April 3rd, 2014

On arXiv.org, the main repository of publically available theoretical computer science research, a paper was posted.

This paper disproved a widely-held conjecture about the algorithmic complexity of many problems...

Theorem: The 3SUM problem has an algorithm that runs in $\underbrace{O\left(n^2 \cdot \frac{(\log\log n)^{5/3}}{(\log n)^{2/3}}\right)}_{o(n^2)}$ time.

[Grønlund, Pettie, "Threesomes, degenerates, and love triangles", 2014]

This means that the lecture "Reductions" on March 13 is now out-of-date and contains results that are vacuously true.

3SUM-HARDNESS

real RAM

Problem (3SUM): Given an array of integers S, do there exist three integers $a, b, c \in S$ such that $a+b+c=0$?

$[-20, 9, 4, -8, 1, 15, -11, -4, 3]$ YES

$[-4, -9, 8, 11, -1, 7, -20]$ NO

Algorithm for 3SUM:

$S[j]-S[i]+S[k]-S[i]$

1. Sort S. $O(n \log n)$
2. For i from 1 to $S.size()$: $O(n)$

   Construct array R, where $R = S + S[i]$ $O(n)$

   Search R for two elements $R[j], R[k]$ such that $R[j]+R[k]=S[i]$ $O(n)$ using two-finger algorithm

$S = [-20, -11, -8, -4, 1, 3, 4, 9, 15]$

$R = S + S[9] = [-5, 4, 7, 11, 16, 18, 19, 24, 30]$ $R[2]+R[4]=15=S[9]$

~~Conjecture: Any algorithm for 3SUM takes $\Omega(n^2)$ time.~~

Evidence ~~is~~ that people have tried hard and failed plus other stuff

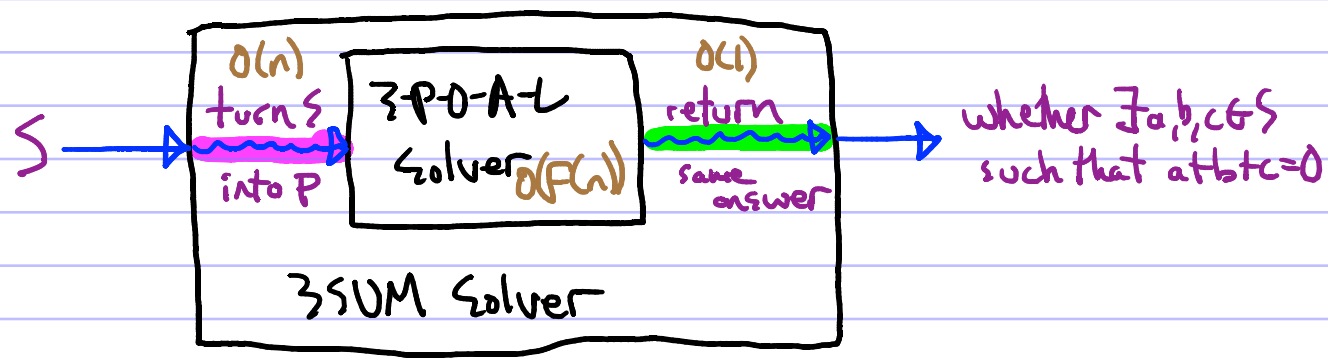
~~If I wanted to show that Problem B also takes $\Omega(n^2)$ time to solve using any algorithm, I can use a reduction from 3SUM to Problem B. Such a reduction implies that Problem B is 3SUM-hard.~~

give evidence of

3SUM-hard PROBLEMS

Problem (3-Points-on-a-line): Given a set of n points in the plane, are any 3 collinear?

Reduction:
- $O(n)$: Take integers $S=\{x_1,x_2,x_3,\dots,x_n\}$. Create point set $P=\{(x_1,x_1^3),(x_2,x_2^3),\dots,(x_n,x_n^3)\}$
- $O(f(n))$: Feed P to 3-P-O-A-L Solver
- $O(1)$: If 3-P-O-A-L says yes, then return yes. Else return no.



3SUM Solver works b/c $a+b+c=0$ iff $(a,a^3),(b,b^3),(c,c^3)$ collinear.

The reduction says "3-P-O-A-L is 3SUM-hard".

If any 3SUM Solver must take $\Omega(n^2)$ time, then since this solver takes $O(n+f(n)+1)$ time, $f(n)=\Omega(n^2)$.

3SUM-hardness $\approx$ Takes $\Omega(n^2)$ time if 3SUM takes $\Omega(n^2)$ time.

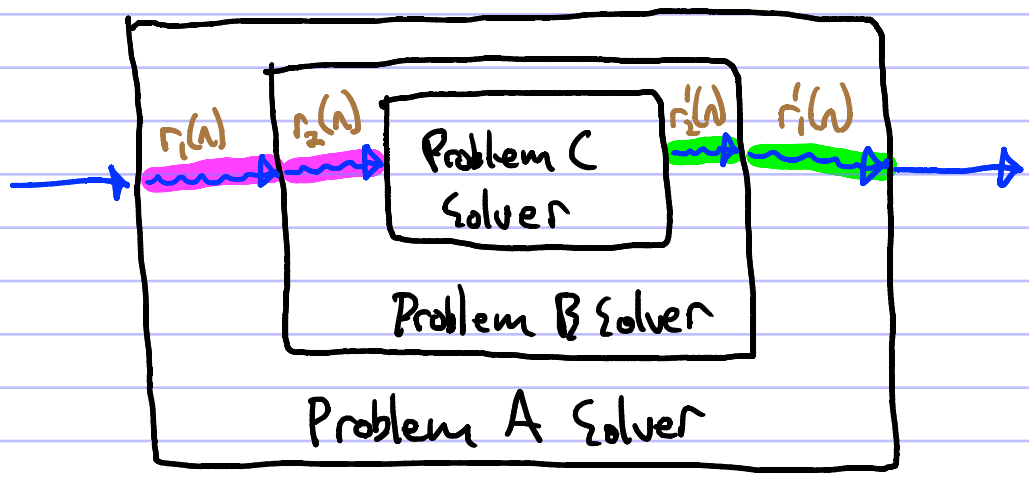
This reduction also implies that if 3-P-O-A-L can be solved in $O(f(n))$ time, then 3SUM can be solved in $O(n+f(n)+1)$ time.

A trivial $f(n)$ is $O(\binom{n}{2})=O(n^2)$, but we already had $O(n^2)$ 3SUM algo.

Problem (Minimum Area Triangle): given a set of n points in the plane, what is the smallest area triangle induced by 3 of the points?

Thm: M-A-T is 3SUM-hard. How to show this?

Suppose we nested reductions:

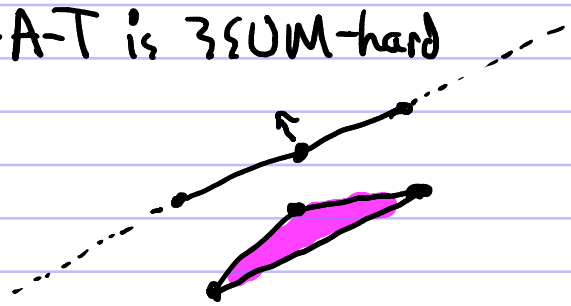


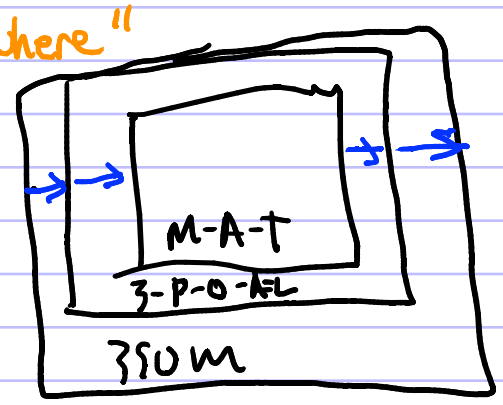
If Problem A takes $\Omega(n^2)$ time, then Problem C takes $\Omega(n^2)$ time provided $r_1(n)+r_2(n)+r_2'(n)+r_1'(n)$ is $o(n^2)$.

"The $\Omega(n^2)$ time has to be spent somewhere"

Claim: M-A-T is 3SUM-hard

Proof:





The New State of 3SUM

1. Can be solved in $o(n^2)$ time.
2. Unknown if solvable in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$.
3. New conjecture: $\Omega(n^{2-\varepsilon})$ time $\forall \varepsilon > 0$?

$o(n^2)$ time algorithm idea: solve in $O(n^{3/2}\sqrt{\log n})$ time in comparison-cost model, then port algorithm to real RAM using tricks to obtain a slight speed-up over naive port.

[Bremner et al 2012]
[Chan 2005]
[Chan 2010]

Linear decision tree model: Pay only for comparing values. Get to specify a linear function of the values and get sign of function out. Can compare $O(1)$ values at once.

3SUM Algorithm

Input: Array A of n numbers.

1. Sort A. Partition into $\frac{n}{g}$ groups $A_0, A_1, \ldots, A_{\frac{n}{g}}$ of size g. $O(n \log n)$

2. Let $D = \{a - a' : a, a' \in A_i\}$. Sort D. $O(n \log n + gn)$ [Fredman, 1976]

3. Let $A_{i,j} := \{a+b : a \in A_i, b \in A_j, i \le j\}$. Sort all $A_{i,j}$. $O(1)$ using Step 2 &

   $a+b < c+d$ iff $a-c < d-b$

   "Fredman's trick"

4. For k from 1 to n: $O(n)$ iterations

4a. Let $l = 0$, $h = \frac{n}{g}$. // $k \in A_h$

4b. while $l \le h$: $O(\frac{n}{g})$ iterations

4c. if $-A[k] \in A_{l,h}$: return "Yes". $O(\log(g^2))$ via BS of $A_{l,h}$

4d. if $\max(A_l) + \min(A_h) > -A(k)$: $--h$, else: $++l$ $O(1)$

5. return "No"

Total comparisons: $O(n \log n + \frac{n^2 \log g}{g})$

Let $g = \sqrt{n \log n}$: $O(n \log n + \frac{n^2 \log(\sqrt{n \log n})}{\sqrt{n \log n}}) = O(n^{3/2} \log n)$

How to port into $o(n^2)$ real RAM algorithm?

Don't do steps 2, 3. Put off 4c queries "$-A[k] \in A_{l,h}$", run in batch later.