Crash Course Complexity Theory — with Emphasis von Randomized Computation

Markus Bläser

Draft — August 4, 2004 and forever

1 Turing machines

We assume that the reader is familiar with the concept of *Turing machines*. Throughout this lecture, we consider multitape Turing machines with a separate read-only two-way input tape. For simplicity, we assume w.l.o.g. that Turing machines have a unique accepting and rejecting state and that the computation stops once the Turing machine accepts or rejects. $\mathsf{DTime}(t(n))$ denotes the class of language that can be decided by a deterministic Turing machine in time $O(t(n))$. $\mathsf{NTime}(t(n))$ denotes the class of languages that can be decided by a nondeterministic Turing machine in time $O(t(n))$. In the same way, $\mathsf{DSpace}(s(n))$ and $\mathsf{NSpace}(s(n))$ denote the classes of languages that can be decided by a deterministic or nondeterministic Turing machine, respectively, that uses $O(s(n))$ cells one the work tapes. Note that we do not count the space on the input tape. In this way, it is possible to speak about sublinear space classes. The language accepted by a Turing machine M is denoted by $L(M)$. Throughout this lecture, we only consider languages over the alphabet $\{0,1\}$. We shorthand the statement that M accepts an input x by $M(x) = 1$ and that M rejects an input x by $M(x) = 0$.

Definition 1.1

1. $\mathsf{P} = \bigcup_{i\in\mathbb{N}} \mathsf{DTime}(n^i)$,
2. $\mathsf{NP} = \bigcup_{i\in\mathbb{N}} \mathsf{NTime}(n^i)$,
3. $\mathsf{E} = \bigcup_{i\in\mathbb{N}} \mathsf{DTime}(2^{in})$,
4. $\mathsf{EXP} = \bigcup_{i\in\mathbb{N}} \mathsf{DTime}(2^{n^i})$,
5. $\mathsf{NEXP} = \bigcup_{i\in\mathbb{N}} \mathsf{NTime}(2^{n^i})$,
6. $\mathsf{SUBEXP} = \bigcap_{\epsilon>0} \mathsf{DTime}(2^{n^\epsilon})$,
7. $\mathsf{L} = \mathsf{DSpace}(\log(n))$
8. $\mathsf{PSPACE} = \bigcup_{i\in\mathbb{N}} \mathsf{DSpace}(n^i)$.

We have

$$L \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP. \tag{1}$$

From the space and time hierachy theorems, it follows that $L \subsetneq PSPACE$ and $P \subsetneq EXP$, but we do not know which of the four inclusions in (1) are strict. Our guess is that all of them are strict. The *space hierachy theorem* basically states the following: If $s_1(n) = o(s_2(n))$, then $DTime(s_1) \subsetneq DTime(s_2)$. Loosely speaking, more space means more power. But this is only true if the space bounds s_1 and s_2 are nice, more precisely, they have to be *space constructible*. That means that there exists a Turing machine that can mark exactly $s_1(n)$ or $s_2(n)$ cells using no more space than $s_1(n)$ or $s_2(n)$, respectively. All the usual functions like polynomials, exponential functions, logarithms, etc. are space constructible. The proof of the space hierachy theorem uses *diagonalization*, one of the rare techniques known for proving lower bounds. The *time hierachy theorem* is somewhat more complicated but things are essentially the same.

For a complexity class C that is defined in terms of Turing machines, we call a Turing machine M that fulfills the requirements of C, in particular $L(M) \in C$, a C-machine for short. Note that $L(M) \in C$ does not necessarily imply that M is a C-machine. M could be inefficient in some way and use more resources than the definition of C allows.

Finally, for a class C, co-C is the class of all languages L such that their complement $\bar{L}$ is in C.

2 Circuits

We will also consider the *non-uniform circuit model*. A (Boolean) circuit C is an acyclic directed graph with exactly one node of outdegree zero. This node is called the output gate. Nodes with indegree zero are called input gates. The number n of input gates is the length of the input. Each other node has either indegree one or two. If it has indegree one, it is labeled with $\neg$ and called a NOT gate. Otherwise, it is labeled with $\vee$ or $\wedge$ and called an OR or AND gate, respectively. Any circuit C accepts a language $L \subseteq \{0,1\}^n$ where n is the number of input gates of C. For a given $x \in \{0,1\}^n$, we assign each gate a Boolean value inductively. The ith input gate gets the value x_i. (Order the input nodes arbitarily.) If all direct predecessors of a gate v have already a value, then the value of v is the Boolean negation, Boolean disjunction or conjunction of the values of its direct predecessors, depending on the type of the gate. The string x is in L, if the output gate evaluates to one, otherwise x is not in L. The *size* of a circuit C is the number of NOT, OR, and AND gates of C. A family of circuits C_n, $n \in \mathbb{N}$ accepts a language $L \subseteq \{0,1\}^*$, if C_n accepts $L \cap \{0,1\}^n$ for all n. In this case, we also write $L = L(C_n)$ for short.

Definition 2.1 *The class* $P/poly$ *is the class of all languages* $L \subseteq \{0,1\}^*$ *such that there is a family of circuits* C_n, $n \in \mathbb{N}$, *and a polynomial* p *with* $L = L(C_n)$ *and* $size(C_n) = O(p(n))$.

Exercise 2.1 *Show that there is a nonrecursive language in* P/poly.

We call a family C_n of size $s(n)$ *uniform*, if there is a $O(\log s(n))$-space bounded Turing machine M that on input n written in unary form on the input tape, outputs a description of C_n on the output tape. Circuits and deterministic Turing machines are polynomially related.

Exercise 2.2 *Prove the following theorem: For any uniform family of circuits C_n of size $s(n)$, there is a deterministic Turing machine M with running time bounded by $s(n)^{O(1)}$ with $L(C_n) = L(M)$.*

Exercise 2.3 *Prove the following theorem: For any deterministic Turing machine M with running time bounded by $t(n)$, there is a family of circuits C_n of size $t(n)^{O(1)}$ with $L(C_n) = L(M)$. This family is even uniform.*

3 Randomized complexity classes

Probabilistic Turing machines have an additional *random tape*. On this tape, the Turing machine gets an one-sided infinite bit string y. The random tape is read-only and one-way.

Right at the moment, we are considering the random string y as an additional input. The name random string is justified by the following definition: A probabilistic Turing machine accepts an input x with *acceptance probability* at least p if $\Pr[M(x, y) = 1] \geq p$. Here the probability is taken over all choices of y. We define the *rejection probability* in the same way. The running time $t(n)$ of a probabilistic Turing machine M is the maximum number of steps, M performs on any input of length n and and any random string y. Note that if $t(n)$ is bounded, then we can consider y to be a finite string of length at most $t(n)$. The maximum number of random bits a Turing maching reads on any input x and random string y is called the amount of randomness used by the machine.

We define $\mathsf{RTime}(t(n))$ to be the class of all languages L such that there is a Turing machine M with running time $O(t(n))$ and for all $x \in L$, M accepts x with probability at least $1/2$ and for all $x \notin L$, M rejects L with probability 1. Such an M is said to have a *one-sided error*. If M in fact accepts each $x \in L$ with probability $\geq 1 - \epsilon \geq 1/2$, then we say that the error probability of M is bounded by ϵ.

The class $\mathsf{BPTime}(t(n))$ is defined in the same manner, but we allow the Turing machine M to err in two ways. We require that for all $x \in L$, M accepts x with probability at least $2/3$ and for all $x \notin L$, M rejects with probability at least $2/3$ (that is, accepts with probability at most $1/3$). Such an error is called a *two-sided error*. If M actually accepts all $x \in L$ with probability $\geq 1 - \epsilon$ and rejects each $x \in L$ with probability $\leq \epsilon$, then we say that the error probability is bounded by ϵ.

Definition 3.1 *1.* $\mathsf{RP} = \bigcup_{i \in \mathbb{N}} \mathsf{RTime}(n^i)$,

2. $\mathsf{BPP} = \bigcup_{i \in \mathbb{N}} \mathsf{BPTime}(n^i)$,

3. $\mathsf{ZPP} = \mathsf{RP} \cap \text{co-RP}$.

The name ZPP stands for zero error probabilistic polynomial time. It is justified by the following statement.

Exercise 3.1 *A language* L *is in* ZPP *if and only if* L *is accepted by a probabilistic Turing machine with error probability zero and expected polynomial running time. Here the expectation is taken over all possible random strings on the random tape.*

For robust classes (such as RP and BPP) the choice of the constants $1/2$ and $2/3$ in the definitions of RTime and BPTime is fairly arbitrary, since both classes allow *probability amplification.*

Lemma 3.2 *Let* M *be a Turing machine for some language* $L \in \mathsf{RP}$ *that runs in time* $t(n)$*, uses* $r(n)$ *random bits, and has error probability* ϵ*. For any* $k \in \mathbb{N}$*, there is a Turing machine* M' *for* L *that runs in time* $O(kt(n))$*, uses* $kr(n)$ *random bits, and has error probability* ϵ^k*.*

Proof. M' simulates M k times, each time using new random bits. M' accepts, if M accepts at least once. Otherwise, M' rejects. The bounds on the time and randomness are obvious.

If $x \notin L$, then M' also rejects, since M does not err on x.

If $x \in L$, then with probability at most ϵ, M rejects x. Since M' performs k independent trials, the probability that M' rejects x is at most ϵ^k. ■

Lemma 3.3 *Let* M *be a Turing machine for some language* $L \in \mathsf{BPP}$ *that runs in time* $t(n)$*, uses* $r(n)$ *random bits, and has error probability* $\epsilon < 1/2$*. For any* $k \in \mathbb{N}$*, there is a Turing machine* M' *for* L *that runs in time* $O(kt(n))$*, uses* $kr(n)$ *random bits, and has error probability* $2^{-c_\epsilon k}$ *for some constant* c_ϵ *that solely depends on* ϵ*.*

Proof. M' simulates M k times, each time with fresh random bits. M' accepts, if M accepted at least as many as times as it rejected the input. Otherwise, M' rejects. Let μ be the expected number of times that a simulated run of M accepts.

If $x \in L$, then $\mu \geq (1-\epsilon)k$. The probability that less than half of the simulated runs of M accept is $< e^{-\frac{(1-\epsilon)\delta^2}{2}k}$ with $\delta = 1 - \frac{1}{2(1-\epsilon)}$ by the Chernoff bound (see below). The case $x \notin L$ is treated similarly. In both cases, the error probability is bounded by 2^{ck} for some constant c only depending on ϵ. ■

In the proof above, we used the so-called *Chernoff bound.* A proof of it can be found in the book by Motwani and Raghavan or in most books on probability theory.

Lemma 3.4 (Chernoff bound) *Let $X_1, \ldots, X_m$ be independent 0–1 valued random variables and let $X = X_1 + \cdots + X_m$. Let $\mu = E(x)$. Then for any $\delta > 0$,*

$$\Pr[X > (1+\delta)\mu] < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \quad \textit{and} \quad \Pr[X < (1-\delta)\mu] < e^{-\frac{\mu\delta^2}{2}}.$$

We have

$$P \subseteq ZPP \subseteq \begin{matrix} RP \\ \text{co-RP} \end{matrix} \subseteq BPP. \tag{2}$$

The latter two inclusion follow by amplifying the acceptance probability once.

Finally, we define *probabilistic circuits*: They have an additional number of indegree zero gates, so-called random gates. Beside the input x, the circuit gets an additional bit string y via the random gates. (Note that random gates and choice gates is basically the same construction. Only the acceptance condition is different.) A circuit C is a bounded two-sided error probabilistic circuit if $\Pr_{y\in\{0,1\}^m}[C(x,y) = 1] \geq 2/3$. or $\Pr_{y\in\{0,1\}^m}[C(x,y) = 1] \leq 1/3$. (Here, m is the number of random gates.) In the first case, C is said to accept x, in the second case, it rejects x.

C is a bounded one-sided error probabilistic circuit, if $\Pr_{y\in\{0,1\}^m}[C(x,y) = 1] \geq 1/2$. or $\Pr_{y\in\{0,1\}^m}[C(x,y) = 1] = 0$. In the first case, C is said to accept x, in the second case, it rejects x.

Exercise 3.2 *Prove the following theorem: For any probabilistic Turing machine M with running time bounded by $t(n)$, there is a family of probabilistic circuits C_n of size $t(n)^{O(1)}$ such that for any input x, the acceptance probability of C_n on x is the same as the one of M on x. This family is even uniform.*

4 Promise Problems

To show that $P = NP$, one possible strategy is "simply" to show $L \in P$ for some NP-complete language L. Is there a similar stragegy for settling whether $P = BPP$? The roadblock is that we do not know whether BPP and RP have any complete problems. Even such a generic problem as the set of all triples $\langle M, x, 1^t\rangle$ such that M is a BPP-machine that accepts x within t steps is not BPP-complete. It is BPP-hard, but not in BPP, since it is even undecidable whether the error probability of M is bounded by $1/3$. Being a BPP-machine is a *semantic* property, while being an NP-machine is a *syntactic* one.

There is an easy "solution" to this rather annoying problem: Instead of only considering languages we also study *partial languages*: A partial languages is a pair of languages $L \subseteq U$. A Turing machine M accepts such a partial language, if it accepts all input in L and rejects all inputs in $U \setminus L$. We do not care about the behaviour of M on inputs in $\Sigma^* \setminus U$. Informally, we give M the promise that it will only get inputs from U. Therefore, partial languages are often called

promise problems. The trick is that we can choose any language, even a nonrecursive one for U. For instance, in the above generic Turing machine simulation problem, we would choose U to be the set of all $\langle M, x, 1^t \rangle$ such that M is a probabilistic Turing machine that has error probability $\leq 1/3$. In this way, we overcome the problem that it is not decidable whether a Turing machine has error probability $\leq 1/3$. We simply do not care what the simulating machine does if M does not have error probability bounded by $1/3$.

For any complexity class C, we can define a corresponding promise class prC in the obvious way. For classes that are defined in terms of syntactic properties, like P or NP, it does not make a real difference whether we consider promise problems or not. For instance, the statements $P = NP$ and $prP = prNP$ are equivalent. For classes defined by semantic properties, like RP and BPP, promise version are much easier to treat than the original classes. The additional set U gives the promise classes complete problems.

The following two problems are complete for BPP and RP, respectively.

Definition 4.1 *1. Circuit acceptance probability estimation* CAPE*: Given a circuit C with the promise that either $\Pr_{x \in \{0,1\}^n}[C(x) = 1] \leq 1/3$ or $\Pr_{x \in \{0,1\}^n}[C(x) = 1] \geq 2/3$, decide which of the two properties is fulfilled by C. (Here n is the length of the input.)*

2. One-sided acceptance probability estimation CAPE_1*: Given a circuit C with the promise that either $\Pr_{x \in \{0,1\}^n}[C(x) = 1] = 0$ or $\Pr_{x \in \{0,1\}^n}[C(x) = 1] \geq 1/2$, decide which of the two properties is fulfilled by C.*

Lemma 4.2 *1.* CAPE *is* prBPP*-complete (under logarithmic space many-one reductions).*

2. CAPE_1 *is* prRP*-complete (under logarithmic space many-one reductions).*

Proof. First, CAPE is in prBPP. We simply pick an x at random and compute $C(x)$. We accept C iff $C(x) = 1$. The promise on C ensures that the error probability is $\leq 1/3$. The same arguments show that $\text{CAPE}_1 \in \text{prRP}$.

Next, we show that CAPE is also hard for prBPP. Let $L \subseteq U$ be in prBPP and let M be a prBPP-machine for $L \subseteq U$. Let t be the running time of M. By Exercise 2.3, we can construct in space $O(\log t(n))$ a randomized circuit C_n such that the acceptance probability of C_n on input x is the same as M on input x

Now given an input z for M we construct a circuit C_z as follows. We first construct the circuit C_n as above, which has two inputs: the string x and the random string y. Then we fix the entries of x to be the bits of z. This gives the circuit C_z. C_z computes the functon $y \to C(z, y)$. If M accepts $z \in U$, then the acceptance probability of C_z is at least $2/3$. If M rejects $z \in U$, then the rejection probility of C_n is at least $2/3$. Thus the reduction transforms yes-instances to yes-instances and no-instances to no-instances. It is computable in logarithmic space, since t is polynomial.

The proof for $\mathtt{CAPE}_1$ is essentially the same. ■

Exercise 4.1 *Let* C_1 *and* C_2 *be complexity classes. Show that* $\mathsf{prC}_1 = \mathsf{prC}_2$ *implies* $\mathsf{C}_1 = \mathsf{C}_2$*? What about the converse?*

Note that for complexity classes defined by syntactic properties, it usually does not matter whether we also consider promise problems or not. In particular, we have the following result.

Exercise 4.2 *Show the following:* $\mathsf{prP} = \mathsf{prNP}$ *if and only if* $\mathsf{P} = \mathsf{NP}$.

5 Some easy derandomization results

We start with comparing RP and BPP with the complexity classes defined in Definitions 1.1 and 2.1. Since these are non-randomized complexity classes, one can view these results as a kind of derandomization.

Theorem 5.1 $\mathsf{BPP} \subseteq \mathsf{PSPACE}$.

Proof. Let M be a BPP-machine for some $L \in \mathsf{BPP}$. Assume that M reads at most $r(n)$ random bits on inputs of length n. Turing machine M' simulates M as follows: M' systematically lists all bit strings of length $r(n)$ and simulates M with the current string as random string. M' counts how often M accepts and rejects. If the number of accepting computations exceeds the number of rejecting computations, M' accepts. Otherwise, M' rejects. Since M is polynomial time, $r(n)$ is bounded by a polynomial. Hence M' uses only polynomial space. ■

Corollary 5.2 $\mathsf{BPP} \subseteq \mathsf{EXP}$.

Theorem 5.3 $\mathsf{RP} \subseteq \mathsf{NP}$.

Proof. Let M be an RP-machine for some $L \in \mathsf{RP}$. We convert M into an NP-machine M' as follows. Whenever M would read a bit from the random tape, M' nondeterministically branches to the two states that M would enter after reading zero or one, respectively. If M does not accept x, then there is no random string such that M on input x reaches an accepting configuration. Thus there is no accepting path in the computation tree of M' either.

On the other hand, if M accepts x, then M reaches an accepting configuration on at least half of the random strings. Thus at least half of the paths in the computation tree of M' are accepting ones. In particular, there is at least one accepting path. Hence M' accepts x. ■

Next we turn to the relation between BPP and circuits. The class P/poly can be viewed as the languages accepted by polynomial time deterministic Turing machines with polynomial *advice*. Such a Turing machine has an additional read-only advice tape.

Definition 5.4 *Let* t *and* a *be two functions* $\mathbb{N} \to \mathbb{N}$. *A language* L *is in the class* $\mathsf{DTime}(t)/a$ *if there is a deterministic Turing machine* M *with running time* $O(t)$ *and with an additional advice tape and a sequence of strings* $\alpha(n) \in \{0,1\}^{a(n)}$ *such that the following holds: For all* $x \in L$, M *accepts* x *with* $\alpha(|x|)$ *written on the advice tape. For all* $x \notin L$, M *rejects* x *with* $\alpha(|x|)$ *written on the advice tape.*

This definition extends to nondeterministic classes and space classes in the obvious way. We can also extend the definition to sets of functions T and A. We define $\mathsf{DTime}(T)/A = \bigcup_{t \in T, a \in A} \mathsf{DTime}(t)/a$. If we choose T and A both to be the class of all polynomials, then we get exactly $\mathsf{P/poly}$.

For each input length n, we give the Turing machine an advice $\alpha(n)$. Note that we do no restrict this sequence, except for the length. In particular, the sequence need not be to computable at all.

Lemma 5.5 *For all languages* L, *if there is a polynomial time Turing machine* M *with polynomial advice accepting* L, *then* $L \in \mathsf{P/poly}$.

Proof. For the moment, let us view the advice $\alpha(n)$ as a part of the input, i.e., M gets $\alpha(|x|)$ concatenated with its regular input x. By Exercise 2.3, for each n, there is a circuit C_n such that $C_n(x, \alpha(n)) = M(x, \alpha(n))$ for all x of length n. Let C'_n be the sequence of circuits obtained from C_n by fixing the second part of the input to $\alpha(n)$. This gives a sequence of polynomial size circuits such that $C'_n(x) = C_n(x, \alpha(n)) = M(x, \alpha(n))$ for all x of length n. Thus $L \in \mathsf{P/poly}$. ■

Exercise 5.1 *Show that the converse of Lemma 5.5 holds, too.*

Theorem 5.6 (Adleman 1978) $\mathsf{BPP} \subseteq \mathsf{P/poly}$.

Proof. Let $L \in \mathsf{BPP}$. By Lemma 3.3, there is a BPP-Machine with error probability $< 2^{-n}$ that accepts L. There are 2^n possible input strings of length n. Since for each string x of length n, the error probability of M is $< 2^{-n}$, M can err on x only for a fraction of all possible random strings that is smaller than 2^{-n}. Thus there must be one random string that is good for all inputs of length n. We take this string as an advice string for the inputs of length n. By Lemma 5.5, $L \in \mathsf{P/poly}$. ■

How do we find this good random string? If we amplify the error probability even further, say to 2^{-2n}, then almost all, namely a fraction of $1 - 2^{-n}$ random strings are good. Thus picking the advice at random is a good strategy. (This, however, requires randomness!)