CS359D, Winter 2007

PCP Theorem Quick Reference Sheet

First Step: The PCP Theorem reduces to showing that MAX CG (given a constraint graph satisfiability problem $G = (V, E)$ over a constant-size alphabet Σ, maximize the number of satisfied clauses/edges) is hard to approximate to within some constant.

[Recall proof: the PCP verifier, after preprocessing the input x of the NP-complete language L to transform it into a hard MAX CG instance, and given a purported proof in the form of a variable assignment for this CG instance, checks the validity of a random edge constraint. $O(1)$ repeated trials gives soundness 1/2.]

Gap Amplification Overview: Suffices to design a reduction that maps a CG instance G to a CG instance G' (over the same alphabet) that satisfies three properties: (1) if G is satisfiable, so is G'; (2) if G has gap γ (i.e., every assignment leaves at least a γ fraction of the constraints unsatisfied), then G' has gap at least 2γ (unless it already has a gap exceeding some fixed constant, in which case it remains at least this constant); (3) the size of G' is at most a *constant factor* times that of G.

[Recall proof: to prove that MAX CG is NP-hard to approximate to within some constant, start with an instance of 3-Coloring (which either has zero gap or has gap at least $1/m$, where m is the number of edges), and apply Gap Amplification $\Theta(\log m)$ times. Note this would *not* yield a poly-time reduction if we relaxed constraint (3) above to allow a super-constant blow-up in size each iteration.]

Gap Amplification in 4 Steps:

(1) **Degree-Reduce**(G): The goal is to map the CG instance G to another CG instance G' (over the same alphabet) such that:

- (1a) G' is a d_1-regular graph for some absolute constant d_1.
- (1b) $\text{gap}(G') \geq \text{gap}(G)/c_1$, where c_1 is some absolute constant.

(Recall the proof is basically just the Expander Replacement Lemma with equality edge constraints.)

(2) **Expanderize**(G): The goal is to map the CG instance G (assumed d_1-regular) to another CG instance G' (over the same alphabet) such that:

- (2a) G' is a d_2-regular expander graph for some absolute constant d_2. (I.e., the second eigenvalue of its Laplacian should be at least some absolute constant $\lambda > 0$.)
- (2b) $\text{gap}(G') \geq \text{gap}(G)/c_2$, where c_2 is some absolute constant.

(Recall the proof is to just to throw in a constant-degree regular expander with trivial edge constraints.)

(3) **Powering**(G, t): Given an arbitrarily large constant t, the goal is to map the CG instance $G = (V, E, \Sigma)$ (assumed to be a d_2-regular λ-expander) to another CG instance $G' = (V', E', \Sigma')$ such that:

- (3a) $\text{gap}(G') \geq \frac{t}{c_3} \min\{\text{gap}(G), \frac{1}{t}\}$, where c_3 is some absolute constant (independent of t).

Side effects: screws up degree, expansion, and blows up the alphabet (from Σ to Σ').

(4) **Alphabet-Reduce**(G): The goal is to map the CG instance $G = (V, E, \Sigma)$ to another CG instance $G' = (V', E', \Sigma')$ such that:

- (4a) $|\Sigma'| = 64$.
- (4b) $\text{gap}(G') \geq \text{gap}(G)/c_4$, where c_4 is some absolute constant. In particular, c_4 should be independent of $|\Sigma|$, and therefore independent of the constant t chosen in the Powering step.

In addition to the listed goals, all four subroutines map satisfiable CG instances to satisfiable CG instances, and blow up the size of the given CG instance by at most a constant factor. (In the Powering and Alphabet-Reduce steps, this blow-up is allowed to depend on the constant t chosen in the former step in an arbitrary way.)

[Note these four subroutines implement Gap Amplification—just compose them, taking $t = 2c_1c_2c_3c_4$ in the Powering step.]

The (Untruncated) Amplified Verifier for the Powering Step:

(0) Given: a constant t, a CG instance $G = (V, E, \Sigma)$ (assumed to be a d_2-regular λ-expander), and a proof σ in the following format: for each $v \in V$, σ_v assigns a character $\sigma_v(w)$ of Σ to each vertex w within t hops of v in G. (We view σ_v as a "character" in the polynomially larger alphabet Σ'.)

(1) Choose a vertex $a \in V$ uniformly at random.

(2) Perform an "A-walk" from a, defined as follows:

(2a) Follow a random outgoing edge (v, w) from the current vertex v.

(2b) Stop the random walk and go to Step (3) with probability $1/t$; otherwise return to Step (2a).

(3) Let P denote the walk taken, and let b denote its final endpoint. Query a and b to get σ_a and σ_b.

(4) For each (oriented) step (v, w) of P, perform the following test:

(4a) If $\sigma_a(v)$ and $\sigma_b(w)$ are both defined (i.e., $dist_G(a, v), dist_G(b, w) \leq t$), and if the assignment $(\sigma_a(v), \sigma_b(w))$ fails to satisfy the constraint (v, w) of G, then REJECT.

(5) ACCEPT if all steps of P pass the test in Step (4a).

[Notes: this amplified verifier issues two queries over the larger alphabet Σ'. Also, the "expected number of random bits" required is $\log |E| + O(1)$ since d_2 and t are constants.]

Assignment Testers: A *q-query Assignment Tester (AT)*, with gap $\beta > 0$ and alphabet Σ_0, is a "compiler" with the following properties. It accepts a constraint—for concreteness, given by a Boolean circuit C over a set X of Boolean variables—and outputs a system Ψ of q-ary constraints over $X \cup Y$. Here Y is a set of auxiliary variables defined over the alphabet Σ_0. By definition, it satisfies two conditions:

(1) If $a : X \to \{0, 1\}$ is an assignment that satisfies the given predicate (i.e., $C(a) = 1$), then there is a satisfying extension $b : Y \to \Sigma_0$ (i.e., $a \cup b$ satisfies all of the constraints of Ψ).

(2) If a is δ-far (i.e., has Hamming distance greater than $\delta|X|$) from every satisfying assignment for C, then for every extension $b : Y \to \Sigma_0$, $a \cup b$ violates at least a $\beta\delta$ fraction of Ψ's constraints.

The circuit C is not assumed to be poly-size in X; and Ψ need not be poly-size in X or C.