29 Hardness of Approximation

A remarkable achievement of the theory of exact algorithms is that it has provided a fairly complete characterization[1] of the intrinsic complexity of natural computational problems, modulo some strongly believed conjectures. Recent impressive developments raise hopes that we will some day have a comprehensive understanding of the approximability of **NP**-hard optimization problems as well. In this chapter we will give a brief overview of these developments.

Current hardness results fall into three important classes. For minimization problems, the hardness factors for these classes are constant (> 1), $\Omega(\log n)$, and n^ε for a fixed constant $\varepsilon > 0$, where n is the size of the instance. For maximization problems, the factors are constant (< 1), $O(1/\log n)$, and $1/n^\varepsilon$ for a fixed $\varepsilon > 0$. In this chapter we will present hardness results for MAX-3SAT, vertex cover, and Steiner tree in the first class, set cover in the second class, and clique in the third class. For all these problems, we will establish hardness for their cardinality versions, i.e., the unit cost case.

29.1 Reductions, gaps, and hardness factors

Let us start by recalling the methodology for establishing hardness results for exact optimization problems. The main technical core is the Cook–Levin theorem which establishes the hardness, assuming $\mathbf{P} \neq \mathbf{NP}$, of distinguishing between instances of SAT that are satisfiable and those that are not. To show hardness of computing an optimal solution to, say the cardinality vertex cover problem, one shows, via a polynomial time reduction from SAT, that it is hard to distinguish between graphs that have covers of size at most k from graphs that don't, where k is provided as part of the input. Since an exact algorithm can make this distinction, this reduction establishes the non-existence of an efficient exact algorithm.

The main technical core of hardness of approximation results is the PCP theorem, which is stated in Section 29.2. For establishing a hardness of approximation result for, say, the vertex cover problem, this theorem is used to

[1] A few (important) exceptions, such as the graph isomorphism problem, remain uncharacterized.

show the following polynomial time reduction. It maps an instance ϕ of SAT to a graph $G = (V, E)$ such that

- if ϕ is satisfiable, G has a vertex cover of size $\leq \frac{2}{3}|V|$, and
- if ϕ is not satisfiable, the smallest vertex cover in G is of size $> \alpha \cdot \frac{2}{3}|V|$,

where $\alpha > 1$ is a fixed constant.

Claim 29.1 *As a consequence of the reduction stated above, there is no polynomial time algorithm for vertex cover that achieves an approximation guarantee of α, assuming $\mathbf{P} \neq \mathbf{NP}$.*

Proof: Essentially, this reduction establishes the hardness, assuming $\mathbf{P} \neq \mathbf{NP}$, of distinguishing graphs having a cover of size $\leq \frac{2}{3}|V|$ from those having a cover of size $> \alpha \cdot \frac{2}{3}|V|$. An approximation algorithm for vertex cover, having a guarantee of α or better, will find a cover of size $\leq \alpha \cdot \frac{2}{3}|V|$ when given a graph G from the first class. Thus, it will be able to distinguish the two classes of graphs, leading to a contradiction. □

The reduction stated above introduces a gap, of factor α, in the optimal objective function value achieved by the two classes of graphs (if $\alpha = 1$ then this is an ordinary polynomial time reduction from SAT to vertex cover). Let us formally state the central notion of a *gap-introducing reduction*. The definition is slightly different for minimization and maximization problems. For simplicity, let us assume that we are always reducing from SAT.

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α. Given an instance ϕ of SAT, it outputs, in polynomial time, an instance x of Π, such that

- if ϕ is satisfiable, $\mathrm{OPT}(x) \leq f(x)$, and
- if ϕ is not satisfiable, $\mathrm{OPT}(x) > \alpha(|x|) \cdot f(x)$.

Notice that f is a function of the instance (such as $\frac{2}{3}|V|$ in the example given above), and α is a function of the size of the instance. Since Π is a minimization problem, the function α satisfies $\alpha(|x|) \geq 1$.

If Π is a maximization problem, we want the reduction to satisfy

- if ϕ is satisfiable, $\mathrm{OPT}(x) \geq f(x)$, and
- if ϕ is not satisfiable, $\mathrm{OPT}(x) < \alpha(|x|) \cdot f(x)$.

In this case, $\alpha(|x|) \leq 1$. The gap, $\alpha(|x|)$, is precisely the hardness factor established by the gap-introducing reduction for the **NP**-hard optimization problem.

Once we have obtained a gap-introducing reduction from SAT (or any other **NP**-hard problem) to an optimization problem, say Π_1, we can prove a hardness result for another optimization problem, say Π_2, by giving a special reduction, called a *gap-preserving reduction*, from Π_1 to Π_2. Now there are four possibilities, depending on whether Π_1 and Π_2 are minimization or maximization problems. We give the definition below assuming that Π_1 is

a minimization problem and Π_2 is a maximization problem. The remaining cases are similar.

A gap-preserving reduction, Γ, from Π_1 to Π_2 comes with four parameters (functions), f_1, α, f_2, and β. Given an instance x of Π_1, it computes, in polynomial time, an instance y of Π_2 such that

- $\text{OPT}(x) \leq f_1(x) \Rightarrow \text{OPT}(y) \geq f_2(y)$,
- $\text{OPT}(x) > \alpha(|x|) f_1(x) \Rightarrow \text{OPT}(y) < \beta(|y|) f_2(y)$.

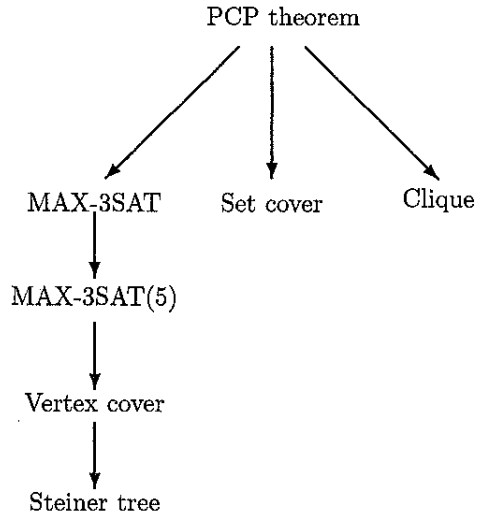
Observe that x and y are instances of two different problems, and so it would be more appropriate to write $\text{OPT}_{\Pi_1}(x)$ and $\text{OPT}_{\Pi_2}(y)$ instead of $\text{OPT}(x)$ and $\text{OPT}(y)$, respectively. However, we will avoid this extra notation, since the context clarifies the problems being talked about. In keeping with the fact that Π_1 is a minimization problem and Π_2 is a maximization problem, $\alpha(|x|) \geq 1$ and $\beta(|y|) \leq 1$.

Composing a gap-introducing reduction with a gap-preserving reduction gives a gap-introducing reduction, provided all the parameters match up. For example, suppose that in addition to the reduction Γ defined above, we have obtained a gap-introducing reduction, Γ', from SAT to Π_1, with parameters f_1 and α. Then, composing Γ' with Γ, we get a gap-introducing reduction from SAT to Π_2, with parameters f_2 and β. This composed reduction shows that there is no $\beta(|y|)$ factor approximation algorithm for Π_2, assuming $\mathbf{P} \neq \mathbf{NP}$. In each gap-preserving reduction stated below, we will take special care to ensure that the parameters match up.

Remark 29.2

- The "gap" β can, in general, be bigger or smaller than α. In this sense, "gap-preserving" is a slight misnomer.
- We do not require any guarantee from reduction Γ if instance x of Π_1 falls in the first gap, i.e., satisfies $f_1(x) < \text{OPT}(x) \leq \alpha(|x|) f_1(x)$.
- An approximation algorithm for Π_2 together with a gap-preserving reduction Γ from Π_1 to Π_2 does not necessarily yield an approximation algorithm for Π_1. Observe the contrast with an approximation factor preserving reduction (see Section A.3.1 for definition). The latter reduction additionally requires a means of transforming a near-optimal solution to the transformed instance y of Π_2 into a near-optimal solution to the given instance x of Π_1.
  On the other hand, Γ together with an appropriate gap-introducing reduction from SAT to Π_1 does suffice for proving a hardness of approximation result for Π_2. Obviously the less stringent requirement on gap-preserving reductions makes them easier to design.
- We have already presented some gap-introducing reductions, e.g., Theorems 3.6 and 5.7. The reader may wonder why these do not suffice as the starting point for further hardness results and why the PCP theorem was needed. The reason is that these reductions simply exploit the freedom to choose edge costs and not the deep combinatorial structure of the problem.

The following figure shows the gap-preserving reductions presented in this chapter:



29.2 The PCP theorem

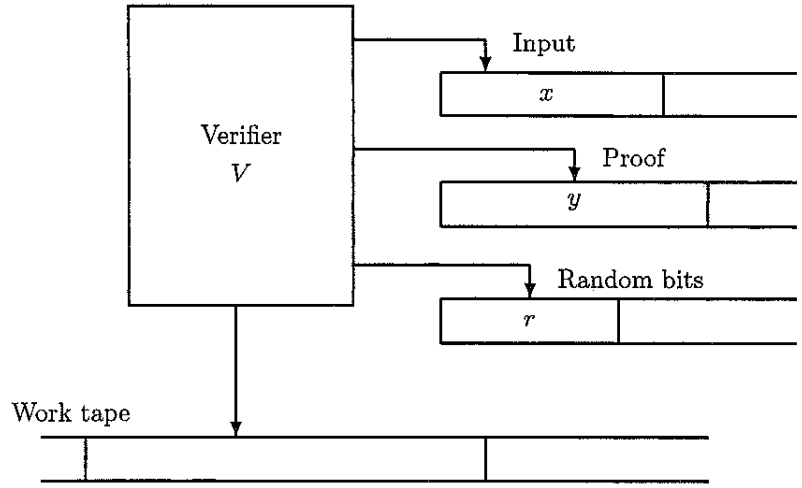
Probabilistic characterizations of the class **NP** yield a general technique for obtaining gap-introducing reductions. The most useful of these characterizations is captured in the PCP theorem. PCP stands for *probabilistically checkable proof systems.*

Recall the usual definition of **NP** (see Appendix A) as the class of languages whose yes instances support short (polynomial in the length of the input) witnesses that can be verified quickly (in polynomial time). Informally, a probabilistically checkable proof for an **NP** language encodes the witness in a special way so that it can be verified probabilistically by examining very few of its bits.

A probabilistically checkable proof system comes with two parameters, the number of random bits required by the verifier, and the number of bits of the witness that the verifier is allowed to examine. In keeping with established terminology, let us call a witness string the *proof.* The most useful setting for these parameters is $O(\log n)$ and $O(1)$, respectively. This defines the class $\mathbf{PCP}(\log n, 1)$.

The *verifier* is a polynomial time Turing machine which, besides its input tape and work tape, has a special tape that provides it with a string of random bits and another special tape on which it is provided with the proof. The machine can read any bit of the proof by simply specifying its location. Of course, the particular locations it examines are a function of the input

string and the random string. At the end of its computation, the machine goes into either an accept state or a reject state.



A language $L \in \mathbf{PCP}(\log n, 1)$ if there is a verifier V, and constants c and q, such that on input x, V obtains a random string, r, of length $c \log |x|$ and queries q bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability 1,
- if $x \notin L$, then for every proof y, V accepts with probability $< 1/2$,

where the probability is over the random string r. The probability of accepting in case $x \notin L$ is called the *error probability*.

In general, for two functions $r(n)$ and $q(n)$, we can define the class $\mathbf{PCP}(r(n), q(n))$, under which the verifier obtains $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof. The acceptance criteria for input strings are the same as above. In this terminology, $\mathbf{NP} = \mathbf{PCP}(0, \text{poly}(n))$, where $\text{poly}(n) = \bigcup_{k \geq 0} \{n^k\}$. In this case, the verifier is not allowed any random bits. It must deterministically accept strings in the language and reject strings not in the language, as in the definition of **NP**. The PCP theorem gives another characterization of **NP**.

Theorem 29.3 $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$.

One half of this theorem, that $\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NP}$, is easy to prove (see Exercise 29.1). The other half, that $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$, is a difficult result, and gives a useful tool for establishing hardness of approximation results. The currently known proof of this half is too complicated for exposition in this book. Fortunately, the statement of the theorem is sufficient to derive the hardness results.

In order to provide the reader with some feel for the PCP theorem, let us make an observation. It is easy to construct a verifier for 3SAT whose error

probability (i.e., probability of accepting unsatisfiable formulae) is $\leq 1 - 1/m$, where m is the number of clauses in the input 3SAT formula, say ϕ. The verifier expects a satisfying truth assignment to ϕ as the proof. It uses the $O(\log n)$ random bits to pick a random clause of ϕ. It then reads the truth assignments for the three variables occurring in this clause. Notice that this is only a constant number of bits. It accepts iff the truth setting for these three variables satisfies the clause. Clearly, if ϕ is satisfiable, there is a proof that makes the verifier accept with probability 1, and if ϕ is not satisfiable, on every proof, the verifier accepts with probability $\leq 1 - 1/m$. The interesting and difficult part of the PCP theorem is decreasing the error probability to $< 1/2$, even though the verifier is allowed to read only a constant number of bits of the proof. It involves a complex algebraic construction that ensures that small parts of the proof depend on every bit of the input.

The PCP theorem directly gives an optimization problem – in particular, a maximization problem – for which there is no factor 1/2 approximation algorithm, assuming $\mathbf{P} \neq \mathbf{NP}$.

Problem 29.4 (Maximize accept probability) Let V be a $\mathbf{PCP}(\log n, 1)$ verifier for SAT. On input ϕ, a SAT formula, find a proof that maximizes the probability of acceptance of V.

Claim 29.5 *Assuming* $\mathbf{P} \neq \mathbf{NP}$, *there is no factor* 1/2 *approximation algorithm for Problem 29.4.*

Proof: If ϕ is satisfiable, then there is a proof that makes V accept with probability 1, and if ϕ is not satisfiable, then on every proof, V accepts with probability $< 1/2$. Suppose there is a factor 1/2 approximation algorithm for Problem 29.4. If ϕ is satisfiable, then this algorithm must provide a proof on which V's acceptance probability is $\geq 1/2$. The acceptance probability can be computed in polynomial time, by simply simulating V for all random strings of length $O(\log n)$. Thus, this approximation algorithm can be used for deciding SAT in polynomial time, contradicting the assumption $\mathbf{P} \neq \mathbf{NP}$. □

Claim 29.5 directly gives the following corollary. In subsequent sections, we will use the PCP theorem to obtain hardness results for natural computational problems.

29.6 Hardness of clique

The best approximation algorithms known for some problems, including clique, are extremely weak – to the extent that the solution produced by the best known algorithm is only very slightly better than picking a trivial feasible solution. Recent hardness results have been invaluable in explaining why this is so: these problems are inherently inapproximable (essentially). In this section, we will establish this for clique:

Problem 29.15 (Clique) Given an undirected graph $G = (V, E)$ with nonnegative weights on vertices, find a clique of maximum weight. A *clique* in G is a subset of vertices, $S \subseteq V$, such that for each pair $u, v \in S$, $(u, v) \in E$. Its *weight* is the sum of weights of its vertices.

Consider the cardinality version of this problem, i.e., when all vertex weights are unit. In this section we will show that there is a constant $\varepsilon_q > 0$, such that there is no $1/(n^{\varepsilon_q})$ factor approximation algorithm for this problem, assuming $\mathbf{P} \neq \mathbf{NP}$. Let us first prove the following weaker result.

Lemma 29.16 *For fixed constants b and q, there is a gap-introducing reduction from SAT to clique that transforms a Boolean formula ϕ of size n to a graph $G = (V, E)$, where $|V| = 2^q n^b$, such that*

- *if ϕ is satisfiable,* $\mathrm{OPT}(G) \geq n^b$, *and*
- *if ϕ is not satisfiable,* $\mathrm{OPT}(G) < \frac{1}{2} n^b$.

Proof: Let F be a $\mathbf{PCP}(\log n, 1)$ verifier for SAT that requires $b \log n$ random bits and queries q bits of the proof. We will transform a SAT instance, ϕ, of size n to a graph $G = (V, E)$ as follows. For each choice of a binary string, r, of $b \log n$ bits, and each truth assignment, τ, to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Thus, $|V| = 2^q n^b$.

Let $Q(r)$ represent the q positions in the proof that F queries when it is given string r as the "random" string. We will say that vertex $v_{r,\tau}$ is *accepting* if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof; it is *rejecting* otherwise. Vertices v_{r_1,τ_1} and v_{r_2,τ_2} are *consistent* if τ_1 and τ_2 agree at each position at which $Q(r_1)$ and $Q(r_2)$ overlap. Clearly, a necessary condition for consistency is that $r_1 \neq r_2$. Two distinct vertices v_{r_1,τ_1} and v_{r_2,τ_2} are connected by an edge in G iff they are consistent and they are both accepting. Vertex $v_{r,\tau}$ is *consistent* with proof p if positions $Q(r)$ of p contain τ.

If ϕ is satisfiable, there is a proof, p, on which F accepts for each choice, r, of the random string. For each r, let $p(r)$ be the truth setting assigned by proof p to positions $Q(r)$. Now, the vertices $\{v_{r,p(r)} \mid |r| = b \log n\}$ form a clique in G of size n^b.

Next, suppose that ϕ is not satisfiable, and let C be a clique in G. Since the vertices of C are pairwise consistent, there is a proof, p, that is consistent with all vertices of C. Therefore, the probability of acceptance of F on proof p is at least $|C|/n^b$ (notice that the vertices of C must correspond to distinct random strings). Since the probability of acceptance of any proof is $< 1/2$ the largest clique in G must be of size $< \frac{1}{2} n^b$. □

As a consequence of Lemma 29.16, there is no factor $1/2$ approximation algorithm for clique assuming $\mathbf{P} \neq \mathbf{NP}$. Observe that the hardness factor established is precisely the bound on the error probability of the probabilistically checkable proof for SAT. By the usual method of simulating the verifier a constant number of times, this can be made $1/k$ for any constant k, leading to a similar hardness result for clique. In order to achieve the claimed hardness, the error probability needs to be made inverse polynomial. This motivates generalizing the definition of **PCP** as follows. Let us define two additional parameters, c and s, called *completeness* and *soundness*, respectively. A language $L \in \mathbf{PCP}_{c,s}[r(n), q(n)]$ if there is a verifier V, which on input x of length n, obtains a random string of length $O(r(n))$, queries $O(q(n))$ bits of the proof, and satisfies:

- if $x \in L$, there is a proof y that makes V accept with probability $\geq c$,

- if $x \notin L$, then for every proof y, V accepts with probability $< s$.

Thus, the previously defined class $\mathbf{PCP}[r(n), q(n)] = \mathbf{PCP}_{1,\frac{1}{2}}[r(n), q(n)]$. In general, c and s may be functions of n.

We would like to obtain a PCP characterization of **NP** which has inverse polynomial soundness. An obvious way of reducing soundness is to simulate a $\mathbf{PCP}[\log n, 1]$ verifier multiple number of times and accept iff the verifier accepts each time. Simulating k times will reduce soundness to $1/2^k$; however, this will increase the number of random bits needed to $O(k \log n)$ and the number of query bits to $O(k)$. Observe that the number of vertices in the graph constructed in Lemma 29.16 is $2^{O(r(n)+q(n))}$. To achieve inverse polynomial soundness, k needs to be $\Omega(\log n)$. For this value of k, the number of bits queried is $O(\log n)$, which is not a problem. However, the number of random bits needed is $O(\log^2 n)$, which leads to a superpolynomial sized graph.

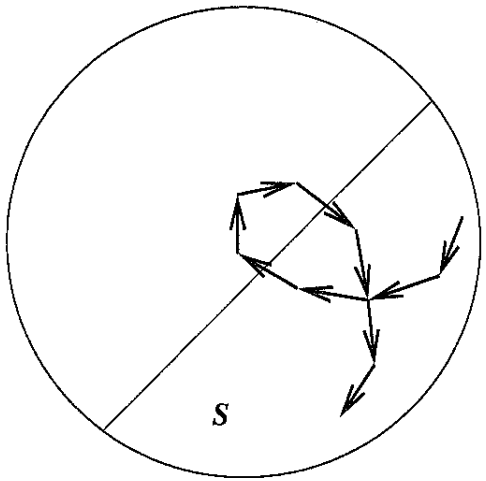
The following clever idea overcomes this difficulty. We will use a constant degree expander graph to generate $O(\log n)$ strings of $b \log n$ bits each, using only $O(\log n)$ truly random bits. The verifier will be simulated using these $O(\log n)$ strings as the "random" strings. Clearly, these are not truly random strings. Properties of expanders help show that they are "almost random" – the probability of error still drops exponentially in the number of times the verifier is simulated.

Let H be a constant degree expander on n^b vertices, each vertex having a unique $b \log n$ bit label. A random walk on H of length $O(\log n)$ can be constructed using only $O(\log n)$ bits, $b \log n$ bits to pick the starting vertex at random and a constant number of bits to pick each successive vertex. (Observe that the random walk is started in the stationary distribution, which is uniform since the graph is regular.) The precise property of expanders we will need is the following.

Theorem 29.17 *Let S be any set of vertices of H of size $< (n^b)/2$. There is a constant k such that*

$$\mathbf{Pr}[\textit{ all vertices of a } k \log n \textit{ length random walk lie in } S\ ] < \frac{1}{n}.$$

For intuitive justification for Theorem 29.17, observe that a constant fraction of the edges incident at vertices of S have their other end points in $\overline{S}$ – these help the walk escape from S. The following figure shows a walk on H that does not lie in S:



Theorem 29.18 $\mathbf{NP} = \mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n]$

Proof: We will prove the difficult half,

$$\mathbf{PCP}_{1,\frac{1}{2}}[\log n, 1] \subseteq \mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n],$$

and leave the rest as Exercise 29.5. Let $L \in \mathbf{PCP}_{1,\frac{1}{2}}[\log n, 1]$. Let F be a verifier for L which requires $b \log n$ random bits and queries q bits of the proof, where b and q are constants.

Next, we give a $\mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n]$ verifier for L, F', which constructs the expander graph H defined above. It then constructs a random walk of length $k \log n$ on H, using $O(\log n)$ random bits. Both constructions can be accomplished in polynomial time. The label of each vertex on this path specifies a $b \log n$ bit string. It uses these $k \log n + 1$ strings as the "random" strings on which it simulates verifier F. F' accepts iff F accepts on all $k \log n + 1$ runs.

Consider string $x \in L$, and let p be a proof that makes verifier F accept x with probability 1. Clearly, F', given proof p, also accepts x with probability 1. Hence the completeness of the new proof system is 1.

Next, consider string $x \notin L$, and let p be an arbitrary proof supplied to F'. When given proof p, verifier F accepts on $< (n^b)/2$ random strings of length $b \log n$. Let S denote the corresponding set of vertices of H, $|S| < (n^b)/2$. Now, F' accepts x iff the random walk remains entirely in S. Since the probability of this event is $< 1/n$, the soundness of F' is $1/n$. Finally observe that F' requires only $O(\log n)$ random bits and queries $O(\log n)$ bits of the proof. □

Theorem 29.19 *For fixed constants b and q, there is a gap-introducing reduction from SAT to clique that transforms a Boolean formula ϕ of size n to a graph $G = (V, E)$, where $|V| = n^{b+q}$, such that*

- *if ϕ is satisfiable,* $\mathrm{OPT}(G) \geq n^b$, *and*
- *if ϕ is not satisfiable,* $\mathrm{OPT}(G) < n^{b-1}$.

Proof: Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}[\log n, \log n]$ verifier for SAT that requires $b \log n$ random bits and queries $q \log n$ bits of the proof. The transformation of SAT instance ϕ to graph G is exactly as in Lemma 29.16. The only difference is that the increased number of bits queried results in a larger number of vertices.

The correctness of the construction also along the lines of Lemma 29.16. If ϕ is satisfiable, let p be a good proof, and pick the n^b vertices of G that are consistent with p, one for each choice of the random string. These vertices will form a clique in G. Furthermore, any clique C in G gives rise to a proof that is accepted by F with probability $\geq |C|/n^b$. Since the soundness of F is $1/n$, if ϕ is not satisfiable, the largest clique in G is of size $< n^{b-1}$. □

Corollary 29.20 *There is no $1/(n^{\varepsilon_q})$ factor approximation algorithm for the cardinality clique problem, assuming $\mathbf{P} \neq \mathbf{NP}$, where $\varepsilon_q = 1/(b+q)$, for constants b and q defined in Theorem 29.19.*