammar G, string s

$|G| = O(dn)$

$|s| = O(dn)$

Proof:

fix some $a \in A$:

in string s:

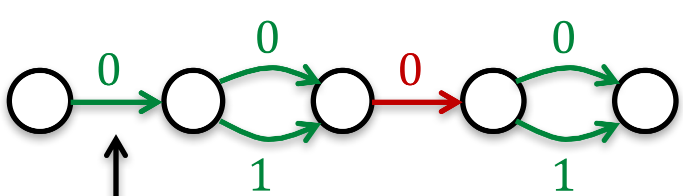
0011

$\uparrow$

$= a_1 a_2 \dots a_d$

fix some $b \in B$:

in NFA G:



if $b_i = 1$

if $b_i = 0$

Proof

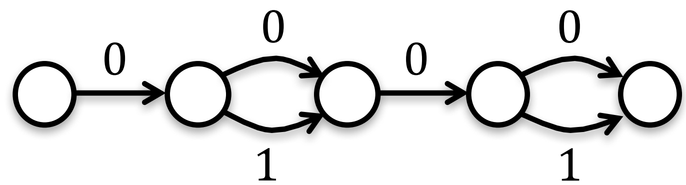
fix some $a \in A$:

in string s:

0011

fix some $b \in B$:

in NFA G:



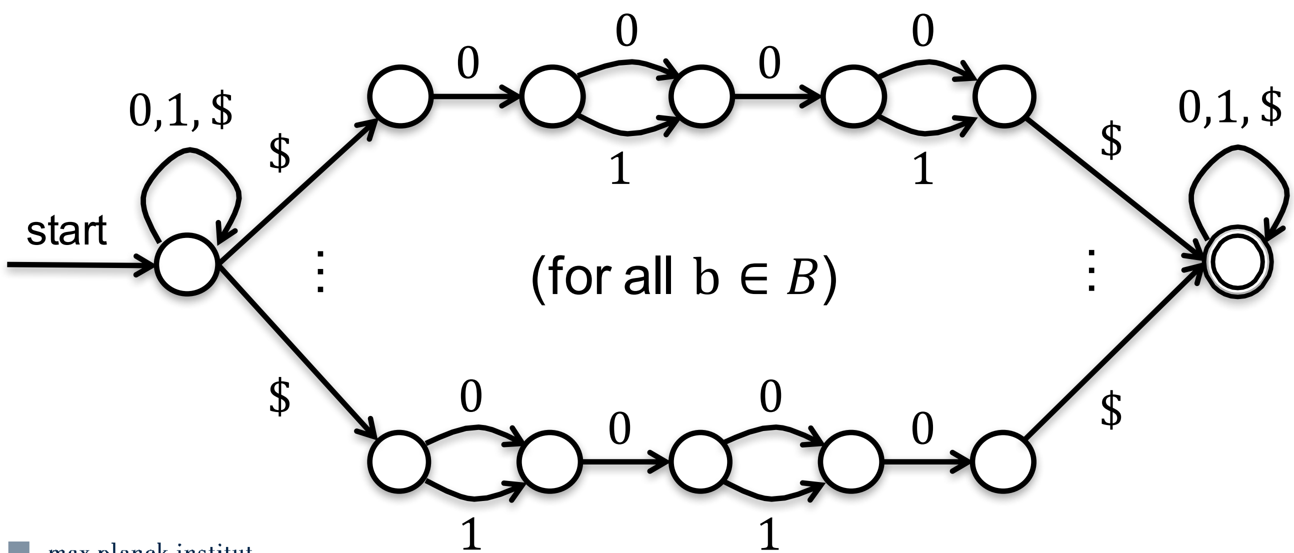
string $\boldsymbol{s} = \$1100\$0110\$ \dots \$0011\$$

(for all $a \in A$)

✔ equivalent to OV instance

✔ size $|s| = |G| = O(dn)$

NFA $\boldsymbol{G}$**:**



VI. Four Russians

Method of Four Russians

Arlazarov, Dinic, Kronrod, and Faradzev 1970

Algorithm for Boolean matrix multiplication

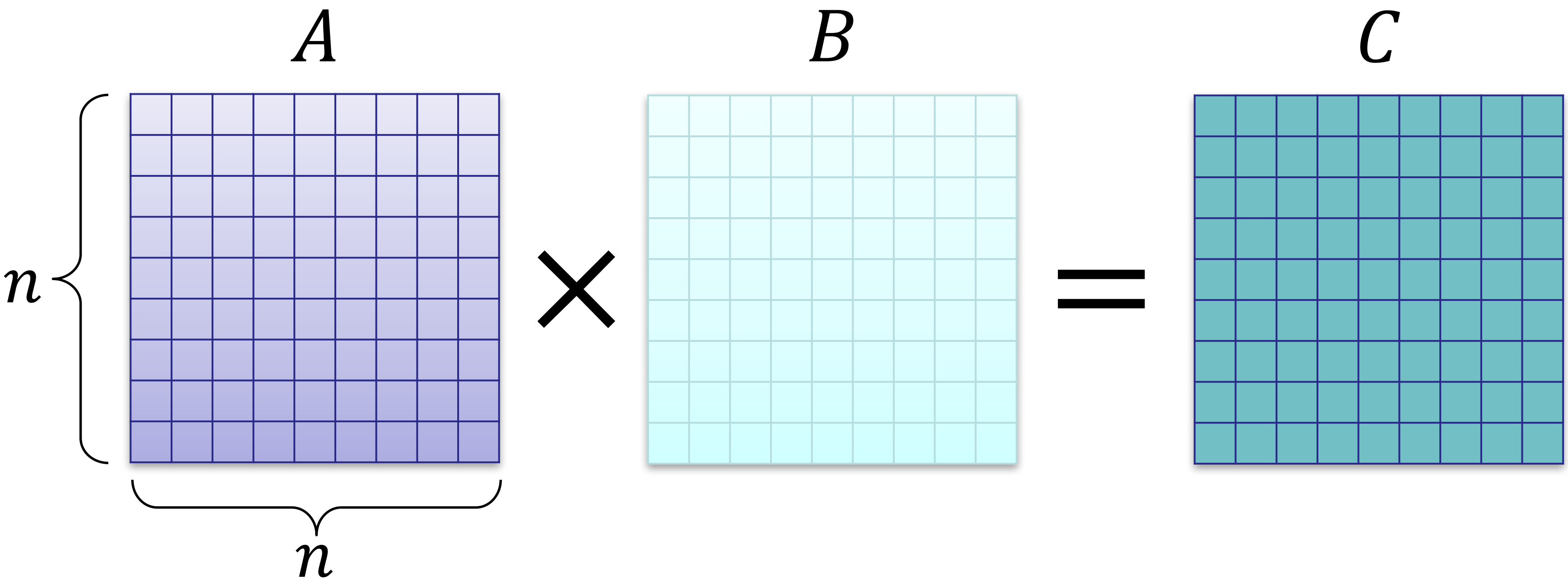
Not all of them were Russian…

Better name: Four Soviets??

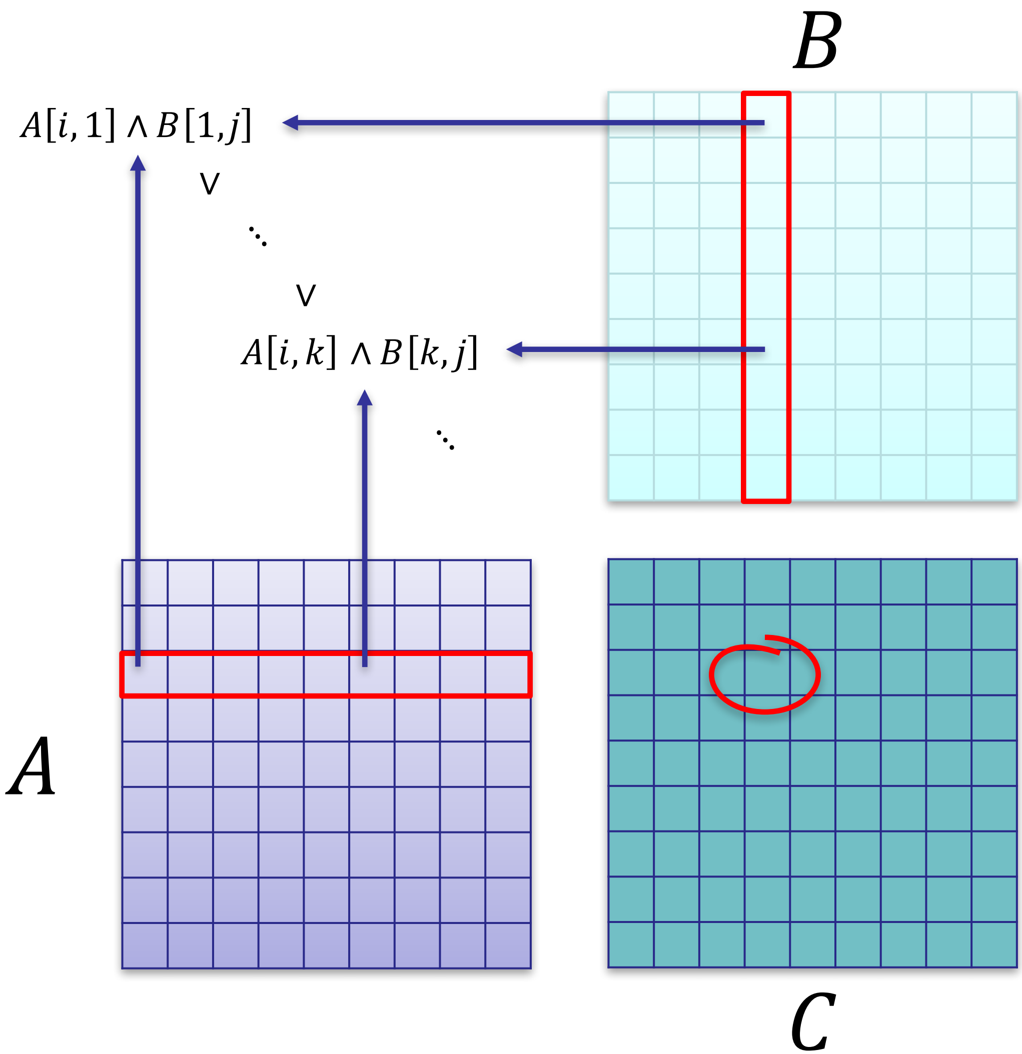
Boolean Matrix Multiplication

Input: Boolean (0/1) matrices A and B

Output: $A \times B$ where $+$ is OR and $*$ is AND



Naïve Algorithm



Running time:

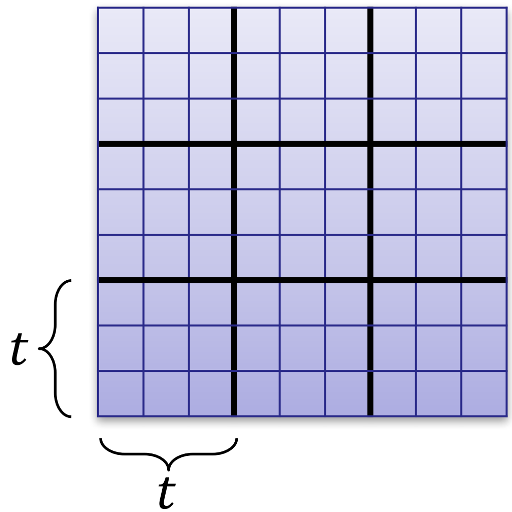
- time $O(n)$ per inner product
- #inner products: n^2
- $\Rightarrow O(n^3)$ total time

$$C[i,j] = \bigvee_{1 \le k \le n} A[i,k] \wedge B[k,j]$$

Is there a k such that $A[i,k] = 1$ and $B[k,j] = 1$?

Main Idea

Divide A into blocks of size $t \times t$



We use $t = 0.5 \log n$

Number of blocks:

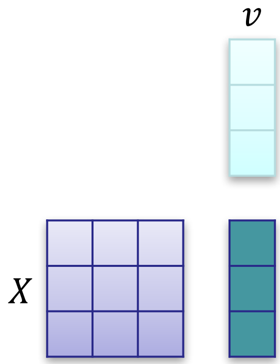
$$\left(\frac{n}{t}\right)^2 = \frac{n^2}{\log^2 n}$$

1. Preprocess blocks and construct lookup table

2. Speed up naïve algorithm using lookup table

Preprocessing a Block X

1. For every t-dimensional vector v: precompute product $X \cdot v$



inner products: t

$\Rightarrow O(t^2)$ time

2. Store results in lookup table: $\Rightarrow$ retrieve $X \cdot v$ in time $O(t)$

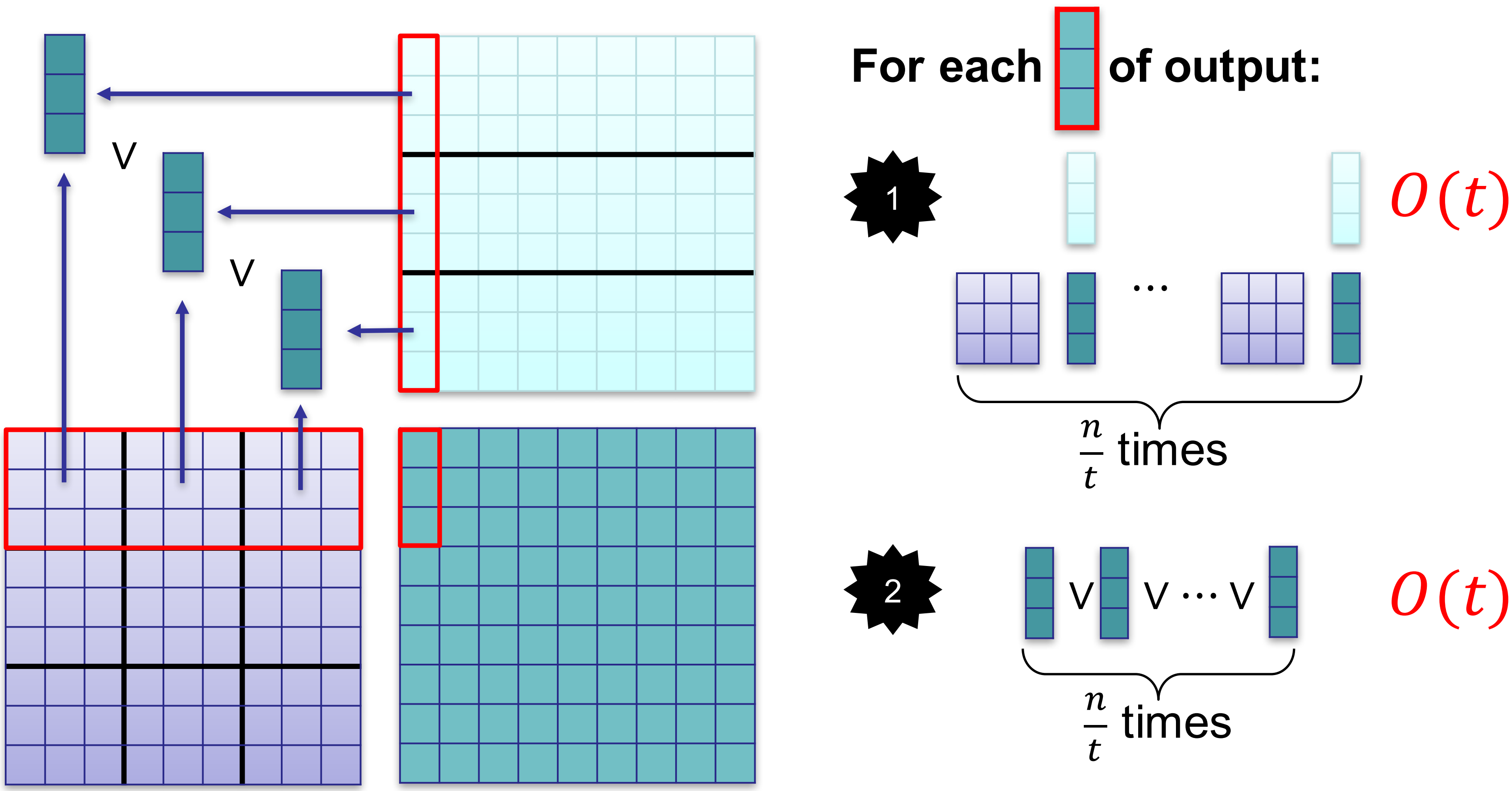
t-dimensional vectors: $2^t = 2^{0.5 \log n} = n^{0.5}$

Total time: $\left(\frac{n}{t}\right)^2 \times 2^t \times t^2 = n^2 2^t = n^{2.5} = O\left(\frac{n^3}{\log n}\right)$

#blocks #vectors inner product

max planck institut informatik

Multiplying with Lookup Table



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

Summary

Key property: small number of possible cell entries

Main idea: speedup from lookup tables after preprocessing

Discussion:

- In preprocessing: need to prepare for **all** t-dimensional input vectors
- Counter-intuitive at first ❢
  Only *some* of the t-dimensional vectors might really appear in B
- # different t-dimensional vectors in general: $2^t = n^{0.5}$
- # t-dimensional vectors per block in matrix multiplication: n
- $\Rightarrow$ Reusability outweighs preprocessing cost
- $\Rightarrow$ Four Russians in some sense a **charging trick**:
  Charge running time to the t-dim. vectors instead of the columns

Beyond Four Russians

Very general technique:

- Matrix problems, dynamic programming problems (e.g. LCS)
- "Shaving off" logarithmic factors popular for certain problems
- $\Rightarrow$ Race to fastest algorithm

Example: All-pairs shortest paths (APSP)
Floyd-Warshall: $O(n^3)$

State of the art: $O\left(\frac{n^3}{2^{\Omega(\sqrt{\log n})}}\right)$

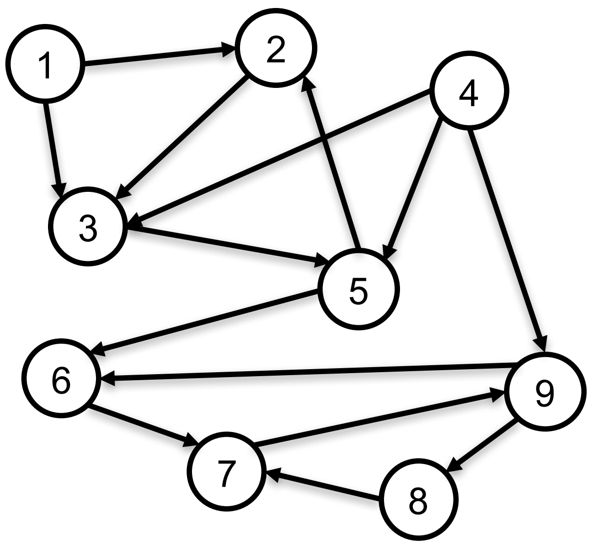
$2^{\sqrt{\log n}}$ grows faster than $\log^c n$ for any constant c

The Polynomial Method

- Ideas from circuit complexity
- We will teach it in May

Transitive Closure Problem

Directed graph G, n nodes



$TC[i,j] = 1$
iff i can reach j

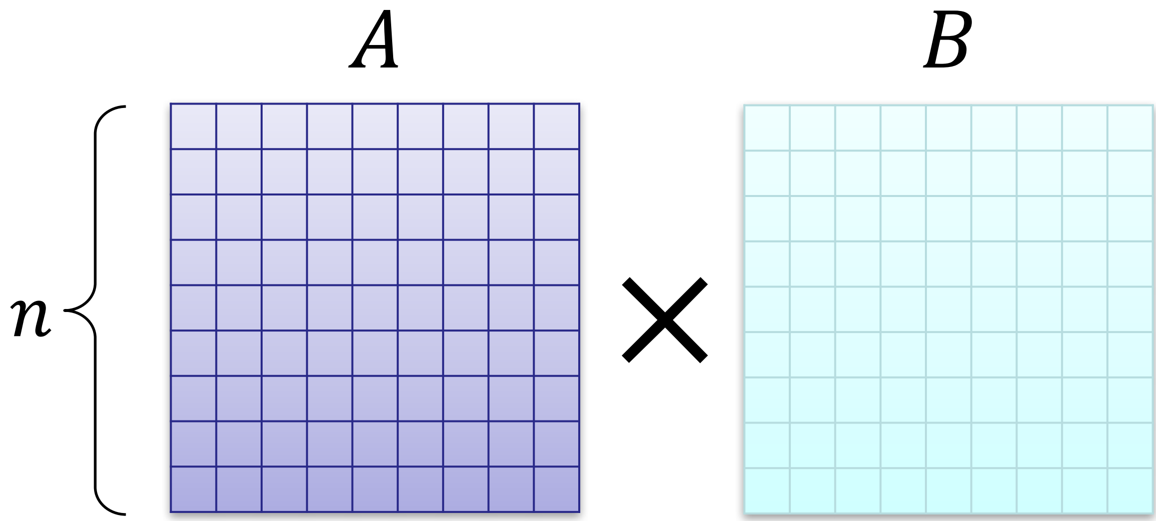
TC matrix

1	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1	1

Thm: BMM in time $O(T(n)) \Leftrightarrow$ TC in time $O(T(n))$

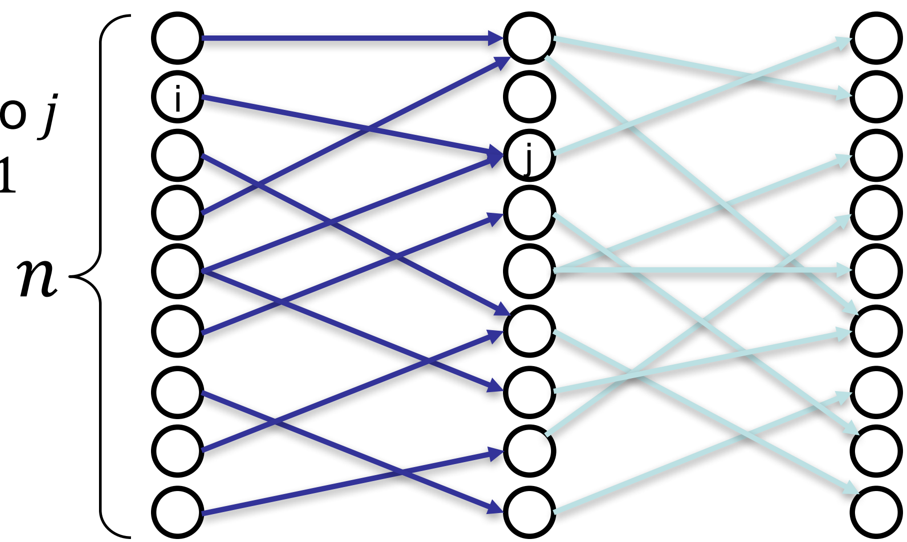
Naïve Algorithm: $O(n^3)$ by breadth-first search from each node

Reduction: From BMM ($T'(n)$) to TC ($T(n)$)



2-layered graph:

edge from i to j iff $\mathrm{A}[i,j]=1$



$TC = A \times B$

$T'(n) = T(3n)$

Reduction: From TC ($T'(n)$) to BMM ($T(n)$)

A: adjacency matrix of G
$A[i,j] = 1$ if and only if G has edge (i,j)
I: identity matrix

Fact: $(A \vee I)^k[i,j] = 1$ if and only if $\exists$path from i to j of length at most k

Fact: $TC = (A \vee I)^n$

Repeated squaring:
$$M^2 = M \times M$$
$$M^4 = M^2 \times M^2$$
$$M^8 = M^4 \times M^4$$
...

Lem: TC can be computed using $\log n$ Boolean matrix multiplications

Drawback of Reduction

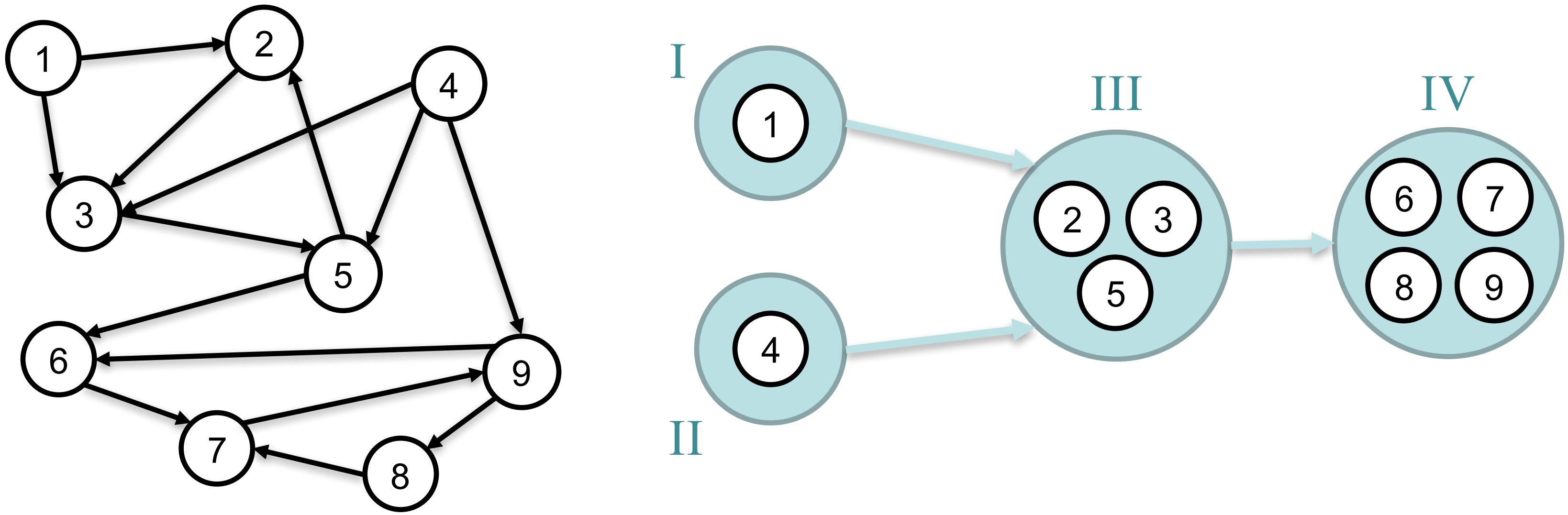
Lem: TC can be computed with $\log n$ Boolean matrix multiplications

$$T'(n) = T(n) \times \log n = \frac{n^3}{\log n} \times \log n = n^3$$

Log-factor improvement of Four Russians is gone!

Goal: Better reduction with $T'(n) = O(T(n))$

Reducing Problem to DAG



1. Compute strongly connected components (SCCs)
   - i and j in same component iff i can reach j and j can reach i
   - Suffices to solve problem on graph of SCCs
   - Graph of SCCs is directed acyclic graph (DAG)

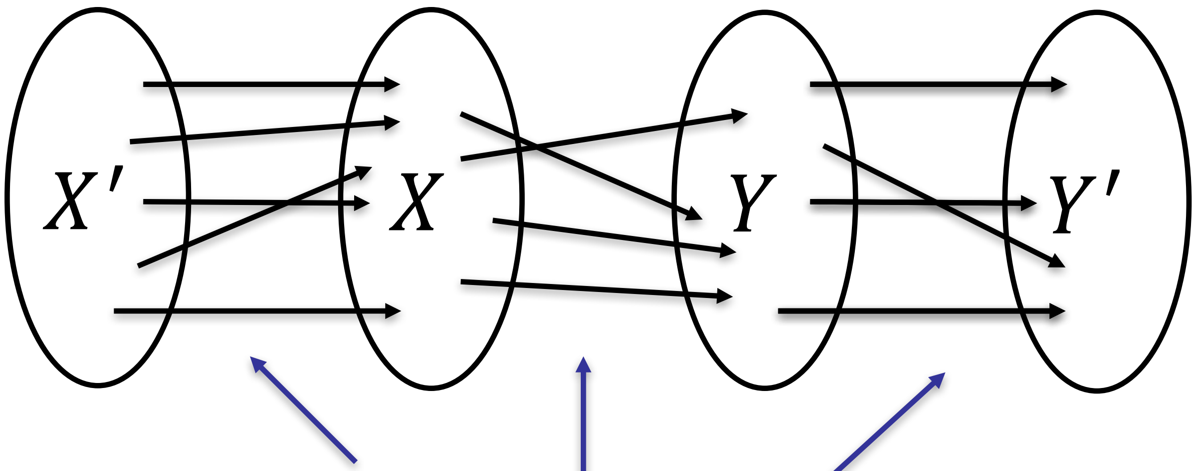
2. Compute topological order $\prec$ on DAG: edge (i,j) in DAG $\Rightarrow i \prec j$

$O(n^2)$

Recursive Transitive Closure

Input: DAG G with n nodes in topological order

X: First $n/2$ edges in topological order
Y: Last $n/2$ edges in topological order } **RECURSE**
M: Adjacency matrix of edges between X and Y



$$TC(G) = TC(X) \times M \times TC(Y)$$

Master Theorem*

$$T'(n) = 2T'\left(\frac{n}{2}\right) + O\big(T(n)\big) = O(T(n))$$

* $2T'\left(\frac{n}{2}\right) \leq cT'(n)$ for some $0 < c < 1$

Summary

BMM and TC have same asymptotic time complexity

Same status:

- All-pairs shortest paths (APSP) and
- Min-plus matrix multiplication