CSE 200 - Computability and Complexity
Two tapes are (quadratically) better than one

Russell Impagliazzo and Jiawei Gao

Sept, 2016
Today’s agenda

1. Look at one case of the Church-Turing thesis: one tape vs. multitape TM’s.
2. In doing so, introduce concepts to be used throughout the class: non-determinism, randomization.
3. A whirlwind tour of communication complexity, which will be used to prove lower bounds on any one tape TM.
We can (and will) define analogs of these classes for other models of computation. We’ll use the subscript “Model” to say the class relative to some other model.

The Church-Turing thesis can be then stated: For any reasonable model of computation, \( \text{COMP}_{\text{Model}} = \text{COMP} \).

The Time-bounded version is: \( P_{\text{Model}} = P \).

It would really be nice if we had a fine-grained time bounded version, e.g., \( \text{TIME}_{\text{Model}}(T(n)) = \text{TIME}(T(n)) \). We will see soon why this is too much to hope for.
Sandwiching

We might never have a single model that addresses all concerns simultaneously. But if we can show that both a model that seems too weak and one that seems too strong are equivalent to TM’s, that is evidence that TM’s approximate whatever real-world computation is. Our too weak model will be one-tape TM, and our too strong model will be a variant of RAM.
A physicist might think multi-tape TMs are not realistic, in that they require instantaneous communication over arbitrarily large distances.
A one-tape TM doesn’t require this, having only one tape head. Can we simulate an arbitrary TM with a one-tape TM? If so, what is the cost, in terms of additional time?

**Theorem 1:** For any \( T(n) \geq n \),
\[
TIME_{k-TM}(T(n)) \subseteq TIME_{1-TM}((T(n))^2)
\]

**Corollary** \( COMP_{k-TM} = COMP_{1-TM}, P_{k-TM} = P_{1-TM} \).

**Theorem 2:** There is a language \( L \in TIME_{2-TM}(n) \) but \( L \notin TIME_{1-TM}(o(n^2)) \).
Example language

Let $L = \{x \circ 2^+ \circ x \mid x \in \{0, 1\}^*\}$.

What would a two-tape TM for $L$ look like?
What would a one-tape TM for $L$ look like?
With two tapes

Let $L = \{x \circ 2^+ \circ x | x \in \{0, 1\}^*\}$.

1. While symbol on first tape head is either 0 or 1, copy it to the second tape, and keep on moving both heads right.
2. When we see a 2, move the second tape head back to start.
3. While we see 2’s, move the first tape head right.
4. Move the two tape heads together, comparing symbols. If we always match and get to the end at the same time, accept, else reject.
With one tape

Let $L = \{x \circ 2^+ \circ x \mid x \in \{0, 1\}^*\}$. Use a special symbol to mark where we are in the string before and after the 2’s. Compare the strings before and after the two’s by moving the tape head through the whole input, moving the marker by 1 to the right each time (and saving the read bit in the state). If they always match, and end at the same time, accept. Otherwise, reject. This takes $O(n^2)$ steps total. But is this the best we can do?
The simulation of multi-tape TM by one tape TM

We first want to prove: \( \text{TIME}_{k-TM}(T(n)) \subseteq \text{TIME}_{1-TM}(T(n)^2) \) for any \( T(n) \geq n \).

To do this, we need to show how to take as input an arbitrary \( k \)-tape TM \( M \) that runs in some time \( T(n) \), and simulate it by a one-tape TM \( S \). (Note: we are given \( M \), but get to define \( S \) from \( M \)).

First question: How do we put \( k \) tapes worth of information on one tape?
The simulation

There are many possible answers to this. We could interleave the cells of the different tapes, say mapping the $j$ th cell of tape $i$ to cell $jk + i - 1$. We could concatenate the various symbols on each tape, with cell $j$ containing a list of symbols, listing the contents of the $j$’th cells from each tape. Or, as pictured below by Ian Finlayson of UMW, we could concatenate the tapes. In any case, we need to keep track of where the tape heads are on each tape.

For symbol $\sigma$ in $M_k$’s alphabet, we use symbol $\downarrow \sigma$ to represent symbol $\sigma$ with a tape head on it. We’ll also introduce a special symbol to denote where the tapes are separated.
Picturing the simulation

\[
\begin{align*}
M & \quad 01010 \dddot{u} \\
& \quad \dddot{aa}u \\
& \quad \\
S & \quad \#01010\#\dddot{aa}a\#\dddot{ba}u \\
\end{align*}
\]
Initializing $S$

Before we start actually simulating $M$, we need to put the memory of $S$ in the format above. Since at the start, all of the $k - 1$ non-input tapes of $M$ are empty we can do this by:

1. Move to the end of the input
2. Repeat $k - 1$ times: Add a tape divider symbol and an empty cell with tape head symbol to the end of the tape.
3. Add a tape divider symbol at the end
4. Move the tape head back to the start

This takes $O(n)$ steps of $S$. 
Simulating one step of $M$

To simulate a step of $M$, $S$ needs to first know what is under each tape head of $M$. Then $S$ can look up the appropriate action (Write, Left, Right, etc.) and then simulate this action.

1. We start with the previous state of $M$ stored, and the tape head at the front of the tape.
2. Sweep entire tape.
3. Store the $i$’th symbol marked with a tape head as the value under tape head on $i$’th tape (can be stored in finite state control)
4. Look up new state and action to perform.
5. If action involves tape $i$, we scan until we see $i$ tape dividers, then look for the tape head symbol.
6. Write, Left, Accept, Reject are easily simulated. If the action is to move Right and we are already at the end of the tape, we first need to move every symbol to the right over one space (start at the far right, and proceed until we reach the $i$’th tape head symbol.) and then create a new blank with tape head symbol in the freed up space.
How much time did we waste?

For each step of $M$, $S$ scans through the whole memory at least once, sometimes twice, and then moves back to the start. That means the cost for $S$ of each step of $M$ is $O(\text{used memory of } M)$. But $M$ starts with $n$ used places in memory, and only can access one new cell each step. Thus, the amount of memory of $M$ is at most $n + T(n)$. Since we use $O(n)$ time to initialize and $O(n + T(n))$ time per each of $T(n)$ steps, we get a total time of $O(n + T(n)(n + T(n))) = O((T(n))^2)$ since $T(n) \geq n$. 
Let \( L = \{ x \circ 2^+ \circ x \mid x \in \{0, 1\}^* \} \).

We want to prove that \( L \not\in \text{TIME}_{1-TM}(o(n^2)) \), that any one-tape TM for \( L \) takes time \( \Omega(n^2) \).

How can we reason about an unknown TM?

One intuition: Say the input has \( n \) 2's, so is of the form \( x2^n y \).

Then we need to decide whether \( x = y \). But the information about \( x \) is separated from the information about \( y \) by a chasm that takes \( n \) steps to cross. If we can show \( n \) bits of information need to cross this chasm, this shows that time is on the order of \( \Omega(n^2) \).
Communication complexity

We can formalize the question of how much information needs to flow between the two parts of the input in terms of *communication complexity*. (Thanks to Moni Naor for the picture)

**Communication Complexity**

Let \( f : X \times Y \mapsto Z \)

Input is split between two participants

Want to compute outcome: \( z = f(x, y) \)

while exchanging as few bits as possible
Communication protocol

To make this precise, we assume without loss of generality that Alice and Bob alternate sending one bit messages. Alice gets input $x$, and her strategy is a function $A(x, C)$ where $C$ is an evolving bit string. Bob get input $y$ and has strategy $B(y, C)$. Note that we don’t assume either $A$ or $B$ are efficiently computable, we just want them to be functions of what Alice and Bob know.

Then we define $C_0$ is the empty string, and for odd $t$, $b_t = A[x, C_{t-1}]$ and for even $t$, $b_t = B[y, C_{t-1}]$, and then $C_t = C_{t-1} \circ b_t$. In other words, Alice or Bob send one more bit that is a function of the conversation so far and their private input. That bit is then public knowledge, so either of them can use it to compute future messages.
Communication complexity of a problem

We look at communication complexity for solving a decision problem, is the pair \((x, y) \in L\)?

In a \(T\)-bit protocol for \(L\), at the end of the protocol, the last bit \(b_T\) should be a 1 if and only if \((x, y) \in L\).

The communication complexity of a language of \(L\) on \(n\) bit strings, denoted \(CC(L_n)\), is the minimum \(T\) so that there is a \(T\) bit protocol for \(L\) when the inputs are of size \(n\). Note that this measure is \textit{concrete} and \textit{non-uniform}, in that protocols don't have to treat different input sizes in a uniform way, and there is a finite collection of possible “algorithms” for a fixed size, so there is always a specific number that represents the communication complexity. In contrast, a TM uses the same algorithm every input size, and the complexity of a problem is only defined asymptotically. Nevertheless, communication complexity will help us understand TM’s.
Communications complexity was introduced by Andy Yao in 1979, as a model for distributed computation.
Since then, it has been used to understand a huge variety of seemingly unrelated subjects, some of which are:

1. Time-space trade-offs for computation
2. Area-time trade-offs in VLSI
3. Streaming algorithms and their limitations
4. Lower bounds on data structures
5. Circuit lower bounds
6. Pseudo-random generators for small space computation
7. Proof complexity

and many more. We’ll only have time for a superficial treatment.
A seeming digression

To relate communication complexity to one-tape TM’s, we’ll introduce two seemingly unrelated generalizations, non-determinism and randomization. We could, and in past years, we did, give the proof without talking about these concepts. But since they will turn out to be central concepts throughout the class, and introducing them simplifies the argument, I thought it would be good to introduce them right away. We’ll first look at these concepts for TM’s and then for communication complexity.
A non-deterministic TM has a transition function where instead of a single state and action, it could return a list of possible states and actions. We “non-deterministically choose” one from the list each step. Thus, instead of a single run, on one input, we could have a set of many different runs. Somewhat arbitrarily, we say that a non-deterministic machine accepts $x$ if some run of the machine on $x$ accepts. We still use the worst-case number of steps over all runs as the time of the machine. Non-deterministic machines are not a reasonable model of computation, but they are a useful conceptual tool to classify problems.
Randomized TM’s

A randomized TM is the same, except we view the machine as picking uniformly at random whenever there are multiple choices. Thus, we get a probability distribution over possible runs, and so both the acceptance or rejection and the time are random variables. A zero-error randomized machine for $L$ never outputs the wrong answer, but we can consider its expected running time. A bounded error randomized machine for $L$ gets the right answer on the overwhelming majority of runs (say $2/3$ fraction at least.) We could consider either the worst-case or expected time for such a machine.

Since many physical phenomena are apparently random, randomized machines are both a reasonable model of computation which challenges the Time-bounded Church-Turing thesis, and a useful tool to understand issues for deterministic complexity.
Non-deterministic and randomized communication complexity

In a non-deterministic protocol, Alice and Bob’s strategies are functions $A(x, C)$ and $B(x, C)$ that might return the set of both 0 and 1 “non-deterministically” chosen as well as having the option to return the set containing a single bit as before. A possible conversation $C = b_1...b_T$ is one where for odd $t$, $b_t \in A[x, C_{t-1}]$ and for even $t$, $b_t \in B[y, C_{t-1}]$, and then $C_t = C_{t-1} \circ b_t$. In other words, Alice or Bob send one more bit that is a function of the conversation so far and their private input. That bit is then public knowledge, so either of them can use it to compute future messages. Thus, there is a set of possible conversations, some accepting some rejecting. We consider the protocol as accepting $(x, y)$ if any of the conversations for $(x, y)$ accept. The smallest number of bits in a non-deterministic protocol accepting precisely $L_n$ is denoted $NCC(L_n)$.
Randomized communication complexity

A randomized protocol is the same as a non-deterministic protocol, except we view the non-specified bits as being chosen at random, giving a probability distribution on the possible conversations (and we allow a variable number of bits to be sent).

The zero-error communication complexity of a language (ZPCC($L_n$), sometimes called $R_0(L_n)$) is the expected number of bits sent (on the worst-case input) for a randomized protocol with no probability of error.

The bounded-error communication complexity of a language, $BPCC(L_n)$ or sometimes $R_{1/3}(L_n)$ is the number of bits sent in a protocol that has at most a $1/3$ chance of a mistake.
Since we could view a zero-error protocol as a non-deterministic protocol, and a deterministic protocol as a zero-error protocol, \(NCC(L_n) \leq ZPCC(L_n) \leq CC(L_n)\). Similarly, \(BPCC(L_n) \leq CC(L_n)\). The relationship between \(NCC\) and \(BPCC\) is “complicated”.
We are interested in the following communication protocol:
Equality: Alice gets $n$ bit string $x$, Bob $y$, and they need to see if $x = y$.
Inequality: Same, but accept if $x \neq y$.
These seem like the same problem, and in most models they are, but non-deterministically, they are quite different.
Claim: $NCC(Inequality_n) \in O(\log n)$
To prove that two strings are not equal, we guess (non-deterministically choose) and then accept if:
Claim: \( NCC(Equality_n) \in \Omega(n) \).
Assume there were a non-deterministic protocol \( A, B \) with \( T < n \).
Then for every \( x \in \{0, 1\}^n \) there is an accepting conversation \( C_x \in C(x, x) \) (pick one arbitrarily if there are more than one).
Since there are fewer than \( T \) bits describing such a conversation, there are \( 2^T < 2^n \) possible conversations, so by PHP, there are \( x \neq y \) with \( C_x = C_y = b_1...b_T \). Each odd \( b_i \in A(x, b_1...b_{i-1}) \), and each even \( b_i \in B(y, b_1...b_{i-1}) \), so \( C_x \in C(x, y) \), i.e., it is also an accepting conversation when Alice gets input \( x \) and Bob another input \( y \neq x \), This is a contradiction, since \( C_x \) must both be accepting and rejecting.
Corollary: \( ZPCC(Equality_n) \in \Omega(n) \).
Relating to one-tape TM

Lemma: Let $M$ be a one-tape TM for $L = \{x^{2^+}x | x \in \{0, 1\}^n\}$. Then $ZPCC(\text{Equality}_n) \in O(T(3n)/n)$.

Corollary: For such a one-tape TM, $T(n) \in \Omega(n^2)$.

$n \leq ZPCC(\text{Equality}_n) \leq cT(3n)/n$, so $T(3n) \geq n^2/c$. 
Alice picks a random $i \in \{1, \ldots, n\}$, and sends it to Bob. Alice and Bob together will simulate $M$ on input $x2^n y$. Alice is in charge of the first $n + i$ cells of the tape, Bob the rest. To maintain the configuration, Alice needs to send Bob the state of the control whenever the tape head moves from position $n + i$ to position $n + i + 1$, and Bob likewise needs to send the state to Alice when it moves in the opposite direction. In this way, they can keep track of the configuration at all times, and in particular see whether $M$ accepts or rejects.
M runs for $T(3n)$ steps, and in each step, the tape head can only be in one position, moving in one direction. Since $i$ is picked at random, the probability that any particular time step $t$ communication is required is at most $1/n$. Each such time, the amount of communication is the log of the number of states, $O(1)$. Thus, the expected amount of communication is at most $O(T(3n)/n)$. 

What is the expected communication
Claim: $BPCC(Equality_n) \in O(\log n)$.

View $x$ and $y$ as binary numbers $< 2^{n+1}$. If $x \neq y$, $x - y$ is at most $2^{n+1}$ and has at most $n$ prime factors. Enumerate the first $3n$ primes, $p_1, \ldots, p_{3n}$. The prime number theorem says that all of these primes are at most $O(n \log n)$ in value. Alice picks a prime $p$ from this list at random and sends it to Bob, and sends $x \mod p$ to Bob. Bob checks whether this is also $y \mod p$. If not, they know $x \neq y$ and reject, and otherwise they accept.

Since $x \mod p < p < cn \log n$, the number of bits required is $O(\log n)$.

At most $n$ of the $3n$ primes on the list divide $x - y$, and if not, $x \mod p \neq y \mod p$. So the probability that the protocol accepts incorrectly is at most $1/3$.
Conclusions

One-tape machines can simulate arbitrary TM’s, but at a quadratic expense. Randomness can solve problems exponentially faster, at least in terms of communication.