Subcubic Equivalences

Reductions so far:

<table>
<thead>
<tr>
<th>Problem P</th>
<th>reduction</th>
<th>Problem Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance I</td>
<td>T(n) time</td>
<td>instance J</td>
</tr>
</tbody>
</table>

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

Then \( T(n) \)-time algorithm for Q implies \( T(n) + S(n) \)-time algorithm for P.

(Or: If P has no \( T(n) + S(n) \)-time alg., then Q has no \( T(n) \)-time algorithm.)

More sophisticated reduction:

<table>
<thead>
<tr>
<th>Problem P</th>
<th>reduction</th>
<th>Problem Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance I</td>
<td>total time ( T(n) )</td>
<td>instance I_j</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instance I_k</td>
</tr>
</tbody>
</table>

instance \( I_j \) might depend on results for instances \( I_1, \ldots, I_k \).

Def.: A subcubic reduction from P to Q is an algorithm \( A \) for P
oracle access to Q s.t.

* For every instance \( x \) of P, \( A(x) \) solves the problem \( P \) on \( x \).
* Excluding oracle calls, \( A \) runs in time \( O(n^{3-\varepsilon}) \) (for some constant \( \varepsilon > 0 \))
on instances of size \( n \).
* For every constant \( \varepsilon > 0 \) there is a constant \( S > 0 \) such that for every
instance \( x \) of P of size \( n \) we have \( \sum_{i=1}^k n_i^{3-\varepsilon} \leq n^{3-\varepsilon+S} \), where \( n_i \) is the size of the instance in the \( i \)-th oracle call to Q in \( A(x) \).
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 $O(n^{3-\varepsilon})$-time algorithm for some $\varepsilon > 0$.

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 $A \leq C$
If: Homework

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

⇒ 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 $c \leq 1, \ldots, n^\varepsilon$
for some constant $\varepsilon$

Task: For every pair of vertices $u, v$, compute (length of the)
shortest path from $u$ to $v$

Known algorithms: $O(n^3)$ [Floyd-Warshall '62]

$O(n^{3/2 + \Omega(\log^2 n)})$ [Williams '14]

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

\[ \text{APSP} \downarrow \]

Min-Plus Product

\[ \text{All-Pairs} \]

Negative Triangle

Subcubic reduction

Given \( n \times n \) matrices \( A, B \in \mathbb{R}_{\geq 0} \), compute the min-plus product \( C \) given by \( C_{ij} = \min_{k=1}^{n} (A_{ik} + B_{kj}) \).

Solve APSP in \( 3 \)-layered graph, \( G = (V, E) \) with \( V = I \cup J \cup K \) and \( E \subseteq (I \times J) \cup (J \times K) \).

Given graph \( G \) with vertex set \( V = I \cup J \cup K \), decide for every \( (i, j) \in I \) whether there is a \( k \) such that \( w(i, j) + w(i, k) + w(k, j) < 0 \).

Does given graph contain a triangle of negative total weight, i.e., negative edges \( (i, j), (j, k), (k, i) \), such that \( w(i, j) + w(j, k) + w(k, i) < 0 \)?

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 \( \text{Negative Triangle} \leq P \leq \text{APSP} \).

From Negative Triangle to APSP

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

\[ E_i := \{ ((x^{(i)}, y^{(i)}), (u^{(i)}, v^{(i)})) \in V_i \times V_i : (u,v) \in E \} \quad i = 1, 2, 3 \]

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 \( x, y, z \)

\[ \Rightarrow \text{The distance from} \ x^{(1)} \ \text{to} \ x^{(3)} \ \text{in} \ G' \ \text{is negative} \]

Running Time: \( T_{31 \text{APSP}} (n) + O(n^3) \)
From All-Pairs Negative Triangle to Negative Triangle

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

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

While $G[I_x, J_y, K_z]$ contains a neg. triangle:
  - Find a neg. triangle $(i, j, k)$ in $G[I_x, J_y, K_z]$
  - 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: $(\# \text{ triplets} + \# \text{ triangles found}) \cdot T_{\text{FindNT}}(s)$

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

Yet $s = n^{\frac{1}{3}}$, assume $T_{\text{FindNT}}(s) = O\left(s^{2.25}\right)$

$$= O\left(n^2 \cdot n^{1 - \frac{2.25}{3}}\right) = O\left(n^3 - \frac{2.25}{3}\right)$$
Finding neg. triangle, assuming decision algorithm:
(in a tripartite graph)

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

If \( G \) contains a neg. triangle, at least one of the \( 2^3 \) subgraphs \( G[I_1, J_1, K_1] \) 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: \( \forall i, j, k \) have constant \( v \); just check by brute force)

Running Time:
\[
T_{\text{findVT}}(n) \leq 2^3 \cdot T_{\text{decideVT}}(n) + T_{\text{findVT}}(\frac{n}{2})
\]
\[
= O\left( T_{\text{decideVT}}(n) \right)
\]