# Lecture 1. Samples of possibility and impossibility results in algorithm designing

The course is about the ultimate efficiency that algorithms and communication protocols can achieve in various settings. In the first lecture, we’ll see a couple of cleverly designed algorithms/protocols, and also prove several impossibility results to show limits of algorithms in various computation settings.

## 1. Communication complexity.

### 1.1 Setup

In information theory, we are usually concerned with transmitting a message through a noisy channel. In *communication complexity*, we consider the problem of computing a function with input variables distributed to two (or more) parties. Since each party only holds part of the input, they need to communicate in order to compute the function value. The communication complexity measures the minimum communication needed.

The most common communication models include

* *two-way*: two parties, Alice and Bob, holding inputs x and y, respectively, talk to each other back and forth.
* *one-way*: two parties, Alice and Bob, holding inputs x and y, respectively, but only Alice sends a message to Bob.
* *Simultaneous Message Passing (SMP)*: Three parties: Alice, Bob and Referee. Alice and Bob hold inputs x and y, respectively, and they each send one message to Referee, who then outputs an answer.
* *multiparty*: k parties, each holding some input variables and they communicate in a certain way.

Some example functions are:

* **Equality (EQn)**: To decide whether x = y, where x,y{0,1}n.
* **Inner** **Product (IPn)**: To compute x,y = x1y1 + … + xnyn mod 2, where x,y{0,1}n.
* **Disjointness (Disjn)**: To decide whether i s.t. xi = yi = 1 for x,y{0,1}n.

The computation modes include

* *deterministic*: all messages are deterministically specified in the protocol. Namely, each message of any party depends only on her input and the previous messages that she saw.
* *randomized*: parties can use randomness to decide the messages they send.
* *quantum*: the parties have quantum computers and they send quantum messages. (We will not study this mode, or in general, any quantum computing topic, in this course.)

In the two-way and one-way model, we say that a protocol computes the function if on all possible input instances, at the end of the protocol, both parties know the function value on the input. (In the SMP model, the Referee is required to know the function value.) For randomized protocol, we sometimes allow a small error probability. The *cost* of a protocol is measured by the total communication bits. The communication complexity of a function f is the minimum cost of any protocol that computes the function. We ignore the issue of local computation cost by assuming that all parties are computationally all-powerful. Though all the computation factors are gone and only *communication* is focused on, it turns out to be very helpful for communication complexity to be exploited to prove impossibility results for other *computational* complexity. (More on this later.)

### 1.2 An efficient deterministic protocol

Let’s consider the basic case of two parties. Note that if x and y are both at most n-bit long, then at the very least Alice can send her input to Bob, who then can compute f(x,y) since he has both inputs. Finally he tells Alice the answer by 1 bit communication. Thus the communication complexity of any 2n-bit function is at most n+1.

What makes communication complexity interesting on its own is that some functions enjoy much more efficient protocols.

**Example 1.1. Max.** Alice and Bob hold x,y{0,1}n , respectively. Interpret x as a subset of [n] = {1,2,…,n}; the subset contains the positions i s.t. xi = 1. Similarly for y. The task is to compute Max(x,y), the maximal number in x∪y. If Alice sends her entire input, it takes n communication bits. But she can clearly do something better: She merely needs to send the maximal number in her set x, which only takes log(n) bits. Bob then compares this value with his maximal number in y, and sends the larger one back to Alice as the answer. Altogether the protocol needs only 2log(n) bits.

This protocol seems trivial, but some others need more thoughts.

**Example 1.2**. **Median**. Alice and Bob again hold subsets x,y{0,1}n , respectively. Now they wish to compute the median of the multi-set x∪y. Recall that the median of k numbers is defined as the k/2th smallest number.

An efficient protocol is as follows. At any point, they maintain an interval [l,u] which guarantees to contain the median. Initially l = 1 and u = n. They halve the interval in each round, so it takes only log(n) rounds to finish. How do they halve the interval? Actually very simple: Alice sends to Bob the number of her elements that are at least m = (l+u)/2, and the number of elements that are less than m. This information is enough for Bob to know whether the median is above m or not. (Verify this!) Bob uses 1 bit to indicate the answer, and then the protocol goes to the next round.

Since each round has only O(log(n)) bits, and there are only log(n) rounds, the total communication cost is O(log2(n)), much more efficient than the trivial protocol of cost n+1.

### 1.3 Hardness for deterministic and rescue by randomness

We've seen examples of efficient protocols. On the other hand, for some problems, there isn't any protocol using communication less than n bits.

First, let’s observe that a deterministic communication protocol partitions the communication matrix into monochromatic rectangles. Here for a function f(x,y), the *communication matrix* Mf is defined by [f(x,y)](x,y), namely the rows are indexed by x and the columns are indexed by y, and the (x,y)-entry is f(x,y). A *rectangle* is a pair of subsets (S,T) where S is a set of rows and T is a set of columns. A rectangle (S,T) is *monochromatic* if Mf restricted on it is a submatrix containing only one value. (Namely, all entries in the submatrix are of the same value.) For any value b in range(f), a *b-rectangle* is a monochromatic rectangle in which all entries of Mf are of value b. By considering how the protocol goes round-by-round, we can observe the following fact.

**Fact 1.1**. *A c-bit deterministic communication protocol for f partitions the communication matrix Mf into 2c monochromatic rectangles*.

**Example 1.3**. **Equality**.

The above fact enables us to show that there is no efficient protocol for solving the Equality function EQn. Indeed, the communication matrix for EQn is IN, the identity matrix of size N=2n. Notice that for this matrix, all 1-rectangles are of size 11. So no matter how one partitions the matrix into monochromatic rectangles, there are always N 1-rectangles. Considering that there are also 0-rectangles, we have 2c > N = 2n, so c > n. We have thus proved the following theorem.

**Theorem 1.2**. Any deterministic protocol computing EQn needs n+1 bits of communication.

Despite the above negative result, next we’ll see that the Equality function does enjoy a very efficient (and one-way!) protocol if we are allowed to use some randomness in the protocol. Later in the course, we will see that whether the randomness is shared by the two parties or not doesn’t affect the communication complexity much. So let’s assume that Alice and Bob have shared randomness; we call such protocols *public-coin*.

**Theorem 1.3**. There is a one-way public-coin randomized protocol computing EQn with only O(1) bits of communication. (On any input (x,y), the protocol outputs the correct f(x,y) with probability 99%.)

Let’s first design a protocol with 50% 1-sided error probability, and later boost the success probability. The protocol is as follows.

* **Alice**:
	+ compute x,r = x1r1 + … + xnrn mod 2, where r is a public random string of length n,
	+ send the 1-bit result to Bob.
* **Bob**:
	+ compute y,r = y1r1 + … + ynrn mod 2, where r is the same random string that Alice used.
	+ Compare x,r and y,rOutput “x=y” if x,r = y,r and “x≠y” if x,r≠y,r

Note that the communication cost of the protocol is only 1 bit. Let’s see what the protocol does. To compare x and y using little communication, Alice tries to give a very short summary of her input x, and send only this summary to Bob, who computes the same summary of his input y. The summary is so short that it’s only 1 bit, so it contains very very little information of the input string. So … could this possibly work?

Let’s analyze it case by case. If x = y, then of course x,r = y,r, and the protocol is correct with certainty. If x ≠ y, we claim that x,r ≠ y,r with probability exactly 1/2! (Namely, if x≠y, then with half probability, the 1-bit summary can catch this distinction.) Actually, when x≠y, they differ at at least one position. Say it’s position i. then

x,r - y,rx-y,rx1-y1)r1 + … + xn-yn)rn mod 2

= xi-yi)ri + ∑j≠i xj-yj)rj mod 2

= ri + ∑j≠i xj-yj)rj mod 2.

Now notice that ri takes value 0 and 1 each with half probability, so regardless of the value of the second item, the summation is 1 always with probability half. Thus with probability half the protocol detects that x ≠ y.

To make the error probability smaller, one can simply repeat the above protocol k times, dropping the error probability to 1/2k.

### 1.4 Using communication complexity to show computational hardness: a taste

Apart from being a very natural and interesting subject on its own, communication complexity is also widely used to prove limitation results for numerous computational models, including data structures, circuit complexity, streaming algorithms, decision tree complexity, VLSI, algorithmic game theory, optimization, pseudo-randomness… Here we just give one such example, with more coming up later in the course.

**Example 1.4**. **Lower bound on time-space tradeoff on TM for explicit decision problems**.

First let’s recall the concept of [multi-tape](http://en.wikipedia.org/wiki/Multi-tape_Turing_machine#Multi-tape_Turing_machines) [Turing machine](http://en.wikipedia.org/wiki/Turing_machine) (TM). Below I just copied the short description in [KN97], which is pretty intuitive without losing much rigor. For detailed definition, you can see standard textbooks such as [Sip02], or the wiki links above.



The figure is here:



Proving lower bounds (i.e. computational hardness) is one major task for computational complexity theory, and the more powerful the computation model is, the harder lower bounds are to prove. The most powerful computational model is Turing machine, or equivalently, (general) circuits, thus not surprisingly, the lower bounds on those models are extremely hard to show. After all, the whole [P vs NP](http://en.wikipedia.org/wiki/P_versus_NP_problem) problem is nothing but to show that some NP problem has super-polynomial lower bound on time complexity. The best lower bound on any explicit function, however, is still linear in input size.

Here we will show a quadratic lower bound on time-space tradeoff, which implies a linear lower bound on time complexity already! The idea is to identify some functions f which has a Turing machine M to compute. Then distribute the input variables to two parties, forming a communication complexity question. We will then design a communication protocol by simulating the Turing machine M, with communication cost upper bounded by the complexity of M. On the other hand, we’ll also show a lower bound of the communication complexity. Combining the two, we get a lower bound for the complexity of M.

Let’s carry out the above plan more precisely. The function f is to decide whether an input of 4n bits is of the form zzR, where z is a 2n-bit string and zR is the reverse of z. For example, if z = 01001, then zR = 10010.

**Theorem 1.4** Any multi-tape Turing machine deciding f using T(n) time and S(n) space has

T(n)S(n) ≥ 2n2.

Here we start the proof. Suppose that there is a Turing machine M computing the function using T(n) time and S(n) space. Then it can in particular decide input x02ny, where x and y are both n-bit strings. Of course, on such input, f = 1 iff y = xR.

Now distribute input x to Alice and y to Bob, and their task is to decide whether y = xR. Note that this is essentially the EQn function---as long as Bob first reverses his input y. So we know that the deterministic communication complexity of this protocol is n, even if only one party finally knows the answer. (Whether one party or two parties knowing the answer differs only by 1 communication bit, because whoever first knows the answer can send it to the other party.) Next we will design a protocol for the problem using T(n)S(n)/2n communication bits, implying that T(n)S(n) ≥ 2n2, as desired. (Note that in time T(n), a machine can use at most T(n) space, namely S(n) ≤ T(n). So the above time-space tradeoff implies a linear lower bound on time complexity: T(n) ≥ n.)

We will design the protocol by simulating M. Suppose that the read-head of the input tape is initially at the position of x1. Alice runs M until the read-head crosses from the 3nth bit to the (3n+1)th bit (namely y1), at which point Alice transfers all the content of the read-write tapes as well as the state information in the automata to Bob. With all this info, Bob can take over and continue to run M, until the read-head crosses from the (n+1)th bit to the nth bit. Then Bob gives all the info he has back to Alice, and so on. Finally, when M stops and outputs the answer, Alice or Bob also output the same answer.

What’s the communication cost of this protocol? For each transfer, the communication is at most S(n)+O(1). How many such transfers happen? Note that each transfer implies the read-head to travel 2n bits---those middle input bits with all 0’s---which takes at least 2n steps of time. Since we have T(n) time in total, the number of transfers is at most T(n)/2n. Altogether, the communication is at most S(n)T(n)/2n, as claimed.

## 2. Query complexity.

While we usually talk about O(n2) or O(n3) time complexity, sometimes the data is too large and it's not even affordable to read the entire data. (Consider the Internet traffic, or phone record, citation graph, etc.) Fortunately, for some computational problems, we don't need to read the entire input (though the function does depend on all input variables).

For a function f:{0,1}n→{0,1}, a *query algorithm* makes queries in the form of “xi = ?”. A query algorithm *computes* f if it outputs f(x) correctly for each input x. The algorithm can be deterministic, randomized, or quantum. As previously, we sometimes allow a small error probability for randomized and quantum algorithms (though we won’t talk about quantum algorithms in this course). The cost of an algorithm is the maximum number of queries it needs to make, and the query complexity of a function f is the minimum cost of any query algorithm that computes f.

**Example 2.1**: **Address functions**. The function f has two parts of input, an n-bit string x and a log(n)-bit string i. The function is defined as f(x,i) = xi, namely treat the second part input as an index of position, on which the value of the first part input is the function value. Note that the function does depend on all n+log(n) input variables. But there is a simple query algorithm reading only log(n)+1 input bits: First read all bits of the second part input, and then read the ith bit of the first part. The query complexity, log(n)+1, is much smaller than n+log(n), the number of variables.

**Example 2.2**: **AND-OR Tree**. The function is defined by a complete binary tree, with gates at internal nodes being AND and OR alternatively with levels. For example, the top gate is AND, and the two second-level gates are OR, and the four third-level gates are AND, and so on. The n leafs correspond to the n input variables. If we evaluate along the gates in the tree from bottom leaves upwards, then the top level gate outputs a value, which defines the function value on that input.

If we want to compute the function deterministically, then it’s not hard to see that it needs to read all n input variables in the worst case. (Can you show it?) However, if we use randomness, then reading a very small fraction of it is enough.

**Theorem 2.1**. There is a zero-error randomized query algorithm solving the AND-OR Tree problem with only O(n0.753…) queries in expectation.

It’s not hard to see that the problem is the same as NAND Tree, where each node is the NAND gate. (Verify this yourself.) The algorithm is sometimes called alpha-beta pruning. It’s very simple: it recursively runs in a top-down manner. For each gate G, randomly choose one Gi of the two children G1, G2 to evaluate it (recursively). If Gi evaluates to 0, then we know that G also evaluates to 1, thus we don’t need to compute the other child of G. If Gi evaluates to 1, then we have to compute the other child.

The algorithm surely has zero error. What’s the expected number of queries? Suppose the height of the current subtree is h, and denote the expected time to evaluate the subtree is Tb, where b is the value that the subtree evaluates to. Then it’s not hard to see that

 T0(h) = 2T1(h-1) // we have to compute both children (who both evaluate to 1)
 T1(h) = ½ T0(h-1) + ½ (T1(h-1)+T0(h-1)) = T0(h-1) + ½ T1(h-1) // w.p. 1/2, we evaluate only one child.

Combining the two relations, we have T1(h) = ½ T1(h-1) + 2T1(h-2). Solving this recursion we get

T1(h) = $\left(\frac{1+\sqrt{33}}{4}\right)^{h}$= (2h)0.753… = n0.753…

Finally, let’s mention that the algorithm is optimum in the sense that any randomized algorithm needs these many queries. If you are indeed interested in more details of the algorithm and its optimality, see paper [SW86].

**Example 2.3: Sink problem**. This time we are given a directed graph, and we wish to know whether the graph has a “sink” vertex, which has out-degree n-1 and in-degree 0. (Namely, everyone else points to it, but it doesn’t point to anyone.) What we can ask is for any ordered pair (i,j), whether there is an edge from i to j. What’s the minimum number of queries we need?

## References

[KN97] Eyal Kushilevitz and Noam Nisan, **Communication Complexity**, *Cambridge University Press*, 1997.

 [SW86] Michael E. Saks, Avi Wigderson, **Probabilistic Boolean Decision Trees and the Complexity of Evaluating Game Trees**, *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pp. 29-38, 1986.

[Sip12] Michael Sipser, **Introduction to the Theory of Computation**, 3 edition, *Course Technology*, 2012.