CSE 190, Sp 2024: Fine-grained complexity and algorithms

Day 2: Problems equivalent to 3-sum

Russell Impagliazzo, rimpagliazzo@ucsd.edu

Barna Saha bsaha@ucsd.edu

A 1st : 3 sum equivalence

•*Anka Gajentaan, Mark H. Overmars:*
On a Class of O(n^2) Problems in Computational Geometry. *Comput. Geom. 5: 165-185 (1995)*

•*de Berg, M.; van Kreveld, M.; Overmars, M. H.; Schwarzkopf, O. (1997). Computational Geometry: Algorithms and Applications. Springer-Verlag.* 2nd ed., 2000; 3rd ed., 2008.[15][16]

GameMaker™

Basic problems we use: 3-sum

1. 3-Sum (generalization k-sum) : Given lists of n integers each, A, B, C, is there an a+b=c, $a \in A, b \in B, c \in C.$?

$Best\ algorithm\ about\ O(n^2/(\log n)^2)$ (Timothy Chan 18) ,

k-Sum: Given k lists is there a number in the last list that is a sum of one element from each of the previous lists?

Best algorithm: $O(n^{\lceil \frac{k}{2} \rceil}/\log^2 \mathrm{n})$. Conjecture : no algorithm in time $O(n^{\lceil \frac{k}{2} \rceil - \varepsilon})$.

How do we know when problems are equivalent?

Reductions: ``Any way of giving an improved algorithm for problem X could be used to give an improved algorithm for problem Y''

Fine grained reductions

We use essentially the same notion of reduction as in traditional algorithms and complexity theory. However, we look at quantitative details more closely.

Reduction from X to Y:

Map instance x to instance y via y=F(x).

x and y need to be equivalent.

For any AY that solves Y, we can define AX= AY(F(x)) that solves X.

$$T_{AX}(|x|) = \mathrm{T_F}(|\mathrm{x}|) + \mathrm{T_{AY}}(|\mathrm{y}|)$$

Traditional: As long as F is polytime, and AY is polynomial time, this is polytime.

Fine grained reductions

.Easiest case

Map instance x to instance y via y=F(x).

x and y need to be equivalent.

For any AY that solves Y, we can define AX= AY(F(x)) that solves X.

$$T_{AX}(|x|) = \mathrm{T_F}(|\mathrm{x}|) + \mathrm{T_{AY}}(|\mathrm{y}|)$$

Exact complexity preserving: $|y| = O(|x|\ polylog(|x|)), T_F\ is\ o\big(CX(n)\big),$

where CX is the ``conjectured time'' to solve X

3-sum hardness (simple case)

Reduce 3-sum to new problem Y. (Known hard problem to unknown problem)

x=(A,B,C) and y=F(x) need to be equivalent: y is a True instance of F if and only if there is a summing triple in A,B,C

For any AY that solves Y, we can define A3sum= AY(F(x)) that solves X.

$$T_{A3sum}(n) = \mathrm{T_F}(n) + \mathrm{T_{AY}}(|\mathrm{y}|)$$

Need: |y| = O(n) (or close) and $T_F(|x|) \in O(n^c),\ c < 2.$

3-sum hardness (simple case)

For any AY that solves Y, we can define A3sum= AY(F(x)) that solves X.

$$T_{A3sum}(n) = \mathrm{T_F}(n) + \mathrm{T_{AY}}(|\mathrm{y}|)$$

Need: |y| = O(n) (or close) and $T_F(|x|) \in O(n^c),\ c < 2$.

Then if there were an algorithm for Y in time O(n^d), d<2,

we get T_{A3sum}(n) ∈ $O(n^{\max(c,d)})$ which would be better than the conjectured $O(n^2)$. Usually, c will be close to 1.

Warm-up example

This gives a simple example of such a reduction that is also useful as a stepping stone to other problems.

3-zerosum is the problem where we are given a single list of n integers, S, and want to decide whether there are three elements a, b, c $\in S$ so that a+b+c=0. This is in flavor very close to the 3-Sum problem as we defined it earlier, and is sometimes just called 3-sum.

We'll show the two problems are equivalent.

The reduction:

We are given the three sets A, B, and C, and want to create a single set S. We'll add one element to S for each element of the three lists, but modified in a small way for each list.

For each $a \in A$, we add $a' = 10a + 1$ to S.

For each $b \in B, we\ add\ b' = 10b + 3\ to\ S$

For each $c \in C$, we add c'= -10 c-4 to S

Why these values? I want to make sure the only way we can add up to zero is to have one element in each set, A B C, so I want to make sure 3 elements of A cannot add up to zero, 3 elements of B, or 2 elements of A and one element of B, and so on.

Equivalence

If a+b=c is a solution to the 3-sum problem we started with.

$a' + b' + c' = 10a + 1 + 10b + 3 \ - 10\, c \ - 4 = 10(a + b - c) + 0 = 0$

So if there is a positive answer for the 3-sum problem, the answer for 3zerosum for S is True.

Equivalence

If x+y+z=0 is a solution to the 3zerosum problem for S:

Then x+y+z =0 mod 10. If none of them are c' , each mod 10 is either 1 or 3, so the sum mod 10 is between 3 and 9, not zero. If all three are c', the sum mod 10 is -12= 8 which is not zero. If two are c' s, their sum mod 10 is even, whereas the remaining one mod 10 is odd, so the overall sum is odd, not zero mod 10.

So exactly one of the three, say z, is c'= -10 c-4 for some c in C.

If both the others are from A, their sum mod 10 is 2, and 2-4 is not zero mod 10. If both are from B, their sum mod 10 is 6, and again the overall sum isn't zero. So one, say x is a'=10a+1 and the other y is b'=10b+3.

Then 10a+1+10b+3 -10c-4 = 0, or 10(a+b-c) =0, or a+b=c is a positive

solution to the original 3-sum problem.

Complexity

Size of S is 3n

Time to compute S is O(n).

So any algorithm for 3zerosum in time $O\left(n^d\right), 1 \leq d < 2$, would give an $O\left(n^d\right)$ time algorithm for 3-Sum.

Reducing 3zerosum to 3sum

This direction is very simple.

Let A=S, B=S, C= -s, $s \in S$

x+y+z =0 if and only if x+y=-z.

A more interesting problem

Co-linearity : Given a set P of points (x,y) with integer valued coordinates, are there three colinear points (points on the same line)?

Given (x1,y1) and (x2, y2), the line going through both has the formula:

x=my + B, for m the slope and B the intercept. m will be $m = \frac{y2-y1}{x2-x1}$.

There is only one line with slope m going through a point, so if a third point (x3,y3) has the same m with (x1,y1), it will be on the same line.

So the three points are colinear if and only if :

$$\frac{y2-y1}{x2-x1} = \frac{y3-y1}{x3-x1}$$

A more interesting reduction

Co-linearity : Given a set P of points (x,y) with integer valued coordinates, are there three colinear points (points on the same line)?

$$\frac{y2-y1}{x2-x1} = \frac{y3-y1}{x3-x1}$$

We can use this to give an $O(n^2 \log n)$ time algorithm for Colinearity:

For each point, compute the slope of the line to all other points.

If any have the same slope, we have found 3 colinear points (Use sorting or hashing to check.)

If we never do, there are no colinear points.

A tricky reduction

We'll reduce from 3ZeroSum to Colinearity. Since we know that 3Sum is equivalent to 3ZeroSum, this shows Colinearity is 3Sum hard.

Let S be the set of n integers that are inputs to 3ZeroSum.

We'll define the set of points P by including the point

(s, s^3) in P for each $s \in S$

(I am really not sure how they thought to do this.)

Equivalence

Claim : $(x, x^3), (y, y^3), (z, z^3)$ are collinear if and only if

x+y+z=0

Proof: The slope between $(x, x^3)\ and\ (y, y^3) is \frac{y^3-x^3}{y-x} = y^2 + xy + x^2$

Similarly, that between $(x, x^3)\ and\ (z, z^3) is\ z^2 + xz + x^2$

These are the same if and only if $z^2 + xz + x^2 = y^2 + yz + x^2$,

Or equivalently, $z^2 + xz = y^2 + xy$

or $z^2 - y^2 = x(y - z)$

or $(z - y)(z + y) = -x\,(z - y), i.e., z + y = -x$,

$z + y + x = 0$

Conclusion

Since |P|=|S|, and we can compute P from S in linear time,

any time $O(n^d)$ $algorithm$ $1 \leq< 2$ for $Colinear$ gives us a similar algorithm for 3ZeroSum, and hence 3Sum.

So if the 3Sum conjecture is true, there is no algorithm for Colinearity testing that is sub-quadratic (i.e., exponent less than 2).

Class problem

Consider the problem: 3-Intersection.

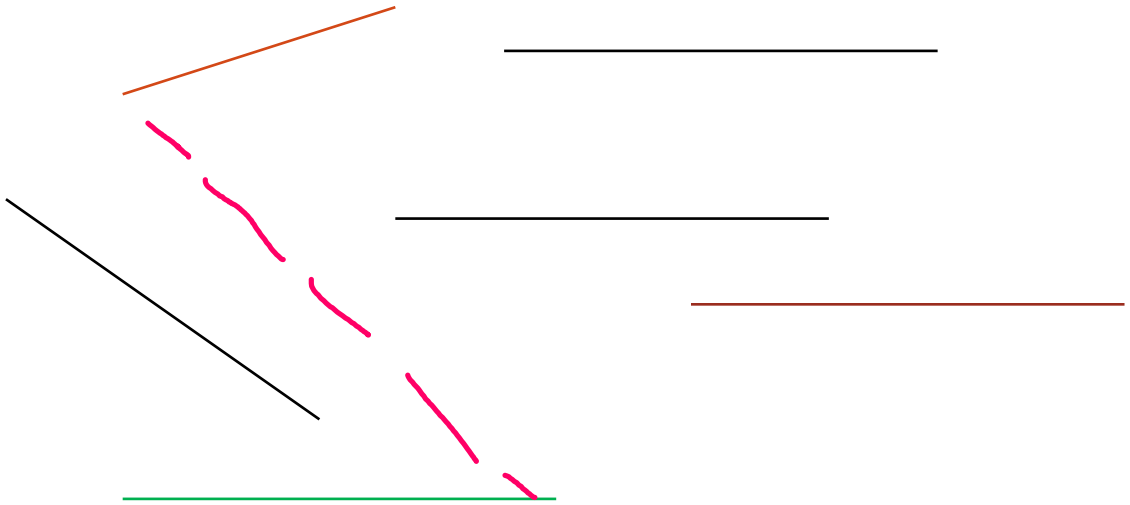
Given n lines by slope and intercept, are there 3 of the lines that pass through a single point?

Show that 3-Intersection is equivalent to Colinearity

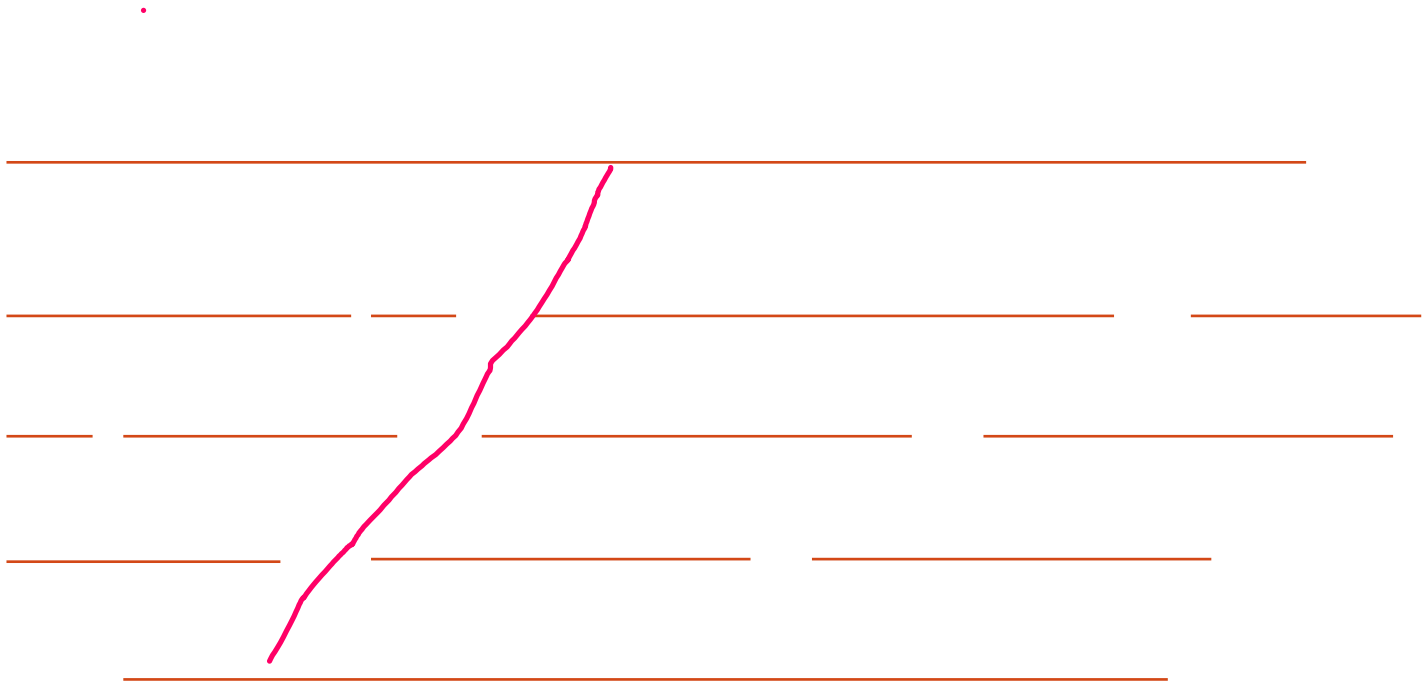
Visibility:

Instance: A set of n line segments called obstacles and two specified line segments L1 and L2

Is there a point p1 on L1 that can see a point p2 on L2? In other words, the line segment between p1 and p2 does not intersect any of the obstacles.
Special case that is still hard: All the line segments are horizontal



Reduction from 3 sum

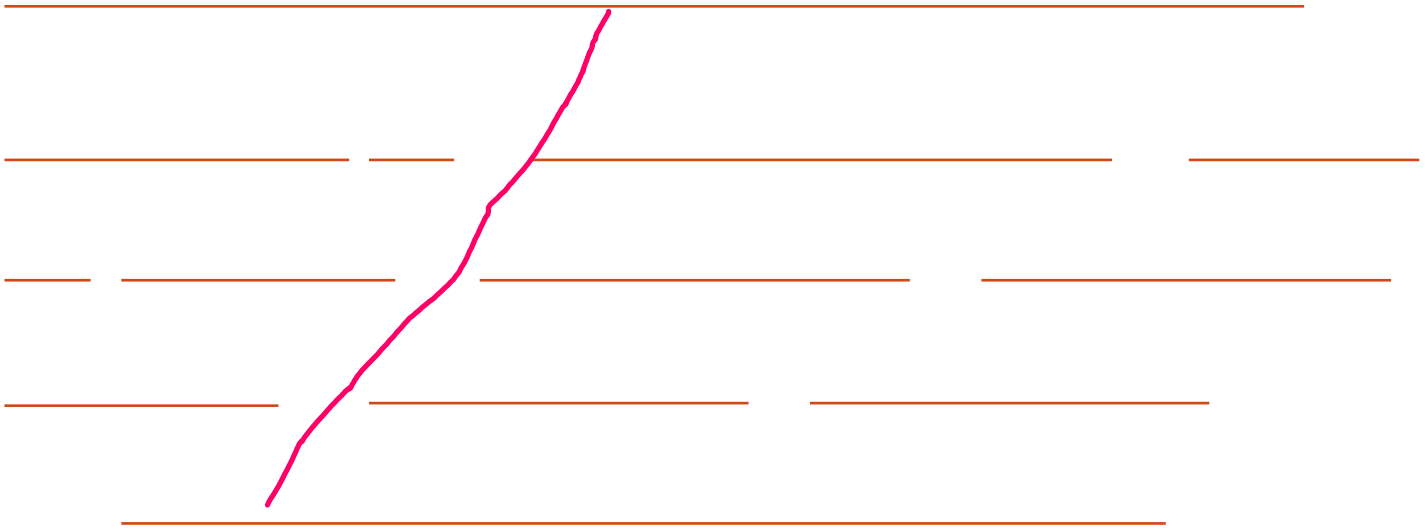


Reduction from 3 sum

Create 5 rows of lines, distance 1 between each pair of rows.

Top and bottom are solid and are L1 and L2 .

Second row has small gaps at each $a \in A$, fourth row has gaps at each $b \in$B. Middle row has gaps at each c/2 , c $\in C$

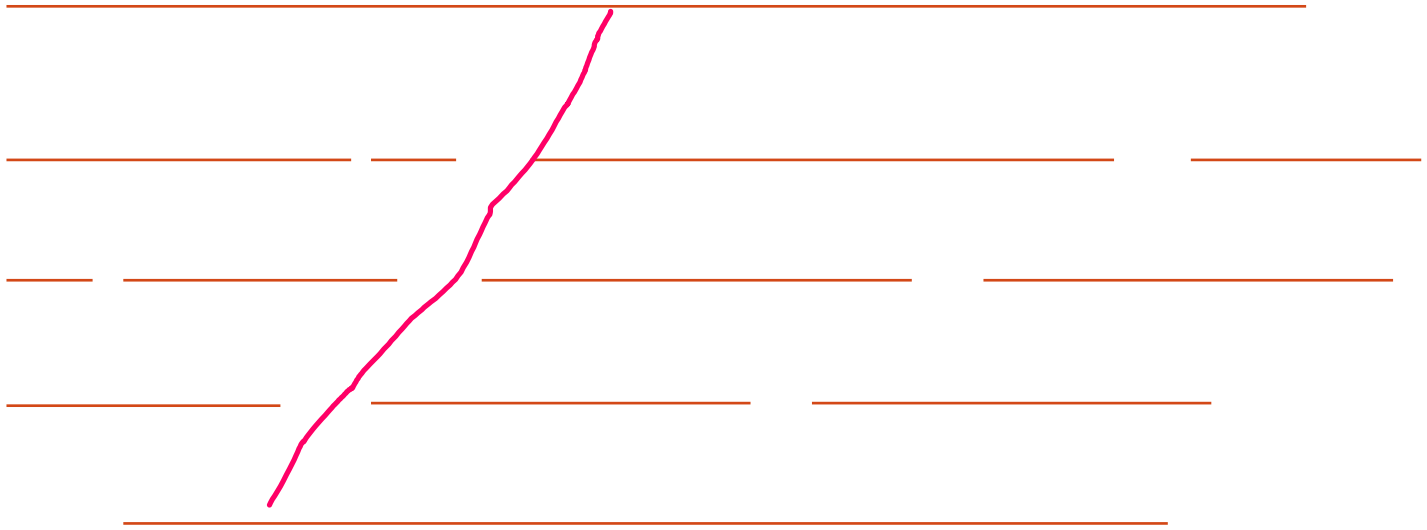


Equivalence to 3Sum instance

Second row has small gaps at each $a \in A$, fourth row has gaps at each $b \in B$. Middle row has gaps at each c/2 , $c \in C$.

To be visible, 3 gaps must form line: (a,1), (c/2, 2), (b,3).

Slopes to middle point match when c/2-a = b-c/2, or c = a+ b.



Consequences

Compute the new segments in O(n log n) time (need to sort).

3 n +2 segments

So any $O(n^d)$ algorithm for visibility with 1 < d < 2 would give an

algorithm for 3-Sum with the same time, contradicting 3-Sum conjecture.

Question: What is the best algorithm for visibility? They don't seem to mention it in this paper.

This is a small subset

Of all the problems they show 3-sum hard.

If the 3-sum conjecture is true, then while we can solve these problems in polynomial time, there are limitations to how fast we can solve them.

Next class

Improving algorithms for NP-complete problems

The k-SAT problem

The Paturi-Pudlak-Zane k-SAT algorithm

The Schoning k-SAT algorithm (maybe)