1 Introduction

2 Algebraic computation trees

3 Topological lower bound

4 Discussion

5 3SUM-hardness
Outline

Hardness results for some geometric problems.

Examples:

- Line segment intersection detection.
- Diameter of a point-set.
- Degeneracy testing.

Two techniques:

- Topological lower bound.
- Reduction from 3-SUM.

References:

- Textbook by Preparata and Shamos.
- Dave Mount’s lecture notes, Lecture 26.
- Ben-Or’s paper.
- Gajentaan and Overmars paper.
Introduction

In the algorithms course (CS 260), you saw that:

**Theorem (Lower bound for sorting)**

Any comparison-based sorting algorithm makes $\Omega(n \log n)$ comparisons in the worst case.

Proof (sketch):

- Model the algorithm as a binary decision tree.
- Each internal node is a comparison, branching to its two children.
- There are $n!$ possible outcomes, i.e. permutation of the input.
- Hence there are at least $n!$ leaves.
- So the tree has height $\Omega(\log(n!)) = \Omega(n \log n)$. 

Antoine Vigneron (KAUST)  
CS 372 Lecture 4  
September 20, 2012
Introduction

In Lecture 2, we showed that it yields the same $\Omega(n \log n)$ lower bound for computing a convex hull.

- Proof: After mapping a set of numbers to a parabola, the numbers appear in sorted order along the convex hull.
- So the argument holds because a convex hull algorithm outputs a point sequence.

This argument does not work if the output has constant size.

Examples:
- Intersection detection. (Output: a boolean.)
- Diameter. (Output: a real number.)

We will give a different technique that yields an $\Omega(n \log n)$ lower bound for these two problems, and others.
Example (1)

Given \((x_1, x_2, x_3, x_4) \in \mathbb{R}^4\) such that \(x_1 \leq x_2\) and \(x_3 \leq x_4\), this Algebraic Computation Tree (ACT) decides whether \([x_1, x_2] \cap [x_3, x_4] \neq \emptyset\).
Algebraic Computation Trees

Node $v_1$: $f_{v_1} := x_3 - x_1$

Node $v_2$: $f_{v_2} := x_6 - x_2$

Node $v_3$: $f_{v_3} := f_{v_1} \times f_{v_2}$

Node $v_4$: $f_{v_4} := x_5 - x_1$

Node $v_5$: $f_{v_5} := x_4 - x_2$

Node $v_6$: $f_{v_6} := f_{v_4} \times f_{v_5}$

Node $v_7$: $f_{v_7} := f_{v_3} - f_{v_6}$

Node $v_8$: $f_{v_7} > 0$?

Example (CCW predicate)
Given $(x_1, x_2, x_3, x_4, x_5, x_6) \in \mathbb{R}^6$, this ACT decides whether the triangle $((x_1, x_2), (x_3, x_4), (x_5, x_6))$ is counterclockwise.

YES  NO
**Algebraic Computation Trees**

Formal definition from Ben-Or’s [paper](#):

**Definition (Algebraic computation tree)**

An algebraic computation tree with input \((x_1, \ldots, x_n) \in \mathbb{R}^n\) is a binary tree \(T\) with a function that assigns:

- to any vertex \(v\) with exactly one child an operational instruction of the form
  
  \[ f_v := f^1_v \circ f^2_v \text{ or } f_v := c \circ f^1_v \text{ or } f_v := \sqrt{f^1_v} \]

  where \(f^i_v = f_{v_i}\) for an ancestor \(v_i\) of \(v\), or \(f^i_v \in \{x_1, \ldots, x_n\}\),
  \(\circ \in \{+, -, \times, /\}\), and \(c \in \mathbb{R}\) is a constant.

- to any vertex \(v\) with two children (branching vertex) a test instruction of the form
  
  \[ f^1_v > 0 \text{ or } f^1_v \geq 0 \text{ or } f^1_v = 0. \]

  where \(f^1_v\) is \(f_{v_1}\) for an ancestor \(v_1\) of \(v\), or \(f^1_v \in \{x_1, \ldots, x_n\}\).

- to any leaf an output YES or NO.
Algebraic Computation Trees

Informally:

- Given an input $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the program traverses a path $P(x)$ from the root to a leaf of $T$.
- Along the path, it applies operations $+, -, /, \times, \sqrt{\cdot}$ to input numbers $x_i$ or intermediate results obtained at previous nodes along $P(x)$.
- It may also branch using a test $>, \geq, =$.
- At the leaf, it outputs YES or NO.

Definition

Let $T$ be an algebraic computation tree with input $x \in \mathbb{R}^n$. Let $W \subset \mathbb{R}^n$ be the set of points $x \in \mathbb{R}^n$ such that $T$ outputs YES. We say that $T$ decides $W$. 
Topological Lower Bound

**Theorem (Ben-Or, 1983)**

Any algebraic computation tree that decides a set $W \subset \mathbb{R}^n$ has height $\Omega(\log(\#W) - n)$, where $\#W$ is the number of connected components of $W$.

Interpretation:

- The height of an ACT is its worst-case running time.
- A program can often be *unfolded* onto an ACT.
  - Then its worst-case running time is at least the height of the ACT.
- But some operations cannot be simulated by an ACT in $O(1)$ time.

Examples:

- The floor function.
- Bitwise operations on integers (AND, OR, XOR).
- Random number generation.
Element Distinctness

Problem (Element Distinctness)

Determine whether the elements of a list of numbers are distinct. That is, given \((x_1, \ldots, x_n) \in \mathbb{R}^n\), determine whether \(x_i \neq x_j\) for all \(i \neq j\).

- Let \(W^+\) denote the set of positive instances of Element Distinctness:

  \[
  W^+ = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \forall i \neq j, x_i \neq x_j\}.
  \]

- Note that an instance of Element Distinctness is modeled as a \textit{single point} in \(\mathbb{R}^n\). Hence \(W^+\) is a subset of \(\mathbb{R}^n\).
- How many connected components are there in \(W^+\)?
Element Distinctness

Lemma

The set $W^+$ of positive instances of Element Distinctness has exactly $n!$ connected components.

Proof.

Done in class.
In the algebraic computation tree model, the complexity of element distinctness is $\Theta(n \log n)$.

Proof.

By Ben-Or’s theorem, since $W^+$ has $n!$ connected components, any ACT solving the element distinctness problem has height $\Omega(\log(n!) - n)$, which is $\Omega(n \log n)$.

Conversely, there is an ACT with depth $O(n \log n)$ that solves element distinctness: First it sorts the input, then checks consecutive values. For instance, mergesort can be unfolded into a $\Theta(n \log n)$-depth ACT.
Let $W^-$ denote the set of negative instances of Element Distinctness:

$$W^- = \mathbb{R}^n \setminus W^+.$$

How many connected components are there in $W^-$?
**Application: Line Segment Intersection Detection**

**Theorem**

Any ACT that solves the line segment intersection detection problem has height $\Omega(n \log n)$. Hence, the complexity of line segment intersection detection is $\Theta(n \log n)$ in the ACT model.

**Proof.**

Suppose $T$ is an ACT that solves the line segment intersection detection problem. Let $(x_1, \ldots, x_n) \in \mathbb{R}^n$. We construct an ACT $T'$ by plugging $(x_1, 0, x_1, 1, x_2, 0, x_2, 1, \ldots, x_n, 0, x_n, 1)$ to the input of $T$. That is, $T'$ detects intersection between the segments $[(x_i, 0), (x_i, 1)]$. So $T'$ solves the element distinctness problem. Thus $\text{height}(T') = \Omega(n \log n)$. But by construction, $\text{height}(T') = \text{height}(T) + 4n$. Therefore $\text{height}(T) = \Omega(n \log n)$. 

Antoine Vigneron (KAUST)  
CS 372 Lecture 4  
September 20, 2012  
15 / 28
Set Disjointness

Problem (Set Disjointness)

Given two sequences \((a_1, \ldots, a_n) \subseteq \mathbb{R}\) and \((b_1, \ldots, b_m) \subseteq \mathbb{R}\) such that \(m \leq n\), determine whether there exists a pair \(i, j\) such that \(a_i = b_j\).

Theorem

Any ACT that solves the set disjointness problem has height \(\Omega(n \log n)\).

Proof?

- Done in class.
- Note: The numbers \(a_1, \ldots, a_n\) are not necessarily distinct, so a direct reduction to Element Distinctness fails.
**Problem (Diameter of a Point-Set)**

The diameter $\text{diam}(P)$ of a set $P = \{p_1, \ldots, p_n\}$ of $n$ points is the maximum distance between any two points:

$$\text{diam}(P) = \max_{i,j} d(p_i, p_j).$$
Maximum Gap

Problem (maximum gap)

Given a set of n (unsorted) numbers, the maximum gap problem is to find the largest gap between two consecutive numbers in sorted order.

Example:

- INPUT: 0.5, 2, 5, 4.5, -1
- OUTPUT: 2.5
Optimization Algorithms

- The ACT model deals with *decision* problems: problems with a YES-NO answer.
- We will abuse notation and use ACTs for *optimization* problems:
  - The 2D-diameter problem.
  - Maximum gap.
- In fact, our lower bound argument will be for the associated decision problems: For some $\delta \in \mathbb{R}$,
  - Is the diameter at least $\delta$?
  - Is the maximum gap at most $\delta$?
- An optimization problem is at least as hard as the corresponding decision problem, because once you have the optimal value, you can compare it with $\delta$ in $O(1)$ time.
- So a lower bound on the decision problem implies a lower bound on the optimization problem.
Given an instance $A, B$ of set disjointness, we map $A$ to the right side and $B$ to the left side of the unit circle, respectively.
The diameter is 2 iff two points $p_i, p'_j$ are diametrically opposed, that is $a_i = b_j$. 
Theorem

In the ACT model, the complexity of the 2D-diameter problem is $\Omega(n \log n)$.

Proof.

An instance of set disjointness is mapped to the unit circle as follows:

- Each $a_i$ is mapped to the point $p_i = (x_i, y_i)$ such that $y_i = a_i x_i$, $x_i > 0$, and $x_i^2 + y_i^2 = 1$.
- Each $b_j$ is mapped to the point $p'_j = (x'_j, y'_j)$ such that $y_j = b_j x_j$, $x_j < 0$, and $x_j^2 + y_j^2 = 1$.

Then the diameter of $\{p_1, \ldots, p_n, p'_1, \ldots, p'_n\}$ is 2 iff $p_i = -p'_j$ for some $i, j$, which means that $a_i = b_j$ and hence $A \cap B \neq \emptyset$.

Remark: it is crucial in this argument that the coordinates of $p_i$ and $p'_j$ can be computed in constant time by an ACT.
Maximum Gap

Theorem

In the ACT model, the maximum gap problem has complexity $\Theta(n \log n)$.

Proof?

Problem: there is a simple $O(n)$-time algorithm.

- This linear-time algorithm uses the floor-function, hence it cannot be translated into an ACT.
Discussion

It would seem that the ACT model is not powerful enough, as the lower bound for maxgap can be broken using the floor function.

- However, it is not reasonable to allow the use of the floor function: It has been proved that the floor function together with $+, -, \times, /$ allows to solve NP-hard problems in polynomial time. (Schönhage, *On the power of random access machines*, ICALP 79.)

In fact, the ACT model is very powerful:

- It allows *exact* computation in constant time per arithmetic operation.
- Even with high degree polynomials, or square roots.
Discussion

Ben-Or’s theorem often gives $\Omega(n \log n)$ lower bounds for geometric problems.

- Optimal for several fundamental problems. (Intersection detection, 2D-diameter...)
- But it often gives the same bound in cases where no near-linear time algorithm is known.
  - Example: 4D diameter, Hopcroft’s problem.
  - Does not always give $\Omega(n \log n)$: it gives $\Omega(n^2)$ for knapsack. But this problem is NP-hard.

Another approach: show that your problem is harder than some well-known problem, for which no near-linear time algorithm is known.

- 3-SUM. Best algorithm runs in $O(n^2)$. (See following slides.)
- Hopcroft’s problem. Best algorithm runs in roughly $O(n^{4/3})$. 
3SUM

Problem (3SUM)

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

- 3SUM can be solved in $O(n^2)$ time. (How?)
- There is an $\Omega(n \log n)$ lower bound in the ACT model. (How?)
- Despite a lot of effort, no better bound is known with usual models of computation.
- Hence, if we can argue that a problem is harder than 3SUM, it means that an $o(n^2)$-time algorithm is currently out of reach.
**Definition (3SUM-hard)**

A problem is 3SUM-hard if an \( o(n^2) \)-time algorithm for this problem implies an \( o(n^2) \)-time algorithm for 3SUM.

We give an example in the next slide. Other examples can be found in the [paper](#) by Gajentaan and Overmars, for instance.
3SUM-Hardness

Problem (2D degeneracy-testing)

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

Theorem

2D degeneracy-testing is 3SUM-hard.

Proof.

Based on the following fact:
For any three distinct numbers $a, b, c$, $a + b + c = 0$ iff the points $(a, a^3), (b, b^3), (c, c^3)$ are collinear.