

Advanced Complexity Theory

Markus Bläser & Bodo Manthey
Universität des Saarlandes

Draft—February 9, 2011 and forever

1 Complexity of optimization problems

1.1 Optimization problems

The study of the complexity of solving optimization problems is an important practical aspect of complexity theory. A good textbook on this topic is the one by Ausiello et al. [ACG⁺99]. The book by Vazirani [Vaz01] is also recommend, but its focus is on the algorithmic side.

Definition 1.1. An optimization problem P is a 4-tuple $(I_P, S_P, m_P, \text{goal}_P)$ where

1. $I_P \subseteq \{0, 1\}^*$ is the set of valid instances of P ,
2. S_P is a function that assigns to each valid instance x the set of feasible solutions $S_P(x)$ of x , which is a subset of $\{0, 1\}^*$.¹
3. $m_P : \{(x, y) \mid x \in I_P \text{ and } y \in S_P(x)\} \rightarrow \mathbb{N}^+$ is the objective function or measure function. $m_P(x, y)$ is the objective value of the feasible solution y (with respect to x).
4. $\text{goal}_P \in \{\min, \max\}$ specifies the type of the optimization problem. Either it is a minimization or a maximization problem.

When the context is clear, we will drop the subscript P . Formally, an optimization problem is defined over the alphabet $\{0, 1\}$. But as usual, when we talk about concrete problems, we want to talk about graphs, nodes, weights, etc. In this case, we tacitly assume that we can always find suitable encodings of the objects we talk about.

Given an instance x of the optimization problem P , we denote by $S_P^*(x)$ the set of all optimal solutions, that is, the set of all $y \in S_P(x)$ such that

$$m_P(x, y) = \text{goal}\{m_P(x, z) \mid z \in S_P(x)\}.$$

(Note that the set of optimal solutions could be empty, since the maximum need not exist. The minimum always exists, since we m_P only attains values in \mathbb{N}^+ . In the following we will assume that there are always optimal

¹Some authors also assume that for all $x \in I_P$, $S_P(x) \neq \emptyset$. In this case the class NPO defined in the next section would be equal to the class exp-APX (defined somewhere else).

solutions provided that $S_P(x) \neq \emptyset$.) The objective value of any $y \in S^*(x)$ is denoted by $\text{OPT}_P(x)$.²

Given an optimization problem P , there are (at least) three things one could do given a valid instance x :

1. compute an optimal solution $y \in S^*(x)$ (*construction problem*).
2. compute $\text{OPT}(x)$ (*evaluation problem*)
3. given an additional bound B , decide whether $\text{OPT}(x) \geq B$ (if goal = max) or whether $\text{OPT}(x) \leq B$ (if goal = min) (*decision problem*).

The first task seems to be the most natural one. Its precise formalization is however a subtle task. One could compute the function $F : I \rightarrow \mathcal{P}(\{0, 1\}^*)$ mapping each x to its set of optimal solutions $S^*(x)$. However $S^*(x)$ could be very large (or even infinite). Moreover, one is almost always content with only one optimal solution. A *cut* of F is any function $f : I \rightarrow \{0, 1\}^*$ that maps every x to some $y \in S^*(x)$. We say that we solve the construction problem associated with P if there is a cut of F that we can compute efficiently.³ It turns out to be very useful to call such a cut again P and assume that positive statements containing P are implicitly \exists -quantified and negative statements are \forall -quantified. (Do not worry too much now, everything will become clear.)

The second task is easy to model. We want to compute the function $x \mapsto \text{OPT}(x)$. We denote this function by P_{eval} .

The third task can be modelled as a decision problem. Let

$$P_{\text{dec}} = \begin{cases} \{\langle x, \text{bin}(B) \rangle \mid \text{OPT}(x) \geq B\} & \text{if goal} = \text{max} \\ \{\langle x, \text{bin}(B) \rangle \mid \text{OPT}(x) \leq B\} & \text{if goal} = \text{min} \end{cases}$$

Our task is now to decide membership in P_{dec} .

1.2 PO and NPO

We now define optimization analogs of P and NP.

Definition 1.2. *NPO is the class of all optimization problems $P = (I, S, m, \text{goal})$ such that*

²The name $m_P^*(x)$ would be more consequent, but $\text{OPT}_P(x)$ is so intuitive and convenient.

³Note that not every cut can be computable, even for very simple optimization problems like computing minimum spanning trees and even on very simple instances. Consider a complete graph K_n , all edges with weight one. Then every spanning tree is optimal. But a cut that maps K_n to a line if the n th Turing machine halts on the empty word and to a star otherwise is certainly not computable.

1. $I \in P$, i.e., we can decide in polynomial time whether a given x is a valid instance,
2. there is a polynomial p such that for all $x \in I$ and $y \in S(x)$, $|y| \leq p(|x|)$, and for all y with $|y| \leq p(|x|)$, we can decide $y \in S(x)$ in time polynomial in $|x|$,
3. m is computable in polynomial time.

Definition 1.3. PO is the class of all optimization problems $P \in NPO$ such that the construction problem P is deterministically polynomial time computable. (Recall that this means that there is a cut that is polynomial time computable).

We will see the relation of PO and NPO to P and NP in Section 1.5. Even though it is not explicit in Definition 1.2, NPO is a nondeterministic complexity class.

Theorem 1.4. For each $P \in NPO$, $P_{\text{dec}} \in NP$.

Proof. Let p be the polynomial in the definition of NPO. The following nondeterministic Turing machine M decides P_{dec} in polynomial time:

Input: instance $x \in I$ and bound B

1. M guesses a string y with $|y| \leq p(|x|)$.
2. M deterministically tests whether $y \in S(x)$.
If not, M rejects.
3. If y is, then M computes $m(x, y)$ and tests whether $m(x, y) \leq B$ (minimization problem) or $m(x, y) \geq B$ (maximization problem).
4. If the test is positive, then M accepts, otherwise, M rejects.

It is easy to see that M indeed decides P_{dec} and that its running time is polynomial. ■

1.3 Example: TSP

Problem 1.5 (TSP, Δ -TSP). The Traveling Salesperson Problem (TSP) is defined as follows: Given a complete loopless (undirected) graph $G = (V, E)$ and a weight function $w : E \rightarrow \mathbb{N}^+$ assigning each edge a positive weight, find a Hamiltonian tour of minimum weight. If in addition w fulfills the triangle inequality, i.e.,

$$w(\{u, v\}) \leq w(\{u, x\}) + w(\{x, v\}) \quad \text{for all nodes } u, x, v,$$

then we speak of the Metric Traveling Salesperson Problem (Δ -TSP).

In the example of the Traveling Salesperson Problem TSP, we have the following:

- The set of all valid instances is the set (of suitable encodings) of all edge-weighted complete loopless graphs G . In the special case Δ -TSP, the edge weights should also fulfill the triangle inequality (which can be easily checked).
- Given an instance x , a feasible solution is any Hamiltonian tour of G , i.e., a permutation of the vertices of G . (Note that for TSP, the set of feasible solutions only depends on the number of nodes of G .)
- The objective value of a solution y of an instance x is the sum of the edges used in the tour specified by y . (This can be interpreted as the length of the tour.)
- Finally, TSP and Δ -TSP are minimization problems.

It is easy to verify, that $\text{TSP} \in \text{NPO}$. However it is very unlikely that it is in PO . Even finding a very rough approximate solution seems to be very hard.

Exercise 1.1. *Assume that there is a polynomial time algorithm that given an instance x of TSP, returns a Hamiltonian tour whose weight is at most $2^{p(n)} \cdot \text{OPT}$ for some polynomial p , where n is the number of nodes of the given graph. Then $\text{P} = \text{NP}$. (Hint: Show that under this assumption, one can decide whether a graph has a Hamiltonian circuit.)*

1.4 Construction, evaluation, and decision

Let us investigate the relation between the construction, evaluation, and decision problem associated with a problem $P \in \text{NPO}$.

Theorem 1.6. *Let $P \in \text{NPO}$. Then*

1. $P_{\text{dec}} \leq_{\text{P}}^{\text{T}} P_{\text{eval}}$ and $P_{\text{eval}} \leq_{\text{P}}^{\text{T}} P_{\text{dec}}$.
2. $P_{\text{eval}} \leq_{\text{P}}^{\text{T}} P$. (Since this is a negative statement about P , it means that $P_{\text{eval}} \leq_{\text{P}}^{\text{T}} P$ holds for all cuts P .)

Proof. We start with the first statement: $P_{\text{dec}} \leq_{\text{P}}^{\text{T}} P_{\text{eval}}$ is seen easily: On input $\langle x, \text{bin}(B) \rangle$, we can compute $\text{OPT}(x)$ using the oracle P_{eval} , and compare it with B .

$P_{\text{eval}} \leq_{\text{P}}^{\text{T}} P_{\text{dec}}$ is a little trickier: Since m is polynomial time computable, $\text{OPT}(x) \leq 2^{q(|x|)}$ for some polynomial q . Using binary search, we can find $\text{OPT}(x)$ with $q(n)$ oracle queries.

For the second statement, note that when we have an optimum solution, then we can compute $\text{OPT}(x)$. ■

If P_{dec} is NP-complete, then the optimization problem is not harder than the decision problem.

Theorem 1.7. *Let $P \in \text{NPO}$ such that P_{dec} is NP-complete. Then $P \leq_{\text{P}}^{\text{T}} P_{\text{dec}}$.*

Proof. Assume that P is a maximization problem, the minimization case is symmetric. Let q be a polynomial such for every $x \in \text{I}$ and $y \in \text{S}(x)$, $|y| \leq q(|x|)$ and $m(x, y)$ is bounded by $2^{q(|x|)}$.

For given x , fix some polynomial time computable total order on the set $\{0, 1\}^{\leq q(|x|)}$. For $y \in \{0, 1\}^{\leq q(|x|)}$, let $\lambda(y)$ be the rank that y has with respect to this order.

We derive a new problem \hat{P} from P by defining a new objective function. The objective function of \hat{P} is given by

$$\hat{m}(x, y) = 2^{q(n)+1} m(x, y) + \lambda(y).$$

Note that the first summand is always bigger than $2^{q(n)+1} > \lambda(y)$. This implies that for all x and $y_1, y_2 \in \text{S}(x)$, $\hat{m}(x, y_1) \neq \hat{m}(x, y_2)$. Furthermore, if $\hat{m}(x, y_1) \geq \hat{m}(x, y_2)$ then $m(x, y_1) \geq m(x, y_2)$. Thus if $y \in \hat{\text{S}}^*(x)$ then $y \in \text{S}^*(x)$, too. (Here $\hat{\text{S}}^*(x)$ is the set of optimum solutions of x as an instance of \hat{P} .)

An optimal solution y of $\hat{\text{S}}^*(x)$ can be easily derived from $\hat{\text{OPT}}(x)$: We compute the remainder of the division $\hat{\text{OPT}}(x)$ with $2^{q(n)+1}$. This remainder is $\lambda(y)$ from which we can obtain y . Thus $P \leq_{\text{P}}^{\text{T}} \hat{P} \leq_{\text{P}}^{\text{T}} \hat{P}_{\text{eval}}$.

By Theorem 1.6, $\hat{P}_{\text{eval}} \leq_{\text{P}}^{\text{T}} \hat{P}_{\text{dec}}$. Since $\hat{P}_{\text{dec}} \in \text{NP}$ and P_{dec} is NP-complete by assumption, $\hat{P}_{\text{dec}} \leq_{\text{P}} P_{\text{dec}}$. Using transitivity, we get $P \leq_{\text{P}}^{\text{T}} P_{\text{dec}}$. ■

1.5 NP-hard optimization problems

Definition 1.8. *An optimization problem P is NP-hard if for all $L \in \text{NP}$, $L \leq_{\text{P}}^{\text{T}} P$.*

Theorem 1.9. *If P is NP-hard and $P \in \text{PO}$, then $\text{P} = \text{NP}$.*

Exercise 1.2. *Prove Theorem 1.9.*

Theorem 1.10. *Let $P \in \text{NPO}$. If P_{dec} is NP-hard, then P is NP-hard.*

Proof. Since P_{dec} is NP-hard, $L \leq_{\text{P}} P_{\text{dec}}$ for all $L \in \text{NP}$. Since many-one reducibility is a special case of Turing reducibility and $\leq_{\text{P}}^{\text{T}}$ is transitive, we get $L \leq_{\text{P}}^{\text{T}} P$. ■

Some authors prefer to call an optimization problem NP-hard if P_{dec} is NP-hard. Theorem 1.10 states that this definition is potentially more restrictive than our definition.

Corollary 1.11. *If $P \neq NP$, then $PO \neq NPO$*

Proof. There is a problem P in NPO such that P_{dec} is NP-hard, for instance Δ -TSP. If P would belong to PO, then also $P_{\text{dec}} \in P$ by Theorem 1.7, a contradiction. ■

2 Approximation algorithms and approximation classes

In the most general sense, an approximation algorithm is an algorithm that given a valid instance x is able to compute some feasible solution.

Definition 2.1. *A deterministic Turing machine A is an approximation algorithm for an optimization problem $P = (I, S, m, \text{goal})$ if*

1. *the running time A is polynomial,*
2. *$A(x) \in S(x)$ for all $x \in I$.*

Of course, there are good and not so good approximation algorithms and we develop a framework to measure the quality or *approximation performance* of such an algorithm.

Definition 2.2. *1. Let P be an optimization problem, $x \in I$, and $y \in S(x)$. The performance ratio of y with respect to x is defined as*

$$\text{PR}(x, y) = \max \left\{ \frac{m(x, y)}{\text{OPT}(x)}, \frac{\text{OPT}(x)}{m(x, y)} \right\}. \quad 1$$

2. *Let $\alpha : \mathbb{N} \rightarrow \mathbb{Q}$. An approximation algorithm A is an α -approximation algorithm, if for all $x \in I$,*

$$\text{PR}(x, A(x)) \leq \alpha(|x|).$$

The definition of $\text{PR}(x, y)$ basically means that in the case of a minimization problem, we measure how many times the objective value of the computed solution exceeds the objective value of an optimum solution. In the case of a maximization problem, we do the same but we take the reciprocal. This may seem strange at a first glance but it has the advantage that we can treat minimization and maximization problems in a uniform way. (Be aware though that some authors use $m(x, y)/\text{OPT}(x)$ to measure the approximation performance in case of maximization problems. But this is merely a question of faith.)

Definition 2.3. *1. Let F be some set of functions $\mathbb{N} \rightarrow \mathbb{Q}$. An optimization problem $P \in \text{NPO}$ is contained in the class $F\text{-APX}$ if there is an $f \in F$ such that there exists an f -approximation algorithm for P .*

¹Note that m only attains positive values. Thus, the quotient is always defined.

2. APX := $O(1)$ -APX.

(I hope that the elegant definition above clarifies why PR was defined for maximization problems as it is.) There is a well-known 2-approximation algorithm for Δ -TSP that is based on minimum spanning trees, thus

$$\Delta\text{-TSP} \in \text{APX}.$$

Even stronger is the concept of a *polynomial time approximation scheme*.

Definition 2.4. A deterministic Turing machine A is a polynomial time approximation scheme (PTAS) for an optimization problem $P = (I, S, m, \text{goal})$ if on input $\langle x, \epsilon \rangle$ for all small enough $\epsilon > 0$,

1. the running time of A is polynomial in the size of x (but not necessarily in ϵ), and
2. $A(x, \epsilon)$ is a feasible solution for x with performance ratio $1 + \epsilon$.

We do not have to distinguish between minimization and maximization problems. If a solution y has performance ratio $1 + \epsilon$ in the case of a maximization problem, then we know that $m(x, y) \geq \frac{1}{1+\epsilon} \text{OPT}(x)$. We have

$$\frac{1}{1+\epsilon} = 1 - \frac{\epsilon}{1+\epsilon} \geq 1 - \epsilon,$$

which is exactly what we want.

Definition 2.5. PTAS is the class of all problems in NPO that have a PTAS.

We have

$$\text{PO} \subseteq \text{PTAS} \subseteq \text{APX}$$

If $\text{P} \neq \text{NP}$, then both inclusions are strict. Under this assumption, a problem in $\text{APX} \setminus \text{PTAS}$ is Maximum Satisfiability (see the next chapters for a proof), a problem in $\text{PTAS} \setminus \text{PO}$ is Knapsack (solve the next exercise for a proof).

Problem 2.6. Knapsack is the following problem:

Instances: rational numbers w_1, \dots, w_n (weights), p_1, \dots, p_n (profits), and B (capacity bound) such that $w_\nu \leq B$ for all ν .

Solutions: $I \subseteq \{1, \dots, n\}$ such that $\sum_{i \in I} w_i \leq B$

Measure: $\sum_{i \in I} p_i$, the total profit of the items packed

Goal: max

We may assume w.l.o.g. that all the p_ν are natural numbers. If this is not the case, assume that $p_\nu = x_\nu/y_\nu$ with $\text{gcd}(x_\nu, y_\nu) = 1$. Let $Y = y_1 \cdots y_n$. We now replace p_ν by $p_\nu \cdot Y \in \mathbb{N}$. Any knapsack that maximizes the old objective function also maximizes the new one. The size of the instance is only polynomially larger. (Note that we encode all inputs in binary.)

Exercise 2.1. 1. Show that there is an algorithm for **Knapsack** with running time polynomial in n and $P := \max_{1 \leq \nu \leq n} p_\nu$. (Compute by dynamic programming sets of indices $I(i, p)$ such that

- $\nu \leq i$ for all $\nu \in I(i, p)$,
- the sum of the p_ν with $\nu \in I(i, p)$ is exactly p , and
- the sum of all w_ν with $\nu \in I(i, p)$ is minimum among all such set of indices.)

2. Show that we get a PTAS out of this pseudopolynomial algorithm as follows:

- Let $S = \epsilon P/n$ and $\hat{p}_\nu = \lfloor p_\nu/S \rfloor$ for $1 \leq \nu \leq n$.
- Find an optimum solution for the instance $w_1, \dots, w_n, \hat{p}_1, \dots, \hat{p}_n$, and B .

Remark 2.7. The running of the PTAS constructed in the previous exercise is also polynomial in $\frac{1}{\epsilon}$. This is called an fully polynomial time approximation scheme (FPTAS). The corresponding complexity class is denoted by FPTAS.

Exercise 2.2. A super fully polynomial time approximation scheme is a PTAS whose running time is polynomial in $\log \frac{1}{\epsilon}$. Show that if **Knapsack** has a super fully polynomial time approximation scheme, then $P = NP$.

2.1 Gap problems

A promise problem is a tuple of languages $Q = (L, U)$ with $L \subseteq U$. (Think of U as the universe of admissible inputs.) A Turing machine decides a promise problem, if for all $x \in U$, $M(x) = 1$ if $x \in L$ and $M(x) = 0$ if $x \in U \setminus L$. On inputs not in U , M may output whatever it wants. Since we do not have to care about the behaviour of M on inputs not in U , we can also think that we get an input with the additional promise that it is in U . The elements in L are often called *yes*-instances, the elements in $U \setminus L$ are called *no*-instances, and the elements not in U are called *don't care*-instances. "Ordinary" decision problems are a special case of promise problems, we just set $U = \{0, 1\}^*$.

Many-one-reductions can be extended to promise problems in a natural way. Let $Q = (L, U)$ and $Q' = (L', U')$ be two promise problems. Q is polynomial time many-one reducible to Q' if there is a polynomial time computable function f such that

$$\begin{aligned} x \in L &\implies f(x) \in L' && \text{and} \\ x \in U \setminus L &\implies f(x) \in U' \setminus L' \end{aligned}$$

That means that yes-instances are mapped to yes-instances and no-instances are mapped to no-instances. A promise problem Q is C -hard for some class C of decision or promise problems, if every problem in C is polynomial time many-one reducible to Q .

Definition 2.8. Let $P = (I_P, S_P, m_P, \text{goal})$ be an optimization problem and $a < b$. $\text{gap}(a, b)$ - P is the promise problem (L, U) where

$$U = \{x \mid \text{OPT}(x) \leq a \text{ or } \text{OPT}(x) \geq b\}$$

and

$$L = \begin{cases} \{x \mid \text{OPT}(x) \geq b\} & \text{if goal} = \max \\ \{x \mid \text{OPT}(x) \leq a\} & \text{if goal} = \min \end{cases}$$

That is, we get an instance x and the promise that the objective value is at most a or at least b and we shall decide which of these two options is the case. There is a difference in the definition of L for maximization and minimization problems because the yes-instances shall be the inputs with solutions that have a "good" objective value. We will also allow a and b to be functions $\mathbb{N} \rightarrow \mathbb{N}$ that depend on $|x|$.

Theorem 2.9. If $\text{gap}(a, b)$ - P is NP-hard for polynomial time computable functions a and b with input given in unary and output given in binary, then there is no α -approximation algorithm for P with $\alpha < b/a$, unless $P = \text{NP}$.

Proof. Suppose on the contrary that such an algorithm A exists. We only show the case $\text{goal}_P = \min$, the other case is treated similarly. Since $\text{gap}(a, b)$ - P is NP-hard, there is a polynomial time many-one reduction f from SAT to $\text{gap}(a, b)$ - P . We design a polynomial time algorithm for SAT as follows:

Input: formula ϕ in CNF

1. Compute $x = f(\phi)$ and $y = A(x)$.
2. If $m_P(x, y) < b(|x|)$, then accept, else reject.

Let us see why this algorithm is correct. If $\phi \in \text{SAT}$, then $\text{OPT}_P(x) \leq a(x)$ and

$$m_P(x, y) \leq \alpha(|x|) \cdot \text{OPT}_P(x) < b(|x|).$$

If $\phi \notin \text{SAT}$, then $\text{OPT}_P(x) \geq b(|x|)$ and

$$m_P(x, y) \geq \text{OPT}_P(x) \geq b(|x|).$$

Thus the algorithm works correctly. It is obviously polynomial time. Therefore, $P = \text{NP}$. ■

In Exercise 1.1, we have seen that there is no polynomial p such that TSP can be approximated within $2^{p(n)}$ where n is the number of nodes of the given graph. However, note that n is not the size of the instance but $O(p(n)n^2)$. Thus $\text{gap}(n, 2^{n^{1-\epsilon}})$ -TSP is NP-hard

Since TSP \in NPO, we get the following result.

Theorem 2.10. *If $P \neq \text{NP}$, then $\text{APX} \subsetneq \text{NPO}$.*

We can always approximate TSP within $2^{O(|x|)}$ where x is the given instance, since with $|x|$ symbols we can encode integers up to $2^{O(|x|)}$. Thus TSP is contained in the class exp-APX, as defined below.

Definition 2.11. $\text{exp-APX} = \{2^p \mid p \text{ is a polynomial}\}$ -APX.

Thus the theorem above can be strengthened to the following statement.

Theorem 2.12. *If $P \neq \text{NP}$, then $\text{APX} \subsetneq \text{exp-APX}$.*

Exercise 2.3. *What is the difference between exp-APX and NPO?*

2.2 Approximation preserving reductions and hardness

Let P and P' be two optimization problems. If P is reducible to P' (in some sense to be defined), then we would like to turn approximate solution of P' back into approximate solutions of P . That is, we do not only need a function that maps instances of P to instances of P' , we also need to transfer solutions of P' back to solutions of P like we did for #P functions. Many of the reductions between NP-complete problems give also this second function for free. But what they usually do not do is that they preserve approximation factors.

Problem 2.13. Maximum Clique (Clique) *is the following problem:*

Instances: graph $G = (V, E)$

Solutions: all cliques of G , i.e., all $C \subseteq V$ such that for all $u, v \in C$ with $u \neq v$, $\{u, v\} \in E$

Measure: $\#C$, the size of the clique

Goal: max

Problem 2.14. Vertex Cover (VC) *is the following problem:*

Instances: graph $G = (V, E)$

Solutions: all subsets C of V such that for each $\{u, v\} \in E$, $C \cap \{u, v\} \neq \emptyset$

Measure: $\#C$

Goal: min

Exercise 2.4. *There is an easy reduction $\text{Clique}_{\text{dec}} \leq_P \text{VC}_{\text{dec}}$ that simply maps G to its complement.*

1. How does one get a clique of G from a vertex cover of the complement?
2. Assume we have a vertex cover that is a 2-approximation. What approximation do we get for **Clique** from this?

Definition 2.15. Let $P, P' \in \text{NPO}$. P is reducible to P' by an approximation preserving reduction (short: P is AP-reducible to P' or even shorter, $P \leq_{\text{AP}} P'$) if there are two functions $f, g : \{0, 1\}^* \times \mathbb{Q}^+ \rightarrow \{0, 1\}^*$ and an $\alpha \geq 1$ such that

1. for all $x \in \text{I}_P$ and $\beta > 1$, $f(x, \beta) \in \text{I}_{P'}$,
2. for all $x \in \text{I}_P$ and $\beta > 1$, if $\text{S}_P(x) \neq \emptyset$ then $\text{S}_{P'}(f(x, \beta)) \neq \emptyset$,
3. for all $x \in \text{I}_P$, $y \in \text{S}_{P'}(f(x, \beta))$, and $\beta > 1$,

$$g(x, y, \beta) \in \text{S}_P(x),$$
4. f and g are deterministically polynomial time computable for fixed $\beta > 1$,
5. for all $x \in \text{I}_P$ and all $y \in \text{S}_{P'}(f(x, \beta))$, if y is a β -approximate solution of $f(x, \beta)$, then $g(x, y, \beta)$ is an $(1 + \alpha(\beta - 1))$ -approximate solution of x .

(f, g, α) is called an AP-reduction from P to P' .²

Lemma 2.16. If $P \leq_{\text{AP}} P'$ and $P' \in \text{APX}$, then $P \in \text{APX}$.

Proof. Let (f, g, α) be an AP-reduction from P to P' and let A' be a β -approximation algorithm for P' . Given $x \in \text{I}_P$, $A(x) := g(x, A'(f(x, \beta)), \beta)$ is a $(1 + \alpha(\beta - 1))$ -approximate solution for x . This follows directly from the definition of AP-reduction. Furthermore, A is polynomial time computable.

■

Exercise 2.5. Let $P \leq_{\text{AP}} P'$. Show that if $P' \in \text{PTAS}$, so is P .

The reduction in Exercise 2.4 is not an AP-reduction. This has a deeper reason. While there is a 2-approximation algorithm for **VC**, **Clique** is much harder to approximate. Håstad [Hås99] shows that any approximation algorithm with performance ratio $n^{1-\epsilon_0}$ for some $\epsilon_0 > 0$ would imply $\text{ZPP} = \text{NP}$ (which is almost as unlikely as $\text{P} = \text{NP}$).

²The functions f, g depend on the quality β of the solution y . I am only aware of one example where this dependence seems to be necessary, so usually, f and g will not depend on β .

Problem 2.17. Maximum Independent Set (IS) is the following problem:

Instances: graph $G = (V, E)$

Solutions: independent sets of G , i.e., all $S \subseteq V$ such that for all $u, v \in S$ with $u \neq v$, $\{u, v\} \notin E$

Measure: $\#S$

Goal: max

Exercise 2.6. Essentially the same idea as in Exercise 2.4 gives a reduction from Clique to IS. Show that this is an AP-reduction.

Definition 2.18. Let $\mathbf{C} \subseteq \text{NPO}$. A problem P is \mathbf{C} -hard (under AP-reductions) if for all $P' \in \mathbf{C}$, $P' \leq_{\text{AP}} P$. P is \mathbf{C} -complete if it is in \mathbf{C} and \mathbf{C} -hard.

Lemma 2.19. \leq_{AP} is transitive.

Proof. Let $P \leq_{\text{AP}} P'$ and $P' \leq_{\text{AP}} P''$. Let (f, g, α) and (f', g', α') be the corresponding reductions. Let $\gamma = 1 + \alpha'(\beta - 1)$. We claim that $(F, G, \alpha\alpha')$ is an AP-reduction from P to P'' where

$$\begin{aligned} F(x, \beta) &= f'(f(x, \gamma), \beta), \\ G(x, y, \beta) &= g(x, g'(f(x, \gamma), y, \gamma), \beta). \end{aligned}$$

We verify that $(F, G, \alpha\alpha')$ is indeed an AP-reduction by checking the five conditions in Definition 2.15:

1. Obvious.
2. Obvious, too.
3. Almost obvious, thus we give a proof. Let $x \in I_P$ and $y \in S_{P''}(F(x, \beta))$. We know that $g'(f(x, \gamma), y, \beta) \in S_{P'}(f(x, \gamma))$, since (f', g', α') is an AP-reduction. But then also $g(x, g'(f(x, \gamma), y, \gamma), \beta) \in S_P(x)$, since (f, g, α) is an AP-reduction.
4. Obvious.
5. Finally, if y is a β -approximation to $f'(f(x, \gamma), \beta)$, then $g'(f(x, \gamma), y, \beta)$ is a $(1 + \alpha'(\beta - 1))$ -approximation to $f(x)$. But then $g(x, g'(f(x, \gamma), y, \beta), \gamma)$ is a $(1 + \alpha\alpha'(\beta - 1))$ -approximation to x , as

$$1 + \alpha(1 + \alpha'(\beta - 1) - 1) = 1 + \alpha\alpha'(\beta - 1).$$

Lemma 2.20. Let $\mathbf{C} \subseteq \text{NPO}$. If $P \leq_{\text{AP}} P'$ and P is \mathbf{C} -hard, then P' is also \mathbf{C} -hard.

Proof. Let $Q \in \mathcal{C}$ be arbitrary. Since P is \mathcal{C} -hard, $Q \leq_{\text{AP}} P$. Since \leq_{AP} is transitive, $Q \leq_{\text{AP}} P'$. ■

Thus once we have identified one APX-hard problem, we can prove the APX-hardness using AP-reductions. A canonical candidate is of course the following problem:

Problem 2.21 (Max-SAT). *The Maximum Satisfiability problem (Max-SAT) is defined as follows:*

Instances: formulas in CNF
Solutions: Boolean assignments to the variables
Measure: the number of clauses satisfied
Goal: max

Proposition 2.22. *Max-SAT is APX-hard.*

The proof of this proposition above is very deep, we will spend the next few weeks with it.

Exercise 2.7. *Give a simple 2-approximation algorithm for SAT.*

2.3 Further exercises

Here in an NPO-complete problem.

Problem 2.23. *Maximum Weighted Satisfiability is the following problem:*

Instances: Boolean formula ϕ with variables x_1, \dots, x_n having nonnegative weights w_1, \dots, w_n
Solutions: Boolean assignments $\alpha : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ that satisfy ϕ
Measure: $\max\{1, \sum_{i=1}^n w_i \alpha(x_i)\}$
Goal: max

Exercise 2.8. 1. *Show that every maximization problem in NPO is AP-reducible to Maximum Weighted Satisfiability. (Hint: Construct an NP-machine that guesses a solution y to input x and computes $m(x, y)$. Use a variant of the proof of the Cook-Karp-Levin Theorem to produce an appropriate formula in CNF. Assign only nonzero weights to variables that contain the bits of $m(x, y)$.)*

2. *Show that every minimization problem in NPO is AP-reducible to Minimum Weighted Satisfiability.*

3. *Show that Maximum Weighted Satisfiability is AP-reducible to Minimum Weighted Satisfiability and vice versa.*

4. *Conclude that Maximum (Minimum) Weighted Satisfiability is NPO-complete*

The world of optimization classes

$$PO \subseteq PTAS \subseteq APX \subseteq \text{exp-APX} \subseteq NPO$$

All of these inclusion are strict, provided that $P \neq NP$. Under this assumption, we have for instance

- Knapsack $\in PTAS \setminus PO$
- TSP $\in \text{exp-APX} \setminus APX$
- Weighted Satisfiability $\in NPO \setminus \text{exp-APX}$.

The goal of the next chapters is to prove that Max-SAT is in $APX \setminus PTAS$ provided that $P \neq NP$.

3 Probabilistically checkable proofs and inapproximability

3.1 Probabilistically checkable proofs (PCPs)

3.1.1 Probabilistic verifiers

A *polynomial time probabilistic verifier* is a polynomial time probabilistic Turing machine that has oracle access to a proof $\pi \in \{0, 1\}^*$ in the following way: The proof π induces a function $\{0, 1\}^{\log(|\pi|)} \rightarrow \{0, 1\}$ by mapping $b \in \{0, 1\}^{\log(|\pi|)}$ to the bit of π that stands in the position that is encoded by the binary representation b . By abuse of notation, we will call this function again π . If the verifier queries a bit outside the range of π , then the answer will be 0.

A verifier described above may query π several times and each query may depend on previous queries. Such a behavior is called *adaptive*. We need a more restricted kind of verifiers, called *nonadaptive*: A nonadaptive verifier gets the proof π again as an oracle, but in a slightly different form: The verifier can write down several positions of π at one time. If it enters the query state, it gets the values of all the positions that it queries. But the verifier may enter the query state only once, i.e., the verifier has to decide in advance which bits it wants to query.

A nonadaptive probabilistic verifier is called $(r(n), q(n))$ -restricted if it uses $r(n)$ -bits of randomness and queries $q(n)$ bits of π for all n and all inputs x of length n .

Definition 3.1. Let $r, q : \mathbb{N} \rightarrow \mathbb{N}$. A language L belongs to the class $\text{PCP}[r, q]$ if there exists a (r, q) -restricted nonadaptive polynomial time probabilistic verifier such that the following holds:

1. For any $x \in L$, there is a proof π such that

$$\Pr_y[V^\pi(x, y) = 1] = 1.$$

2. For any $x \notin L$ and for all proofs π ,

$$\Pr_y[V^\pi(x, y) = 0] \geq 1/2.$$

The probabilities are taken over the the random strings y .

In other words, if x is in L , then there is a proof π that convinces the verifier regardless of the random string y . If x is not in L , then the verifier will detect a “wrong” proof with probability at least $1/2$, that is, for half of the random strings.

Since the verifier is $r(n)$ -restricted, there are only $2^{r(n)}$ (relevant) random strings. For any fixed random string, the verifier queries at most $q(n)$ bits of the proof. Therefore, for an input x of length n , we only have to consider proofs of length $q(n)2^{r(n)}$, since the verifier cannot query more bits than that.

3.1.2 A different characterization of NP

Once we have defined the PCP classes, the obvious question is: What is this good for and how is it related to other classes? While complexity theorists also like to answer the second part of the question without knowing an answer to the first part, here the answer to the second part also gives the answer to the first part.

Let R and Q denote sets of functions $\mathbb{N} \rightarrow \mathbb{N}$. We generalize the notion of $\text{PCP}[r, q]$ in the obvious way:

$$\text{PCP}[R, Q] = \bigcup_{r \in R, q \in Q} \text{PCP}[r, q].$$

The characterization of NP by polynomial time verifiers immediately yields the following result.

Proposition 3.2. $\text{NP} = \text{PCP}[0, \text{poly}(n)]$.

In the theorem above, we do not use the randomness at all. The next result, the celebrated PCP theorem [ALM⁺98], shows that allowing a little bit of randomness reduces the number of queries dramatically.

Theorem 3.3 (PCP Theorem). $\text{NP} = \text{PCP}[O(\log n), O(1)]$.

What does this mean? By allowing a little randomness—note that $O(\log n)$ are barely sufficient to choose $O(1)$ bits of the proof at random—and a bounded probability of failure, we can check the proof π by just reading a constant number of bits of π ! This is really astonishing.

Exercise 3.1. Show that $\text{PCP}[O(\log n), O(1)] \subseteq \text{NP}$. (*Hint: How many random strings are there?*)

The other direction is way more complicated, we will spend the next few lectures with its proof. We will not present the original proof by Arora et al. [ALM⁺98] but a recent and—at least compared to the first one—elegant proof by Irit Dinur [Din07].

3.2 PCPs and gap problems

The PCP theorem is usually used to prove hardness of approximation results. Dinur's proof goes the other way around, we show that the statement of the PCP theorem is equivalent to the NP-hardness of some gap problem

Theorem 3.4. *The following two statements are equivalent:*

1. $\text{NP} = \text{PCP}[O(\log n), O(1)]$.
2. *There is an $\epsilon > 0$ such that $\text{gap}(1 - \epsilon, 1)\text{-Max-3-SAT}$ is NP-hard.*¹

Proof. “ \implies ”: Let L be any NP-complete language. By assumption, there is an $(r(n), q)$ -restricted nonadaptive polynomial time probabilistic verifier V with $r(n) = O(\log n)$ and $q = O(1)$. We can assume that V always queries exactly q bits.

Let x be an input for L of length n . We will construct a formula in 3-CNF ϕ in polynomial time such that if $x \in L$, then ϕ is satisfiable and if $x \notin L$, then every assignment can satisfy at most a fraction of $1 - \epsilon$ of the clauses for some fixed $\epsilon > 0$.

For each position i in the proof, there will be one Boolean variable v_i . If v_i is set to 1, this will mean that the corresponding i th bit is 1; if it is set to zero, then this bit is 0. Since we can restrict ourselves to proofs of length $\leq q \cdot 2^{r(n)} = \text{poly}(n)$, the number of these variables is polynomial.

For a random string y , let $i(y, 1), \dots, i(y, q)$ denote the positions of the bits that the verifier will query. (Note that the verifier is nonadaptive, hence these position can only depend on y .) Let A_y be the set of all q -tuples $(b_1, \dots, b_q) \in \{0, 1\}^q$ such that if the $i(y, j)$ th bit of the proof is b_j for $1 \leq j \leq q$, then the verifier will reject (with random string y).

For each tuple $(b_1, \dots, b_q) \in A_y$, we construct a clause of q literals, that is true iff the variables $v_{i(y,1)}, \dots, v_{i(y,q)}$ do not take the value b_1, \dots, b_q , i.e., $v_{i(y,1)}^{1-b_1} \vee \dots \vee v_{i(y,q)}^{1-b_q}$. (Here, for a Boolean variable v , $v^1 = v$ and $v^0 = \bar{v}$.) The formula ϕ has $\leq |A_y| 2^{r(n)} \leq 2^{q+r(n)} = \text{poly}(n)$ many clauses. These clauses have length q . Like in the reduction of SAT to 3SAT, for each such clause c , there are $q - 2$ clauses c_1, \dots, c_{q-2} of length three in the variables of c and some additional variables such that any assignment that satisfies c can be extended to an assignment that satisfies c_1, \dots, c_{q-2} and conversely, the restriction of any assignment that satisfies c_1, \dots, c_{q-2} satisfies c , too. This replacement can be computed in polynomial time.

The formula ϕ can be computed in polynomial time: We enumerate all (polynomially many) random strings. For each such string y , we simulate

¹Instead of stating the absolute bounds $(1 - \epsilon)m$ and m , where m is the number of clauses of the given instance, we just state the relative bounds $1 - \epsilon$ and 1. This is very convenient here, since there is an easy upper bound of the objective value, namely m .

the verifier V to find out which bits he will query. Then we can give him all the possible answers to the bits he queried to compute the sets A_y .

If $x \in L$, then there will be a proof π such that $V^\pi(x, y) = 1$ for every random string y . Therefore, if we set the variables of ϕ as given by this proof π , then ϕ will be satisfied.

If $x \notin L$, then for any proof π , there are at least $2^{r(n)}/2$ random strings y for which $V^\pi(x, y) = 0$. For each such y , one clause corresponding to a tuple in A_y will not be satisfied. In other words, for any assignment, $2^{r(n)}/2$ clauses will not be satisfied. The total number of clauses is bounded by $(q-2)2^{q+r(n)}$. The fraction of unsatisfied clauses therefore is

$$\geq \frac{2^{r(n)}/2}{(q-2)2^{q+r(n)}} \geq 2^{-q-1}/(q-2),$$

which is a constant.

“ \Leftarrow ”: By Exercise 3.1, it suffices to show that $\text{NP} \subseteq \text{PCP}[\text{O}(\log n), \text{O}(1)]$. Let $L \in \text{NP}$. By assumption, there is a polynomial time computable function f such that

- $x \in L \implies f(x)$ is a satisfiable formula in 3-CNF,
- $x \notin L \implies f(x)$ is a formula in 3-CNF such that every assignment satisfies at most $(1 - \epsilon)$ of the clauses.

We construct a probabilistic verifier as follows:

Input: input x , proof π

1. Compute $f(x)$.
2. Randomly select a clause c from $f(x)$.
3. Interpret π as an assignment to $f(x)$ and read the bits that belong to the variables in c .
4. Accept if the selected clause c is satisfied. Reject otherwise.

Let m be the number of clauses of $f(x)$. To select a clause at random, the verifier reads $\log m$ random bits and interprets it as a number. If it “selects” a nonexisting clause, then it will accept. So we can think of m being a power of two at the expense of replacing ϵ by $\epsilon/2$.

Now assume $x \in L$. Then $f(x)$ is satisfiable and therefore, there is a proof that will make the verifier always accept, namely a satisfying assignment of $f(x)$. If $x \notin L$, then no assignment will satisfy more than $1 - \epsilon$ of the clauses. In particular, the probability that the verifier selects a clause that is satisfied is at most $1 - \epsilon$. By repeating this process for a constant number of times, we can bring the error probability down to $1/2$.

Since $f(x)$ is in 3-CNF, the verifier needs $\text{O}(\log m) = \text{O}(\log |x|)$ random bits, and it only queries $\text{O}(1)$ bits of the proof. ■

Exercise 3.2. *Let c be a clause of length q . Construct clauses c_1, \dots, c_{q-2} of length three in the variables of c and some additional variables such that any assignment that satisfies c can be extended to an assignment that satisfies c_1, \dots, c_{q-2} and conversely, the restriction of any assignment that satisfies c_1, \dots, c_{q-2} satisfies c , too.*

Note that we get an explicit value for ϵ in terms of q . Thus in order to get good nonapproximability results from the PCP theorem, we want q to be as small as possible.

3.3 Further exercises

Exercise 3.3. *Show that $\text{PCP}[O(\log n), 2] = \text{P}$.*

It can be shown—tadah!—that three queries are enough to capture NP; however, it is not possible to get error probability $1/2$ and one-sided error, see [GLST98] for further discussions.

A Max-3-SAT is APX-hard

In this chapter, we will strengthen the result of the previous one by showing that Max-3-SAT is in fact APX-hard. We do this in several steps. First, we show that any maximization problem in APX is AP-reducible to Max-3-SAT. Second, we show that for every minimization problem P , there is a maximization problem P' such that $P \leq_{\text{AP}} P'$. This will conclude the proof.

Our proof of the PCP-Theorem will also yield the following variant, which we will use in the following.

Theorem A.1 (PCP-Theorem'). *There are $\epsilon > 0$ and polynomial time computable functions f_{PCP} and g_{PCP} such that for every formula ψ in 3-CNF:*

1. $f_{\text{PCP}}(\psi)$ is a formula in 3-CNF,
2. if ψ is satisfiable, so is $f_{\text{PCP}}(\psi)$,
3. if ψ is not satisfiable, then any assignment can satisfy at most a fraction of $1 - \epsilon$ of the clauses in $f_{\text{PCP}}(\psi)$,
4. if a is an assignment for $f_{\text{PCP}}(\psi)$ that satisfies more than a fraction of $1 - \epsilon$ of the clauses, then $g_{\text{PCP}}(\psi, a)$ is an assignment that satisfies ψ .

Theorem A.2. *Let $P = (I_P, S_P, m_P, \max)$ be a maximization problem in APX. Then $P \leq_{\text{AP}} \text{Max-3-SAT}$.*

Proof. Our goal is to construct an AP reduction (f, g, α) from P to Max-3-SAT. Let f_{PCP} and g_{PCP} be the functions constructed in Theorem A.1 and let ϵ be the corresponding constant. Let A be a b -approximation algorithm for P . Let

$$\alpha = 2(b \log b + b - 1) \frac{1 + \epsilon}{\epsilon}.$$

Our goal is to define the functions f and g given β . Let $r = 1 + \alpha(\beta - 1)$. If $r < b$, then

$$\beta = \frac{r - 1}{\alpha} + 1 = \frac{\epsilon}{2(1 + \epsilon)} \cdot \frac{r - 1}{b \log b + b - 1} + 1 < \frac{\epsilon}{2k(1 + \epsilon)} + 1 \quad (\text{A.1})$$

where $k = \lceil \log_r b \rceil$. The last inequality follows from

$$k \leq \frac{\log b}{\log r} + 1 \leq \frac{r \log b}{r - 1} + 1 \leq \frac{b \log b + b - 1}{r - 1}.$$

Let $\mu(x) = m_P(x, A(x))$. Since A is a b -approximation algorithm, $\mu(x) \leq \text{OPT}_P(x) \leq b\mu(x)$.

The following Turing machine computes f :

Input: $x \in \{0, 1\}^*$, $\beta \in \mathbb{Q}^+$

1. Construct formulas $\phi_{x,i}$ in 3-CNF that are true if $\text{OPT}_P(x) \geq i$.
(These formulas $\phi_{x,i}$ can be uniformly constructed in polynomial time, cf. the proof of Cook's theorem.)
2. Let $\psi_{x,\kappa} = f_{\text{PCP}}(\phi_{x,\mu(x)r^\kappa})$, $1 \leq \kappa \leq k$.
By padding with dummy clauses, we may assume that all the $\psi_{x,\kappa}$ have the same number of clauses c .
3. Return $\psi_x = \bigvee_{\kappa=1}^k \psi_{x,\kappa}$.

The function g is computed as follows:

Input: $x \in \{0, 1\}^*$, assignment a with performance ratio β

1. If $b \leq 1 + \alpha(\beta - 1)$, then return $A(x)$.¹
2. Else let κ_0 be the largest κ such that $g_{\text{PCP}}(\phi_{x,\mu(x)r^\kappa,a})$ satisfies $\phi_{x,\mu(x)r^\kappa}$.
(We restrict a to the variables of $\phi_{x,\mu(x)r^\kappa}$.)
3. This satisfying assignment corresponds to a feasible solution y with $m_P(x, y) \geq \mu(x)r^{\kappa_0}$.
Return y .

If $b \leq 1 + \alpha(\beta - 1)$, then we return $A(x)$. This is a b -approximation by assumption. Since $b \leq 1 + \alpha(\beta - 1)$, we are done.

Therefore, assume that $b > 1 + \alpha(\beta - 1)$. We have

$$\text{OPT}_{\text{Max-3-SAT}}(\psi_x) - m_{\text{Max-3-SAT}}(\psi_x, a) \leq \text{OPT}_{\text{Max-3-SAT}}(\psi_x) \frac{\beta - 1}{\beta} \leq kc \frac{\beta - 1}{\beta}.$$

Let β_κ denote the performance ratio of a with respect to $\psi_{x,\kappa}$, i.e., we view a as an assignment of $\psi_{x,\kappa}$. We have

$$\begin{aligned} \text{OPT}_{\text{Max-3-SAT}}(\psi_x) - m_{\text{Max-3-SAT}}(\psi_x, a) &\geq \text{OPT}_{\text{Max-3-SAT}}(\psi_{x,\kappa}) - m_{\text{Max-3-SAT}}(\psi_{x,\kappa}, a) \\ &= \text{OPT}_{\text{Max-3-SAT}}(\psi_{x,\kappa}) \frac{\beta_\kappa - 1}{\beta_\kappa} \\ &\geq \frac{c}{2} \cdot \frac{\beta_\kappa - 1}{\beta_\kappa}. \end{aligned}$$

¹Here is the promised dependence on β .

The last inequality follows from the fact that any formula in CNF has an assignment that satisfies at least half of the clauses. This yields

$$\frac{c}{2} \cdot \frac{\beta_\kappa - 1}{\beta_\kappa} \leq kc \frac{\beta - 1}{\beta}$$

and finally

$$\beta_\kappa \leq \frac{1}{1 - 2k(\beta - 1)/\beta}.$$

Exploiting (A.1), we get, after some routine calculations,

$$\beta_\kappa \leq 1 + \epsilon.$$

This means that a satisfies at least a fraction of $1/\beta_\kappa \geq 1 - \epsilon$ of the clauses of $\psi_{x,\kappa}$. Then $g_{\text{PCP}}(a)$ satisfies $\phi_{x,\mu(x)r^\kappa}$ if and only if $\phi_{x,\mu(x)r^\kappa}$ is satisfiable. This is equivalent to the fact that $\text{OPT}_P(x) \geq \mu(x)r^\kappa$. By the definition of κ_0 ,

$$\mu(x)r^{\kappa_0+1} > \text{OPT}_P(x) \geq \mu(x)r^{\kappa_0}.$$

This means that $m_P(x, y) \geq \mu(x)r^{\kappa_0}$. But then y is an r -approximate solution. Then we are done, since $r = 1 + \alpha(\beta - 1)$ by definition. ■

Theorem A.3. *For every minimization problem $P \in \text{APX}$, there is a maximization problem $P' \in \text{APX}$ such that $P \leq_{\text{AP}} P'$.*

Proof. Let A be a b -approximation algorithm for P . Let $\mu(x) = m_P(x, A(x))$ for all $x \in I_P$. Then $\mu(x) \leq b \text{OPT}_P(x)$. P' has the same instances and feasible solutions as P . The objective function is however different:

$$m_{P'}(x, y) = \begin{cases} (k+1)\mu(x) - k m_P(x, y) & \text{if } m_P(x, y) \leq \mu(x) \\ \mu(x) & \text{otherwise} \end{cases}$$

where $k = \lceil b \rceil$. We have $\mu(x) \leq \text{OPT}_{P'}(x) \leq (k+1)\mu(x)$. This means, that A is a $(k+1)$ -approximation algorithm for P' . Hence, $P' \in \text{APX}$.

The AP reduction (f, g, α) from P' to P is defined as follows: $f(x, \beta) = x$ for all $x \in I_P$. (Note that we do not need any dependence on β here). Next, we set

$$g(x, y, \beta) = \begin{cases} y & \text{if } m_P(x, y) \leq \mu(x) \\ A(x) & \text{otherwise} \end{cases}$$

And finally, $\alpha = k + 1$.

Let y be a β -approximate solution to x under $m_{P'}$, that is, $R_{P'}(x, y) = \text{OPT}_{P'}(x)/m_{P'}(x, y) \leq \beta$. We have to show that $R_P(x, y) \leq 1 + \alpha(\beta - 1)$.

We distinguish two cases: The first one is $m_P(x, y) \leq \mu(x)$. In this case,

$$\begin{aligned}
m_P(x, y) &= \frac{(k+1)\mu(x) - m_{P'}(x, y)}{k} \\
&\leq \frac{(k+1)\mu(x) - \text{OPT}_{P'}(x)/\beta}{k} \\
&\leq \frac{(k+1)\mu(x) - (1 - (\beta - 1)) \text{OPT}_{P'}(x)}{k} \\
&\leq \text{OPT}_P(x) + \frac{\beta - 1}{k} \text{OPT}_{P'}(x) \\
&\leq \text{OPT}_P(x) + \frac{\beta - 1}{k} (k+1)\mu(x) \\
&\leq \text{OPT}_P(x) + (\beta - 1)(k+1)\mu(x)/r \\
&\leq (1 + \alpha(\beta - 1)) \text{OPT}_P(x).
\end{aligned}$$

This completes the first case.

For the second case, note that

$$m_P(x, g(x, y)) = m_P(x, A(y)) \leq b \text{OPT}_P(x) \leq (1 + \alpha(\beta - 1)) \text{OPT}_P(x).$$

Thus, $P \leq_{\text{AP}} P'$. ■

Now Theorems A.2 and A.3 imply the following result.

Theorem A.4. *Max-3-SAT is APX-hard.*

A.1 Further exercises

Exercise A.1. *Show that Max-3-SAT \leq_{AP} Clique. (In particular, Clique does not have a PTAS, unless P = NP.)*

The k th cartesian product of a graph $G = (V, E)$ is a graph with nodes V^k and there is an edge between (u_1, \dots, u_k) and (v_1, \dots, v_k) if either $u_i = v_i$ or $\{u_i, v_i\} \in E$ for all $1 \leq i \leq k$.

Exercise A.2. 1. *Prove that if G has a clique of size s , then G^k has a clique of size s^k .*

2. *Use this to show that if Clique \in APX, then Clique \in PTAS. Now apply Exercise A.1.*

Håstad [Hås99] shows that any approximation algorithm with performance ratio $n^{1-\epsilon_0}$ for some $\epsilon_0 > 0$ would imply ZPP = NP. On the other hand, achieving a performance ratio of n is trivial.

4 The BLR test

4.1 Linearity and approximate linearity

Consider a Boolean function

$$f : \{0, 1\}^n \rightarrow \{0, 1\}$$

and identify $\{0, 1\}$ with $\text{GF}(2)$. In this way, $\{0, 1\}^n$ becomes a vector space of dimension n . A Boolean function is called linear, if $f(x+y) = f(x) + f(y)$ for all $x, y \in \{0, 1\}^n$. It is a well known fact from linear algebra that f is linear iff there are scalars $\alpha_1, \dots, \alpha_n$ such that

$$f(x_1, \dots, x_n) = \sum_{\nu=1}^n \alpha_\nu x_\nu$$

Since $\{0, 1\}$ has only two elements, we can rephrase this as follows: A function f is linear iff there is an $S \subseteq \{1, \dots, n\}$ such that

$$f(x_1, \dots, x_n) = \sum_{\nu \in S} x_\nu.$$

Definition 4.1. Let $0 \leq \epsilon \leq 1$ and let \mathcal{S} be a set of Boolean functions.

1. Two Boolean functions $f, g : \{0, 1\}^n \rightarrow \{0, 1\}$ are ϵ -close, if

$$\Pr_{x \in \{0, 1\}^n} [f(x) = g(x)] \geq 1 - \epsilon.$$

2. f and g are ϵ -far from each other if they are not ϵ -close.
3. f is ϵ -close to \mathcal{S} if there is a $g \in \mathcal{S}$ such that f and g are ϵ -close.

If f and g are ϵ -close, then they agree on a fraction of at least $1 - \epsilon$ of all inputs $\{0, 1\}^n$. We call a function ϵ -close to linear, if it is ϵ -close to the set \mathcal{L} of all linear functions $\{0, 1\}^n \rightarrow \{0, 1\}$.

If f is ϵ -close to linear, then there is a set S such that

$$\Pr_x [f(x) = \sum_{\nu \in S} x_\nu] \geq 1 - \epsilon.$$

In particular, $f(x+y) = f(x) + f(y)$ holds whenever f agrees with the linear function $\sum_{\nu \in S} x_\nu$ on $x+y$, x , and y .

Exercise 4.1. *Estimate the probability that this happens!*

On the other hand, if $f(x+y) = f(x) + f(y)$ holds for many pairs x and y , can we conclude that f is close to linear? If this was true, then we get an efficient linearity test by just taking some random pair x and y and check whether $f(x, y) = f(x) + f(y)$. This is a result in the spirit of the PCP theorem: To check something, we just need to look at a constant number of randomly chosen values.

If f is linear, then we can find the set S by evaluating f at the unit vectors e_1, \dots, e_n . This does however not work if f is only close to linear, since the $f(e_i)$ need to agree with the linear function f is close to. So this “naive” approach does not work.

4.2 Fourier expansion of Boolean function

To show that a function f which satisfies $f(x+y) = f(x) + f(y)$ for a large fraction of pairs x and y is indeed close to linear, we will try to find the linear function that is closest to f . The Fourier expansion will be a tool for doing so.

Every function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be interpreted as a vector in $\{0, 1\}^{2^n}$, a table filled with its values. (We just take any ordering on $\{0, 1\}^n$ and fill the components of the vectors with the values $f(x)$ with respect to this ordering.) If it is clear from the context, we will call this vector again f .

Of course, all functions $\{0, 1\}^n \rightarrow \{0, 1\}$ form a GF(2) vector space of dimension 2^n . But we want to work over \mathbb{R} (or \mathbb{C}), since we need an inner product. We could embed $\{0, 1\}$ to \mathbb{R} by mapping 0 (in GF(2)) to 0 (in \mathbb{R}) and 1 (in GF(2)) to 1 (in \mathbb{R}) and consider our functions as functions $\{0, 1\}^n \rightarrow \mathbb{R}$. However, this simple translation does not respect addition ($1 + 1 = 0$ over GF(2) but $1 + 1 \neq 0$ over \mathbb{R}). Therefore, we change the representation of the Boolean functions: Boolean functions will be functions from $\{-1, 1\}^n \rightarrow \{-1, 1\}$ and we identify the Boolean value 0 (“false”) with 1 and the Boolean value 1 (“true”) with -1 . Note that this transformation is achieved by the mapping $x \mapsto (-1)^x$. Now we can view $\{-1, 1\}$ as a subset of \mathbb{R} : Addition in GF(2) becomes multiplication in \mathbb{R} . (This mapping is an isomorphism between the additive cyclic group GF(2) and the multiplicative cycle group $\{-1, 1\}$.)

Let $\ell : \{0, 1\}^n \rightarrow \{0, 1\}$ be a linear function corresponding to some set $S \subseteq \{1, \dots, n\}$. As a function $\{-1, 1\}^n \rightarrow \{-1, 1\}$, this function is given by $x \mapsto \prod_{\nu \in S} x_\nu$. We denote this function by χ_S . To avoid confusion we will not call this function a linear function but prefer to call it a *parity function*.

Definition 4.2. Let $f, g : \{-1, 1\}^n \rightarrow \mathbb{R}$. The correlation of f and g is

$$\langle f, g \rangle = \frac{1}{2^n} \sum_{x \in \{-1, 1\}^n} f(x)g(x) = \mathbb{E}_{x \in \{-1, 1\}^n} [f(x)g(x)].$$

The correlation is an inner product on the vector space of all functions $\{-1, 1\}^n \rightarrow \mathbb{R}$.

Exercise 4.2. Let $f, g : \{-1, 1\}^n \rightarrow \{-1, 1\}$. Prove the following facts:

1. $\langle f, g \rangle \in [-1, 1]$.
2. $\langle f, f \rangle = 1$.
3. f and g are ϵ -close iff $\langle f, g \rangle \geq 1 - 2\epsilon$.

Definition 4.3. Let $f : \{-1, 1\}^n \rightarrow \mathbb{R}$ and $S \subseteq \{1, \dots, n\}$. The fourier coefficient with respect to χ_S is

$$\hat{f}(S) = \langle f, \chi_S \rangle.$$

The fourier coefficient $\hat{f}(S)$ measures the closeness between f and the parity function χ_S .

Lemma 4.4. Let $S, T \subseteq \{1, \dots, n\}$. If $S \neq T$, then $\langle \chi_S, \chi_T \rangle = 0$, that is, χ_S and χ_T are orthogonal.

Proof. W.l.o.g. assume that there is an $i_0 \in S \setminus T$. We have

$$\begin{aligned} \langle \chi_S, \chi_T \rangle &= \sum_{x \in \{-1, 1\}^n} \prod_{i \in S} x_i \prod_{j \in T} x_j \\ &= \sum_{\substack{x \in \{-1, 1\}^n \\ x_{i_0} = 1}} \prod_{i \in S \setminus \{i_0\}} x_i \prod_{j \in T} x_j - \sum_{\substack{x \in \{-1, 1\}^n \\ x_{i_0} = -1}} \prod_{i \in S \setminus \{i_0\}} x_i \prod_{j \in T} x_j \\ &= 0. \quad \blacksquare \end{aligned}$$

Every Boolean function f fulfills $\langle f, f \rangle = 1$. Furthermore, there are 2^n functions χ_S , which are pairwise orthogonal. Since the dimension of the vector space of all functions $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$ is 2^n , we get the following result.

Theorem 4.5. The functions χ_S , $S \subseteq \{1, \dots, n\}$, form an orthonormal basis.

Theorem 4.6. For every function $f : \{-1, 1\}^n \rightarrow \mathbb{R}$,

$$f = \sum_{S \subseteq \{1, \dots, n\}} \hat{f}(S) \chi_S. \quad (4.1)$$

Proof. We can write $f = \sum_S \alpha_S \chi_S$. We have

$$\langle f, \chi_T \rangle = \left\langle \sum_S \alpha_S \chi_S, \chi_T \right\rangle = \sum_S \alpha_S \langle \chi_S, \chi_T \rangle = \alpha_S. \quad \blacksquare$$

The expression (4.1) is called the *Fourier expansion* of f . The construction works with any orthonormal basis.

Exercise 4.3. Let $f, g : \{-1, 1\}^n \rightarrow \mathbb{R}$. Prove Plancharel's Theorem:

$$\langle f, g \rangle = \sum_{S \subseteq \{1, \dots, n\}} \hat{f}(S) \hat{g}(S).$$

(An important special case is Parseval's identity: $\langle f, f \rangle = \sum_{S \subseteq \{1, \dots, n\}} \hat{f}(S)^2$.)

4.3 The BLR test

Next we show that if a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ fulfills $f(x + y) = f(x) + f(y)$ for many pairs x and y , then it is close to linear. The following test and its analysis is due to Blum, Luby, and Rubinfeld [BLR93].

Input: oracle access to a Boolean function $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$

Output: $\begin{cases} \text{accept} & \text{if } f \text{ is linear,} \\ \text{reject with probability } \geq \epsilon & \text{if } f \text{ is } \epsilon\text{-far from linear} \end{cases}$

1. Choose x and y uniformly at random from $\{-1, 1\}^n$.
2. Let $z = x \cdot y := (x_1 y_1, \dots, x_n y_n)$.
3. Query f on x , y , and z .
4. Accept iff $f(x)f(y)f(z) = 1$.

Note that $f(x)f(y)f(z) = 1$ is just the “translation” of $f(x) + f(y) = f(x + y)$.

Theorem 4.7. 1. If f is linear, then the BLR test accepts with probability 1.

2. If f is ϵ -far from linear, then the BLR test accepts with probability $< 1 - \epsilon$.

Lemma 4.8. For every function $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$,

$$\Pr_{x, y \in \{-1, 1\}^n} [\text{BLR test accepts } f] = \frac{1}{2} + \frac{1}{2} \sum_{S \subseteq \{1, \dots, n\}} \hat{f}(S)^3.$$

Proof. Define the following indicator variable

$$I(x, y, z) = \begin{cases} 1 & \text{BLR test accepts when } x \text{ and } y \text{ are chosen} \\ 0 & \text{otherwise} \end{cases}$$

We have $I = \frac{1}{2} + \frac{1}{2}f(x)f(y)f(z)$. Hence

$$\Pr_{x,y}[\text{BLR test accepts}] = \mathbb{E}_{x,y}[I] = \frac{1}{2} + \frac{1}{2} \mathbb{E}_{x,y}[f(x)f(y)f(z)]. \quad (4.2)$$

Replacing the function by its Fourier expansion, we get

$$\begin{aligned} \mathbb{E}_{x,y}[f(x)f(y)f(z)] &= \mathbb{E}_{x,y} \left[\left(\sum_S \hat{f}(S)\chi_S(x) \right) \left(\sum_T \hat{f}(T)\chi_T(y) \right) \left(\sum_U \hat{f}(U)\chi_U(z) \right) \right] \\ &= \mathbb{E}_{x,y} \left[\sum_{S,T,U} \hat{f}(S)\hat{f}(T)\hat{f}(U)\chi_S(x)\chi_U(y)\chi_T(z) \right] \\ &= \sum_{S,T,U} \hat{f}(S)\hat{f}(T)\hat{f}(U) \mathbb{E}_{x,y}[\chi_S(x)\chi_U(y)\chi_T(z)]. \end{aligned} \quad (4.3)$$

We have

$$\begin{aligned} \chi_S(x)\chi_T(y)\chi_U(z) &= \prod_{i \in S} x_i \prod_{j \in T} y_j \prod_{k \in U} z_k \\ &= \prod_{i \in S} x_i \prod_{j \in T} y_j \prod_{k \in U} x_k y_k \\ &= \prod_{i \in S \Delta U} x_i \prod_{j \in T \delta U} y_j, \end{aligned}$$

where $A \Delta B = A \setminus B \cup B \setminus A$ denotes the symmetric difference. Since the x_i and y_j are independent, we get

$$\mathbb{E}_{x,y}[\chi_S(x)\chi_U(y)\chi_T(z)] = \prod_{i \in S \Delta U} \mathbb{E}_{x,y}[x_i] \prod_{j \in T \Delta U} \mathbb{E}_{x,y}[y_j].$$

Since $\mathbb{E}[x_i] = 0$ and $\mathbb{E}[y_j] = 0$, the righthand side is zero, except when the product is empty. Then it equals 1. The product is empty iff $S \Delta U = T \Delta U = \emptyset$. But this means $S = T = U$. Therefore, by (4.2) and (4.3),

$$\mathbb{E}_{x,y}[\text{BLR test accepts } f] = \frac{1}{2} + \frac{1}{2} \sum_S \hat{f}(S)^3. \quad \blacksquare$$

Lemma 4.9. *If*

$$\Pr_{x,y \in \{-1,1\}^n}[\text{BLR test accepts } f] \geq 1 - \varepsilon,$$

then f is ε -close to linear.

Proof. By Lemma 4.8,

$$\frac{1}{2} + \frac{1}{2} \sum_S \hat{f}(S)^3 = \Pr_{x,y}[\text{BLR tests accepts } f] \geq 1 - \epsilon.$$

From this, we get

$$1 - 2\epsilon \leq \sum_S \hat{f}(S)^3 \leq \max_S \hat{f}(S) \cdot \underbrace{\sum_T \hat{f}(T)^2}_{=1}.$$

The sum on the righthand side is 1 by Parseval's identity and the fact that f is Boolean. Thus there is a set S_0 such that $\hat{f}(S_0) \geq 1 - 2\epsilon$. But $\hat{f}(S_0)$ is the correlation between f and χ_{S_0} . Therefore, f and χ_{S_0} are ϵ -close. ■

Proof of Theorem 4.7. The first statement is clear. For the second statement note that if the test accepted f with probability $\geq 1 - \epsilon$, then f would be ϵ -close to linear by Lemma 4.9. ■

5 PCP “light”

In this chapter, we will prove a “light” version of the PCP theorem, namely, $\text{NP} \subseteq \text{PCP}[\text{poly}(n), O(1)]$. The verifier V may use a polynomial number of random bits, thus the proof π can be exponentially long. Still V has to run in polynomial time.

5.1 The Walsh-Hadamard code

Definition 5.1. *The relative Hamming distance of two vectors $x, y \in \{0, 1\}^n$ is $\Delta(x, y) = \Pr_{i \in \{0, 1\}^n} [x_i \neq y_i]$.*

The relative Hamming distance is the number of places where x and y differ, normalized by the length n .

Definition 5.2. *For every $\delta \in [0, 1]$, a function $e : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is an error correcting code with distance δ , if*

$$\Delta(x, y) \geq \delta \quad \text{for all } x, y \in \{0, 1\}^n, x \neq y.$$

The image $\text{im}(e)$ of e is the set of code words.

For a (column) vector $x \in \{0, 1\}^n$, x^T denotes the transposed (row) vector. For two vectors $x, y \in \{0, 1\}^n$, $x^T \cdot y = \sum_{\nu} x_{\nu} y_{\nu}$ is the scalar product of x and y (modulo 2). The *Walsh-Hadamard code* is the function $\text{wh} : \{0, 1\}^n \rightarrow \{0, 1\}^{2^n}$ mapping $x \in \{0, 1\}^n$ to a vector $a \in \{0, 1\}^{2^n}$ that consists of the values $x^T \cdot z$ for all $z \in \{0, 1\}^n$. (To do this, we order the strings in $\{0, 1\}^n$ in a standard way, e.g., lexicographically.) Note that the code words of the Walsh-Hadamard code are the tables of all linear maps $\{0, 1\}^n \rightarrow \{0, 1\}$. Thus we can think of the code words as being linear functions.

Lemma 5.3. *For every $x \in \{0, 1\}^n$, $x \neq 0$,*

$$\Pr_{y \in \{0, 1\}^n} [x^T \cdot y = 0] = \frac{1}{2}.$$

Proof. W.l.o.g. $x_1 = 1$. For $y = (y_1, \dots, y_n) \in \{0, 1\}^n$,

$$x^T \cdot y = y_1 + \sum_{\nu=2}^n x_{\nu} y_{\nu}.$$

Thus, for every $(y_2, \dots, y_n) \in \{0, 1\}^n$, either $x^T \cdot (1, y_2, \dots, y_n) = 0$ and $x^T \cdot (0, y_2, \dots, y_n) = 1$ or vice versa. ■

Lemma 5.4. *The function wh is an error correcting code with distance $\frac{1}{2}$.*

Proof. The function wh is linear (over $\text{GF}(2)$). The set of code words is a linear space.¹ Take two code words $a = \text{wh}(x)$ and $b = \text{wh}(y)$. $\Delta(a, b)$ is the probability that the vector $a - b$ contains a 1. Note that $a - b$ is again a code word. Therefore to show that wh has distance $\frac{1}{2}$, it is sufficient to show that every non-zero code word c , the probability that c contains a 1 is at least $\frac{1}{2}$. But this follows from Lemma 5.3. ■

Error correcting codes $e : \{0, 1\}^n \rightarrow \{0, 1\}^m$ with some distance δ should not only exist, we want to be able to compute them. First of all, this means that the mapping e is efficiently computable, that is, in time polynomial in m (“efficient encoding”). Second, for every $a \in \{0, 1\}^m$, we should be able to find a $b \in \text{im}(e)$ such that $\Delta(a, b) < \delta/2$ if such a b exists. Note that in this case, b is unique (“efficient decoding”). Third, we want m to be as small as possible. We also do not just want to be an error correcting code for one n , but a family of codes for every n .

The function wh naturally extends to a families of functions. Encoding is easy. The parameter m , however, is exponential in n . Usual, this is bad. But for our purposes, this will be fine, since the Walsh-Hadamard code allows *local decoding*: For every $a \in \{0, 1\}^m$ that is ϵ -close to linear with $\epsilon < \frac{1}{4}$, there is a unique $b \in \text{im}(\text{wh})$ with $\Delta(a, b) < \frac{1}{4}$. Given a and an index $1 \leq i \leq m$, we can compute b_i , the i th bit of b , in time polynomial in n (and *not* $m!$) by just looking at a constant number of bits of a .

Input: oracle access to $a : \{0, 1\}^n \rightarrow \{0, 1\}$ that is ϵ -close to linear ($\epsilon < \frac{1}{4}$), index $x \in \{0, 1\}^n$

Output: $b(x)$ with b being the unique code word with $\Delta(a, b) < \frac{1}{4}$

1. Choose $y \in \{0, 1\}^n$ uniformly at random.
2. Let $z = x + y$.
3. Query $a(y)$ and $a(z)$.
4. Return $a(z - y)$.

Lemma 5.5. *If a is ϵ -close to linear, $\epsilon < \frac{1}{4}$, then with probability $\geq 1 - 2\epsilon$, the algorithm above returns $b(x)$.*

Proof. Since a is ϵ -close to b , the probability that $a(y) \neq b(y)$ for a randomly chosen y is $\leq \epsilon$. Furthermore, the probability that $a(z) \neq b(z)$ is also $\leq \epsilon$, since z is uniformly distributed, too. Therefore, the algorithm is correct. ■

¹Such a code is called a *linear code*.

Note that above, we cannot just return $a(x)$. This would only be correct with constant probability if we choose x at random! Our algorithm is however correct with constant probability for *all* x and the probability is only taken over the internal random bits of the algorithm.

5.2 Quadratic Equations over GF(2)

A quadratic equation over GF(2) in variables x_1, \dots, x_n is an equation of the form

$$\sum_{i=1}^n \sum_{j=1}^n \alpha_{i,j} x_i x_j = \beta.$$

with $\alpha_{i,j}, \beta \in \{0, 1\}$. Since $u_i^2 = u_i$ over GF(2), we implicitly allow linear terms. Furthermore, if the monomials $x_i x_j$ and $x_j x_i$ are both present (with coefficient 1), then they cancel each other out, but this is ok. A set of quadratic equations is satisfiable, if there is a $\{0, 1\}$ -valued assignment to the variables such that every equation is satisfied. **Quad-Eq** is the set of all systems of polynomial equations that are satisfiable.

Lemma 5.6. *Quad-Eq is NP-complete.*

Proof sketch. **Quad-Eq** is obviously in NP. For the hardness proof, one can reduce the circuit satisfiability problem CSAT to **Quad-Eq**. For every gate (including the inputs), we have a variable g . If g for instance computes the conjunction of two gates with variables g' and g'' , then we add the equation $g = g' \cdot g''$. For the output gate, we add the equation $g = 1$. ■

For two vectors $u, v \in \{0, 1\}^n$, their tensor product $u \otimes v$ is the $n \times n$ -matrix with entries $u_i v_j$, $1 \leq i \leq n$, $1 \leq j \leq n$. Note that $u \otimes v$ is a rank one matrix. We will identify $n \times n$ -matrices with vectors of length n^2 by rearranging the entries in some canonical way and vice versa. The tensor product can be used to “linearize” quadratic equations. Let $A = (\alpha_{i,j})$ be the matrix corresponding to a quadratic equation and think of it as a vector of length n^2 . For any $X \in \{0, 1\}^{n^2}$, the scalar product $A^T \cdot X$ is the value of the equation evaluated at X . But wait, not every X is a consistent assignment to the variables, only those that can be written as $x \otimes x$ for some $x \in \{0, 1\}^n$.

So **Quad-Eq** can be rewritten as follows: Given a matrix $A \in \{0, 1\}^{m \times n^2}$ and a vector $b \in \{0, 1\}^m$, is there a vector $X = x \otimes x$ for some $x \in \{0, 1\}^n$ such that $AX = b$.

5.3 The verifier for Quad-Eq

5.3.1 Testing tensor products

First of all, we present an algorithm for testing whether some vector X is of the form $x \otimes x$.

Input: Oracle access to $C = \text{wh}(X)$ and $c = \text{wh}(x)$ for some $X = \{0, 1\}^{n^2}$ and $x \in \{0, 1\}^n$

Output: $\begin{cases} \text{accept} & \text{if } X = x \otimes x \\ \text{reject with probability } \geq 1/4 & \text{otherwise} \end{cases}$

1. Choose random $y, z \in \{0, 1\}^n$.
2. Accept if $C(y \otimes z) = c(y) \cdot c(z)$

Lemma 5.7. *If $X = x \otimes x$, then the algorithm accepts with probability 1. Otherwise, it rejects with probability at least $\frac{1}{4}$.*

Proof. We have

$$\begin{aligned} c(y) \cdot c(z) &= (x^T \cdot y) \cdot (x^T \cdot z) \\ &= \left(\sum_{i=1}^n x_i y_i \right) \left(\sum_{j=1}^n x_j z_j \right) \\ &= \sum_{1 \leq i, j \leq n} (x_i x_j) (y_i z_j) \\ &= y^T \cdot (x \otimes x) \cdot z, \end{aligned}$$

(we view $x \otimes x$ as a $n \times n$ -matrix) and

$$C(y \otimes z) = \sum_{1 \leq i, j \leq n} X_{i,j} y_i z_j = y^T \cdot X \cdot z_j$$

(and X as well as an $n \times n$ -matrix). If $X = x \otimes x$, then $C(y \otimes z) = c(y) \cdot c(z)$ for all y and z . If $X \neq x \otimes x$, then there is a column of $X \neq x \otimes x$ in which $X \neq x \otimes x$ differ. Thus $y^T \cdot X \neq y^T \cdot (x \otimes x)$ with probability at least $\frac{1}{2}$ by Lemma 5.3. In this case, $y^T \cdot X \cdot z \neq y^T \cdot (x \otimes x) \cdot z$ again with probability $\frac{1}{2}$. Therefore, if $X \neq x \otimes x$, the algorithm rejects with probability $\geq \frac{1}{4}$. ■

5.3.2 Testing satisfiability

Next, we test whether some x encodes a satisfying assignment.

Input: $A \in \{0, 1\}^{m \times n^2}$, $b \in \{0, 1\}^m$, oracle access to $C = \text{wh}(x \otimes x)$ for some $x \in \{0, 1\}^n$.

Output: $\begin{cases} \text{accept} & \text{if } x \text{ is a satisfying assignment of } (A, b) \\ \text{reject with probability } \geq \frac{1}{2} & \text{otherwise} \end{cases}$

1. Choose $y \in \{0, 1\}^m$ uniformly at random.
2. Query $C(y^T A) = (x \otimes x) \cdot y^T A$ (we view $y^T A$ as a vector of length n^2).
3. Accept if $(x \otimes x) \cdot y^T A = y^T b$

Lemma 5.8. *If x satisfies (A, b) , then the algorithm above always accepts. Otherwise, it rejects with probability $\geq \frac{1}{2}$.*

Proof. We have

$$\begin{aligned} (x \otimes x) \cdot y^T A &= \sum_{1 \leq i, j \leq n} x_i x_j \sum_{k=1}^m y_k A_{k, (i, j)} \\ &= \sum_{k=1}^m y_k \sum_{1 \leq i, j \leq n} A_{k, (i, j)} x_i x_j = y^T \cdot A \cdot x \otimes x \end{aligned}$$

If $A \cdot x \otimes x = b$, then $y^t \cdot A \cdot x \otimes x = y^t b$ for all y . Otherwise $y^t \cdot A \cdot x \otimes x \neq y^t b$ with probability $\geq \frac{1}{2}$ by Lemma 5.3. ■

5.3.3 The verifier

Theorem 5.9. $\text{NP} \subseteq \text{PCP}[\text{poly}(n), O(1)]$.

Proof. We claim that the following verifier is a verifier for Quad-Eq. Since Quad-Eq is NP-complete, the claim follows.

Input: $A \in \{0, 1\}^{m \times n^2}$, $b \in \{0, 1\}^m$ oracle access to a proof $\pi \in \{0, 1\}^{2n^2 + 2n}$. π is interpreted as $C = \text{wh}(X)$ with $X \in \{0, 1\}^{n^2}$ and $c = \text{wh}(x)$ with $x \in \{0, 1\}^n$.

1. Run the BLR test on C and c . If any of these tests fails, then reject.
2. Test whether $X = x \otimes x$ using the tensor product test. If not, reject. (Use the self-correction property of the Walsh-Hadamard code to query $\text{wh}(X)$ and $\text{wh}(x)$.)
3. Test whether x satisfies (A, b) . If yes, accept. Otherwise, reject. (Use the self-correction property of the Walsh-Hadamard code to query $\text{wh}(X)$ and $\text{wh}(x)$.)

If $(A, b) \in \text{Quad-Eq}$, then we choose x as a satisfying assignment, $X = x \otimes x$, $C = \text{wh}(X)$, and $c = \text{wh}(x)$. By construction, all tests are passed and the self-correction algorithm always returns the true value. Therefore, the verifier accepts with probability 1.

Assume $(A, b) \notin \text{Quad-Eq}$. If C is 0.1-far from linear or c is 0.1-far from linear, then π is rejected with probability 0.1.

So we can assume that C and c are 0.1-close to linear. We do three queries in step 2 and one in step 3. If C and c are 0.1-close to linear, then the probability that all of them are correct is $\geq 1 - 4 \cdot 0.2 = 0.2$.

If $X \neq x \otimes x$, then the verifier will detect this in step 2 with probability $\geq \frac{1}{4}$. So proofs π with C and c being 0.1-close to linear and $X \neq x \otimes x$ are rejected with probability $0.2/4 = 0.05$.

If $X = x \otimes x$, then the verifier will detect in step 3 that x is not a satisfying assignment with probability at least $\frac{1}{2}$. Therefore, π is rejected with probability $0.2/2 = 0.1$.

In any case, the verifier will reject with probability ≥ 0.05 . We can amplify this probability by running the verifier a constant number of times.

■

6 Assignment Tester

6.1 Constraint graph satisfiability

A *constraint graph* G over some alphabet Σ is a directed¹ multigraph (V, E) together with a mapping $c : E \rightarrow \mathcal{P}(\Sigma)$. An assignment is a mapping $V \rightarrow \Sigma$. The assignment a satisfies the (constraint at) edge $e = (u, v)$ if $(a(u), a(v)) \in c(e)$. The *unsatisfiability value* of a is the number of constraints not satisfied by a divided by the number of constraints (edges). This value is denoted by $\text{UNSAT}_a(G)$. The unsatisfiability value of G is $\text{UNSAT}(G) = \min_a \text{UNSAT}_a(G)$.

Problem 6.1. Maximum Constraint graph satisfiability **Max-CGS** is the following problem:

Instances: constraint graphs $((V, E), \Sigma, c)$

Solutions: assignments $a : V \rightarrow \Sigma$

Measure: $(1 - \text{UNSAT}_a(G)) \cdot |E|$

Goal: max

Exercise 6.1. The following two statements are equivalent:

1. There is an $\epsilon > 0$ such that $\text{gap}(1 - \epsilon, 1)$ -Max-3-SAT is NP-hard.
2. There is an $\epsilon > 0$ such that $\text{gap}(1 - \epsilon, 1)$ -Max-CGS is NP-hard.

Thus, to prove the PCP-Theorem, we can show the NP-hardness of $\text{gap}(1 - \epsilon, 1)$ -Max-CGS instead of $\text{gap}(1 - \epsilon, 1)$ -Max-3-SAT. The former one has the advantage that it is easier to apply results from graph theory, in particular expander graphs, which we will introduce in the next chapter.

6.2 Assignment testers

Assignment testers provide a way to reduce the alphabet size of a constraint graph satisfiability problem. Every constraint over a large alphabet Σ_0 will be replaced by a graph over a small alphabet Σ . This will be needed in our proof of the PCP theorem.

¹We consider directed graphs here to allow asymmetric relations on the edges. By giving the edges a direction, we have a first and a second node. Alternatively, we could order the nodes globally. For any other purposes, we will ignore the direction of the edges completely.

Definition 6.2. Let G_0 be a graph with two nodes x and y , one edge $e_0 = (x, y)$ with a constraint c_0 over some (large) alphabet Σ_0 . An assignment tester over Σ is a deterministic algorithm that given G_0 outputs a constraint graph $G = ((V, E), \Sigma, c)$ such that the following holds:

1. There are two disjoint sets $X, Y \subseteq V$ and mappings $f_x : \Sigma_0 \rightarrow \Sigma^{|X|}$, $f_y : \Sigma_0 \rightarrow \Sigma^{|Y|}$, $g_x : \Sigma^{|X|} \rightarrow \Sigma_0$, $g_y : \Sigma^{|Y|} \rightarrow \Sigma_0$.
2. If (a, b) is an assignment that satisfies G_0 , then there is an assignment d to the nodes in $V \setminus (X \cup Y)$ such that $(f_x(a), f_y(b), d)$ satisfies G .
3. There is an $\epsilon > 0$ such that for every assignment $A = (a', b', d)$ to G with $\text{UNSAT}_A(G) \leq \epsilon$, $(g_x(a'), g_y(b'))$ satisfies G_0 .

Above, $(f_x(a), f_y(b), d) : V \rightarrow \Sigma$ the assignment that gives every node in X the corresponding value of $f_x(a)$, every node in Y the corresponding value of $f_y(b)$ and every other node the corresponding value of d . We do not care for running times, since we will apply the assignment tester only to constant size instances.

The maps f_x and f_y map a satisfying assignment of G_0 to an assignment of X and Y that can be extended to a satisfying assignment of G . The maps g_x and g_y map an assignment which satisfies many constraints of G to an assignment that satisfies G_0 .

Theorem 6.3. For every constraint graph G_0 as above, there exists an assignment tester.

Proof. Let G_0 be the given constraint graph. We identify Σ_0 with $\{0, 1\}^{k_0}$ where $k_0 = \lceil \log_2 |\Sigma_0| \rceil$. $\{0, 1\}^{k_0}$ might be larger than Σ_0 ; we assume that the constraint c_0 is not satisfied if one of the nodes x or y is assigned one of these surplus values. Let C be a boolean circuit on $2k_0$ inputs that computes c_0 viewed as a function $\{0, 1\}^{k_0} \times \{0, 1\}^{k_0} \rightarrow \{0, 1\}$.

Using the construction of Lemma 5.6, we construct an instance Q of **Quad-Eq** that is equivalent to C . Note that the reduction of Lemma 5.6 is parsimonious, that is, the number of satisfying assignments of C and Q are the same. The instance Q has three sets of variables, two of them correspond to the two inputs to c_0 , the third one to the auxiliary variables introduced by the reduction to Q .

Consider the verifier M for **Quad-Eq** constructed in Theorem 5.9. We modify it in such a way that the assignment to the variables stored in x will be given by three different Walsh-Hadamard codes, one for each of the three sets of variables in Q . M will need two additional BLR tests and whenever we want to query $\text{wh}(x)$, we make three queries to the three parts and add the answers (modulo 2). We assume that M has large enough rejection probability, e.g., $\geq \frac{1}{2}$.

Let $k \in O(1)$ be the number of queries made by M . We set $\Sigma = \{0, 1\}^k$. For every bit of the proof we will have one node. They are grouped into three sets X , Y , and Z . X and Y correspond to the two sets of input variables of Q , Z to the auxiliary variables in Q and to the variables of the Walsh-Hadamard code of the tensor product. An assignment to these nodes is interpreted as a Boolean assignment by just looking at the first component of Σ . For every random string r of M , we have an additional node v_r . v_r is connected to all the nodes in X , Y , and Z that correspond to bits queried by M with random string r . The value assigned to v_r is considered as the answers to the queries made by M with random string r . Consider an edge connecting v_r with some $u \in X \cup Y \cup Z$. Let s be the value assigned to v_r and t be the value assigned to u . The constraint on the edge (u, v_r) is satisfied by (s, t)

- M with random string r and answers to the queries as given by t accepts.
- The first bit of s is the same as the bit of t that corresponds to the query to u .

We claim that this yields an assignment tester. The mappings f_x and f_y map the assignments to x and y of G_0 to the Walsh-Hadamard code of x and y . By the construction of M , we can extend this to a satisfying assignment.

The mappings g_x and g_y map the assignments a' and b' to the nodes in X and Y to the unique strings a and b with $\Delta(\text{wh}(a), a') \leq \frac{1}{4}$ and $\Delta(\text{wh}(b), b') \leq \frac{1}{4}$ if a' and b' are $(\frac{1}{4} - \epsilon)$ -close to linear. Otherwise, g_x and g_y map a' and b' to anything. Assume we have an assignment A to the nodes of G with $\text{UNSAT}_A(G) \leq \frac{1}{3k}$. This means that for a fraction of $\frac{2}{3}$ of the v_r , all constraints on edges incident with v_r are fulfilled. But this means, that M with random string r and proof as given by A accepts. In particular, the acceptance probability of M on Q with proof given by A is $\geq \frac{2}{3}$. Therefore, Q is satisfiable and a' and b' have to be close to linear, because otherwise the rejection probability would be less than $\frac{1}{2}$. ■

Implementation details

- The assignment is split into three parts, two of length k_0 corresponding to the two nodes of the original instance G_0 and one of length ℓ corresponding to the auxiliary variable introduced in the reduction from CSAT to Quad-Eq.
- Therefore, instead of one string $c (= \text{wh}(x))$, we have three strings a' , b' , and d' , each of them interpreted as a Walsh-Hadamard code.
- If the verifier of Theorem 5.9 queries $c(y)$, where $y \in \{0, 1\}^{2k_0+\ell}$, the new verifier queries the three strings a' , b' , and d' at $y_1, y_2 \in \{0, 1\}^{k_0}$ and $y_3 \in \{0, 1\}^\ell$ respectively, where $y = (y_1, y_2, y_3)$, and add the results (modulo 2).
- We repeat the BLR test such that we can have rejection probability $< \frac{1}{2}$ only if the strings in the proof are all $(\frac{1}{4} - \epsilon)$ -close to linear.

7 Expander graphs

Throughout this chapter, we are considering undirected multigraphs $G = (V, E)$ with self-loops. The degree $d(v)$ of a node v is the number of edges that v belongs to. This particularly means that a node with a self-loop and no other edges has degree 1 (and not 2, which is a meaningful definition, too). This definition of degree will be very convenient in the following. A graph is called *d-regular* if $d(v) = d$ for all $v \in V$.

It is a well-known fact that for graphs without self-loops, the sum of the degrees of the nodes is twice the number of edges (proof by double-counting). With self-loops, the following bounds hold.

Fact 7.1. 1. We have $|E| \leq \sum_{v \in V} d(v) \leq 2|E|$.

2. If G is *d-regular*, then $|E| \leq d|V| \leq 2|E|$.

A *walk* in a graph $G = (V, E)$ is a sequence $(v_0, e_1, v_1, e_2, \dots, e_\ell, v_\ell)$ such that $e_\lambda = \{v_{\lambda-1}, v_\lambda\}$ for all $1 \leq \lambda \leq \ell$. v_0 is the start node, v_ℓ is the end node of the walk. Its length is ℓ . A walk can visit the same node or edge several times, i.e., it is allowed that $v_i = v_j$ or $e_i = e_j$ for some $i \neq j$.

A graph is *connected* if for all pairs of nodes u and v , there is a walk from u to v . The neighbourhood $N(v)$ of v is the set of all nodes u such that $\{v, u\} \in E$. In general, the t -neighbourhood is the set of all nodes u such that there is a walk from v to u of length t .

7.1 Algebraic graph theory

The *adjacency matrix* of G is the $|V| \times |V|$ -matrix

$$A = (a_{u,v})_{u,v \in V}$$

where $a_{u,v}$ is the number of edges between u and v . We will usually index the rows and columns by the nodes itself and not by indices from $\{1, \dots, |V|\}$. But we will assume that the nodes have some ordering, so that when we need it, we can also index the rows by $1, \dots, |V|$.

We will now apply tools from linear algebra to A in order to study properties of G . This is called *algebraic graph theory*. The book by Biggs [Big93] is an excellent introduction to this field. Everything you want to know about expander graphs can be found in [HLW06].

Because G is undirected, A is symmetric. Therefore, A has n real eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ and there is an orthonormal basis consisting of eigenvectors.

Lemma 7.2. *Let G be a d -regular graph with adjacency matrix A and eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.*

1. $\lambda_1 = d$ and $\mathbf{1}_n = (1, \dots, 1)^T$ is a corresponding eigenvector.
2. G is connected if and only if $\lambda_2 < d$.

Proof. We start with 1: Since G is d -regular,

$$d = d(v) = \sum_{u \in V} a_{v,u} \quad \text{for all } v$$

and

$$A \cdot \mathbf{1}_n = d \cdot \mathbf{1}_n.$$

Thus d is an eigenvalue and $\mathbf{1}_n$ is an associated eigenvector.

Let λ be any eigenvalue and b be an associated eigenvector. We can scale b in such a way that the largest entry of b is 1. Let this entry be b_v . Then

$$\lambda = \lambda \cdot b_v = \sum_{u \in V} a_{v,u} b_u \leq \sum_{u \in V} a_{v,u} = d.$$

Therefore, d is also the largest eigenvalue.

Now comes 2. “ \implies ”: Let b be an eigenvector associated with the eigenvalue d . As above, we scale b such that the largest entry is 1. Let b_v be this entry. We next show that for every node $u \in N(v)$, $b_u = 1$, too. Since G is connected, $b = \mathbf{1}_n$ follows by induction. But this means that d has multiplicity 1 and $\lambda_2 < d$.

$A \cdot b = d \cdot b$ implies

$$d = db_v = \sum_{u \in V} a_{v,u} b_u = \sum_{u \in N(v)} a_{v,u} b_u.$$

Since $b_u \leq 1$ for all u and since $d = \sum_{u \in N(v)} a_{v,u}$, this equation above can only be fulfilled if $b_u = 1$ for all $u \in N(v)$.

“ \impliedby ”: If the graph G is not connected, then $A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$. Therefore $(1, \dots, 1, 0, \dots, 0)$ and $(0, \dots, 0, 1, \dots, 1)$ (with the appropriate number of 1’s and 0’s) are linearly independent eigenvectors associated with d . ■

Let $\|\cdot\|$ denote the Euclidean norm of $\mathbb{R}^{|V|}$, that is $\|b\| = \sqrt{\sum_{v \in V} b_v^2}$.

Definition 7.3. *Let G be a graph with adjacency matrix A . Then*

$$\lambda(G) = \max_{b \perp \mathbf{1}_n} \frac{\|Ab\|}{\|b\|}.$$

Theorem 7.4. *Let G be a d -regular graph with adjacency matrix A and eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.*

1. $\lambda(G) = |\lambda_j|$ for some j .
2. $\lambda(G) = \max_{b \perp 1_n} \frac{\|Ab\|}{\|b\|}$ is attained for any eigenvector b associated with λ_j .
3. $\lambda(G) = \max\{|\lambda_2|, |\lambda_n|\}$.
4. $\lambda(G) \leq d$.

Proof. Let b be a vector for which the maximum is attained in the definition of $\lambda(G)$. W.l.o.g. let $\|b\| = 1$. Let c_1, \dots, c_n be an orthonormal basis consisting of eigenvectors of A . W.l.o.g. let $c_1 = 1_n$. Since b is orthogonal to 1_n , we have

$$b = \beta_2 c_2 + \dots + \beta_n c_n,$$

Since c_1, \dots, c_n is a orthonormal family,

$$1 = \|b\|^2 = \beta_2^2 + \dots + \beta_n^2.$$

Let λ_j be the eigenvalue c_j is associated with. We have

$$\begin{aligned} \lambda(G) &= \|Ab\| \\ &= \|\beta_2 A c_2 + \dots + \beta_n A c_n\| \\ &= \sqrt{(\beta_2 \lambda_2)^2 + \dots + (\beta_n \lambda_n)^2}. \end{aligned}$$

Since b is a vector for which the maximum is attained, β_j can only be nonzero for a λ_j whose absolute value is maximal among $\lambda_2, \dots, \lambda_n$.

It is an easy exercise to derive the statements 1–4 from this. ■

Exercise 7.1. Prove statements 1–4 of Theorem 7.4.

$\lambda(G)$ is also called the *second largest eigenvalue*. (More correctly, it should be called the second largest absolute value of the eigenvalues, but this is even longer.)

7.2 Edge expansion

Definition 7.5. Let G be a d -regular graph. The edge expansion $h(G)$ of G is defined as

$$h(G) = \min_{S \subseteq V: |S| \leq |V|/2} \frac{E(S, \bar{S})}{|S|}.$$

$E(S, \bar{S})$ is the set of all edges with one endpoint in S and one endpoint in \bar{S} . G is called an h -expander if $h(G) \geq h$.

Large edge expansion means that any set S has many neighbours that are not in S . This will be a very useful property. Families of expanders can be constructed in polynomial time, one construction is [RVW02]. We will prove it in some later chapter.

Theorem 7.6. *There are constants $d_0 \in \mathbb{N}$ and $h_0 > 0$ and a deterministic algorithm that given n constructs in time polynomial in n a d_0 -regular graph G_n with $h(G) > h_0$.*

Large edge expansion means small second largest eigenvalue and vice versa. We will need the following bound.

Theorem 7.7. *Let G be a d -regular graph. If $\lambda(G) < d^1$ then*

$$\lambda(G) \leq d - \frac{h(G)^2}{2d}.$$

To prove the theorem, it is sufficient to prove

$$h(G)^2 \leq 2d(d - \lambda) \tag{7.1}$$

$$h(G)^2 \leq 2d(d + \lambda) \tag{7.2}$$

by Theorem 7.4. The proofs of both inequalities are very similar, we will only show the first one. Let

$$B = dI - A$$

$$B' = dI + A$$

where I is the $n \times n$ -identity matrix. Let $f \in \mathbb{R}^n$. Think of f being a vector of weights or values on the nodes. Later, we will derive f from an eigenvector of A . In the following, a summation over “ $e = \{u, v\}$ ” is a sum over all edges in E with two end nodes and a summation over “ $e = \{v\}$ ” is a sum over all self-loops in E .

Lemma 7.8.

$$f^T B f = \sum_{e=\{u,v\}} (f_u - f_v)^2$$

$$f^T B' f \geq \sum_{e=\{u,v\}} (f_u + f_v)^2$$

Proof. We have

$$\begin{aligned} f^T B f &= \sum_{v \in V} d f_v^2 - f^T A f \\ &= \left(\sum_{e=\{u,v\}} (f_u^2 + f_v^2) + \sum_{e=\{v\}} f_v^2 \right) - \left(\sum_{e=\{u,v\}} 2 f_u f_v + \sum_{e=\{v\}} f_v^2 \right) \\ &= \sum_{e=\{u,v\}} (f_u - f_v)^2. \end{aligned}$$

¹We have to exclude bipartite graphs, which have $\lambda_n = -d$ but can have edge expansion > 0 . Our prove will break down if $\lambda_n = -d$, because $(d + \lambda)$ must not be zero when proving the counter part of (7.3).

The second inequality is proven in a similar manner. ■

To a given f , let

$$F = \sum_{e=\{u,v\}} |f_u^2 - f_v^2|.$$

Let $\beta_0 < \beta_1 < \dots < \beta_r$ be the different values that f attains. Let

$$U_j = \{u \in V \mid f_u \geq \beta_j\},$$

$$U'_j = \{u \in V \mid f_u \leq \beta_j\}$$

be the set of all nodes whose value f_u is at least or at most β_j , respectively.

Lemma 7.9.

$$F = \sum_{j=1}^r |E(U_j, \bar{U}_j)|(\beta_j^2 - \beta_{j-1}^2)$$

$$F = \sum_{j=0}^{r-1} |E(U'_j, \bar{U}'_j)|(\beta_{j+1}^2 - \beta_j^2)$$

Proof. Let $e = \{u, v\} \in E$ be an edge that is no self-loop. Assume that $f_u = \beta_i \geq \beta_j = f_v$. The contribution of e to F is $\beta_i^2 - \beta_j^2$. On the other hand, e crosses U_k and \bar{U}_k for $j \leq k \leq i - 1$. Thus the contribution of e to right-hand side of the first equation in the statement of the lemma is

$$(\beta_i^2 - \beta_{i-1}^2) + (\beta_{i-1}^2 - \beta_{i-2}^2 + \dots + (\beta_{j+1}^2 - \beta_j^2)) = \beta_i^2 - \beta_j^2.$$

Thus both sides of the equation are equal. The second equation is shown in a similar manner. ■

Lemma 7.10. *We have*

$$F \leq \sqrt{2d} \sqrt{f^T B f} \|f\|.$$

If $f(v) \leq 0$ for all v , then

$$F \leq \sqrt{2d} \sqrt{f^T B' f} \|f\|.$$

Proof. We have

$$\begin{aligned} F &= \sum_{e=\{u,v\}} |f_u^2 - f_v^2| \\ &= \sum_{e=\{u,v\}} |f_u - f_v| \cdot |f_u + f_v| \\ &\leq \sqrt{\sum_{e=\{u,v\}} (f_u - f_v)^2} \cdot \sqrt{\sum_{e=\{u,v\}} (f_u + f_v)^2} \\ &= \sqrt{f^T B f} \cdot \sqrt{\sum_{e=\{u,v\}} (f_u + f_v)^2} \end{aligned}$$

by the Cauchy–Schwartz and Lemma 7.8 We can bound the second factor by

$$\begin{aligned} \sqrt{\sum_{e=\{u,v\}} (f_u + f_v)^2} &\leq \sqrt{2 \sum_{e=\{u,v\}} (f_u^2 + f_v^2)} \\ &\leq \sqrt{2d \sum_{v \in V} f_v^2} \\ &\leq \sqrt{2d} \|f\|. \end{aligned}$$

The second inequality is proven in a similar manner. ■

Lemma 7.11. *Let $f_v \geq 0$ for all $v \in V$ or $f_v \leq 0$ for all $v \in V$. If $|\text{supp}(f)| \leq n/2$, then $F \geq h(G)\|f\|^2$.*

Proof. We only show the statement for $f_v \geq 0$, the other case is completely similar. Since $|\text{supp}(f)| \leq n/2$, we have $\beta_0 = 0$ and $|U_j| \leq n/2$ for $j > 0$. We have $|E(U_j, \bar{U}_j)| \geq h(G)|U_j|$. By Lemma 7.9,

$$\begin{aligned} F &= \sum_{j=1}^r |E(U_j, \bar{U}_j)| (\beta_j^2 - \beta_{j-1}^2) \\ &\geq h(G) \sum_{j=1}^r |U_j| (\beta_j^2 - \beta_{j-1}^2) \\ &= h(G) \sum_{j=1}^{r-1} \beta_j^2 \underbrace{(|U_j| - |U_{j+1}|)}_{=|\{v|f_v=\beta_j\}|} + \beta_r^2 |U_r| \\ &= h(G)\|f\|^2. \quad \blacksquare \end{aligned}$$

Finally, we will now prove (7.1) and (7.2). We only show (7.1), (7.2) is proven in the same manner. Let $\lambda < d$ be an eigenvector of A . $d - \lambda$ is an eigenvector of $B = dI - A$ and every eigenvector of A associated with λ is an eigenvector of B associated with $d - \lambda$. Let g be such an eigenvector. g is orthogonal to 1_n . We can assume that g has at most $n/2$ entries that are ≥ 0 , otherwise we consider $-g$ instead. We define

$$f_v = \begin{cases} g_v & \text{if } g_v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and $W = \text{supp}(f)$. By construction, $|W| \leq n/2$. For $v \in W$, we have

$$\begin{aligned} (Bf)_v &= df_v - \sum_{u \in V} a_{v,u} f_u \\ &= dg_v - \sum_{u \in W} a_{v,u} g_u \\ &\leq dg_v - \sum_{u \in V} a_{v,u} g_u \\ &= (d - \lambda)g_v. \end{aligned}$$

Since $f_v = 0$ for $v \notin W$, this implies

$$\begin{aligned} f^T Bf &= \sum_{v \in V} f_v (Bf)_v \\ &\leq (d - \lambda) \sum_{v \in V} f_v g_v \\ &\leq (d - \lambda) \sum_{v \in V} f_v^2 \\ &= (d - \lambda) \|f\|^2. \end{aligned} \tag{7.3}$$

By Lemmas 7.10 and 7.11,

$$h(G) \|f\|^2 \leq \sqrt{2d} \sqrt{f^T Bf} \|f\|.$$

Squaring this and exploiting the inequality before, we get

$$h(G)^2 \|f\|^4 \leq 2d \cdot f^T Bf \cdot \|f\|^2 \leq 2d(d - \lambda) \|f\|^4.$$

Because g is orthogonal to 1_n and nonzero, it has at least one entry > 0 . Therefore, $\|f\| > 0$ and we get

$$h(G)^2 \leq 2d(d - \lambda).$$

The second inequality is proven in the same manner.

8 Random walks on expanders

Consider the following method RW to generate a walk in a d -regular graph $G = (V, E)$.

Input: d -regular graph G , $\ell \in \mathbb{N}$

1. Randomly choose a vertex v_0 .
2. For $\lambda = 1, \dots, \ell$ choose $v_\lambda \in N(v_{\lambda-1})$ uniformly at random.
3. Return (v_0, \dots, v_ℓ) .

Let \mathcal{W}_ℓ be the set of all ℓ -walks in G . We have $|\mathcal{W}_\ell| = nd^\ell$, since a path is uniquely specified by its start node (n choices) and a vector $\{1, \dots, d\}^\ell$ which tells us which of the d edges of the current node is the next in the given walk. For this, we number the d edges incident with a node arbitrarily from 1 to d . It is now clear that method RW generates the walks according to the uniform distribution on RW.

Lemma 8.1. *Method RW returns each walk $W \in \mathcal{W}_\ell$ with a probability of $1/(nd^\ell)$. ■*

Instead of choosing a start node, we can choose a node in the middle. This modified method RW' also generates the uniform distribution on all walks of length ℓ .

Input: d -regular graph G , $\ell \in \mathbb{N}$, $0 \leq j \leq \ell$

1. Randomly choose a vertex v_j .
2. For $\lambda = j - 1, \dots, 0$ choose $v_\lambda \in N(v_{\lambda+1})$ uniformly at random.
3. For $\lambda = j + 1, \dots, \ell$ choose $v_\lambda \in N(v_{\lambda-1})$ uniformly at random.
4. Return (v_0, \dots, v_ℓ) .

In the following, let G be a d -regular graph with adjacency matrix A . Let $\tilde{A} = \frac{1}{d} \cdot A$ be the normalized adjacency matrix. \tilde{A} is *doubly stochastic*, i.e., all entries are nonnegative and all row sums and all column sums are 1. λ is an eigenvalue of \tilde{A} iff $d \cdot \lambda$ is an eigenvalue of A .

Let $x = (x_v)_{v \in V}$ be a probability distribution on the nodes v and consider x as an element of \mathbb{R}^n . If we now select a node v according to x , then select an edge $\{v, u\}$ ($u = v$ is allowed) incident with v uniformly at random, and then go to u , the probability distribution that we get is given by $\tilde{A} \cdot x$. Applying induction we get the following result.

Lemma 8.2. *Let x be a probability distribution on V . If we run method RW for ℓ steps and draw the start node according to x , then this induces a probability distribution on V given by $\tilde{A}^\ell x$. ■*

Let $F \subseteq E$ be a set of edges. We want to estimate the probability that a random walk of length j that starts in an edge of F ends in an edge of F , too.

To this aim, we first calculate the probability x_v that a random walk that starts with an edge of F starts in v . The distribution $x = (x_v)_{v \in V}$ is generated by the following process: First choose an edge $f \in F$ at random. Then choose one of its nodes uniformly at random as the start node (and the other node as the second node). Here it makes a difference whether f is a self-loop or not. We have

$$\begin{aligned} x_v &= \frac{1}{|F|} \cdot \left(\frac{1}{2} |\{e = \{u, v\} \mid e \in F, u \neq v\}| + |\{e = \{v\} \mid e \in F\}| \right) \\ &\leq \frac{d}{|F|} \end{aligned} \quad (8.1)$$

By symmetry, this is also the probability that v is the second node in a walk that starts with an edge in F .

Second we estimate the probability y_v that if we choose a random edge incident to v , we have chosen an edge in F . This is simply

$$\begin{aligned} y_v &= \frac{1}{d} \cdot |\{e = \{u, v\} \mid e \in F\}| \\ &= \frac{2|F|}{d} \cdot \frac{1}{2|F|} |\{e = \{u, v\} \mid e \in F\}| \\ &\leq \frac{2|F|}{d} \cdot x_v. \end{aligned} \quad (8.2)$$

Now the probability distribution on the nodes after performing a walk of length j that starts in F is given by $\tilde{A}^{j-1} x_v$. (Note that x_v is also the probability that v is the second node in the walk.) The probability that the $(j+1)$ th edge is in F is then given by

$$\langle y, \tilde{A}^{j-1} x \rangle \quad (8.3)$$

where $\langle \cdot, \cdot \rangle$ is the ordinary scalar product in \mathbb{R}^n .

To estimate (8.3), we will exploit the Cauchy-Schwartz inequality. For this, we need estimate $\|x\|$. $x_1 = \frac{1}{n} \mathbf{1}_n$ is an eigenvector of \tilde{A} associated with 1. Let $x^\perp = x - x_1$. x^\perp is orthogonal to x_1 , because

$$\langle \mathbf{1}_n, x^\perp \rangle = \sum_{v \in V} (x_v - 1/n) = 1 - 1 = 0.$$

$\tilde{A}^k x^\perp$ is also orthogonal to x_1 for every k , since

$$\langle \mathbf{1}_n, \tilde{A}^k x^\perp \rangle = \langle (\tilde{A}^k)^T \mathbf{1}_n, x^\perp \rangle = \langle \tilde{A}^k \mathbf{1}_n, x^\perp \rangle = \langle \mathbf{1}_n, x^\perp \rangle = 0.$$

Let $\tilde{\lambda} = \lambda(G)/d$. We have

$$\|\tilde{A}^{j-1} x^\perp\| \leq |\tilde{\lambda}|^{j-1} \|x^\perp\|$$

and

$$\begin{aligned} \|x^\perp\|^2 &= \|x - x_1\|^2 \\ &= \|x\|^2 - 2\langle x, x_1 \rangle + \|x_1\|^2 \\ &= \|x\|^2 - \frac{2}{n} \sum_{v \in V} x_v + \frac{1}{n} \\ &< \|x\|^2. \end{aligned}$$

By (8.1),

$$\|x\|^2 \leq \max_{v \in V} x_v \cdot \sum_{v \in V} x_v = \max_{v \in V} x_v \leq \frac{d}{|F|}.$$

Altogether, we have

$$\langle y, \tilde{A}^{j-1} x^\perp \rangle \leq \|y\| \cdot \|\tilde{A}^{j-1} x^\perp\| \leq \frac{2|F|}{d} \|x\| \cdot |\tilde{\lambda}|^{j-1} \|x\| \leq 2|\tilde{\lambda}|^{j-1}.$$

Finally, the probability that $e_{j+1} \in F$ can be bounded by

$$\begin{aligned} \langle y, \tilde{A}^{j-1} x \rangle &= \langle y, \tilde{A}^{j-1} x_1 \rangle + \langle y, \tilde{A}^{j-1} x^\perp \rangle \\ &\leq \langle y, x_1 \rangle + 2|\tilde{\lambda}|^{j-1} \\ &= \frac{1}{n} \sum_{v \in V} y_v + 2|\tilde{\lambda}|^{j-1} \\ &\leq \frac{2|F|}{dn} + 2|\tilde{\lambda}|^{j-1} \\ &\leq 2 \left(\frac{|F|}{|E|} + \left(\frac{\lambda}{d} \right)^{j-1} \right). \end{aligned}$$

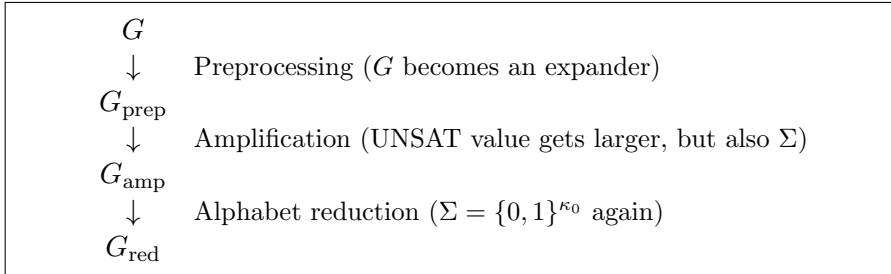
Lemma 8.3. *Consider a random walk on a d -regular graph $G = (V, E)$ starting with an edge from a set $F \subseteq E$. Then the probability that the $(j+1)$ th-edge of the walk is again in F is bounded by*

$$2 \left(\frac{|F|}{|E|} + \left(\frac{\lambda(G)}{d} \right)^{j-1} \right). \quad \blacksquare$$

If F does not contain any self-loops, then (8.1) can be bounded by $\frac{d}{2|F|}$ and we can get rid of the 2 in the estimate. Then this bound says that even after a logarithmic number of steps, the $(j+1)$ th edge is almost drawn at random.

9 The final proof

Finally, we can start with the proof of the PCP theorem. We begin with the observation that $\text{gap}(1 - \frac{1}{|E|}, 1)$ -Max-CGS is NP-hard (over the alphabet $\Sigma = \{0, 1\}^{\kappa_0}$ for all $\kappa_0 \geq 3$). Let G be a given constraint graph. We apply three procedures to G :



If we do this $O(\log |E|)$ times, then we bring the (relative) size of the gap from $\frac{1}{|E|}$ to constant and we are done.

9.1 Preprocessing

Throughout this chapter, d_0 and h_0 will be “global” constants that come out of the construction of a constant degree d_0 expander X_n with constant edge expansion h_0 , see Theorem 7.6. The size of the alphabet is $\{0, 1\}^{\kappa_0}$ where κ_0 is the number of queries made by our assignment tester.

Lemma 9.1. *Let $G = ((V, E), \Sigma, c)$ be a constraint graph. There is a constant $\gamma_1 > 0$ such that we can construct in polynomial time a $(d_0 + 1)$ -regular graph $G_1 = ((V_1, E_1), \Sigma : c_1)$ with $\text{size}(G_1) = O(\text{size}(G))$ and*

$$\gamma_1 \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_1) \leq \text{UNSAT}(G).$$

Proof. Let X_n be the expander from Theorem 7.6. G_1 is constructed as follows:

1. Replace each $v \in V$ by a copy Y_v of $X_{d(v)}$.
2. For each edge $\{u, v\} \in E$, insert an edge from Y_u to Y_v . Do this in such a way that every node of Y_v is incident with exactly one such extra edge. In this way, the resulting graph will be $(d_0 + 1)$ -regular.

3. Let E_{int} be the edges within the copies Y_v and E_{ext} be the edges between two different copies. For all $e \in E_{\text{int}}$, $c_1(e)$ is an equality constraint that is satisfied iff both nodes have the same value (“internal constraint”). For all $e \in E_{\text{ext}}$, $c_1(e)$ is the same constraint as the original edge has (“external constraint”).

We have $|V_1| \leq \sum_{v \in V} d(v) \leq 2|E|$ and $|E_1| \leq |V_1|(d_0 + 1) \leq 2|E|(d_0 + 1)$. Thus $\text{size}(G_1) = O(\text{size}(G))$.

Next, we show that $\text{UNSAT}(G_1) \leq \text{UNSAT}(G)$. Chose an assignment $\sigma : V \rightarrow \Sigma$ with $\text{UNSAT}(G) = \text{UNSAT}_\sigma(G)$ (i.e., an optimal assignment). We define $\sigma_1 : V_1 \rightarrow \Sigma$ by $\sigma_1(u) = \sigma(v)$ iff u belongs to $V(Y_v)$, the vertex set of the copy Y_v that replaces v . In this way, all internal constraints are fulfilled by construction. Every external constraint is fulfilled iff it was fulfilled under σ in G . Therefore,

$$\text{UNSAT}(G_1) \leq \text{UNSAT}_\sigma(G_1) \leq \text{UNSAT}(G),$$

where the second equation follows from the fact that $|E| \leq |E_1|$.

The interesting case is $\gamma \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_1)$. Let $\sigma_1 : V_1 \rightarrow \Sigma$ be an optimum assignment. We define $\sigma : V \rightarrow \Sigma$ by a majority vote: $\sigma(v)$ is the value $a \in \Sigma$ that is the most common among all values $\sigma_1(u)$ with $u \in V(Y_v)$. Ties are broken arbitrarily. Let $F \subseteq E$ be the set of all unsatisfied constraints under σ and $F_1 \subseteq E_1$ the set of all unsatisfied constraints under σ_1 . Let $S = \{u \in V(Y_v) \mid v \in V, \sigma_1(u) \neq \sigma(v)\}$ and $S^v = S \cap V(Y_v)$, i.e., all the looser nodes that voted for a different value for $\sigma(v)$. Let $\alpha := |F|/|E| = \text{UNSAT}_\sigma(G)$. We have

$$\alpha|E| = |F| \leq |F_1| + |S|,$$

since, if a constraint in F is not satisfied, then either the corresponding external constraint in $|F_1|$ is not satisfied or one of the nodes is a looser node.

Case 1: $|F_1| \geq \frac{\alpha}{2} \cdot |E|$. We have

$$\text{UNSAT}(G_1) = \frac{|F_1|}{|E_1|} \geq \frac{\alpha}{2} \cdot \frac{|E|}{|E_1|} = \frac{\alpha}{4(d_0 + 1)} \geq \frac{\text{UNSAT}(G)}{4(d_0 + 1)}.$$

Case 2: $|F_1| < \frac{\alpha}{2}|E|$. In this case, we have

$$\frac{\alpha}{2}|E| + |S| > |F_1| + |S| \geq \alpha|E|.$$

Thus $|S| \geq \frac{\alpha}{2}|E|$. Consider some $v \in V$ and let $S_a^v = \{u \in S^v \mid \sigma_1(u) = a\}$. We have $S^v = \bigcup_{a \neq \sigma(v)} S_a^v$. Because we took a majority vote, $|S_a^v| \leq \frac{1}{2}|V(Y_v)|$ for all $a \neq \sigma(v)$. As Y_v is an expander,

$$|E(S_a^v, \bar{S}_a^v)| \geq h_0 \cdot |S_a^v|,$$

where the complement is taken “locally”, i.e., $\bar{S}_a^v = V(Y_v) \setminus S_a^v$. Since we have equality constraints on all internal edges, all edges in $|E(S_a^v, \bar{S}_a^v)|$ are not satisfied. Thus,

$$\begin{aligned}
|F_1| &\geq \frac{1}{2} \sum_{v \in V} \sum_{a \neq \sigma(v)} |E(S_a^v, \bar{S}_a^v)| \\
&\geq \sum_{v \in V} \frac{1}{2} h_0 \cdot \sum_{a \neq \sigma(v)} |S_a^v| \\
&\geq \frac{1}{2} h_0 \sum_{v \in V} |S^v| \\
&= \frac{1}{2} h_0 |S| \\
&> \frac{\alpha}{4} h_0 |E|.
\end{aligned}$$

Thus

$$\begin{aligned}
\text{UNSAT}(G_1) &= \frac{|F_1|}{|E_1|} \\
&> \frac{\alpha h_0}{4} \cdot \frac{|E|}{|E_1|} \\
&\geq \frac{\alpha h_0}{8(d_0 + 1)} \\
&\geq \frac{h_0}{8(d_0 + 1)} \text{UNSAT}(G).
\end{aligned}$$

We set γ_1 to be the minimum of the constants in the two cases. ■

Lemma 9.2. *Let G be a d -regular constraint graph. We can construct in polynomial time a constraint graph G_2 such that*

- G_2 is $(d + d_0 + 1)$ -regular,
- every node of G_2 has a self loop,
- $\lambda(G_2) \leq d + d_0 + 1 - \frac{h_0^2}{2(d+d_0+1)}$,
- $\text{size}(G_2) = O(\text{size}(G))$,
- $\frac{d}{2+2(d_0+1)} \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_2) \leq \text{UNSAT}(G)$.

Proof. Assume that G has n nodes. We take the union of G and X_n (both graphs have the same node set) and attach to each node a self-loop. The edges from X_n and the self loops get trivial constraints that are always fulfilled. $G_2 = ((V, E_2), \Sigma, c_2)$ is clearly $d + d_0 + 1$ -regular.

We have $h(G) \geq h(X_n) \geq h_0$. Since G is connected ($\lambda_2 < d$) and not bipartite ($\lambda_n > -d$),

$$\lambda(G_2) \leq d + d_0 + 1 - \frac{h_0^2}{2(d + d_0 + 1)}.$$

Finally,

$$|E_2| = |E| + |E(X_n)| + n \leq |E| + (d_0 + 1)|V| \leq \frac{d + 2(d_0 + 1)}{d} |E|.$$

Thus the size increase is linear. Furthermore, the UNSAT value can at most shrink by this factor. ■

By combining these two lemmas, we get the following result.

Theorem 9.3. *There is a constant $\beta_{\text{prep}} > 0$ and $0 < \lambda < \delta$ such that for all constraint graphs G , we can construct in polynomial time a constraint graph G_{prep} over the same alphabet with*

- G_{prep} is δ -regular,
- every node in G_{prep} has a self-loop,
- $\lambda(G_{\text{prep}}) \leq \lambda$,
- $\text{size}(G_{\text{prep}}) = O(\text{size}(G))$,
- $\beta_{\text{prep}} \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_{\text{prep}}) \leq \text{UNSAT}(G)$.

We set $\delta = d + d_0 + 1$, $\beta_{\text{prep}} = \gamma \cdot \frac{d}{d+2(d_0+1)}$, and $\lambda = \delta - \frac{h_0^2}{2\delta}$.

9.2 Gap amplification

Definition 9.4. *Let $G = ((V, E), \Sigma, c)$ be a d -regular constraint graph such that every node has a self loop. Let $t \in \mathbb{N}$ be odd. The t -fold amplification product $G^t = ((V, E^t), \Sigma^{d^{t/2}}, c^t)$ is defined as follows:*

- For every walk W of length t from u to v , there is an edge $\{u, v\}$ in E^t . If there are several walks between u and v , we introduce several edges between u and v . But we disregard the directions of the walks, that is, for every walk W and its reverse, we put only one edge into E^t .
- An assignment $\hat{\sigma}$ maps every node to a vector from $\Sigma^{d^{t/2}}$. We index the entries with walks of length $t/2$ starting in v . (There are exactly $d^{t/2}$ such walks. Let W be such a walk and let u be the other end node. $\hat{\sigma}(v)_W$ is called “the opinion of v about u with respect to W ”. Since there might be many walks from v to u , v can have many opinions

about u . We will usually assume that nodes are not “schizophrenic”, i.e., that they always have the same opinion about u . In this case, we will also write $\hat{\sigma}(v)_u$ for the opinion of v about u .

- It remains to define c^t . Let $e = \{u, v\} \in E^t$ and $\hat{\sigma}$ be an assignment. Let G_e be the subgraph of G induced by $N_{t/2}(u) \cup N_{t/2}(v)$. $c^t(e)$ is satisfied by $\hat{\sigma}$ iff all opinions (of u and v) about every $x \in G_e$ are consistent and all constraints in G_e are satisfied. (Since G will be an expander, if one constraint of G “appears” in many constraints of G^t .)

If t is a constant, G^t is polynomial time computable from G and we have $\text{size}(G^t) = O(\text{size}(G))$. t will be odd in the following, whenever we write $t/2$ we mean t rounded down. A walk of length t can be decomposed into an initial walk of length $t/2$, one middle edge, and another walk of length $t/2$.

Theorem 9.5. *Let $\lambda < d$ be two constants, Σ an alphabet. There is a constant β_{amp} solely depending on λ , d , and $|\Sigma|$ such that for all d -regular constraint graphs G with self loops at every node and $\lambda(G) \leq \lambda$:*

1. $\text{UNSAT}(G^t) \geq \beta_{amp} \cdot \sqrt{t} \cdot \min\{\text{UNSAT}(G), \frac{1}{2t}\}$
2. $\text{UNSAT}(G) = 0 \Rightarrow \text{UNSAT}(G^t) = 0$.

Proof. We start with showing 2: Let σ be a satisfying assignment for G . We define $\hat{\sigma} : V \rightarrow \Sigma^{d^{t/2}}$ by setting $\hat{\sigma}(v)_W = \sigma(u)$ where W is a walk of length $t/2$ starting in v and u is the other end node of W . By construction, $\hat{\sigma}$ fulfills all constraints of G^t .

For 1, let $\hat{\sigma}$ be an optimum assignment for G^t . We can assume that there are not any schizophrenic nodes v because otherwise all constraints involving v are not satisfied and therefore, we cannot increase the UNSAT value by changing the assignment to v .

We will define an assignment σ with

$$\text{UNSAT}_{\hat{\sigma}}(G^t) \geq \Omega(\sqrt{t}) \cdot \min\{\text{UNSAT}_{\sigma}(G), \frac{1}{2t}\}.$$

σ is again defined by a majority vote. $\sigma(v)$ is the majority of all opinions of the nodes u that are reachable by a walk of length $t/2$ from v . (These are exactly the nodes that have an opinion about v .) If several paths go from v to u , then each paths contributes one opinion.

We choose an $F \subseteq E$ as large as possible such that all constraints in F are not satisfied by σ and $|F|/|E| \leq 1/t$. Then

$$\min\{\text{UNSAT}_{\sigma}(G), \frac{1}{2t}\} \leq \frac{|F|}{|E|} \leq \frac{1}{t}.$$

Let \mathcal{W}_t denote the set of all walks of length t .

Definition 9.6. $W = (v_0, e_1, v_1, \dots, v_t) \in \mathcal{W}_t$ is “hit at j ” if $e_j \in F$ and the opinion of v_0 about v_{j-1} and of v_t about v_j are equal to $\sigma(v_{j-1})$ and to $\sigma(v_j)$, respectively. (In particular, both nodes have an opinion about the corresponding node.)

If an edge is hit, then it is not satisfied and it is not satisfied because it is really not satisfied and not just because $\hat{\sigma}$ and σ were inconsistent.

We set $I = \{j \in \mathbb{N} \mid t/2 - \sqrt{t} + 1 < j < t/2 + \sqrt{t} + 1\}$, the set of “middle indices”. For a walk W , we set

$$N(W) = |\{j \in I \mid W \text{ is hit at } j\}|.$$

Let e_W be the edge in G^t corresponding to W . If $N(W) > 0$, then e_W is not satisfied by $\hat{\sigma}$, since e_j is not satisfied in G under σ and σ is consistent with $\hat{\sigma}$ on v_j and v_{j-1} . In formulas,

$$\begin{aligned} \Pr[N(W) > 0] &\leq \Pr_{\hat{e} \in E^t} [\hat{\sigma} \text{ does not satisfy } \hat{e}] \\ &= \text{UNSAT}_{\hat{\sigma}}(G^t) \\ &= \text{UNSAT}(G^t). \end{aligned}$$

We will show that $\Omega(\sqrt{t}) \frac{|F|}{|E|} \leq \Pr[N(W) > 0]$. This will finish the proof. In Lemma 9.8, we show that $\mathbb{E}[N(W)] \geq \Omega(\sqrt{t}) \frac{|E|}{|F|}$ and in Lemma 9.7, $\mathbb{E}[N(W)^2] \leq O(\sqrt{t}) \frac{|E|}{|F|}$. Now the claim follows from $\Pr[Z > 0] \geq \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]}$ for any nonnegative random variable Z . ■

Let

$$N_j(W) = \begin{cases} 1 & \text{if } W \text{ is hit in } j, \\ 0 & \text{otherwise.} \end{cases}$$

Then $\sum_{j \in I} N_j(W) = N(W)$.

Lemma 9.7. $\mathbb{E}[N(W)^2] \leq O(\sqrt{t}) \frac{|E|}{|F|}$.

Proof. By linearity of expectation,

$$\mathbb{E}[N(W)^2] = \sum_{i \in I} \mathbb{E}[N_i] + \sum_{i, j \in I, i \neq j} \mathbb{E}[N_i N_j].$$

We have

$$\sum_{i \in I} \mathbb{E}[N_i] = |I| \cdot \frac{|E|}{|F|}$$

and

$$\begin{aligned} \mathbb{E}[N_i N_j] &= \Pr[N_i N_j = 1] = \Pr[N_i = 1 \mid N_j = 1] \Pr[N_j = 1] \\ &= 2 \cdot \frac{|F|}{|E|} \cdot \left(\frac{|F|}{|E|} + \left(\frac{\lambda}{d} \right)^{|j-i|} \right) \end{aligned}$$

because

$$\begin{aligned} & \Pr[N_i(W) = 1 | N_j(W) = 1] \\ &= \Pr[\text{a random walk of length } |i - j + 1| \text{ ends in } F \mid \text{it started in } F] \\ &\leq 2 \left(\frac{|F|}{|E|} + \left(\frac{\lambda}{d} \right)^{|j-i|} \right) \end{aligned}$$

by Lemma 8.3. Therefore,

$$\begin{aligned} \sum_{i,j \in I, i \neq j} \mathbb{E}[N_i N_j] &\leq \sum_{i,j \in I, i \neq j} 2 \cdot \frac{|F|}{|E|} \cdot \left(\frac{|F|}{|E|} + \left(\frac{\lambda}{d} \right)^{|j-i|} \right) \\ &\leq 2|I|^2 \frac{|F|^2}{|E|^2} + 2|I| \frac{|F|}{|E|} + \sum_{i=0}^{|I|} \lambda^i = O(\sqrt{t}) \frac{|F|}{|E|}, \end{aligned}$$

because $|I| = \Theta(\sqrt{t})$ and $\frac{|F|}{|E|} \leq \frac{1}{t}$. ■

Lemma 9.8. *For all $j \in I$, $\Pr_{W \in \mathcal{W}_t}[N_j(W) = 1] = \Omega\left(\frac{|F|}{|E|}\right)$.*

Proof. Fix $j \in I$. We generate a walk $W = (v_0, e_1, v_1, \dots, v_t)$ uniformly at random by using the method RW' with parameter j . Then, the edge e_j is chosen uniformly at random. Furthermore, v_0 only depends on v_{j-1} and v_t only depends on v_t . Therefore,

$$\Pr_{W \in \mathcal{W}_t}[N_j = 1] = \frac{|F|}{|E|} pq$$

where $p = \Pr_{W \in \mathcal{W}_t}[\hat{\sigma}(v_0)_{v_{j-1}} = \sigma(v_{j-1})]$ and $q = \Pr_{W \in \mathcal{W}_t}[\hat{\sigma}(v_t)_{v_j} = \sigma(v_j)]$. We are done if we can show that p and q are constant. Since both cases are symmetric, we will only present a proof for p .

Let X_{j-1} be the random variable generated by the following process. We start in v_{j-1} and perform a random walk of length $j - 1$. Let u be the node that we reach. We output the opinion $\sigma(u)_{v_{j-1}}$ of u about v_{j-1} . If u has no opinion about v_{j-1} (this can happen, since j can be greater than $t/2$; but this will not happen too often) then we output some dummy value not in Σ . Obviously, $p = \Pr[X_{j-1} = \sigma(v_{j-1})]$.

If $j = t/2 + 1$, then we reach all the nodes that have an opinion about v_{j-1} and v_j , respectively. Since $\sigma(v_{j-1})$ and $\sigma(v_j)$ are chosen by a majority vote, $p, q \geq \frac{1}{|\Sigma|}$ in this case.

We will now show that for all $j \in I$, the probability $\Pr[X_{j-1} = \sigma(v_{j-1})]$ cannot differ by too much from this, in particular, it is $\Omega(1/|\Sigma|)$. The self loops will play a crucial role here, since they ensure that a random path with ℓ edges visit not more than $(1 - 1/d)\ell$ different nodes on the average.

Let $B(\ell, p)$ be a random variable with binomial distribution with $p = 1 - \frac{1}{d}$. Then $\Pr[B(\ell, p) = k]$ is the probability that a random walk of length ℓ in G uses k edges that are not self loops, or, if a node has more than one self loop, that are not the first self loop of this node. (We order the self loops in an arbitrary way.)

Let $Y_{i,j-1}$ be the random variable generated by the following process: Perform a random walk of length i starting in v_{j-1} not using the first self loop of any vertex and output the opinion of the node we reach about v_{j-1} . Then for $a \in \Sigma$,

$$\Pr[X_{j-1} = a] = \sum_{i=0}^{j-1} \Pr[B(j-1, p) = i] \Pr[Y_{i,j-1} = a]$$

By the Lemma 9.10,

$$\Pr[B(j-1, p) = i] \geq e^{-cd} \Pr[B(t/2, p) = i]$$

if $|j-1-t/2| \leq c\sqrt{t/2}$. Choose the constant c such that

$$\Pr[B(j-1, p) \leq p \cdot t/2 - c\sqrt{t/2}] + \Pr[B(j-1, p) \geq p \cdot t/2 + c\sqrt{t/2}] \leq \frac{1}{2|\Sigma|}.$$

Such a c exists by the fact that B is highly concentrated. (Use Fact 9.9 and $\Pr[B(n, p) \geq k] = \Pr[B(n, 1-p) \leq n-k]$.)

Assume that a maximizes $\Pr[X_{j-1} = a]$. Let $j \in I$. Then

$$\begin{aligned} \Pr[X_{j-1} = a] &\geq \sum_{i=p \cdot t/2 - c\sqrt{t/2}}^{p \cdot t/2 + c\sqrt{t/2}} \Pr[B(j-1, p) = i] \Pr[Y_{i,j-1} = a] \\ &\geq e^{-cd} \sum_{i=p \cdot t/2 - c\sqrt{t/2}}^{p \cdot t/2 + c\sqrt{t/2}} \Pr[B(t/2, p) = i] \Pr[Y_{i,j-1} = a] \\ &\geq e^{-cd} \left(\Pr[X_{t/2} = a] - \frac{1}{2|\Sigma|} \right) \\ &\geq e^{-cd}/2 \cdot \Pr[X_{t/2} = a]. \end{aligned}$$

The second inequality follows from Lemma 9.10 below. The third inequality holds because the sum over *all* i in the second row would be $\Pr[X_{t/2} = a]$ and we dropped indices with a total probability bounded by $1/(2|\Sigma|)$. The last inequality is due to the fact that a maximizes $\Pr[X_{t/2} = a]$. Therefore, $p \geq e^{-cd}/(2|\Sigma|)$. q is bounded in the same way. ■

Let $f(k, n, p)$ be the binomial distribution and $F(k, n, p)$ be its cumulative distribution function.

Fact 9.9. $F(k, n, p) \leq e^{-2\frac{-(np-k)^2}{n}}$ (by Hoeffding's inequality).

Let $p = 1 - \frac{1}{d}$. Let $0 \leq i \leq \sqrt{n}$. We have

$$\begin{aligned} f(k, n+i, p) &= p^k (1-p)^{n-k} \binom{n}{k} \cdot \frac{(n+1) \cdots (n+i)}{(n-k+1) \cdots (n-k+i)} (1-p)^i \\ &= f(k, n, p) \cdot \frac{(n+1) \cdots (n+i)}{(n-k+1) \cdots (n-k+i)} (1-p)^i. \end{aligned}$$

If $k = pn + j$ for some j , then

$$\frac{(1-p)^i (n+i)^i}{((1-p)n+i+j)^i} \leq \frac{(n+1) \cdots (n+i)}{(n-k+1) \cdots (n-k+i)} (1-p)^i \leq \frac{(1-p)^i n^i}{((1-p)n+j)^i}.$$

Assume that $0 \leq j \leq \sqrt{n}$. In this case, the right-hand side is ≤ 1 . We bound the lefthand side by

$$\begin{aligned} \frac{(1-p)^i (n+i)^i}{((1-p)n+i+j)^i} &\geq \left(1 - \frac{j}{(1-p)n+i+j}\right)^i \\ &\geq \left(1 - \frac{\sqrt{n}}{(1-p)n + \sqrt{n}}\right)^{\sqrt{n}} = \left(1 - \frac{d}{\sqrt{n}}\right)^{\sqrt{n}} \\ &\geq e^{-d} \end{aligned}$$

If $-\sqrt{j} \leq j < 0$, then 1 is a lower bound for the lefthand side and e^d is an upper bound for the righthand side.

If $\sqrt{n} \leq i < 0$, then we can prove the same bounds by exchanging the roles of n and $n+i$.

Lemma 9.10. For all $-\sqrt{n} \leq i, j \leq \sqrt{n}$,

$$e^{-d} \leq \frac{f(k, n, p)}{f(k, n+i, p)} \leq e^d$$

where $k = pn + j$ and $p = 1 - \frac{1}{d}$.

If we replace the bounds $-c\sqrt{n} \leq i, j \leq c\sqrt{n}$ for some constant c , then e^{-d} and e^d are replaced by e^{-cd} and e^{cd} .

9.3 Alphabet reduction

In the last section, we increased the UNSAT value of the constraint graph but also enlarged the alphabet. To apply the construction iteratively, we need that in the end, the alphabet is again $\Sigma = \{0, 1\}^{\kappa_0}$. This is achieved by the procedure in this section.

Lemma 9.11. *There is a constant β_{red} such that for all constraint graphs $G = ((V, E), \hat{\Sigma}, c)$ we can construct in polynomial time a constraint graph $G_{red} = ((V', E'), \{0, 1\}^3, c')$ such that*

1. $\text{size}(G_{red}) \leq O(\text{size}(G))$ where the constant only depends on $|\hat{\Sigma}|$,
2. $\beta_{red} \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_{red}) \leq \text{UNSAT}(G)$.

Proof. Let $k = |\hat{\Sigma}|$. We replace every edge $e = (x, y)$ by an assignment tester $G_e = ((V_e, E_e), \{0, 1\}^{k_0}, c_e)$ (see Theorem ??). The graph G_{red} is the union of all these assignment testers. The nodes X that are used to represent $x \in V$ in G_e , are shared by all assignment tester corresponding to an edge that contains x . The constraints of each edge in G_{pre} are the constraints of the G_e . We can assume that each G_e has the same number of edges, say, r . Thus G_{pre} has $r|E|$ edges. Note that the functions f_x, f_y, g_x , and g_y only depend on the set X or Y , respectively, hence if e and e' share the node x , they use the same function f_x and g_x , so there is no consistency issue.

Each assignment tester is a constant size graph whose size only depends on $|\hat{\Sigma}|$. This immediately yields the upper bound on the size of G_{pre} .

For the second statement of theorem, consider an optimal assignment σ of G . We construct an assignment σ' for G_{red} as follows: If σ satisfies the constraint c_e , then, by the properties of assignment testers, we can extend σ in such a way that all constraints of G_e are satisfied. If σ does not satisfy c_e , then we extend σ in any way. In the worst case, no constraints of G_e are satisfied. Thus for every constraint satisfied in G , at least r constraints are satisfied in G_{pre} . Thus

$$\text{UNSAT}(G_{red}) \leq \frac{r \cdot |E| \cdot \text{UNSAT}(G)}{|E'|} = \text{UNSAT}_\sigma(G) = \text{UNSAT}(G).$$

For the other inequality, let σ' be an optimum assignment for G_{red} . We construct an assignment σ for G by using the functions g_x for every $x \in V$. Assume that σ does not satisfy e . Then σ' does not satisfy at least an ϵ fraction of the constraints of G_e by the properties of an assignment tester. Since σ' is an optimum assignment, $\epsilon \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_{pre})$ ■

9.4 Putting everything together

If we put together the constructions of the three previous sections, we get the following result.

Lemma 9.12. *There are constants $C > 0$ and $1 > a > 0$ such that for every constraint graph G over the alphabet $\Sigma = \{0, 1\}^3$, we can construct a constraint graph G' over the same alphabet in polynomial time such that*

1. $\text{size}(G') \leq C \cdot \text{size}(G)$,

2. If $\text{UNSAT}(G) = 0$, then $\text{UNSAT}(G') = 0$,
3. $\text{UNSAT}(G') \geq \min\{2 \cdot \text{UNSAT}(G), a\}$.

Proof. We start with G , make it an expander, then amplify the gap (the value t is yet to choose) and finally reduce the alphabet. It is clear that if we choose t to be a constant, then the first two statements are fulfilled.

It remains to choose t in such a way that the third statement is fulfilled. We have

$$\begin{aligned} \text{UNSAT}(G') &\geq \beta_{\text{red}} \cdot \beta_{\text{amp}} \cdot \sqrt{t} \cdot \min\{\text{UNSAT}(G_{\text{pre}}), \frac{1}{2t}\} \\ &\geq \beta_{\text{red}} \cdot \beta_{\text{amp}} \cdot \sqrt{t} \cdot \min\{\beta_{\text{pre}} \cdot \text{UNSAT}(G), \frac{1}{2t}\} \end{aligned}$$

If we now set $t = 4 \left(\frac{1}{\beta_{\text{pre}} \beta_{\text{amp}} \beta_{\text{red}}} \right)^2$, we get

$$\text{UNSAT}(G') \geq \min\{2 \cdot \text{UNSAT}(G), a\}$$

with $a = \frac{\beta_{\text{pre}} \beta_{\text{amp}}^2 \beta_{\text{red}}^2}{4}$. ■

With this lemma, the proof of the PCP theorem follows easily. We start with the observation that the decision version of constraint graph satisfaction is NP-complete, i.e., $\text{gap}(1 - 1/|E|, 1)\text{-Max-CGS}$ is NP-hard. Let G be an input graph. If we now apply the above lemma $\log |E|$ times, we get an graph G' that can be computed in time polynomial in $\text{size}(G)$ with the property that

$$\text{UNSAT}(G') \geq \min\{2^{\log |E|} \cdot \frac{1}{|E|}, a\} = a$$

is constant. Thus we have a reduction from $\text{gap}(1 - 1/|E|, 1)\text{-Max-CGS}$ to $\text{gap}(1 - a, 1)\text{-Max-CGS}$. In particular, the latter problem is also NP-complete. But this is equivalent to the statement of the PCP theorem.

10 Explicit constructions of expanders

We call a family of (multi)graphs $(G_n)_{n \in \mathbb{N}}$ a family of d -regular λ -expanders if

1. G_n has n nodes
2. G_n is d -regular
3. $\lambda(G_n) \leq \lambda$

for all n . Here, d and λ are constants.

The family is called *explicit* if the function

$$1^n \rightarrow G_n$$

is polynomial time computable. It is called *strongly explicit* if

$$(n, v, i) \mapsto \text{the } i\text{th neighbour of } v \text{ in } G_n$$

is polynomial time computable. Here the input and output size is only $O(\log n)$, so the algorithm runs in time only $\text{poly}(\log n)$. In our case, it is also possible to return the whole neighbourhood, since d is constant.

Let G be a d -regular graph with adjacency matrix A . In this chapter, it will be very convenient to work with the normalized adjacency matrices $\tilde{A} = \frac{1}{d}A$. These matrices are also called random walk matrices, since they describe the transition probabilities of one step of a random walk. $\tilde{\lambda}(G)$ is the second largest (absolute value of an) eigenvalue of \tilde{A} . Obviously, $\tilde{\lambda}(G) = \lambda(G)/d$.

We will now describe three graph transformations. One of them increases the number of nodes. This will be used to construct larger expanders from smaller ones. The second one will reduce the degree. This is used to keep the degree of our family constant. An the last one reduces the second largest eigenvalue. This is needed to keep $\lambda(G)$ below λ .

10.1 Matrix products

Let G be a d -regular graph with normalized adjacency matrix \tilde{A} . The k -fold *matrix product* G^k of G is the graph given by the normalized adjacency matrix \tilde{A}^k . This transformation is also called path product, since there is an edge between u and v in G^k if there is path of length k in G between u and v .

It is obvious that the number of nodes stays the same and the degree becomes d^k .

Lemma 10.1. $\tilde{\lambda}(G^k) = \tilde{\lambda}(G)^k$ for all $k \geq 1$.

Proof. Let x be an eigenvector of \tilde{A} associated with the eigenvalue λ such that $\lambda = \tilde{\lambda}(G)$. Then $\tilde{A}^k x = \lambda^k x$ (induction in k). Thus $\tilde{\lambda}(G^k) \geq \lambda^k$. It cannot be larger, since otherwise $\tilde{\lambda}(G) > \lambda$. ■

Matrix product

	nodes	degree	$\tilde{\lambda}(G)$
G	n	d	λ
G^k	n	d^k	λ^k

Given oracle access to the neighbourhoods of G , that is, we may ask queries “Give me a list of all neighbours of v !”, we can compute the neighbourhood of a node v in G^k in time $O(d^k \log n)$ by doing a breadth first search starting in v . From v , we can reach at most d^k vertices and the description size of a node is $O(\log n)$.

10.2 Tensor products

Let G be a d -regular graph with n nodes and normalized adjacency matrix \tilde{A} and let G' be a d' -regular graph with n' nodes and normalized adjacency matrix \tilde{A}' . The *tensor product* $G \otimes G'$ is the graph given by the normalized adjacency matrix $\tilde{A} \otimes \tilde{A}'$. Here $\tilde{A} \otimes \tilde{A}'$ denotes the Kronecker product of the two matrices, which is given by

$$\tilde{A} \otimes \tilde{A}' = \begin{pmatrix} a_{1,1}\tilde{A}' & \dots & a_{1,n}\tilde{A}' \\ \vdots & \ddots & \vdots \\ a_{n,1}\tilde{A}' & \dots & a_{n,n}\tilde{A}' \end{pmatrix},$$

where $A = (a_{i,j})$.

The new graph has nn' nodes and its degree is dd' .

Lemma 10.2. Let A be a $m \times m$ -matrix and B be a $n \times n$ -matrix with eigenvalues $\lambda_1, \dots, \lambda_m$ and μ_1, \dots, μ_n . The eigenvalues of $A \otimes B$ are $\lambda_i \mu_j$, $1 \leq i \leq m$, $1 \leq j \leq n$.

Proof. Let x be an eigenvector of A associated with the eigenvalue λ , and y be an eigenvector of B associated with the eigenvalue μ . Let $z := x \otimes y$ be the vector

$$\begin{pmatrix} x_1 y \\ \vdots \\ x_n y \end{pmatrix}.$$

where $x = (x_i)$. z is an eigenvector of $A \otimes B$ associated with $\lambda\mu$:

$$\begin{aligned} A \otimes B \cdot z &= \begin{pmatrix} a_{1,1}x_1By + \cdots + a_{1,m}x_mB_y \\ \vdots \\ a_{m,1}x_1By + \cdots + a_{m,m}x_mB_y \end{pmatrix} \\ &= \mu \cdot \begin{pmatrix} (a_{1,1}x_1 + \cdots + a_{1,m}x_m)y \\ \vdots \\ (a_{m,1}x_1 + \cdots + a_{m,m}x_m)y \end{pmatrix} \\ &= \lambda\mu \cdot \begin{pmatrix} x_1y \\ \vdots \\ x_my \end{pmatrix} \\ &= \lambda\mu z. \end{aligned}$$

These are all eigenvalues, since one can show that if x_1, \dots, x_m and y_1, \dots, y_n are bases, then $x_i \otimes y_j$, $1 \leq i \leq m$, $1 \leq j \leq n$, form a basis, too. ■

From the lemma, it follows that $\tilde{\lambda}(G \otimes G') = \max\{\tilde{\lambda}(G), \tilde{\lambda}(G')\}$, since $1 \cdot \tilde{\lambda}(G')$ and $\tilde{\lambda}(G) \cdot 1$ are eigenvalues of $\tilde{A} \otimes \tilde{A}'$, but the eigenvalue $1 \cdot 1$ is excluded in the definition of $\tilde{\lambda}(G \otimes G')$.

Tensor product

	nodes	degree	$\tilde{\lambda}(G)$
G	n	d	λ
G'	n'	d'	λ'
$G \otimes G'$	nn'	dd'	$\max\{\lambda, \lambda'\}$

Given oracle access to the neighbourhoods of G and G' , we can compute the neighbourhood of a node v in $G \otimes G'$ in time $O(d^2 \log \max\{n, n'\})$. (This assume that from the names of the nodes v in G and v' in G' we can compute in linear time a name of the node that corresponds to $v \otimes v'$.)

10.3 Replacement product

Let G be a D -regular graph with n nodes and adjacency matrix A and H be a d -regular graph with D nodes and adjacency matrix B . The replacement product $G \textcircled{R} H$ is defined as follows:

- For every node v of G , we have one copy H_v of H .
- For every edge $\{u, v\}$ of G , there are d parallel edges between node i in H_u and node j in H_v where v is the i th neighbour of u and u is the j th neighbour of v .

We assume that the nodes of H are the number from 1 to D and that the neighbours of each node of G are ordered. Such an ordering can for instance be induced by an ordering of the nodes of G .

We can think of $G \circledast H$ of having an inner and an outer structure. The inner structures are the copies of H and the outer structure is given by G . For every edge of G , we put d parallel edges into $G \circledast H$. This ensures that when we choose a random neighbour of some node v , the probability that we stay in H_v is the same as the probability that we go to another H_u . In other words, with probability $1/2$, we perform an inner step and with probability $1/2$, we perform an outer step. The normalized adjacency matrix of $G \circledast H$ is given by

$$\frac{1}{2}\hat{A} + \frac{1}{2}I \otimes B,$$

where I is the $n \times n$ -identity matrix. The $nD \times nD$ -matrix \hat{A} is defined as follows: Think of the rows and columns labeled with pairs (v, j) , v is a node of G and j is a node of H . Then there is a 1 in the position $((u, i), (v, j))$ if v is the i th neighbour of u and u is the j th neighbour of v . \hat{A} is a permutation matrix.

Obviously, $G \circledast H$ has nD nodes and it is $2d$ -regular.

Excursus: Induced matrix norms

For a norm $\|\cdot\|$ on \mathbb{R}^n , the *induced matrix norm* on $\mathbb{R}^{n \times n}$ is defined by

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|.$$

It is a norm that is subadditive and submultiplicative. By definition, it is compatible with the vector norm, that is,

$$\|Ax\| \leq \|A\| \cdot \|x\|.$$

It is the “smallest” norm that is compatible with the given vector norm.

For the Euclidian norm $\|\cdot\|_2$ on \mathbb{R}^n , then induced norm is the so-called *spectral norm*, the square root of the largest of the absolute values of the eigenvalues of $A^H A$. If A is symmetric, then this is just the largest of the absolute values of the eigenvalues of A . In particular,

$$\lambda(G) \leq \|A\|_2.$$

If A is symmetric and doubly stochastic, then $\|A\|_2 \leq 1$.

Lemma 10.3. *If $\tilde{\lambda}(G) \leq 1 - \epsilon$ and $\tilde{\lambda}(H) \leq 1 - \delta$, then $\tilde{\lambda}(G \circledast H) \leq 1 - \epsilon\delta^2/24$.*

Proof. By Bernoulli's inequality, it is sufficient to show that $\tilde{\lambda}(G \circledast H)^3 \leq 1 - \epsilon \delta^2/8$. Since $\tilde{\lambda}(G \circledast H)^3 = \tilde{\lambda}((G \circledast H)^3)$, we analyze the threefold matrix power of $G \circledast H$. Its normalized adjacency matrix is given by

$$\left(\frac{1}{2} \hat{A} + \frac{1}{2} I \otimes \tilde{B} \right)^3. \quad (10.1)$$

\hat{A} and $I \otimes \tilde{B}$ are doubly stochastic, so their spectral norm is bounded by 1. Since the spectral norm is submultiplicative, we can expand (10.1) into

$$\begin{aligned} &= \frac{1}{8} \left(\text{sum of seven matrices of spectral norm} \leq 1 + (I \otimes \tilde{B}) \hat{A} (I \otimes \tilde{B}) \right) \\ &= \frac{7}{8} M + \frac{1}{8} \underbrace{(I \otimes \tilde{B}) \hat{A} (I \otimes \tilde{B})}_{=:(*)} \end{aligned}$$

with $\|M\| \leq 1$. By Exercise 10.1, we can write $\tilde{B} = (1 - \delta)C + \delta J$ with $\|C\| \leq 1$. Thus

$$\begin{aligned} (*) &= (I \otimes (1 - \delta)C + I \otimes \delta J) \hat{A} (I \otimes (1 - \delta)C + I \otimes \delta J) \\ &= (1 - \delta^2)M' + \delta^2(I \otimes J) \hat{A} (I \otimes J) \end{aligned}$$

with $\|M'\| \leq 1$. A direct calculation shows that

$$(I \otimes J) \hat{A} (I \otimes J) = A \otimes J'$$

where the entries of J' are all equal to $1/D^2$. Thus, the second largest eigenvalue of

$$\begin{aligned} \lambda((I \otimes J) \hat{A} (I \otimes J)) &= \lambda(A \otimes J') \leq \lambda(\tilde{A}) \\ &= \lambda(A) \cdot \|J'\| = \lambda(A)/D = \lambda(\tilde{A}). \end{aligned}$$

Hence,

$$\left(\frac{1}{2} \hat{A} + \frac{1}{2} I \otimes \tilde{B} \right)^3 = (1 - \frac{\delta^2}{8})M'' + \frac{\delta^2}{8}(A \otimes J')$$

with $\|M''\| \leq 1$ and

$$\begin{aligned} \lambda \left(\frac{1}{2} \hat{A} + \frac{1}{2} I \otimes \tilde{B} \right)^3 &\leq 1 - \frac{\delta^2}{8} + \frac{\delta^2}{8}(1 - \epsilon) \\ &= 1 - \frac{\delta^2 \epsilon}{8}, \end{aligned}$$

because λ is subadditive (i.e. $\lambda(A+B) \leq \lambda(A) + \lambda(B)$) and $\lambda(M'') \leq \|M''\|$.

■

The only term in the analysis that we used was the $(I \otimes \tilde{B}) \hat{A} (I \otimes \tilde{B})$ term. This corresponds to doing an “inner” step in H , then an “outer step” in G and again an “inner” step in H . The so-called *zig-zag* product is a product similar to the replacement product that only allows such steps.

Exercise 10.1. Let A be the normalized adjacency matrix of a d -regular λ -expander. Let

$$J = \begin{pmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \ddots & \vdots \\ \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix}.$$

Then

$$A = (1 - \lambda)J + \lambda C$$

for some matrix C with $\|C\| \leq 1$.

Replacement product

	nodes	degree	$\tilde{\lambda}(G)$
G	n	D	$1 - \epsilon$
H	D	d	$1 - \delta$
$G \otimes H$	nD	$2d$	$1 - \epsilon\delta^2/24$

Given oracle access to the neighbourhoods of D and H , we can compute the neighbourhood of a node v in $G \otimes G'$ in time $O((D + d) \log n)$. (This assumes that the oracle gives us the neighbourhoods in the same order than the one used when building the replacement product.)

10.4 Explicit construction

We first construct a family of expanders (G_m) such that G_m has c^m nodes. In a second step (Exercise!), we will show that we can get expanders from G_m of all sizes between $c^{m-1} + 1$ and c^m . The constants occurring in the proof are fairly arbitrary, they are just chosen in such a way that the proof works. We have taken them from the book by Arora and Barak.

For the start, we need the following constant size expanders. Since they have constant size, we do not need a constructive proof, since we can simply enumerate all graphs of the particular size and check whether they have the mentioned properties.

Exercise 10.2. For large enough d , there are

1. a d -regular 0.01 -expander with $(2d)^{100}$ nodes.
2. a $2d$ -regular $(1 - \frac{1}{50})$ -expander with $(2d)^{200}$ nodes

We now construct the graphs G_k inductively:

1. Let H be a d -regular 0.01 -expander with $(2d)^{100}$ nodes.
2. Let G_1 be a $2d$ -regular $(1 - \frac{1}{50})$ -expander with $(2d)^{100}$ nodes and G_2 be a $2d$ -regular $(1 - \frac{1}{50})$ -expander with $(2d)^{200}$ nodes.

3. For $k \geq 3$, let

$$G_k := ((G_{\lfloor \frac{k-1}{2} \rfloor} \otimes G_{\lceil \frac{k-1}{2} \rceil})^{50}) \textcircled{R} H$$

Theorem 10.4. *Every G_k is a $2d$ -regular $(1 - \frac{1}{50})$ -expander with $(2d)^{100k}$ nodes. Furthermore, the mapping*

$$(\text{bin } k, \text{bin } i, \text{bin } j) \mapsto j\text{th neighbour of node } i \text{ in } G_k$$

is computable in time polynomial in k . (Note that k is logarithmic in the size of G_k !)

Proof. The proof of the first part is by induction in k . Let n_k denote the number of nodes of G_k .

Induction base: Clear from construction.

Induction step: The number of nodes of G_k is

$$n_{\lfloor \frac{k-1}{2} \rfloor} \cdot n_{\lceil \frac{k-1}{2} \rceil} \cdot (2d)^{100} = (2d)^{100(k-1)} \cdot (2d)^{100} \cdot (2d)^{100k}.$$

The degree of $G_{\lfloor \frac{k-1}{2} \rfloor}$ and $G_{\lceil \frac{k-1}{2} \rceil}$ is $2d$ by the induction hypothesis. The degree of their tensor product is $(2d)^2$ and of the 50th matrix power is $(2d)^{100}$. Then we take the replacement product with H and get the graph G_k of degree $2d$.

Finally, the second largest eigenvalue of $G_{\lfloor \frac{k-1}{2} \rfloor} \otimes G_{\lceil \frac{k-1}{2} \rceil}$ is $\leq 1 - \frac{1}{50}$. Thus,

$$\tilde{\lambda}((G_{\lfloor \frac{k-1}{2} \rfloor} \otimes G_{\lceil \frac{k-1}{2} \rceil})^{50}) \leq (1 - \frac{1}{50})^{50} \leq \frac{1}{e} \leq \frac{1}{2}$$

Thus $\tilde{\lambda}(G_k) \leq 1 - \frac{1}{2} \cdot 0.99^2/24 \leq 1 - \frac{1}{50}$.

For the second part note that the definition of G_k gives a recursive scheme to compute the neighbourhood of a node. The recursion depth is $\log k$. We have shown how to compute the neighbourhoods of G^{50} , $G \otimes G'$, and $G \textcircled{R} H$ from the neighbourhoods of the given graphs. The total size of the neighbourhood of a node in G_k is $D^{\log k} = \text{poly}(k)$ for some constant D .

■

11 UCONN \in L

We modify the transition relation of k -tape nondeterministic Turing machines as follows: A transition is a tuple (p, p', t_1, \dots, t_k) where p and p' are states and t_κ are triples of the form $(\alpha\beta, d, \alpha'\beta')$. The interpretation is the following: if $d = 1$, the head of M stands on α , and β is the symbol to the right of the head, then M may go to the right and replace the two symbols by α' and β' . If $d = -1$, then the head has to be on β and M goes to the left. In both cases, the machine changes its state from p to p' . An “ordinary” Turing machine can simulate such a Turing machine by always first looking at the symbols to the left and right of the current head position and storing them in its finite control.

By defining a transition like above, every transition T has a reverse transition T^{-1} that undoes what T did. M is now called *symmetric* if for every T in the transition relation Δ , $T^{-1} \in \Delta$.

Definition 11.1.

$$\text{SL} = \{L \mid \text{there is a logarithmic space bounded symmetric Turing machine } M \text{ such that } L = L(M)\}$$

L is a subset of SL. We simply make the transition relation of a deterministic Turing machine M symmetric by adding T^{-1} to it for every T in it. Note that the weakly connected components of the configuration graph of M are directed trees that converge into a unique accepting or rejecting configuration. We cannot reach any other accepting or rejecting configuration by making edges in the configuration graph bidirectional, so the accepted language is the same.

In the same way, we can see that $\text{UCONN} \in \text{SL}$: Just always guess a neighbour of the current node until we reach the target t . The guessing step can be made reversible and the deterministic steps between the guessing steps can be made reversible, too. UCONN is also hard for SL under deterministic logarithmic space reductions. The NL-hardness proof CONN works, we use the fact that the configuration graph of a symmetric Turing machine is undirected. Finally, if $A \in \text{SL}$ and $B \leq_{\log} A$, then $B \in \text{SL}$.

Less obvious are the facts that

- planarity testing is in SL,
- bipartiteness testing is in SL,

- a lot of other interesting problems are contained in SL, see the compendium by [AG00].
- SL is closed under complementation [NTS95].

In this chapter, we will show that $\text{UCONN} \in \text{L}$. This immediately also yields space efficient algorithms for planarity or bipartiteness testing.

11.1 Connectivity in expanders

Lemma 11.2. *Let $c < 1$ and $d \in \mathbb{N}$. The following promise problem can be decided by a logarithmic space bounded deterministic Turing machine:*

Input: *a d -regular graph, such that every connected component is a λ -expander with $\lambda/d \leq c$, nodes s and t .*

Output: *accept if there is a path between s and t , otherwise reject.*

Proof. The Turing machine enumerates all paths of length $O(\log n)$ starting in s . If it sees the node t , it accepts; after enumerating all the paths without seeing t , it rejects.

Since G has constant degree, we can enumerate all paths in space $O(\log n)$. Every path is described by a sequence $\{1, \dots, d\}^{O(\log n)}$. Such a sequence $\delta_0, \delta_1, \dots$ is interpreted as “Take the δ_0 th neighbour of s , then the δ_1 th neighbour of this node, ...”.

If the machine accepts, then there certainly is a path between s and t . For the other direction note that, by Assignment 6, Exercise 6.4 a random walk on G that starts in s converges to the uniform distribution on the connected component containing s . After $O(\log n)$ steps, every node in the same connected component of s has a positive probability of being reached. In particular there is some path of length $O(\log n)$ to it. ■

11.2 Converting graphs into expanders

Lemma 11.3. *There is a logarithmic space computable transformation that transforms any graph $G = (V, E)$ into a cubic regular graph $G' = (V', E')$ such that $V \subseteq V'$ and for any pair of nodes $s, t \in V$, there is a path between s and t in G iff there is one in G' .*

Proof. If a node v in G

1. has degree $d > 3$, then we replace v by a cycle of length d and connect every node of the cycle to one of the neighbours of v .
2. has degree $d \leq 3$, then we add $3 - d$ self loops.

For every node v with degree > 3 , we identify one of the new nodes of the cycle with v . Let the resulting graph be G' . By construction, G' is cubic and if there is a path between s and t in G then there is one between in G' and vice versa.

With a little care, the construction can be done in logarithmic space. (Recall that the Turing machine has a separate output tape that is write-only and oneway, so once it decided to output an edge this decision is not reversible.) We process each node in the order given by the representation of G . For each node v , we count the number m of its neighbours. If $m \leq 3$, then we just copy the edges containing v to the output tape and output the additional self loops. If $m > 3$, then we output the edges $\{(v, i), (v, i + 1)\}$, $1 \leq i < m$ and $\{(v, m), (v, 1)\}$. Then we go through all neighbours of v . If u is the i th neighbour of v , then we determine which neighbour v of u is, say the j th, and output the edge $\{(v, i), (u, j)\}$. (We only need to do this if v is processed before u because otherwise, the edge is already output.) ■

Let d be large enough such that there is a $d/2$ -regular 0.01-expander H with d^{50} nodes. (Again, the constants are chosen in such a way that the proof works; they are fairly arbitrary and we have taken them from the book by Arora and Barak.) We can make our cubic graph G d^{50} -regular by adding $d^{50} - 3$ self loops per node. Recursively define

$$\begin{aligned} G_0 &:= G \\ G_k &:= (G_{k-1} \circledast H)^{50}. \end{aligned}$$

Lemma 11.4. *For all $k \geq 1$,*

1. G_k has $d^{50k} \cdot n$ nodes,
2. G_k is d^{50} -regular,
3. $\tilde{\lambda}(G_k) \leq 1 - \epsilon_k$, where $\epsilon_k = \min\{\frac{1}{20}, \frac{1.5^k}{8d^{50}n^3}\}$.

Proof. The proof is by induction in k . Let n_k be the number of nodes of G_k .

Induction base: G_0 has n nodes and degree d^{50} . By by Assignment 6, Exercise 6.4, $\tilde{\lambda}(G_0) \leq 1 - \frac{1}{8d^{50}n^3} \leq 1 - \epsilon_0$.

Induction step: The replacement product $G_k \circledast H$ has $n_k \cdot d^{50} = n_{k+1}$ nodes. Its degree is d . G_{k+1} has the same number of nodes and the degree becomes d^{50} . We have

$$\tilde{\lambda}(G_k \circledast H) \leq 1 - \frac{\epsilon_k}{24} \cdot 0.99^2 \leq 1 - \frac{\epsilon_k}{25}$$

and

$$\tilde{\lambda}(G_k) \leq \left(1 - \frac{\epsilon_k}{25}\right)^{50} \leq e^{-2\epsilon_k} \leq 1 - 2\epsilon_k + 2\epsilon_k^2 = 1 - 2\epsilon_k(1 - \epsilon_k).$$

If $\epsilon_k = \frac{1}{20}$, then $\tilde{\lambda}(G_k) \leq 1 - \frac{1}{20}$. If $\epsilon_k = \frac{1.5^k}{8d^{50}n^3} < \frac{1}{20}$, then

$$\tilde{\lambda}(G_k) \leq 1 - 1.5\epsilon_k = 1 - \epsilon_{k+1}. \quad \blacksquare$$

If we set $k = O(\log n)$, then G_k is a constant degree expander with $\tilde{\lambda}(G_k) \leq \frac{19}{20}$. For such graphs, connectivity can be decided in deterministic logarithmic space by Lemma 11.2. So we could first make our input graph cubic, then compute G_k for $k = O(\log n)$ and finally use the connectivity algorithm for expander graphs. Since L is closed under logarithmic space computable reductions, this would show $\text{UCONN} \in \text{SL}$.

But there one problem: To compute G_k , we cannot compute G_0 , then G_1 , then G_2 , and so on, since L is only closed under application of a *constant* number of many-one-reductions. Thus we have to compute G_k from G_0 in one step.

Lemma 11.5. *The mapping $G_0 \rightarrow G_k$ with $k = O(\log n)$ is deterministic logarithmic space computable.*

Proof. Assume that G_0 has nodes $\{1, \dots, n\}$. Then the nodes of G_k are from $\{1, \dots, n\} \times \{1, \dots, d^{50}\}^k$. The description length of a node of G_k is $\log n + 50 \log d \cdot k = O(\log n)$. We will identify $\{1, \dots, d^{50}\}$ with $\{1, \dots, d\}^{50}$, since an edge in G_k corresponds to a path of length 50 in $G_{k-1} \otimes H$.

Now given a node $v = (i, \delta_1, \dots, \delta_k)$ of G_k and $j \in \{1, \dots, d^{50}\}$, we want to compute the j th neighbour of v in G_k . We interpret j as a sequence $(j_1, \dots, j_{50}) \in \{1, \dots, d\}^{50}$.

Input: node $v = (i, \delta_1, \dots, \delta_k)$ of G_k , index $j = (j_1, \dots, j_{50})$

Output: the j th neighbour of v in G_k

1. For $h = 1, \dots, 50$ compute the j_h neighbour of the current node in $G_{k-1} \otimes H$.

So it remains to compute the neighbours in $G_{k-1} \otimes H$.

Input: node $v = (i, \delta_1, \dots, \delta_k)$ of G_k , index j

Output: the j th neighbour of v in $G_{k-1} \otimes H$

1. If $j \leq d/2$, then return $(i, \delta_1, \dots, \delta_{k-1}, \delta')$ where δ' is the j neighbour of δ_k in H . Since H is constant, this can be hard-wired.
(We perform an internal step inside a copy of H .)
2. Otherwise, recursively compute the δ_k th neighbour of $(i, \delta_1, \dots, \delta_{k-1})$ in G_{k-1} .
(We perform an external step between two copies of H .)

Note that we can view $(v, \delta_1, \dots, \delta_k)$ as a stack and all the recursive calls operate on the same step. Thus we only have to store one node at a time. \blacksquare

Theorem 11.6 (Reingold [Rei08]). $\text{UCONN} \in \text{SL}$.

Corollary 11.7. $\text{L} = \text{SL}$.

Implementation details

In the proof, we always assume that the graph G is connected. If it is not connected, then we apply the construction to each connected component. Convince yourself that all the constructions work, in particular, if the graph is not connected, then all the logspace algorithms apply the transformation to each connected component separately.

12 Average-case complexity

Being intractable, e.g., NP-complete, does not completely reflect the difficulty of a problem. Approximability is one way of refining the notion of intractability: We have seen some NP-hard optimization problems, for which finding a close-to-optimal solution is easy, and others, for which finding even a very weak approximation is as hard as solving an NP-complete problem.

Average-case complexity is another way of refining the intractability of problems: Unless $P = NP$, no efficient algorithm exists to solve an NP-complete problem efficiently on all instances. However, we may still hope that we can solve the problem efficiently on most instances or on typical instances, where “typical” here means something like “sampled from a distribution that reflects practical instances.”

Another motivation for studying average-case complexity is cryptography: A cryptographic system is secure only if any efficient attempt to break it succeeds only with a very small probability. Thus, it does not help if a cryptographic system is hard to break only in the worst case. In fact, most of cryptography assumes that NP problems exist that are intractable not only in the worst but even in the average case, i.e., on random inputs.

Bogdanov and Trevisan give a well-written survey about average-case complexity [BT06]. They also cover connections of average-case complexity to areas like cryptography.

12.1 Probability distributions and distributional problems

What probability distribution should we take? What probability distributions should we allow? How should we model that we have inputs of various sizes?

There are essentially two ways to deal with inputs of various sizes: First, for each $n \in \mathbb{N}$, we can have a distribution D_n , from which we draw the instances of “size” n (e.g., n can be the length of the strings). Combining D_1, D_2, D_3, \dots , we get an *ensemble* $\mathcal{D} = (D_n)_{n \geq 1}$ of probability distributions.

The second possibility is to have a single probability distribution \mathcal{D} for strings of all lengths. This is convenient in some applications, and it leads to a simple notion of reducibility that preserves average-case tractability. But it is difficult, e.g., to define circuit complexity for this case, and it is also sometimes counterintuitive. For instance, since $\{0, 1\}^*$ is a countable infi-

nite set, there is no “uniform” distribution on $\{0, 1\}^*$. Instead, the distribution that is commonly called the “uniform distribution” assigns probability $\frac{6}{\pi^2}n^{-2}2^{-n}$ to every string of length $n \geq 1$ (or, to get rid of the factor $6/\pi^2$, we assign probability $\frac{1}{n(n+1)}2^{-n}$ to strings of length n). We will use ensembles of distributions as it will be more convenient. However, most results hold independent of which possibility we choose. The uniform distribution $\mathcal{U} = (U_n)_{n \in \mathbb{N}}$ is then given by $U_n(x) = 2^{-n}$ for $x \in \{0, 1\}^n$.

In order to allow for different probability distributions, we do not fix one, but we will consider *distributional problems* in the following.

Definition 12.1. A distributional decision problem is a pair $\Pi = (L, \mathcal{D})$, where L is a language and $\mathcal{D} = (D_n)_{n \geq 1}$ is an ensemble of probability distributions, where each D_n has finite support.

By $\text{supp}(D) = \{x \mid D(x) > 0\}$, we denote the support of a probability distribution. You can think of $\text{supp}(D_n) \subseteq \{0, 1\}^n$, but this will not always be the case. However, we will have $\text{supp}(D_n) \subseteq \{0, 1\}^{\leq p(n)}$ for some polynomial p for the distributions that we consider.

What we would now like to have is something like an “average-case NP-hard” distributional problem $\Pi = (L, \mathcal{D})$: If Π is average-case tractable, then (L', \mathcal{D}') is average-case tractable for each $L' \in \text{NP}$ and each ensemble \mathcal{D}' .

However, as we will show, a statement like “every (L', \mathcal{D}') is average-case tractable” is the same as “ $\text{P} = \text{NP}$ ”. (The previous sentence is non-trivial: For the same language L' , we can use different algorithms for different distributions \mathcal{D}' .) Thus, the average-case analog of NP-completeness cannot refer to arbitrary probability distributions. We will restrict ourselves to two possible sets of distributions, namely polynomial time samplable and polynomial time computable ensembles.

Definition 12.2. An ensemble $\mathcal{D} = (D_n)$ is polynomial-time samplable if there exists a randomized algorithm A that takes an input $n \in \mathbb{N}$ and produces a string in $A(n) \in \{0, 1\}^*$ with the following properties:

- There exists a polynomial p such that A on input n is $p(n)$ time bounded, regardless of the random bits A reads.
- For every $n \in \mathbb{N}$ and every $x \in \{0, 1\}^*$, we have $\Pr(A(n) = x) = D_n(x)$.

PSamp denotes the set of all polynomial-time samplable ensembles.

Several variants of definitions of polynomial-time samplable distributions exist. For instance, one can relax the strict bound on the running-time and require only that A runs in expected polynomial time. Such finer distinctions are important, e.g., in the study of zero-knowledge proofs, but we will not elaborate on this.

To define the second set of distributions, let \leq denote the lexicographic ordering between bit strings. (We have $x \leq y$ if $|x| < |y|$ or $|x| = |y|$ and x appears before y in lexicographic order.) Then the *cumulative probability* of x with respect to a probability distribution D is defined by

$$f_D(x) = \sum_{y \leq x} D(y).$$

Definition 12.3. An ensemble $\mathcal{D} = (D_n)$ is polynomial-time computable if there exists a deterministic algorithm A that, on input $n \in \mathbb{N}$ and $x \in \{0, 1\}^*$, runs in time $\text{poly}(n)$ and computes $f_{D_n}(x)$.

PComp denotes the set of all polynomial-time computable ensembles.

If a distribution $f_{D_n}(x)$ is computable in time $\text{poly}(n)$, then also the density function $D_n(x)$ is computable in time $\text{poly}(n)$. The converse does not hold unless $\text{P} = \text{NP}$.

Exercise 12.1. Show that there exists an ensemble $\mathcal{D} = (D_n)_{n \in \mathbb{N}}$ such that the density functions D_n are polynomial-time computable but $\mathcal{D} \notin \text{PComp}$ unless $\text{P} = \text{NP}$. The latter means that if the functions f_{D_n} are computable in polynomial time, then $\text{P} = \text{NP}$.

In the following, we will focus on ensembles from **PComp**. Many results will nevertheless carry over to the wider class **PSamp**.

Exercise 12.2. Show that $\text{PComp} \subseteq \text{PSamp}$.

The converse is unlikely to be true.

Exercise 12.3. Show that $\text{PComp} = \text{PSamp}$ if and only if $\text{P} = \text{P}^{\#\text{P}}$.

Hint: For \Rightarrow , define an appropriate distribution on $\langle \varphi, a \rangle$ (choose pairing function and encoding carefully), where φ is a Boolean formula and a is an assignment for φ . Given φ , the probability of $\langle \varphi, a \rangle$ should essentially depend on whether a satisfies φ . Show that the cumulative distribution function can be used to compute the number of satisfying assignments.

One can argue that **PSamp** is the class of natural distributions: A distribution is natural if we can efficiently sample from it. This, however, does not mean that their density or distribution function is efficiently computable.

12.2 Average polynomial time and heuristics

When can we say that a problem can be solved efficiently on average? A first attempt might be to consider the expected running-time with respect to a given probability distribution. However, this definition, although very natural, does not yield a robust class of average tractable problems.

Exercise 12.4. *Show that the class*

$$\{(L, \mathcal{D}) \mid L \text{ can be solved in expected polynomial time with respect to } \mathcal{D}\}$$

is not invariant under changes of the machine model: Let A be an algorithm with running-time t , and let B be a simulation of A that is, say, quadratically slower. Give a function t such that A has expected polynomial running-time but B has not.

Another way would be to consider the median of the running-time. At least, this would be a definition that is robust against changes of the machine model. However, you probably would not call an algorithm efficient if it requires linear time on 70% of the instances and exponential time on the rest. So what about requiring that the algorithm runs in polynomial time on 99% of the instances? The threshold 99% would be arbitrary, there is no reason why 99% is preferable to 98% or 99.9%. But with any such threshold, there would still be a constant fraction of the inputs, for which the algorithm performs poorly.

But what would be a natural way of defining “average tractability”? Intuitively, one would probably say that an algorithm is efficient on average if instances that require longer and longer running time show up with smaller and smaller probability: An algorithm A has polynomial average running-time if there exists a constant $c > 0$ such that the probability that A requires more than time T is at most $\text{poly}(n)/T^c$. In this way, we have a polynomial trade-off between running-time and fraction of inputs.

Definition 12.4. *An algorithm A with running-time t_A has average polynomial running-time with respect to the ensemble \mathcal{D} if there exists an $\varepsilon > 0$ and a polynomial p such that, for all n and t ,*

$$\Pr_{x \sim D_n} (t_A(x, n) \geq t) \leq \frac{p(n)}{t^\varepsilon}.$$

If an algorithm A has average polynomial running-time, then its median running-time is polynomial, it runs in polynomial time on all but a $1/\text{poly}(n)$ fraction of inputs, and in time $n^{\text{polylog}(n)}$ on all but an $n^{-\text{polylog}(n)}$ fraction of the inputs, and so on.

The whole field of average-case complexity theory was essentially founded by Leonid Levin [Lev86]. (Leonid Levin is the guy who did not get a Turing award for inventing the Russian analog of NP completeness. His paper [Lev86] is a good candidate for the shortest paper in theoretical computer science that founded an area: The conference version of the paper is one page long, the final journal paper has two pages.) Levin’s original definition is different, but turns out to be equivalent: An algorithm has average polynomial running-time with respect to \mathcal{D} if there exists an $\varepsilon > 0$ such that

$$\mathbb{E}_{x \sim D_n} (t_A(x, n)^\varepsilon) = O(n).$$

Exercise 12.5. Prove that Definition 12.4 and Levin’s definition are indeed equivalent.

Exercise 12.6. Actually, Levin’s original definition uses a single distribution $D : \{0, 1\}^+ \rightarrow [0, 1]$. Given an ensemble $\mathcal{D} = (D_n)_{n \geq 1}$ with $\text{supp}(D_n) \subseteq \{0, 1\}^n$, we obtain a single distribution $D : \{0, 1\}^+ \rightarrow [0, 1]$ by setting $D(x) = \frac{6}{\pi^2 |x|^2} D_{|x|}(x)$.

A function $t : \{0, 1\}^+ \rightarrow \mathbb{N}$ is called polynomial on D -average if there exists constants k and c such that

$$\sum_{x \in \{0, 1\}^+} \frac{t(x)^{1/k}}{|x|} D(x) \leq c.$$

Prove that t is polynomial on D -average if and only if $\Pr_{x \sim D_n}(t(x, n) \geq t) \leq \frac{p(n)}{t^\varepsilon}$ for all n , some polynomial p , and some $\varepsilon > 0$.

In addition to how to measure time, we have the choice to use deterministic or randomized algorithms or even non-uniform families of circuits.

“In practice”, we would probably not run an algorithm forever, but only for a polynomial number of steps. We can model this by an algorithm with worst-case polynomial running-time that “fails” on some inputs. This leads to the following definition, where “failure” means that the algorithm says “I don’t know.”

Definition 12.5. Let $\Pi = (L, \mathcal{D})$ be a distributional problem with $\mathcal{D} = (D_n)$. An algorithm A is a (fully polynomial-time) errorless heuristic scheme for Π if there is a polynomial p such that the following holds:

- For every n , every $\delta > 0$, and every $x \in \text{supp}(D_n)$, $A(x, n, \delta)$ outputs either $L(x)$ or the failure symbol \perp .
- For every n , every $\delta > 0$, and every $x \in \text{supp}(D_n)$, $A(x, n, \delta)$ runs in time $p(n, 1/\delta)$.
- For every n and every $\delta > 0$, we have $\Pr_{x \sim D_n}(A(x, n, \delta) = \perp) \leq \delta$.

This, however, is yet another way of defining average-polynomial time.

Exercise 12.7. Show that a distributional problem Π admits an errorless heuristic scheme if and only if it admits an algorithm whose running-time is average-polynomial according to Definition 12.4.

Now that we have three equivalent definitions of “tractable on average”, we will finally define a complexity class of distributional problems that are tractable on average.

Definition 12.6. The class AvgP is the set of all distributional problems that admit an errorless heuristic scheme.

Exercise 12.8. *Let*

$$3\text{COL} = \{G \mid G \text{ is 3-colorable}\},$$

let $G_{n,1/2}$ be the uniform distribution on graphs with n vertices, and let $\mathcal{G}_{1/2} = (G_{n,1/2})_{n \in \mathbb{N}}$. Show that $(3\text{COL}, \mathcal{G}_{1/2}) \in \text{AvgP}$.

Note: If we include every edge of $\binom{[n]}{2}$ with a probability of p , i.e., the probability of getting $G = (V, E)$ is $p^{|E|}(1-p)^{\binom{n}{2}-|E|}$, then this probability distribution is called $G_{n,p}$.

Instead of having δ as part of the input, we can also have a function $\delta : \mathbb{N} \rightarrow (0, 1]$ of failure probabilities. This leads to the following definition.

Definition 12.7. *Let $\Pi = (L, \mathcal{D})$ be a distributional problem, and let $\delta : \mathbb{N} \rightarrow (0, 1]$. An algorithm A is an errorless heuristic algorithm for Π with failure probability at most δ if the following properties are fulfilled:*

- For every $n \in \mathbb{N}$ and every $x \in \text{supp}(D_n)$, $A(x, n)$ outputs either $L(x)$ or \perp .
- For every $n \in \mathbb{N}$, we have $\Pr_{x \sim D_n}(A(x, n) = \perp) \leq \delta(n)$.

For a time bound t , we say that $\Pi \in \text{Avg}_\delta \text{DTime}(t)$ if there exists an errorless heuristic deterministic algorithm A that, for every n , runs on time at most $t(n)$ for all $x \in \text{supp}(D_n)$ and has failure probability at most δ .

$$\text{Let } \text{Avg}_\delta \text{P} = \bigcup_{p: \text{polynomial}} \text{Avg}_\delta \text{DTime}(p).$$

So far, all algorithms that we considered never produced wrong answers. They might return “don’t know”, but if they give an answer, it is the correct answer. Weakening the requirement yields the following definition.

Definition 12.8. *An algorithm A is called a (fully polynomial-time) heuristic scheme for $\Pi = (L, \mathcal{D})$ if there exists a polynomial p such that the following holds:*

1. For every n , every $x \in \text{supp}(D_n)$, and every $\delta > 0$, $A(x, n, \delta)$ runs in time $p(n, 1/\delta)$.
2. For every $\delta > 0$, $A(\cdot, \cdot, \delta)$ is a heuristic algorithm for Π with error probability at most δ .

Let HeurP be the set of all distributional problems that admit a heuristic scheme.

Definition 12.9. *Let $\Pi = (L, \mathcal{D})$ be a distributional problem, and let $\delta : \mathbb{N} \rightarrow (0, 1]$. An algorithm A is a heuristic algorithm with error probability at most δ for Π if, for all n ,*

$$\Pr_{x \sim D_n} (A(x, n) \neq L(x)) \leq \delta(n).$$

For a time bound t and $\delta : \mathbb{N} \rightarrow (0, 1]$, we have $\Pi \in \text{Heur}_\delta \text{DTime}(t)$ if there exists a heuristic deterministic algorithm A such that, for every n and every $x \in \text{supp}(D_n)$, $A(x, n)$ runs in time $t(n)$ with failure probability at most $\delta(n)$.

Let $\text{Heur}_\delta \text{P} = \bigcup_{p: \text{polynomial}} \text{Heur}_\delta \text{DTime}(p)$.

Exercise 12.9. Prove the following: For every constant c , $\text{AvgP} \subseteq \text{Avg}_{n-c} \text{P}$ and $\text{HeurP} \subseteq \text{Heur}_{n-c} \text{P}$.

So far, we have defined classes of problems that are tractable (to various degrees) on average. Now we define the average-case analog of NP, which is called DistNP.

Definition 12.10. DistNP is the class of all distributional problems $\Pi = (L, \mathcal{D})$ with $L \in \text{NP}$ and $\mathcal{D} \in \text{PComp}$.

Similar to P versus NP, the central question in average-case complexity is whether $\text{DistNP} \subseteq \text{AvgP}$. Note that $\text{AvgP} = \text{DistNP}$ does not hold: First, there is no restriction on the probability distributions for the problems in AvgP. Second, AvgP might contain problems that are not even in NP.

12.3 A distribution for which worst case equals average case

There exists an ensemble for which worst case and average case are equivalent. Thus, the study of average-case complexity with respect to all (instead of only samplable or computable) distributions reduces to worst-case complexity. To get meaningful results in average-case complexity, we thus have to restrict ourselves to sufficiently simple sets of ensembles like PComp or PSamp.

Theorem 12.11. There exists an ensemble \mathcal{K} such that if L is a decidable language and the distributional problem (L, \mathcal{K}) is in $\text{Heur}_{o(1)} \text{P}$, then $L \in \text{P}$.

For the proof of this theorem, we need *Kolmogorov complexity* (see also the script of the lecture “Theoretical Computer Science”). To define Kolmogorov complexity, let us fix a universal Turing machine U . The Kolmogorov complexity of a string $x \in \{0, 1\}^*$ is the length of the shortest string c such that U on input $c = \langle g, y \rangle$ outputs x . Here, g is a Gödel number of a Turing machine and y is an input for the Turing machine M_g . We denote the length $|c|$ by $K(x)$.

The *conditional Kolmogorov complexity* $K(x|z)$ is the length of the shortest string $c = \langle g, y \rangle$ such that M_g on input y and z outputs x .

Example 12.12. We have $K(0^n) = \log n + O(1)$ and $K(0^n | \text{bin}(n)) = O(1)$ and $K(x) \leq |x| + O(1)$.

By abusing notation, we write $K(x|n)$ instead of $K(x|\text{bin}(n))$ for $|x| = n$. This is also called the *length-conditioned Kolmogorov complexity*.

Proof of Theorem 12.11. The universal probability distribution $\mathcal{K} = (K_n)_{n \in \mathbb{N}}$ assigns every string x of length n a probability proportional to $2^{-K(x|n)}$: We have $\sum_{x \in \{0,1\}^n} 2^{-K(x|n)} = \sum_x K_n(x) \leq 1$ since $\langle \cdot, \cdot \rangle$ is a prefix-free code (this follows from Kraft's inequality). By scaling with an appropriate constant ≥ 1 , we make sure that the sum equals 1.

Since $L \in \text{Heur}_{o(1)}\mathbb{P}$, there exists a heuristic algorithm A for L that fails with probability at most $o(1)$. Consider a string x of length n such that $A(x) \neq L(x)$. Since the overall probability of such strings is at most $o(1)$, we have $K_n(x) = o(1)$. Thus,

$$K(x|n) = -\log K_n(x) + O(1) = \omega(1).$$

This lower bound for the Kolmogorov complexity holds for all strings on which A fails.

Now let x_0 be the lexicographically first string of length n on which A fails. Since L is decidable, x_0 can be computed given n . This implies $K(x_0|n) = O(1)$. To conclude, we observe that for sufficiently large n , no string exists on which A fails. Thus, A fails only on finitely many strings, which proves $L \in \mathbb{P}$. ■

Exercise 12.9 immediately yields the following result.

Corollary 12.13. *If $(L, \mathcal{K}) \in \text{Heur}\mathbb{P}$, then $L \in \mathbb{P}$.*

Exercise 12.10. *An interesting feature of the universal distribution is the following: Let A be any algorithm with running-time $t : \{0,1\}^* \rightarrow \mathbb{N}$. Then the expected running-time with respect to \mathcal{K} is asymptotically equal to the worst-case running-time:*

$$\mathbb{E}_{x \sim K_n} t(x) = \Theta \left(\max_{x \in \{0,1\}^n} t(x) \right),$$

where the constant hidden by Θ depends only on the algorithm A .

Prove this!

Exercise 12.11. *Let $M = \{1, \dots, m\}$. An encoding of M is an injective mapping $c : M \rightarrow \{0, 1, \dots, \gamma - 1\}^*$. The code c is called prefix-free if there are no $i \neq j$ in M such that $c(i)$ is a prefix of $c(j)$.*

Prove the following: Assume that we are given lengths $\ell_1, \dots, \ell_m \in \mathbb{N}$. Then there exists a prefix-free code c for M with $|c(i)| = \ell_i$ if and only if $\sum_{i=1}^m \gamma^{-\ell_i} \leq 1$. (This is called Kraft's inequality.)

13 Average-case completeness

The goal of this section is to prove that $\text{DistNP} = (\text{NP}, \text{PComp})$ contains a complete problem. There are three issues that we have to deal with: First, we need an appropriate notion of reduction. Second, we have to take care of the different probability distributions, which can differ vastly. Third, and this is the easiest task, we need an appropriate problem with an appropriate probability distribution.

13.1 Reductions

For reductions between distributional problems, the usual many-one reductions do not suffice: A feature that a suitable reduction should have is that if (A, \mathcal{D}) reduces to (A', \mathcal{D}') and $(A', \mathcal{D}') \in \text{AvgP}$, then $(A, \mathcal{D}) \in \text{AvgP}$. Let us consider the following example: We use the identity mapping as reduction between (A, \mathcal{D}) and (A, \mathcal{D}') . But \mathcal{D} assigns high probability to the hard instances, whereas \mathcal{D}' assigns small probability to hard instances. Then (A, \mathcal{D}') is tractable on average, but (A, \mathcal{D}) is not.

What we learn is that a meaningful notion of reduction must take into account the probability distributions.

Definition 13.1. *Let $\Pi = (L, \mathcal{D})$ and $\Pi' = (L', \mathcal{D}')$ be distributional problems. Then Π reduces to Π' , denoted by $\Pi \leq_{\text{AvgP}} \Pi'$, if there is a function f that, for every n and every x in the support of D_n , can be computed in time polynomial in n such that*

1. (correctness) $x \in L$ if and only if $f(x, n) \in L'$ and
2. (domination) there exist polynomials p and m such that, for every n and every y in the support of $D'_{m(n)}$,

$$\sum_{x:f(x,n)=y} D_n(x) \leq p(n) \cdot D'_{m(n)}(y).$$

The first condition is usual for many-one reductions. The second condition forbids the scenario sketched above: Drawing strings according to D_n and then using $f(\cdot, n)$ yields a probability distribution of instances for L' . (Of course, we just get binary strings again. But we view them as instances for L' .) Then the second property makes sure that no string y is generated with much larger a probability than if y had been drawn according to $D'_{m(n)}$.

Lemma 13.2. *Let $\mathsf{C} \in \{\text{AvgP}, \text{HeurP}\}$. If $\Pi \leq_{\text{AvgP}} \Pi'$ and $\Pi' \in \mathsf{C}$, then $\Pi \in \mathsf{C}$.*

Proof. We only consider the case $\mathsf{C} = \text{AvgP}$. The other case is similar. Suppose $\Pi' \in \text{AvgP}$, and let A' be a fully polynomial-time errorless heuristic scheme for Π' . Let f be a reduction from Π to Π' , and let p and m be the polynomials of Definition 13.1.

We claim that $A(x, n, \delta) = A'(f(x, n), m(n), \delta/p(n))$ is a fully polynomial-time errorless heuristic scheme for Π . To prove this, let $B_n = \{y \in \text{supp}(D'_{m(n)}) \mid A'(y, m(n), \delta/p(n)) = \perp\}$ be the set of strings on which A' fails.

Since A' is a fully polynomial-time errorless heuristic scheme, we have $D'_{m(n)}(B_n) \leq \delta/p(n)$. With this, we get

$$\begin{aligned} \Pr_{x \sim D_n} (A(x, n, \delta) = \perp) &= \Pr_{x \sim D_n} (A'(f(x, n), m(n), \delta/p(n)) = \perp) \\ &= \sum_{x: f(x, n) \in B_n} D_n(x) \\ &\leq \sum_{y \in B_n} p(n) D'_{m(n)}(y) \\ &= p(n) \cdot D'_{m(n)}(B_n) \leq \delta. \end{aligned}$$

The inequality holds because the reduction must fulfill the domination condition. Thus, $\Pi \in \text{AvgP}$. ■

It is also not hard to show that \leq_{AvgP} is transitive.

Lemma 13.3. *Let $\Pi = (L, \mathcal{D})$, $\Pi' = (L', \mathcal{D}')$, and $\Pi'' = (L'', \mathcal{D}'')$ be distributional problems with $\Pi \leq_{\text{AvgP}} \Pi'$ and $\Pi' \leq \Pi''$. Then $\Pi \leq_{\text{AvgP}} \Pi''$.*

Proof. Let f be a reduction from Π to Π' , and let g be a reduction from Π' to Π'' . Let p and m the polynomials of Definition 13.1 for f , and let p' and m' be the polynomials for g . Obviously, h given by $h(x, n) = g(f(x, n), m(n))$ is polynomial-time computable and a many-one reduction from L to L'' . To prove that it fulfills domination remains to be done. Therefore, Let $q(n) = p(n) \cdot p'(m(n))$, and let $\ell(n) = m'(m(n))$. The functions q and ℓ are obviously polynomials. Now let n be arbitrary, and let $z \in \text{supp}(D''_{\ell(n)})$. Then

$$\begin{aligned} \sum_{x: h(x, n) = z} D_n(x) &= \sum_{y: g(y, m(n)) = z} \sum_{x: f(x, n) = y} D_n(x) \\ &\leq \sum_{y: g(y, m(n)) = z} p(n) D'_{m(n)}(y) \\ &\leq p(n) p'(m(n)) D''_{m'(m(n))}(z) = q(n) D''_{\ell(n)}(z). \end{aligned}$$

■

13.2 Bounded halting

In this section, we present a problem that is complete for DistNP. It is the bounded halting problem, which is *the* generic NP-complete problem:

$$\text{BH} = \{ \langle g, x, 1^t \rangle \mid M_g \text{ is a non-deterministic Turing machine} \\ \text{that accepts } x \text{ in at most } t \text{ steps} \}.$$

We will show that $(\text{BH}, \mathcal{U}^{\text{BH}})$ is DistNP-complete, where $\mathcal{U}^{\text{BH}} = (U_n^{\text{BH}})_{n \in \mathbb{N}}$ is some kind of uniform distribution on the inputs for BH.

The main challenge in the proof that $(\text{BH}, \mathcal{U}^{\text{BH}})$ is DistNP-complete is that problems $\Pi = (L, \mathcal{D})$ from DistNP have very different distributions. Thus, just using the many-one reduction from L to BH is unlikely to work.

The key idea is to find an injective mapping C with the property that $C(x)$ is almost uniformly distributed if x is distributed according to \mathcal{D} . The following lemma makes this more precise and proves that such a function exists.

Lemma 13.4. *Let $\mathcal{D} = (D_n)_{n \in \mathbb{N}} \in \text{PComp}$ be an ensemble. Then there exists an algorithm C with the following properties:*

1. $C(x, n)$ runs in time polynomial in n for all $x \in \text{supp}(D_n)$,
2. for every n and $x, x' \in \text{supp}(D_n)$, $C(x, n) = C(x', n)$ implies $x = x'$ (so C is somewhat “injective”), and
3. $|C(x, n)| \leq 1 + \min\{|x|, \log \frac{1}{|D_n(x)|}\}$.

Proof. Consider any $x \in \text{supp}(D_n)$. If $D_n(x) \leq 2^{-|x|}$, then let $C(x, n) = 0x$. If $D_n(x) > 2^{-|x|}$, then let y be the string that precedes x in lexicographic order, and let $p = f_{D_n}(y)$. Then we set $C(x, n) = 1z$, where z is the longest common prefix of the binary representation of p and $f_{D_n}(x) = p + D_n(x)$. Since $\mathcal{D} \in \text{PComp}$, the string z can be computed in polynomial time. Thus, C can be computed in polynomial time. (This also shows that $|C(x, n)|$ is bounded by a polynomial in $|x|$.)

It remains to prove that C is injective and fulfills the length condition. C is injective because no three strings can have the same longest common prefix: If z is the longest common prefix of x_1 and x_2 and z is also a prefix of x_3 , then also either $z0$ or $z1$ is a prefix of x_3 and it is also a prefix of either x_1 and x_2 .

Finally, we observe that either $C(x, n) = 0x$ or $C(x, n) = 1z$ for the z described above. In the former case, we have $|C(x, n)| \leq 1 + |x|$ and $D_n(x) \leq 2^{-|x|}$. Thus, also $1 + \log(1/D_n(x)) \geq |x| + 1 = |C(x, n)|$. In the latter case, we have $D_n(x) \leq 2^{-|z|}$. Thus, $|C(x, n)| = 1 + |z| \leq \log(1/D_n(x))$ and, since $D_n(x) \geq 2^{-|x|}$, $|C(x, n)| \leq 1 + |x|$. ■

Now let us focus on \mathcal{U}^{BH} . The instances of BH are triples $\langle g, x, 1^t \rangle$ of length $2 \log |g| + 2 \log |x| + 2 \log t + |x| + |g| + t + \Theta(1)$. Note that this representation is prefix-free. We draw such instances of length at most N as follows: We flip random bits b_1, b_2, \dots until either $i = N$ or $b_1 \dots b_i$ has the form $\langle g, x \rangle$. In the former case, we output $b_1 \dots b_N$, in the latter we output $\langle g, x, 1^{N-i} \rangle$. We denote this distribution by U_N^{BH} . The probability of an instance $\langle g, x, 1^t \rangle$ under this distribution is

$$U_N^{\text{BH}}(\langle g, x, 1^t \rangle) = 2^{-(2 \log |g| + 2 \log |x| + |g| + |x| + \Theta(1))},$$

where $t = N - |\langle g, x \rangle|$.

With this preparation, we can prove the main theorem of this section. The key idea is to use C to “compress” the inputs: While D_n can be any distribution, the images $C(x, n)$ with x drawn according to D_n are, more or less, uniformly distributed in the sense of \mathcal{U}^{BH} : If x is likely, then $C(x, n)$ is short. If x is unlikely, then $C(x, n)$ is long. Thus, if we draw random bits until we have seen an image $y = C(x, n)$, then the probability of seeing y is roughly $D_n(x)$.

Theorem 13.5. $(\text{BH}, \mathcal{U}^{\text{BH}})$ is DistNP with respect to \leq_{AvgP} .

Proof. Let $\Pi = (L, \mathcal{D}) \in \text{DistNP}$ be arbitrary, i.e., $L \in \text{NP}$ and $\mathcal{D} \in \text{PComp}$. Let M be a nondeterministic Turing machine that accepts an input string y if and only if there exists a string $x \in L$ with $C(x, n) = y$. Since C is polynomial-time computable and $L \in \text{NP}$, we can assume that M obeys a polynomial time bound q . Let g be the Gödel number of M .

Let us describe the reduction from (L, \mathcal{D}) to $(\text{BH}, \mathcal{U}^{\text{BH}})$: On input x and parameter n , the reduction outputs an instance $\langle g, C(x, n), 1^{t(x)} \rangle$ of length $N = N(n)$. We choose N to be a sufficiently large polynomial to make sure that $t(x) \geq q(n)$.

Obviously, we have $x \in L$ if and only if $\langle g, C(x, n), 1^{t(x)} \rangle \in \text{BH}$. To verify the domination condition, we exploit that C is injective. Thus, it suffices to check that, for every n and every $x \in \text{supp}(D_n)$, we have $D_n(x) \leq \text{poly}(n) \cdot U_N^{\text{BH}}(\langle g, x, 1^{t(x)} \rangle)$.

Let $\ell = |g|$ be the length of the encoding of M , which is fixed. Then

$$U_N^{\text{BH}}(\langle g, C(x, n), 1^{q(n)} \rangle) = 2^{-(2 \log \ell + 2 \log |C(x, n)| + \ell + |C(x, n)| + \Theta(1))}.$$

Now $\log |C(x, n)| \leq \log(m(n)) + 1$ and $|C(x, n)| \leq \log\left(\frac{1}{D_n(x)}\right) + 1$ yields

$$U_N^{\text{BH}}(\langle g, C(x, n), 1^{q(n)} \rangle) \geq \underbrace{2^{-(2 \log \ell + \ell)}}_{=\Theta(1)} \cdot \frac{1}{(m(n) + 1)^2} \cdot D_n(x) \cdot \Omega(1),$$

which proves that domination is fulfilled. ■

Let us make a final remark on the function C . This function C is sometimes called a *compression function* for \mathcal{D} . It plays a crucial role in the

completeness proof, as it makes the reductions, which have to meet the domination requirement, possible in the first place. Why is C called compression function? Assume that we are given samples drawn according to \mathcal{D} . If we compress them using C , we have a compression with close to optimal compression rate.

13.3 Heuristic algorithms vs. heuristic schemes

In the previous chapter, we distinguished between algorithms and schemes: An algorithm has a fixed failure probability (fixed means a fixed function, not a fixed constant), whereas a scheme works for all failure probabilities δ , but the running-time depends on δ .

By Exercise 12.9, if a problem Π admits a heuristic scheme, then it admits heuristic algorithms with error probabilities n^{-c} for every constant c .

The containment in the other direction does not hold. For instance, $\text{Avg}_{1/n}\text{P}$ contains undecidable problems, whereas AvgP does not.

Exercise 13.1. 1. Show that there exists an undecidable problem L with $(L, \mathcal{U}) \in \text{Avg}_{1/n}\text{P}$.

2. Show that AvgP does not contain undecidable problems (L, \mathcal{U}) .

But if we restrict ourselves to problems in DistNP , the other containment can be proved: DistNP as a whole admits heuristic schemes if and only if it admits heuristic algorithms.

Theorem 13.6. Let $c > 0$ be arbitrary. If $(\text{BH}, \mathcal{U}^{\text{BH}}) \in \text{Avg}_{n^{-c}}\text{P}$, then $\text{DistNP} \subseteq \text{AvgP}$. The same holds for $\text{Heur}_{n^{-c}}\text{P}$ and HeurP .

Proof. For simplicity, we will only consider AvgP and $c = 1$. By the completeness of $(\text{BH}, \mathcal{U}^{\text{BH}})$, it suffices to show $(\text{BH}, \mathcal{U}^{\text{BH}}) \in \text{AvgP}$.

Let A be an errorless heuristic algorithm for $(\text{BH}, \mathcal{U}^{\text{BH}})$ with error probability $1/n$. We will use A to construct an errorless heuristic scheme A' . The idea is to use padding to map short instances of BH to longer instances. Then we exploit that the error probability of A decreases with growing input length.

Let N be the length of an instance $I = \langle g, x, 1^t \rangle$ of BH . Then we set

$$A'(I, N, \delta) = A \left(\langle g, x, 1^{t+1/\delta} \rangle, N + \frac{1}{\delta} \right).$$

(A' immediately rejects inputs that are not syntactically correct.) Note that

$$U_N^{\text{BH}}(I) = U_{N+\frac{1}{\delta}}^{\text{BH}}(\langle g, x, 1^{t+1/\delta} \rangle)$$

by the definition of U_N^{BH} . On inputs from $U_{N+\frac{1}{\delta}}^{\text{BH}}$, algorithm A outputs \perp with a probability of at most $1/(N + 1/\delta) < \delta$. Thus, A' outputs \perp on at most a δ fraction of the instances obtained from U_N^{BH} . ■

13.4 More DistNP-complete problems

Here, we list some more DistNP-complete problems without proving that they are. For the proofs as well as some other problems, we refer to Wang's survey of DistNP-complete problems [Wan97].

There are not too many DistNP-complete distributional problems (L, \mathcal{D}) , where both L and \mathcal{D} are natural. The main issue is that we lack a powerful tool to prove the a problem is hard-on-average like the PCP theorem for (in)approximability. So, in some sense, average-case complexity is in a similar state as the complexity of optimization was before the PCP theorem.

Tiling

A *tile* is a square with a symbol on each of its four sides. Tiles must not be rotated or turned over. If we have a set T of tiles, we assume that we have an infinite number of tiles of any kind in T . A *tiling* of an $n \times n$ square is an arrangement of n^2 tiles that cover the square such that the symbols of the adjacent sides of the tiles agree. The size of a tile is the length of the binary representation of its four symbols.

Instance: A finite set T of tiles, an integer $n > 0$, a sequence $s_1, \dots, s_k \in T$ of tiles that match each other (the right side of s_i matches the left side of s_{i+1}) The size of the instance is n plus the sizes of the tiles in T plus the sizes of s_1, \dots, s_k .

Question: Can s_1, \dots, s_k be extended to a tiling of the $n \times n$ square using only tiles from the set T ?

Distribution: Given n , select T using your favorite probability distribution (this really does not matter much; for the reduction, T just represents the Turing machine deciding a language, and this Turing machine has constant size for any fixed language). Select k uniformly at random from $\{1, \dots, n\}$. Finally, select s_1 uniformly from T and select s_{i+1} randomly from T such that it matches s_i .

Levin's original DistNP-complete problem was a variant of tiling, where the corners instead of the sides of the tiles had to match.

Post correspondence

The Post correspondence problem is one of the better known undecidable problems. In a restricted variant, it becomes NP-complete. Together with an appropriate probability distribution, it becomes DistNP-complete.

Instance: A positive integer n , and a list $\langle x_1, y_1 \rangle, \dots, \langle x_m, y_m \rangle$ of pairs of strings. The length N of the instance is $n + \sum_{i=1}^m |x_i| + |y_i|$.

Question: Is there a sequence $i_1, \dots, i_n \in \{1, \dots, m\}$ of indices such that $x_{i_1}x_{i_2} \dots x_{i_n} = y_{i_1}y_{i_2} \dots y_{i_n}$?

Distribution: Draw m according to $\Pr(m = \mu) = \Theta(1/\mu^2)$. Then draw x_1, \dots, x_m and y_1, \dots, y_m according to the uniform distribution on $\{0, 1\}^+$ defined in Section 12.1.

Arbitrary NP-complete problems

If some problem (L, \mathcal{D}) with $L \in \text{NP}$ and $\mathcal{D} \in \text{PComp}$ is hard-on-average, then every NP-complete language A is hard-on-average with respect to some samplable ensemble \mathcal{E} . The ensemble \mathcal{E} , however, might look a bit unnatural. In particular, for every NP-complete language A , there exists an ensemble $\mathcal{E} \in \text{PSamp}$ such that (A, \mathcal{E}) is DistNP-hard.

14 Average case versus worst case

In this section, we will show some connections between average-case and worst-case complexity. We will first provide a condition under which a distributional problem is *not* DistNP-complete unless $\text{EXP} = \text{NEXP}$. Second, we will show that DistNP is not contained in AvgP unless $\text{E} = \text{NE}$. (Recall that $\text{E} = \text{DTime}(2^{O(n)})$ and $\text{NE} = \text{NTime}(2^{O(n)})$, whereas $\text{EXP} = \text{DTime}(2^{\text{poly}(n)})$ and $\text{NEXP} = \text{DTime}(2^{\text{poly}(n)})$.)

14.1 Flatness and DistNP-complete problems

So under which conditions is a distributional problem DistNP-complete? Gurevich gave a partial answer: $\Pi = (L, \text{DistNP})$ cannot be DistNP-complete if \mathcal{D} assigns only very little weight to all strings in $\text{supp}(D_n)$. The intuition is the following: Assume that we have a distributional problem $\Psi = (A, \mathcal{E})$ that reduces to Π . Assume further that \mathcal{E} assigns high weight to few strings. Then, in order to satisfy the domination requirement, also \mathcal{D} must assign somewhat high weight to some strings. The following definition makes the notion of “assigns very little weight to all strings” precise.

Definition 14.1. *An ensemble $\mathcal{D} = (D_n)_{n \in \mathbb{N}}$ is flat if there exists an $\varepsilon > 0$ such that, for all n and x , $D_n(x) \leq 2^{-n^\varepsilon}$.*

Exercise 14.1. *Show that $\mathcal{G}_{1/2}$ (introduced in Exercise 12.8) is flat.*

Exercise 14.2. *Show that \mathcal{U}^{BH} (see Section 13.2) is not flat.*

Theorem 14.2 (Gurevich [Gur91]). *If there is a DistNP-complete problem (L, \mathcal{D}) with a flat ensemble \mathcal{D} , then $\text{NEXP} = \text{EXP}$.*

Proof overview: Assume that $\Pi = (L, \mathcal{D})$ is DistNP-complete, \mathcal{D} is flat, and there exists a reduction from $\Psi = (A, \mathcal{E})$ to Π , where $\mathcal{E} = (E_n)_{n \in \mathbb{N}}$ is non-flat. Let f be a reduction from Ψ to Π . In order to maintain domination, f must map strings $x \in \text{supp}(E_n)$ to very short strings $f(x)$. Short strings, however, mean (relatively) short running-time.

Proof. Assume that there exists a DistNP-complete distributional problem $\Pi = (L, \mathcal{D})$, where \mathcal{D} is a flat ensemble. Obviously, $L \in \text{EXP}$. Now let $A \in \text{NEXP}$ be arbitrary. Let p be a polynomial such that $A \in \text{NTime}(2^{p(n)})$. For a string $x \in \{0, 1\}^*$ with $|x| = n$, let $x' = x01^{2^{p(n)} - n - 1}$. Let $A' = \{x' \mid$

$x \in A$. Since $A \in \text{NEXP}$, the language A' is in NP. Let $\mathcal{E} = (E_n)_{n \in \mathbb{N}}$ be the following ensemble:

$$E_{2^{p(n)}}(z) = \begin{cases} 2^{-|x|} & \text{if } z = x' \text{ for some string } x \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Since \mathcal{E} is computable, we have $(A', \mathcal{E}) \in \text{DistNP}$. Thus, there exists a reduction f from (A', \mathcal{E}) to (L, \mathcal{D}) . Let us make a few observations.

- Given x of length n , $f(x', 2^{p(n)})$ can be computed in time $2^{q(n)}$ for some polynomial q .
- The function $x \mapsto f(x', 2^{p(n)})$ is a many-one reduction from A to L , i.e., $x \in A$ if and only if $f(x', 2^{p(n)}) \in L$.
- There exist polynomials m and r such that

$$\sum_{z: f(x', 2^{p(n)}) = f(z, 2^{p(n)})} E_{2^{p(n)}}(z) \leq r(2^{p(n)}) \cdot D_{m(2^{p(n)})}(f(x', 2^{p(n)}))$$

for all n and x . (This might look confusing at first glance since the strings x' , z and $f(x')$ are exponentially long, but it is just the domination condition.)

Now we have $E_{2^{p(n)}}(x') = 2^{-n}$. Thus, domination implies that

$$D_{m(2^{p(n)})}(f(x', 2^{p(n)})) \geq \frac{2^{-n}}{r(2^{p(n)})} = 2^{-s(n)} \quad (14.1)$$

for some polynomial s . Since \mathcal{D} is flat, there exists an $\varepsilon > 0$ such that

$$D_{m(2^{p(n)})}(f(x', 2^{p(n)})) \leq 2^{-(m(2^{p(n)}))^\varepsilon} \quad (14.2)$$

From the two bounds (14.1) and (14.2) on $D_{m(2^{p(n)})}$, we get

$$s(n) \geq m(2^{p(n)})^\varepsilon. \quad (14.3)$$

Now we are almost done. Since the images $f(x', 2^{p(n)})$ are in $\text{supp}(D_{m(2^{p(n)})})$, they are polynomially bounded in $m(2^{p(n)})$. By (14.3), this quantity is bounded from above by a polynomial $s(n)$. Hence, $A \in \text{EXP}$: (1) $x \in A$ if and only if $f(x', 2^{p(n)}) \in L$. (2) We can compute $y = f(x', 2^{p(n)})$ in time $2^{q(n)}$. (3) We can decide whether $y \in L$ in time $2^{p(|y|)} \leq 2^{p(m(2^{p(n)}))} \leq 2^{p(s(n))} = 2^{\text{poly}(n)}$, where the second inequality holds because of (14.3). ■

Since the uniform distribution $\mathcal{U} = (U_n)_{n \in \mathbb{N}}$ is flat, we immediately get the following result as a special case.

Corollary 14.3. *There is no $L \in \text{NP}$ such that (L, \mathcal{U}) is DistNP-complete unless $\text{NEXP} = \text{EXP}$.*

14.2 Collapse in exponential time

Our second result concerning connections between average-case and worst-case complexity shows that it is unlikely that DistNP is a subset of AvgP . If this is the case, then nondeterministic exponential time collapses to deterministic exponential time.

To show this, we need the following two lemmas.

Lemma 14.4. $E \neq \text{NE}$ if and only if there exists a unary language $L \in \text{NP} \setminus \text{P}$.

Proof. “ \implies ”: Assume that $\text{NE} \neq E$, and let $L' \in \text{NE} \setminus E$. Let $L = \{1^{\text{cod}(x)} \mid x \in L'\}$ be a unary language. Let us first show that $L \in \text{NP}$: Given $y = 1^{\text{cod}(x)}$, x can be computed in polynomial time. We have $|x| = O(\log |y|)$. Since $L' \in \text{NE}$, there exists a $2^{O(|y|)}$ time bounded nondeterministic Turing machine that decides L' . On input x , this machine needs time $2^{O(\log n)} = n^{O(1)}$.

Now let us prove that $L \notin \text{P}$. Assume to the contrary that $L \in \text{P}$. Then, given any string x , we can compute $y = 1^{\text{cod}(x)}$ in time $2^{O(n)}$. By definition, $y \in L$ if and only if $x \in L'$. Since $L \in \text{P}$, we can decide in time $|y|^{O(1)} = 2^{O(|x|)}$ if $y \in L$. This would imply $L' \in E$ – a contradiction.

“ \impliedby ”: Assume that there exists a unary language $L \subseteq \{1\}^*$ in $\text{NP} \setminus \text{P}$. Consider the language $L' = \{\text{bin}(n) \mid 1^n \in L\}$. We will show that $L' \in \text{NE} \setminus E$. On input y , we can compute an $x = 1^n$ with $\text{bin}(n) = y$ in time $2^{O(|y|)}$. Then we can use the nondeterministic polynomial-time Turing machine that witnesses $L \in \text{NP}$ to decide $x \in L$ in time $n^{O(1)} = 2^{O(|y|)}$. Thus, $L' \in \text{NE}$.

Lastly, we have to show that $L' \notin E$. Assume to the contrary that $L' \in E$. Then there is a deterministic $2^{O(n)}$ time bounded Turing machine that decides L' . Now, on input $x = 1^n$, we can compute $y = \text{bin}(n)$ in polynomial time. We have $|y| = O(\log n)$. Thus, $y \in L'$ can be decided in time $2^{O(\log n)} = n^{O(1)}$. Since $y \in L'$ if and only if $x \in L$, this would imply $L \in \text{P}$ – again a contradiction. ■

Lemma 14.5. Let $\mathcal{Q} = (Q_n)_{n \in \mathbb{N}}$ be given by $Q_n(1^n) = 1$. Then, for every unary language $L \subseteq \{1\}^*$, we have $L \in \text{P}$ if and only if $(L, \mathcal{Q}) \in \text{AvgP}$.

Proof. Clearly, if $L \in \text{P}$, then $(L, \mathcal{Q}) \in \text{AvgP}$. To see that the converse also holds, consider any algorithm A that witnesses $(L, \mathcal{Q}) \in \text{AvgP}$. Let t be the running-time of A . Then we have, for some $\varepsilon > 0$, $\mathbb{E}_{x \sim Q_n}(t^\varepsilon(x, n)) = O(n)$. Since $\text{supp}(Q_n) = \{1^n\}$, this is equivalent to $t(1^n, n) = O(n^{1/\varepsilon})$. Thus, A runs in worst-case polynomial time. ■

With these two lemmas, the main theorem of this section can be proved easily.

Theorem 14.6 (Ben-David, Chor, Goldreich, Luby [BDCGL92]). *If $E \neq \text{NE}$, then $\text{DistNP} \not\subseteq \text{AvgP}$.*

Proof. Let \mathcal{Q} be the ensemble of Lemma 14.5. Obviously, $\mathcal{D} \in \text{PComp}$. Thus, $L \in \text{NP}$ if and only if $(L, \mathcal{D}) \in \text{DistNP}$.

We have $\mathbf{E} \neq \mathbf{NE}$ if and only if there exists a unary language $L \in \text{NP} \setminus \text{P}$ by Lemma 14.4. This in turn holds if and only if $(L, \mathcal{Q}) \in \text{DistNP} \setminus \text{AvgP}$ by Lemma 14.5. ■

15 Decision versus search

A *search algorithm* for an NP relation V is an algorithm that, on input x , computes a witness w of length $\text{poly}(|x|)$ such that $\langle x, w \rangle \in V$. Recall that the corresponding language $L \in \text{NP}$ is $L = \{x \mid \exists^P w : \langle x, w \rangle \in V\}$. By abusing notation, we will call A also a search algorithm for a language $L \in \text{NP}$. This is ambiguous since L does not uniquely define a corresponding NP relation.

Obviously, if we have an efficient search algorithm for a language $L \in \text{NP}$, then we can use it to get an efficient decision algorithm for L . What about the opposite? If L is NP-complete, then we can use an efficient decision algorithm for L to efficiently compute witnesses (see the script of the lecture “Computational Complexity Theory”). Thus, if $\text{P} = \text{NP}$, then every problem in NP admits efficient search algorithms. So for NP as a whole, decision and search are equally hard. Nevertheless, it is believed that in general, efficient decision does not imply efficient search. For instance, one-way permutations, if they exist, yield problems for which decision is easy but search is hard.

Exercise 15.1. A one-way permutation is a bijective function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that

- $|f(x)| = |x|$ for every $x \in \{0, 1\}^*$,
- given x , $f(x)$ can be computed in polynomial time, and
- the problem of finding an x with $f(x) = y$ for a given y cannot be solved in polynomial time. (Since f is bijective, we can equivalently say that f^{-1} cannot be computed in polynomial time.)

Show that the existence of one-way permutations implies that there are problems for which search is harder than decision.

In this section, we consider the question of decision versus search in the average-case setting: Assume that all DistNP problems admit efficient-on-average decision algorithms, i.e., $\text{DistNP} \subseteq \text{AvgP}$. Do then all problems in DistNP also have efficient-on-average search algorithms? We will give a partial answer to this question: If all languages in NP with the uniform distribution admit efficient-on-average randomized algorithms, then all languages in NP with the uniform distribution admit efficient-on-average randomized search algorithms.

We have not yet defined what an efficient-on-average randomized algorithm is. (Here, the instances are drawn at random and, in addition, also the

algorithm itself is allowed to use randomness to solve the instance.) Furthermore, we also do not know yet what an efficient-on-average (randomized) search algorithm is. We will define all this in the next section and postpone the main theorem of this section to Section 15.3.

15.1 Randomized decision algorithms

We first generalize AvgP, Avg_δP and so on to randomized algorithms.

Definition 15.1. *Let $\Pi = (L, \mathcal{D})$ be a distributional problem. An algorithm A is a randomized errorless heuristic scheme for Π if A runs in time polynomial in n and $1/\delta$ for every $\delta > 0$ and $x \in \text{supp}(D_n)$ and*

$$\Pr_A(A(x, n, \delta) \notin \{L(x), \perp\}) \leq \frac{1}{4} \quad (15.1)$$

(the probability is taken over A 's coin tosses) and

$$\Pr_{x \sim D_n} \left(\Pr_A(A(x, n, \delta) = \perp) \geq \frac{1}{4} \right) \leq \delta \quad (15.2)$$

(the inner probability is again over A 's coin tosses, the outer probability over the random instances).

AvgBPP is the class of all distributional problems that admit a randomized errorless heuristic scheme.

We stress that “errorless” refers to the random input, not to the internal coin tosses of the algorithm.

Definition 15.1 probably need some explanation. Fix some input $x \in \text{supp}(D_n)$, consider running $A(x, n)$ k times for some large k . If significantly more than $k/4$ of these runs return \perp , then we can interpret this as A not knowing the answer for x . This follows from the second condition of Definition 15.1. On the other hand, if \perp is returned fewer than $k/4$ times, then the first condition guarantees that we will see the right answer at least $k/2$ times (with high probability due to Chernoff's bound). The choice of the constant $1/4$ in Definition 15.1 is arbitrary: Any constant strictly smaller than $1/3$ serves well.

Excursus: Chernoff bounds

Chernoff bounds are frequently used to bound large deviations from the expected value of random variables that are sums of independent indicator variables. The rough statement is: If we toss n unbiased coins, we see $n/2 \pm O(\sqrt{n})$ heads with high probability.

More precisely: Let X_1, \dots, X_n be independent random variables that assume only values in $\{0, 1\}$. Let $\Pr(X_i = 1) = p_i$, let $X = \sum_{i=1}^n X_i$, and let $E(X) = \sum_{i=1}^n p_i = \mu$. Then

$$\Pr(X > E(X) + a) < \exp\left(\frac{-2a^2}{n}\right)$$

for all $a > 0$. By symmetry, we have the same bound for $\Pr(X < E(X) - a)$.

There are many variants of Chernoff bounds. Sometimes, they lead to slightly different bounds. For most applications, however, it does not matter which version we use.

Exercise 15.2. *Let A be a randomized errorless heuristic scheme. Let A' be the an algorithm that executes A $k = k(n)$ times on inputs from $\text{supp}(D_n)$ and outputs the majority vote. Prove that*

$$\Pr_{A'}(A'(x, n, \delta) \notin \{L(x), \perp\}) \leq 2^{-\Omega(k(n))}$$

and

$$\Pr_{x \sim D_n} \left(\Pr_{A'}(A'(x, n, \delta) = \perp) \geq 2^{-\Omega(k(n))} \right) \leq \delta.$$

As in Definition 12.2, it is also possible to define randomized errorless heuristic algorithms or randomized heuristics that are allowed to make errors, but we will not do so here. We can also replace the constant $1/4$ in (15.1) by 0. Then we obtain zero-error randomized errorless heuristic schemes.

Now what is the difference between errorless and zero-error? Note that we have two types of “errors” or “failure”: We can be unlucky to get a hard instance, and the algorithm, since randomized, may fail. Errorless means that there is no instance on which the algorithm A errs. It is just allowed to produce \perp . However, if A is randomized it may still have bad luck with its coin tosses, which may cause it to output a wrong answer. If this is not the case, then A is called zero-error.

Exercise 15.3. *We can also define a non-uniform variant of AvgP: A distributional problem $\Pi = (L, \mathcal{D})$ is in AvgP/poly if there exists an algorithm A and an advice function $a : \mathbb{N} \times (0, 1] \rightarrow \{0, 1\}^*$ with $|a(n, \delta)| \leq \text{poly}(n, 1/\delta)$ such that the following holds:*

1. *For every n , every $\delta > 0$, and every $x \in \text{supp}(D_n)$, $A(x, n, \delta, a(n, \delta))$ outputs either $L(x)$ or the failure symbol \perp .*
2. *For every n , every $\delta > 0$, and every $x \in \text{supp}(D_n)$, $A(x, n, \delta, a(n, \delta))$ runs in time $p(n, 1/\delta)$.*
3. *For every n and every $\delta > 0$, we have $\Pr_{x \sim D_n}(A(x, n, \delta, a(n, \delta)) = \perp) \leq \delta$.*

(One might prefer to define AvgP/poly in terms of circuits rather than Turing machines that take advice. But we want to have one circuit for each n and δ , and $\text{supp}(D_n)$ can contain strings of different lengths. This technical problem can be solved, but why bother?)

Prove that AvgBPP \subseteq AvgP/poly.

15.2 Search algorithms

Now we turn to the definition of search algorithms. In order to avoid confusion, we will first define deterministic search algorithms that are efficient on average, although we will never use them. After that, we allow our search algorithms to use randomness.

Definition 15.2. *Let $\Pi = (L, \mathcal{D})$ be a distributional problem with $L \in \text{NP}$. An algorithm A is a deterministic errorless search scheme for Π if there is a polynomial p such that the following holds:*

1. *For every n , $\delta > 0$, and every $x \in \text{supp}(D_n)$, A runs in time at most $p(n, 1/\delta)$.*
2. *For every n , $\delta > 0$, and every $x \in L \cap \text{supp}(D_n)$, $A(x, n, \delta)$ outputs a witness for $x \in L$ or \perp .*
3. *For every n and $\delta > 0$, we have $\Pr_{x \sim D_n}(A(x, n, \delta) = \perp) \leq \delta$.*

For $x \notin L$, the algorithm $A(x, n, \delta)$ can output anything. The above definition is not completely precise since the witness language is not unique. However, this really makes no difference here.

In the next definition, we allow our search algorithm to use randomness.

Definition 15.3. *Let $\Pi = (L, \mathcal{D})$ be a distributional problem with $L \in \text{NP}$. An algorithm A is a randomized errorless search scheme for Π if there is a polynomial p such that the following is true:*

1. *For every n and $\delta > 0$, A runs in time $p(n, 1/\delta)$ and outputs either a string w or \perp .*
2. *For every n , $\delta > 0$, and $x \in L \cap \text{supp}(D_n)$,*

$$\Pr_A(A(x, n, \delta) \text{ outputs a witness for } x \text{ or } A(x, n, \delta) = \perp) > \frac{1}{2}.$$

3. *For every n and $\delta > 0$,*

$$\Pr_{x \sim D_n} \left(\Pr_A(A(x, n, \delta) = \perp) > \frac{1}{4} \right) \leq \delta.$$

What does this definition mean? For any $x \in L$, A may output a non-witness w . According to item (2), this happens with bounded probability. This is an internal failure of the algorithm A and not due to x being a hard instance. Item (3) bounds the probability that A outputs \perp . Intuitively, A outputs \perp not because of internal failure, but because x is a hard instance.

This is called an external failure. However, there is at most a δ fraction of strings x (measured with respect to D_n) on which A outputs \perp with significant probability.

The constants $1/2$ and $1/4$ in the definition are to some extent arbitrary. We can replace them by any constants c and c' with $1 > c > c' > 0$. Furthermore, these two failure probabilities can be decreased to $2^{-\Omega(k)}$ by executing the algorithm k times: If we ever get a witness, we output this witness. If we see \perp more than $c'k$ times, then we output \perp . Otherwise, we output an arbitrary string.

Definition 15.3 allows the algorithm A to output anything on input $x \notin L$ (but even then \perp only with bounded probability). Thus, a randomized errorless search scheme can be used as a randomized decision algorithm: If we get a witness, then we know that $x \in L$. If we get neither a witness nor \perp , then we claim that $x \notin L$. If the answer is \perp , then we do not know. By amplifying probabilities, we can make sure that the probability of outputting $x \notin L$ though there exists a witness for x is small.

15.3 Search-to-decision reduction

Recall that $\mathcal{U} = (U_n)_{n \in \mathbb{N}}$ is what we call the uniform distribution on $\{0, 1\}^*$. Namely, $U_n(x) = 2^{-n}$ for $|x| = n$ and $U_n(x) = 0$ otherwise.

In the following, we will reduce search to decision in the average-case setting. Let us first see why the usual approach from worst-case complexity does not work. Let $L = \{x \mid \exists^P w : \langle x, w \rangle \in V\} \in \text{NP}$ with being V the corresponding witness language. Then, given x and y , deciding if there exists a witness w that is lexicographically smaller than y is an NP language as well. Let

$$W = \{\langle x, y \rangle \mid \exists w : w \leq y \wedge \langle x, w \rangle \in V\},$$

where $w \leq y$ means “lexicographically smaller”. Assuming that decision is easy, namely $\text{P} = \text{NP}$, we can use binary search to find a witness w with $\langle x, w \rangle \in L$.

What about the average-case? Let w_x be the lexicographically smallest witness for x . Suppose our efficient-on-average algorithm for W works well on all instances $\langle x, y \rangle$ except for those y that are close to w_x . Then our algorithm is able to find the most significant bits of w_x , but it fails to find a few least significant bits of w_x . Since most strings are not close to w_x , our algorithm can still be efficient on average.

Our goal in the remainder of this section is still to prove that search-to-decision reduction is possible in the average-case setting, despite what we sketched above.

To do this, let us first consider the scenario where every $x \in L$ has a unique witness w_x . Then we can ask NP questions like “is the i th bit of the witness for x a 1?” Let p be the (polynomial) length of witnesses. By

querying the above for $i \in \{1, 2, \dots, p(|x|)\}$, we can find the witness. In the following, let $|x| = n$ and $p = p(n)$.

Of course, we cannot assume in general that witnesses are unique. But we know tools to make witnesses unique (recall the Valiant–Vazirani theorem from the lecture “Computational Complexity Theory” [VV86]). We use a family \mathcal{H} of pairwise independent hash functions $h : \{0, 1\}^n \rightarrow \{0, 1\}^p$: \mathcal{H} consists of all functions $x \mapsto Ax + b$, where $A \in \{0, 1\}^{p \times n}$ and $b \in \{0, 1\}^p$. By restricting $h \in \mathcal{H}$ to the first j bits, we obtain $h_{|j}$. Also $\{h_{|j} \mid h \in \mathcal{H}\}$ is a family of pairwise independent hash functions.

Now we consider the language

$$W' = \{\langle x, h, i, j \rangle \mid \exists w : \langle x, w \rangle \in V \wedge w_i = 1 \wedge h_{|j}(w) = 0^j\}.$$

We build the quadruple $\langle x, h, i, j \rangle$ such that, for $|x| = n$, it always has length $q = q(n)$ for some polynomial q . Furthermore, we make sure that x , h , i and j are independent. This means that we can equivalently draw $x \in \{0, 1\}^n$, $h \in \mathcal{H}$ as well as $i, j \in \{1, \dots, p\}$ uniformly and independently at random and pair them to $\langle x, h, i, j \rangle$. In this way, we get the same distribution. This can be done since the lengths of x and h is fixed once n is known. (We can, for instance, assume that p be a power of 2. Then we can write i and j as binary string of length $\log p$, possibly with leading 0s.)

It can be shown (see again the script of the lecture “Computational Complexity Theory”) that if j is the logarithm of the number of witnesses for x , then, with at least constant, positive probability (taken over the choice of $h \in \mathcal{H}$), there is a unique witness w for x that also satisfies $h_{|j}(w) = 0$.

Now we proceed as follows:

1. Draw $h \in \mathcal{H}$ uniformly at random.
2. If, for some $j \in \{1, \dots, p\}$, the sequence of answers to the queries $\langle x, h, 1, j \rangle, \dots, \langle x, h, p, j \rangle$ yields a witness w for $x \in L$, then we output this witness. (Note that $\langle x, w \rangle \in V$ can be checked in polynomial time.)
3. If an answer to some $\langle x, h, i, j \rangle$ is \perp , then we also output \perp . Otherwise, we output an arbitrary string.

We call this algorithm B . Apart from technical details, which we will prove below, this proves the following theorem. The essential subtlety is that h is part of the (random) input for W' , where h is part of the internal randomness of B for L . This means that h appears in the outer probability in (15.2) of Definition 15.1 and in the inner probability of item (3) of Definition 15.3.

Theorem 15.4 (Ben-David et al. [BDCGL92]). *If $(\text{NP}, \mathcal{U}) \subseteq \text{AvgBPP}$, then every problem in (NP, \mathcal{U}) has an errorless randomized search algorithm.*

Proof. We have already done the lion's share of the work. It remains to estimate the failure probabilities. Let $L \in \text{NP}$ be arbitrary such that witnesses for L have length p for some polynomial p , and let A be a randomized errorless heuristic scheme for (W', \mathcal{U}) with

$$W' = \{ \langle x, h, i, j \rangle \mid \exists w : \langle x, w \rangle \in V \wedge w_i = 1 \wedge h_{|j}(w) = 0^j \}.$$

Let $x \in L \cap \text{supp}(U_n) = L \cap \{0, 1\}^n$, and let $\delta > 0$ be arbitrary. Our search algorithm B proceeds as described above. We call A with a failure probability of α to achieve a failure probability of δ for B . Furthermore, we amplify item (2) of Definition 15.1 to

$$\Pr_A(A(y, q, \alpha) \notin \{W'(y), \perp\}) \leq \beta \quad (15.3)$$

for every $y = \langle x, h, i, j \rangle$ and item (3) to

$$\Pr_{y=\langle x, h, i, j \rangle \sim U_q} \left(\Pr_A(A(y, q, \alpha) = \perp) \geq \gamma \right) \leq \alpha.$$

We will specify α , β , and γ later on.

The algorithm described above, which we will call B , runs obviously in polynomial time. The failure probabilities remain to be analyzed: We have to find constants c' and c with $0 < c' < c < 1$ such that

$$\Pr_B(B(x, n, \delta) \text{ yields a witness or } \perp) \geq c \quad (15.4)$$

for each $x \in L \cap \{0, 1\}^n$ and

$$\Pr_{x \sim U_n} \left(\Pr_B(B(x, n, \delta) = \perp) > c' \right) \leq \delta \quad (15.5)$$

for every n and $\delta > 0$.

To show (15.4), consider any $x \in L$. The probability that we draw a hash function h with the property that there exists a j such that x possesses a unique witness w with $h_{|j}(w) = 0$ is at least $1/8$ (script of the lecture "Computational Complexity Theory", Lemma 17.3). We call such an h *good for x* . Fix an arbitrary good h . If B draws this h , then a sufficient condition for B output a witness or \perp is that A never outputs a wrong answer. The probability that A outputs a wrong answer (i.e., neither correct nor \perp) is at most $p^2\beta$ by a union bound over all i and j . Thus, the probability that

1. B samples an h that is good for x and
2. A never gives a wrong answer

is at least $c = \frac{1}{8} \cdot (1 - p^2\beta)$. We choose $\beta = \frac{1}{5p^2}$, which yields $c = 1/10$.

Before specifying the parameters α and γ , let us also analyze (15.5). To do this, let

$$Z = \left\{ x \in \{0, 1\}^n \mid \Pr_h \left(\exists i, j : \Pr_A(A(\langle x, h, i, j \rangle, q, \alpha) = \perp) \geq \gamma \right) \geq \phi \right\}$$

be the set of bad strings. Let us analyze the probability $\Pr_{x \sim U_n}(x \in Z)$ that a random x is bad. We have

$$\Pr_{x, h} \left(\exists i, j : \Pr_A(A(\langle x, h, i, j \rangle, q, \alpha) = \perp) \geq \gamma \right) \geq \phi \cdot \Pr_{x \sim U_n}(x \in Z).$$

Thus,

$$\underbrace{\Pr_{y=\langle x, h, i, j \rangle} \left(\Pr_A(A(y, q, \alpha) = \perp) \geq \gamma \right)}_{\leq \alpha \text{ by (15.3)}} \geq \frac{\phi \cdot \Pr_{x \sim U_n}(x \in Z)}{p^2}.$$

From this, we learn $\Pr_{x \sim U_n}(x \in Z) \leq \frac{\alpha p^2}{\phi}$. We want

$$\Pr_{x \sim U_n}(x \in Z) \leq \delta, \tag{15.6}$$

thus we put the constraint $\frac{\alpha p^2}{\phi} \leq \delta$. For $x \notin Z$, we have

$$\Pr_B(B(x, n, \delta) = \perp) \leq \phi + (1 - \phi)p^2\gamma.$$

Now we choose $\phi = 1/40$ and $\gamma = \frac{1}{40p^2}$. This also specifies α to $\alpha = \delta\phi/p^2$, which satisfies our constraint $\frac{\alpha p^2}{\phi} \leq \delta$. This specification of ϕ and γ yields

$$\Pr_B(B(x, n, \delta) = \perp) \leq \frac{1}{20}. \tag{15.7}$$

for $x \notin Z$. We set $c' = 1/20 < c$. Then item (3) of Definition 15.3 follows from (15.6) and (15.7). ■

16 Hardness amplification

Assume that we have a function f such that f is hard-on-average in a weak sense. This means that every algorithm has a non-negligible chance of making a mistake when evaluating f on a random input. But there still might be algorithms that get a huge portion (for instance, a $1 - 1/\text{poly}(n)$ fraction) of the inputs right. The goal of hardness amplification is the following: If there is such a function f , then we can get a related problem g from f such that g is hard-on-average in the strongest possible sense: No algorithm can do significantly better than simply tossing a fair coin. To put it the other way round: If, for some class of functions, we can compute every function in this class with a non-trivial error probability (i.e., significantly less than $1/2$). Then we can amplify this to get the error probability very small (i.e., $1 - 1/\text{poly}(n)$). (Note that this does not work by simple Chernoff bounds: it is the hardness of the instance that causes the algorithm to fail, not bad luck with its random bits. In fact, our algorithms in this section will always be deterministic.)

Yao's XOR lemma is a powerful tool for hardness amplification. The idea is simple: If f is slightly hard on average, then g , given by $g(x_1, \dots, x_k) = f(x_1) \oplus \dots \oplus f(x_k)$, is very hard on random x_1, \dots, x_k . The intuitive reason is as follows: Although the probability that a specific x is hard, the probability that at least one of x_1, \dots, x_k is hard is much higher. However, intuition says that we need all $f(x_1), \dots, f(x_k)$ to compute $g(x_1, \dots, x_k)$ correctly.

Exercise 16.1. Let $X_1, \dots, X_n \in \{0, 1\}^n$ be independent random variables with $\Pr(X_i = 1) = p$. Prove that

$$\Pr\left(\sum_{i=1}^n X_i \text{ is even}\right) = \frac{1 + (1 - 2p)^n}{2}.$$

For simplicity, we restrict ourselves to (non-uniform) circuits in the section. In the next section, we will state hardness amplification results for NP languages.

Definition 16.1. We say that a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is (s, δ) -hard with respect to a distribution D if, for every circuit C of size at most s , we have

$$\Pr_{x \sim D}(f(x) \neq C(x)) > \delta.$$

What does this mean? For every circuit C of size at most s , there exists a set H of size $2\delta 2^n$ such that using C to compute $f(x)$ for $x \in H$ is about as good as tossing a fair coin.

We also need the advantage of a circuit has in computing a certain function.

Definition 16.2. *Let f be a function, C be a circuit, and D be a distribution on inputs for f and C . If*

$$\Pr_{x \sim D}(f(x) = C(x)) = \frac{1}{2}(1 + \varepsilon),$$

then we say that C has an advantage of ε with respect to D . (By definition, the advantage ε is a number in the interval $[0, 1]$.)

We will prove Yao's XOR lemma in Section 16.2. There are several different proofs of this lemma. An elegant (and quite intuitive) proof is via Impagliazzo's hard-core set lemma (Section 16.1). This lemma is also interesting in its own right and a somewhat surprising result. There are at least two different proofs of this lemma. An elegant (and quite intuitive) proof is via von Neumann's min-max theorem. Impagliazzo attributes this proof to Nisan.

16.1 Impagliazzo's hard-core set lemma

Note the quantifiers in Definition 16.1: For all circuits C , a hard set H exists. Impagliazzo's hard-core set lemma states that we can switch the quantifiers: There exists a set H such that for all C computing f on H is hard. We will prove the hard-core set lemma in two steps: First, we show that there is a probability distribution over $\{0, 1\}^n$ such that f is hard with respect to this probability distribution. Second, we show how to get a set from this distribution.

Lemma 16.3. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be an (s, δ) -hard function with respect to the uniform distribution on $\{0, 1\}^n$, and let $\varepsilon > 0$. Then there is a probability distribution D on $\{0, 1\}^n$ with the following properties:*

1. f is $(s \cdot \frac{-\varepsilon^2}{8 \cdot \log(\varepsilon \delta)}, \frac{1}{2} - \varepsilon)$ -hard with respect to D .
2. $D(x) \leq \frac{1}{\delta} \cdot 2^{-n}$ for all $x \in \{0, 1\}^n$.

Proof overview: We have to switch quantifiers: We have “for all circuits, there exists a hard set”, and we want “there exists a hard set such that for all circuits”. We model this by a zero-sum game (see excursus below): One player's strategies are circuits C , the other player's strategies are sets H . The amount that the first player (who plays C) gets from the second player (who plays H) is proportional to the number of inputs of H that C gets right. Then we can use von Neumann's min-max theorem to switch quantifiers.

Proof. Consider the following two-player game: Player D picks a set T of $\delta 2^n$ strings from $\{0, 1\}^n$. Player C picks a circuit of size $s' = s \cdot \frac{-\varepsilon^2}{8 \cdot \log(\varepsilon \delta)}$. The payoff for C is $\Pr_{x \sim \{0, 1\}^n}(f(x) = C(x))$. This is a zero-sum game, and we can apply the min-max theorem: Either the player D has a mixed strategy so that there is no mixed strategy for player C for which C achieves a payoff of at least $\frac{1}{2} + \varepsilon$. Or there is a mixed strategy for player C with which C gets a payoff of at least $\frac{1}{2} + \varepsilon$ for any (mixed) strategy of player D.

Consider the first case. This means the following: There exists a distribution D' on sets of size $\delta 2^n$ such that every circuit C of size at most s , which corresponds to the pure strategies of player C, achieves only

$$\mathbb{E}_{T \sim D'} \Pr_{x \sim T}(f(x) = C(x)) \leq \frac{1}{2} + \varepsilon.$$

This is the same as the probability that $f(x) = C(x)$ if x is drawn according to the following distribution D : First, draw $T \sim D'$. Second, draw $x \in T$ uniformly at random. This probability distribution D is as stated in the lemma: For each $x_0 \in \{0, 1\}^n$, we have $\Pr_{x \sim D}(x = x_0) = \Pr_{T \sim D'}(x_0 \in T) \cdot \frac{1}{\delta} 2^{-n} \leq \frac{1}{\delta} \cdot 2^{-n}$. Thus, f is $(s, \frac{1}{2} - \varepsilon)$ -hard with respect to D . The lemma follows for this case since $s' \leq s$.

Now consider the second case. There exists a probability distribution \mathcal{C} on circuits of size s' such that, for every subset $T \subseteq \{0, 1\}^n$ of cardinality $\delta 2^n$, we have

$$\Pr_{x \sim T, C \sim \mathcal{C}}(C(x) = f(x)) \geq \frac{1}{2} + \varepsilon,$$

which corresponds to an average advantage of 2ε .

Let U be the set of inputs x for which the distribution \mathcal{C} on circuits achieves an advantage of at most ε in computing f . ("Advantage" is generalized to distributions over circuits in the obvious way.)

Claim 16.4. $|U| \leq \delta(1 - \varepsilon)2^n$.

Proof of Claim 16.4. Assume to the contrary that $|U| > \delta(1 - \varepsilon)2^n$. If $|U| \geq \delta 2^n$, then U would give rise to a strategy of player D to keep the payoff to at most $\frac{1}{2}(1 + \varepsilon)$, which contradicts the assumption.

Otherwise, consider any set $T' \supseteq U$ of cardinality $\delta 2^n$ for which \mathcal{C} achieves the smallest advantage. Since $|U| > \delta(1 - \varepsilon)2^n$, we have $|T' \setminus U| < \varepsilon \delta 2^n$. Then the advantage of \mathcal{C} on T' would be smaller than

$$\frac{1}{\delta 2^n} \cdot (|T' \cap U| \cdot \varepsilon + |T' \setminus U|) < \frac{1}{\delta 2^n} \cdot (\varepsilon \delta 2^n + \varepsilon \delta 2^n) = 2\varepsilon.$$

This contradicts the average advantage of at least 2ε on this set T' (which was the assumption of the second case). ■

Now we construct a circuit \tilde{C} of size s that gets $f(x)$ right for more than a $1 - \delta$ fraction of the inputs, which contradicts the assumption that f is (s, δ) -hard. The idea is as follows: On U , we might have only little chance to get the right answer. Thus, we ignore inputs from U . For inputs from $\{0, 1\}^n \setminus U$, however, we have a non-trivial chance of computing the right answer if we sample circuits according to the distribution \mathcal{C} , which is an optimal strategy for player C. Then we amplify probabilities by sampling more than one circuit and taking the majority outcome.

More precisely, let us draw t independent random circuits according to the distribution \mathcal{C} . Our new circuit \tilde{C} outputs the majority output of these t circuits. Fix any $x \notin U$. We can bound the probability that \tilde{C} gets $f(x)$ wrong using Chernoff bounds: \tilde{C} errs only if at most $t/2$ of its subcircuits give a wrong output. The expected number of correct outputs is at least $\frac{1}{2}(1 + \varepsilon)$. Thus, Chernoff bounds yields an upper bound of $\exp(-2(t\varepsilon/2)^2/t) = \exp(-t\varepsilon^2/2)$. We set $t = \frac{-4 \cdot \log(\varepsilon\delta)}{\varepsilon^2} \geq \frac{-2 \cdot \log(\varepsilon\delta/2)}{\varepsilon^2}$. This gives a probabilistic construction of \tilde{C} that errs on only a $\varepsilon\delta/2$ fraction of the inputs not from U . (Note that we do not need to construct \tilde{C} explicitly. Its existence suffices.) Since $|U| \leq \delta(1 - \varepsilon)2^n$, \tilde{C} errs with a probability of at most $\delta(1 - \varepsilon) + \varepsilon\delta/2 \leq \delta$ for random $x \in \{0, 1\}^n$.

Since \tilde{C} consists of t circuits of size s' , its size is at most $2ts' = s$. This contradicts the assumption that f is (s, δ) -hard. ■

Excursus: Min-max theorem

A *zero-sum game* is a game between two players such that the loss of one player is the gain of the other. A zero-sum game can be modeled by a matrix $A = (a_{i,j})_{1 \leq i \leq m, 1 \leq j \leq n} \in \mathbb{R}^{m \times n}$. The game consists of one player, called the maximizer, choosing an $i \in \{1, \dots, m\}$ and the other player, called minimizer, choosing a $j \in \{1, \dots, n\}$. Then the minimizer has to pay $a_{i,j}$ to the maximizer. (If $a_{i,j} < 0$, then the maximizer has to pay $-a_{i,j}$ to the minimizer.) The set $\{1, \dots, m\}$ is the set of *pure strategies* of the maximizer. The set $\{1, \dots, n\}$ is the set of pure strategies of the minimizer.

The order in which the players choose matters, as can be seen easily from the simple game with $m = n = 2$ and $a_{i,j} = (-1)^{i+j}$.

However, if we allow the players to use randomized strategies (so-called mixed strategies), then the order of play does not matter. This is what the min-max theorem by von Neumann [vN28] says.

More precisely: A *mixed strategy* is a probability distribution over the pure strategies of a player. In our case, it is simply a vector $p \in [0, 1]^n$ with $\sum_{j=1}^n p_j = 1$ for the minimizer and a vector $q \in [0, 1]^m$ with $\sum_{i=1}^m q_i = 1$ for the maximizer. The outcome of the game is then $q^T A p$. Then the min-max theorem says

$$\min_{\substack{p \in [0, 1]^n \\ \sum_{j=1}^n p_j = 1}} \max_{\substack{q \in [0, 1]^m \\ \sum_{i=1}^m q_i = 1}} q^T A p = \max_{\substack{q \in [0, 1]^m \\ \sum_{i=1}^m q_i = 1}} \min_{\substack{p \in [0, 1]^n \\ \sum_{j=1}^n p_j = 1}} q^T A p.$$

The number $\min_p \max_q q^T A p = \max_q \min_p q^T A p$ is called the *value* of the game.

The goal of the next lemma is to get a hard-core set from the hard-core distribution just constructed.

Lemma 16.5. *Let $D : \{0, 1\}^n \rightarrow [0, 1]$ be a probability distribution such that $D(x) \leq \frac{1}{8}2^{-n}$ for all $x \in \{0, 1\}^n$. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a function such that f is $(s, \frac{1}{2} - \frac{\varepsilon}{2})$ with respect to D for $2n < s < \frac{1}{16n}2^n(\varepsilon\delta)^2$*

Then there exists a set $H \subseteq \{0, 1\}^n$ such that f is $(s, \frac{1}{2} - \varepsilon)$ -hard with respect to the uniform distribution on H .

Proof overview: We use the probabilistic method: We draw a set according to the hard-core distribution. Then we take a union bound over all possible circuits to bound the probability that there exists a circuit that achieves a significant advantage on this (random) set. Since this probability will be bounded away from 1, there exists a set such that no circuit achieves a significant advantage on this set. This will be our hard-core set.

Proof. The construction of our set H will again be probabilistic. Let H be the random set obtained by placing x into H with a probability of $\delta 2^n D(x)$. The expected number of elements in H is $\delta 2^n$. With non-zero probability, this set H will have the desired property.

The number of circuits of size s is upper-bounded by

$$(2(2n + s))^{2s} \leq 2^{2ns} < \frac{1}{4} \exp\left(\frac{2^n \varepsilon^2 \delta^2}{8}\right).$$

Let C be any circuit of size s . Let

$$A_C(H) = |\{x \in H \mid f(x) = C(x)\}|$$

be the number of strings x that C gets right. We have

$$\mathbb{E}_H(A_C(H)) \leq \left(\frac{1}{2} + \frac{\varepsilon}{2}\right) \cdot \delta 2^n$$

by the assumption that $\Pr_{x \sim D}(C(x) = f(x)) \leq \frac{1}{2} + \frac{\varepsilon}{2}$ for every C .

The random variable $A_C(H)$ consists of 2^n independent indicator random variables, one for each string x . This brings Chernoff bounds into play:

$$\begin{aligned} \Pr_H\left(A_C(H) \geq \left(\frac{1}{2} + \frac{3\varepsilon}{4}\right) \cdot \delta 2^n\right) &\leq \Pr_H\left(A_C(H) \geq \mathbb{E}(A_C(H)) + \frac{\varepsilon\delta}{4}2^n\right) \\ &< \exp\left(-\frac{2\left(\frac{\varepsilon\delta}{4}2^n\right)^2}{2^n}\right) \\ &= \exp\left(-\frac{\varepsilon^2\delta^2 2^n}{8}\right). \end{aligned}$$

Furthermore, also by Chernoff bounds, H is unlikely to be small:

$$\Pr_H \left(|H| < \delta 2^n \cdot \left(1 - \frac{\varepsilon}{4}\right) \right) < \exp \left(-\frac{\varepsilon^2 \delta^2 2^n}{8} \right).$$

Now we take a union bound over all circuits of size s : The probability that, for a random set H , there exists a circuit C with $A_C(H) \geq \left(\frac{1}{2} + \frac{3\varepsilon}{4}\right) \cdot \delta 2^n$ or $|H| < \delta 2^n \cdot \left(1 - \frac{\varepsilon}{4}\right)$ is bounded by

$$\begin{aligned} & \Pr_H \left(\exists C : A_C(H) \geq \left(\frac{1}{2} + \frac{3\varepsilon}{4}\right) \cdot \delta 2^n \vee |H| < \delta 2^n \cdot \left(1 - \frac{\varepsilon}{4}\right) \right) \\ & \leq \frac{1}{4} \cdot \exp \left(\frac{\varepsilon^2 \delta^2 2^n}{8} \right) \cdot 2 \cdot \exp \left(-\frac{\varepsilon^2 \delta^2 2^n}{8} \right) = \frac{1}{2}. \end{aligned}$$

We can conclude that a set H with the desired properties exists with the following properties:

- $|H| \geq \delta 2^n \cdot \left(1 - \frac{\varepsilon}{4}\right)$.
- There is no circuit C of size s that gets more than $\left(\frac{1}{2} + \frac{3\varepsilon}{4}\right) \cdot \delta 2^n$ strings from x right.

If $|H| = \delta 2^n$, then we are done. If $|H| < \delta 2^n$, we add $\delta 2^n - |H|$ arbitrary elements to H and call the new set again H . No circuit gets more than $\left(\frac{1}{2} + \varepsilon\right) \cdot \delta 2^n$ strings of H right. If $|H| > \delta 2^n$, then we remove $|H| - \delta 2^n$ elements from H and again call the new set H . Since we only remove elements, no circuit gets more strings of the new set right than it got for the old set. Thus, this set also meets our requirements. ■

From the two lemmas above, the main result of this section follows easily.

Theorem 16.6 (Impagliazzo [Imp95]). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a function that is (s, δ) -hard with respect to the uniform distribution. Then, for every $\varepsilon > 0$, there exists a set $H \subseteq \{0, 1\}^n$ (called hard-core set for f) of cardinality at least $\delta 2^n$ such that f is $\left(s \cdot \frac{-\varepsilon^2}{64 \log(\varepsilon \delta)}, \frac{1}{2} - \varepsilon\right)$ -hard with respect to the uniform distribution over H .*

Proof. By Lemma 16.3, there exists a distribution D such that f is $\left(s \cdot \frac{-\varepsilon^2}{64 \log(\varepsilon \delta)}, \frac{1}{2} - \frac{\varepsilon}{2}\right)$ -hard with respect to D and D 's density is bounded by $\frac{1}{\delta} 2^{-n}$. (Note that we use $\varepsilon/2$ instead of ε , which yields the worse constant.) Now Lemma 16.5 shows the existence of a set H of cardinality $\delta 2^n$ such that f is $\left(s \cdot \frac{-\varepsilon^2}{64 \log(\varepsilon \delta)}, \frac{1}{2} - \varepsilon\right)$ -hard with respect to the uniform distribution on H . ■

Exercise 16.2. *It is, in fact, possible to show an even stronger statement: Assume that $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is (s, δ) -hard, and let $\eta > 0$ be an arbitrary constant. Then there exists a set $H \subseteq \{0, 1\}^n$ of cardinality at least $(2 -$*

$\eta)\delta 2^n$ such that f is $(s \cdot \text{poly}(\varepsilon, \delta, \eta), \frac{1}{2} - \varepsilon)$ -hard with respect to the uniform distribution.

Prove this!

Hint: Modify Lemma 16.3, then the rest follows.

This is close to optimal: Assume that there is a circuit C that errs only with a probability of δ , and consider any set H of cardinality significantly larger than $2\delta 2^n$. Then the probability that C errs on a random input is significantly smaller than $1/2$.

16.2 Yao's XOR lemma

The XOR lemma is attributed to Yao [Yao82]. The version that we present here is due to Impagliazzo [Imp95].

Theorem 16.7 (Yao's XOR lemma). *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be (s, δ) -hard with respect to the uniform distribution. Let $k \geq 1$, and let $g : \{0, 1\}^{kn} \rightarrow \{0, 1\}$ be given by*

$$g(x_1, \dots, x_k) = f(x_1) \oplus f(x_2) \oplus \dots \oplus f(x_k).$$

Then, for every $\varepsilon > 0$, the function g is $(s \cdot \frac{-\varepsilon^2}{100 \log(\varepsilon\delta)}, \frac{1}{2} - \varepsilon - (1 - \delta)^k)$ -hard with respect to the uniform distribution.

Proof overview: Let H be a hard-core set of f as in Theorem 16.6. The probability that one specific x_i is in H is δ . We ignore ε for the moment. If $x_i \in H$, then computing $g(x_1, \dots, x_k)$ correctly stay about the same if we replace $f(x_i)$ by a random bit b and compute

$$f(x_1) \oplus \dots \oplus f(x_{i-1}) \oplus b \oplus f(x_{i+1}) \oplus \dots \oplus f(x_k).$$

A random bit xor-ed with something is still a random bit. Thus, we get the right answer in this case only with probability $1/2$.

Our only hope is that none of x_1, \dots, x_k is in H . This happens with a probability of $(1 - \delta)^k$. Thus, we compute g correctly only with a probability of $\frac{1}{2} + (1 - \delta)^k$.

The problem with this proof idea is that a circuit for computing g does not necessarily proceeds by first computing $f(x_1), \dots, f(x_k)$. It is allowed to do anything. Nevertheless this idea can be turned into a proof of the XOR lemma.

Proof. Let H be a hard-core set for f of cardinality at least $\delta 2^n$ as in Theorem 16.6. Assume to the contrary that there exists a circuit C of size $s' = s \cdot \frac{-\varepsilon^2}{100 \log(\varepsilon\delta)}$ such that

$$\Pr_{x_1, \dots, x_k} (C(x_1, \dots, x_k) = g(x_1, \dots, x_k)) > \frac{1}{2} + (1 - \delta)^k + \varepsilon. \quad (16.1)$$

Let D be the uniform distribution over (x_1, \dots, x_k) with $x_i \in \{0, 1\}^n$ and conditioned on at least one x_i being in the hard-core set H . This yields

$$\begin{aligned} & \Pr_{(x_1, \dots, x_k) \sim D} (C(x_1, \dots, x_k) = g(x_1, \dots, x_k)) & (16.2) \\ & \geq \Pr_{x_1, \dots, x_k} (C(x_1, \dots, x_k) = g(x_1, \dots, x_k)) - \Pr_{x_1, \dots, x_k} (\exists i : x_i \in H) \\ & > \frac{1}{2} + \varepsilon. \end{aligned}$$

Let us take a different view on the distribution D : First, we pick a non-empty set $T \subseteq \{1, \dots, k\}$ with an appropriate distribution. Then, we choose $x_i \in H$ for $i \in T$ uniformly at random and $x_i \in \{0, 1\}^n \setminus H$ for $i \notin T$. Let the latter distribution be D_T . Thus, we can rewrite (16.2) as

$$\mathbb{E}_T \Pr_{(x_1, \dots, x_k) \sim D_T} (C(x_1, \dots, x_k) = g(x_1, \dots, x_k)) > \frac{1}{2} + \varepsilon.$$

Fix a set T that maximizes the inner probability. Without loss of generality, we assume that $1 \in T$. Then we can further rewrite the probability as

$$\mathbb{E}_{x_2, \dots, x_k} \Pr_{x_1 \sim H} (C(x_1, \dots, x_k) = g(x_1, \dots, x_k)) > \frac{1}{2} + \varepsilon,$$

where, by abusing notation, $x_1 \sim H$ means that x_1 is drawn uniformly at random from H . Now let a_j for $j > 1$ be the assignment for x_j that maximizes the above probability. This yields

$$\Pr_{x_1 \sim H} (C(x_1, a_2, \dots, a_k) \oplus f(a_2) \oplus \dots \oplus f(a_k) = f(x_1)) > \frac{1}{2} + \varepsilon,$$

where we have rearranged terms to isolate $f(x_1)$. To get a circuit for f from C , we replace x_2, \dots, x_k by the constants a_2, \dots, a_k . Then we observe that $f(a_2) \oplus \dots \oplus f(a_k)$ is a constant. Thus, we possibly have to negate the output of C . This increases the size by at most 1. Thus, we have a circuit C' of size $s' + 1 \leq s \cdot \frac{-\varepsilon^2}{64 \log(\varepsilon \delta)}$ for f that has success probability greater than $\frac{1}{2} + \varepsilon$ on the hard-core set H for f . This would contradict Theorem 16.6. ■

Using Exercise 16.2, we can even improve the hardness almost to $\frac{1}{2} - \varepsilon - (1 - 2\delta)^k$.

Analogously to Exercise 15.3, we can generalize $\text{Heur}_\delta \text{P}$ to non-uniform circuits of polynomial size in a straight-forward way. In this way, we obtain $\text{Heur}_\delta \text{P/poly}$.

Corollary 16.8. *Let \mathcal{C} be a class of Boolean functions with the following property: If $f = (f_n)_{n \in \mathbb{N}} \in \mathcal{C}$, then also $g \in \mathcal{C}$ with $g(x_1, \dots, x_k) = \bigoplus_{i=1}^k f(x_i)$, where k can be a function of $|x_i|$.*

Suppose there exists a family of functions $f = (f_n)_{n \in \mathbb{N}} \in \mathcal{C}$ such that $f \notin \text{Heur}_{1/p(n)} \text{P/poly}$. Then, for every constant $c > 0$, there exists a family g of functions such that $g \in \mathcal{C}$ and $g \notin \text{Heur}_{\frac{1}{2} - n^{-c}} \text{P/poly}$.

Exercise 16.3. *Prove Corollary 16.8.*

We can phrase the XOR lemma and Corollary 16.8 also the other way round: Assume that a class \mathcal{C} of functions is closed under \otimes , and suppose that every function in \mathcal{C} can be computed with a success probability of at least $\frac{1}{2} + \varepsilon$ for some not too small $\varepsilon > 0$. (Say, $\varepsilon = 1/\text{poly}(n)$.) Then we can reduce the error probability to $1/\text{poly}(n)$. (If this were not the case, then we would be able to amplify the hardness of $1/\text{poly}(n)$ to $\frac{1}{2} - \varepsilon$ – a contradiction.) So if we are able to compute a function with a non-trivial advantage, then we can bring the advantage close to 1. This is closely related to *boosting*, which is a concept in computational learning theory. Klivans and Servedio [KS03] explain the connections between boosting and hard-core sets.

17 Amplification within NP

Our goal in this section is to show a statement of the form “if there is a language in NP that is mildly hard on average, then there is a language in NP that is very hard on average.” Unfortunately, the XOR lemma does not yield such a result: If $L \in \text{NP}$, then it is unclear if computing $L(x) \oplus L(y)$ is also in NP. For instance, if L is co-NP-complete, then $L(x) \oplus L(y)$ can only be computed in NP if $\text{NP} = \text{co-NP}$.

We circumvent this problem by replacing parity by a monotone function $g : \{0, 1\}^k \rightarrow \{0, 1\}$. If $L \in \text{NP}$, then computing $g(L(x_1), \dots, L(x_k))$ from x_1, \dots, x_k is also in NP.

Exercise 17.1. *Prove the above statement. More precisely, prove the following stronger statement:*

Assume that $\text{NP} \neq \text{co-NP}$. Prove that the following two statements are equivalent for any function $g : \{0, 1\}^k \rightarrow \{0, 1\}$:

1. *For all $L \in \text{NP}$, also $\{(x_1, \dots, x_k) \mid g(L(x_1), \dots, L(x_k)) = 1\} \in \text{NP}$.*
2. *g is monotonically increasing. This means that for any $y, z \in \{0, 1\}^k$ with $y \leq z$ (component-wise), we have $g(y) \leq g(z)$.*

17.1 The main idea

For the results of this chapter, which mainly due to O’Donnell [O’D04], we need some preparation. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$, and let $g : \{0, 1\}^k \rightarrow \{0, 1\}$. Then $g \otimes f : (\{0, 1\}^n)^k \rightarrow \{0, 1\}$ denotes the function given by $(g \otimes f)(x_1, \dots, x_k) = g(f(x_1), \dots, f(x_k))$. Our goal is to analyze the hardness of $g \otimes f$ in terms of properties of g and the hardness of f . The property of g that we need is the bias of g or, more precisely, the expected bias of g subject to a random restriction.

Definition 17.1. *The bias of a Boolean function h is*

$$\text{bias}(h) = \max_x \{\Pr(h(x) = 0), \Pr(h(x) = 1)\} \in [1/2, 1].$$

The function h is called balanced if $\text{bias}(h) = 1/2$.

In fact, not the bias of g is the parameter that plays a role, but the expected bias of g with respect to a random restriction.

Definition 17.2. A restriction ρ of a function $h : \{0, 1\}^m \rightarrow \{0, 1\}$ is a mapping $\rho : \{1, 2, \dots, m\} \rightarrow \{0, 1, \star\}$. Then h_ρ denotes the subfunction of h obtained by substituting each coordinate i with $\rho(i) \in \{0, 1\}$ by $\rho(i)$.

For a $\delta \in [0, 1]$, we denote by P_δ^m the probability space over all restrictions, where a restriction ρ is drawn according to the following rules:

- $\rho(1), \dots, \rho(m)$ are independent.
- $\Pr(\rho(i) = \star) = \delta$.
- $\Pr(\rho(i) = 0) = \Pr(\rho(i) = 1) = \frac{1-\delta}{2}$.

If $\rho \sim P_\delta^m$, then ρ is called a random restriction with parameter δ .

The expected bias of h at δ is

$$\text{EBias}_\delta(h) = \mathbb{E}_{\rho \sim P_\delta^m}(\text{bias}(h_\rho)).$$

Exercise 17.2. Let parity_m be the parity function of m bits. Compute $\text{EBias}_\delta(\text{parity}_m)$.

Give estimates for $\text{EBias}_\delta(\text{and}_m)$, where and_m is the AND of m bits.

The main result of this chapter is the following theorem, which is a generalized version of the XOR lemma. It states the hardness of $g \otimes f$ in terms of the hardness of f and the expected bias of g . The technical restriction is that we require that the function f be balanced.

Theorem 17.3. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be an (s, δ) -hard balanced function, and let $g : \{0, 1\}^k \rightarrow \{0, 1\}$ be arbitrary. Then, for every $\eta > 0$, the function $g \otimes f$ is $(s', 1 - \text{EBias}_{(2-\eta)\delta}(g) - \varepsilon)$ -hard, where $s' = \Omega\left(s \cdot \frac{\varepsilon^2}{\log(1/\delta)k}\right)$.

We will not give a full proof of the result. Rather, we will give an intuition why it should be true. In the next sections, we will discuss which functions g are suitable to amplify hardness.

Proof overview: Suppose that x_1, \dots, x_k are drawn at random. Our task is to compute $(g \otimes f)(x_1, \dots, x_k)$, where f is both balanced and δ -hard. We model the hardness of f by computing $g(y_1, \dots, y_k)$ with imperfect information about y_1, \dots, y_k . This means that $y_i = f(x_i)$, but the hardness of f obscures the true values and we see only corrupted values z_1, \dots, z_k . Since f is balanced, $\Pr(y_i = 1) = \Pr(y_i = 0) = 1/2$, i.e., the values y_1, \dots, y_k are drawn uniformly and independently at random. Since f is δ -hard, we have $\Pr(z_i = y_i) \leq 1 - \delta$.

We abstract away f and just use the δ -hardness of f . We model this by setting drawing z_i according to $\Pr(z_i = y_i) = 1 - \delta$ and $\Pr(z_i \neq y_i) = \delta$.

Now we might take simply output $g(z_1, \dots, z_k)$. Then we would compute the correct value $g(y_1, \dots, y_k)$ with a probability of $\text{NStab}_\delta(g)$. (The noise stability, denoted by NStab , is defined below.) More sophisticated, we might

compute $g(z')$ for different z' close to $z = (z_1, \dots, z_k)$ and output a maximum likelihood answer.

However, in the true setting involving f , we might not only have $z_i = y_i$, but we may also know that z_i is correct. Taking this to its extreme, we get the following scenario: With a probability of $1 - 2\delta$, we have $z_i = y_i$, and we know for sure that $z_i = y_i$. With a probability of 2δ , z_i is a random bit, and we know that z_i is a corrupted bit. (But, of course, we do not know if $z_i = y_i$ or $z_i \neq y_i$. Question: Why 2δ , where f is only δ -hard?)

So what can we do now? We take the values z_i for which we are certain that $z_i = y_i$ for granted, and we replace the corrupted values by \star . In this way, we obtain a restriction ρ . Then we compute $\Pr_a(g_\rho(a) = 0)$ and $\Pr_a(g_\rho(a) = 1)$. If the first is larger, then we output 0, otherwise, we output 1. The error probability is thus $1 - \text{bias}(g_\rho)$.

To compute the overall probability that this strategy succeeds, we must take into account that ρ is in fact a random restriction, drawn from $P_k^{2\delta}$. Thus, the probability of outputting the correct answer is nothing else but $\text{EBias}(g)$. The other way round, this looks as if $g \otimes f$ be $\text{EBias}_{2\delta}(g)$ -hard.

The idea sketched above can be turned into a proof. It proceeds as follows: First, one can show that f does not only possess a hard-core set, but a balanced hard-core set H . This is not surprising: If a hard-core set H is not balanced, then either always outputting 1 or always outputting 0 gives a non-trivial advantage. Then we transfer the idea using arguments similar to those of the proof of Theorem 16.7.

Exercise 17.3. Use Theorem 17.3 and Exercise 17.2 to derive a weaker form of the XOR lemma (Theorem 16.7), which holds only for balanced functions.

17.2 Noise stability and expected bias

The expected bias is closely related to another measure for Boolean functions, called noise stability. Lemma 17.5 states this connection precisely. We use noise stability since it is sometimes easier to compute, although the expected bias is the “right” parameter for Theorem 17.3.

Definition 17.4. The noise stability of a Boolean function $h : \{0, 1\}^m \rightarrow \{0, 1\}$ is defined as

$$\text{NStab}_\delta(h) = \Pr_{x \sim \{0,1\}^m, y \sim N_\delta(x)} (f(x) = f(y)),$$

where x is drawn uniformly at random and y is obtained from x by flipping each bit of x independently with a probability of δ .

The quantity

$$\text{NSens}_\delta(h) = 1 - \text{NStab}_\delta(h) = \Pr_{x \sim \{0,1\}^m, y \sim N_\delta(x)}(f(x) \neq f(y))$$

is called the noise sensitivity of h .

Exercise 17.4. Compute $\text{NStab}_\delta(\text{parity}_m)$ and $\text{NStab}_\delta(\text{and}_m)$.

Depending on the context, either noise stability or noise sensitivity will be more convenient to use or to analyze.

In the following, let $x^* = 2x - 1 \in [0, 1]$ for any quantity $x \in [1/2, 1]$.

Lemma 17.5. For any Boolean function $h : \{0, 1\}^m \rightarrow \{0, 1\}$, we have

$$\text{NStab}_\delta^*(h) \leq \text{EBias}_{2\delta}^*(h) \leq \sqrt{\text{NStab}_\delta^*(h)}.$$

Proof. We exploit the following fact.

Exercise 17.5. Prove the following: For any Boolean function h and independently and uniformly drawn x and y , we have $h(x) = h(y)$ with a probability of $\frac{1}{2} + \frac{1}{2} \text{bias}(h)^2$. In other words,

$$\frac{1}{2} + \frac{1}{2} (\text{bias}(h)^*)^2 = \text{NStab}_{1/2}(h).$$

We take a different view on $\text{NStab}_\delta(h)$: First, we draw $\rho \in P_{2\delta}^m$. We set $x_i = y_i = \rho_i$ if $\rho_i \neq \star$. For $\rho_i = \star$, we draw $x_i, y_i \in \{0, 1\}$ uniformly and independently at random. Then x is drawn uniformly at random and y_i differs from x_i with a probability of δ . Furthermore, x and y are identically distributed. And given ρ , they are drawn independently. Let x' and y' be the vectors obtained by removing all positions i with $\rho_i \neq \star$.

Together with Exercise 17.5, we get

$$\begin{aligned} \text{NStab}_\delta(h) &= \mathbb{E}_{\rho \sim P_{2\delta}^m} \Pr_{x', y'}(h_\rho(x') = h_\rho(y')) \\ &= \mathbb{E}_{\rho \sim P_{2\delta}^m} \left(\frac{1}{2} + \frac{1}{2} \text{bias}(h_\rho)^2 \right). \end{aligned}$$

Thus,

$$\text{NStab}_\delta^*(h) = \mathbb{E}_{\rho \sim P_{2\delta}^m} (\text{bias}^*(h_\rho)^2).$$

by linearity of expectation.

Since $\text{bias}^*(h_\rho) \in [0, 1]$, we have $\text{bias}^*(h_\rho)^2 \leq \text{bias}^*(h_\rho)$, which yields the first inequality. The second inequality follows from Jensen's inequality since squaring is a convex function:

$$\begin{aligned} \sqrt{\text{NStab}_\delta^*(h)} &= \sqrt{\mathbb{E}_{\rho \sim P_{2\delta}^m} (\text{bias}^*(h_\rho)^2)} \\ &\geq \mathbb{E}_{\rho \sim P_{2\delta}^m} \left(\sqrt{\text{bias}^*(h_\rho)^2} \right) = \mathbb{E}_{\rho \sim P_{2\delta}^m} (\text{bias}^*(h_\rho)) \\ &= \text{EBias}_{2\delta}^*(h). \end{aligned}$$

■

The following lemma will be very useful to compute the noise stability and noise sensitivity of the functions g that we use to amplify hardness.

Lemma 17.6. *Let $h : \{0, 1\}^m \rightarrow \{0, 1\}$ be a balanced Boolean function, and let $g : \{0, 1\}^k \rightarrow \{0, 1\}$ be an arbitrary Boolean function. Then*

$$\text{NSens}_\delta(g \otimes h) = \text{NSens}_{\text{NSens}_\delta(h)}(g).$$

Proof. Let us take a closer look at

$$\begin{aligned} & \text{NSens}_\delta(g \otimes h) \\ = & \Pr_{\substack{x_1, \dots, x_k \in \{0, 1\}^m \\ y_1 \sim N_\delta(x_1), \dots, \\ y_k \sim N_\delta(x_k)}} (g(h(x_1), \dots, h(x_k)) \neq g(h(y_1), \dots, h(y_k))) . \end{aligned}$$

Let $z_i = h(x_i)$ and $z'_i = h(y_i)$. Then $\Pr(z_i = 1) = \Pr(z_i = 0) = 1/2$ since h is balanced. Furthermore, the probability that $z'_i \neq z_i$ is just $\text{NSens}_\delta(h) = 1 - \text{NStab}_\delta(h)$. Thus,

$$\begin{aligned} \text{NSens}_\delta(g \otimes h) &= \Pr_{\substack{z \in \{0, 1\}^k \\ z' \sim N_{\text{NSens}_\delta(h)}}} (g(z) \neq g(z')) \\ &= \text{NSens}_{\text{NSens}_\delta(h)}(g) \end{aligned}$$

as claimed. ■

17.3 Recursive majority-of-three and tribes

The function g should be nearly balanced subject to a random restriction in order to keep $\text{EBias}_\delta(g)$ close to $1/2$.

Two functions that turn out to be very useful: The first one is the “recursive majority of 3” function, which we will define recursively: Let $M_k : \{0, 1\}^{3^k} \rightarrow \{0, 1\}$. Then $M_1(x, y, z) = 1$ if and only if at least two of x, y , and z are set to 1. For $k \geq 1$, we have $M_{k+1} = M_1 \otimes M_k$. This means that

$$\begin{aligned} & M_{k+1}(x_1, \dots, x_{3^k}, y_1, \dots, y_{3^k}, y_1, \dots, y_{3^k}) \\ = & M_1(M_k(x_1, \dots, x_{3^k}), M_k(y_1, \dots, y_{3^k}), M_k(y_1, \dots, y_{3^k})) . \end{aligned}$$

Lemma 17.7. *For $\ell \geq \log_{1.1}(1/\delta)$, we have $\text{NStab}_\delta^*(M_\ell) \leq \delta^{-1.1} \cdot (3^\ell)^{-0.15}$.*

Exercise 17.6. *Prove Lemma 17.7. You can also show a slightly weaker variant: Prove that there exist constants $a > 1, b \geq 1, c > 0$ such that for $\ell \geq \log_a(1/\delta)$, we have $\text{NStab}_\delta(M_\ell)^* \leq \delta^{-b}(3^\ell)^{-c}$.*

Hint: Calculate $\text{NSens}_\delta(M_1)$ explicitly. Then use Lemma 17.6. Make a case distinction whether $\text{NSens}_\delta(M_k)$ is large or small for $k \leq \ell$.

Majority-of-three is helpful to amplify a $(1/\text{poly}(n))$ -hard language to become somewhat hard, namely to $(\frac{1}{2} - n^{-\alpha})$ -hard for some small constant $\alpha > 0$.

To amplify further to $(\frac{1}{2} - n^{-1/2+\eta})$ -hardness, we use our second function, which is called “tribes”. Tribes does particularly well if the function whose hardness we want to amplify is already somewhat hard.

Let $w \in \mathbb{N}$, and let $n = n(w) \in \mathbb{N}$ be the smallest multiple of w such that $(1 - 2^{-w})^{n/w} \leq 1/2$. Then the tribes function T_n of n variables is defined as

$$T_n(x_1, \dots, x_n) = \bigvee_{i=0}^{n/w-1} \bigwedge_{j=1}^w x_{iw+j}.$$

If we write w as a function of n , we get $w = \log n - \log \ln n + o(1)$.

To estimate the expected bias of tribes is technically more challenging than it was for majority-of-three, and we will omit a proof here.

Lemma 17.8. *For every constant $\eta > 0$, there is a constant $r > 0$ such that $\text{EBias}_{1-r}(T_n) \leq \frac{1}{2} + n^{-1/2+\eta}$.*

Exercise 17.7. *Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function. The influence of the j -th variable is defined as*

$$I_j = \Pr_{x \sim \{0,1\}^n} (f(x) \neq f(x^{(j)})),$$

where $x^{(j)}$ is obtained from x by flipping the j -th entry of x . The total influence of f is $I = \sum_{j=1}^n I_j$.

Compute (or give as good as possible estimates for) $I(\text{and}_m)$, $I(\text{parity}_m)$, $I(\text{majority}_m)$, $I(M_k)$, $I(T_k)$. ($\text{majority}_m(x_1, \dots, x_m)$ is 1 if and only if at least $m/2$ of the x_i are 1.)

Exercise 17.8. *Show that*

$$\text{NSens}_\delta(f) \leq \delta \cdot I(f).$$

17.4 Hardness within NP

Recursive majority-of-three turns out to be very useful to amplify δ -hardness for relatively small values of δ . More precisely, if f is $1/\text{poly}(n)$ -hard, then recursive majority-of-three we can amplify its hardness to $\frac{1}{2} - n^{-\alpha}$ for some small constant α . Then tribes comes into play, which can, if δ is not too small, bring the hardness to $\frac{1}{2} - n^{-1/2+\varepsilon}$ for arbitrarily small $\varepsilon > 0$.

Using Lemma 17.7 and Theorem 17.3, we can show that majority-of-three can amplify hardness close to $1/2$.

Lemma 17.9. *If there is a family of functions in NP which is infinitely often balanced and $(\text{poly}(n), 1/\text{poly}(n))$ -hard. (This means that this function is $1/\text{poly}(n)$ -hard for circuits of polynomial size.) Then there is a family of functions (h_m) in NP that is infinitely often balanced and $(1/2 + m^{-0.07})$ -hard for circuits of polynomial size.*

Proof. Suppose a family $f = (f_n)$ of functions is infinitely often balanced and $1/n^c$ -hard for polynomial-size circuits. Choose $k = n^C$ for some sufficiently large constant C , and set $\ell = \log_3(k) = C \log_3 n$ for some sufficiently large constant C .

Let $h_m = M_\ell \otimes f_n$. The function h_m has input length $m = nk = n^{C+1}$. The family $h = (h_m)_{m \in \mathbb{N}}$ is in NP since f is in NP and M_ℓ is monotone and in P. Moreover, h_m is balanced whenever f_n is balanced.

We have to show that h_m is $(1/2 - m^{-0.07})$ -hard whenever f_n is hard and balanced. We apply Theorem 17.3 with $\eta = 1$, $\varepsilon = n^{-C}$, and $\delta = n^{-c}$.

Lemmas 17.5 (for converting noise stability to expected bias) and 17.7 (for the noise stability of M_ℓ) yield

$$\text{EBias}_\delta(M_\ell) \leq \frac{1}{2} + \frac{1}{2} \left(\frac{\delta}{2} \right)^{-0.55} 3^{-0.075\ell}.$$

This assumes that ℓ is sufficiently large, which can be ensured by choosing C large.

Now we observe that

$$\frac{1}{2} \left(\frac{\delta}{2} \right)^{-0.55} 3^{-0.075\ell} \leq (2n^c)^{0.55} \left(\frac{n^C}{3} \right)^{-0.075} + n^{-C} \leq n^{-0.074C}$$

for large enough (but still constant) C . Finally, $n^{-0.074C} \leq m^{-0.07}$ for large enough C since $m = n^{C+1}$. ■

Using the tribes function, we can further improve the hardness.

Theorem 17.10. *Suppose that there is a family $f = (f_n)_{n \in \mathbb{N}}$ of functions in NP which is infinitely often balanced and $(\text{poly}(n), 1/\text{poly}(n))$ -hard. (This means that f_n is $1/\text{poly}(n)$ -hard for circuits of polynomial size.) Then there is a family of functions in NP which is infinitely often $(\text{poly}(n), \frac{1}{2} - n^{-1/2+\varepsilon})$ -hard for any $\varepsilon > 0$.*

Exercise 17.9. *Prove Theorem 17.10 using the tools Lemma 17.8, Lemma 17.9, and Theorem 17.3.*

At the expense of a small loss in the final hardness, we get even get rid of the requirement that the initial function is balanced.

Theorem 17.11. *Suppose that there is a family of functions in NP which is infinitely often $(\text{poly}(n), 1/\text{poly}(n))$ -hard. Then there is a family of functions in NP which is infinitely often $(\text{poly}(n), \frac{1}{2} - n^{-1/3+\varepsilon})$ -hard.*

We can rephrase Theorem 17.11 in “boosting form”, which sounds more positive.

Theorem 17.11’. *Suppose that $(L, \mathcal{U}) \in \text{Heur}_{\frac{1}{2}-n^{-0.33}}\text{P/poly}$ for all $L \in \text{NP}$. Then $(L, \mathcal{U}) \in \text{Heur}_{1/p}\text{P/poly}$ for every polynomial p and every language $L \in \text{NP}$.*

Exercise 17.10. *If we allow arbitrary Boolean functions (which are not required to be from, say, NP or PSPACE), we can even find a function that is exponentially close to 1/2-hard for circuits of exponential size.*

Prove the following: There exists a universal constant $\gamma \geq 1/8$ such that, for all sufficiently large $n \in \mathbb{N}$, there exists a function $h : \{0, 1\}^n \rightarrow \{0, 1\}$ which is $(2^{\gamma n}, \frac{1}{2} - 2^{-\gamma n})$ -hard.

This is almost as hard as possible: No function is harder than 1/2-hard even for very small circuits. And just by hard-wiring the correct function value for one input and outputting either 0 or 1 on all other inputs (depending on whether more function values are 0 or 1), we can bring the hardness down to $\frac{1}{2} - 2^{-n}$.

18 Natural proofs

18.1 Pseudorandom generators

Definition 18.1. Let $g = (g_n)$ be a family of functions, $g_n : \{0, 1\}^n \rightarrow \{0, 1\}^m$ with $m = m(n) > n$. g is $(\delta(n), s(n))$ -pseudorandom against circuits if for all $s(n)$ size bounded families of circuits $C = (C_n)$,

$$\left| \Pr_{y \in \{0, 1\}^m} [C_m(y) = 1] - \Pr_{x \in \{0, 1\}^n} [C_m(g(x)) = 1] \right| \leq \delta(|x|)$$

for all $n \geq n_0$ for some constant n_0 .

In the same way, we can define functions that are pseudorandom against PTMs, DTMs etc. A PTM computes a function, if it outputs the right value with probability $\geq 3/4$.

Exercise 18.1. Show that if a family g is $(\delta(n), s(n))$ -pseudorandom against nonuniform circuits, then g is $(\delta(n), c \cdot s(n))$ -pseudorandom against PTMs for some constant $c > 0$.

We will need pseudorandom generators to construct pseudorandom functions. But we can use them to derandomize BPP, too.

Definition 18.2. $\text{SUBEXP} = \bigcap_{\epsilon > 0} \text{DTime}(2^{n^\epsilon})$.

Theorem 18.3. If for every $c, \ell > 0$, there is a poly-time computable family $g = (g_n)$ with $g_n : \{0, 1\}^n \rightarrow \{0, 1\}^{n^c}$ that is $(1/8, O(n^\ell))$ -pseudorandom against non-uniform circuits, then $\text{BPP} \subseteq \text{SUBEXP}$.

Proof. We will show that $\text{BPP} \subseteq \text{DTime}(2^{n^\epsilon})$ for every $\epsilon > 0$.

Let M be a BPP machine¹ with running time $p(n)$ for some polynomial p . Choose g such that it stretches $\ell = n^\epsilon$ bits into $m(\ell) \geq p(n)$ bits and is $(\frac{1}{8}, p(\ell))$ -pseudorandom. We build a new BPP machine \hat{M} for the same language that takes n^ϵ random bits, stretches them to $p(n)$ bits, and then simulates M using the pseudorandom string. We derandomize \hat{M} by enumerating all 2^{n^ϵ} seeds and taking a majority vote, see Algorithm 1.

The running time of the deterministic simulation is $O(2^{n^\epsilon} \cdot p(n))$. The deterministic simulation of \hat{M} is obviously correct. It remains to show that M and \hat{M} accept the same language (except for a finite number of strings).

¹A BPP machine is a poly-time PTM with twosided error probability that is bounded by $1/3$. In the same way, we will use the terms NP machine, etc.

Algorithm 1 Deterministic Simulation**Input:** w

```

1: for all  $r \in \{0, 1\}^{n^\epsilon}$  do
2:   Stretch  $r$  to  $r' \in \{0, 1\}^{p(n)}$  using  $g$ .
3:   Simulate  $M$  on input  $w$  with random string  $r'$ .
4: end for
5: if in the majority of all simulations,  $M$  accepted then
6:   return 1
7: else
8:   return 0
9: end if

```

Suppose that this is not the case: Then w.l.o.g. there is an infinite sequence of inputs w_1, w_2, \dots such that $M(w_i) \neq \hat{M}(w_i)$. Here $\hat{M}(w_i)$ need not have bounded error probability. But this means that for all i

$$\left| \Pr_{y \in \{0,1\}^m} [M(w_i, y) = 1] - \Pr_{x \in \{0,1\}^n} [M(w_i, g(x)) = 1] \right| \geq \frac{1}{6}$$

since the first probability is at least $2/3$ and the other one is at most $1/2$ or the first is at most $1/3$ and the other one is $> 1/2$. Now instead of viewing M as a PTM with input w_i and random string y , we view it as a DTM with input y and advice string w_i . Since $p(n)$ -time Turing machines with advice are equivalent to $O(p(n))$ -size bounded nonuniform families of circuits, we found a family of nonuniform circuits of polynomial size that has distinguishing probability $1/6 > 1/8$ for infinitely many input lengths, a contradiction. ■

One of the central questions that we would like to answer is whether BPP equals P? Compared to the theorem above, the trick here is to use pseudorandom generators $g = (g_n)$ with a huge stretch, namely, $g_n : \{0, 1\}^n \rightarrow \{0, 1\}^{m(n)}$ with $m(n) = t(c^n)$ if the Turing machine that we want to derandomize is t time bounded. If we now take c^n as our input length, the seed of the pseudorandom generator is logarithmic in the input length. Thus we can try all possible seeds in time polynomial in the input length and take a majority vote.

18.2 Pseudorandom functions

Let $B_{n,m}$ denote the set of all functions $\{0, 1\}^n \rightarrow \{0, 1\}^m$. Let $f_n : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^m$ where $m = m(n)$. For each $s \in \{0, 1\}^m$, let $f_n|_s$ denote the function $\{0, 1\}^n \rightarrow \{0, 1\}^m$ defined by $x \mapsto f_n(s, x)$. In other words, f_n generates a family of 2^m functions. Let $f = (f_n)$.

For each n , we now can generate a function $f_n|_s$ by picking s at random. When does $f_n|_s$ look pseudorandom?

Algorithm 2 Pseudorandom function generator

Input: $s \in \{0, 1\}^m$, $x \in \{0, 1\}^n$

- 1: Let $y = s$
- 2: **for** $i = 1, \dots, n$ **do**
- 3: Let $y_1 y_2 = g(y)$ with $y_1, y_2 \in \{0, 1\}^m$.
- 4: **if** $x_i = 1$ **then**
- 5: $y = y_1$
- 6: **else**
- 7: $y = y_2$
- 8: **end if**
- 9: **end for**
- 10: return y

Definition 18.4. Let $f = (f_n)$ be a family of functions, $f_n : \{0, 1\}^m \times \{0, 1\}^n \rightarrow \{0, 1\}^m$ with $m = m(n)$. f is $(\delta(n), s(n))$ -pseudorandom if for every $s(n)$ -time bounded oracle PTM B ,

$$\left| \Pr_{h \in B_{n,m}} [B^h(1^n) = 1] - \Pr_{s \in \{0,1\}^m} [B^{f_n|s}(1^n) = 1] \right| \leq \delta(|x|)$$

for all $n \geq n_0$ for some constant n_0 .

In the definition above, B gets a function as an oracle. That means, it can evaluate such a function by writing the argument on the oracle tape and then gets the function value “for free” by entering the query state. This oracle is here considered to be the input of B . B does not get a “real” input, just a string of 1s to give B enough time. We cannot give B the function as an input, since then the input size would be too large.

Assume we have a $t(\ell)$ -time computable family $g = (g_\ell)$ with $g_\ell : \{0, 1\}^\ell \rightarrow \{0, 1\}^{2^\ell}$ that is $(\delta(\ell), r(\ell))$ -pseudorandom. With g , we build a pseudorandom function generator as follows: Given $s \in \{0, 1\}^m$, construct a complete binary tree T_s of height n as follows: The root is labeled with s . If any node is labeled with $y \in \{0, 1\}^m$, we compute $g_m(y)$ and label the left child with the first half of $g_m(y)$ and the right child with the second half of $g_m(y)$. Any $x \in \{0, 1\}^n$ defines a path from the root to some leaf in T_s by identifying “1” with “left” and “0” with “right”. The label at the leaf at the end of this path is the value $f_n|_s(x)$. Algorithm 2 shows how to compute $f_n|_s(x)$.

Theorem 18.5. 1. Given $(s, x) \in \{0, 1\}^m \times \{0, 1\}^n$, $f_n|_s(x)$ can be computed in time $O(nt(n))$.

2. $f = (f_n)$ is $(\delta(n) \frac{r(n)}{t(n)}, \frac{r(n)}{nt(n)})$ -pseudorandom.

Proof. For the first part, note that we have to evaluate g_m n times. Thus f can be evaluated in time $O(nt(n))$, where t is the time needed to evaluate g .

For the second part, we use a *hybrid argument*. We construct a sequence of complete binary trees T_0, \dots, T_n with depth n . We view these trees as layered graphs with $n + 1$ layers. The root is assumed to be the 0th layer. In T_0 , all nodes are labeled with values chosen uniformly at random from $\{0, 1\}^n$. For an element $p \in \{0, 1\}^{n'}$ where $n' \leq n$, $T_0(p)$ is the label that one reaches when one takes the path from the root specified by p . (1 is left and 0 is right.) The mapping $x \mapsto T_0(x)$ with $x \in \{0, 1\}^n$ is a random function, since the values at the leaves are randomly chosen and to choose a random function it suffices to choose its values at random. The tree T_i for $i > 1$ are obtained as follows: The nodes with depth at most $n - i$ are labeled with values uniformly at random. For nodes with greater depth, we do the following. If a node is labeled with y , say, and the children are not labeled so far, then we compute $g_m(y) = y_0y_1$ and label the right children with y_0 and the left children with y_1 . In this way, T_n is the tree that corresponds to the computation of the pseudorandom function f .

Assume that $f = (f_n)$ is not a pseudorandom function, i.e., it can be distinguished from a random function with probability at least $\delta'(n) = \delta(n)nt(n)$. By assumption, we know that

$$|\Pr[B^{T_0}(1^n) = 1] - \Pr[B^{T_n}(1^n) = 1]| \geq \delta'(n).$$

(Here we also use the term T_i to not only denote the tree but also the function generated by the above process.)

Now by writing the difference of the two probabilities as a telescoping sum, there is an i_0 such that

$$|\Pr[B^{T_{i_0}}(1^n) = 1] - \Pr[B^{T_{i_0+1}}(1^n) = 1]| \geq \delta'(n)/n. \quad (18.1)$$

The tree T_{i_0} and T_{i_0+1} differ exactly in the $(n - i_0 - 1)$ th layer: T_{i_0} has random values there whereas T_{i_0+1} has values generated by g_m from the values of the parent nodes. Out of B , we now try to build a distinguisher for g_m . There are two problems:

- We have to get rid of the oracle.
- We cannot generate the whole tree, since it is too large.

To get rid of the oracle, we simulate it by another Turing machine. Since this Turing machine cannot generate the whole tree, it will generate the values on the fly, remembering all the values generated so far to stay consistent. Assume that B is $s(n)$ -time bounded. Since B is $s(n)$ -time bounded, it can query the oracle at most $s(n)$ times.

Algorithm 3 shows the simulation of the oracle. This algorithm has access to a list U of $2s(n)$ random strings from $\{0, 1\}^m$. These values U will be used as the labels of the $(i_0 + 1)$ th layer. If the values U are uniformly at random, then the simulator, call it S , will return $T_{i_0}(x)$. If the values of U are pairs of the form $y_{j,0}, y_{j,1}$ where $y_{j,0}y_{j,1} = g_m(x_j)$ for some random strings $x_j \in \{0, 1\}^m$, then S will compute $T_{i_0+1}(x)$. S looks whether a string with the same prefix of length i_0 as x has already been queried. If no, then it takes the next pair of values from U and labels the two children of the node specified by the prefix of x in the $(i_0 + 1)$ th layer of the tree with the two values. Then it goes on by computing the value by applying g_m as described above.

Now we build a distinguisher D as follows: As an input, we get a list U of $s(n)$ values. D simulates B step by step, whenever B queries the oracle, D uses the simulator S . By construction,

$$\begin{aligned}\Pr[B^{T_{i_0}}(1^n) = 1] &= \Pr[D(u_1, u_2, \dots, u_{2s(n)-1}, u_{2s(n)}) = 1] \\ \Pr[B^{T_{i_0+1}}(1^n) = 1] &= \Pr[D(y_{1,0}, y_{1,1}, \dots, y_{s(n),0}, y_{s(n),1}) = 1]\end{aligned}$$

In the first line, the u_j are chosen uniformly at random, in the second line $y_{j,0}y_{j,1} = g_m(x_j)$ and the x_j are drawn uniformly at random. The running time of D is $O(s(n) \cdot n \cdot t(n))$ where $t(n)$ is the time needed to compute g . Thus we can distinguish a list of random values from a list of pairs generated by a pseudorandom generator. The next exercise shows that we then also can break the pseudorandom generator. The distinguishing probability is $\frac{\delta'(n)}{ns(n)}$.

If $s(n) = r(n)/(n \cdot t(n))$, then D has running time $r(n)$. If $\delta'(n) = \delta(n) \cdot n \cdot s(n) = \delta(n) \cdot r(n)/t(n)$, then D distinguishes g with probability $\geq \delta(n)$. This is a contradiction to the assumption that g is $(\delta(n), r(n))$ -pseudorandom. ■

Exercise 18.2. Use a hybrid argument to finish the proof above.

(Sketch: Look at

$$p_i := \Pr[D(u_1, u_2, \dots, u_{2i-1}, u_{2i}, y_{i+1,0}, y_{i+1,1}, \dots, y_{s(n),0}, y_{s(n),1}) = 1].$$

There is an i_0 such that $|p_{i_0} - p_{i_0-1}| \geq \delta(n)/(ns(n))$. We can break the pseudorandom generator by running D using the internal random bits as u_1, \dots, u_{2i_0} , then we read the input, and then we use the internal random bits and g to generate $y_{i_0+2,0}, \dots, y_{s(n),1}$.)

18.3 Natural proofs

Consider a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Let p be some polynomial. Every proof that f is not computable by circuits of size $p(n)$ exhibits a property P

Algorithm 3 Oracle simulator**Input:** $x \in \{0, 1\}^n$ **Output:** $T_{i_0}(x)$ or $T_{i_0+1}(x)$

- 1: **if** x is the first string with prefix $x_1 \dots x_{i_0}$ that is queried **then**
- 2: Take the next pair y_0, y_1 of unused values from U
- 3: Store $(x_1 \dots x_{i_0} 0, y_0)$ and $(x_1 \dots x_{i_0} 1, y_1)$
- 4: **end if**
- 5: Retrieve the stored value $(x_1 \dots x_{i_0} x_{i_0+1}, y)$. {There is always such a value.}
- 6: Use the rest $x_{i_0+2} \dots x_n$ of x to compute the result z by applying g_m to y and then taking the first or the second part of $g_m(y)$ and so on as specified by the bits of $x_{i_0+2} \dots x_n$
- 7: return z

(on the set of all Boolean functions) such that

$$\begin{aligned} P(f) &= 1 \\ P(g) &= 0 \quad \text{for all } g \text{ computable by circuits of size } p(n). \end{aligned}$$

Such a predicate P is called $p(n)$ -useful.

P is called *natural* if it possesses the following two properties:

Constructiveness: There is a polynomial time algorithm A_P that given the truth table of a function g , computes $P(g)$.

Since the size of the truth table of $g : \{0, 1\}^n \rightarrow \{0, 1\}$ is 2^n , the running time of A_P is $2^{O(n)}$ when measured in n .

Largeness: $\Pr_{g \in B_{n,1}}[P(g) = 1] \geq \frac{1}{n}$.

Example 18.6. 1. Consider the predicate P_1 : $P_1(f) = 1$ if f is not computable by a circuit of size $O(n^{\log n})$. It is not clear that the predicate satisfies constructiveness since to check this, we do not know any other way than running through all circuits of size $O(n^{\log n})$, which takes time $2^{O(n^{\log n})}$.

2. Let g be some fixed function. The predicate $P_2(f) = 1$ if $f = g$ does not satisfy largeness.

Theorem 18.7 (Razborov & Rudich). *If for some $\epsilon > 0$, there is a $(\frac{1}{\text{poly}(n)}, 2^{n^\epsilon})$ -pseudorandom generator with stretch $2n$, then there is no p -useful predicate for some polynomial p .*

(Note that if there is no p -useful predicate, then there is no p' -useful predicate for any polynomial p' of larger degree. In other words, natural proofs can only show superpolynomial bounds, if there are no pseudorandom generators.)

Proof. If there is a $(\frac{1}{\text{poly}(n)}, 2^{n^\epsilon})$ -pseudorandom generator, then there is a $(\frac{1}{\text{poly}(n)}, 2^{n^{\epsilon'}})$ -pseudorandom family (f_n) of functions by the preceding section for some $\epsilon' > 0$. The projection to (f_n) to the first output bit is also pseudorandom with the same parameters.

We can now use P to build a distinguisher for pseudorandom functions. Given a candidate h (which is either truly random or $f_n|_s$), we set $\ell = n^{\epsilon'/2}$ and $g(x) = h(x0^{n-\ell})$. g is a function $\{0, 1\}^\ell \rightarrow \{0, 1\}$ and we can construct its truth table in time $2^{O(\ell)}$. Then we run the algorithm that computes P on this truth table.

If h is truly random so is g and $P(g)$ is 1 with probability $1/\ell$. If $h = f_n|_s$, then given s , g can be computed by a Turing machine with polynomial running time and therefore, g is computable by polynomial size circuits (with s hard-wired into the circuit). Therefore, $P(g) = 0$.

The distinguisher distinguishes f_n from a random function with probability $1/n^{\epsilon'/2}$. Its running time is $2^{O(\ell)} = 2^{n^{\epsilon'}}$. This is a contradiction.

■

19 Hardness based on derandomization

Before we start, we introduce the following notion: We say that the permanent is in NP, if the following language

$$\{(A, v) \mid A \text{ is a } \{0, 1\}\text{-matrix and } \text{perm}(A) = v\}$$

is in NP. We abuse of notation, we call this language perm again.

19.1 Testing arithmetic circuits for the permanent

The aim of this section is given some polynomial f , to construct a circuit C such that C computes the zero polynomial if and only if f is the permanent.

19.1.1 Division-free circuits over \mathbb{Z}

Let p_n be a polynomial in n^2 indeterminates $X_{i,j}$, $1 \leq i, j \leq n$. Assume that p_n computes the permanent of $n \times n$ -matrices, i.e., $p_n(X) = \text{perm}(X)$, where X denotes the matrix with entries $X_{i,j}$.

We can use p_n to compute the permanent of any size $\leq n$: If A is an $i \times i$ -matrix, then we place A into the lower right corner, place ones to the remaining entries on the diagonal, and fill the rest of the entries with zeros. Let p_i be the restriction obtained from p_n . By the definition of the permanent, we then have

$$p_1(X^{(1)}) = X_{n,n} \tag{19.1}$$

$$p_i(X^{(i)}) = \sum_{j=1}^i X_{n-i+1, n-i+j} p_{i-1}(X_j^{(i)}) \tag{19.2}$$

where $X^{(i)}$ is the $i \times i$ -matrix in the lower right corner of X and $X_j^{(i)}$ is the j th minor of $X^{(i)}$ along the first row, i.e., the matrix obtained from $X^{(i)}$ by deleting the first row and the j th column.

On the other hand, any sequence of polynomials p_1, \dots, p_n fulfilling (19.1) and (19.2) necessarily computes perm.

Exercise 19.1. *Prove this last claim.*

Lemma 19.1. *The language*

$$\text{ACP} := \{ \langle C, n \rangle \mid C \text{ is an arithmetic circuit for} \\ \text{perm of } n \times n\text{-matrices over } \mathbb{Z} \}$$

is polynomial-time many-one reducible to ACIT.

Proof. Assume that C computes a polynomial p_n . Let p_i be the restriction of p_n such that p_i computes the permanent of $i \times i$ -matrices provided that $p_n(X) = \text{perm}(X)$.

To check whether p_n computes indeed the permanent, it suffices to check whether p_1, \dots, p_n fulfill (19.1) and (19.2). In other words, we have to check whether

$$h_1(X) = p_1(X^{(1)}) - X_{n,n} \tag{19.3}$$

$$h_i(X) = p_i(X^{(i)}) - \sum_{j=1}^i X_{n-i+1, n-i+j} p_{i-1}(X_j^{(i)}), \quad 2 \leq i \leq n, \tag{19.4}$$

are identically zero. To test whether h_1, \dots, h_n are identically zero, we can equivalently test whether

$$h(X, Y) = h_1(X) + h_2(X)Y + \dots + h_n(X)Y^{n-1}$$

is identically zero, where Y is a new variable.

By construction, C computes $\text{perm}(X)$ iff $h(X, Y) = 0$. Since every h_i is computable by a circuit of size polynomial in the size of C , h is also computable by such a circuit. This circuit can be constructed from C in polynomial time. ■

Corollary 19.2. *Suppose that ACIT over \mathbb{Z} is in NP. If perm over \mathbb{Z} is computable by division-free arithmetic circuits of polynomial size over \mathbb{Z} , then $\text{perm} \in \text{NP}$.*

Proof. If perm is computable by arithmetic circuits of polynomial size, then we can nondeterministically guess such a circuit C that computes the permanent of $n \times n$ -matrices in time polynomial in n . Since ACIT is in NP, so is ACP by Lemma 19.1. Therefore, we can verify our guess for C nondeterministically in polynomial time. Once we have found C , we evaluate it deterministically at the given $\{0, 1\}$ -matrix A in polynomial time, by doing all operations modulo $2^{n \log n} + 1$. Note that $2^{n \log n}$ has only polynomially many bits and that the permanent of a $\{0, 1\}$ -matrix cannot exceed $2^{n \log n}$, since it has at most $n!$ terms. Finally, we simply check whether the computed result equals v , the second part of the input. ■

19.2 Lower bound

We now come to the proof of the lower bound based on derandomization of ACIT.

Definition 19.3. *We say that $\text{NEXP} \cap \text{co-NEXP}$ is computable by polynomial-size circuit if the following two conditions hold:*

1. $\text{NEXP} \cap \text{co-NEXP} \subseteq \text{P/poly}$ and
2. perm over \mathbb{Z} is computable by polynomial-size arithmetic circuits (without divisions).

Exercise 19.2. *Computing perm of a $\{0,1\}$ -matrix is clearly possible in deterministic exponential time. Why does the first condition of Definition 19.3 not imply the second one?*

To prove our main theorem, we need the following results from complexity theory.

Theorem 19.4 (Toda). $\text{PH} \subseteq \text{P}^{\text{perm}}$.

You saw a proof of this theorem in the complexity theory lecture.

Exercise 19.3. *If $\text{perm} \in \text{NP}$, then $\text{P}^{\text{perm}} \subseteq \text{NP}$.*

Theorem 19.5 (Meyer). *If $\text{EXP} \subseteq \text{P/poly}$, then $\text{EXP} = \Sigma_2^{\text{P}} \cap \Pi_2^{\text{P}}$.*

The proof of this theorem was an exercise in the complexity theory lecture.

Exercise 19.4. *Conclude that $\text{EXP} \subseteq \text{P/poly}$ implies $\text{P} \neq \text{NP}$.*

Theorem 19.6 (Impagliazzo, Kabanets & Wigderson). *If $\text{NEXP} \subseteq \text{P/poly}$ then $\text{NEXP} = \text{EXP}$.*

We do not give a proof of this result here.

Corollary 19.7. *If $\text{NEXP} \subseteq \text{P/poly}$, then perm over \mathbb{Z} is NEXP-hard.*

Proof. If $\text{NEXP} \subseteq \text{P/poly}$, then $\text{NEXP} = \text{EXP} = \text{PH}$ by Theorems 19.5 and 19.6. Since perm is PH-hard by Theorem 19.4, it is also NEXP-hard. ■

Finally, we come to the main result of this chapter.

Theorem 19.8 (Kabanets & Impagliazzo). *If ACIT over \mathbb{Z} is in NP, then $\text{NEXP} \cap \text{co-NEXP}$ is not computable by polynomial-size circuits (in the sense of Definition 19.3).*

Proof. If perm is not computable by polynomial-size arithmetic circuits, then the proof is finished by definition. It remains the case that perm is computable by polynomial-size arithmetic circuits.

perm is PH-hard by Theorem 19.4. Since ACIT \in NP, we also know that perm \in NP by Corollary 19.2. By Exercise 19.3, we know that the polynomial hierarchy collapses to NP, i.e., PH = NP = co-NP. A simply padding argument (see Exercise 19.5) implies that NEXP = co-NEXP.

If NEXP $\not\subseteq$ P/poly, then we are done, because NEXP = co-NEXP = NEXP \cap co-NEXP. Therefore, assume that NEXP \subseteq P/poly. By Corollary 19.7, perm over \mathbb{Z} is NEXP-hard. On the other hand, perm \in NP. Thus co-NEXP = NEXP = NP.

co-NEXP = NP is easily disproved by the following diagonalization argument. We can even refute the weaker statement co-NEXP \subseteq NTime(2^n): We define a co-NEXP-machine M as follows: On input x of length n , M simulates the x th nondeterministic Turing machine M_x for 2^n steps. If M_x accepts, then M rejects. Otherwise M accepts. Since M is a co-NEXP-machine, it is easy for M to flip the outcome of the computation of M_x .

■

Exercise 19.5. Show that NP = co-NP implies that NEXP = co-NEXP.

Bibliography

- [ACG⁺99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, 1999.
- [AG00] Carme Alvarez and Raymond Greenlaw. A compendium of problems complete for symmetric logarithmic space. *Comput. Complexity*, 9:73–95, 2000.
- [ALM⁺98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
- [BDCGL92] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992.
- [Big93] Norman Biggs. *Algebraic graph theory*. Cambridge University Press, second edition, 1993.
- [BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correction with application to numerical problems. *J. Comput. Syst. Sci.*, 47:549–595, 1993.
- [BT06] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1):1–106, 2006.
- [Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3), 2007.
- [GLST98] Venkatesan Guruswami, Daniel Lewin, Madhu Sudan, and Luca Trevisan. A tight characterization of NP with 3-query PCPs. In *Proc. 39th Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS)*, pages 8–17, 1998.
- [Gur91] Yuri Gurevich. Average case completeness. *J. Comput. Syst. Sci.*, 42(3):346–398, 1991.
- [Hås99] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

- [HLW06] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, pages 439–561, 2006.
- [Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard functions. In *Proc. 36th Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS)*, pages 538–545, 1995.
- [KS03] Adam R. Klivans and Rocco A. Servedio. Boosting and hard-core set construction. *Machine Learning*, 51(3):217–238, 2003.
- [Lev86] Leonid A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, 1986.
- [NTS95] Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995.
- [O'D04] Ryan O'Donnell. Hardness amplification within NP. *J. Comput. Syst. Sci.*, 69(1):68–94, 2004.
- [Rei08] Omer Reingold. Undirected connectivity is in log-space. *J. ACM*, 55(4), 2008.
- [RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product and new constant degree expanders and extractors. *Annals of Mathematics*, 155(1):157–187, 2002.
- [Vaz01] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.
- [vN28] John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.
- [VV86] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theoret. Comput. Sci.*, 47(1):85–93, 1986.
- [Wan97] Jie Wang. Average-case intractable NP problems. In Ding-Zhu Du and Ker-I Ko, editors, *Advances in Languages, Algorithms, and Complexity*, pages 313–378. Kluwer, 1997.
- [Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS)*, pages 80–91, 1982.