Advanced Complexity Exam 2018

All written documents allowed. No Internet access, no cell phone.

1 The Zachos-Heller theorem

Let $\Sigma = \{0, 1\}$. All our random tapes r, r_1, r_2, $\dots$, are strings over Σ.

Take $L \in \mathbf{BPP}$, so that :

— if $x \in L$ then $Pr_r[\mathcal{M}(x, r) \text{ accepts}] \geq 1 - 1/2^n$,

— if $x \notin L$ then $Pr_r[\mathcal{M}(x, r) \text{ accepts}] \leq 1/2^n$,

where $\mathcal{M}$ is a deterministic Turing machine working in polynomial time $p(n)$, and using $q(n)$ random bits (meaning that the size of r is $q(n)$).

For a bit $b \in \Sigma$, say that $\mathcal{M}(x, r) = b$ to abbreviate « either $b = 1$ and $\mathcal{M}(x, r)$ accepts, or $b = 0$ and $\mathcal{M}(x, r)$ rejects ». $\mathcal{M}(x, r) \neq b$ is the negation of $\mathcal{M}(x, r) = b$. Let R_{xb} be the set of those r such that $\mathcal{M}(x, r) \neq b$.

Let also $b = (x \in L)$ mean « either $b = 1$ and $x \in L$, or $b = 0$ and $x \notin L$ », and $b \neq (x \in L)$ be its negation.

1. Let L' be the language of all tuples (x, b, H) such that R_{xb} has a collision for H, where $b \in \Sigma$ and $H = (h_1, \cdots, h_\ell)$ is a tuple of linear hash functions from $\Sigma^{q(n)}$ to $\Sigma^{m'}$, and where ℓ and m' are polynomials in the size n of x, $\ell \geq m'$, to be determined later. Show that L' is in $\mathbf{NP}$.
2. We define the following algorithm. On input x, we draw b and H (as described above) at random, uniformly and independently. Then we test whether $(x, b, H) \in L'$. If so, we return the special symbol fail, otherwise we return b.
   (a) Show that if $b \neq (x \in L)$, then that algorithm must return fail... under a constraint on n, ℓ, and m' that you will give explicitly. We will name that constraint (A).
   (b) Show that if $b = (x \in L)$, then the probability that the algorithm returns fail is smaller than or equal to $1/2^{\ell - m' + 1}$ ... under a constraint on n, ℓ, and m' that you will give explicitly. We will name that constraint (B).
   (c) We simply take $\ell = m'$. Show that, for n large enough, one can find m' so that (A) and (B) are satisfied, and such that m' is bounded by a polynomial in n.

3. Conclude that **BPP** is included in the class $\mathbf{ZPP}^{\mathbf{NP}}$ of languages that can be decided in expected polynomial time with zero error, on a randomized Turing machine with access to an **NP** oracle. This is the *Zachos-Heller theorem.*
4. Why is $\mathbf{ZPP}^{\mathbf{NP}}$ equal to $\mathbf{RP}^{\mathbf{NP}} \cap \mathbf{coRP}^{\mathbf{NP}}$? A brief answer is enough. The classes $\mathbf{RP}^{\mathbf{NP}}$ and $\mathbf{coRP}^{\mathbf{NP}}$ are defined just like **RP** and **coRP**, except the Turing machine has access to an oracle deciding some language in **NP**.
5. Show that $\mathbf{RP}^{\mathbf{NP}} \subseteq \Sigma_2^p$.
6. Deduce a new proof of the Sipser-Gács-Lautemann theorem $\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$.

2 L/poly, branching programs, and BP · L

For a function $f\colon \mathbb{N} \to \mathbb{N}$, a language L is in the class $\mathbf{L}/f$ if and only if there is a family of so-called *advice words* $(adv_n)_{n\in\mathbb{N}}$, where adv_n is of size $O(f(n))$ (and is not necessarily computable), and a logarithmic space deterministic Turing machine $\mathcal{M}$, such that for every input x of size n, $x \in L$ if and only if $\mathcal{M}(x, adv_n)$ accepts. Note that $\mathcal{M}$ works in space $O(\log n)$, where n is the size of x, not counting the size of adv_n.

As usual, by space we mean the size used by the work tapes, and ignore all other tapes, notably the read-only input tape x and the read-only advice tape.

L/poly is the union of the classes $\mathbf{L}/f$ when f ranges over the polynomials with coefficients in $\mathbb{N}$. We use that every input x is given in binary.

A *branching program* (for short, *BP*) π is just like a circuit, except that its gates are built from the `if` x_i `then _ else _` connective instead of $\wedge$, $\vee$, $\neg$; the notation x_i specifies bit i of the input x. Additionally, the two wires 0 and 1 specify false (rejection) and true (acceptance) respectively. Formally, a *net-list* for π is a list of wire specifications of the form :

$$m\colon \texttt{if } x_i \texttt{ then } j \texttt{ else } k$$

where $m > j, k, 1$ (m, j and k are wire numbers), and where consecutive wire specifications have values of m that increase by exactly 1, and start at 2. For example, the following branching program computes (at its last specified wire, number 4) $(x_3 \wedge \neg x_5) \vee (\neg x_3 \wedge x_2)$:

$$2\colon \texttt{if } x_5 \texttt{ then } 0 \texttt{ else } 1$$
$$3\colon \texttt{if } x_2 \texttt{ then } 1 \texttt{ else } 0$$
$$4\colon \texttt{if } x_3 \texttt{ then } 2 \texttt{ else } 3$$

A BP π is of *length* n if it can take inputs of size n, namely if every x_i in π is such that $0 \leq i < n$. The *size* of π is its size as a net-list, where x_i is given by writing i in binary. Wire numbers are also written in binary.

We say that a BP *accepts* its input x if and only if the value of its final wire, evaluating each x_i as bit i of x, is 1. A language L *has polynomial BPs* if and only if, for every $n \in \mathbb{N}$, there is a length n branching program π_n of polynomal size $p(n)$ such that for every input x of size n, $x \in L$ if and only if π_n accepts x.

7. Show that every language L that has polynomial BPs is in $\mathbf{L/poly}$. Be careful about the size of the work tapes your Turing machine uses.
8. Conversely, show that every language L in $\mathbf{L/poly}$ has polynomial BPs. Hint : given a logspace Turing machine $\mathcal{M}$ with polynomial advice, some form of the configuration graph of $\mathcal{M}$ on inputs of size n has polynomial size in $n\dots$ and you need polynomially many wires. You may assume that $\mathcal{M}$ has only one work tape, and always terminates.

The class $\mathbf{BP}\cdot\mathbf{L}$ is defined as the class of languages L such that there is a deterministic Turing machine $\mathcal{M}$ such that if $x \in L$ then $Pr_r[\mathcal{M}(x,r) \text{ accepts}] \geq 2/3$, and otherwise $Pr_r[\mathcal{M}(x,r) \text{ accepts}] \leq 1/3$—and such that $\mathcal{M}(x,r)$ works in space $k \log n$, where n is the size of x, for some constant k that (notably) does not depend on r.

9. Let $L \in \mathbf{BP}\cdot\mathbf{L}$. Let n denote the size of x. Why can we assume r to be of size polynomial in n ?
10. Show that $\mathbf{BP}\cdot\mathbf{L}$ admits error reduction : for every language in L, for every polynomial q, there is a deterministic Turing machine $\mathcal{M}$ working in space $O(\log n)$ (independently of the size of r) such that if $x \in L$ then $Pr_r[\mathcal{M}(x,r) \text{ accepts}] \geq 1 - 1/2^{q(n)}$, and otherwise $Pr_r[\mathcal{M}(x,r) \text{ accepts}] \leq 1/2^{q(n)}$.
11. Show that every language of $\mathbf{BP}\cdot\mathbf{L}$ has polynomial branching programs.

Branching programs are a relaxed form of *binary decision diagrams* (BDD), a fundamental data structure used in symbolic model-checking. A BDD on an n-bit input x has exponential size in the worst case, and that worst case is attained often, even in practice. The above delineates when polynomial size is achievable.

3 PCP

A (R,Q,T)*-restricted verifier* is a randomized Turing machine with direct access to a proof tape that works in three phases :

— it computes $Q(n)$ positions on the proof tape, in polynomial time, while accessing the input tape x and the random tape r only (not the proof tape) ; the random tape contains only $R(n)$ bits ;

— it reads the bits on the proof tape y at these positions ;

— using x, r, and the bits just read from y, it decides to accept or reject in time $T(n)$ (in this phase, the machine cannot access the proof tape).

The class $\mathbf{PCP}(R,Q,T)$ is the class of languages L such that there is an (R,Q,T)-restricted verifier V such that :

— if $x \in L$, then there is a proof y such that $\Pr_r[V(x,y,r) \text{ rejects}] = 0$;

— if $x \notin L$, then for every proof y, $\Pr_r[V(x,y,r) \text{ accepts}] \leq 1/2$.

We do not require any particular bound on the size of y.

12. Show that graph non-isomorphism is in $\mathbf{PCP}(O(n \log n), 1, O(1))$. You should of course get some inspiration from one of the algorithms we gave in the lectures for that problem.