03684155: On the P vs. BPP problem. 30/12/2016 – Lecture 12

Time Hierarchy Theorems

Amnon Ta-Shma and Dean Doron

1 Diagonalization arguments

Throughout this lecture, for a TM M, we denote M^t to be the machine M, when restricted to run for at most t steps by some time-out mechanism. M^t rejects if M did not halt within t steps.

1.1 Deterministic time hierarchy

The most basic approach to show hierarchies uses simulation and diagonalization. Say we want to show a hierarchy for DTIME, like we do in the complexity course. Consider an effective enumeration of DTIME machines, assigning to each input x a machine M_x in the enumeration. We define L using a machine M which given an input x simulates M_x^t on x and does the opposite (it can do the opposite sine DTIME is closed under complement).

L is not in $\mathsf{DTIME}(t(n))$, since for any DTIME machine M' running in time $t(n)$, $M(M') \neq M'(M')$. Thus, we have showed the required lower bound on the complexity of L. The upper bound depends on the complexity of simulations of machines in the enumeration. If $\mathsf{DTIME}(t(n))$ machines can be simulated in time T, we obtain the required upper bound on the complexity on L, and $L \in \mathsf{DTIME}(T(n)) \setminus \mathsf{DTIME}(t(n))$. Specifically, we know that:

Theorem 1. *For any time-constructible t and T such that $t(n)\log t(n) = o(T(n))$, $\mathsf{DTIME}(t) \subsetneq \mathsf{DTIME}(T)$.*

This simulation and diagonalization approach works for any complexity class that both has an effective enumeration of machines defining languages in the class (this *excludes* BPTIME) and is closed under complement. Thus, it works also for deterministic and nondeterministic space, and for PP.

Throughout this lecture, the notion of time-constructibility will be important to us. Formally:

Definition 2. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We say that f is* time-constructible *if $n \leq f(n) \leq 2^n$ and there is a deterministic TM M such that on input n, M runs at most $f(n)$ steps and outputs $f(n)$.*

1.2 Indirect diagonalization – nondeterministic time hierarchy

Obviously, the above approach fails for NTIME, as we cannot simply "do the opposite". We can avoid this problem by an "indirect diagonalization" – assume that a hierarchy theorem does not hold and then find a way to amplify this assumption until deriving a contradiction to some other hierarchy theorem proven by direct diagonalization. Simply stated, we can negate if we have exponentially more time. But then we won't get a fine hierarchy at all. Thus, we only try to disagree at least once in an exponentially-large interval.

Theorem 3. *For any time-constructible functions t and T such that $t(n+1) = o(T(n))$, $\mathsf{NTIME}(t) \subsetneq \mathsf{NTIME}(T)$.*

We prove this theorem for the weaker assumption of $t(n+1)\log(t(n+1)) = o(T(n))$.

Proof. For the k-th nondeterministic TM M_k, associate an interval $I_k = (\ell_k, u_k] \subseteq \mathbb{N}$, and they should all be contiguous. Also, the upper bound should be exponentially larger than the lower bound, so take $u_k = 2^{\ell_k^2}$. Now, define a nondeterministic TM D, that on input 1^n:

1. Find k for which $n \in I_k$.
2. (a) If $n < u_k$, simulate $M_k^{T(n)}(1^{n+1})$ (nondeterministically).
   (b) If $n = u_k$, deterministically decide if M_k accepts 1^{ℓ_k+1} by trying all computation paths, for $T(\ell_k + 1)$ steps, and do the opposite.

Note that in step 2(a) we do note negate, but simulate on a larger input, and in step 2(b) we negate, but on a much smaller input.

By construction, D runs in $\mathsf{NTIME}(T(n))$ as step 2(b) takes $2^{T(\ell_k+1)} \leq T(n)$ time. Assume towards contradiction that there exists a nondeterministic TM M running in nondeterministic time cn such that $L(D) = L(M)$. We may assume that there are infinitely many k-s for which $L(M) = L(M_k)$, so in particular we can think of $k = k_0$ as being arbitrarily large. Specifically, on inputs of length $n \geq \ell_{k_0}$, M_{k_0} can be simulated in less than $T(n)$ time.

We thereby have, on input 1^n where $n \in I_{k_0}$:

1. If $n < u_k$ then $D(1^n) = M_{k_0}(1^{n+1})$.
2. If $n = u_k$ then $D(1^{u_k}) \neq M_{k_0}(1^{u_k+1})$.

By our assumption, M_{k_0} and D agree on all inputs in the interval I_{k_0}. Together with item (1), this implies that $D(1^{u_{k_0}}) = M_{k_0}(1^{\ell_k+1})$, in contradiction to item (2). □

2 Hierarchies for probabilistic classes with advice

For classes like BPTIME or RTIME we cannot effectively enumerate the corresponding Turing machines: A probabilistic TM that decides a BPTIME language must possess a very special property. It must hold that for *any* x, either $\Pr[M(x) = 1] > 2/3$ or $\Pr[M(x) = 1] < 1/3$. It is undecidable to test whether a machine M satisfies this property and it is also unknown whether one can test if M satisfies this property for a specific $x \in \{0,1\}^n$ using less than 2^n steps.

Thus, other than giving trivial results (one of which we will see in the exercise), the above techniques do not work.

The work of Barak [2] overcomes this obstacle by establishing hierarchies with *advice*, as opposed to hierarchies for completely uniform classes:

Theorem 4. *For every constant $d \geq 1$ there exists a constant γ such that*

$$\mathsf{BPTIME}(n^d)/\gamma \log n \subsetneq \mathsf{BPTIME}(n^{d+1})/\gamma \log n.$$

The above theorem will follow easily from the following one, which will constitute our main result for this lecture.

Theorem 5. *For every constant $d \geq 1$, $\mathsf{BPP}/\log\log n \not\subseteq \mathsf{BPTIME}(n^d)/\log n$.*

Proof. (of Theorem 4). For simplicity, consider $d = 1$, and suppose towards contradiction that for every constant γ,

$$\mathsf{BPTIME}(n)/\gamma\log n = \mathsf{BPTIME}(n^2)/\gamma\log n.$$

By padding, this implies that

$$\mathsf{BPTIME}(t(n))/\gamma\log t(n) = \mathsf{BPTIME}(t(n)^2)/\gamma\log t(n)$$

for every time-constructible $t(n)$. In particular, choose $t(n) = n^2$ and $\gamma = \frac{1}{2}$, so

$$\mathsf{BPTIME}(n^2)/\log n = \mathsf{BPTIME}(n^4)/\log n.$$

However, $\mathsf{BPTIME}(n^2)/\log n = \mathsf{BPTIME}(n)/\log n$, so $\mathsf{BPTIME}(n)/\log n = \mathsf{BPTIME}(n^4)/\log n$. Continuing the same way, we have that for every constant c,

$$\mathsf{BPTIME}(n^c)/\log n = \mathsf{BPTIME}(n)/\log n.$$

But then,

$$\mathsf{BPTIME}(n^c)/\log\log n \subseteq \mathsf{BPTIME}(n^c)/\log n = \mathsf{BPTIME}(n)/\log n,$$

which implies $\mathsf{BPP}/\log\log n \subseteq \mathsf{BPTIME}(n)/\log n$, contradicting Theorem 5. □

2.1 Some preliminaries

We will need some "scaling-up" lemmas. The proofs are by padding, and we give them as an exercise.

Lemma 6. *For every constant $d \geq 1$, if $\mathsf{BPTIME}(n^d) = \mathsf{BPTIME}(n^{d+1})$ then $\mathsf{BPTIME}(n^d) = \mathsf{BPP}$.*

Also:

Lemma 7. *For every constant $d \geq 1$, if $\mathsf{BPTIME}(n^d) = \mathsf{BPP}$ then $\mathsf{BPTIME}(t(n)) = \mathsf{BPTIME}(t(n)^c)$ for every constant $c \geq 1$ and time-constructible function t that satisfies $t(n) \geq n^d$.*

Combining the two lemmas, we obtain:

Corollary 8. *For every constant $d \geq 1$, if there exists a time-constructible function t and a constant $c > 1$ such that $t(n) \geq n^d$ and $\mathsf{BPTIME}(t(n)) \subsetneq \mathsf{BPTIME}(t(n)^c)$ then $\mathsf{BPTIME}(n^d) \subsetneq \mathsf{BPTIME}(n^{d+1})$.*

2.2 Attempt I – without the advice

Suppose we want to prove Theorem 5 for $d = 1$ without the advice so assume towards contradiction that $\mathsf{BPTIME}(n) = \mathsf{BPP}$. We follow two conceptual ideas:

2.2.1 Optimal algorithms

An *optimal algorithm* for a language L is an algorithm that is at worst polynomially slower than any algorithm for L. We know by Lemma 7 that $\mathsf{BPTIME}(t) = \mathsf{BPTIME}(t^c)$ for every constant c and time-constructible t. This means that if we have a language L for which there exists a $\mathsf{BPTIME}(T)$ algorithm, it will also have a $\mathsf{BPTIME}(T^{1/c})$ algorithm. Wee see that no algorithm in BPP has an optimal algorithm.

To derive a contradiction, we need to come up with an optimal algorithm! That is, a $\mathsf{BPTIME}(T)$ TM A that solves some L and any other TM A' that solves L must take at least $T^{1/c}$ steps for some constant c.

How can we come up with such an optimal algorithm? The idea goes as follows: A will enumerate all TMs of size, say, $\log n$, and will try to use each machine to solve L. Then, for all but finitely-many inputs, A' will be one of the machines enumerated by A. Therefore, A is able to use A' to solve L and so A will be at most polynomially slower than A'. There is an obvious obstacle – how do we know which TM solves L?

2.2.2 Instance Checkers

As we are free to choose L as we wish, we choose it to be one that is equipped with an *instance checker*.

Definition 9. *An* instance checker *for a language L is a probabilistic polynomial-time oracle TM C with output in $\{0, 1, quit\}$ such that for every x and every oracle O,*

$$\Pr[C^O(x) \notin \{L(x), quit\}] < 2^{-\Omega(|x|)}$$

and $\Pr[C^L(x) = L(x)] = 1$. *All the queries C makes should be of linear length in $|x|$.*

Given an input x and an instance checker C for L, our algorithm A will do the following on input $x \in \{0,1\}^n$:

- For $m = 1, 2, \ldots$, for each probabilistic TM M of size $\log n$:
  - Compute $C^{M^m}(x)$.
  - If $a \neq quit$, output a.

By our definition, we see that if there exists a $\mathsf{BPTIME}(t)$ algorithm A' for L, then with high probability, our algorithm A is only polynomially-slower, and outputs $L(x)$.

As a consequence of the $\mathsf{EXP} \subseteq \mathsf{MIP}$ proof, every EXP-complete language has an instance checker [1], so we take L to be such a language. Now, define the function T such that $T(n)$ is the minimal number k such that for every $x \in \{0,1\}^n$, $A(x)$ halts with probability at least $\frac{2}{3}$. We assume T is super-polynomial, as otherwise $\mathsf{BPP} = \mathsf{EXP}$ – for which we have hierarchy theorems.

It may seem like we are done – L can be solved in $\mathsf{BPTIME}(T)$, but not in $\mathsf{BPTIME}(T^{1/c})$ for some constant c, since otherwise A would have halted sooner. However, there are still two problems:

1. A $\mathsf{BPTIME}(T)$ machine must halt with probability 1 on every input, and not only with high probability.

2. The function T is not necessarily time-constructible, so even if we do show that $\mathsf{BPTIME}(T) \not\subseteq \mathsf{BPTIME}(T^{1/c})$ we will not be able to prove $\mathsf{BPTIME}(n) \neq \mathsf{BPP}$.

Note that the first problem is not significant since had T been time-constructible, we could have simply halt within $T(n)$ steps.

This is where non-uniformity enters the picture.

2.3 Attempt II – with advice

Every function $n \leq T(n) \leq 2^n$ can be approximated by a time-constructible function T' using only $\log\log n$ bits of advice – it is simply the value $\log\log T(n)$. By defining $T'(n) = 2^{2^{\lceil \log\log T(n) \rceil}}$, we have that $T(n) \leq T'(n) \leq T(n)^2$.

This approach does not "put the advice in both sides". That is, it only proves that there exists an $L \in \mathsf{BPTIME}(T)/\log\log n$ such that $L \notin \mathsf{BPTIME}(T^{1/c})$, which is not what we want (and also trivial). However, re-visiting our algorithm A, we see that it works even against non-uniform algorithms with $\log n$ advice bits. This will give us

$$\mathsf{BPTIME}(T)/\log\log n \not\subseteq \mathsf{BPTIME}(T^{1/c})/\log n$$

and a careful padding will lead us to our goal of $\mathsf{BPP}/\log\log n$ not in $\mathsf{BPTIME}(n^d)/\log n$. We now make things more concrete.

We assume $\mathsf{EXP} \not\subseteq \mathsf{io\text{-}BPP}$, as otherwise a standard argument, not within our new technique, will lead to proving Theorem 5. Let L be an EXP-complete language decidable in $\mathsf{DTIME}(2^n)$ equipped with an instance checker C for which every query length is at most $c|x|$.

Let $I = \left\{2^k \mid k \in \mathbb{N}\right\}$ and assume w.l.o.g. that L only contains strings whose length is in I. By our assumption, there is no probabilistic polynomial-time TM that solves L on infinitely many $n \in I$.

Define $\hat{t} : \{0,1\}^\star \to \mathbb{N}$ so that $\hat{t}(x)$ is the minimum number m such that there exists a probabilistic TM of size $\log m$ such that $\Pr[C^{M^m}(x) \neq \mathit{quit}] > \frac{2}{3}$. Define $t : I \to \mathbb{N}$ as follows:

$$t(n) \;=\; \frac{1}{n} \max_{x \in \{0,1\}^{n/c}} \hat{t}(x).$$

By the way we defined t, we wanted that both $L \notin \mathsf{BPTIME}(t)$ and our algorithm A decides L in $\mathrm{poly}(n) \cdot \mathrm{poly}(t(c \cdot \frac{n}{c})) = \mathrm{poly}(t(n))$ time.

Indeed, if we take A and change it so that we will go over all TMs of size $\log m$ instead of $\log n$ then on input $x \in \{0,1\}^n$ with high probability it will output $L(x)$ after at most $\mathrm{poly}(t(cn))$ steps. That is, there exists a constant e such that $L \in \mathsf{BPTIME}(T')$ for every time-constructible function T' satisfying $T'(n) \geq t(cn)^e$.

Finally, we have:

Claim 10. *$L \notin \mathsf{BPTIME}(t)/\log t$. Furthermore, this holds for almost all input lengths in I.*

Proof. Assume towards contradiction that $L \in \mathsf{BPTIME}(t)/\log t$. Then, for some $n \in I$, there exists a $\log m$-sized TM such that $\Pr[M^m(x) = L(x)] > \frac{2}{3}$ for every $x \in \{0,1\}^n$, where $m \leq t(n)$. Take $\tilde{M}$ to be the amplified success probability variant of M, so that $\Pr[\tilde{M}^{\tilde{m}}(x) = L(x)] > 1 - 2^{\Omega(n)}$, where $\tilde{m} = \frac{n}{2}m$. By definition of t, there exists $x \in \{0,1\}^{n/c}$ such that $\tilde{t}(x) > \tilde{m}$ so for every

$\log \tilde{m}$-sized TM, and in particular for $\tilde{M}$, the probability that $C^{\tilde{M}^{\tilde{m}}}(x) \neq quit$ is at most $\frac{2}{3}$. This is a contradiction, since $\Pr[C^L(x) \neq quit] = 1$, and by the union bound, the probability that the checker will ask a query x' and get an answer from $\tilde{M}^{\tilde{m}}$ that is different from $L(x')$ is negligible. □

From the above claim, proving Theorem 5 is via a padding argument and we will skip it.

Goldreich, Sudan and Trevisan [3] abstracted out this idea to show that any separation of a time-bounded class with less than logarithmic advice against a class using slightly more advice can be translated to a separation of the first class with 1 bit of advice against the second class with slightly more advice. A corollary of their work gives the state-of-the-art separation of BPP:

Theorem 11. *For any reals* $1 \leq a < b$, $\mathsf{BPTIME}(n^a)/1 \subsetneq \mathsf{BPTIME}(n^b)/1$.

3 Hierarchies for probabilistic classes using a complete problem

Barak also shows that if BPP has a complete problem, then we can get rid of the advice, obtaining $\mathsf{BPTIME}(n^d) \subsetneq \mathsf{BPTIME}(n^{d+1})$.

3.1 The reductions

As we want to do reductions for fixed-polynomial classes, we need our reductions to be more subtle.

Definition 12. *We say that L is* BPTIME*-hard if there exists a constant c such that for any time-constructible function t and any language $L' \in \mathsf{BPTIME}(t)$ there exists a deterministic $t(|x|)^c$-time computable function f such that for any x, $x \in L'$ iff $f(x) \in L$.*

Note that the constant c is independent of L' and t. We say L is BPP-complete if L is BPTIME-*hard* and $L \in \mathsf{BPP}$.

3.2 A little bit about promise problems

Unlike BPP, for PromiseBPP we have a hierarchy, and in particular

$$\mathsf{PromiseBPTIME}(n^d) \subsetneq \mathsf{PromiseBPTIME}(n^{d+1})$$

for every constant d.

Definition 13 (The promise problem CAP)**.** *The promise problem Circuit Acceptance Probability (CAP) is the pair (CAP_Y, CAP_N) where CAP_Y contains all circuits C such that $\Pr_x[C(x) = 1] > \frac{2}{3}$ and CAP_N contains all circuits C such that $\Pr_x[C(x) = 1] < \frac{1}{3}$.*

Clearly, $CAP \in \mathsf{PromiseBPP}$. A "Cook-Levin"-reasoning will show that:

Claim 14. *Let L be a language consistent with CAP (that is, if $x \in CAP_Y$ then $x \in L$ and if $x \in CAP_N$ then $x \notin L$). Then, L is* BPTIME*-hard.*

We thus have the following corollary:

Corollary 15. *If there exists a language L such that L is consistent with CAP and $L \in \mathsf{BPP}$ then there exists a* BPP*-complete language.*

We will talk more about promise problems in the next lecture.

3.3 The hierarchy theorem

We prove:

Theorem 16. *Suppose that* BPP *has a complete problem. Then, there exists a constant* c *such that for every time-constructible* t *it holds that* $\mathsf{BPTIME}(t) \subsetneq \mathsf{BPTIME}(t^c)$.

By Corollary 8, this proves $\mathsf{BPTIME}(n^d) \subsetneq \mathsf{BPTIME}(n^{d+1})$ for every constant $d \geq 1$.

Proof. Let L be a BPP-complete problem and let M_L be its accepting TM that runs in time n^a for some constant a. We know that there exists a constant b such that for every time-constructible function t, every language in $\mathsf{BPTIME}(t)$ is reducible to L using a t^b-time deterministic reduction.

For a string i, let M_i be the i-th deterministic TM. Define the language K such that $x \in K$ iff $M_x^{t(|x|)^b}(x) \notin L$. We have that:

1. $K \in \mathsf{BPTIME}(t^{O(ab)})$.
2. $K \notin \mathsf{BPTIME}(t)$.

Item (1) is true since deciding K can be done by returning $1 - M_L(M_x^{t(|x|)^b}(x))$, and it takes $t(|x|)^{O(ab)}$ time. To prove item (2), assume towards contradiction that $K \in \mathsf{BPTIME}(t)$. L is complete for BPP, so there exists an i such that $i \in K$ iff $M_i^{t(|i|)^b}(i) \in L$. Yet, by the definition of K this happens iff $i \notin K$ and we get a contradiction. □

References

[1] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1(1):3–40, 1991.

[2] Boaz Barak. A probabilistic-time hierarchy theorem for slightly non-uniform algorithms. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 194–208. Springer, 2002.

[3] Oded Goldreich, Madhu Sudan, and Luca Trevisan. From logarithmic advice to single-bit advice. In *Electronic Colloquium on Computational Complexity, TR04-093*, 2004.