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

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

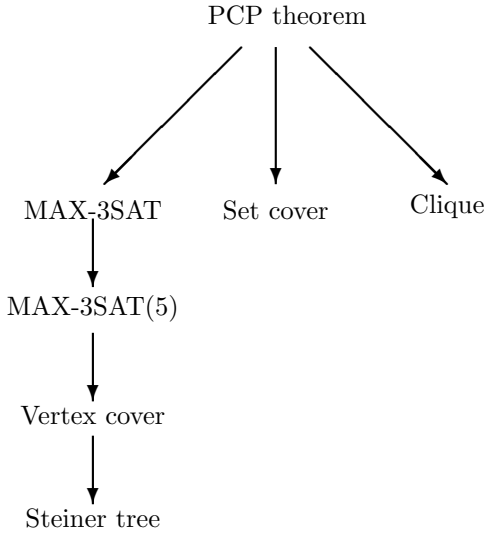
Observe that x and y are instances of two different problems, and so it would be more appropriate to write $\mathrm{OPT}_{\Pi_1}(x)$ and $\mathrm{OPT}_{\Pi_2}(y)$ instead of $\mathrm{OPT}(x)$ and $\mathrm{OPT}(y)$, respectively. However, we will avoid this extra notation, since the context clarifies 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 $f_1(x) < \mathrm{OPT}(x) \leq \alpha(|x|) f_1(x)$.
- 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

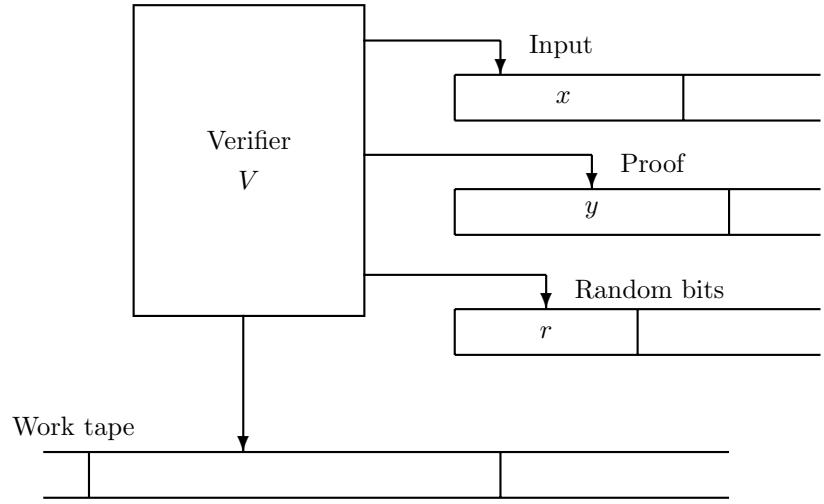
Probabilistic characterizations of the class **NP** yield a general technique for obtaining gap-introducing reductions. The most useful of these characterizations is captured in the PCP theorem. PCP stands for *probabilistically checkable proof systems*.

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, $\mathbf{NP} = \mathbf{PCP}(0, poly(n))$, where $poly(n) = \bigcup_{k \geq 0} \{n^k\}$. 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. A similar corollary follows in each case.

Corollary 29.6 *Assuming* $\mathbf{P} \neq \mathbf{NP}$*, there is no PTAS for Problem 29.4.*

29.3 Hardness of MAX-3SAT

MAX-3SAT is the restriction of MAX-SAT (see Problem 16.1) to instances in which each clause has at most three literals. This problem plays a similar role in hardness of approximation as 3SAT plays in the theory of **NP**-hardness,

as a "seed" problem from which reductions to numerous other problems have been found. The main result of this section is:

Theorem 29.7 *There is a constant $\varepsilon_M > 0$ for which there is a gap-introducing reduction from SAT to MAX-3SAT that transforms a Boolean formula ϕ to ψ such that*

- *if ϕ is satisfiable,* $\mathrm{OPT}(\psi) = m$, *and*
- *if ϕ is not satisfiable,* $\mathrm{OPT}(\psi) < (1 - \varepsilon_M)m$,

where m is the number of clauses in ψ.

Corollary 29.8 *There is no approximation algorithm for MAX-3SAT with an approximation guarantee of $1 - \varepsilon_M$, assuming $\mathbf{P} \neq \mathbf{NP}$, where $\varepsilon_M > 0$ is the constant defined in Theorem 29.7.*

The exact solution of MAX-3SAT is shown hard under the assumption $\mathbf{P} \neq \mathbf{NP}$. It is interesting to note that hardness of approximate solution of MAX-3SAT is also being established under the same assumption.

For clarity, let us break the proof into two parts. We will first prove hardness for the following problem.

Problem 29.9 (MAX k-FUNCTION SAT) Given n Boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$, each of which is a function of k of the Boolean variables, find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of functions satisfied. Here k is assumed to be a fixed constant. Thus, we have a class of problems, one for each value of k.

Lemma 29.10 *There is a constant k for which there is a gap-introducing reduction from SAT to MAX k-FUNCTION SAT that transforms a Boolean formula ϕ to an instance I of MAX k-FUNCTION SAT such that*

- *if ϕ is satisfiable,* $\mathrm{OPT}(I) = m$, *and*
- *if ϕ is not satisfiable,* $\mathrm{OPT}(I) < \frac{1}{2}m$,

where m is the number of formulae in I.

Proof: Let V be a $\mathbf{PCP}(\log n, 1)$ verifier for SAT, with associated parameters c and q. Let ϕ be an instance of SAT of length n. Corresponding to each string, r, of length $c \log n$ (the "random" string), V reads q bits of the proof. Thus, V reads a total of at most qn^c bits of the proof. We will have a Boolean variable corresponding to each of these bits. Let B be the set of Boolean variables. Thus, the relevant part of each proof corresponds to a truth assignment to the variables in B.

We will establish the lemma for $k = q$. Corresponding to each string r, we will define a Boolean function, f_r. This will be a function of q variables from B. The acceptance or rejection of V is of course a function of ϕ, r, and the q bits of the proof read by V. For fixed ϕ and r, consider the restriction of this function to the q bits of the proof. This is the function f_r.

Clearly, there is a polynomial time algorithm which, given input ϕ, outputs the $m = n^c$ functions f_r. If ϕ is satisfiable, there is a proof that makes V accept with probability 1. The corresponding truth assignment to B satisfies all n^c functions f_r. On the other hand, if ϕ is not satisfiable, then on every proof, V accepts with probability $< 1/2$. Thus, in this case every truth assignment satisfies $< \frac{1}{2}n^c$ of these functions. The lemma follows. □

Proof of Theorem 29.7: Using Lemma 29.10 we transform a SAT formula ϕ to an instance of MAX k-FUNCTION SAT. We now show how to obtain a 3SAT formula from the n^c functions.

Each Boolean function f_r constructed in Lemma 29.10 can be written as a SAT formula, say ψ_r, containing at most 2^q clauses. Each clause of ψ_r contains at most q literals. Let ψ be the SAT formula obtained by taking the conjunct of all these formulae, i.e., $\psi = \bigwedge_r \psi_r$.

If a truth assignment satisfies formula f_r, then it satisfies all clauses of ψ_r. On the other hand, if it does not satisfy f_r, then it must leave at least one clause of ψ_r unsatisfied. Therefore, if ϕ is not satisfiable, any truth assignment must leave $> \frac{1}{2}n^c$ clauses of ψ unsatisfied.

Finally, let us transform ψ into a 3SAT formula. This is done using the standard trick of introducing new variables to obtain small clauses from a big clause. Consider clause $C = (x_1 \vee x_2 \vee \ldots \vee x_k)$, with $k > 3$. Introduce $k-2$ new Boolean variables, $y_1, \ldots, y_{k-2}$, and consider the formula

$$f = (x_1 \vee x_2 \vee y_1) \wedge (\overline{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\overline{y}_{k-2} \vee x_{k-1} \vee x_k).$$

Let τ be any truth assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to a truth assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for every way of setting $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.

We apply this construction to every clause of ψ containing more than 3 literals. Let ψ' be the resulting 3SAT formula. It contains at most $n^c 2^q(q-2)$ clauses. If ϕ is satisfiable, then there is a truth assignment satisfying all clauses of ψ'. If ϕ is not satisfiable, $> \frac{1}{2}n^c$ of the clauses remain unsatisfied, under every truth assignment. Setting $\varepsilon_M = 1/(2^{q+1}(q-2))$ gives the theorem. □

29.4 Hardness of MAX-3SAT with bounded occurrence of variables

For each fixed k, define MAX-3SAT(k) to be the restriction of MAX-3SAT to Boolean formulae in which each variable occurs at most k times. This problem leads to reductions to some key optimization problems.

Theorem 29.11 *There is a gap preserving reduction from MAX-3SAT to MAX-3SAT(29) that transforms a Boolean formula ϕ to ψ such that*

- *if* $\mathrm{OPT}(\phi) = m$, *then* $\mathrm{OPT}(\psi) = m'$, *and*
- *if* $\mathrm{OPT}(\phi) < (1 - \varepsilon_M)m$, *then* $\mathrm{OPT}(\psi) < (1 - \varepsilon_b)m'$,

where m and m' are the number of clauses in ϕ and ψ, ε_M is the constant determined in Theorem 29.7, and $\varepsilon_b = \varepsilon_M/43$.

Proof: The proof critically uses expander graphs. Recall, from Section 20.3, that graph $G = (V, E)$ is an *expander* if every vertex has the same degree, and for any nonempty subset $S \subset V$,

$$|E(S, \overline{S})| > \min(|S|, |\overline{S}|),$$

where $E(S, \overline{S})$ denotes the set of edges in the cut $(S, \overline{S})$, i.e., edges that have one endpoint in S and the other in $\overline{S}$. Let us assume that such graphs are efficiently constructible in the following sense. There is an algorithm $\mathcal{A}$ and a constant N_0 such that for each $N \geq N_0$, $\mathcal{A}$ constructs a degree 14 expander graph on N vertices in time polynomial in N (Remark 29.12 clarifies this point).

Expanders enable us to construct the following device whose purpose is to ensure that in any optimal truth assignment, a given set of Boolean variables must have consistent assignment, i.e., all true or all false. Let $k \geq N_0$, and let G_x be a degree 14 expander graph on k vertices. Label the vertices with distinct Boolean variables $x_1, \ldots, x_k$. We will construct a CNF formula ψ_x on these Boolean variables. Corresponding to each edge (x_i, x_j) of G_x, we will include the clauses $(\overline{x}_i \vee x_j)$ and $(\overline{x}_j \vee x_i)$ in ψ_x. A truth assignment to $x_1, \ldots, x_k$ is said to be *consistent* if either all the variables are set to true or all are set to false. An inconsistent truth assignment partitions the vertices of G_x into two sets, say S and $\overline{S}$. Assume w.l.o.g. that S is the smaller set. Now, corresponding to each edge in the cut $(S, \overline{S})$, ψ_x will have an unsatisfied clause. Therefore, the number of unsatisfied clauses, $|E(S, \overline{S})|$, is at least $|S| + 1$. We will use this fact critically.

Next, we describe the reduction. We may assume w.l.o.g. that every variable occurs in ϕ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula in any essential way.

Let B denote the set of Boolean variables occurring in ϕ. For each variable $x \in B$, we will do the following. Suppose x occurs $k \geq N_0$ times in ϕ. Let $V_x = \{x_1, \ldots, x_k\}$ be a set of completely new Boolean variables. Let G_x be a degree 14 expander graph on k vertices. Label its vertices with variables from V_x and obtain formula ψ_x as described above. Finally, replace each occurrence of x in ϕ by a distinct variable from V_x. After this process is carried out for each variable $x \in B$, every occurrence of a variable in ϕ is replaced by a distinct variable from the set of new variables

$$V = \bigcup_{x \in B} V_x.$$

Let ϕ' be the resulting formula. In addition, corresponding to each variable $x \in B$, a formula ψ_x has been constructed.

Finally, let

$$\psi = \phi' \wedge (\bigwedge_{x \in B} \psi_x).$$

Observe that for each $x \in B$, each variable of V_x occurs exactly 29 times in ψ – once in ϕ', and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3SAT(29). We will say that the clauses of ϕ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.

Now, the important claim is that an optimal truth assignment for ψ must satisfy all Type II clauses, and therefore must be consistent for each set $V_x, x \in B$. Suppose that this is not the case. Let τ be an optimal truth assignment that is not consistent for V_x, for some $x \in B$. τ partitions the vertices of G_x into two sets, say S and $\overline{S}$, with S being the smaller set. Now, flip the truth assignment to variables in S, keeping the rest of the assignment the same as τ. As a result, some Type I clauses that were satisfied under τ may now be unsatisfied. Each of these must contain a variable of S, and so their number is at most $|S|$. On the other hand we get at least $|S| + 1$ new satisfied clauses corresponding to the edges in the cut $(S, \overline{S})$. Thus, the flipped assignment satisfies more clauses than τ, contradicting the optimality of τ.

Let m and m' be the number of clauses in ϕ and ψ. The total number of occurrences of all variables in ϕ is at most $3m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42m$ Type II clauses. In addition, ψ has m Type I clauses. Therefore, $m' \leq 43m$.

If ϕ is satisfiable, then so is ψ. Next, consider the case that $\text{OPT}(\phi) < (1 - \varepsilon_M)m$, i.e., $> \varepsilon_M m$ clauses of ϕ remain unsatisfied under any truth assignment. If so, by the above claim, $> \varepsilon_M m \geq \varepsilon_M m'/43$ of the clauses of ψ must remain unsatisfied. The theorem follows. □

Remark 29.12 The assumption about the efficient construction of expander graphs is slightly untrue. It is known that for each $N \geq N_0$, an expander of size $\leq N(1+o(1))$ can be constructed efficiently (see Section 29.9). The reader can verify that this does not change the status of Theorem 29.11.

Exercise 29.4 extends Theorem 29.11 to establishing hardness for MAX-3SAT(5).

29.5 Hardness of vertex cover and Steiner tree

In this section, we will apply the machinery developed above to some graph theoretic problems. For integer $d \geq 1$, let VC(d) denote the restriction of the cardinality vertex cover problem to instances in which each vertex has degree at most d.

Theorem 29.13 *There is a gap preserving reduction from MAX-3SAT(29) to VC(30) that transforms a Boolean formula ϕ to a graph $G = (V, E)$ such that*

- *if* $\mathrm{OPT}(\phi) = m$, *then* $\mathrm{OPT}(G) \leq \frac{2}{3}|V|$, *and*
- *if* $\mathrm{OPT}(\phi) < (1 - \varepsilon_b)m$, *then* $\mathrm{OPT}(G) > (1 + \varepsilon_v)\frac{2}{3}|V|$,

where m is the number of clauses in ϕ, ε_b is the constant determined in Theorem 29.11, and $\varepsilon_v = \varepsilon_b/2$.

Proof: Assume w.l.o.g. that each clause of ϕ has exactly 3 literals (this can be easily accomplished by repeating the literals within a clause, if necessary). We will use the standard transformation. Corresponding to each clause of ϕ, G has 3 vertices. Each of these vertices is labeled with one literal of the clause. Thus, $|V| = 3m$. G has two types of edges (see the illustration below):

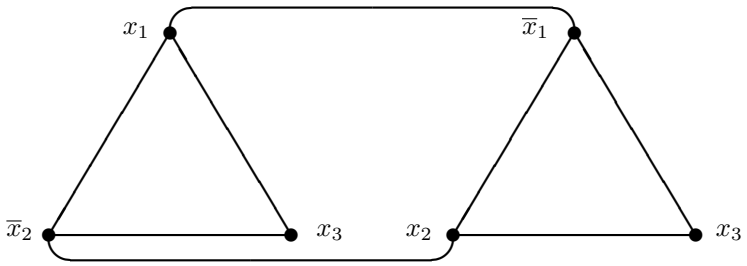
- for each clause, G has 3 edges connecting its 3 vertices, and
- for each $u, v \in V$, if the literals labeling u and v are negations of each other, then (u, v) is an edge in G.

Each vertex of G has two edges of the first type and at most 28 edges of the second type. Hence, G has degree at most 30.

We claim that the size of a maximum independent set in G is precisely $\mathrm{OPT}(\phi)$. Consider an optimal truth assignment and pick one vertex, corresponding to a satisfied literal, from each satisfied clause. Clearly, the picked vertices form an independent set. Conversely, consider an independent set I in G, and set the literals corresponding to its vertices to be true. Any extension of this truth setting to all variables must satisfy at least $|I|$ clauses.

The complement of a maximum independent set in G is a minimum vertex cover. Therefore, if $\mathrm{OPT}(\phi) = m$ then $\mathrm{OPT}(G) = 2m$. If $\mathrm{OPT}(\phi) < (1-\varepsilon_b)m$, then $\mathrm{OPT}(G) > (2 + \varepsilon_b)m$. The theorem follows. □

As an illustration, consider the formula $(x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$. The graph produced by the reduction given in Theorem 29.13 is given below:



Theorem 29.14 *There is a gap preserving reduction from VC(30) to the Steiner tree problem. It transforms an instance $G = (V, E)$ of VC(30) to an instance $H = (R, S, \text{cost})$ of Steiner tree, where R and S are the required and Steiner vertices of H, and* cost *is a metric on $R \cup S$. It satisfies:*

- *if* $\text{OPT}(G) \leq \frac{2}{3}|V|$, *then* $\text{OPT}(H) \leq |R| + \frac{2}{3}|S| - 1$, *and*
- *if* $\text{OPT}(G) > (1 + \varepsilon_v)\frac{2}{3}|V|$, *then* $\text{OPT}(H) > (1 + \varepsilon_s)(|R| + \frac{2}{3}|S| - 1)$,

where $\varepsilon_s = 4\varepsilon_v/97$, and ε_v is the constant determined in Theorem 29.13.

Proof: Graph $H = (R, S, \text{cost})$ will be such that G has a vertex cover of size c iff H has a Steiner tree of cost $|R| + c - 1$. H will have a required vertex r_e corresponding to each edge $e \in E$ and a Steiner vertex s_v corresponding to each vertex $v \in V$. The edge costs are as follows. An edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.

Let us show that G has a vertex cover of size c iff H has a Steiner tree of cost $|R| + c - 1$. For the forward direction, let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.

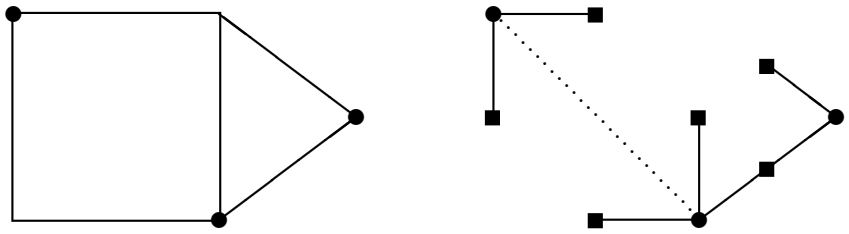
For the reverse direction, let T be a Steiner tree in H of cost $|R| + c - 1$. We will show below that T can be transformed into a Steiner tree of the same cost that uses edges of cost 1 only. If so, the latter tree must contain exactly c Steiner vertices. Moreover, every required vertex of H must have a unit cost edge to one of these Steiner vertices. Therefore, the corresponding c vertices of G form a cover.

Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. (If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.) Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G. Now, removing (u, v) from T gives two connected components. Let the

set of required vertices in these two sets be R_1 and R_2. Clearly, u and v lie in different sets, so path p must have two adjacent edges, say (a, b) and (b, c) such that their corresponding vertices, say w and w', lie in R_1 and R_2, respectively. Let the Steiner vertex, in H, corresponding to b be s_b. Now, throwing in the edges (s_b, w) and (s_b, w') must connect the two components. Observe that these two edges are of unit cost.

Now, if $\mathrm{OPT}(G) \leq \frac{2}{3}|V|$, then $\mathrm{OPT}(H) > |R| + \frac{2}{3}|S| - 1$, and if $\mathrm{OPT}(G) > (1+\varepsilon_v)\frac{2}{3}|V|$, then $\mathrm{OPT}(H) > |R| + (1+\varepsilon_v)\frac{2}{3}|S| - 1$. The theorem follows. □

The reduction is illustrated below. Graph G is an instance of the vertex cover problem. The highlighted vertices form a cover. Graph H shows the Steiner tree corresponding to this cover in the reduced graph. Required vertices have been marked with squares, and the three Steiner vertices corresponding to the cover have been marked with circles (the remaining Steiner vertices have been omitted for clarity). The edge between two Steiner vertices in the tree is dotted to distinguish it from the remaining edges, which connect required and Steiner vertices.



G H

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.

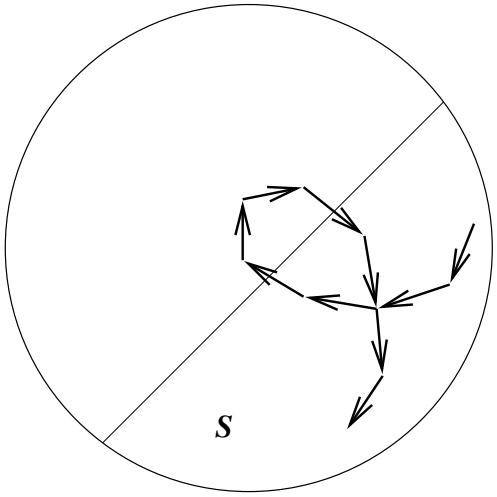
The following clever idea overcomes this difficulty. We will use a constant degree expander graph to generate $O(\log n)$ strings of $b \log n$ bits each, using only $O(\log n)$ truly random bits. The verifier will be simulated using these $O(\log n)$ strings as the "random" strings. Clearly, these are 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.*

29.7 Hardness of set cover

As stated in Chapter 2, the simple greedy algorithm for the set cover problem, which is perhaps the first algorithmic idea one would attempt, has remained essentially the best algorithm. Since set cover is perhaps the single most important problem in the theory of approximation algorithms, a lot of effort was expended on obtaining an improved algorithm.

In this section, we will present the remarkable result that the approximation factor of this algorithm is tight up to a constant multiplicative factor. Improved hardness results show that it is tight up to lower order terms as well (see Section 29.9). This should put to rest nagging doubts about the true approximability of this central problem.

29.7.1 The two-prover one-round characterization of NP

Observe that for the purpose of showing hardness of MAX-3SAT and clique (Theorems 29.7 and 29.19), we did not require a detailed description of the kinds of queries made by the verifier – we only required a bound on the number of queries made. In contrast, this time we do need a description, and moreover, we want to first establish that a particularly simple verifier