Probabilistically checkable proofs and expander graphs

Markus Bläser
Universität des Saarlandes

Draft—June 15, 2012 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* $(\mathrm{I}_P, \mathrm{S}_P, \mathrm{m}_P, \mathrm{goal}_P)$ *where*

1. $\mathrm{I}_P \subseteq \{0,1\}^*$ *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 $\mathrm{S}_P(x)$ *of* x, *which is a subset of* $\{0,1\}^*$.[1]
3. $\mathrm{m}_P : \{(x,y) \mid x \in \mathrm{I}_P \text{ and } y \in \mathrm{S}_P(x)\} \to \mathbb{N}^+$ *is the* objective function *or* measure function. $\mathrm{m}_P(x,y)$ *is the objective value of the feasible solution* y *(with respect to* x*).*
4. $\mathrm{goal}_P \in \{\min, \max\}$ *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 $\mathrm{S}^*_P(x)$ the set of all optimal solutions, that is, the set of all $y \in \mathrm{S}_P(x)$ such that

$$\mathrm{m}_P(x,y) = \mathrm{goal}\{\mathrm{m}_P(x,z) \mid z \in \mathrm{S}_P(x)\}.$$

(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

[1] Some authors also assume that for all $x \in \mathrm{I}_P$, $\mathrm{S}_P(x) \neq \emptyset$. 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 $\mathrm{S}_P(x) \neq \emptyset$.) The objective value of any $y \in \mathrm{S}^*(x)$ is denoted by $\mathrm{OPT}_P(x)$.[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 $y \in \mathrm{S}^*(x)$ (*construction problem*).
2. compute $\mathrm{OPT}(x)$ (*evaluation problem*)
3. given an additional bound B, decide whether $\mathrm{OPT}(x) \geq B$ (if goal = max) or whether $\mathrm{OPT}(x) \leq B$ (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 $F : \mathrm{I} \to \mathcal{P}(\{0,1\}^*)$ mapping each x to its set of optimal solutions $\mathrm{S}^*(x)$. However $\mathrm{S}^*(x)$ 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 $f : \mathrm{I} \to \{0,1\}^*$ that maps every x to some $y \in \mathrm{S}^*(x)$. 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 $x \mapsto \mathrm{OPT}(x)$. We denote this function by P_{eval}.

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

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

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* $P = (\mathrm{I}, \mathrm{S}, \mathrm{m}, \mathrm{goal})$ *such that*

[2] The name $\mathrm{m}^*_P(x)$ would be more consequent, but $\mathrm{OPT}_P(X)$ is so intuitive and convenient.

[3] Note 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 nth Turing machine halts on the empty word and to a star otherwise is certainly not computable.

1. $\mathrm{I} \in \mathsf{P}$, *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 $x \in \mathrm{I}$ and $y \in \mathrm{S}(x)$, $|y| \leq p(|x|)$, and for all y with $|y| \leq p(|x|)$, we can decide $y \in \mathrm{S}(x)$ in time polynomial in $|x|$,*
3. m *is computable in polynomial time.*

Definition 1.3. PO *is the class of all optimization problems $P \in \mathsf{NPO}$ 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 $P \in \mathsf{NPO}$, $P_{\mathrm{dec}} \in \mathsf{NP}$.*

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 $x \in \mathrm{I}$ and bound B

1. M guesses a string y with $|y| \leq p(|x|)$.
2. M deterministically tests whether $y \in \mathrm{S}(x)$.
   If not, M rejects.
3. If y is, then M computes $\mathrm{m}(x, y)$ and tests whether $\mathrm{m}(x, y) \leq B$ (minimization problem) or $\mathrm{m}(x, y) \geq B$ (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 $w : E \to \mathbb{N}^+$ assigning each edge a positive weight, find a Hamiltonian tour of minimum weight. If in addition w fulfills the triangle inequality, i.e.,*

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

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 TSP $\in$ 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 $2^{p(n)} \cdot \mathrm{OPT}$ for some polynomial p, where n is the number of nodes of the given graph. Then* P = 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 $P \in$ NPO.

Theorem 1.6. *Let $P \in$ NPO. Then*

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

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

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

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

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

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

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

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

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

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

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

1.5 NP-hard optimization problems

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

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

Exercise 1.2. *Prove Theorem 1.9.*

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

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

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* $\mathsf{P} \neq \mathsf{NP}$*, then* $\mathsf{PO} \neq \mathsf{NPO}$

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 $P_{\text{dec}} \in \mathsf{P}$ 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 $P = (\mathrm{I}, \mathrm{S}, \mathrm{m}, \mathrm{goal})$ if*

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

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, $x \in \mathrm{I}$, and $y \in \mathrm{S}(x)$. The* performance ratio *of y with respect to x is defined as*

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

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

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

The definition of $\mathrm{PR}(x, y)$ 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 $\mathrm{m}(x, y)/\mathrm{OPT}(x)$ 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 $\mathbb{N} \to \mathbb{Q}$. An optimization problem $P \in \mathsf{NPO}$ is contained in the class F-APX if there is an $f \in F$ such that there exists an f-approximation algorithm for P.*

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

2. $\mathsf{APX} := O(1)\text{-}\mathsf{APX}$.

(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

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

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 $\mathrm{m}(x, y) \geq \frac{1}{1+\epsilon}\,\mathrm{OPT}(x)$. 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

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

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

Problem 2.6. Knapsack *is the following problem:*

Instances:	*rational numbers $w_1, \ldots, w_n$ (weights), $p_1, \ldots, p_n$ (profits), and B (capacity bound) such that $w_\nu \leq B$ for all ν.*
Solutions:	*$I \subseteq \{1, \ldots, n\}$ 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 $\gcd(x_\nu, y_\nu) = 1$. 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 $P := \max_{1 \le \nu \le n} p_\nu$. (Compute by dynamic programming sets of indices $I(i,p)$ such that*

- *$\nu \le i$ 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 $1 \le \nu \le n$.*
- *Find an optimum solution for the instance $w_1, \ldots, w_n$, $\hat{p}_1, \ldots, \hat{p}_n$, 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* $\mathsf{P} = \mathsf{NP}$.

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

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

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* $P = (\mathrm{I}_P, \mathrm{S}_P, \mathrm{m}_P, \mathrm{goal})$ *be an optimization problem and* $a < b$. $\mathrm{gap}(a,b)$-P *is the promise problem* (L, U) *where*

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

and

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

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 two be functions $\mathbb{N} \to \mathbb{N}$ that depend on $|x|$.

Theorem 2.9. *If* $\mathrm{gap}(a,b)$-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* $\mathsf{P} = \mathsf{NP}$.

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

Input: formula ϕ in CNF

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

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

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

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

$$\mathrm{m}_P(x,y) \geq \mathrm{OPT}_P(x) \geq b(|x|).$$

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

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 $\texttt{TSP} \in \mathsf{NPO}$, we get the following result.

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

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

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

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

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

1. *for all $x \in \mathrm{I}_P$ and $\beta > 1$, $f(x,\beta) \in \mathrm{I}_{P'}$,*

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

3. *for all $x \in \mathrm{I}_P$, $y \in \mathrm{S}_{P'}(f(x,\beta))$, and $\beta > 1$,*

$$g(x,y,\beta) \in \mathrm{S}_P(x),$$

4. *f and g are deterministically polynomial time computable for fixed $\beta > 1$,*

5. *for all $x \in \mathrm{I}_P$ and all $y \in \mathrm{S}_{P'}(f(x,\beta))$, 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 $P \leq_{\mathrm{AP}} P'$ and $P' \in \mathsf{APX}$, then $P \in \mathsf{APX}$.*

Proof. Let (f,g,α) be an AP-reduction from P to P' and let A' be a β-approximation algorithm for P'. Given $x \in \mathrm{I}_P$, $A(x) := g(x, A'(f(x,\beta)),\beta)$ 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 $P \leq_{\mathrm{AP}} P'$. Show that if $P' \in \mathsf{PTAS}$, 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 $\mathsf{ZPP} = \mathsf{NP}$ (which is almost as unlikely as $\mathsf{P} = \mathsf{NP}$).

[2] The 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* $\mathsf{C} \subseteq \mathsf{NPO}$*. A problem* P *is* C-hard *(under AP-reductions) if for all* $P' \in \mathsf{C}$*,* $P' \leq_{\mathrm{AP}} P$*.* P *is* C-complete *if it is in* C *and* C*-hard.*

Lemma 2.19. $\leq_{\mathrm{AP}}$ *is transitive.*

Proof. Let $P \leq_{\mathrm{AP}} P'$ and $P' \leq_{\mathrm{AP}} P''$. 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

$$F(x, \beta) = f'(f(x, \gamma), \beta),$$
$$G(x, y, \beta) = g(x, g'(f(x, \gamma), y, \gamma), \beta).$$

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 $x \in \mathrm{I}_P$ and $y \in \mathrm{S}_{P''}(F(x, \beta))$. We know that $g'(f(x, \gamma), y, \beta) \in \mathrm{S}_{P'}(f(x, \gamma))$, since (f', g', α') is an AP-reduction. But then also $g(x, g'(f(x, \gamma), y, \gamma), \beta) \in \mathrm{S}_P(x)$, 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* $\mathsf{C} \subseteq \mathsf{NPO}$*. If* $P \leq_{\mathrm{AP}} P'$ *and* P *is* C*-hard, then* P' *is also* C*-hard.*

Proof. Let $Q \in \mathsf{C}$ be arbitrary. Since P is C-hard, $Q \leq_{\mathrm{AP}} P$. Since $\leq_{\mathrm{AP}}$ is transitive, $Q \leq_{\mathrm{AP}} P'$. ∎

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 $x_1, \ldots, x_n$ having nonnegative weights $w_1, \ldots, w_n$*
Solutions:	*Boolean assignments $\alpha : \{x_1, \ldots, x_n\} \to \{0, 1\}$ that satisfy ϕ*
Measure:	$\max\{1, \sum_{i=0}^{n} w_i \alpha(x_i)\}$
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 $\mathrm{m}(x, y)$. 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 $\mathrm{m}(x, y)$.)*

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

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

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

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

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

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 $\{0,1\}^{\log(|\pi|)} \to \{0,1\}$ 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 of length n.

Definition 3.1. *Let $r, q : \mathbb{N} \to \mathbb{N}$. A language L belongs to the class $\mathsf{PCP}[r,q]$ 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 verifiers 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 $\mathbb{N} \to \mathbb{N}$. We generalize the notion of $\mathsf{PCP}[r, q]$ in the obvious way:

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

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

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

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)**.** $\mathsf{NP} = \mathsf{PCP}[\mathrm{O}(\log n), \mathrm{O}(1)]$.

What does this mean? By allowing a little randomness—note that $\mathrm{O}(\log n)$ are barely sufficient to choose $\mathrm{O}(1)$ 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* $\mathsf{PCP}[\mathrm{O}(\log n), \mathrm{O}(1)] \subseteq \mathsf{NP}$. *(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. $\mathsf{NP} = \mathsf{PCP}[\mathrm{O}(\log n), \mathrm{O}(1)]$.

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

Proof. "$\Longrightarrow$": Let L be any NP-complete language. By assumption, there is an $(r(n), q)$-restricted nonadaptive polynomial time probabilistic verifier V with $r(n) = \mathrm{O}(\log n)$ and $q = \mathrm{O}(1)$. 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 ith bit is 1; if it is set to zero, then this bit is 0. Since we can restrict ourselves to proofs of length $\leq q \cdot 2^{r(n)} = \mathrm{poly}(n)$, the number of these variables is polynomial.

For a random string y, let $i(y,1), \ldots, i(y,q)$ 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 $(b_1, \ldots, b_q) \in \{0,1\}^q$ 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 $(b_1, \ldots, b_q) \in A_y$, we construct a clause of q literals, that is true iff the variables $v_{i(y,1)}, \ldots, v_{i(y,q)}$ do not take the value $b_1, \ldots, b_q$, i.e, $v_{i(y,1)}^{1-b_1} \vee \cdots \vee v_{i(y,q)}^{1-b_q}$. (Here, for a Boolean variable v, $v^1 = v$ and $v^0 = \bar{v}$.) The formula ϕ has $\leq |A_y| 2^{r(n)} \leq 2^{q+r(n)} = \mathrm{poly}(n)$ 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 $c_1, \ldots, c_{q-2}$ 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 $c_1, \ldots, c_{q-2}$ and conversely, the restriction of any assignment that satisfies $c_1, \ldots, c_{q-2}$ 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

[1] Instead 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.

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

$$\begin{aligned} x \in L &\Longrightarrow f(x) \text{ is a satisfiable formula in 3-CNF,} \\ x \notin L &\Longrightarrow f(x) \text{ is a formula in 3-CNF such that every assignment} \\ &\qquad \text{satisfies at most } (1-\epsilon) \text{ of the clauses.} \end{aligned}$$

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 $\mathrm{O}(\log m) = \mathrm{O}(\log |x|)$ random bits, and it only queries $\mathrm{O}(1)$ bits of the proof. ∎

Exercise 3.2. *Let c be a clause of length q. Construct clauses $c_1, \ldots, c_{q-2}$ 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 $c_1, \ldots, c_{q-2}$ and conversely, the restriction of any assignment that satisfies $c_1, \ldots, c_{q-2}$ 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* $\mathsf{PCP}[\mathrm{O}(\log n), 2] = \mathsf{P}$.

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 $P = (\text{I}_P, \text{S}_P, \text{m}_P, \max)$ be a maximization problem in* APX*. Then $P \leq_{\text{AP}}$ 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

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

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

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 $\mathrm{OPT}_P(x) \geq i$.
   (These formulas $\phi_{x,i}$ can be uniformly constructed in polynomial time, cf. the proof of Cook's theorem.)

2. Let $\psi_{x,\kappa} = f_{\mathrm{PCP}}(\phi_{x,\mu(x)r^\kappa})$, $1 \leq \kappa \leq k$.
   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 $g_{\mathrm{PCP}}(\phi_{x,\mu(x)r^\kappa}, a)$ 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 $\mathrm{m}_P(x, y) \geq \mu(x)r^{\kappa_0}$.
   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

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

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

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

[1] Here 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 $g_{\mathrm{PCP}}(a)$ 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 $\mathrm{OPT}_P(x) \geq \mu(x)r^\kappa$. By the definition of κ_0,

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

This means that $\mathrm{m}_P(x, y) \geq \mu(x)r^{\kappa_0}$. 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 $P \in \mathsf{APX}$, there is a maximization problem $P' \in \mathsf{APX}$ such that $P \leq_{\mathrm{AP}} P'$.*

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

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

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

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

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

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

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

We distinguish two cases: The first one is $\mathrm{m}_P(x,y) \leq \mu(x)$. In this case,

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

This completes the first case.

For the second case, note that

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

Thus, $P \leq_{\mathrm{AP}} P'$. ■

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 $\leq_{\mathrm{AP}}$ Clique*. (In particular,* Clique *does not have a PTAS, unless* P = NP*.)*

The kth cartesian product of a graph $G = (V, E)$ is a graph with nodes V^k and there is an edge between $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_k)$ 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

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

and identify $\{0,1\}$ with 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 $\alpha_1, \ldots, \alpha_n$ such that

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

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

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

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

1. *Two Boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ 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 $\{0,1\}^n \to \{0,1\}$.

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 $e_1, \ldots, e_n$. 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 $f : \{0,1\}^n \to \{0,1\}$ 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 $\{0,1\}^n \to \{0,1\}$ 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 a functions $\{0,1\}^n \to \mathbb{R}$. 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 $\{-1,1\}^n \to \{-1,1\}$ 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 $\ell : \{0,1\}^n \to \{0,1\}$ be a linear function corresponding to some set $S \subseteq \{1, \ldots, n\}$. As a function $\{-1,1\}^n \to \{-1,1\}$, 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 $f, g : \{-1,1\}^n \to \mathbb{R}$. The* correlation *of f and g is*

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

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

Exercise 4.2. *Let $f, g : \{-1,1\}^n \to \{-1,1\}$. 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 $f : \{-1,1\}^n \to \mathbb{R}$ and $S \subseteq \{1,\ldots,n\}$. 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 $S, T \subseteq \{1,\ldots,n\}$. 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

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

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 $f : \{-1,1\}^n \to \{-1,1\}$ is 2^n, we get the following result.

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

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

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

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

$$\langle f, \chi_T \rangle = \langle \sum_S \alpha_S \chi_S, \chi_T \rangle = \sum_S \alpha_S \langle \chi_S, \chi_T \rangle = \alpha_S. \qquad \blacksquare$$

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

Exercise 4.3. *Let $f, g : \{-1,1\}^n \to \mathbb{R}$. 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 $f : \{0,1\}^n \to \{0,1\}$ 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 $f : \{-1,1\}^n \to \{-1,1\}$

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 form linear, then the BLR test accepts with probability $< 1 - \epsilon$.

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

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

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

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

Replacing the function by its Fourier expansion, we get

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

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

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

Since $\mathrm{E}[x_i] = 0$ and $\mathrm{E}[y_j] = 0$, 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),

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

Lemma 4.9. *If*

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

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, $\mathsf{NP} \subseteq \mathsf{PCP}[\mathrm{poly}(n), \mathrm{O}(1)]$. 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* $e : \{0,1\}^n \to \{0,1\}^m$ *is an* error correcting code *with* distance δ*, if*

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

The image $\mathrm{im}(e)$ *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 $x, y \in \{0,1\}^n$, $x^T \cdot y = \sum_\nu^n x_\nu y_\nu$ is the scalar product of x and y (modulo 2). The *Walsh-Hadamard code* is the function $\mathrm{wh} : \{0,1\}^n \to \{0,1\}^{2^n}$ 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 $\{0,1\}^n \to \{0,1\}$. 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 $y = (y_1, \ldots, y_n) = \{0,1\}^n$,

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

Thus, for every $(y_2, \ldots, y_n) \in \{0,1\}^n$, either $x^T \cdot (1, y_2, \ldots, y_n) = 0$ and $x^T \cdot (0, y_2, \ldots, y_n) = 1$ 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 GF(2)). The set of code words is a linear space.[1] Take two code words $a = \mathrm{wh}(x)$ and $b = \mathrm{wh}(y)$. $\Delta(a, b)$ 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 $e : \{0,1\}^n \to \{0,1\}^m$ 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 $b \in \mathrm{im}(e)$ 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 $b \in \mathrm{im}(\mathrm{wh})$ with $\Delta(a, b) < \frac{1}{4}$. Given a and an index $1 \leq i \leq m$, we can compute b_i, the ith 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 $a : \{0,1\}^n \to \{0,1\}$ 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. ∎

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

Note that above, we cannot just return $a(x)$. This would only be correct with constant probabilty 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 $x_1, \ldots, x_n$ 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 algoritm for testing whether some vector X is of the form $x \otimes x$.

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

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

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 satisfiabiliy

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 $C = \mathrm{wh}(x \otimes x)$ 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

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

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. $\mathsf{NP} \subseteq \mathsf{PCP}[\mathrm{poly}(n), \mathrm{O}(1)]$.

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 $\pi \in \{0,1\}^{2^{n^2}+2^n}$. π is interpreted as $C = \mathrm{wh}(X)$ with $X \in \{0,1\}^{n^2}$ and $c = \mathrm{wh}(x)$ 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 $\mathrm{wh}(X)$ and $\mathrm{wh}(x)$.)
3. Test whether x satisfies (A, b). If yes, accept. Otherwise, reject (Use the self-correction property of the Walsh-Hadamard code to query $\mathrm{wh}(X)$ and $\mathrm{wh}(x)$.)

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

Assume $(A, b) \notin$ `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 directed[1] multigraph (V, E) together with a mapping $c : E \to \mathcal{P}(\Sigma)$. An assignment is a mapping $V \to \Sigma$. 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 $\mathrm{UNSAT}_a(G)$. The unsatisfiability value of G is $\mathrm{UNSAT}(G) = \min_a \mathrm{UNSAT}_a(G)$.

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

Instances:	*constraint graphs* $((V, E), \Sigma, c)$
Solutions:	*assignments* $a : V \to \Sigma$
Measure:	$(1 - \mathrm{UNSAT}_a(G)) \cdot \lvert E \rvert$
Goal:	max

Exercise 6.1. *The following two statements are equivalent:*

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

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

Thus, to prove the PCP-Theorem, we can show the NP-hardness of $\mathrm{gap}(1-\epsilon, 1)$-`Max-CGS` instead of $\mathrm{gap}(1-\epsilon, 1)$-`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.

[1] We 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 $f_x : \Sigma_0 \to \Sigma^{|X|}$, $f_y : \Sigma_0 \to \Sigma^{|Y|}$, $g_x : \Sigma^{|X|} \to \Sigma_0$, $g_y : \Sigma^{|Y|} \to \Sigma_0$.*

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 $\mathrm{UNSAT}_A(G) \leq \epsilon$, $(g_x(a'), g_y(b'))$ satisfies G_0.*

Above, $(f_x(a), f_y(b), d) : V \to \Sigma$ 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 $\{0,1\}^{k_0} \times \{0,1\}^{k_0} \to \{0,1\}$.

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 auxilliary 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 $\mathrm{wh}(x)$, 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 $k \in \mathrm{O}(1)$ 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 auxilliary 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 $\Delta(\mathrm{wh}(a), a') \leq \frac{1}{4}$ and $\Delta(\mathrm{wh}(b), b') \leq \frac{1}{4}$ 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 $\mathrm{UNSAT}_A(G) \leq \frac{1}{3k}$. 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 auxilliary variable introduced in the reduction from `CSAT` to `Quad-Eq`.
- Therefore, instead of one string c ($= \mathrm{wh}(x)$), 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 c' 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 probablity $< \frac{1}{2}$ 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 $(v_0, e_1, v_1, e_2, \ldots, e_\ell, v_\ell)$ 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 $\{1, \ldots, |V|\}$. But we will assume that the nodes have some ordering, so that when we need it, we can also index the rows by $1, \ldots, |V|$.

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 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ and there is a orthonormal basis consisting of eigenvectors.

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

1. *$\lambda_1 = d$ and $1_n = (1, \ldots, 1)^T$ 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,

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

and

$$A \cdot 1_n = d \cdot 1_n.$$

Thus d is an eigenvalue and 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 by 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 eigenvector.

Now comes 2. "$\Longrightarrow$": 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, $b = 1_n$ 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 $u \in N(v)$.

"$\Longleftarrow$": If the graph G is not connected, then $A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$. Therefore $(1, \ldots, 1, 0, \ldots, 0)$ and $(0, \ldots, 0, 1, \ldots, 1)$ (with the appropriate number of 1's and 0's) are linearly independent eigenvectors associated with d. ∎

Let $\|.\|$ 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*

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

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

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 $c_1, \ldots, c_n$ 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

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

Since $c_1, \ldots, c_n$ is a orthonormal family,

$$1 = \|b\|^2 = b_2^2 + \cdots + b_n^2.$$

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

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

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 $\lambda_2, \ldots, \lambda_n$.

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

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

[1] We 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 $\beta_0 < \beta_1 < \cdots < \beta_r$ 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

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

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 $|\operatorname{supp}(f)| \leq n/2$, 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 $|\operatorname{supp}(f)| \leq n/2$, 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,

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

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 $W = \operatorname{supp}(f)$. 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

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

By Lemmas 7.10 and 7.11,

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

Squaring this and exploiting the inequality before, we get

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

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 $\lambda = 1, \ldots, \ell$ choose $v_\lambda \in N(v_{\lambda-1})$ uniformly at random.
3. Return $(v_0, \ldots, v_\ell)$.

Let $\mathcal{W}_\ell$ 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 $\{1, \ldots, d\}^\ell$ 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 $\lambda = j-1, \ldots, 0$ choose $v_\lambda \in N(v_{\lambda+1})$ uniformly at random.
3. For $\lambda = j+1, \ldots, \ell$ choose $v_\lambda \in N(v_{\lambda-1})$ uniformly at random.
4. Return $(v_0, \ldots, v_\ell)$.

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

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

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

$$\begin{aligned} y_v &= \frac{1}{d} \cdot |\{e = \{u, v\} \mid e \in F\}| \\ &= \frac{2|F|}{d} \cdot \frac{1}{2|F|} |\{e = \{u, v\} \mid e \in F\}| \\ &\leq \frac{2|F|}{d} \cdot x_\nu. \end{aligned} \tag{8.2}$$

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

$$\left\langle y, \tilde{A}^{j-1} x \right\rangle \tag{8.3}$$

where $\langle ., . \rangle$ 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 $\|x\|$. $x_1 = \frac{1}{n} 1_n$ is an eigenvector of $\tilde{A}$ associated with 1. Let $x^\perp = x - x_1$. $x^\perp$ is orthogonal to x_1, because

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

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

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

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 $(j+1)$the 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 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:

$$
\begin{array}{cl}
G & \\
\downarrow & \text{Preprocessing } (G \text{ becomes an expander}) \\
G_{\text{prep}} & \\
\downarrow & \text{Amplification (UNSAT value gets larger, but also } \Sigma) \\
G_{\text{amp}} & \\
\downarrow & \text{Alphabet reduction } (\Sigma = \{0,1\}^{\kappa_0} \text{ again}) \\
G_{\text{red}} &
\end{array}
$$

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 out 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 $\text{size}(G_1) = \text{O}(\text{size}(G))$.

Next, we show that $\text{UNSAT}(G_1) \leq \text{UNSAT}(G)$. Chose an assignment $\sigma : V \to \Sigma$ with $\text{UNSAT}(G) = \text{UNSAT}_\sigma(G)$ (i.e., an optimal assignment). We define $\sigma_1 : V_1 \to \Sigma$ 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 $\sigma_1 : V_1 \to \Sigma$ be an optimum assignment. We define $\sigma : V \to \Sigma$ 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 $S^v_a = \{u \in S^v \mid \sigma_1(u) = a\}$. We have $S^v = \bigcup_{a \neq \sigma(v)} S^v_a$. Because we took a majority vote, $|S^v_a| \leq \frac{1}{2}|V(Y_v)|$ for all $a \neq \sigma_1(u)$. As Y_v is an expander,

$$|E(S^v_a, \bar{S}^v_a)| \geq h_0 \cdot |S^v_a|,$$

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

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

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,*
- $\lambda(G_2) \leq d + d_0 + 1 - \frac{h_0^2}{2(d+d_0+1)}$,
- $\mathrm{size}(G_2) = \mathrm{O}(\mathrm{size}(G))$,
- $\frac{d}{2+2(d_0+1)} \cdot \mathrm{UNSAT}(G) \leq \mathrm{UNSAT}(G_2) \leq \mathrm{UNSAT}(G)$.

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 are constants $\beta_{prep} > 0$ 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,*
- *$\lambda(G_{prep}) \leq \lambda$,*
- *$\text{size}(G_{prep}) = \text{O}(\text{size}(G))$,*
- *$\beta_{prep} \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_{prep}) \leq \text{UNSAT}(G)$.*

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 $\text{size}(G^t) = \text{O}(\text{size}(G))$. 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 initital 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 $\hat{\sigma} : V \to \Sigma^{d^{t/2}}$ 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 $\hat{G}^t$.

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. $W = (v_0, e_1, v_1, \ldots, v_t) \in \mathcal{W}_t$ *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,

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

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 $\mathrm{E}[N(W)] \geq \Omega(\sqrt{t})\frac{|E|}{|F|}$ and in Lemma 9.7, $\mathrm{E}[N(W)^2] \leq \mathrm{O}(\sqrt{t})\frac{|E|}{|F|}$. Now the claim follows from $\Pr[Z > 0] \geq \frac{\mathrm{E}[Z]^2}{\mathrm{E}[Z^2]}$ 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. $\mathrm{E}[N(W)^2] \leq \mathrm{O}(\sqrt{t})\frac{|E|}{|F|}$.

Proof. By linearity of expectation,

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

We have

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

and

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

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,

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

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

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

Proof. Fix $j \in I$. We generate a walk $W = (v_0, e_1, v_1, \ldots, v_t)$ 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 a 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. $\text{size}(G_{red}) \leq \text{O}(\text{size}(G))$ *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 $G_e = ((V_e, E_e), \{0,1\}^{\kappa_0}, c_e)$ (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

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

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, $\epsilon \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G_{\text{pre}})$ ∎

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* $\mathrm{UNSAT}(G) = 0$, *then* $\mathrm{UNSAT}(G') = 0$,

3. $\mathrm{UNSAT}(G') \geq \min\{2 \cdot \mathrm{UNSAT}(G), a\}$.

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

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

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

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

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

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., $\mathrm{gap}(1 - 1/|E|, 1)$-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 $\mathrm{size}(G)$ with the property that

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

is constant. Thus we have a reduction from $\mathrm{gap}(1 - 1/|E|, 1)$-Max-CGS to $\mathrm{gap}(1-a, 1)$-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

$$1^n \to G_n$$

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 In this chapter, it will we 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 probabilites 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 $\mathrm{O}(d^k \log n)$ 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 $\mathrm{O}(\log n)$.

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 $\mu_1, \dots, \mu_n$. 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 $\lambda\mu$:

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

These are all eigenvalues, since one can show that if $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ are bases, then $x_i \otimes y_j$, $1 \le i \le m$, $1 \le j \le n$, 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 $\mathrm{O}(d^2 \log \max\{n, n'\})$. (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 $G \circledR H$ 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 ith neighbour of u and u is the jth 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 $G \circledR H$ 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 $G \circledR H$. 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 $G \circledR H$ 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 ith neighbour of u and u is the jth neighbour of v. $\hat{A}$ is a permutation matrix.

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

Excursus: Induced matrix norms

For a norm $\|.\|$ 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 $\|.\|_2$ 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 $\tilde{\lambda}(G \circledR H) \leq 1 - \epsilon\delta^2/24$.*

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

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

$\hat{A}$ 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 replacment 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	$1-\delta$
$G \circledR H$	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 $\mathrm{O}((D+d)\log n)$. (This assume 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 occuring 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
$$G_k := ((G_{\lfloor \frac{k-1}{2} \rfloor} \otimes G_{\lceil \frac{k-1}{2} \rceil})^{50}) \circledR H$$

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 $G \circledR H$ 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 $(p, p', t_1, \ldots, t_k)$ 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 it 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.

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

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 UCONN $\in$ 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 $A \in \mathsf{SL}$ and $B \leq_{\log} A$, then $B \in \mathsf{SL}$.

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 UCONN ∈ 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_0th neighbour of s, then the δ_1th 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 ith neighbour of v, then we determine which neighbour v of u is, say the jth, 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

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

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 $G_k \circledR H$ 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

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

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

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

If we set $k = \mathrm{O}(\log n)$, then G_k is a constant degree expander with $\bar{\lambda}(G_k) \leq \frac{19}{20}$. 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 $k = \mathrm{O}(\log n)$ and finally use the connectivity algorithm for expander graphs. Since L is closed under logarithmic space computable reductions, this would show UCONN ∈ 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 $G_0 \to G_k$ with $k = \mathrm{O}(\log n)$ is deterministic logarithmic space computable.*

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

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

Input: node $v = (i, \delta_1, \ldots, \delta_k)$ of G_k, index $j = (j_1, \ldots, j_{50})$
Output: the jth neighbour of v in G_k

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

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

Input: node $v = (i, \delta_1, \ldots, \delta_k)$ of G_k, index j
Output: the jth neighbour of v in $G_{k-1} \circledR H$

1. If $j \leq d/2$, then return $(i, \delta_1, \ldots, \delta_{k-1}, \delta')$ 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_kth neighbour of $(i, \delta_1, \ldots, \delta_{k-1})$ in G_{k-1}.
   (We perform an external step between two copies of H.)

Note that we can view $(v, \delta_1, \ldots, \delta_k)$ 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])**.** $\texttt{UCONN} \in \mathsf{SL}$.

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

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.

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.

[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.

[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.

[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.

[NTS95] Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995.

[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.