CSE520: Computational Geometry
Lecture 20
3SUM and Reductions

Antoine Vigneron

Ulsan National Institute of Science and Technology

June 15, 2020

Introduction

- In Lecture 5 and 6, we showed how to prove lower bounds on computational geometry problems by topological methods.
- In this lecture, we present a different way of proving a hardness result:
- Proving that a problem is harder than a well-known problem, for which no subquadratic algorithm is known.

- Reference: Gajentaan and Overmars paper.

Introduction

- So far, we have seen two ways of proving hardness results:
- Giving a reduction from the sorting problem.
- The topological lower bound method of Ben-Or.

- These two methods usually yield $\Omega(n \log n)$ lower bounds.
- Can we do better?
- At this point, no better technique is known.

- But we can try to prove that the problem we are trying to solve is harder than a well-known problem, for which no $O(n \log n)$ algorithm has been found.
- This is done through a reduction.
- We will now give examples.

3SUM

Problem (3SUM)

Given a set S of n integers, is there a triple $a, b, c \in S$ such that $a + b + c = 0$?

- 3SUM has an $\Omega(n \log n)$ lower bound in the ACT model.
- It can be solved in $O(n^2)$ time.
- The best known algorithm is slightly faster: $n^2(\log\log n)^{O(1)}/\log^2 n$. (T. Chan, 2018.)
- Despite a lot of effort, no better bound is known.
- Hence, if we can argue that a problem is *harder* than 3SUM, it means that an algorithm running in time $O(n^{1.99})$ is currently out of reach.

Algorithms for 3SUM

- How much time does it take by brute force? $\Theta(n^3)$.
- A faster algorithm:

Pseudocode

```
1: procedure 3SUM(S)
2:     record S in a dictionary data structure D
3:     for a ∈ S do
4:         for b ∈ S do
5:             if −a − b ∈ D then
6:                 return YES
7:     return NO
```

- Using a sorted array for $\mathcal{D}$, line 5 takes $O(\log n)$ time, so the running time is $O(n^2 \log n)$.
- Can we do better?

Algorithms for 3SUM

- We first consider a related problem:

Problem (Sorted sequence disjointness)

Given two sorted sequences of numbers $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$, return NO if there exist i, j such that $u_i = v_j$, and return YES otherwise.

- How fast can we solve it?
- $O(n)$ time.
- Idea: use the merging procedure of mergesort.
- See next slide.

Algorithms for 3SUM

Linear-time algorithm for SORTED SEQUENCE DISJOINTNESS

procedure DISJOINT(u, v)
    $i \leftarrow 1$, $j \leftarrow 1$
    while $i \leqslant n$ and $j \leqslant n$ **do**
        if $u_i = v_j$ **then**
            return NO
        if $u_i < v_j$ **then**
            $i \leftarrow i + 1$
        else
            $j \leftarrow j + 1$
    return YES

- How can we solve 3SUM using this procedure?

Algorithms for 3SUM

Quadratic algorithm for 3SUM

procedure $\mathrm{3SUM}(S)$
  sort S
  for $a \in S$ **do**
    if DISJOINT$(a + S, \hat{S})$=NO **then**
      return YES
  return NO

- $a + S$ is the sequence $a + s_1, a + s_2, \ldots, a + s_n$.
- $\hat{S}$ is the sequence $-s_n, \ldots, -s_2, -s_1$.

- This algorithm runs in $O(n^2)$ time.

Lower Bound for 3SUM

Theorem

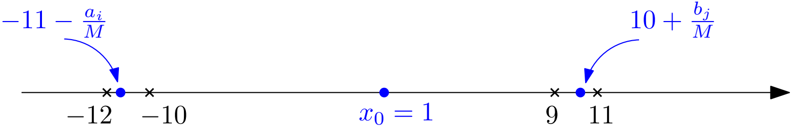
Any Algebraic Computation Tree (ACT) that solves 3SUM has depth $\Omega(n \log n)$.

- We prove it by a reduction from SET DISJOINTNESS.
- Let $(a_1, \ldots, a_n, b_1, \ldots, b_n)$ be an instance of SET DISJOINTNESS.
- So it is a negative instance iff there exist i and j such that $a_i = b_j$.
- We first compute in $O(n)$ time

$$M = \max_i \left(\max(|a_i|, |b_i|)\right).$$

- We construct the following instance of 3SUM:

Lower Bound for 3SUM



$$x_0 = 1$$

$$x_i = -11 - \frac{a_i}{M} \qquad \text{for } i = 1, \ldots, n$$

$$x_{j+n} = 10 + \frac{b_j}{M} \qquad \text{for } j = 1, \ldots, n$$

Lower Bound for 3SUM

- Three of these numbers can only sum to 0 if they are of the form x_0, x_i and x_{j+n}, and if $a_i = b_j$.
- So it is a positive instance of 3SUM iff the original instance of SET DISJOINTNESS is negative.

- Thus, in order to solve our instance of SET DISJOINTNESS, we can construct the instance of 3SUM described above in $O(n)$ time, and then solve this instance of 3SUM.
- It shows that 3SUM is harder than SET DISJOINTNESS.
- As SET DISJOINTNESS has a lower bound $\Omega(n \log n)$ in the ACT model, the same is true for 3SUM.

- No better lower bound is known for 3SUM, even though the best known algorithms are only slightly subquadratic.

3SUM-Hardness

Definition (3SUM-hard)

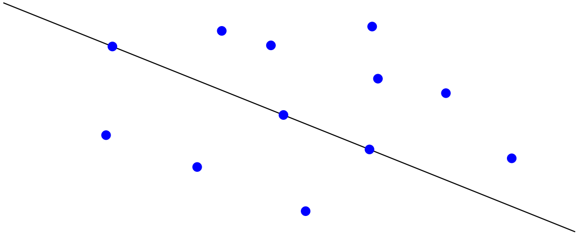
A problem P is 3SUM-hard if any instance of 3SUM can be solved by solving $O(1)$ instances of P of size $O(n)$, and spending an additional $O(n^c)$ time with $c < 2$.

- Intuitively, it means that P is at least as hard as 3SUM, and thus at this point, we don't know how to solve it in time $O(n^{1.99})$.
- We give now give examples of 3SUM-hard problems.

Degeneracy Testing

Problem (2D degeneracy testing)

Given a set S of n points in the plane, the 2D degeneracy-testing problem is to decide whether there are three collinear points in S.

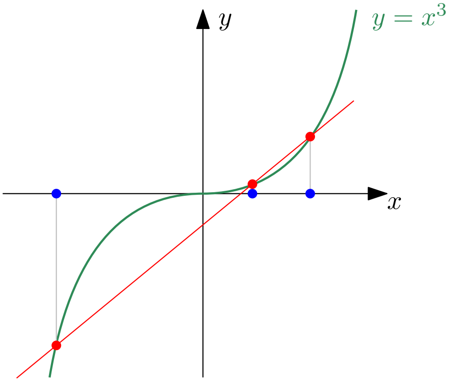


Theorem

2D degeneracy testing is 3SUM-hard.

- We now prove this theorem.

Degeneracy Testing



Lemma

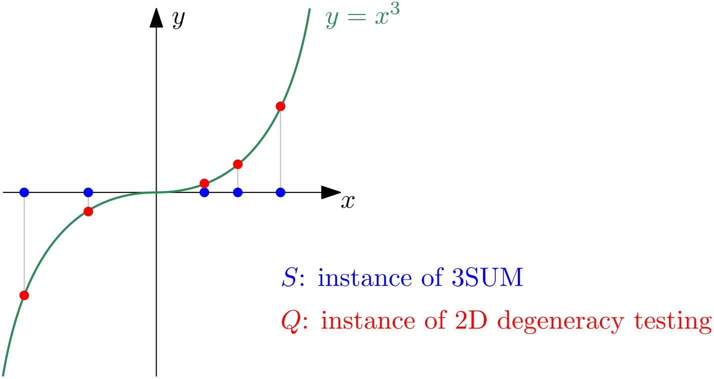
For any distinct real numbers a, b and c, the points (a, a^3), (b, b^3), and (c, c^3) are collinear if and only if $a + b + c = 0$.

Degeneracy Testing: Lemma Proof

- If (a, a^3), (b, b^3) and (c, c^3) are collinear then they are on a line with equation $y = \lambda x + \mu$.
- So a, b and c are roots of the equation $x^3 = \lambda x + \mu$.
- Therefore $x^3 - \lambda x - \mu = (x - a)(x - b)(x - c)$.
- The coefficient of x^2 on the LHS is 0, and it is $-a - b - c$ on the RHS, so $a + b + c = 0$.

- Conversely, suppose $a + b + c = 0$.
- Then $(x - a)(x - b)(x - c) = x^3 - \lambda x - \mu$ for some λ, μ.
- So a, b and c are the roots of the equation $x^3 - \lambda x - \mu = 0$.
- Thus the points (a, a^3), (b, b^3) and (c, c^3) are on the line $y = \lambda x + \mu$.

Degeneracy Testing: Reduction



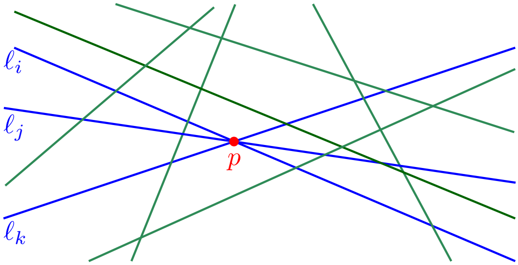
- Let S be an instance of 3SUM.
- We construct the following instance of 2D degeneracy testing:

$$Q = \{(x, x^3) \mid x \in S\}.$$

Degeneracy Testing

- By the lemma above, Q is a positive instance of 2D degeneracy testing iff S is a positive instance of 3SUM.
- So in order to solve our instance S of 3SUM, we can construct in $O(n)$ time the instance Q of 2D degeneracy testing, and solve Q.
- This shows that 2D degeneracy testing is 3SUM hard.
- Conclusion: An $O(n^{1.99})$-time algorithm for 2D degeneracy testing is currently out of reach.

Point on 3 Lines



Problem (Concurrent lines)

Are there 3 concurrent lines in a set L of n input lines?

Point on 3 Lines

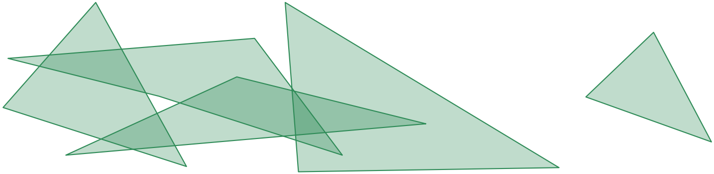
Theorem

The concurrent lines problem is 3SUM-hard.

Proof.

Three lines $\ell_i, \ell_j, \ell_k \in L$ intersect at a common point p if and only if the dual line p^* contains the dual points ℓ_i^*, ℓ_j^* and ℓ_k^*. So the concurrent lines problem is exactly the dual of the degeneracy testing problem. Hence it is 3SUM-hard as well. $\square$

Triangle Area



Problem (Triangle area)

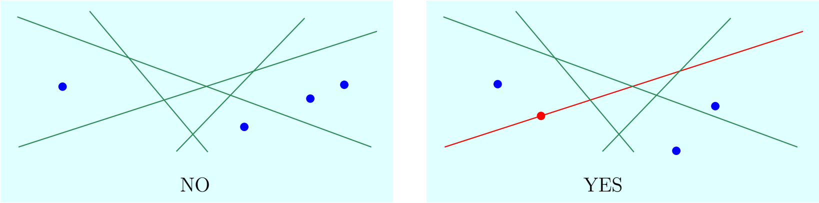
Determine the area of the union of n triangles.

- The proof of this result is not covered in this course.
- It can can be found in the paper by Gajentaan and Overmars, as well as several other examples of 3SUM-hard problems.

Hopcroft's Point-Line Incidence Problem

Problem

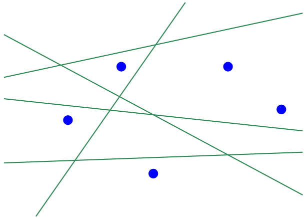
Given a set L of n lines in the plane, and a set S of n points, decides whether there is a point $p \in S$ and a line $\ell \in L$ such that $p \in \ell$.

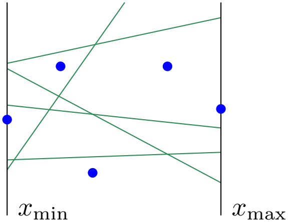


- Brute force: $O(n^2)$.
- But this problem is *not* known to be 3SUM-hard.
- Reason: We will give an $O(n^{1.5} \log n)$ algorithm
- Best known algorithm: Slightly more than $O(n^{4/3})$.

Hopcroft's Point-Line Incidence Problem

- We consider a variation of the problem where there are m lines and n points, and $m \leqslant n$. How fast can we solve this version of Hopcroft's problem?





- Find the minimum and maximum x-coordinates of the points.
- Clip the lines at $x = x_{\text{min}}$ and $x = x_{\text{max}}$.
- We are now looking for incidence (=intersection) between a set of points and a set of line segments.

Hopcroft's Point-Line Incidence Problem

- This is a special case of line segment intersection reporting where n of the segments are single points. (See Lecture 3.)
- So we can solve it by plane sweep in time $O((m+n+k)\log(m+n))$ where k is the number of intersection points.
- There are $O(m^2)$ intersections between lines and $\leqslant n$ intersections between points and lines, so $k = O(n+m^2)$
- As $m \leqslant n$, the running time is $O((n+m^2)\log n)$.

- How to obtain a running time $O(n^{1.5}\log n)$ when $m = n$?
- Observation: When $m = \sqrt{n}$, the algorithm runs in $O(n \log n)$.
- So we group the n lines in $\sqrt{n}$ groups of $\sqrt{n}$ lines, and we solve the problem separately for each group of $\sqrt{n}$ lines with n points.
- If one point-line incidence is found, this is a positive instance of the original problem. It takes time $\sqrt{n} \cdot O(n \log n) = O(n^{1.5}\log n)$.

Concluding Remarks

- Hopcroft's problem is also used to prove hardness results in the same way as we used 3SUM.
- For instance, computing the diameter of a set of n points in $\mathbb{R}^7$ is harder than Hopcroft's problem.
- As the best known algorithm for Hopcroft's problem runs in time roughly $O(n^{4/3})$, we cannot hope to do better for the diameter problem in dimension 7.
- This is different from the closest pair problem, for which there are $O(n \log n)$-time algorithms in any fixed dimension.