Lower Bounds for Geometric Problems

Subhash Suri

November 30, 2015

1 Model of Computation

- To analyze an algorithm's performance, or to reason about the intrinsic complexity of a computational problem, one needs a *formal model* of computation.
- The model specifies the *primitive operations* that may be executed and their *costs*.
- Examples include Turing Machine, or Random Access Model. The primary difference for our purposes between these models is how the manipulation of individual numbers is treated. TM uses bits, and so to add two k bits numbers has $O(k)$ cost—namely, the cost grows in proportion to the operand length.
- RAM allows two numbers to be manipulated in constant time, in line with the hardware of digital computers, with the implicit assumption that each number fits in a hardware *word*.

2 Real RAM

- Geometric computation introduces another level of complexity: even when the input numbers are small integers, their geometric calculations may entail more complex numbers, including irrationals. Length of the diagonal of a unit square, for instance.
- So, the Real RAM permits the abstraction that dispenses with round-off errors in the approximate representation of real numbers, but we should make sure our software libraries provide mechanisms to deal with these overflows and roundoffs, when needed.
- In simple terms, the Real RAM allows each memory location to *hold a single real number, and allows the following primitives at O(1) cost*:
  1. Arithmetic $(+, -, \star, /)$

2. Comparison ($<, \leq, =, \neq, \geq, >$)
3. Indirect addressing with integer addresses
4. kth root, trig functions, analytic functions (exp, log etc).

- This model fairly closely captures the primitives of all modern programming languages.

3 Transformations and Reductions

- The most common technique for proving lower bounds is reduction. Reduce problem A to B.

  1. Input of problem A is converted to a suitable input for B.
  2. Solve problem B.
  3. Transform the output into a correct solution to problem A.

- If the transformation steps 1 and 3 take time $\tau(N)$ on input size N, then we say that A is $\tau(N)$-reducible to B.
- [**Reduction Theorem.**] If problem A is known to require $T(N)$ time to solve, and A is $\tau(N)$-reducible to B, then B requires at least $T(N) - O(\tau(N))$ time.
- In other words, hardness of A proves hardness of B.
- Similarly, if B can be solved in $T(N)$, then A can be solved in $T(N) + O(\tau(N))$.
- In the precious reduction, we only transformed in the direction from A to B. If the $\tau(N)$-reduction works in both directions, then A and B are called *equivalent.*.
- But to get started, we first need a lower bound on A. How does one prove that some problem A must require $T(N)$ time no matter what algorithm is used?

4 ADT—Algebraic Decision (Computation) Tree

- While Real RAM is the right model for designing algorithms, it is not terribly well-suited for proving lower bounds.
- Instead, a slightly different *but computationally equivalent* model is more convenient. Essentially, the ADT more closely mimics the way we think about "programs" or algorithms—as an interleaving of compute and branch instructions.

- Specifically, assume the input involves a set of real variables $x_1, \ldots, x_n$. Then, a ADT is a program with statements $L_1, L_2, \ldots, L_p$ of the form:

  1. Compute a function $f(x_1, \ldots, x_n)$. If $f > 0$, go to statement L_i; else go to L_j.
  2. Halt and output YES
  3. Halt and output NO

- The function f is an *algebraic function* (a polynomial of some degree).

- We assume that the program has been "unrolled" so it has no loops. Therefore, it has the structure of a *tree* T, where each internal node v is associated with a polynomial function evaluation and comparison:

$$f_v(x_1, \ldots, x_n) > 0 \ ?$$

- Note that we can collapse all the computation that occurs between two comparison nodes into the next comparison node, without loss of generality.

- The ADT is a dth order tree if d is the largest degree used. 1st order tree is also called Linear Decision Trees—only linear functions are evaluated.

- (If you recall the comparison-based sorting lower bound, it only used a linear decision tree.)

- We also assume, wlog, that the tree is a binary tree—comparisons are binary. Any k-way tree can be simulated by a sequence of $k - 1$ binary comparisons, if needed.

- It is easy to see that any Real RAM program corresponds to an ADT—each execution is a root-to-leaf path. Therefore, the worst-case complexity of the program is (at least) proportional to the *longest path in the tree.*

- **Remark:** We are not using/allowing any *randomization* in our programs. However, similar but slightly more complicated arguments apply to randomized versions as well.

5 Using ADT for Lower Bounds

- The central idea is simple, but abstract. (Such abstraction is necessary to be able to subsume all possible algorithms within the model constraints.)

- The idea originated with Steele-Yao and Ben-Or (1982–83).

- We will only consider Decision problems, because any optimization problem is at least as hard as its decision counterpart.
- Let $x_1, \ldots, x_n$ be the variables of the decision problem.
- Each instance of the problem (involving n reals) can therefore be viewed as a *point in the n-dimensional Euclidean space* R^n.
- Some instances of the problem evaluate to YES, others to NO (otherwise the problem is trivial).
- Let W be the subspace of R^n that contains all the YES instances. That is, the algorithm outputs YES if and only if the input $(x_1, \ldots, x_n) \in W$.
- Let $\#W$ denote the number of *disjoint connected components of* W.

  Figure.
- Suppose T is the ADT corresponding to an algorithm that solves this problem. Each execution of T traverses a unique path $v_1, \ldots, v_l$, where v_1 is the root, and v_l the leaf node.
- Each node v_j of this path is associated with a function $f_{v_j}(x_1, \ldots, x_n)$ so that $(x_1, \ldots, x_n)$ satisfies

$$f_{v_j} = 0, \text{ or } f_{v_j} > 0, \text{ or } f_{v_j} \geq 0 \quad \text{for all nodes of the path}$$

6 Argument for the Linear Functions

- The intuitive part of the proof technique is best understood for the linear decision model. This is the (easier) framework introduced in Dobkin-Lipton (1979).
- Let T be the binary linear decision tree embodying the algorithm A that solves the membership in W.
- Associated with each leaf of T is a region of R^n, and each leaf is either "accepting" or "rejecting." (This is the final node in the tree, so the algorithm must output the answer.)
- Suppose
  1. $W_1, \ldots, W_p$ are the (connected) components of W,
  2. $l_1, \ldots, l_r$ the set of leaves of T, and

3. $D_j \subset R^n$ is the domain associated with leaf l_j.

- By definition of the algorithm's correctness, l_j is accepting if and only if $D_j \subset W$.
- **The lower bound is shown by proving that $r \geq \#W$.** That is, T must have as many leaves as connected components of W.
- Define $Y(W_i)$ to be the minimum index $j \in \{1, 2, \ldots r\}$ such that $D_j \cap W_i \neq \emptyset$.
- That is, $Y(W_i)$ is the smallest index leaf whose domain intersects with the ith component of W.
- We show that two different regions W_i, W_j must have different Y indices.
  1. Suppose not, and assume (for the sake of contradiction) that $Y(W_i) = Y(W_j) = h$.
  2. The algorithm A solves the membership of an instance (point) $q \in W_i$, leaf l_h must be accepting.
  3. By definition of Y, we must have $W_i \cap D_h$ non-empty. Similarly, for W_j.
  4. Pick two points $q' \in W_i \cap D_h$ and $q'' \in W_j \cap D_h$.
  5. Since T is a *linear* decision tree, the region D_h is the intersection of halfspaces in R^n, and therefore a *convex set*.
  6. Therefore, any convex combination of q' and q'' must also lie in D_h. In particular, the entire line segment $q'q''$ lies in D_h.
  7. But since q' and q'' lie in disjoint components W_i and W_j, there is at least one point q''' on this segment such that $q''' \notin W$.
  8. A contradiction: the segment cannot lie in $D_h \subset W$ and still have a point outside W.
  9. Thus, T has at least as many leaves as number of components in W.
- Thus, the height of T is at least $\log_2 \#W$.

7 Extension to Algebraic Functions (ADT)

- The main difficulty with the previous argument is that if the functions f are *non-linear*, the domain associated with a leaf is no longer *convex* or (most importantly) *connected*.
- The joint result of degree d polynomial inequalities can be quite complex and highly disconnected. How many pieces?

- To make progress on this question requires ideas from algebraic geometry, and builds on important results proved by Milnor and Thom (1960s).
- The intuitive idea is this: suppose we take a number of polynomial functions, each of degree at most d, in m-dimensional space: $g_i(x_1, x_2, \ldots, x_m) = 0$. Then the number of connected components in the solution set of these equations is upper bounded as

$$d(2d-1)^{m-1}$$

- In order to import this ideas to our lower bound, we need to make sure we can handle *polynomial inequalities* and not just equations, and that was done by Steele-Yap and Ben-Or.
- What SY and Ben-Or show is this: Suppose h is the depth of the ADT tree T, corresponding to our algorithm A, operating on a problem with n variables, using degree d polynomial functions. Then, T has at most 2^h leaves, and each leaf accounts for at most $d(2d-1)^{n+h-}$ components of W.
- Therefore, following the same line of logic as before, we get

$$\#W \quad \leq \quad 2^h d(2d-1)^{n+h-1}$$

- In simplified form, it gives the lower bound on the height of T (running time of A):

$$h \quad \geq \quad \frac{\log_2 \#W}{1 + \log_2(2d-1)}$$

8 Element Distinctness

- We now show a concrete problem and a lower bound on its complexity under the ADT model.
- Given a set of n numbers $x_1, \ldots, x_n$, decide if they are all distinct. That is, $x_i \neq x_j$, whenever $i \neq j$.
- This should be "easier" than sorting. Is it?
- Let $W \subset R^n$ denote the set of all *YES* instances of the problem, namely, instances where elements are all unique.
- How many connected components does W have?

- **Claim:** $\#W = n!$.
- **Proof.** Recall that $n!$ is the number of distinct permutations of $\{1, 2, \ldots, n\}$.
    1. Each instance $\{x_1, x_2, \ldots, x_n\} \in W$ can be identified with the unique permutation of its numbers.
    2. We claim that each connected component contains only points identified with the same permutation. If not, then let p, p' be two instances with difference permutations, but within the same connected component.
    3. Without loss of generality, suppose that permutations p, p' differ in ordering of elements i and j. In other words, $x_i < x_j$ in p but $x_i > x_j$ in p'.
    4. Since p, p' are in the same component, there is a "path" connecting them, and each point on this path is also a valid YES instance of the problem.
    5. Therefore, there is a sequence of valid instances that starts at p (where $x_i < x_j$) but ends at p' (where $x_i > x_j$).
    6. But in order for the order to switch, at least two elements must become equal at some intermediate point.
    7. But having two equal items means the instance is not a valid YES instance, and thus not in W. Contradiction!
- Using our ADT Theorem, therefore any algebraic decision tree algorithm for Element Distinctness of n numbers must take $\Omega(n \log n)$ time.
- The lower bound assumes that the number of *reals*. What if numbers are rationals or integers?
- The ADT argument doesn't work. However, even for integer numbers, the $\Omega(n \log n)$ lower bound holds, by an extension proved by Lubiw and Racz (1991).

9 Geometric Problem Lower Bounds

- The Element Distinctness Problem turns out to be key to proving similar lower bounds on many other problems.
- **Set Disjointness.** Given two sets of numbers $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$, decide if $a_i = b_j$ for some i, j.
    - Element Distinctness is a special case, with a and b sequences being the same.

- **Maximum Gap.** Given a set of n (unsorted) numbers $x_1, x_2, \ldots, x_n$, what is the maximum gap between two consecutive numbers (in sorted order)?
  - Element Distinctness is a special case: the gap is non-zero precisely when elements are unique.
- **Diameter of 2D Set.** Given a set of n points in $2D$ plane $p_1, p_2, \ldots, p_n$, find the maximum distance between any two points.
  - First, how difficult is this problem in $1D$?
  - In 2D, we reduce the Diameter problem to the Set Disjointness.
  - Let $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$ be the input.
  - For each a_i, produce a point in the 2D plane where the line $y = a_i x$ intersects the unit circle on the right (positive x) side.
  - Specifically, each a_i maps to the point $p_i = (x_i, y_i)$ such that $y_i = a_i x_i$, $x_i > 0$, and $x_i^2 + y_i^2 = 1$.
  - For each b_i, produce a point in the 2D plane where the line $y = b_i x$ intersects the unit circle on the let (negative x) side.
  - Diameter of this collection of $2n$ points is 2 if and only if the sets are NOT disjoint; otherwise the diameter is strictly less than 2.
  - The entire transformation (input to diameter and back) takes $O(n)$ time.
  - So, diameter in $2D$ requires $\Omega(n \log n)$ time.

10 N^2-Hardness

- Unfortunately, for most geometric problems ADT at best yields an $\Omega(n \log n)$ lower bound.
- Using very different techniques and model, Chazelle, Fredman etc. have shown lower bounds for range searching, but that's an entirely different topic.
- Within computational geometry, there are many problems where we have not been able to beat the *quadratic* $O(n^2)$ algorithmic barrier, and it seems unlikely that it's even possible. But no lower bound technique is known.
- As partial progress, we have been able to show an *equivalence* class of many problems that are mutually $\Omega(n^2)$-Hard, meaning if you can devise a sub-quadratic algorithm for any one of them, we can solve all of them in sub-quadratic time bound.

11 3-Sum Problem

- Given a set S of n *integers*, is there a triple $a, b, c \in S$ such that $a + b + c = 0$?
- We can solve the problem in $O(n^2)$ time. (How?)
- The ACT model gives only an $\Omega(n \log n)$ lower bound.
- In spite of significant effort, no sub-quadratic time algorithm is known, under the standard model of computing.
- Imitating the NP-completeness model of problem equivalence classes, the 3-sum problem can be used to show n^2-hardness of other problems.
- "On a class of $O(n^2)$ problems in computational geometry," by Gajentaan and Overmars, CGTA 1995.
- *A problem is* $3SUM$*-Hard if an* $o(n^2)$ *time algorithm for the problem implies an* $o(n^2)$ *time algorithm for* $3SUM$.

12 Degeneracy Testing

- Given a set S of n points in the plane, are three of them collinear?
- We have often conveniently assumed that the points are in *non-degenerate* position. How complex is to check that condition?
- **Theorem.** $2D$ degeneracy testing is $3SUM$-hard.
- **Proof.** Three numbers a, b, c sum to 0 if and only if (a, a^3), (b, b^3), (c, c^3) are collinear.
- Suppose the 3 points lie on a line $y = \mu x + \gamma$. Then, for the first two points, we can infer that:

$$a^3 - b^3 = \mu(a - b) \quad \text{which implies} \quad \mu = a^2 + ab + b^2$$

- Similarly, for the 2nd and 3d point, we get

$$b^3 - c^3 = \mu(b - c) \quad \text{which implies} \quad \mu = b^2 + bc + c^2$$

- Thus, $a^2 + ab + b^2 = b^2 + bc + c^2$, which gives $a + c = -b$, or $a + b + c = 0$.

13 Other $3SUM$-hard Problems

- Using similar ideas, one can show that all of the following problems are $3SUM$-hard.
- Given a set of n lines in the plane, are there three that pass through the same point?
- Given a set of (non-intersecting, axis-parallel) line segments, is there a line that separates them into two non-empty subsets?
- Given a set of (infinite) strips in the plane, do they fully cover a given rectangle?
- Given a set of triangles in the plane, compute their measure.
- Given a set of horizontal triangles in space, can a particular triangle be seen from a particular viewpoint?
- See ??? for many more.