Subcubic Equivalences

Reductions so far:

Problem P, instance I, size n $\xrightarrow[\tau(n) \text{ time}]{\text{reduction}}$ Problem Q, instance J, size $s(n)$

I is a 'yes' instance $\iff$ J is a 'yes' instance

Then: $t(n)$-time algorithm for Q implies $\tau(n) + t(s(n))$-time algorithm for P

(Or: If P has no $\tau(n) + t(s(n))$ time alg., then Q has no $t(n)$-time algorithm)

More sophisticated reduction:

Problem P: instance I, size n — reduction, total time $\tau(n)$ → Problem Q: instance I_1, size n_1; … ; instance I_k, size n_k

instance I_i might depend on results for instances $I_1, \dots, I_{i-1}$

Def.: A subcubic reduction from P to Q is an algorithm $\mathcal{A}$ for P oracle access to Q s.t.

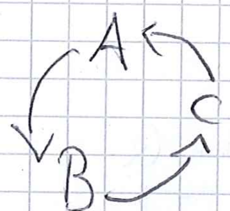
- For every instance x of P, $\mathcal{A}(x)$ solves the problem P on x
- Excluding oracle calls, $\mathcal{A}$ runs in time $O(n^{3-\gamma})$ (for some constant $\gamma > 0$) on instances of size n
- For every constant $\varepsilon > 0$ there is a constant $\delta > 0$ such that for every instance x of P of size n we have $\sum_i n_i^{3-\varepsilon} \leq n^{3-\delta}$, where n_i is the size of the instance in the i-th oracle call to Q in $\mathcal{A}(x)$

Observation: Assume there is a subcubic reduction from P to Q. Then
If ~~there is a~~ Q has an $O(n^{3-\varepsilon})$-time algorithm for some $\varepsilon > 0$,
then P has an $O(n^{3-\delta})$-time algorithm for some $\delta > 0$.

Notation: subcubic reduction: $P \leq Q$
subcubic equivalent: $P \equiv Q$ if $P \leq Q$ and $Q \leq P$

Lemma (Transitivity): If $A \leq B$ and $B \leq C$, then $A \leq C$
Pf: Homework

Thus, if $A \leq B$, $B \leq C$, and $C \leq A$, then A, B, C subcubic equivalent



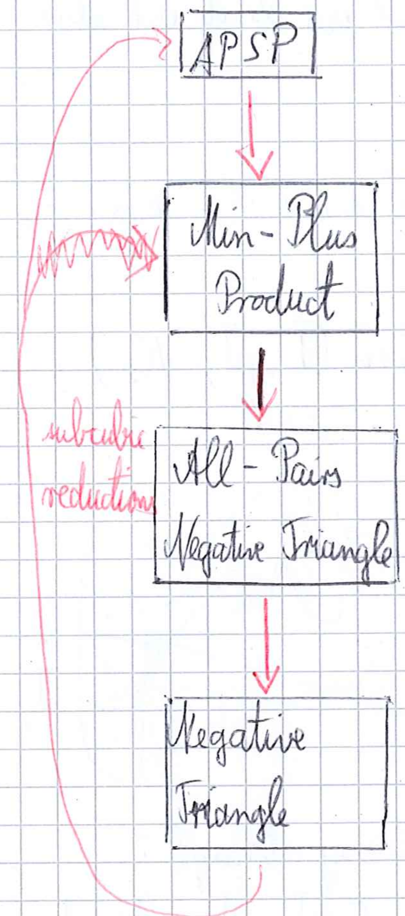
$\Rightarrow$ If one of A, B, C has a truly subcubic algorithm, then all of them have

Def: All-Pairs Shortest Paths (APSP) Problem
Given: Weighted (directed) graph G with edge weights $\in \{1, \dots, n^c\}$ for some constant c
Task: For every pair of vertices u,v ~~compute~~ compute (length of the) shortest path from u to v

Known Algorithms: $O(n^3)$ [Floyd-Warshall '62]
⋮
$O(n^3 / 2^{\Omega(\sqrt{\log n})})$ [Williams '14]

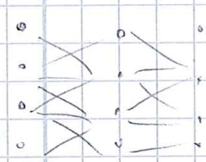
Conjecture: For any constant $\varepsilon > 0$ there is a constant $c > 0$ s.t. APSP admits no algorithm with running time $O(n^{3-\varepsilon})$.

Subcubic Equivalences



Given $n \times n$ matrices $A, B \in \{-n^c, \dots, n^c, \infty\}$
compute min-plus product C given by $C_{i,j} = \min_{1 \le k \le n}(A_{ik} + B_{kj})$

Solve APSP in "3-layered" graphs
$G = (V, E)$ s.t. $V = I \,\dot\cup\, J \,\dot\cup\, K$ and
$E \subseteq (I \times J) \cup (J \times K)$



Given graph with vertex set $V = I \,\dot\cup\, J \,\dot\cup\, K$
Decide for every $i \in I$, $j \in I$ whether there is a $k \in K$
s.t. $w(j,i) + w(i,k) + w(k,j) < 0$

Does given graph contain triangle of negative total weight, i.e. ~~edges~~ edges (i,j), (j,k), (k,i) s.t.
$w(i,j) + w(j,k) + w(k,i) < 0$

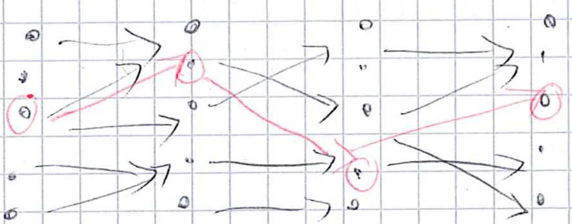
These 4 problems are in the same "subcubic APSP-equivalence class"
To show membership for a problem P, the easiest approach is often
to show Negative Triangle $\le P \le$ APSP

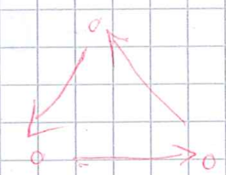
From Negative Triangle to APSP

Consider 4 copies of V: V_1, V_2, V_3, V_4 $\quad V_i = \{v^{(i)} : v \in V\}$

$E_i := \{(u^{(i)}, v^{(i+1)}) \in V_i \times V_{i+1} \mid (u,v) \in E\}$ $\quad i = 1,2,3$

· Solve APSP in the graph $G' = (V_1 \cup V_2 \cup V_3 \cup V_4, E_1 \cup E_2 \cup E_3)$





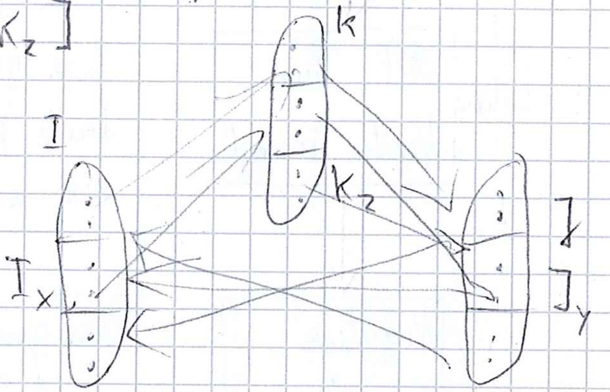
Observation: G contains a negative triangle x, y, z
$\iff$ The distance from $x^{(1)}$ to $x^{(3)}$ in G' is negative

Running Time: $T_{3LAPSP}(n) + O(n^2)$

From All-Pairs Negative Triangle to Negative Triangle

First, assume negative triangle algorithm does not just decide but also outputs a triangle.

- Initialize C as ~~an~~ $n \times n$ all-zeroes matrix
- Split I, J, K ~~into~~ evenly into $\frac{n}{s}$ parts of size s (for param. s):
  $I_1, \dots, I_{n/s}, J_1, \dots, J_{n/s}, K_1, \dots, K_{n/s}$
- For each of the $(n/s)^3$ triples of the form (I_x, J_y, K_z)
  Consider graph $G[I_x \cup J_y \cup K_z]$



While $G[I_x \cup J_y \cup K_z]$ contains a neg. triangle:

Find a neg triangle (i, j, k) in $G[I_x \cup J_y \cup K_z]$

Set $C[i, j] = 1$

Remove edge (i, j) from G

Observation:
- guaranteed termination (at most n^2 edges deleted from G)
- Correctness: if (i, j) is in neg. triangle, it will be found

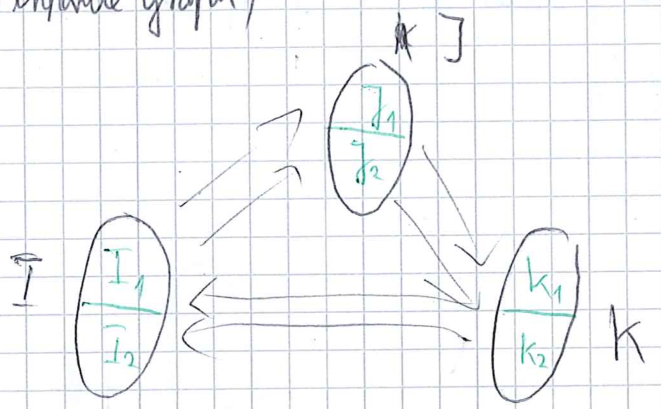
Running Time: $(\#\text{triples} + \#\text{triangles found}) \cdot T_{FindNT}(s)$

$$\leq \left(\left(\frac{n}{s}\right)^3 + n^2\right) \cdot T_{FindNT}(s)$$

Set $s = n^{1/3}$, assume $T_{FindNT}(s) = O(s^{3-\varepsilon})$

$$= O(n^2 \cdot n^{1-\varepsilon/3}) = O(n^{3-\varepsilon/3})$$

Finding neg. triangle, assuming decision algorithm:
(in a tripartite graph)



Partition (evenly) $I = I_1 \dot\cup I_2$, $J = J_1 \dot\cup J_2$, $K = K_1 \dot\cup K_2$

If G contains a neg. triangle, at least one of the 2^3 subgraphs $G[I_{(a)} \cup J_b \cup K_c]$ contains a neg. triangle

Decide for each such ~~neg~~ subgraph if it contains a neg. triangle
Recursively find a triangle in one subgraph

(Base case: ~~If~~ I, J, K have constant size: just check by brute force)

Running Time: $T_{FindNT}(n) \leq 2^3 \cdot T_{DecideNT}(n) + T_{FindNT}(\frac{n}{2})$

$$= O(T_{DecideNT}(n))$$