CSC 2401 Introduction To Complexity—Fall 2001
Lecture #23, 29 November 2001
Lecturer: Allan Borodin
Scribe: Tristan Miller

23.1 Time-space tradeoffs

Recall from the previous lecture $DTISP(t,s) = \{L \mid L$ is decided by some deterministic M running simultaneously in time $t(n)$ and space $s(n)\}$. For notational convenience, we use $TISP(t,s)$ to refer to $DTISP(t,s)$.

Consider the following result by Lipton-Viglas:

$$SAT \notin TISP(n^{\sqrt{2}-\epsilon}, n^{o(1)}).$$

We will now prove this using the method of indirect diagonalization, as follows:

1. Assume $SAT \in TISP(t,s)$. (Essentially, ignoring some log factors this means we assume that $NTIME(n) \subseteq TISP(t,s)$.)
2. Using this assumption, prove other inclusions.
3. Eventually, contradict a known direct diagonalization.

The direct diagonalization used by Lipton-Viglas is

$$t_1(n+1) \in o(t_2(n)) \Rightarrow NTIME(t_1) \subsetneq NTIME(t_2),$$

but as van Melkebeek observes, we can use the easier direct diagonalization

$$\Sigma_1 TIME(t) \not\subseteq \Pi_1 TIME(o(t)),$$

where t is constructible. This says that $\exists L \in \Sigma_1 TIME(t) - \Pi_1 TIME(o(t))$. Sketch of this diagonalization: We need the fairly easy fact that a k-tape non deterministic TM running in time t can be simulated by a 2-tape non deterministic TM running in time $O(t)$. The idea is to guess the entire sequence of non deterministic moves. states and symbols read in advance and then verify the computation for each tape. Now to construct something in $\Sigma_1 TIME(t) - \Pi_1 TIME(o(t))$ we diagonalize by simulating arbitrary Π_1 computations for t steps but changing all universal sattes to existential states and reversing the accepting and rejecting states.

Another simple fact we will use is $C_1 \subseteq C_2 \Rightarrow$ co-$C_1 \subseteq$ co-C_2. Also, if $C_1 =$ co-C_1 (as you would have in any deterministic class, *e.g.* $C_1 = TISP(t,s)$) and $C_1 \subseteq C_2$, then co-$C_1 = C_1 \subseteq$ co-C_2.

23.1.1 Tools for proving other inclusions

The following tools are useful for step 2 of the above-described method.

(a) **Trading alternations for time.**

$$TISP(t,s) \subseteq \Sigma_2 TIME\left(bs + \frac{t}{b} + b\right).$$

All we need to do is choose $b = \sqrt{t/s}$, so

$$TISP(t,s) \subseteq \Sigma_2 TIME(\sqrt{ts}).$$

See the discussion and diagram from van Melkebeek's survey, *pp.* 6–7. Observe that there are b intermediate configurations t/b steps away from each other. We write them in time bs, and then universally verify them.

(b) **Trading time for alternations.**

This is based on the assumption that $NTIME$ is not too difficult to compute. If $NTIME(n) \subseteq TIME(t)$, then

$$\Sigma_2 TIME(\tau) \subseteq \Sigma_1 TIME(t \circ \tau).$$

Why is this true?

What does $L \in \Sigma_2 TIME(\tau)$ mean? It means that

$$\exists y_1 \in \{0,1\}^\tau : \underbrace{\forall y_2 \in \{0,1\}^\tau \; \exists R(x, y_1, y_2)},$$

where R is a certificate linear in τ. Consider the expression marked with an underbrace: it is a $\Pi_1 TIME(\tau)$ thing, so by the assumption it is in $DTIME(t \circ \tau)$. So the whole proposition becomes $\Sigma_1 TIME(t \circ \tau)$.

That is, since $DTIME(t) = \text{co-}DTIME(t)$, $\Pi_1 TIME(n) \subseteq DTIME(t)$, then

$$\Pi_1 TIME(\tau) \subseteq DTIME(t \circ \tau).$$

So let's put this all together. First we assume that $NTIME(n) \subseteq TISP(n^c, n^{o(1)})$. Then $NTIME(\tau) \subseteq TISP(\tau^c, \tau^{o(1)})$. Using (a),

$$TISP(\tau, \tau^{o(1)}) \subseteq \Sigma_2 TIME(\tau^{c/2+o(1)}).$$

Using (b),

$$\Sigma_2 TIME(\tau^{c/2+o(1)}) \subseteq \Sigma_1 TIME(\tau^{c^2/2+o(1)}).$$

Then

$$NTIME(\tau) \subseteq TISP(\tau^c, \tau^{o(1)}) \subseteq NTIME(\tau^{c^2/2+o(1)})$$

. If $c^2/2 < 1$ (*i.e.* $c < \sqrt{2}$), this contradicts the non-deterministic time hierarchy theorem.

Why can we get away with van Melkebeek's simpler diagonalization,

$$\Sigma_1 TIME(t) \not\subseteq \Pi_1 TIME(o(t))?$$

$$TISP(t,s) \subseteq \Pi_2 TIME(\sqrt{ts})$$

$$\Pi_2 TIME(\tau) \subseteq \Pi_1 TIME(t \circ \tau)$$

Using (a),

$$TISP(\tau, \tau^{o(1)}) \subseteq \Pi_2 TIME(\tau^{c/2+o(1)}).$$

Using (b),

$$\Pi_2 TIME(\tau^{c/2+o(1)}) \subseteq \Sigma_1 TIME(\tau^{c/2+o(1)}).$$

23.2 Randomized algorithms

This material is presented in Chapter 8 of the textbook. We now introduce a number of complexity classes based on probabilistic computation:

$$\begin{aligned} BPP &= \text{bounded error probabilistic poly-time} \\ RP &= \text{randomized poly-time} \\ ZPP &= \text{zero-error expected poly-time} \end{aligned}$$

More specifically, $BPP \equiv$ 2-sided error, $RP \equiv$ 1-sided error, and $ZPP = RP \cap \text{co-}RP \equiv$ 0-sided error. These classes fit into our hierarchy as follows:

$$P \subseteq ZPP \subseteq RP \subseteq BPP.$$

23.2.1 Conceptualization

BPP. Think of a Turing machine, recognizing some language L, which flips a biased coin to determine whether some input string x is accepted or rejected. Then we say that for BPP, there is a high probability that the sequence of coin tosses is such that the machine makes the correct determination. That is, if $x \in L$, chances are better that we'll accept rather than reject. (The term *bounded* means bounded away from $Pr = 0.5$.) However, we could still mistakenly reject if $x \in L$, or mistakenly accept if $x \notin L$. Note that if the machine runs in time $p(n)$, there are $2^{p(n)}$ possible coin flip sequecnes. (Without loss of generality and to simplify the development, one can assume that the probabilistic TM flips a coin on every step.)

RP. RP is similar to BPP except that our Turing machine will never incorrectly accept some $x \notin L$. However, it may still incorrectly reject some $x \in L$.

ZPP. ZPP may be conceptualized as follows: we have a Turing machine which keeps flipping coins. On every input x, expected time for completion will be $p(n)$. We never quite know when to stop, but the machine never makes an error.

23.2.2 Formalization

BPP is defined to be 2-sided error with probability of being correct $\geq \frac{1}{2}+\varepsilon$ for some constant $\varepsilon > 0$. Hence the error is $< \frac{1}{2}-\varepsilon$. However, we can run the computation lots of times—*i.e.* $q(n) \cdot n$ independent trials of the machine—and choose the majority answer. In this way, we lower the probability of error to be $\leq 2^{-q(n)}$.

BPP is a good mathematical abstraction of efficiently-computable problems (*cf.* enlarging our previous abstraction identifying effecient with P). When we relate the classes we have just described to known classes, things work out as follows:

$$\left.\begin{array}{l} RP \subseteq BPP \\ RP \subseteq NP \end{array}\right\} \subseteq PSPACE$$

Note, however, that BPP and NP are not known to be comparable.

A better result is as follows:

$$\left.\begin{array}{l} RP \subseteq BPP \\ RP \subseteq NP \end{array}\right\} \subseteq \Sigma_2 \cap \Pi_2$$

Some people now suspect that $BPP = P$, though it is not obvious how to show this. Even so, there is still an important role for randomization—*e.g.* for cryptography, for simulation, for "practical algorithms", etc.