8 Approximation and Complexity

In this section we will demonstrate that the PCP Theorem is very useful for proving that certain problems are hard to approximate. An *approximation algorithm* for an optimization problem is an algorithm that computes solutions whose cost is within a factor of α of the optimal solution, where $\alpha > 1$ is called the *approximation ratio* of the algorithm. If we have a minimization problem, this means that the solutions should be at most α times the optimum, and in case of a maximization problem, the solutions should be at least $1/\alpha$ times the value of the optimal solution. An optimization problem has a *polynomial time approximation scheme* (or PTAS) if the approximation ratio can be made as small as $1+\epsilon$ for any constant $\epsilon > 0$. A *fully polynomial approximation scheme* (or FPTAS) is a PTAS whose runtime is polynomial in $1/\epsilon$ and the size of the input.

8.1 Hardness of approximating 3SAT

As we know from the midterm exam, the problem 3SAT is NP-complete, and so we believe that 3SAT cannot be solved in polynomial time. Clearly, the NP-hardness of this decision problem implies the NP-hardness of the corresponding optimization problem, called MAX3SAT.

In MAX3SAT, we are given a Boolean expression ϕ in 3CNF and the goal is to find an assignment that maximizes the number of satisfied clauses in ϕ. The maximum fraction of satisfiable clauses in ϕ (i.e. the maximum number of satisfiable clauses divided by the total number of clauses in ϕ) is denoted by $\sigma(\phi)$.

To prove the NP-hardness of computing α-approximations to MAX3SAT for some fixed $\alpha > 1$, we will show that the following decision problem is NP-hard for some constant $\epsilon > 0$.

In MAX3SAT(ϵ) we are given a 3CNF expression ϕ with the property that either $\sigma(\phi) = 1$ or $\sigma(\phi) < 1/(1+\epsilon)$. The output for ϕ is 1 if ϕ is satisfiable and 0 otherwise.

The connection between the hardness of approximation and the PCP Theorem is given by the following result, which is due to [ALMSS].

Theorem 8.1 *MAX3SAT(ϵ) is* NP*-hard for some constant $\epsilon > 0$ if and only if $NP \subseteq \mathrm{PCP}(log, O(1))$.*

Proof. $\Rightarrow$: Let L be an arbitrary NP-language. If MAX3SAT(ϵ) is NP-hard, then there is a polynomial time reduction from L to MAX3SAT(ϵ). Now we show that $L \in \mathrm{PCP}(log, O(1))$. Let x be any input. The verifier first reduces x to an instance ϕ of MAX3SAT(ϵ) that is satisfiable if and only if $x \in L$. As a membership proof for x, the verifier expects a satisfying assignment for ϕ. Given such a membership proof, it does a probabilistic check on it by randomly choosing a clause in ϕ and querying the values of the three variables involved in that clause. It accepts if and only if the clause is satisfied. Clearly, if $x \in L$, then a satisfying assignment for ϕ will cause the verifier to accept with probability 1. On the other hand, if $x \notin L$, then every assignment satisfies less than a $\frac{1}{1+\epsilon}$ fraction of the clauses and so the verifier accepts with probability less than $\frac{1}{1+\epsilon}$. By repeating the check $\Theta(1/\epsilon)$ times (which is still some fixed constant), the acceptance probability can be reduced to $1/2$.

$\Leftarrow$: Suppose that $NP \subseteq \mathrm{PCP}(log, O(1))$, and hence in particular, $SAT \in \mathrm{PCP}(log, O(1))$. We show the NP-hardness of MAX3SAT(ϵ) for some fixed $\epsilon > 0$. Let ψ be an instance of SAT.

Given a value for the random string r, let $q_1(\psi, r), \ldots, q_c(\psi, r)$ denote the positions of the proof string queried by the verifier. For each value of r, the computation of the verifier is determined by the value of the proof string at these positions. Therefore, the outcome of the computation is a function of a constant number of bits in the proof and so can be represented by a constant size 3CNF formula $\phi_{\psi,r}$ in the sense that $\phi_{\psi,r}$ is true if and only if $V^{\pi}(\psi, r)$ accepts. Let ϕ_{ψ} be the conjunction of $\phi_{\psi,r}$ for all possible values of r. If ψ is satisfiable, then there is a proof π such that all $\phi_{\psi,r}$ are true, and therefore ϕ_{ψ} is satisfiable. On the other hand, if ψ is not satisfiable, then, for every π, at least half of the $\phi_{\psi,r}$ in ϕ_{ψ} are not satisfied. Since the expression ϕ_{ψ} can be constructed in polynomial time, this shows that SAT reduces to MAX3SAT(ϵ) for some constant $\epsilon > 0$. □

Therefore, by the fact that NP $\subseteq$ PCP($log, O(1)$), we have the following.

Corollary 8.2 *MAX3SAT does not have a PTAS unless* P = NP.

Proof. Suppose there is a polynomial time approximation algorithm for MAX3SAT with approximation ratio less than $(1 + \epsilon)$, where ϵ is chosen as in Theorem 8.1. On input ψ, apply this algorithm to the expression ϕ_{ψ} resulting from the construction for MAX3SAT(ϵ) in the previous proof. If the algorithm finds an assignment that satisfies more than a fraction of $1/(1 + \epsilon)$ of the clauses, then ϕ must be satisfiable, otherwise ϕ is unsatisfiable. Using this as a polynomial time algorithm for 3SAT would imply that P = NP. □

8.2 Hardness of approximating CLIQUE

A graph is said to contain a *clique* of size k if it has a set of k nodes that form a completely connected graph (i.e. there is an edge for every pair of nodes). The *clique number* $w(G)$ of a graph G is the largest size of a clique in it.

In the CLIQUE problem, we are given a graph G and a number $c \in \mathbb{N}$, and the goal is to decide whether G contains a clique of size at least c.

It is not difficult to show that CLIQUE is NP-complete. In this section we will prove that also approximating the clique number of a graph within a reasonably good bound is NP-hard. For this we need the following lemma.

Lemma 8.3 *There is a polynomial time reduction from MAX3SAT to CLIQUE such that for all 3CNF expressions ϕ with m clauses a graph G_{ϕ} with $3m$ nodes can be constructed with the property that for any $c \in [0, 1]$,*

$$\sigma(\phi) = c \;\Leftrightarrow\; w(G_{\phi}) = c \cdot m \; .$$

Proof. Let $\phi = C_1 \wedge \ldots \wedge C_m$ be a 3CNF with variables $x_1, \ldots, x_n$. We construct $\tau(\phi)$ in the following way: represent each C_i based on the Boolean variables $x_{i_1}, x_{i_2}, x_{i_3}$ by the graph $G_i = (V_i, E_i)$ with $V_i = \{x_{i_1}, x_{i_2}, x_{i_3}\}$ and $E_i = \emptyset$. The graph $G_{\phi} = (V, E)$ is constructed by taking the union of the V_i's and drawing an edge from $x_{i_k} \in V_i$ to $x_{j_l} \in V_j$ for $i \neq j$ if and only if the literals they represent are not negations of each other.

We now show that with this construction, $\sigma(\phi) = c \Leftrightarrow w(G_{\phi}) = c \cdot m$. If $\sigma(\phi) = c$, then there must be $c \cdot m$ different sets V_i that contain a node representing a satisfied literal. Since no one

of these nodes can represent the negation of another, they must form a completely connected graph, and therefore $w(G_\phi) \geq c \cdot m$. On the other hand, if $w(G_\phi) = c \cdot m$, then there must be $c \cdot m$ nodes from different V_i that form a completely connected graph. Since no one of these nodes represents the negation of another, it is possible to have an assignment that satisfies all clauses represented by these V_i's. Hence, $MAX3SAT(\phi) \geq c$ □

Combining this result with Corollary 8.2 we immediately obtain the following result.

Corollary 8.4 *CLIQUE does not have a PTAS unless* P = NP.

Next we show that the situation for CLIQUE is much worse. We do this by exploiting a *self-improvement* property of CLIQUE which is defined by means of a graph product (that is, we present a reduction from CLIQUE to CLIQUE which increases the gap). In this graph product we assume that a graph has a self-loop at every one of its nodes. That is, for a graph $G = (V, E)$ define $\hat{E} = E \cup \{(u, u) : u \in V\}$ and consider then instead of G the graph $\hat{G} = (V, \hat{E})$.

Definition 8.5 *For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ their* product $G_1 \times G_2$ *is the graph whose vertex set is $V_1 \times V_2$ and edge set is*

$$\{((u_1, v_1), (u_2, v_2)) : (u_1, u_2) \in \hat{E}_1 \text{ and } (v_1, v_2) \in \hat{E}_2\} .$$

There is an intuitive way to visualize the product of two graphs G_1 and G_2. Suppose that $V_1 = \{u_1, \ldots, u_m\}$ and $V_2 = \{v_1, \ldots, v_n\}$. Take n copies of G_1, which we will call $H_1, \ldots, H_n$, and place them at the location of the vertices $v_1, \ldots, v_n$ in G_2. We join the vertex labeled u_i in H_j with the vertex labeled u_l in H_k if and only if $(v_j, v_k) \in \hat{E}_2$ and $(u_i, u_l) \in \hat{E}_1$. With this way of visualizing the graph product it is easily seen that $w(G_1 \times G_2) = w(G_1) \cdot w(G_2)$.

Now consider the graph $G^k = G \times \ldots \times G$ (k times). Then

$$w(G^k) = m^k \Leftrightarrow w(G) = m .$$

Unfortunately, since k has to be a constant in order to have a polynomial time transformation, a gap of $\frac{1}{1+\epsilon}$ in G can only be increased to $(\frac{1}{1+\epsilon})^k$ in G^k. Thus, the strength of this gap-producing reduction is somewhat limited. Better bounds can be obtained by using the notion of a booster product.

Definition 8.6 *A* (k, n, α)-booster *is a collection $\mathcal{S}$ of subsets of $\{1, 2, \ldots, n\}$, each of size k, such that for every subset $A \subseteq \{1, \ldots, n\}$, the sets in the collection that are subsets of A constitute a fraction between $(\rho - \alpha)^k$ and $(\rho + \alpha)^k$ of all sets in $\mathcal{S}$, where $\rho = |A|/n$. When $\rho < 1.1\alpha$, the quantity $(\rho - \alpha)^k$ should be considered to be 0.*

We will use the following result from [AFWZ93] about constructibility of booster graphs.

Theorem 8.7 *For any $k = O(\log n)$ and $\alpha > 0$, an (n, k, α)-booster of size poly(n) can be constructed in poly(n) time.*

Given a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a booster $\mathcal{S}$, we define their *booster product* to be constructed by taking as vertices the sets of $\mathcal{S}$, and there is an edge between sets $S_i, S_j \in \mathcal{S}$ if and only if $S_i \cup S_j$ is a clique in G. The following lemma is easily proved from the definitions.

Lemma 8.8 *For any graph G and any (k, n, α)-booster, the clique number of the booster product of G lies between $(\frac{w(G)}{n} - \alpha)^k |\mathcal{S}|$ and $(\frac{w(G)}{n} + \alpha)^k |\mathcal{S}|$.*

Now we have all the definitions and facts we need to prove that CLIQUE is hard to approximate.

Theorem 8.9 *There is a constant $\epsilon > 0$ such that approximating CLIQUE within a factor of n^ϵ is* NP*-hard, where n is the number of vertices in the input graph.*

Proof. Take the graph G resulting from the reduction in Lemma 8.3, and suppose that it has n vertices. The reduction guarantees that, for some constant $\beta > 0$, $w(G)$ is either at least $n/3$ or at most $n(1-\beta)/3$, and it is NP-hard to decide which case holds. Because of Theorem 8.7 we can construct an $(n, \log n, \alpha)$-booster for any constant $\alpha > 0$ in polynomial time. Call it $\mathcal{S}$. Choose $\alpha = \beta/9$ and build the booster product. Now, the clique number of the product is either at least

$$\left(\frac{1}{3} - \alpha\right)^{\log n} |\mathcal{S}| = \left(\frac{3-\beta}{9}\right)^{\log n} |\mathcal{S}|$$

or at most

$$\left(\frac{1-\beta}{3} + \alpha\right)^{\log n} |\mathcal{S}| = \left(\frac{3-2\beta}{9}\right)^{\log n} |\mathcal{S}| \ .$$

Thus, the gap is now n^γ for some constant $\gamma > 0$, and further, $|\mathcal{S}| = poly(n)$. So this gap is $|\mathcal{S}|^\epsilon$ for some constant $\epsilon > 0$. □

8.3 L-Reductions

The reductions presented in the previous sections have not taken into account approximability and are actually inadequate for preserving it. We now introduce a careful kind of reduction that preserves approximability.

Suppose that A and B are optimization problems. An *L-reduction* from A to B is a pair of functions R and S, both computable in polynomial time (the requirement is usually logarithmic space, but we don't need it here), with the following properties:

1. If x is an instance of A with optimum cost $OPT(x)$, then $R(x)$ is an instance of B with optimum cost that satisfies

   $$OPT(R(x)) \leq \alpha \cdot OPT(x) \ ,$$

   where α is a positive constant.

2. If s is any feasible solution for $R(x)$, then $S(s)$ is a feasible solution for x such that

   $$|OPT(x) - c(S(s))| \leq \beta \cdot |OPT(R(x)) - c(s)| \ ,$$

   where β is another positive constant particular to the reduction and c denotes the cost in both instances. That is, S is guaranteed to return a feasible solution of x which is not much more suboptimal than the given solution of $R(x)$.

Notice that, by the second property, an L-reduction is a true reduction: if s is the optimum solution of $R(x)$, then indeed $S(s)$ must be the optimum solution of x.

To illustrate the use of an L-reduction, we look at the following two decision problems.

In the (optimization variant of the) INDEPENDENT SET problem, we are given a graph $G = (V, E)$, and the problem is to find a set $I \subseteq V$ of maximum size such that for all $v, w \in I$, $\{v, w\} \notin G$.

In the (optimization variant of the) of the NODE COVER problem, we are also given a graph $G = (V, E)$, and the problem is to find a set $C \subseteq V$ of minimum size such that every edge in G has at least one of its endpoints in C.

The definitions of INDEPENDENT SET and NODE COVER imply that for any subset $U \subseteq V$ it holds that U is a node cover if and only if $V \setminus U$ is an independent set. Hence, a trivial reduction from INDEPENDENT SET to NODE COVER is: R is the identity function, returning the same graph G, while S replaces C with $V \setminus C$. However, this is not an L-reduction. Its flaw is that the optimum node cover may be arbitrarily larger than the optimum independent set (consider the case in which G is a large clique), violating the first condition.

However, if we restrict our graphs to have maximum degree k, the problems go away, and (R, S) is indeed an L-reduction from k-DEGREE INDEPENDENT SET to k-DEGREE NODE COVER. In proof, if the maximum degree is k, then the maximum independent set is at least $|V|/(k+1)$, while the minimum node cover has at most $|V|$ nodes, and so the constant $\alpha = k+1$ satisfies the first condition. Furthermore, the difference between any cover C and the optimum is the same as the difference between $V \setminus C$ and the maximum independent set, and hence we can take $\beta = 1$ in the second condition.

L-reductions have the important composition property of ordinary reductions:

Proposition 8.10 *If (R, S) is an L-reduction from problem A to problem B and (R', S') is an L-reduction from problem B to problem C, then their composition $(R' \cdot R, S \cdot S')$ is an L-reduction from A to C.*

Proof. It is easy to check that $R' \cdot R$ and $S \cdot S'$ are computable in polynomial time. Furthermore, if x is an instance of A, we have $OPT(R'(R(x))) \leq \alpha' OPT(R(x))$ and $OPT(R(x)) \leq \alpha OPT(x)$, where α and α' are the corresponding constants. Thus, $OPT(R'(R(x))) \leq \alpha \cdot \alpha' OPT(x)$. Similarly, it is easy to check that $S \cdot S'$ satisfies the second condition with the constant $\beta \cdot \beta'$. □

The key property of L-reductions is that they preserve approximability.

Proposition 8.11 *If there is an L-reduction (R, S) from A to B with constants α and β and there is a polynomial time $(1+\epsilon)$-approximation algorithm for B, then there is a polynomial time $1+\gamma$-approximation algorithm for A, where $\gamma = \alpha\beta\epsilon$ if A is a minimization problem and $\gamma = \frac{\alpha\beta\epsilon}{1-\alpha\beta\epsilon}$ if A is a maximization problem.*

Proof. The algorithm is this: given an instance x of A, we construct the instance $R(x)$ of B and then apply to it the assumed $(1+\epsilon)$-approximation algorithm for B to obtain solution s. Finally, we compute the solution $S(s)$ of A.

Using this algorithm, it holds that

$$\begin{aligned} |OPT(x) - c(S(s))| &\leq \beta \cdot |OPT(R(x)) - c(s)| \\ &\leq \beta \cdot \max\{(1+\epsilon) - 1,\ 1 - (\tfrac{1}{1+\epsilon})\} \cdot OPT(R(x)) \\ &\leq \beta \cdot \epsilon \cdot \alpha \cdot OPT(x)\ . \end{aligned}$$

Hence, if A is a minimization problem, the algorithm is a $(1+\alpha\beta\epsilon)$-approximation algorithm for A. If A is a maximimization problem, then we know that

$$c(S(s)) \geq OPT(x) - \alpha\beta\epsilon \cdot OPT(x) = (1-\alpha\beta\epsilon)OPT(x) \; .$$

Hence, the algorithm is a $(1+\frac{\alpha\beta\epsilon}{1-\alpha\beta\epsilon})$-approximation algorithm for A. □

The important property of the expressions $1+\alpha\beta\epsilon$ and $1+\frac{\alpha\beta\epsilon}{1-\alpha\beta\epsilon}$ is that if $1+\epsilon$ tends to one, so do the expressions. This implies the following result.

Corollary 8.12 *If there is an L-reduction from A to B and there is a PTAS for B, then there is a PTAS for A.*

8.4 The class MAXSNP

In this section we show how the L-reduction can be used to establish a class of optimization problems with similar approximability properties.

Let MAXSNP_0 be the class of all optimization problems that can be defined in terms of the expression

$$\max_S |\{(x_1, \ldots, x_k) \in U^k : \; \phi(P_1, \ldots, P_m, S, x_1, \ldots x_k)\}| \; ,$$

where U is a finite universe, $P_1, \ldots, P_m$ and S are predicates (i.e. mappings of the form $U^r \rightarrow \{true, false\}$ for some $r \in \mathbb{N}$), and ϕ is a Boolean expression composed of the predicates and variables $x_1, \ldots, x_k$. k and m are independent of the input, whereas U and $P_1, \ldots, P_m$ depend on the input.

We define MAXSNP to be the class of all optimization problems that are L-reducible to a problem in MAXSNP_0.

Next we give some examples of problems that are in MAXSNP_0. In the problem MAXCUT we are given a graph $G = (V, E)$ and the goal is to find at set $U \subseteq V$ for which the number of edges having exactly one node in U is maximal. MAXCUT is in MAXSNP_0, and therefore in MAXSNP. In proof, MAXCUT can be written as follows:

$$\max_S |\{(x, y) : \; ((P(x, y) \vee P(y, x)) \wedge S(x) \wedge \neg S(y))\}| \; .$$

Here we assume that the graph G is directed. The problem as stated asks for the predicate S for which the number of pairs (x, y) with the property that either (x, y) or (y, x) is an edge (expressed by the relation P) and $S(x)$ is true and $S(y)$ is false is maximized. Given such a predicate S, the set of all x for which $S(x)$ is true achieves the maximum cut.

MAX2SAT (maximizing the number of satisfied clauses, where each clause has two literals) is also in MAXSNP. Here we have three predicates P_0, P_1, and P_2. Intuitively, P_i contains all clauses with i negative literals. That is, $P_0(x, y)$ is true if and only if $(x \vee y)$ is a clause of the represented expression, $P_1(x, y)$ is true if and only if $(\neg x \vee y)$ is a clause, and $P_2(x, y)$ is true if and only if $(\neg x \vee \neg y)$ is a clause. With these somewhat complicated input conventions we can write MAX2SAT as

$$\max_S |\{(x, y) : \; \phi(P_0, P_1, P_2, S, x, y)\}| \; ,$$

where ϕ is the following expression:

$$(P_0(x,y) \wedge (S(x) \vee S(y))) \vee (P_1(x,y) \wedge (\neg S(x) \vee S(y))) \vee (P_2(x,y) \wedge (\neg S(x) \vee \neg S(y)))$$

where S stands now for the assignment that achieves a maximum number of satisfiable clauses. A similar construction, with four predicates, establishes that MAX3SAT is in MAXSNP.

We say that a problem in MAXSNP is MAXSNP-*complete* if all problems in MAXSNP L-reduce to it. From Corollary 8.12 we have:

Proposition 8.13 *If a* MAXSNP-*complete problem has a PTAS, then all problems in* MAXSNP *have one.*

Naturally, it is not a priori clear that MAXSNP-complete problems exist. But they do:

Theorem 8.14 *MAX3SAT is* MAXSNP-*complete.*

Proof. Since any problem in MAXSNP by definition can be L-reduced to a problem in MAXSNP_0, it suffices to show that all problems in MAXSNP_0 can be L-reduced to MAX3SAT. Consider such a problem A defined by the expression $\max_S |\{(x_1, \ldots, x_k) : \phi\}|$. For each k-tuple $y = (y_1, \ldots, y_k) \in U^k$, let us substitute these values for $x_1, \ldots, x_k$ in ϕ to obtain an expression ϕ_y. There are two kinds of atomic expressions left in ϕ_y, namely those that use predicates P_i and those that use S. The former kind can be readily evaluated to true or false from the known values of the predicates for y. Thus, ϕ can be simplified to a Boolean expression that consists only of atomic expressions $S(y_{i_1}, \ldots, y_{i_r})$ (assuming that S is a mapping from U^r to $\{true, false\}$).

Since k is independent of the input, each ϕ_y has a constant size and therefore can be transformed into an equivalent 3CNF expression ϕ'_y of constant size. This is done in the following way. First, we draw ϕ_y as a circuit consisting of $\wedge$, $\vee$, and $\neg$ gates. Each ouput of a gate in this circuit is then represented by a new Boolean variable. The 3CNF expression ϕ'_y is then composed of clauses that are satisfied if and only if the outcomes of the gates are correct. Furthermore, the variable representing the output of the circuit is added to ϕ'_y as a clause to make sure that ϕ'_y is satisfiable if and only if ϕ_y is satisfied.

Since ϕ'_y has only a constant size, it is possible to find and discard those expressions ϕ'_y that are not satisfiable. The conjunction of all satisfiable ϕ'_y is the desired instance $R(x)$ of MAX3SAT, where x is the original input for A. From any truth assignment T for this instance we immediately get a feasible solution $S = S'(T)$ (S' represents the function for the second condition of the L-reduction) of the instance x of A by simply recovering S from the Boolean values of the variables. The description of the L-reduction is complete.

It remains to show that this is indeed an L-reduction. Each satisfiable 3CNF expression ϕ'_y consists of at most c_1 clauses, where c_1 is a constant depending on the size of ϕ. Hence, if the number of satisfiable expressions ϕ'_y is m, then $OPT(R(x)) \leq c_1 \cdot m$. On the other hand, the optimum value of instance x is at least $c_2 \cdot m$ for some constant $c_2 > 0$. The reason for this is that for every satisfiable ϕ_y consisting of k atomic expressions $S(y_{i_1}, \ldots, y_{i_r})$ it holds that at least 1 of the 2^k possible truth assignments for the expressions satisfy ϕ_y. Hence, for a random truth assignment for all possible $S(y_{i_1}, \ldots, y_{i_r})$, the expected fraction of satisfied ϕ_y's is at least

2^{-k}. Thus, there must be a truth assignment so that the value for instance x is at least $2^{-k} \cdot m$. Since k is a constant, also $c_2 = 2^{-k}$ is a constant.

So the first condition in the definition of the L-reduction is met with $\alpha = c_1/c_2$. It remains to check the second condition. No matter how S is chosen, it is always possible to come up with a truth assignment for the variables in ϕ'_y so that all clauses apart from the clause representing the output of the circuit are true. Hence, we can lift the cost function for MAX3SAT to the situation that the number of unsatisfied clauses is equal to the number of unsatisfied expressions ϕ'_y. Since this number is identical to the number of unsatisfied ϕ for the recovered S (using the function S'), the second condition is satisfied with $\beta = 1$. □

Since we know from Corollary 8.2 that there cannot be a PTAS for MAX3SAT unless P = NP, Corollary 8.12 immediately implies the following result.

Corollary 8.15 *No* MAXSNP*-complete problem can have a PTAS unless* P = NP.

Examples of MAXSNP-complete problems are MAX2SAT, 4-DEGREE INDEPENDENT SET, NODE COVER, and MAXCUT.

8.5 References

- N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized graph products. In *Computational Complexity*, Birkhauser Verlag, 1995.
- S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM* 45:501–555, 1998.
- S. Arora, S. Safra. Probabilistically checkable proofs: a new characterization of NP. *Journal of the ACM* 45:70–122, 1998.
- S. Arora. Around the PCP Theorem. Technical Report SOCS-97.2, School of Computer Science, McGill University.
- O. Goldreich. Introduction to Complexity Theory. Lecture notes. Dept. of Computer Science and Applied Mathematics, Weizmann Institute.